Proyecto Final de Carrera

Simulación de entornos con robots manipuladores móviles (humanoides) en entorno mediante la Herramienta Webots.

Laura Arnal Benedicto

Director: Enrique Jorge Bernabeu Soler

Titulación: Ingeniería Informática

Septiembre 2011

A mi familia, amigos y compañeros, por su apoyo y ayuda.

Al director de este proyecto, por su paciencia y colaboración.

ÍNDICE

1.- RESUMEN

El objetivo del presente proyecto era realizar una plataforma que permitiera simular el comportamiento de un robot humanoide en entornos con objetos que deberá esquivar, utilizando el software comercial Webots *(http://www.cyberbotics.com)* en su versión 5.10. Para ello, se ha implementado un controlador para el modelo simulado del robot Hoap2 de Fujitsu a partir del controlador disponible como ejemplo en Webots, así como tres tipos de zancadas y un algoritmo de tipo A* para la búsqueda del camino.

Este proyecto puede utilizarse para la prueba de distintos algoritmos para el cálculo de caminos y para sortear objetos, así como para probar diferentes robots humanoides. De igual manera, también puede ampliarse añadiendo visión u otros objetos que el robot deberá manipular.

2.- PALABRAS CLAVE

Robótica: Según Asimov, la robótica se refiere a la ciencia o arte relacionada con la inteligencia artificial (para razonar) y con la ingeniería mecánica (para realizar acciones físicas sugeridas por el razonamiento).

Humanoide: un robot humanoide es aquel que está hecho para asemejarse a un ser humano en apariencia y comportamiento.

Webots: software para simular robots móviles ampliamente usado con fines educativos.

Simulación: proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el funcionamiento del sistema o evaluar nuevas estrategias.

3.- INTRODUCCIÓN

El objetivo del presente proyecto era realizar una plataforma que permitiera simular el comportamiento de un robot humanoide en entornos con objetos que deberá esquivar, utilizando el software comercial Webots.

Como robot humanoide se ha elegido el robot Hoap2 de Fujitsu, del que ya existía un modelo disponible en el software utilizado para la simulación. Este robot mide unos 50 centímetros de altura, tiene un peso de unos 7kg y posee un total de 25 articulaciones: 6 para cada pierna, 5 para cada brazo, dos para la cabeza y uno para la inclinación del cuerpo. Actualmente existe el robot Hoap3, que fue puesto a la venta en 2005.

El entorno en el que se desarrolla la simulación ha sido creado mediante el editor de Webots, y el controlador se ha desarrollado mediante Microsoft Visual Studio 2005.

3.1.- CONTENIDO DE LA MEMORIA

A lo largo de este documento se describirá la realización de este proyecto. Para ello, se ha dividido la memoria en nueve apartados. Los cuatro primeros tratan de hacer una introducción al proyecto y su contexto. A continuación, se habla de las dos herramientas más importantes a la hora de la realización de este proyecto: Webots y Visual Studio 2005. Del primero se intentará dar una visión global de su funcionamiento, mientras que del segundo se indicará como se ha de configurar el proyecto para compilarlo satisfactoriamente y que el controlador resultante pueda ejecutarse en Webots.

A continuación se pasa a describir el controlador implementado. Primero se habla de cómo se ha representado el entorno en el que el robot va a moverse y del algoritmo utilizado para el cálculo del camino. A continuación se describe cómo se han realizado los movimientos y la orientación del robot, así como la forma en la que se van controlando los errores que va cometiendo el robot. Por último, se describe la función que se encarga de inicializar todos los parámetros necesarios para la ejecución y el bucle principal de control de robot.

Después de explicar la implementación, se pone un ejemplo de ejecución del controlador. Por último, se describe cómo modificar el controlador para adaptarlo a nuevos entornos u otros algoritmos de resolución de caminos. También se comentan posibles ampliaciones a este trabajo y unas pequeñas conclusiones finales.

3.2.- MOTIVACIÓN

En la actualidad existen muchas áreas de investigación en el campo de la robótica humanoide. Todavía no existen en el mercado robots fiables y robustos de tamaño humano, y el bipedismo todavía no está resuelto completamente.

Este proyecto podría servir como plataforma para la prueba de movimientos de robots de tipo humanoide y de su interacción con el entorno, esto es, esquivando objetos, planificando caminos, etc, para facilitar la investigación y el desarrollo de los mismos.

4.- CONTEXTO

Este proyecto se sitúa dentro de la ingeniería automática, más concretamente en la robótica humanoide. Un robot humanoide es un robot hecho para asemejarse a un ser humano tanto en apariencia como en comportamiento. Por esta razón suelen tener dos brazos, dos piernas, tronco y cabeza, aunque existen robots humanoides con ruedas. En los robots humanoides con ruedas evitamos el problema de la locomoción bípeda, pero para entornos reales en los que se pueden encontrar obstáculos como escaleras se vuelven necesarios los robots bípedos.

Al intentar simular el comportamiento y la estructura de los humanos, los robots humanoides son de gran complejidad. Poseen un alto número de grados de libertad (más de veinte) y esto hace que su dinámica y cinemática sea muy compleja. También presentan otros problemas de tipo energético: la mayoría de los robots humanoides existentes son incapaces de saltar, ya que la relación entre la energía y el peso no es tan buena como en un cuerpo humano

Los robots humanoides tienen múltiples aplicaciones. Se pueden utilizar en campos como la medicina (para la asistencia a otras personas), el ámbito doméstico, el rescate en situaciones de peligro, entretenimiento, información en congresos o museos, y muchos otros.

Actualmente existen competiciones de robots humanoides para fomentar la investigación, como por ejemplo la RoboCup, que tiene como objetivo enfrentar un equipo de robots humanoides a un equipo humano en el año 2050.

Los robots humanoides también aparecen recurrentemente en la ficción, principalmente dentro del género de la ciencia-ficción, tanto en obras literarias como en obras cinematográficas.

5.- HERRAMIENTAS UTILIZADAS

Las principales herramientas utilizadas para la realización de este proyecto han sido Webots 5.10 y Visual Studio 2005. Además se han utilizado, pero en menor medida, otras herramientas como Excel, para la edición de los ficheros .csv, ya que facilitaba su edición.

El equipo utilizado es un PC con un procesador AMD a 2,4GHz y 512MB de memoria física. El sistema operativo es Windows XP SP3.

5.1.- WEBOTS

Webots es un entorno de desarrollo para modelar, programar y simular robots móviles. Con este software, el usuario puede diseñar configuraciones complejas de robots, con uno o varios robots similares o diferentes, en un entorno compartido. El usuario puede elegir las propiedades de cada objeto, como forma, color, textura, masa, fricción, etc. También hay disponible una gran variedad de sensores y actuadores con la que poder equipar nuestros robots.

Los controladores de los robots se pueden programar en un entorno de desarrollo externo, o en el que hay disponible en Webots. El comportamiento de los robots se puede probar en mundos físicamente realistas. Además, estos controladores se pueden trasladar a un robot físico real.

Webots ha sido desarrollado por el Swiss Federal Institute of Technology de Lausanne durante 10 años, y se utiliza en aproximadamente 750 universidades y centros de investigación de todo el mundo. Tiene versiones tanto para Windows, como para Linux y Mac; aunque en este caso se ha utilizado la versión para Windows.

La primera vez que se inicia Webots, hay que establecer cuál será nuestro directorio de trabajo. El asistente creará en este directorio una serie de carpetas, las más importantes de las cuales son “worlds” y “controllers”, y en ellas se almacenarán los mundos y los controladores creados por el usuario.

5.1.1.- Mundos y controladores en Webots.

Un “mundo” en Webots es un entorno virtual 3D en el cual se pueden crear objetos y robots. Éstos se guardan en la carpeta “worlds”, en un fichero .wbt que contiene una descripción para cada objeto: su posición, orientación, geometría, apariencia, propiedades físicas, tipo de objeto, etc. Un mundo es una estructura jerárquica donde los objetos pueden contener otros objetos (como en VRML97). Por ejemplo, un robot puede contener dos ruedas, un sensor de distancia y un servo que contiene una cámara, lo cual permite que la cámara se

mueva en relación al robot gracias al servomotor. Sin embargo, un fichero .wbt no contiene toda la información necesaria para la simulación. En un archivo de mundo se almacena el controlador de un robot como una referencia a un archivo ejecutable, pero no contiene el ejecutable en sí mismo.

Un controlador es un archivo ejecutable que se utiliza para controlar un robot descrito en un archivo de mundo. Los controladores se almacenan en la carpeta "controllers" de nuestro directorio de trabajo. Los controladores pueden ser archivos ejecutables (.exe en Windows) o archivos .class para el caso de java. Mediante el asistente podemos crear un archivo con la estructura básica de un controlador.

5.1.2.- La ventana principal.

Al iniciar Webots, se abren tres ventanas: la ventana principal, el "scene tree" y la ventana de log. La ventana principal permite ver y simular los entornos creados en un archivo .wbt.

Se puede navegar por el entorno mediante el ratón. Haciendo click con el botón izquierdo y moviendo el ratón se rota la escena: si se ha hecho click en un objeto sólido, la rotación será respecto de ese objeto, de lo contrario la rotación será relativa al origen de coordenadas. De la misma manera, haciendo click y arrastrando con el botón derecho podemos trasladar la cámara. Por último, mediante la rueda del ratón se puede controlar el zoom para alejar o acercar la escena.

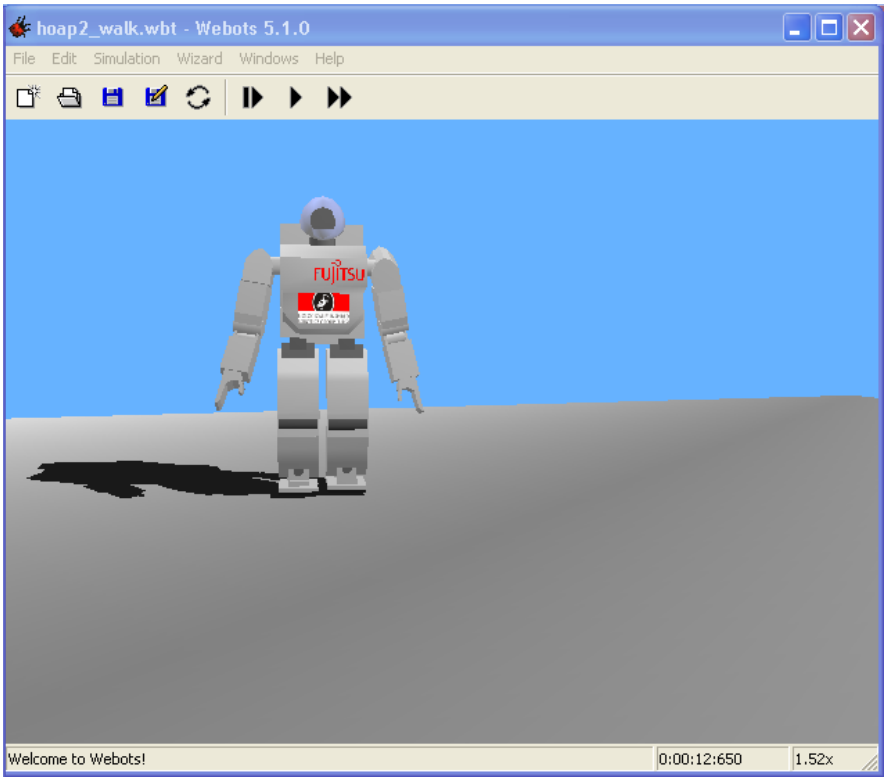

Figura 1: Ventana principal de Webots.

En esta ventana se pueden encontrar, a parte de los habituales menús y botones de Nuevo, Abrir y Guardar, botones que permiten parar, ejecutar o ejecutar paso a paso la simulación. También se pueden crear videos e imágenes de las simulaciones (menú File) y configurar distintos parámetros.

5.1.3.- La ventana de log

La ventana de log es una pequeña ventana en la que podemos imprimir mensajes y que puede ser muy útil para la depuración de nuestros controladores. Para imprimir mensajes en ella, se utiliza la función "robot_console_printf" de la API de Webots. No obstante, también podemos imprimir mensajes por salida estándar o a un fichero con las funciones típicas del lenguaje de programación utilizado.

Los mensajes que aparecen en la ventana de log también se pueden consultar en el archivo "webots.log" que se crea en nuestro directorio de trabajo.

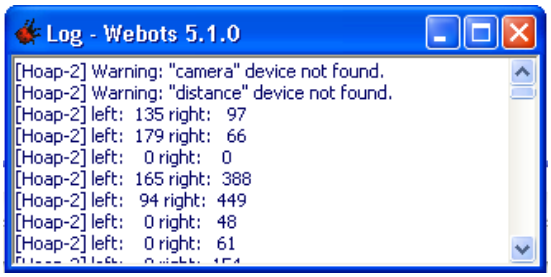

Figura 2: Ventana de log.

5.1.4.- Scene Tree

El scene tree o árbol de escena contiene toda la información necesaria para describir la representación gráfica y la simulación del mundo 3D. El árbol de escena de Webots está estructurado como un fichero VRML97, que se compone de una lista de nodos cada uno de los cuales contiene campos. Estos campos pueden contener valores de diferentes tipos u otros nodos.

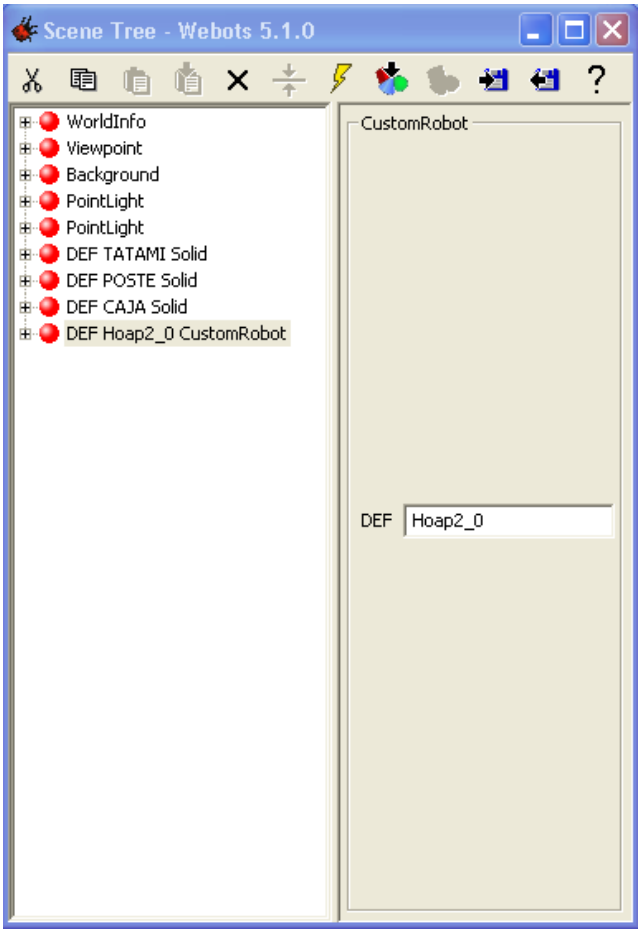

Figura 3: Scene Tree.

5.1.5.- Los nodos VRML97

VRML (Virtual Reality Modeling Language) es formato de archivo normalizado que tiene como objetivo la representación de escenas u objetos interactivos tridimensionales; diseñado particularmente para su empleo en la web. El lenguaje VRML posibilita la descripción de una escena compuesta por objetos 3D a partir de prototipos basados en formas geométricas básicas o de estructuras en las que se especifican los vértices y las aristas de cada polígono tridimensional y el color de su superficie.

Algunos nodos en Webots pertenecen a VRML97 mientras que otros son específicos de Webots. Por ejemplo, el nodo Solid hereda del nodo Transform de VRML97 y se puede seleccionar y mover con los botones en la ventana del principal. En el manual de Webots se puede encontrar una descripción completa de estos nodos.

5.2.- EL ENTORNO DE PROGRAMACIÓN VISUAL STUDIO 2005

Microsoft Visual Studio 2005 es un entorno de desarrollo integrado para Windows que soporta lenguajes como C y C++, entre otros. Para la realización de este proyecto se ha optado por usarlo como entorno de programación. Para poder ejecutar el controlador correctamente desde Webots, hay que configurar correctamente el proyecto. Para ello, los pasos a seguir son los siguientes:

1.- Crear una carpeta (por ejemplo, *my_controller)* en nuestro directorio local de Webots. Lanzar Visual Studio e ir al menú *File New…*

2.- Crear un proyecto de tipo "*Win32 Console Application"* o "*Win32 Application"* si no se necesita una consola para la depuración. Establecer el nombre del proyecto y la ubicación en nuestro directorio local de Webots *(\webots\controllers\my_controller)*. Elegir crear un proyecto vacío.

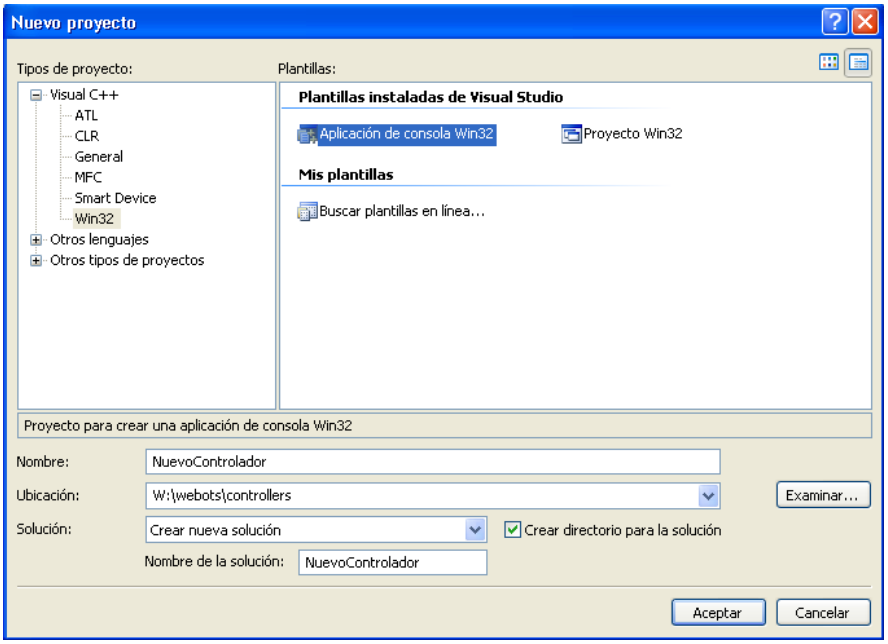

Figura 4: Creación del proyecto

3.- Crear un archivo de código C++ llamado *my_controller.c* mediante el menú *Files New…* en el directorio *my_controller.*

4.- Ir al menú *Build Configurations…* y eliminar la configuración *Win32 Debug Configuration.* Cerrar la ventana de configuraciones.

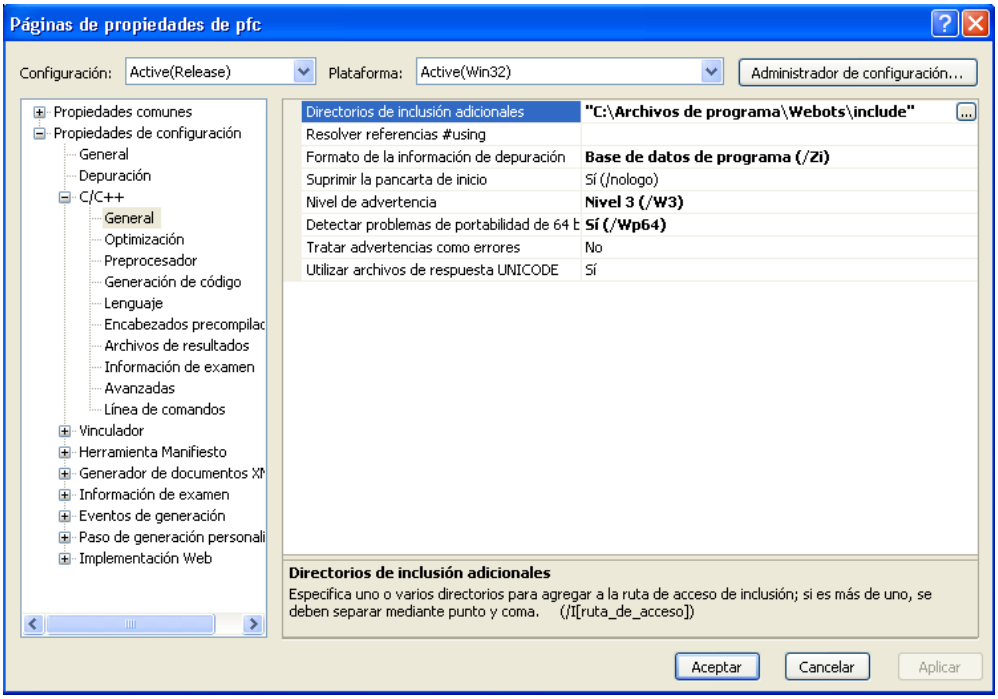

Figura 5: Configuración del proyecto. Pestaña C/C++.

5.- Ir al menú de configuración de proyecto y seleccionar la pestaña de C/C++. Seleccionar la categoría *Preprocessor* y añadir en la entrada *Additional include directories* "C:\Program Files\Webots\include". Posteriormente, en la pestaña *Link,* categoría *General,* cambiar *Output file name* de "Relase/my_controller.exe" a "my_controller.exe". Después, hay que añadir al principio de la lista *Object/library modules* "Controller.lib". Por último, en la categoría *Input* se debe escribir "C:\Program Files\Webots\lib" como *Additional library path.*

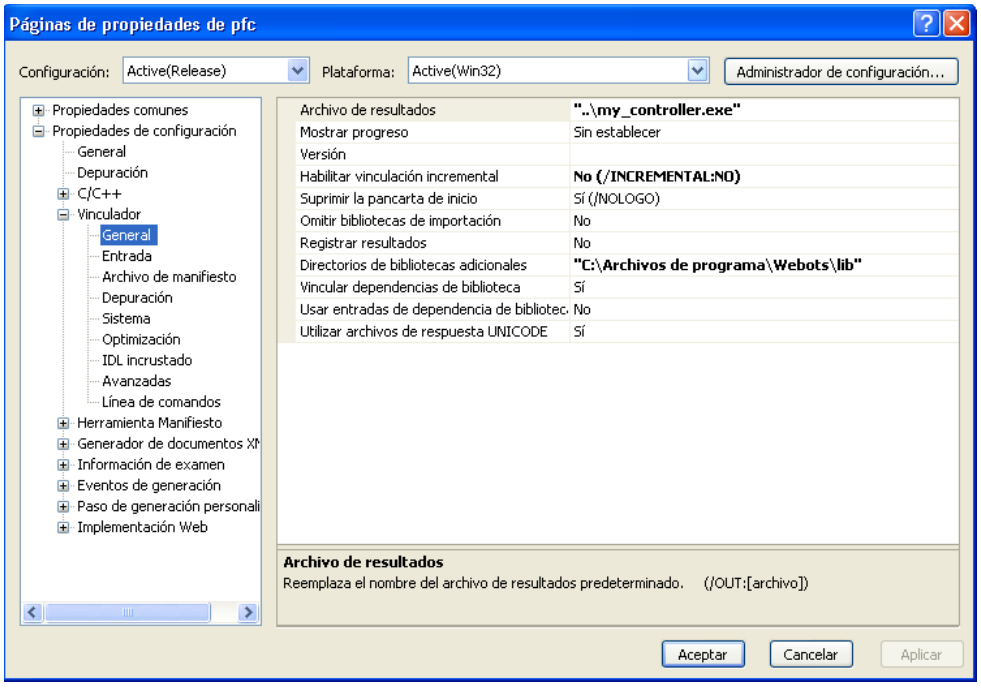

Figura 6: Configuración del proyecto. Pestaña Linker (vinculador).

6.- Ahora ya sólo queda escribir el código fuente del controlador en el archivo “my_controller.c”. En el directorio “C:\Program Files\Webots\controllers\simple” hay un controlador llamado “simple.c” que puede servir como base.

Se puede compilar la aplicación mediante el menú *Build -> Build my_controller.exe* o mediante la tecla F7. Esta opción dará error si se ha lanzado la simulación o si el robot tiene asignado el controlador en el nodo “controller”, ya que en ese instante el controlador está en uso; pero esto se puede solucionar cambiando temporalmente el controlador a “void” y luego volviendo a cargar el controlador creado. El ejecutable no se podrá lanzar desde Visual Studio, sólo desde Webots.

6.- IMPLEMENTACIÓN DEL CONTROLADOR

En el controlador implementado se pueden distinguir dos grandes partes diferenciadas: el cálculo del camino a recorrer y el control de los movimientos del robot. No obstante, antes de pasar a explicar estas partes hay que comentar cómo se ha representado la información del mapa y los objetos.

6.1.- REPRESENTACIÓN DE LOS OBJETOS

Como ya se ha comentado, en el entorno de Webots los objetos se representan mediante nodos VRML y nodos propios de Webots derivados de éstos. En este caso, los objetos se han representado mediante nodos de tipo Solid. Este tipo de nodos son propios de Webots pero heredan muchas de sus características de los nodos Transform de VRML. Como diferencia, los sensores de los robots y el detector de colisiones pueden detectar los objetos de tipo Solid.

Sin embargo, para tenerlos representados con mayor comodidad en el controlador, se ha implementado una sencilla clase llamada CObstaculo en la que se almacena la posición (como coordenadas x, z) y el tamaño (lado o diámetro) del objeto.

6.2.- REPRESENTACIÓN DE LAS POSICIONES

Para representar las posiciones (x,z) del mapa real se ha creado la clase CNodo. Cada nodo tiene una coordenada x, una coordenada y (no utilizado en este problema, ya que no hay elevaciones en el terreno), una coordenada z, el coste real de llegar hasta ese nodo, el valor heurístico de llegar hasta el nodo final y un puntero a un nodo predecesor. Estos tres últimos valores se emplean a la hora de calcular el camino.

De entre los métodos implementados cabe destacar el método compararNodo, que compara si dos nodos pertenecen a la misma posición. Se considera que dos nodos representan la misma posición si la diferencia entre sus coordenadas es menor que 0.00001.

6.3.- REPRESENTACIÓN DEL MAPA

Para representar el mapa y sus elementos, se ha implementado la clase CMapa. En esta clase hay una matriz de booleanos para representar las posiciones del mapa. Sin embargo, no todas las posiciones están representadas en el mapa, sino que se crea una red de posiciones tomándolas según una determinada resolución que le indiquemos en la variable RES_MAPA (definida en el fichero de constantes), formando una especie de rejilla. En este caso vale 0.2, es decir, se toman posiciones de 20 en 20 centímetros. Se han implementado métodos para conocer, dada una posición del mapa, a qué celda de la matriz corresponde ; así como la operación inversa. Cada posición tomará el valor "true" o "false" según si es accesible o no.

En esta clase también se almacena un vector de obstáculos, que contendrá todos los objetos del mapa. También se han implementado métodos de consulta para poder acceder a estos objetos.

A la hora de añadir un obstáculo, se actualiza la matriz ocupando las casillas correspondientes a las posiciones ocupadas por el objeto. Sin embargo, hay que tener en cuenta que algunas posiciones próximas al objeto tampoco serán accesibles debido al tamaño del robot; así que estas posiciones también se marcan como inaccesibles.

En la figura 7 se puede apreciar un ejemplo de mapa de 3.2x3.2 metros, con su rejilla de posiciones correspondiente, así como un ejemplo en el que hay un objeto de 40 centímetros de lado situado en (1.4, 1.6). En verde oscuro se indican las posiciones ocupadas por el objeto, mientras que en verde claro se marcan las posiciones a las que el robot no puede acceder debido a su propio tamaño (en el caso del hoap, 30 centímetros de ancho, la mitad será 15 centímetros y por lo tanto una casilla). Las casillas de los bordes también están marcadas como no visitables para evitar que el robot caiga por un lado del mapa, o chocara con las paredes si las hubiera.

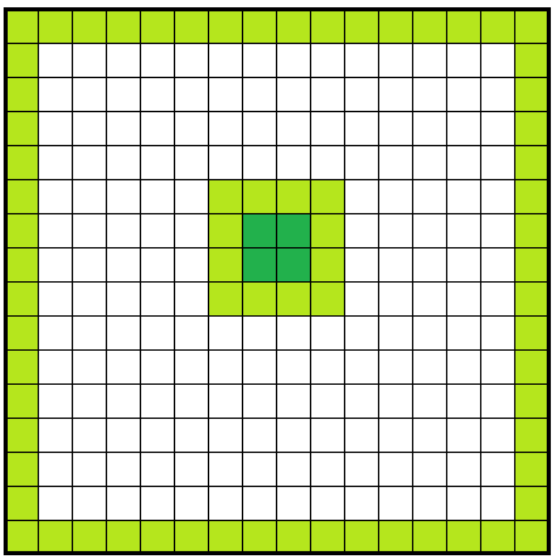

Figura 7: ejemplo de mapa.

En la figura 8 se puede ver el resultado una vez se pone en marcha el controlador. Cuando el robot evita el obstáculo lo hace con una cierta holgura que asegura que no se produzcan colisiones.

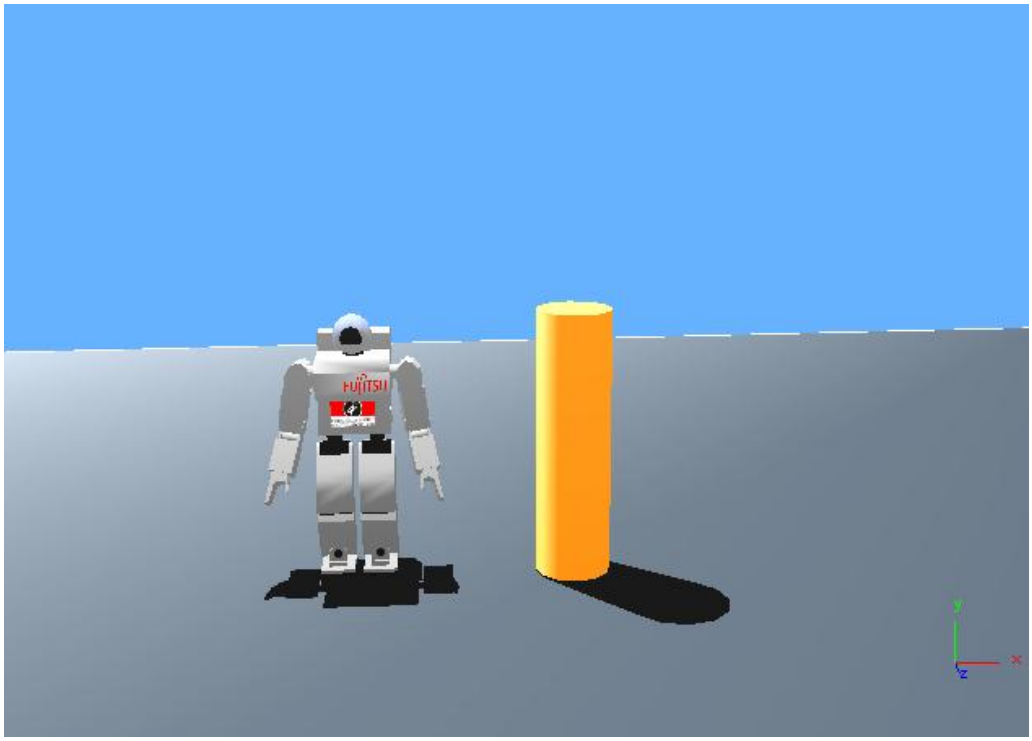

Figura 8: El robot Hoap pasando al lado del obstáculo.

El camino a recorrer (calculado como se explicará más adelante) también se almacena en esta clase. Para ello existe el vector m_Path, en el que se almacenarán los nodos correspondientes a las posiciones intermedias del camino. También hay otras variables relacionadas con este vector: m_iPathSize, que indica el tamaño del vector anterior; y m_PathIndex, que indica cual es la posición del siguiente nodo al que el robot deberá moverse.

6.4.- ALGORITMO DE BÚSQUEDA DE CAMINO

En esta sección se va a describir el algoritmo utilizado para calcular el camino: el GraphSearch. Este algoritmo es de tipo A*, y por lo tanto siempre encuentra una solución en el caso de que esta exista. Además, si la heurística empleada no sobreestima el coste real de alcanzar el nodo objetivo, el algoritmo garantiza optimalidad. Como inconveniente, este tipo de algoritmos emplean mucha memoria, al tener que almacenar todos los posibles nodos accesibles desde un nodo dado.

6.4.1.- Coste y heurística.

Para calcular el camino, hay que puntuar cada nodo con un determinado valor para determinar por dónde se alcanza el camino más corto. Esta puntuación se divide en dos partes: el coste y el valor heurístico.

El coste es la distancia real que se lleva recorrida hasta alcanzar ese nodo. En este caso, el coste de ir de un nodo a otro adyacente será el indicado por la constante RES_MAPA; excepto en el caso de que el trayecto entre nodos se realice en diagonal, en el que el coste será igual a la raíz cuadrada del doble del cuadrado de RES_MAPA. En el caso de este proyecto, en el que dicha constante vale 0.2, el coste de ir en diagonal de un nodo a otro adyacente es de 0.28284.

Mediante la heurística, se busca establecer una estimación de lo que falta para llegar desde ese nodo al nodo destino. Como se ha comentado, si la heurística no sobreestima el coste real de alcanzar el nodo objetivo, se garantiza la optimalidad. Debido a esto, se ha elegido como heurística la distancia euclídea:

$$h(n)=\sqrt{(x_{dest}-x)^2+(z_{dest}-z)^2}$$

Esta heurística nunca dará un valor superior al real, ya que la distancia más corta entre dos puntos es la línea recta, y por lo tanto lograremos el camino más corto entre los nodos inicial y final.

6.4.2.- Algoritmo GraphSearch.

Para el cálculo del camino a seguir se ha implementado el algoritmo GraphSearch, que por las razones ya comentadas proporcionará un camino de longitud mínima. La implementación de este algoritmo se encuentra dentro de la clase CMapa.

El funcionamiento del algoritmo es el siguiente: se crean dos listas ordenadas “open” y “close” En la “open” se insertan aquellos nodos candidatos para el camino y en la “close” los nodos ya analizados. El algoritmo consiste en un bucle que se ejecuta mientras la lista “open” tenga elementos o se haya llegado al nodo final. Dentro de este bucle se buscan los sucesores del nodo actual y se insertan en la lista “open” si es mejor que los que ya están tanto en la “open” como en la “close”. Antes de insertar se rellenan todos los parámetros del nodo (heurística, predecesor…). Cuando se han mirado ya todos los sucesores del nodo actual, éste se inserta en la “close” y la nueva iteración del bucle extraerá el nodo más prometedor de la lista “open”. En el Apéndice se adjunta el pseudocódigo para este algoritmo.

En cuanto a la generación de sucesores, ésta se realiza mediante la función de la clase CMapa “generar_sucesores”. En este método para el nodo que se le pasa como parámetro se calculan los ocho nodos adyacentes, y para cada uno de ellos se establece el coste como el

coste de llegar hasta el nodo actual más el coste de ir del nodo actual al sucesor. También se calcula la heurística para cada uno de estos nodos de la forma que ya se ha comentado. Todos los nodos sucesores generados se almacenan en el vector "sucesores" disponible dentro de la clase CMapa.

6.5.- CONTROL DE MOVIMIENTO

Los robots humanoides tienen un gran número de articulaciones, es decir, grados de libertad, y por esta razón su cinemática y su dinámica es muy compleja. Además, el problema del bipedismo estable no está completamente resuelto. En este caso, se han desarrollado los siguientes tipos de zancada: paso adelante y rotación en ambos sentidos.

6.5.1.- Movimiento del robot.

Los movimientos independientes que puede realizar el robot se desarrollaron mediante un controlador independiente del controlador final de este proyecto de final de carrera. Este controlador era básicamente como la función "mover" que se comentará a continuación.

Como ya se ha indicado, se han realizado tres tipos de movimientos diferentes. Para cada uno de esos movimientos que realiza el robot, se han almacenado en un fichero los valores en radiantes que deben tomar las articulaciones. Por esta razón, en la función "mover" del controlador simplemente se leen los valores que deben tomar cada uno de los servomotores de las articulaciones y se aplican estos cambios. Esta función recibe como parámetros el tipo de movimiento (si es de avance, de giro en sentido positivo o de giro en sentido negativo) y el valor del ángulo para el caso de los giros

El paso hacia adelante ya estaba resuelto en el ejemplo disponible en Webots, y sólo se ha modificado para adaptarlo a las necesidades del proyecto. En el fichero "unPaso.csv" está la secuencia de ángulos para las articulaciones correspondiente a, partiendo de una posición con las dos piernas apoyadas, un paso con la pierna izquierda, un paso con la pierna derecha y vuelta la posición con las dos piernas apoyadas.

Los pasos de rotación se han desarrollado con la ayuda del centro de gravedad del robot como comprobador de estabilidad: si el centro de gravedad cae dentro de la sombra del pie del robot, la posición es estable; en caso contrario, es inestable. Existen dos ficheros, "giroPos.csv" y "giroNeg.csv" que corresponden a realizar un giro en sentido positivo y negativo, respectivamente.

En el caso del giro positivo, el robot partiendo de una posición en la que tiene las dos piernas apoyadas, primero realiza un equilibrio sobre la pierna derecha, posteriormente se cambia el valor la primera articulación (correspondiente a la rotación) de la pierna izquierda con el valor del ángulo que queremos girar. Después, apoya las dos piernas, para hacer un equilibrio sobre la pierna izquierda y volver a una posición en la que las dos piernas tienen una rotación de cero radianes. El caso del giro negativo es equivalente, pero primero se hace el equilibrio sobre la pierna izquierda, y la articulación que se rota es la primera de la pierna derecha.

En el caso concreto de los giros, para que se realicen de forma correcta tanto en sentido positivo como en sentido negativo, si el ángulo de giro es mayor de 45º (0.7854 radianes) éste se divide en varios giros de como máximo 45º hasta realizar el giro completo.

En el desarrollo de los tres tipos de movimientos ha sido de utilidad los sensores de contacto que están situados en la planta de los pies del robot, ya que ayudaban a determinar si los equilibrios se estaban realizando correctamente o si por el contrario el robot aún estaba realizando dos apoyos. Estos sensores están modelados en Webots mediante los nodos "TouchSensor" y se puede obtener el valor de los mismos mediante la función "touch_sensor_get_value" de la API de Webots.

6.5.2.- Cálculo de la orientación.

Una vez conseguidos los movimientos independientes, ya se puede enviar el robot a alcanzar una determinada posición. Esto supone calcular la orientación previa a la que tendrá que dirigirse el robot.

A la hora de calcular los ángulos de giro del robot para orientarse hacia la posición destino, no sólo hay que tener en cuenta la orientación final a la que deberá dirigirse sino también la orientación que en ese instante lleva el robot.

La orientación que lleva el robot en cada momento podemos conocerla gracias al nodo GPS. Este nodo se utiliza, como su propio nombre indica, para modelar sensores de tipo GPS; y con ellos podemos conocer tanto la posición como la orientación del robot llamando a las funciones adecuadas de la API de Webots. Así pues, mediante la llamada a las funciones gps_get_matrix y gps_euler obtenemos la orientación del robot. Para calcular la orientación interesa conocer el ángulo de giro con respecto al eje Y. El valor de este ángulo toma valores positivos si el robot está rotado hacia posiciones negativas del eje X, y valores negativos en el caso contrario.

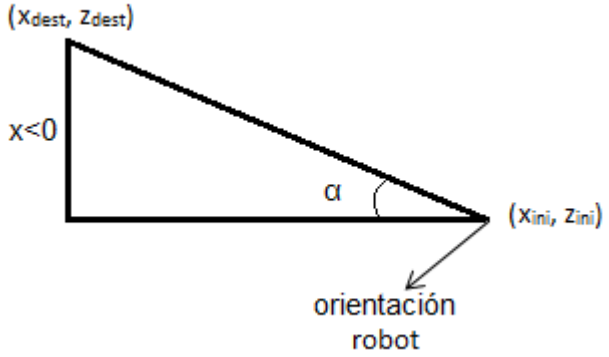

Figura 9: cálculo de la orientación. En este caso, habrá que realizar un giro negativo

La orientación hacia la posición destino se puede calcular por trigonometría. Como se puede ver en la figura 9, si se crea un vector con origen en la posición actual del robot y como destino la posición destino, el seno del ángulo buscado es igual del cociente entre la componente X y el módulo del vector. Por lo tanto:

$$\alpha = \arcsin(dx/\|v\|)$$

Este ángulo indica el ángulo de la orientación con respecto al eje Z, un ángulo positivo indicará que hay que rotar hacia el eje X positivo, mientras que un ángulo negativo indicará que está orientado hacia el eje X negativo. También hay que tener en cuenta el signo del ángulo. El signo resultante para el caso de la figura 9 será un valor negativo, por el contrario, en el caso de la figura 10 el ángulo será positivo.

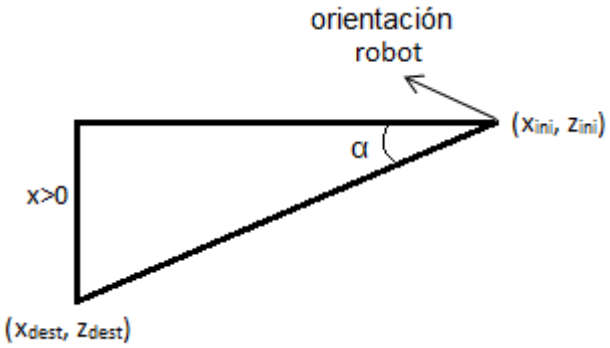

Figura 10: cálculo de la orientación. En este caso, habrá que realizar un giro positivo.

Por último, para calcular el ángulo de giro que deberá realizar el robot, basta con sumar ambos ángulos: el obtenido por el gps y el ángulo de la nueva orientación.

6.5.3.- Control del error.

Para evitar que los errores cometidos por el robot en la realización de las zancadas alejen al robot de la posición destino, se comprueba al final de la realización de cada paso el error cometido hasta el momento. Si éste es mayor que un umbral determinado (en este caso, de un centímetro, pero este valor se puede modificar desde el fichero de constantes) se calcula el ángulo de giro necesario para que el robot tenga la orientación adecuada para llegar a su destino de la forma indicada en el apartado anterior.

Para calcular el error se necesita saber tanto la posición actual del robot como la posición a la que debería haber ido. La distancia entre estas dos posiciones nos dará el error que ha cometido el robot hasta el momento.

Como ya se ha comentado anteriormente, el modelo del robot que se ha utilizado en el proyecto posee un nodo GPS, que nos permite conocer la posición y orientación del robot. En este caso, para conocer la posición, necesitamos llamar a las funciones gps_get_matrix, gps_position_x y gps_position_z de la API de Webots. Como ya se ha comentado, en este entorno no se ha considerado el hecho de que haya desniveles en el terreno y por esta razón no necesitamos conocer la componente Y de la posición.

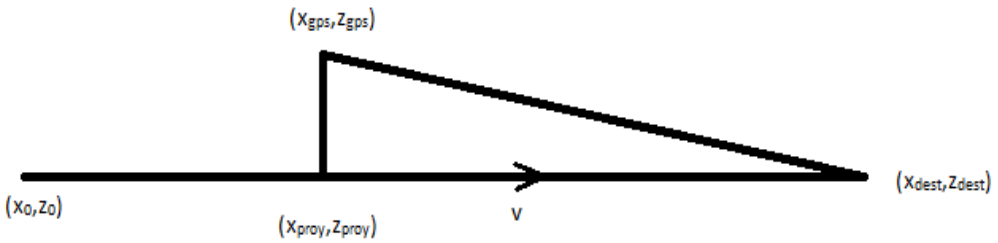

Figura 11: punto proyección

Para calcular la posición a la que debería haber llegado el robot, hay que calcular el punto proyección de la posición real del robot (obtenida mediante el gps) sobre la recta entre los puntos de inicio y destino. El vector director de dicha recta se calcula como:

$$v = (x_{dest} - x_{ini}, z_{dest} - z_{ini})$$

y el punto proyección sobre la recta cuyo vector director es v, será:

$$(x_{proy}, z_{proy}) = (x_{ini}, z_{ini}) + \lambda \cdot v$$

Donde:

$$\lambda = \frac{-v \cdot [(x_{0,} z_0) - (x_{gps}, z_{gps})]}{\|v\|^2}$$

Una vez obtenidas las coordenadas de los dos puntos, calculando la distancia entre ambos se puede conocer el error cometido. Si esta distancia es mayor que el error permitido (definido en el fichero de constantes), se calculará una nueva orientación que sitúe el robot en la dirección correcta para llegar al destino. Una vez calculada la orientación, se cambia el punto de inicio del robot al valor de su posición actual. Esto es necesario para el caso en el que se necesite calcular una nueva orientación debida a futuros errores.

6.6.- LA FUNCIÓN DE INICIALIZACIÓN

Antes de comenzar a realizar los movimientos del robot, es necesario invocar una función de inicialización. Esta función se llama “reset”, y se encarga de iniciar todos los parámetros necesarios para la ejecución del controlador. Por tanto, es en esta función donde se realizan funciones propias inicialización de Webots, como habilitar las articulaciones del robot y los sensores como por ejemplo el GPS. Funciones importantes aquí son “robot_get_device”, que nos devuelve un identificador único para el dispositivo cuyo nombre se pasa como parámetro. Además, también hay que habilitar la lectura de los sensores, e indicar cada cuánto tiempo se ha de actualizar este valor. Esta acción se lleva a cabo mediante funciones como “gps_enable” y “servo_enable_position”.

Otra parte importante de la inicialización es la creación del mapa interno. Para ello hay que crear una variable del tipo CMapa, que hay que iniciar con los valores adecuados del tamaño del mapa, y posición inicial y final del robot. Una vez creado, podemos añadir tantos obstáculos como se deseen.

Cuando se finaliza de configurar el mapa, ya sólo queda calcular el camino invocando al método GraphSearch.

6.7.- BUCLE PRINCIPAL DEL CONTROLADOR

El bucle principal del controlador es el que va recuperando las posiciones intermedias obtenidas mediante el algoritmo GraphSearch y envía al robot las órdenes de moverse o reorientarse en función de cada una de estas posiciones. También es el que controla que el robot no se exceda del máximo error permitido. La función que implementa este bucle es la función “run”.

Para la llegada a una posición objetivo (tanto posiciones intermedias como la posición final) se ha considerado como aceptable un error máximo de 15 centímetros. Este valor se puede modificar desde el fichero de constantes.

Así pues, el bucle principal del controlador va extrayendo las posiciones destino intermedias calculadas hasta que llega a la posición final. Para cada una de estas posiciones, primero se calcula la orientación que debe tomar el robot para llegar al destino de la forma que ya se ha comentado con anterioridad. Una vez calculada esta orientación, se le ordena al robot que realice el giro correspondiente.

En el momento en el que el robot ya está orientado, éste comienza a moverse hacia adelante dando pasos. Tras cada paso se comprueba si ya ha llegado al destino y, por lo tanto, se tiene que extraer una nueva posición intermedia; o si por el contrario aún no ha llegado a ese destino. Si no se ha llegado a la posición, se calcula el error cometido en la trayectoria y se corrige la orientación si es el caso, y el robot se mueve de nuevo.

7.- ENTORNO DE EJEMPLO

A continuación se va a describir un ejemplo de un entorno de Webots en el cual el robot se mueve de una posición inicial a una final esquivando un poste. De esta manera se pretende acabar de explicar todos los detalles implementados con la ayuda del ejemplo.

El entorno tiene las siguientes características:

- El mapa tiene como coordenada mínima en el eje X 0 y máxima 4; y como coordenada mínima en el eje Z 0 y máxima 4.

- La posición inicial del robot es (2,1).

- La posición final a la que el robot debe llegar es (2,3). Se ha añadido un objeto con forma de caja en (2, 3.5) para tener una referencia del punto final al que debe llegar.

- El poste tiene un diámetro de 15 centímetros, y se encuentra en la posición (2,2).

Con estos datos, el entorno queda como se muestra en la Figura 12.

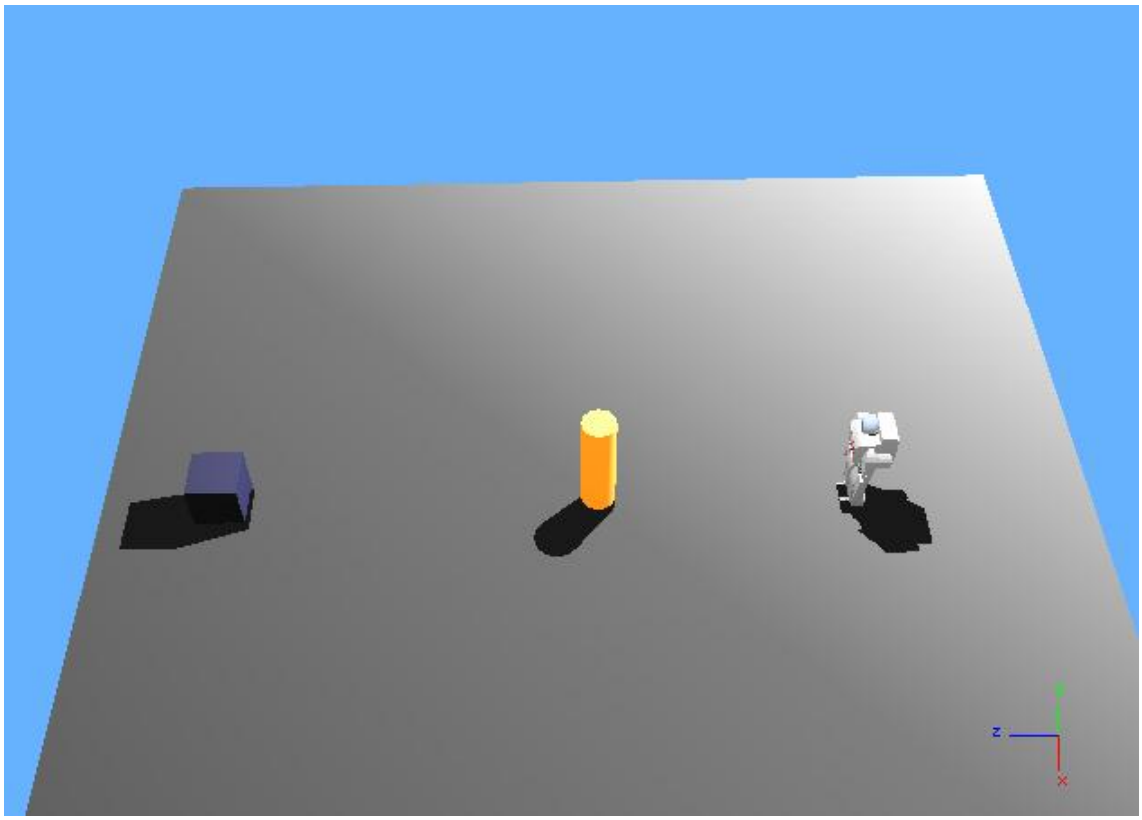

Figura 12: Posición inicial.

Con estos parámetros, se han marcado una serie de posiciones como inaccesibles, y se calcula el camino que debe realizar el robot para llegar al destino indicado.

En ese instante, el robot comienza a moverse. Como se ve en la figura 13, primero se orientará hacia la primera posición intermedia, que en este caso es (1.8, 2.2).

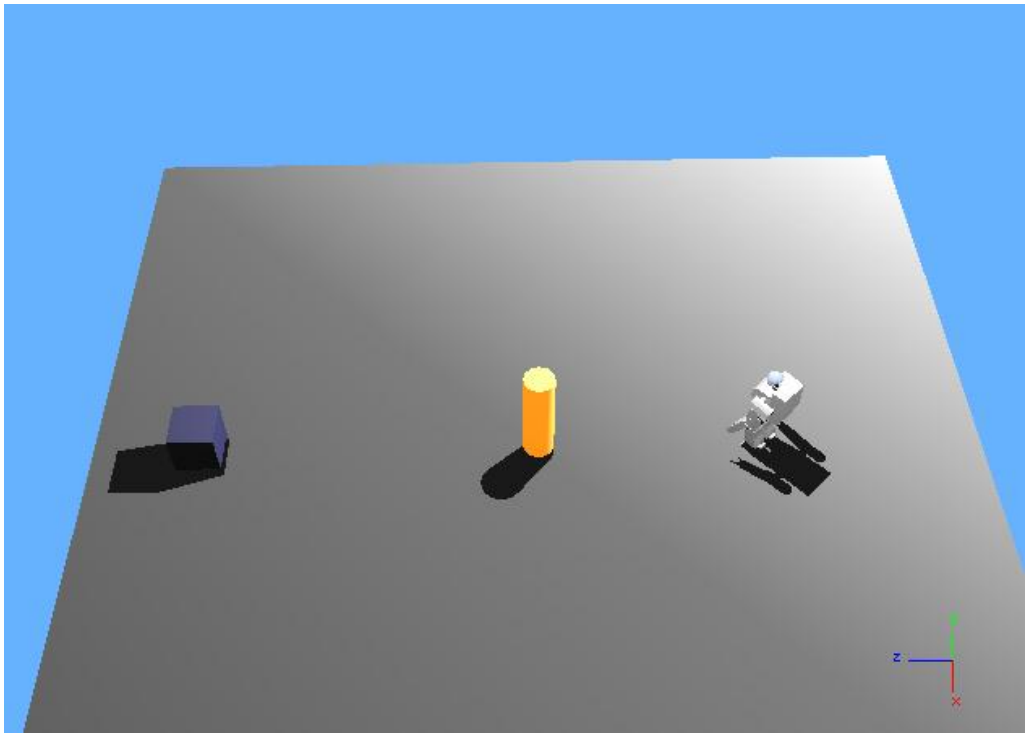

Figura 13: Orientación hacia la primera posición intermedia.

Cuando llegue a la posición (1.8, 2.2) continuará orientándose y moviéndose hacia la siguiente posición. Si en algún momento detecta que se ha desviado un centímetro, corregirá su orientación antes de volver a dar un paso.

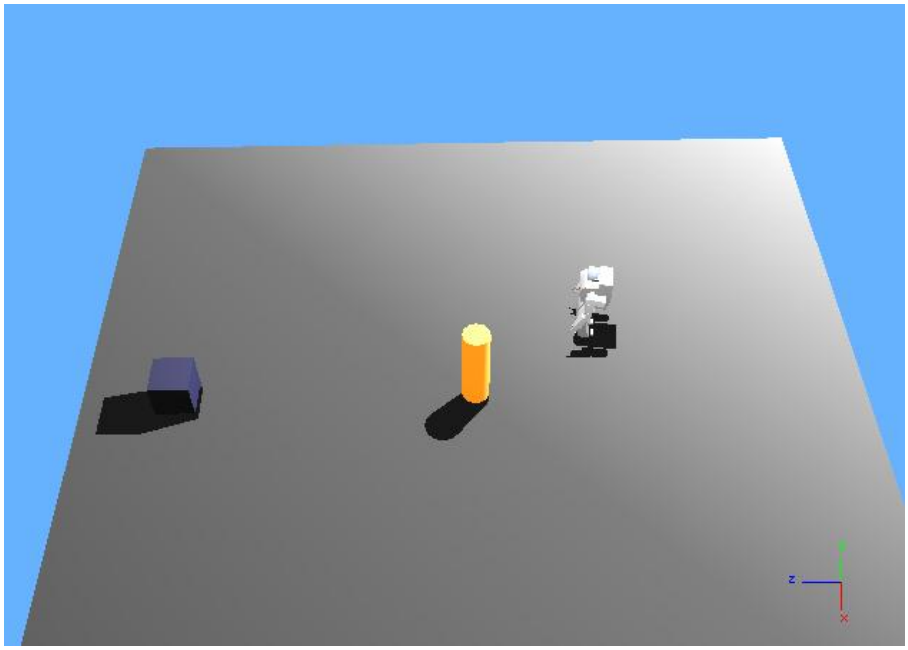

Figura 14: Orientación y posición adecuadas para superar el objeto.

Cuando el robot llega a la altura del obstáculo, se puede ver que pasa con cierta holgura debido a las posiciones del mapa marcadas como prohibidas.

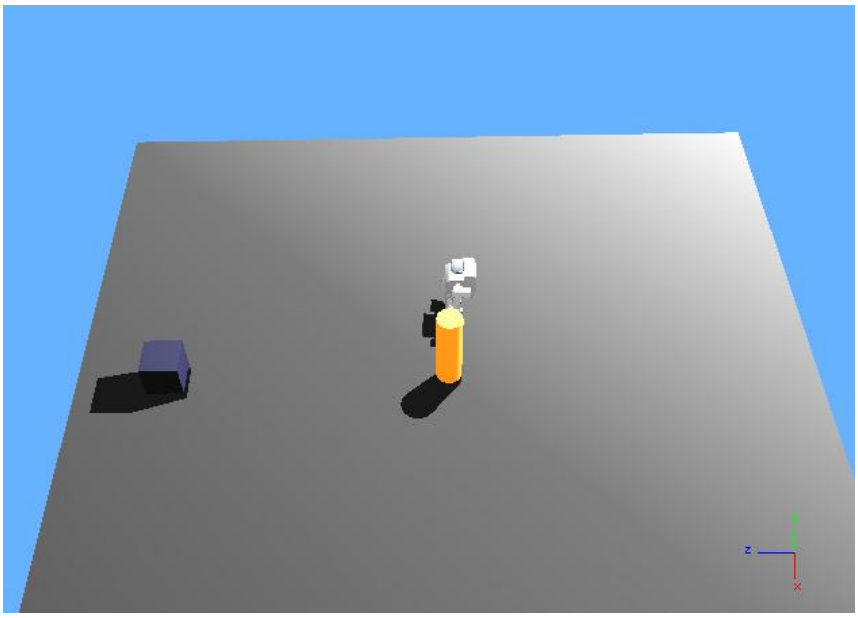

Figura 15: el robot a la altura del objeto.

Una vez ya ha evitado el obstáculo, ya sólo falta llegar hasta la posición final. El robot sigue recorriendo posiciones intermedias hasta que llega al destino deseado.

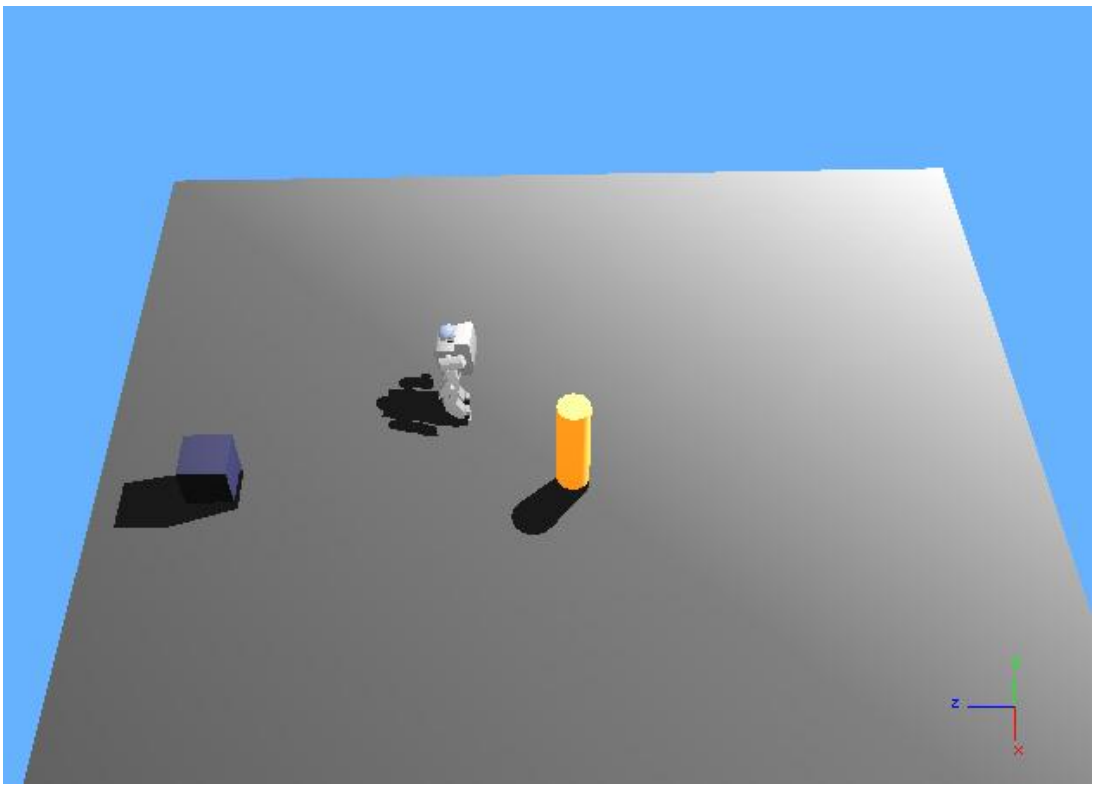

Figura 16: situación tras esquivar el objeto.

Cuando el robot llega a la posición final, se detiene y el controlador acaba su ejecución. El resultado final sería como se ve en la figura:

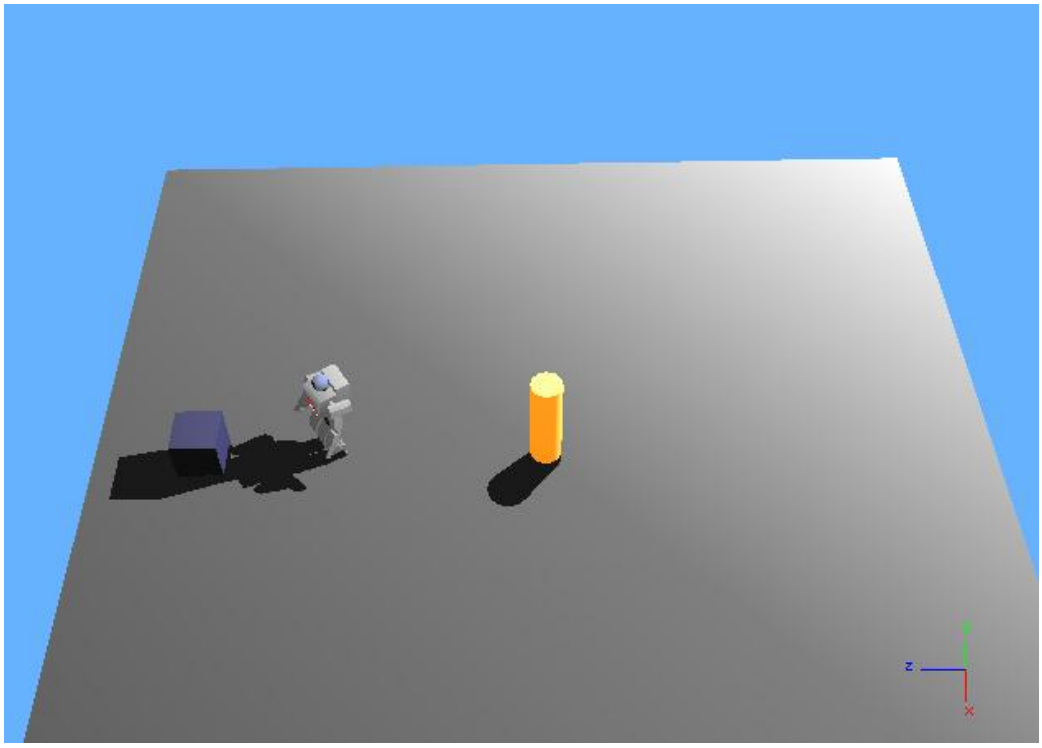

Figura 17: posición final del robot.

8.- MANUAL DE USUARIO

8.1.- AÑADIR OBSTÁCULOS

Para añadir un nuevo obstáculo a nivel del controlador se debe añadir en la función "reset" una llamada al método "anadir_obstaculo" de la clase CMapa, indicando la posición y el tamaño del lado o del diámetro de la base de la figura. Por ejemplo, si queremos añadir un cilindro en la posición (1.5, 2.8) con radio 0.3, la llamada quedará:

Mapa.anadir_obstaculo(1.5, 2.8, 0.6);

Después de recompilar el controlador, ya se puede ejecutar de nuevo la simulación.

A nivel de entorno de Webots, se debe añadir un nuevo nodo de tipo "Solid". Para ello, desde el Scene Tree hay que hacer click en el botón "Insert after" y luego seleccionar un nodo Solid. Haciendo click en el símbolo + delante del nombre podemos cambiar sus características y podemos por ejemplo, darle un nombre al nodo.

Figura 18: Detalle de los botones del Scene Tree: cut, copy, paste, paste after, delete, default value, transform, insert after, insert node, export node, import node y ayuda.

A continuación se selecciona el campo "children" y luego se hace click en el botón "insert after", seleccionándose un nodo de tipo Shape. Haciendo click en este nodo Shape, se inserta un nodo de tipo Appearance en el campo de apariencia mediante el botón "insert node". Ahora hay que repetir la misma operación con el campo "material". Una vez creado el nodo de tipo Material, se pueden modificar las características del objeto para establecer el color del mismo. Si se desea que el objeto tenga una textura en concreto, se utilizará el nodo Texture.

Ahora, falta seleccionar la geometría del objeto. Para ello, desde el nodo Geometry del nodo Shape creado, insertamos un nodo de tipo Cylinder. Si se desea un prisma en vez de un cilindro, el nodo será de tipo Box.

Dentro del nodo Cylinder, se pueden modificar las características del cilindro. Es aquí donde se debe indicar el radio de 0.3 que se estaba usando como ejemplo.

Para asegurar que el robot no choque, se debe añadir un nodo en el campo "boundingObject", de lo contrario el detector de colisiones de Webots no detectaría el poste y

podríamos pensar que el robot lo evita cuando en realidad lo está atravesando. Se puede añadir un nodo en este campo de tipo Cylinder como ya se ha comentado, con las mismas características que el creado anteriormente. Sin embargo, si se cambia el tamaño del poste, habrá que modificar ambos nodos: el boundingObject y el cilindro en Geometry. Para evitar esto, si se selecciona el nodo en "geometry Cylinder" aparece abajo la palabra DEF y un cuadro de texto. Aquí se puede establecer un nombre, por ejemplo NUEVO_POSTE. Ahora si se selecciona el campo boundingObject del nodo Solid y se inserta un nodo, se puede utilizar la opción "use NUEVO_POSTE", con lo que el boundingObject tendrá las mismas características que la geometría definida; y los cambios en ésta se actualizarán automáticamente en el boundingObject.

Ahora sólo faltaría llevar el nodo a la posición deseada. Para ello hay que modificar el campo Transform del primer nodo que hemos añadido, el nodo Solid. El resultado se puede ver en la figura 19.

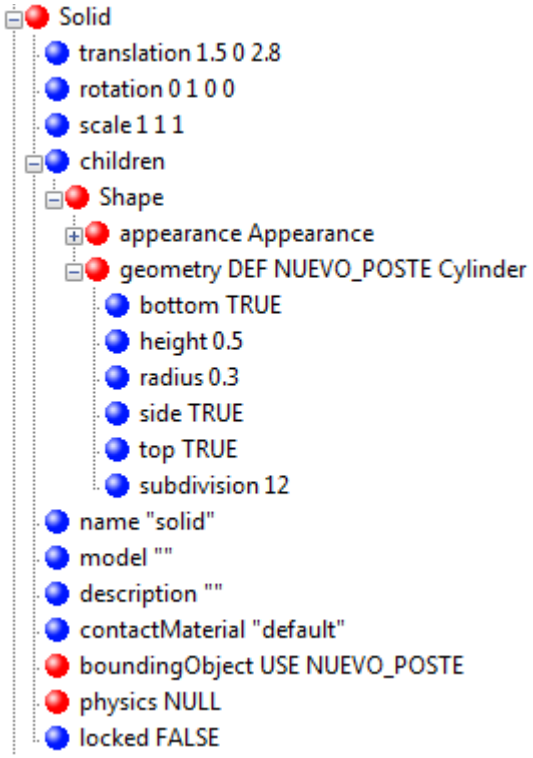

Figura 19: El poste que se ha añadido al Scene Tree.

Si se quiere eliminar un objeto que creado, se debe seleccionar el nodo y hacer click en el botón de eliminar. En el caso de campos como el de boundingObject o material, para eliminar el nodo que se ha insertado, se usará el botón "default value".

También se pueden utilizar las típicas operaciones de cortar y pegar mediante los botones correspondientes.

8.2.- MODIFICACIONES EN EL CONTROLADOR

En el controlador se pueden cambiar diferentes parámetros, tanto los márgenes de error admitidos como los referentes a los objetos y el tamaño del mapa. Evidentemente cada vez que se modifique el código fuente del controlador se deberá recompilar el proyecto. Esto ya se ha comentado cómo se ha de configurar el proyecto y como compilar en el apartado 5.2.

Para la creación del mapa, primero se debe iniciar en la función reset la variable de tipo mapa indicando las coordenadas mínimas y máximas en x,z. También se debe indicar dónde está la posición inicial y final a la que se va a dirigir el robot.

Una vez creado el mapa, se pueden añadir todos los obstáculos que se deseen mediante la llamada al método "añadir_objeto".

Si se desea probar este controlador con otro robot humanoide, se debe modificar la función "mover" y desarrollar pasos de avance y giro nuevos, ya que es posible que el peso y características del nuevo modelo no sean iguales a las del robot Hoap2. Además, en el fichero de constantes se deben modificar las constates referidas al tamaño del robot.

Otra de las modificaciones que se podrían realizar sería cambiar el algoritmo de cálculo del camino. Para ello se puede desarrollar un nuevo método en la clase CMapa que calcule el camino con el método que se desee, y que almacene el resultado en el vector m_Path. Posteriormente, en la función reset se invocará este método en lugar del método GraphSearch.

También es posible que se desee probar con diferentes heurísticas, en ese caso habría que modificar la función "calcula_heuristica" de la clase CMapa.

Por último, desde el fichero de constantes se pueden cambiar los márgenes de error que se consideran admisibles a la hora de corregir la posición o llegar al destino.

9.- CONCLUSIONES

El objetivo del proyecto era la simulación de robots humanoides en un entorno con objetos con la herramienta Webots. Así pues, la primera fase consistió en familiarizarse con el entorno de Webots y la forma de programar controladores, ya que no había utilizado nunca este software.

Una vez conocido el entorno, la siguiente fase fue crear un entorno en el que trabajar y elegir un robot que se adaptara a las características buscadas. Tras esto, continué decidiendo una forma de representar todos los elementos para poder calcular el camino. Esto me hizo relacionar el proyecto con otras asignaturas de la carrera relacionadas con la inteligencia artificial y la algorítmica. Una vez resueltos estos temas, se pasó a desarrollar las zancadas y el control de errores y orientación.

En definitiva, este proyecto me ha ayudado a conocer mejor el campo de la robótica humanoide, a obtener experiencia en un software de simulación de robots móviles y, en menor medida, a otros aspectos vistos a lo largo de la carrera.

9.1.- POSIBLES AMPLIACIONES

Son muchas las ampliaciones que se podrían hacer de este trabajo. Por ejemplo, se podría incluir visión, para que la resolución del camino fuera on-line y no fuera necesario incluir todos los objetos al principio del controlador. El sorteo de obstáculos también se podría realizar mediante sensores de distancia.

En cuanto a los objetos, en este caso sólo se han considerado objetos cilíndricos y prismas a la hora de modelar el mapa. Se podría trabajar con objetos más complejos, o también se podrían añadir objetos como escaleras o rampas para probar otros tipos de movimientos de los robots. Para este último caso, habría que cambiar el mapa para que considerara también la coordenada en el eje Y.

Además, los objetos siempre están estáticos, otra posible ampliación sería que los objetos estuvieran en movimiento, como por ejemplo una pelota golpeada por un segundo robot. A su vez, también se podría ampliar para que el robot cogiera objetos para trasladarlos a otra posición, o que, por ejemplo, los apilara.

10.- TABLA DE ILUSTRACIONES

BIBLIOGRAFÍA

- Webots User Guide (release 5.10) copyright (c) 2005 Cyberbotics Ltd. All rights reserved. (http://www.cyberbotics.com), 22 diciembre de 2005.
- Webots Reference Manual (release 5.10) copyright (c) 2005 Cyberbotics Ltd. All rights reserved. (http://www.cyberbotics.com), 22 diciembre de 2005.
- Wikipedia, the free Encyclopedia. http://en.wikipedia.org/wiki/Main_Page
- Apuntes de la asignatura Robótica de la UPV.
- Apuntes de la asignatura Sistemas Inteligentes de la UPV.

APÉNDICE: Pseudocódigo del algoritmo GraphSearch

Crear un nodo conteniendo el nodo final: node_goal.

Crear un nodo conteniendo el nodo inicial: node_start.

Introducir el node_start en la lista OPEN.

Mientras la lista OPEN no está vacía{

Extraer de la lista OPEN el nodo con el menor valor. Llamar a este nodo node_current.

Si node_current es el mismo que node_goal

Salir del bucle while.

Generar sucesores para node_current.

Para cada nodo sucesor {

Establecer el coste del nodo sucesor como el coste de node_current más el coste de llegar de node_current al nodo sucesor.

Buscar el nodo sucesor en la lista OPEN.

Si el nodo está en la lista OPEN pero el existente es tan bueno o mejor como el sucesor, descartar el sucesor y continuar con el siguiente sucesor.

Si el nodo está en la lista CLOSE pero el existente es tan bueno o mejor como el sucesor, descartar el sucesor y continuar con el siguiente sucesor.

Eliminar las ocurrencias del nodo sucesor de las lista OPEN y CLOSED.

Establecer node_current como el predecesor del nodo sucesor.

Establecer h como la distancia estimada al node_goal usando una función heurística.

Añadir el sucesor a la lista OPEN.

}

Añadir node_current a la lista CLOSED.

}