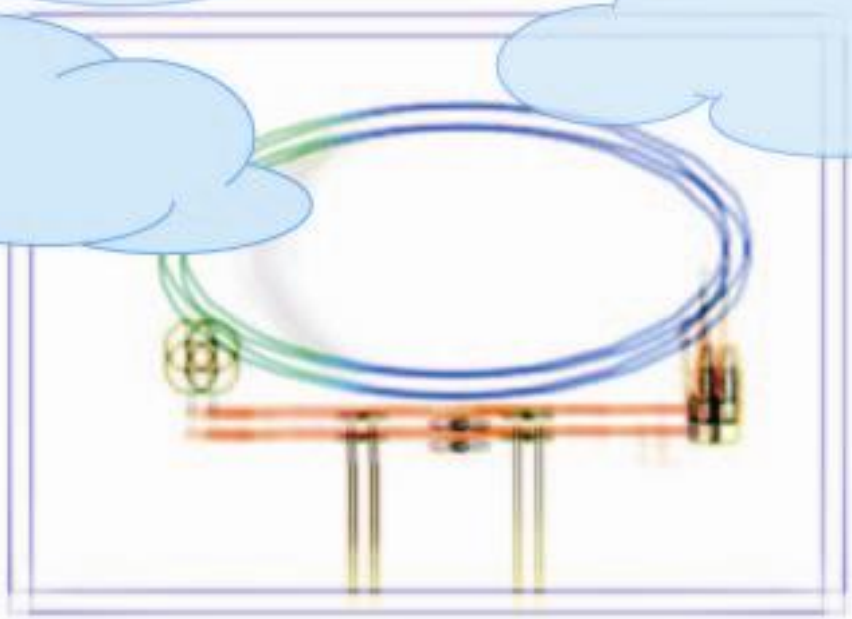

PFC - Simulación de un robot volador autónomo mediante la utilización de la herramienta Webots

UNIVERSIDAD POLITECNICA DE VALENCIA

Hector Díaz Bachiller

Dirigido por:
Enrique Jorge Bernabeu Soler

Curso 2009 - 2010

A todos los que desearon ver este proyecto finalizado.

Gracias.

Índice

1- Introducción

1.1- Descripción de los objetivos del proyecto

La finalidad de este proyecto final de carrera es conseguir simular el comportamiento de un robot volador utilizando el software comercial de Cyberbotics, Webots. Concretamente su versión 5.1.0.

Como robot volador se escogió una especie de globo dirigible, salvo que no lleva timón, de tamaño reducido. Como veremos más adelante, todos los movimientos del robot se controlan mediante hélices. Este zepelín es un robot construído en la École Polytechnique Fédérale de Lausanne (EPFL). Existe un modelo virtual bajo licencia GPL2 o superior para Webots que incluye componentes físicas y que es el punto de partida de este proyecto.

Teniendo como base dicho modelo virtual, que tiene en cuenta factores necesarios como el rozamiento o la inercia, vamos a construir un entorno por el que el globo dirigible deberá volar de forma autónoma. El vuelo se realizará siguiendo un circuito fijado de antemano, y terminará en un espacio destinado para el reposo del robot. Conseguir que el robot se desplace por los puntos deseados permitirá un mayor control y por consiguiente, las aplicaciones futuras serán muy interesantes (vigilancia forestal, fotografía aérea, transporte aéreo automático, etc.).

A lo largo de todo el recorrido que el robot debe realizar en el escenario creado para la simulación, también se añadirán obstáculos. Una parte muy importante de este proyecto va a ser que el globo deberá evitar la colisión con cualquier tipo de obstáculo fijo. Para así conseguir realmente un robot volador autónomo, que no precise de ningún tipo de interacción externa para realizar su cometido. La posición de los obstáculos será conocida en todo momento por el robot y éste deberá ser capaz de evitar todos aquellos que se encuentren en su trayectoria.

Por lo tanto necesitamos crear un mundo virtual en Webots que contenga obstáculos. Además se introducirá el modelo del globo dirigible en este mundo. Y con todo esto se trabajará en el controlador del robot que actuará sobre los motores para manejarlo de forma automática.

1.2- Herramientas utilizadas

Como ya se ha comentado, el programa principal con el que se va a trabajar es la versión 5.1.0 de Webots. Pero a parte de este software, también se ha precisado el uso de diferentes programas. La mayoría de ellos habituales en el ámbito científico (Matlab, Mathematica, Autodesk Map 5…) y otros más utilizados en cualquier otro ámbito (Bloc de notas, Excel, etc.).

El equipo utilizado siempre ha sido un AMD Athlon 64 Processor 3400+ a 2,4GHz y con 512 MB de memoria RAM. Y el sistema operativo sobre el que hemos ejecutado la principal herramienta de trabajo, Webots, ha sido Microsoft Windows XP Profesional SP3.

1.2.1- Webots

Webots es un entorno de desarrollo usado para modelar, programar y simular robots móviles. Con Webots el usuario puede diseñar complejas configuraciones de robots, con uno o más robots, iguales o diferentes, en un entorno compartido. Las propiedades de cada objeto, como forma, color, textura, masa, fricción, etc., las elije el usuario. Para equipar cada robot, se dispone de un amplio abanico de sensores y actuadores simulados.

Además, los controladores de los robots pueden ser programados con el propio entorno de desarrollo integrado (IDE) o con entornos de desarrollo de terceros. El comportamiento del robot puede ser testeado en mundos con física realística. Y los programas de los controladores se pueden transferir a robots reales existentes en el mercado.

Webots es usado en cerca de 700 universidades y centros de investigación alrededor de todo el mundo. Ha sido desarrollado por el Swiss Federal Institute of Technology en Lausana. Testeado, documentado y mantenido durante más de 10 años. El tiempo de desarrollo que logra ahorrar puede ser muy elevado.

Este software tiene versiones para Mac OS X, Linux y Windows. Y como ya hemos comentado para la realización de este PFC hemos utilizado la versión 5.1.0 de Webots que corre bajo el sistema operativo Windows XP. En esta versión la interfaz gráfica se divide en 4 pantallas que podemos modificar, ver u ocultar a nuestro gusto. Pero estos aspectos los trataremos más adelante.

1.2.2- Otras

Aunque la parte más importante del desarrollo de este proyecto se ha realizado con Webots, como en cualquier otra empresa científica ha sido necesaria la utilización de otras herramientas de trabajo menos especializadas pero no por ello prescindibles.

Quizá las más útiles hayan sido tanto Matlab como Mathematica, ambas herramientas han permitido realizar de forma manual los cálculos complejos que lleva a cabo el programa de forma automática. Así pues, cuando ha sido necesario corroborar los resultados obtenidos por el programa, se han utilizado estas herramientas para comprobar que los resultados eran correctos.

Como en muchas ocasiones los resultados obtenidos eran farragosos y difíciles de visualizar a simple vista, ha sido realmente útil en este aspecto disponer de una herramienta de dibujo en 3D. Autodesk Map 5 ha sido el programa instalado en el ordenador que nos ha permitido realizar este tipo de dibujos de forma sencilla. De esta manera, datos complejos como posiciones de esferas en el espacio tridimensional, trayectorias, etc., resultaban mucho más sencillos de tratar cuando los mostrábamos dibujados con esta herramienta.

A parte de estos programas, también se han utilizado otros mucho más extendidos en su uso como pueden ser Microsoft Office Excel o el Bloc de notas del propio sistema operativo. El primero de ellos ha sido especialmente útil para dibujar gráficas con los datos obtenidos del programa y así poder ajustar parámetros de una forma rápida y eficaz. El bloc de notas ha facilitado la programación con su función de reemplazar texto.

1.3- Contenido de la memoria

En esta memoria se explica paso a paso el desarrollo del proyecto final de carrera. Y para ello se ha dividido el contenido en 5 apartados principales, intentando así que la información quede lo más ordenada posible y sea fácil de seguir.

El primer apartado es una introducción que habla del objetivo del proyecto y las herramientas que han sido necesarias para realizarlo. El punto actual se incluye dentro de dicho apartado dedicado a la introducción.

En un segundo apartado hablaremos de la principal herramienta utilizada, el software de Cyberbotics Webots 5.1.0. Hablaremos de las características de esta versión, sobre todo de cómo se divide su interfaz gráfica y lo más importante, cómo la hemos organizado en nuestro proyecto.

El punto más importante será el que contempla la tercera parte de esta memoria. Vamos a describir el entorno de simulación. Qué es lo que necesitamos, cómo es el mundo que hemos creado para nuestras simulaciones, cómo es el robot que vamos a programar y cómo se ha diseñado todo esto. Además de la parte visual, hablaremos del controlador que maneja a nuestro robot volador. Describiremos cómo ha de ser la entrada de datos, cómo se ha logrado el movimiento del robot, la detección de obstáculos y la forma en que se evitan los mismos.

Después de haber explicado todo el trabajo realizado, el siguiente apartado está enfocado al uso posterior de este proyecto. Pues va a intentar dar toda la información posible para que un futuro usuario o programador pueda aprender a utilizar este proyecto. Se expondrán los requisitos para utilizar el proyecto, la forma en que se pueden modificar aspectos del entorno como la entrada de datos, y la forma de modificar el controlador del globo dirigible. Además, una breve explicación de cómo manejar el zepelín de forma manual.

Por último, se indicarán una serie de conclusiones acompañadas de posibles ámbitos de aplicación de este proyecto y también posibles mejoras o ampliaciones que se podrían implementar en un futuro.

2- Webots, herramienta de diseño y simulación

Este proyecto utiliza Webots *(http://www.cyberbotics.com)*, un software comercial de simulación de robots móviles desarrollado por Cyberbotics Ltd. Aunque a fecha de hoy la versión 6.0 de Webots ha cambiado su interfaz gráfica, la licencia que se tenía para trabajar con Webots era de su versión 5.1.0. Por lo tanto todo lo relacionado con Webots a partir de este punto será haciendo referencia a esta versión.

Cuando ejecutamos el programa por primera vez, debemos establecer cuál queremos que sea el directorio de trabajo. Debemos elegir una carpeta donde tengamos privilegios y él nos creará una serie de carpetas que contienen archivos necesarios para el funcionamiento de nuestro proyecto. Estas carpetas son las siguientes: controllers, data, objects, plugins, worlds, transfer, resources, lib, includes y doc. De todas ellas, vamos a prestar especial atención a controllers y worlds. Por la sencilla razón de que cuando creemos un mundo nuevo, se guardará en la carpeta worlds. Y cuando creemos un controlador nuevo para nuestro robot, se guardará en la carpeta controllers.

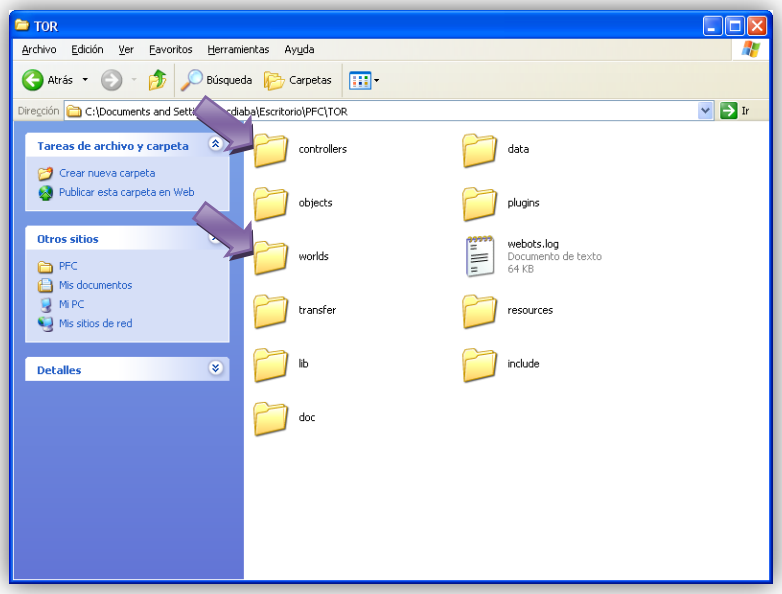

Figura 1: Directorio de trabajo Webots 5.1.0

Como ya hemos comentado antes en esta memoria, la interfaz de Webots se divide en 4 ventanas que podemos disponer donde nos plazca, además de modificar su tamaño de forma independiente. A continuación vamos a tratar de explicar la función de cada una de ellas y cómo la hemos utilizado para este proyecto en concreto.

2.1- VRML

Virtual Reality Modeling Language (VRML) es el lenguaje de programación que se utiliza en Webots para modelar el mundo virtual donde se simulará el comportamiento de los robots. Aunque cabe decir que no todos los nodos y características de este lenguaje están disponibles.

Este lenguaje posibilita la descripción de objetos 3D a partir de prototipos basados en formas geométricas básicas o estructuras especificadas a partir de vértices y aristas. Además se pueden añadir características a los objetos como color o materiales.

Como he comentado, Webots no incluye todos los nodos de VRML97, pero sí que añade algunos muy importantes para trabajar con simulación de robots. Por ejemplo tenemos el nodo *CustomRobot*, que es el que hemos utilizado nosotros para crear nuestro robot volador. Este nodo nos va a permitir modelar un robot a base de utilizar otros nodos y además añadirle aspectos especiales. Por ejemplo, al tratarse de un robot, vamos a poder asociarle un controlador. Pero este tema lo veremos más adelante en un apartado específico.

Nodos que aporta Webots y que podemos utilizar son el nodo *DifferentialWheels* para crear robots terrestres con ruedas. O el nodo *Servo*, que nos va a permitir modelar robots con servos y simular el comportamiento de los mismos. Con el nodo *Camera* podemos añadir una cámara virtual y modificar sus características de modo que obtendríamos una imagen virtual de lo que sería la imagen real si montáramos una cámara en nuestro robot. Podemos añadir sensores de distancia con el nodo *DistanceSensor*, e incluso un receptor GPS añadiendo el nodo *GPS* al robot. Si quisiéramos establecer algún tipo de comunicación, podemos incluir en la simulación los nodos *Emitter* y *Receiver*. Además de otros nodos como pueden ser *Charger*, *Gripper*, *Joint*, *LED*, *Light Sensor*, *Physics* o *TouchSensor*, tenemos el nodo *Supervisor*. Este nodo se puede utilizar para controlar el mundo y todos los robots que éste contiene.

Nuestro mundo sólo va a contener un robot del tipo *CustomRobot*. Y aunque podríamos añadir las físicas que proporciona Webots, las vamos a programar en el controlador del propio robot para poderlas personalizar un poco más. No obstante sí que vamos a utilizar una característica de los nodos *Solid* muy interesante que es el atributo *boundingObject*. Se trata de envolver a los objetos sólidos que forman el mundo con una forma básica que nos va a permitir calcular colisiones entre objetos. De esta manera, añadiremos un nivel más de realismo a nuestra simulación.

2.2- Scene Tree

Para poder crear el mundo virtual sobre el que trabajaremos, tenemos una ventana donde se nos muestra el "Scene Tree" o árbol de escena. Podemos mostrarla u ocultarla haciendo click en *Windows>Scene Tree* en la ventana principal (o Ctrl + T). Se trata de una ventana con la jerarquización de nodos VRML que forman todo el mundo. En esta ventana modificaremos el mundo añadiendo los nodos que sea necesario y modificando cualquiera de sus atributos desplegándolos con el botón "+" de cada uno de ellos.

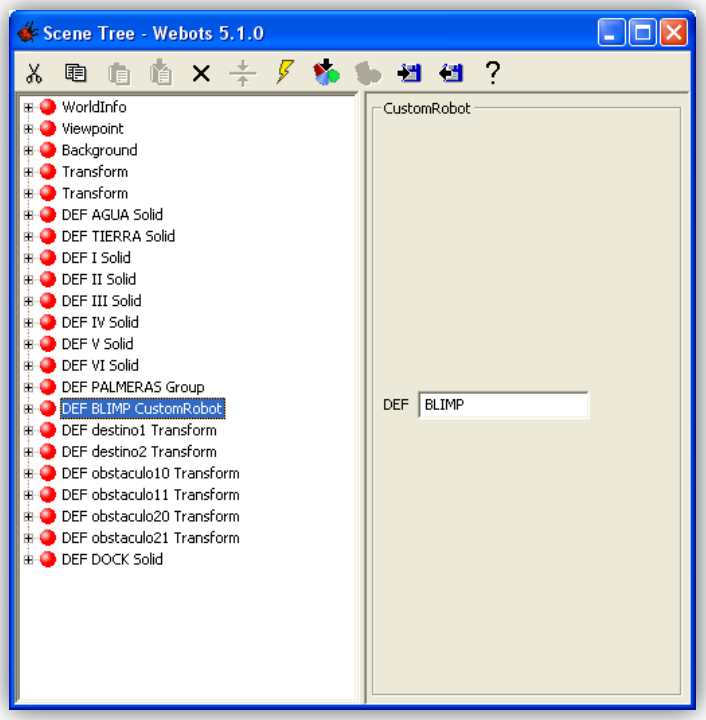

Figura 2: Ventana Scene Tree de Webots 5.1.0

En la parte superior de esta ventana, tenemos una serie de botones con los que manipular los elementos del árbol de escena. De izquierda a derecha son: Cortar, copiar, pegar, pegar después del nodo actual, eliminar nodo, resetear a la última configuración guardada, transformar un nodo en otro, insertar un nuevo nodo después del actual, crear un nuevo nodo, exportar, importar y la ayuda.

El árbol de nuestra escena contiene los nodos típicos de todo mundo virtual reado con VRML: Worldinfo y Viewpoint. Con ellos proporcionamos información de cómo es el mundo y establecemos un punto de vista predeterminado. También tenemos los nodos Background, 2 nodos Transform con las luces, un sólido para representar el agua y otro para la tierra. Los sólidos definidos como del I al VI representan las puertas creadas para las pruebas del robot. Mientras que las palmeras son meros elementos decorativos agrupados en un único nodo. También tenemos un CustomRobot definido como BLIMP, este es nuestro robot volador. Los dos destinos son esferas de color verde y los obstáculos están representados como biesferas. Por último, el sólido DOCK es una estructura creada para el reposo del robot una vez finaliza su recorrido.

2.3- Controladores

En Webots, por una parte tenemos que modelar el entorno de simulación creando un mundo como ya hemos visto que podemos hacer. Pero por otra parte debemos recordar que el gran potencial de este programa reside en la simulación de robots. Por ello, una vez creados todos los objetos del mundo y modelado también nuestro robot, debemos programar las acciones que se llevarán a cabo.

Lo que nos permite Webots es asociar un programa (controlador) a un robot, de tal forma que podemos recibir información de los sensores que hayamos colocado en nuestro robot y mandar información a los actuadores que también le hayamos incorporado. Estos controladores pueden estar escritos en C, C++ o Java. En este proyecto se ha decidido trabajar en C++.

Cuando queremos crear un controlador, debemos hacer click en *Wizard>New robot controller*. Entonces elegimos el lenguaje de programación en que deseamos desarrollarlo y seguidamente el nombre que le vamos a dar a ese nuevo controlador. Automáticamente Webots creará una carpeta con su nombre dentro de la carpeta controllers del directorio de trabajo.

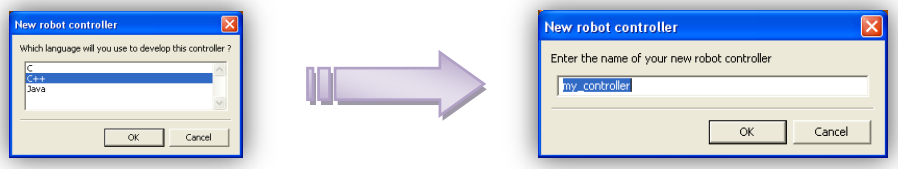

Figura 3: Creación de un controlador

El siguiente paso es asociar ese controlador a un robot en concreto. Pues podemos tener muchos controladores en nuestro directorio de trabajo, pero sólo uno será el que trabaje sobre nuestro robot. Para asociarlo debemos ir a la ventana del árbol de escena que ya hemos visto y desplegar el nodo CustomRobot. Uno de sus atributos es "controller". Si pulsamos sobre él, en la parte derecha de la ventana podemos pulsar sobre el botón de los puntos suspensivos para elegir el controlador a utilizar.

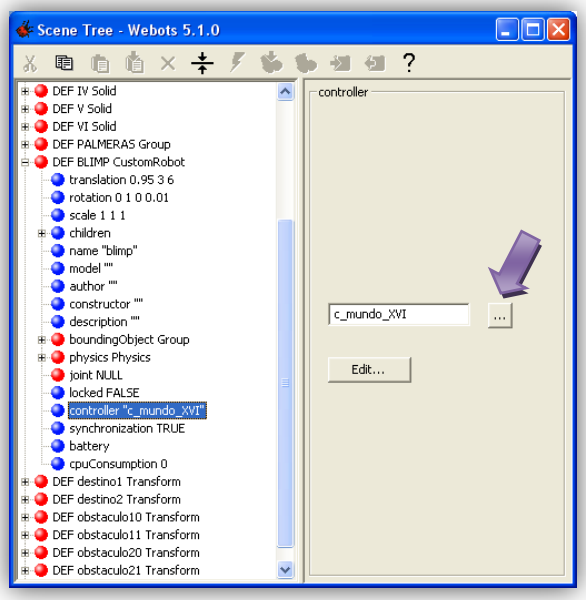

Figura 4: Asociación de un controlador a un CustomRobot

De esta forma creamos el controlador y lo asociamos al robot en cuestión, pero Webots dispone de una ventana más ideada para poder editar el controlador. Esto quiere decir que desde el mismo programa vamos a poder escribir el código del controlador. Esta ventana de edición de texto podemos mostrarla desde *Windows>Text editor*, en la ventana principal.

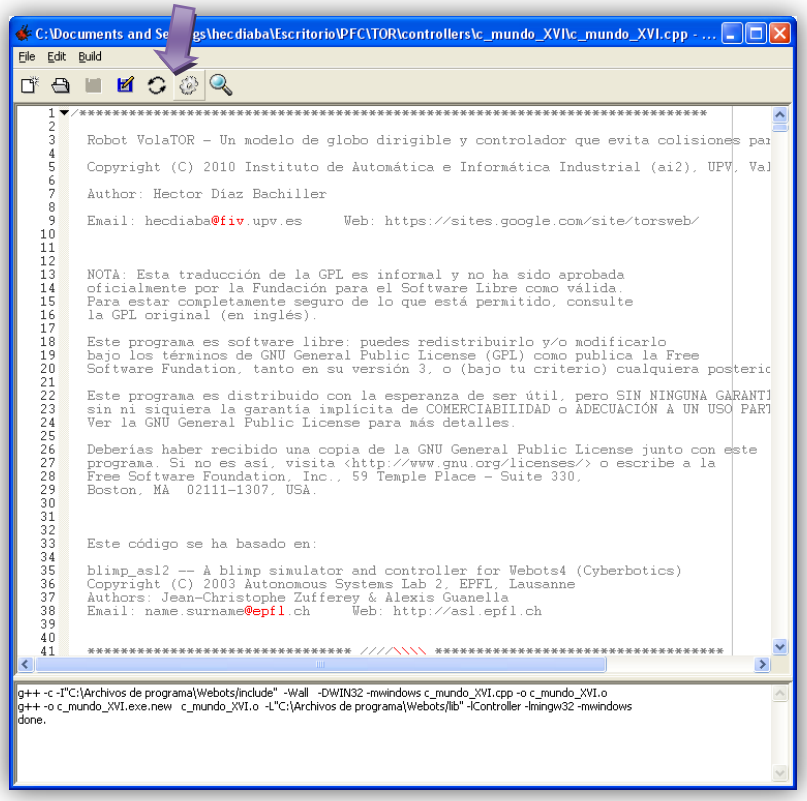

Figura 5: Ventana Text Editor de Webots 5.1.0

La ventana se divide en dos secciones: la primera y principal que es donde se escribe el código, y una segunda en la parte inferior donde se muestran los mensajes de compilación. Se trata de una ventana con las características típicas para crear, abrir y guardar documentos. Pero además de estas funcionalidades, añade el botón de "revert". Este botón devuelve el documento abierto a su última versión guardada. También dispone de la opción "make", que realiza una compilación del código como vemos en la parte inferior de la figura. Esta compilación crea en el directorio del controlador un fichero objeto y otro ejecutable utilizando el "makefile" que contiene esa carpeta.

Aunque para la compilación de los controladores se pueden utilizar otras herramientas, para este proyecto se ha decidido utilizar la opción de compilar desde Webots. Para ello hace falta instalar en el ordenador el entorno MinGW. Éste es un entorno de desarrollo libre basado en gcc, un compilador de código abierto para C y C++. Incluye la utilidad de "make" utilizada en Webots para compilar con el makefile del controlador. Está incluido en el subdirectorio *devel* de Webots para Windows.

2.4- La ventana Log

Por último, queda describir la ventana Log. Como todas las demás, podemos mostrarla desde *Windows>Log*. Es la ventana más simple, pero no por ello menos importante, pues nos va a permitir mostrar todos los datos que necesitemos conocer en tiempo de ejecución. Desde el código podemos mostrar mensajes en esta ventana, pues es la salida estándar de las funciones que muestran datos. Todo lo volcado a esta ventana durante la ejecución se guarda en un fichero *webots.log* en nuestro directorio de trabajo. De esta manera podemos seguir la ejecución y conocer los datos que necesitemos en cualquier momento.

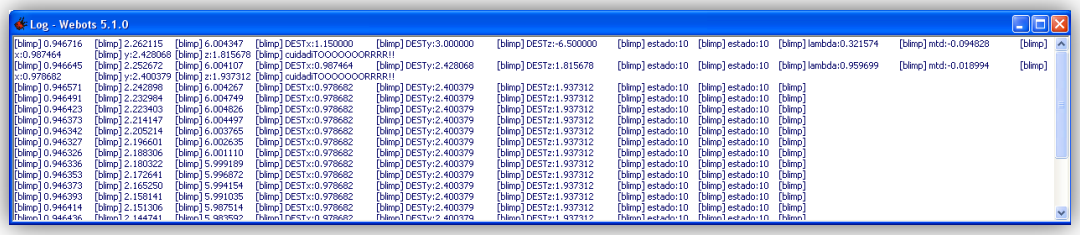

Figura 6: Ventana Log de Webots 5.1.0

3- Entorno de simulación

3.1- Descripción del entorno

En un principio, el entorno de simulación sobre el que se pretendía trabajar con el robot iba a ser lo más parecido posible a un circuito de carreras sobre un río. Sin embargo, debido al tipo de robot volador escogido y a que la creación de mundos amplios se hace un tanto farragosa con Webots, se modificó este entorno dando lugar a uno similar pero con características más adaptadas al proyecto.

El entorno consiste en un río de agua con sus dos orillas de tierra. En las orillas, y a modo de ambientación, hay palmeras para dar un toque más realista. Además, en la orilla derecha del río hay un lugar destinado al docking del globo.

Sobre el agua hay puertas por las que debe pasar el globo volador. Estas puertas, que consisten en dos cilindros separados una cierta distancia, forman el circuito. De que la iluminación sea correcta se encargan dos puntos de luz en el cielo.

Para el desarrollo del proyecto se ha optado por crear unos destinos en este mundo que tienen forma de esfera verde, y envolver los obstáculos con una biesfera roja semitransparente. De esta manera resulta más cómodo trabajar con el robot.

El robot es un globo dirigible blanco que está suspendido en el aire y que después de realizar el circuito marcado va a descansar a la zona violeta de docking.

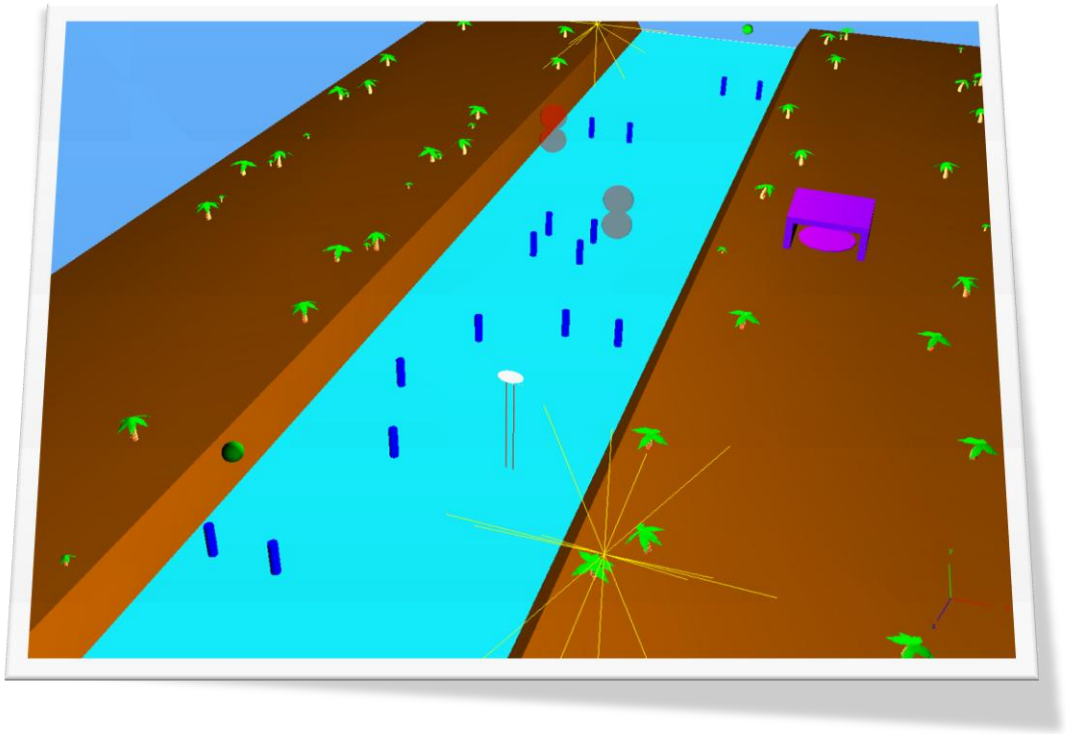

Figura 7: Vista inicial del entorno de simulación

3.2- Diseño de los diferentes elementos 3D

A continuación se va a describir cómo se ha llevado a cabo la creación del entorno 3D descrito en el apartado anterior. Explicando paso a paso de qué objetos y de qué características se compone cada uno de los elementos que forman el mundo. Como ya hemos visto, todo lo relacionado con el entorno virtual se basará en nodos pertenecientes a un subgrupo de nodos de VRML97 más algunos añadidos por Webots.

Debido a su importancia, hablaremos en el primer punto de los componentes del mundo y en un segundo apartado únicamente del modelo del robot volador.

3.2.1- Diseño del mundo

Lo primero que vamos a dejar claro van a ser los ejes de coordenadas con los que vamos a trabajar. Se trata de un sistema dextrógiro:

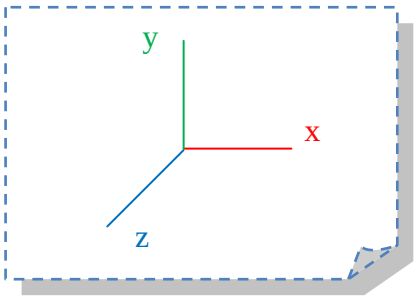

Figura 8: Sistema de coordenadas

Para comenzar, hablaremos de la tierra. Las orillas del río se han modelado mediante una extrusión de color marrón la cual se ha escalado según el factor 2x2x2. La longitud de la extrusión es de 10 metros, y los puntos que forman el *crossSection* son de izquierda a derecha (-3.0,1.0) (-1.0,1.0) (-0.5,0.0) (0.5,0.0) (1.0,1.0) (3.0,0.0), que vendría a ser:

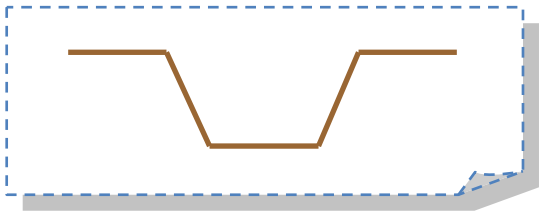

Figura 9: CrossSection de la tierra

A este sólido se le ha añadido un *boundingObject*. Lo forma un grupo de dos *Box*, uno para cada orilla, de la misma longitud que la extrusión. De esta forma será posible calcular las colisiones con las orillas, pero no con el fondo del cauce (no nos interesa).

Para seguir con el río, vamos ahora a comentar cómo se ha creado el agua. También es un sólido, pero su forma es una única *Box* de color azul de 4x2x20 metros. De tal forma que el efecto obtenido junto con las orillas de la tierra es bastante real. Como *boundingObject* se ha utilizado la misma *Box* utilizando la opción DEF de VRML97 para definir nodos y reutilizarlos.

En las orillas del río, es decir, en la tierra encontramos una serie de palmeras de diferentes tamaños. Están pensadas como objeto decorativo y no tienen incorporadas colisiones, pues se trata simplemente de mejorar el aspecto del mundo. Estas palmeras han sido importadas a partir de un modelo VRML97 ya existente. Están agrupadas y se les han aplicado una serie de transformaciones para conseguir esa disposición en concreto.

Con el nodo *Background* lo que hacemos es definir el color del cielo, que para nuestro caso será azul. Y con 2 *PointLight* añadimos la luz necesaria para iluminar todos los objetos. La intensidad de la luz será de 2 unidades para la primera y 4 para la segunda. El color de la luz será blanca para ambas (1,1,1), así como el radio que será de 1000 unidades y sin atenuación. El nodo *Viewpoint* viene por defecto y básicamente lo único que hace es marcar cuál es el punto de vista inicial. Nosotros lo hemos cambiado a uno más acorde con el mundo.

Como decíamos, la idea inicial era crear una especie de circuito, y para ello hemos creado una serie de puertas por las que el robot debe pasar. En concreto son un total de 6 puertas. No todas son iguales, pero son muy similares. Las número 1, 2, 3 y 6 están compuestas por un grupo de dos cilindros azules. La separación entre ambos cilindros es bastante ajustada al tamaño del globo dirigible. El tamaño de cada cilindro es de 0.05 metros de radio y 0,4 m de altura. La puerta 4 es un grupo de 4 cilindros dispuestos formando un cuadrado, mientras que la puerta 5 consta de 3 cilindros en zigzag. Todos los cilindros que forman las puertas son sólidos y también se les aplican colisiones. Aunque para el proceso de desarrollo éste ha sido desactivado.

Los destinos que se le marcan al zepelín han sido modelados en el mundo como esferas verdes de 0,1 metros de radio con 20 subdivisiones. Por otro lado, los obstáculos marcados para evitar se muestran en el mundo como biesferas semitransparentes de color rojo. Cada obstáculo está formado por dos esferas de 0,2 metros de radio y 20 subdivisiones. Por lo tanto, el centro de la esfera superior queda desplazado (con respecto al centro del obstáculo) 0,2 metros positivos en el eje Y, mientras que la esfera inferior lo hará en el sentido negativo del eje Y.

Sólo queda describir el dock, pues el diseño del robot lo dejaremos para el apartado siguiente. El dock es una estructura de color rosa donde el robot irá a descansar una vez haya finalizado su recorrido. Consta de 4 piezas agrupadas para ser reutilizadas como *boundingObject*. En concreto son 2 paredes de 0.1x0.5x1.0 metros separadas 1 metro de distancia en el eje X. Un techo de 1.1x0.1x1.0 metros colocado sobre las paredes anteriores. Y un suelo con forma de cilindro de 0,4 metros de radio y 0,05 metros de altura dispuesto en el centro de la parte inferior de la estructura.

3.2.2- Diseño del robot volador

Vamos a ver una imagen del robot real y otra del modelo del globo dirigible utilizado en Webots señalando las diferentes partes de las que se compone. De esta forma podremos entender de forma visual cómo es el robot y cómo se ha modelado. Cabe decir que este modelo es una modificación del robot *blimp_asl2* que viene como ejemplo en Webots.

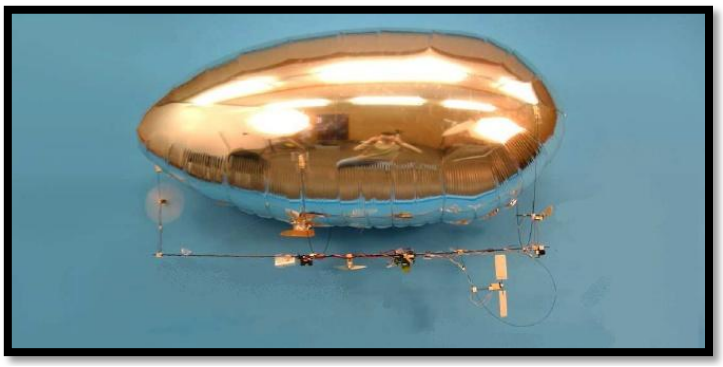

Figura 10: Robot volador real construido en la EPFL

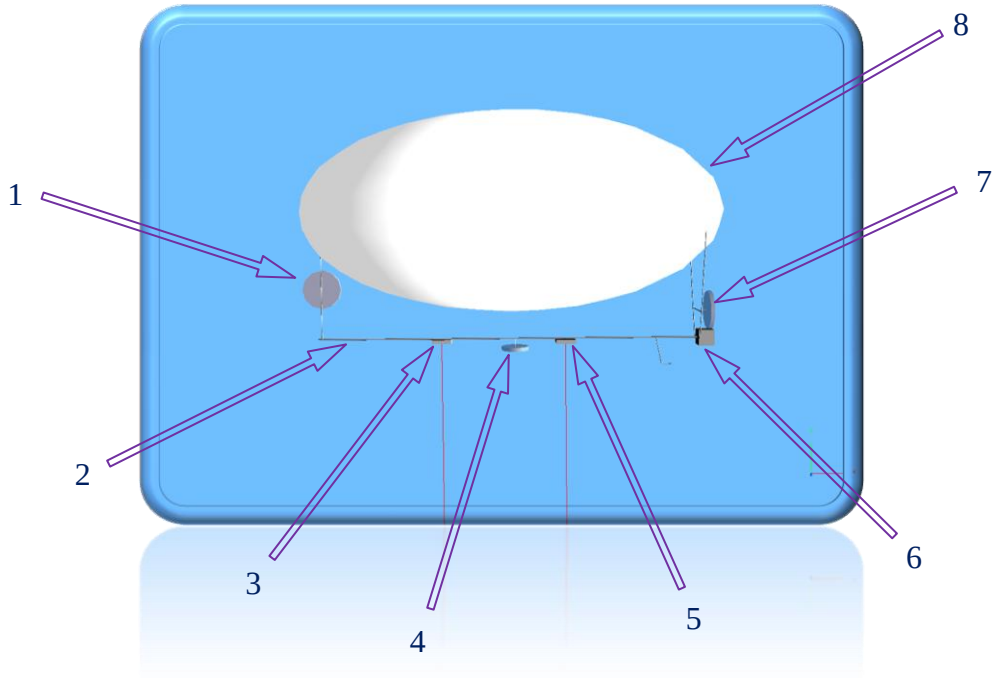

Figura 11: Modelo con las partes del robot volador

A continuación se muestra la lista de elementos que podemos ver en la imagen:

1.- Motor de YAW + hélice
2.- Estructura del globo dirigible
3.- Baterías + sensor de distancia
4.- Motor de ascenso/descenso + hélice + GPS
5.- Procesador + sensor de distancia
6.- Cámara 800x600 color
7.- Motor de avance/retroceso + hélice
8.- Globo de helio

La estructura principal (2) sobre la que se acopla el resto de componentes del robot está modelada a base de cilindros de color grisáceo para conseguir el efecto metálico. Sobre ésta, se sujeta el globo de helio (8) modelado como una esfera blanca. La forma se consigue gracias a un escalado algo mayor al doble en el eje X que en los ejes Y y Z.

Los 3 motores de los que consta el robot para realizar sus movimientos se han montado sobre la estructura y tienen forma de cilindro para simular las hélices. El primero de ellos (1) permite la rotación del globo sobre su eje vertical, es decir, con este motor se consigue lo que llamamos yaw. Se ha montado en la parte trasera. En la parte central encontramos el segundo motor (4), que nos permitirá ascender y descender, y el receptor GPS. Por último y en la parte delantera está el tercer motor (7). Es el encargado de hacer avanzar y retroceder al robot.

A cada uno de los lados del segundo motor encontramos una caja también de color metálico. La que está en la parte trasera corresponde a las baterías del robot (3). En esa posición se ha añadido un sensor de distancia para controlar la altitud del globo. Por otro lado, la caja de la parte delantera simula el microprocesador (5) y otro sensor de distancia con el mismo propósito que el anterior. Únicamente falta hablar de la cámara (6) que se sitúa en la parte frontal del robot. Para modelarla se ha utilizado otra caja (esta vez algo mayor) de un material similar al resto.

3.3- Diseño del controlador

Aunque en Webots podemos crear un mundo con varios robots y cada uno de ellos puede tener asociado un controlador, en nuestro proyecto únicamente existe un robot y bastará con implementar un controlador para él.

Vamos a dividir el diseño del controlador en 4 partes bien diferenciadas. Por una parte hablaremos de cómo se ha diseñado la entrada de datos. Es decir, dónde y cómo se ubican esos datos que el usuario debe introducir para utilizar este proyecto. También tenemos la parte de controlar el movimiento del robot, ya que el mismo puede trabajar de forma automática o manual. El robot deberá detectar los obstáculos que le harán colisionar de entre todos los obstáculos que existan en el mundo. Y por último deberá ser capaz de evitar estos obstáculos de tal forma que logre llegar a los destinos marcados sin colisionar en ningún momento.

A parte de todo lo mencionado, como ya hablamos anteriormente, los cálculos físicos del mundo se harán también en el controlador del robot. De esta forma, podemos controlar mejor todos los aspectos relacionados con el rozamiento, la inercia, etc. que para un robot aéreo cobran una mayor importancia que si trabajáramos con robots terrestres.

3.3.1- Entrada de datos

Los datos que el usuario debe introducir para que el controlador pueda trabajar se dividen en dos partes. Los correspondientes al circuito que se desea que realice el globo dirigible y los que pertenecen a los obstáculos que el globo debe esquivar.

Los primeros son posiciones tridimensionales del mundo que deben estar en zonas accesibles por el robot. Por ejemplo, un destino no debe estar nunca dentro del espacio que corresponde al agua. El robot evitaría sumergirse gracias al tratamiento que se le ha dado a los sensores de distancia que controlan la altitud, pero es un caso que no debería darse. Con estas posiciones colocadas en forma de pila vamos a marcar el circuito que queremos que recorra el robot. O lo que es lo mismo, el robot irá de destino a destino siguiendo el orden en que los coloquemos. Debemos tener en cuenta que las 3 últimas posiciones corresponden a la maniobra de docking y por lo tanto no deberemos modificarlas a no ser que sea necesario.

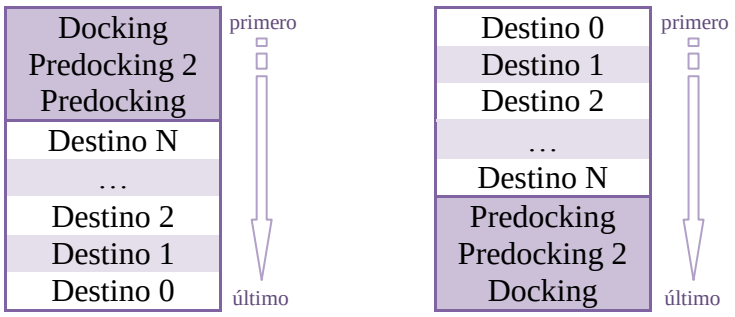

Figura 12: Orden de introducción en la pila (izquierda). Orden de extracción de la pila (derecha)

La introducción de estos datos que marcan los destinos del robot debe hacerse directamente sobre el código del controlador. Una posible extensión de este proyecto

sería adaptar el código para que la pila donde se guardan los destinos fuera rellenada con la lectura de un fichero de entrada que los contuviera. Pero estos temas los trataremos más adelante hacia el final de la memoria.

También debemos introducir los datos de los obstáculos. Estos son biesferas, y por tanto el dato que introduciremos para cada biesfera será el punto de unión de ambas esferas, lo que podríamos llamar centro del obstáculo. En esta ocasión la forma de introducirlo también será directamente en el código del controlador y será en forma de vector, pues el orden no importa a la hora de almacenar los obstáculos. Como antes, se trata de una posición 3D. Pero otro dato importante que debemos introducir sobre los obstáculos es el radio de sus esferas. En un principio el código está preparado para obstáculos de mismo radio, aunque no sería complicado adaptarlo para que los obstáculos pudieran tener radios de diferente longitud.

3.3.2- El movimiento del robot

Nuestro robot consta de 3 motores como ya hemos visto en la parte del diseño del robot. Cada uno de ellos va a permitir al robot realizar diferentes movimientos. Para poder referirnos más claramente a cada uno de ellos y evitar confusiones los numeraremos a partir de ahora de la siguiente manera. Cuando hagamos referencia al motor 1, estaremos hablando del motor ubicado en la parte delantera del robot (7 en la Figura 11: Modelo con las partes del robot volador). El motor 2 será el de la zona baja del robot (4 en la Figura 11). Mientras que hablaremos del motor 3 cuando nos refiramos al de la parte trasera (1 en la Figura 11).

Llegados a este punto debemos hacer una distinción entre el control del robot de forma manual y el control de forma automática. Ya que el controlador del robot está preparado para poder cambiar de un modo al otro pulsando la tecla "P". En los dos casos vamos a terminar trabajando en términos de la aceleración que le aplicamos a los motores, pero en el caso manual, ésta la controlará el usuario a su gusto con la ayuda del teclado y en el caso automático será el propio controlador el que se encargue de calcularla en todo momento.

Para conocer en profundidad el funcionamiento de la física existente en el controlador y causante del movimiento del robot, se puede consultar el documento original de Jean-Christophe Zufferey et al. *"Flying over the Reality Gap: From Simulated to Real Indoor Airships", 2006* Bibliografía. Pero de momento bastará con tener constancia de que al tratarse de un robot aéreo no vamos a trabajar con velocidad como haríamos en un robot terrestre sino que nuestra herramienta para realizar los movimientos será la aceleración de los motores. Así pues, lo que haremos será aplicar un multiplicador de la aceleración a cada motor y de esta forma incrementar o disminuir el valor final de aceleración que se le aplica al motor en cuestión.

Debemos saber que el sistema de coordenadas del globo es diferente al del mundo virtual de Webots. Aunque esto se solventa en la programación y a efectos prácticos el usuario no lo nota en ningún momento. Ni siquiera cuando el control del globo dirigible lo realiza él.

Como la manera de utilizar el robot de forma manual se explicará en el apartado 4.4, vamos a explicar a continuación el funcionamiento automático del globo.

Subir/Bajar

Al tratarse de un robot volador lo primero que vamos a tener que controlar es la altitud a la que se encuentra el robot. Podemos conocerla gracias a la posición que nos proporciona el nodo GPS. Y así, sabiendo la altitud a la que se encuentra el globo, podemos actuar sobre el motor 2 para llevarlo a otra diferente, con lo que conseguiríamos un movimiento de subir o bajar el globo en el espacio. Aunque siempre debemos tener en cuenta que si dejamos de actuar sobre el motor, el globo dirigible caerá por la acción de la aceleración gravitatoria que se ha añadido a las físicas del mundo. Por tanto, en todo momento (aunque lo que deseemos sea dejar el robot inmóvil en un punto) el motor 2 deberá ser controlado y estar actuando.

Para que el globo dirigible no se descontrole en ningún momento, cuando los sensores de distancia ubicados en la parte inferior del globo detectan que éste se encuentra demasiado cerca de la superficie, el motor 2 se pone en funcionamiento en una maniobra de emergencia que le proporciona más altitud. De la misma manera, pero utilizando los datos del GPS en lugar de los de los sensores de distancia, cuando el globo se eleva fuera del rango deseado también se inicia una maniobra automática de emergencia que lo estabiliza a una altitud correcta por debajo de ese umbral.

Sin embargo mientras el globo se encuentra en la zona de maniobras deseada, la altitud es controlada por el motor 2 en función de la altitud actual y la altitud del próximo destino. Pues los destinos son posiciones 3D y nos interesa alcanzarlos con la mayor exactitud posible, incluida la altitud.

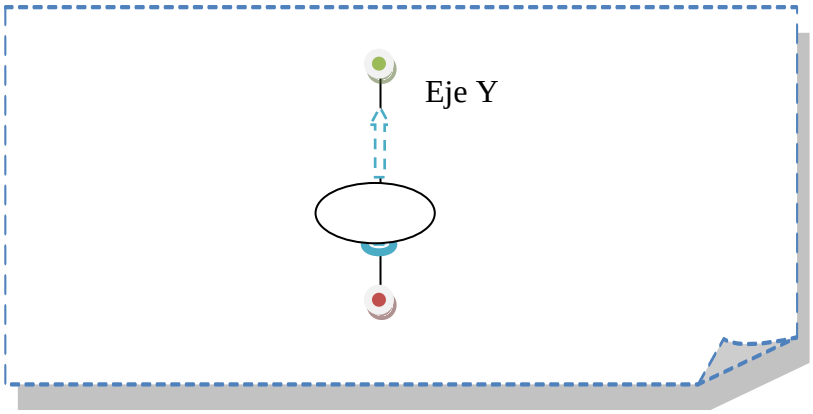

Figura 13: Movimiento Subir/Bajar del globo dirigible

Por lo tanto, dentro del controlador que se encarga de supervisar todo lo relacionado con el robot, necesitaremos programar un "controlador" que actúe sobre el motor 2. Pero debemos distinguir entre el controlador que le decimos a Webots que asigne a nuestro robot y una parte de código de este controlador que se encargará únicamente de actuar sobre el motor 2.

A continuación explicaremos cómo, después de estudiar el caso, el robot, la física asociada, etc. hemos escogido el tipo de controlador y las características que lo hacían útil para nuestro proyecto. La tarea de este controlador es saber en cada momento cuánto se debe actuar sobre el motor 2 para que la nueva altitud del globo se alcance de forma correcta en el período de tiempo correcto.

De entre los tipos de controlador que podríamos haber escogido, después de varias pruebas hemos optado por un controlador proporcional-derivativo. La parte integral de un PID no nos aportaba nada útil y por este motivo no la hemos incorporado. Los valores que se aplican a cada una de las ganancias, varían según el estado en el que se encuentra el robot. Esto es así debido a que no se puede aplicar la misma acción si por ejemplo lo único que queremos es mantenernos a una altitud fija, que si además debemos estar en continuo movimiento. Estos valores forman parte de la programación del controlador y se pueden ver en el código del robot.

Además de la parte de programación correspondiente a controlar la potencia, hace falta cambiar el sentido de ésta ya que el motor 2 está ubicado en el globo de forma que valores positivos hacen bajar al robot. Pero esto es una simple cuestión de programación relacionada con el diseño que está comentada en el código y no implica nada más que un cambio de signo a la variable que controla la potencia suministrada.

Avanzar/Retroceder

Partimos de la base de que el robot se encuentra en una posición actual (origen) y debe llegar a una posición final (destino). En cada iteración el origen habrá cambiado si ha habido movimiento, pero el destino permanecerá sin cambios hasta que se alcance. Para comenzar, pondremos como condición que ambas posiciones se encuentren en el mismo eje (paralelo al eje Z del mundo), pues nuestro mundo tiene profundidad en ese eje y es hacia donde nos interesa que se realice el movimiento. Y además, el robot estará orientado hacia el destino. De esta forma, poner en funcionamiento el motor 1 con un multiplicador positivo significará que el robot avanza hacia adelante a la vez que se va acercando al destino (*16*). Si utilizáramos un multiplicador negativo, el robot se movería hacia atrás, alejándose del destino. Ya que esto implica que el motor gire en sentido contrario y con este la hélice que aporta el movimiento.

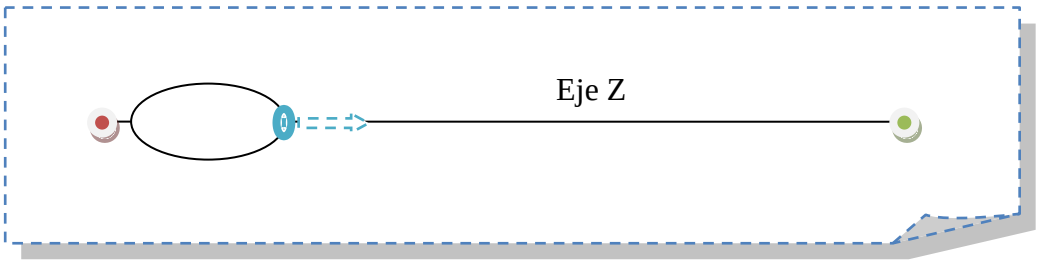

Figura 14: Movimiento Avanzar/Retroceder del robot

Ya sabemos cómo vamos a comenzar a realizar el movimiento más sencillo, pero tenemos que tener en cuenta que no nos vale únicamente con llegar al destino. Si estuviéramos trabajando con un vehículo terrestre el rozamiento actuaría en nuestro favor y al dejar de suministrar potencia a los motores, el vehículo prácticamente pararía en el sitio correcto. Pero como siempre, al tratarse de un vehículo aéreo el rozamiento actuará en menor medida y la inercia hará que el robot se desplace mucho más allá del destino si lo único que hacemos es dejar de suministrar potencia a los motores al llegar al destino. Esto no es viable para nuestro proyecto y debemos controlar esta situación. Para ello lo que haremos será incorporar un controlador proporcional-derivativo al código de manera que actúe sobre la variable del multiplicador de potencia. Así lo que estaremos haciendo es controlar la potencia que se le suministra al motor 1 teniendo en

cuenta la distancia a la que se encuentra el robot del destino. Cuando el robot se encuentre relativamente lejos, la potencia será mayor y el robot se moverá más rápidamente. Sin embargo, conforme el robot se vaya acercando al destino, la potencia con la que se actuará sobre el motor 1 irá disminuyéndose hasta el punto de utilizar incluso un multiplicador negativo para conseguir que la velocidad del globo baje hasta valores cercanos a cero en el punto del destino. De esta forma, nos estamos adelantando a lo que sucedería y conseguimos actuar antes de que el globo sobrepase el destino parándolo en el lugar correcto.

Todo este funcionamiento lo realiza de nuevo un controlador, pero esta vez aplicado a la variable encargada de ajustar la potencia del motor 1. Dicho controlador es del tipo proporcional-derivativo, pero el ajuste de las variables implicadas en el controlador es diferente al del motor 2. Además, igual que en el caso anterior, según el estado en que se encuentre el robot y en el que tenga que actuar este controlador, los valores de las variables cambiarán. Hay que darse cuenta de que así como cuando trabajábamos con el motor 2 era necesario que estuviera actuando de forma continua, en este caso no va a ser necesario puesto que no hay ninguna fuerza gravitatoria ni de ningún otro tipo que esté actuando continuamente sobre el globo. Todas las fuerzas que se ejercen en esta dirección provienen del propio movimiento del robot como por ejemplo la inercia. Lo que se trata de explicar es que habrá que controlar fuerzas como la inercia con este motor, pero no fuerzas continuas como la gravedad.

Izquierda/Derecha

Nuestro globo consta de un tercer motor (motor 3) que le permite realizar giros en el espacio. Pero estos giros tienen una característica muy importante y que ha llevado más tiempo controlar que los demás movimientos. Y es que estamos hablando de un motor que está situado en la cola del globo dirigible a modo de timón. Esto provoca que las fuerzas ejercidas por la hélice de este motor hagan girar la parte trasera del globo. Y con ello se consigue un giro característico muy diferente al que podemos obtener con un robot terrestre que conste de ruedas diferenciales por ejemplo. Esto es debido a que el eje de rotación deja de estar en el centro o en uno de los laterales y pasa a estar en la parte delantera.

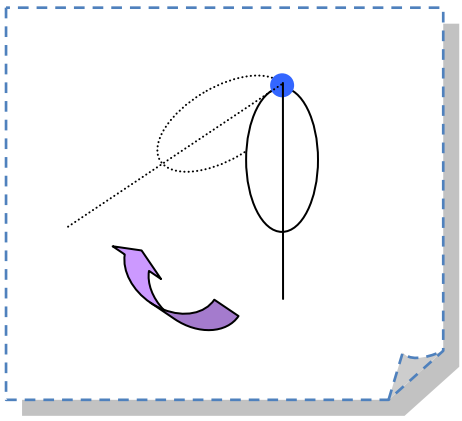

Figura 15: Movimiento de rotación Izquierda/Derecha del robot

Realizar este tipo de giros no es trivial y de nuevo es necesario aplicar un controlador sobre el multiplicador de la potencia del motor 3 para lograr que los giros sean correctos y a la vez precisos.

Lo primero que se intentó para solventar esta dificultad fue en primer lugar buscar la orientación correcta hacia el destino con la ayuda de unos parámetros concretos de controlador PD y una vez orientado el globo dejar de actuar sobre el motor 3 para comenzar un movimiento rectilíneo. El problema es que lo que se conseguía era un movimiento muy artificial, en el sentido de que los movimientos se dividían en fases y además la precisión era inaceptable debido sobre todo a la inercia que seguía actuando sobre el robot.

Así que se abordó el problema de otra manera. Lo que se ha hecho ha sido aprovechar el autómata de estados creado para el robot. En el estado de "BUSCA_ANGULO" se utiliza un controlador PD para orientar el robot hacia su destino al igual que comentábamos anteriormente. Y una vez se consigue tener orientado el globo hacia el destino con un error lo suficientemente pequeño, se pasa al estado "AVANZA".

En el estado "AVANZA", además de seguir controlando los dos movimientos vistos anteriormente, se empieza a controlar el motor 3 de una forma especial. Se hace con un controlador PID, similar al del estado "BUSCA_ANGULO" sólo que esta vez añadimos la parte integral del controlador para obtener el mínimo error posible en el transitorio. Pero con esto únicamente no era suficiente y la opción por la que se optó fue aplicar otro controlador PID pero esta vez sobre el error cometido en el eje X. Es decir, con un primer controlador actuamos para corregir el ángulo del robot, y con el segundo obligamos a corregirlo más o menos dependiendo de lo lejos que se encuentre el globo dirigible del eje paralelo al eje Z que debe alcanzar el globo para llegar al destino. De forma gráfica sería de la siguiente manera:

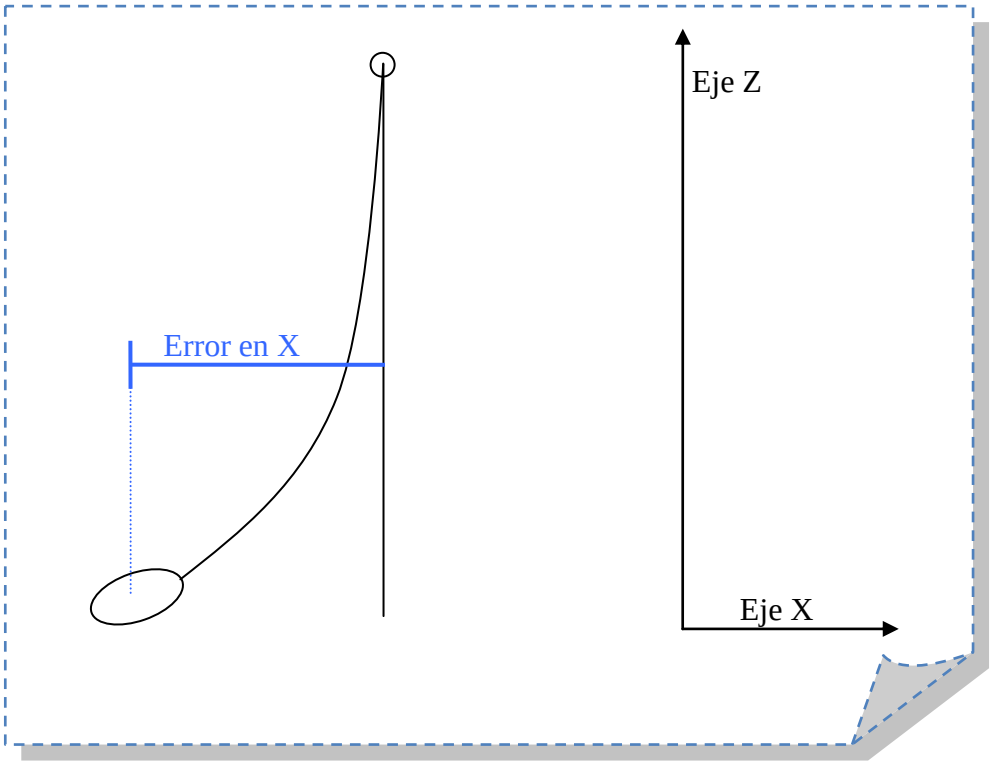

Figura 16: Trayectoria del robot

El último aspecto a tener en cuenta en este caso es que como estamos trabajando con ángulos de Euler deberemos tener especial atención en tratar los casos de ángulos positivos y negativos de la manera adecuada. De ahí que el código esté duplicado en ese aspecto, para tener en cuenta en qué caso de los dos se encuentra y cómo actuar.

Movimiento conjunto

Ya hemos explicado uno a uno los tipos de movimiento que el robot puede realizar y la forma de controlarlos por separado. Pero esto sólo fue el principio, pues hasta que no se tuvo controlada la altitud, no se comenzó a estudiar el caso de mover el robot en línea recta. Y una vez encontrada la configuración adecuada para llegar a un punto situado en línea recta a una altura diferente, se incorporó el controlador del giro. Primero simplemente realizar un giro buscando la orientación correcta hacia el destino, permaneciendo el globo estático. Más tarde añadiendo el movimiento y el controlador explicado más arriba para alcanzar la posición de destino siguiendo su eje paralelo al eje Z. Llegado a ese punto funcionaba, pero no era posible controlar la inercia como era necesario y el robot llegaba al destino con demasiado error en el eje X. La solución vino al tener en cuenta, en el controlador del motor 1, la línea recta que une la posición actual con el destino en lugar del error en el eje Z. Lo que podríamos llamar la hipotenusa del triángulo recto formado por el origen y el destino. Teniendo esto último en cuenta es como se han obtenido los resultados correctos y de esta forma el robot realiza los movimientos de un destino a otro con un error mínimo.

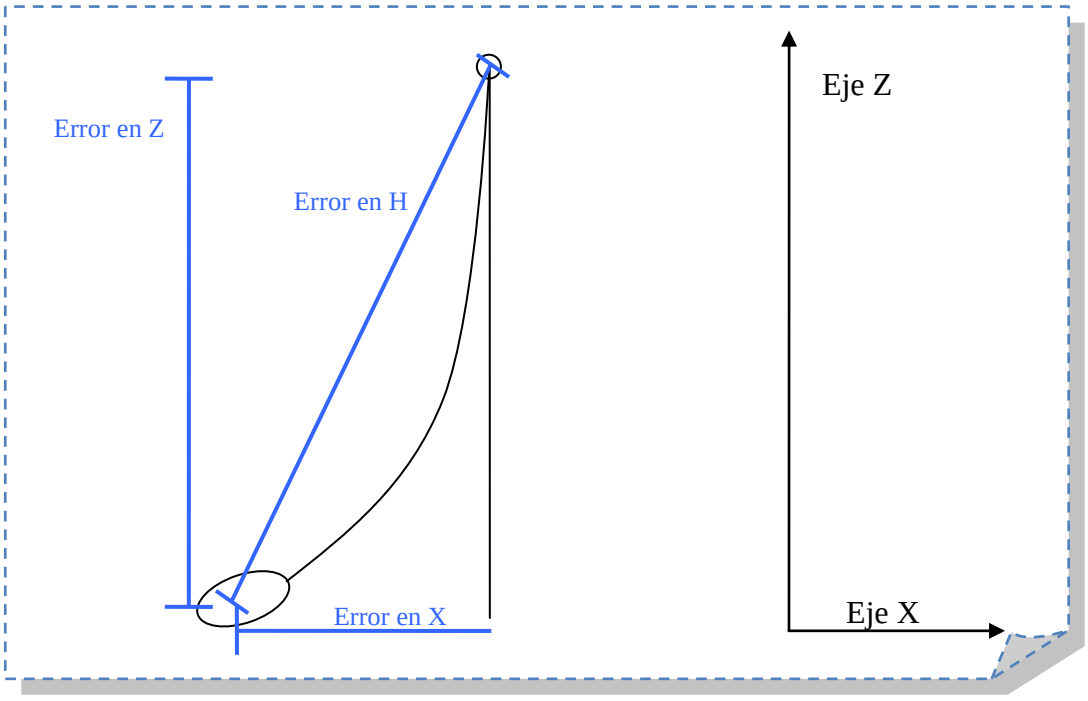

Figura 17: Movimiento conjunto y errores en los ejes

3.3.3- Detección de colisiones

Por el momento tenemos controlado todo lo relacionado con el movimiento del robot. Ya hemos visto cómo, y somos capaces de mandar al robot desde un origen a cualquier destino. El globo dirigible llegará a ese punto cometiendo un error mínimo dentro de los márgenes aceptables. Pero vamos a introducir un elemento más al proyecto que son los obstáculos.

Los obstáculos los vamos a diseñar con forma de cilindro, pues recordemos que la idea original era realizar un circuito. Las puertas de los circuitos serán nuestros obstáculos y por eso se ha elegido el cilindro para representarlas.

Además de modelarlas en el entorno visual de Webots, debemos incorporarlas al código del controlador. Y como los obstáculos van a ser objetos conocidos de antemano, los vamos a representar en memoria como un vector de obstáculos. Cada obstáculo se representará como un punto del espacio 3D. Este punto que almacenamos es el punto que se encuentra en el centro del cilindro.

Hay que hacer una aclaración en este punto y es que se ha considerado para este proyecto que todos los obstáculos son de igual tamaño. Pero una posible ampliación del proyecto sería no sólo almacenar el centro del cilindro, sino también su tamaño. De esta forma estaríamos permitiendo trabajar más tarde con obstáculos de diferente tamaño.

Pero vamos a explicar la manera de detectar obstáculos que hemos implementado en el robot. Porque como hemos dicho, ya somos capaces de llevar el globo dirigible de un punto a otro, pero ahora necesitamos que detecte los obstáculos que hay en su trayectoria para saber si colisionará con ellos o no. O lo que es lo mismo, detectar los obstáculos, para más tarde generar automáticamente trayectorias libres de colisión. Para ello se ha creado una rutina que es llamada para cada uno de los obstáculos del entorno (recordemos que todos los obstáculos son conocidos y están guardados en un vector). A la rutina "detecta_colision" se le pasan como argumentos el origen, el destino y una biesfera creada a partir de la posición del obstáculo.

El funcionamiento de la rutina consiste en calcular los valores que nos permitirán saber si estando en ese origen y queriendo llegar a ese destino, colisionará con ese obstáculo. La rutina tiene implementada la primera parte del algoritmo de **cálculo de distancias entre Poli-Esferas** Bibliografía. La segunda parte de este algoritmo será explicada más adelante en el apartado 3.3.4- Evitación de obstáculos.

Lo que va a hacer la rutina es calcular las diferencias de Minkowski Bibliografía a partir de las 4 esferas que tiene como datos de entrada; origen, destino y las dos esferas que forman la bi-esfera del obstáculo. Formando así una tetra-esfera o poliesfera de Minkowski donde podemos saber que si el punto origen está dentro habrá colisión, y si está fuera no la habrá.

$$\begin{aligned} M_0\ (C_s - C_0,\ r_s + r_0) \\ M_1\ (C_s - C_1,\ r_s + r_1) \\ M_2\ (C_f - C_0,\ r_f + r_0) \\ M_3\ (C_f - C_1,\ r_f + r_1) \end{aligned} \quad \textbf{(1)}$$

Para conocer si un punto está dentro o no de la poliesfera de Minkowski será necesario hacer uso de la transformada de Hough Bibliografía ya que las rectas de Hough nos van a servir tanto para esto como para saber cuál es la distancia que hace que el radio de la esfera quede en la superficie de la poliesfera.

Una vez conocemos los valores de la poliesfera de Minkowski, vamos a trabajar con las 2 tri-esferas que la constituyen si la dividimos en dos, y también con las distintas bi-esferas que la forman. Para conocer la posición del origen O vamos a redefinirlo de la siguiente forma:

$$O = (\lambda_{O1}, \lambda_{O2}, d_O) \tag{2}$$

Para simplificar diremos que λ_{O1} y λ_{O2} son la posición relativa del origen respecto a los centros de una de las tri-esferas. Mientras que d_O es la distancia del origen al triángulo que forman estos centros.

$$\lambda_{01} = \frac{\left(\vec{C}_{02}\cdot C_0\right)\cdot\left(\vec{C}_{01}\cdot\vec{C}_{02}\right) - \left(\vec{C}_{01}\cdot C_0\right)\cdot\|\vec{C}_{02}\|^2}{\|\vec{C}_{01}\|^2\cdot\|\vec{C}_{02}\|^2 - \left(\vec{C}_{01}\cdot\vec{C}_{02}\right)^2}$$

$$\lambda_{02} = \frac{\left(\vec{C}_{01}\cdot C_0\right)\cdot\left(\vec{C}_{01}\cdot\vec{C}_{02}\right) - \left(\vec{C}_{02}\cdot C_0\right)\cdot\|\vec{C}_{01}\|^2}{\|\vec{C}_{01}\|^2\cdot\|\vec{C}_{02}\|^2 - \left(\vec{C}_{01}\cdot\vec{C}_{02}\right)^2}$$

$$d0 = \|O^{\perp}\| \tag{3}$$

Dependiendo de los valores de las lambdas y de su suma, vamos a tener que realizar unos cálculos u otros, pues podemos encontrarnos en el caso de una tri-esfera, en el de una bi-esfera, en el del mínimo entre 2 bi-esferas o también puede afectar a la otra tri-esfera y darse estos mismos casos. Pero una vez calculadas λ_{O1}, λ_{O2} y $\lambda_{O1} + \lambda_{O2}$ ya sabemos en qué caso estaremos y podremos consultar las tablas para saber qué cálculos realizar.

λ'_{01}	λ_{02}	$\lambda'_{01} + \lambda_{02}$	**MTD(O,S_{032})**
≥0	≥0	≤1	MTD(O,S_{032})
<0	≥0	≤1	MTD(O,S_{02})
≥0	<0	≤1	AFECTA A LA TRIESFERA S_{013}
<0	<0	≤1	*min*(MTD(O,S_{01}), MTD(O,S_{02}))
≥0	≥0	>1	MTD(O,S_{32})
<0	≥0	>1	*min*(MTD(O,S_{32}), MTD(O,S_{02}))
≥0	<0	>1	AFECTA A LA TRIESFERA S_{013}

(4)

En el caso de que los cálculos indiquen que se debe trabajar con la otra triesfera, antes de nada es imprescindible realizar un pequeño ajuste en las lambdas y después se puede proceder a continuar con los cálculos:

$$\lambda_{01} = -\lambda_{02}$$
$$\lambda'_{02} = \lambda'_{01} + \lambda_{02}$$

λ_{01}	λ'_{02}	$\lambda_{01} + \lambda'_{02}$	**MTD(O,S_{013})**
≥ 0	≥ 0	≤ 1	MTD(O,S_{013})
< 0	≥ 0	≤ 1	NO PROCEDE
≥ 0	< 0	≤ 1	MTD(O,S_{01})
< 0	< 0	≤ 1	*NO PROCEDE*
≥ 0	≥ 0	> 1	MTD(O,S_{13})
< 0	≥ 0	> 1	NO PROCEDE
≥ 0	< 0	> 1	*min*(MTD(O,S_{01}), MTD(O,S_{13}))

(5)

De tratarse de una triesfera sólo nos queda calcular d_O a partir de la proyección del origen sobre la triesfera. Con esto ya somos capaces de calcular la MTD (Mínima Distancia de Translación) necesaria para dejar el radio en la superficie de la poliesfera de Minkowski.

$$d0 = \| O^{\perp} \|$$

$$O^{\perp} = C_0 + \lambda_{O1}(C_1 - C_0) + \lambda_{O2}(C_2 - C_0)$$

$$MTD = d_0 - \left(C_0 + \lambda_{01} \cdot \left(C_1 + r_0 \right) + \lambda_{02} \cdot \left(C_2 + r_0 \right) \right)$$

(6)

En caso de tener que calcular la distancia a una biesfera debemos calcular una lambda diferente además de la proyección del origen sobre la biesfera. Y con esto y los radios de los centros de la biesfera ya podríamos calcular la MTD. Hay que tener en cuenta que si λ es menor que 0 la MTD será la distancia al centro de la primera esfera menos su radio. Y si es mayor que 1 estaremos en el mismo caso pero de la segunda esfera de la biesfera.

$$\lambda = \frac{-C_0 \cdot \left(C_1 - C_0 \right)}{\| C_1 - C_0 \|^2}$$

$$O^{\perp} = C_0 + \lambda_{O1}(C_1 - C_0)$$

$$MTD = \|O^{\perp}\| - (r_0 + \lambda \cdot (r_1 - r_0))$$
$$\text{Si } \lambda < 0 \qquad MTD = \|C_0\| - r_0$$
$$\text{Si } \lambda > 1 \qquad MTD = \|C_1\| - r_1$$

(7)

Llegados a este punto, puede que hayamos tenido que calcular la distancia a una tri-esfera, a una bi-esfera, a una esfera o a dos bi-esferas y calcular la mínima. Pero en cualquier caso la rutina devolverá la MTD calculada, las lambdas necesarias y la proyección del origen.

Como todo el proceso de la rutina se hace para cada obstáculo, finalmente sólo queda comprobar si la MTD es mayor que 0 (no habrá colisión con ese obstáculo) o si por el contrario la MTD es menor o igual a 0 significando esto que el robot colisionará con ese obstáculo a no ser que se evite de alguna manera. Con esto llegamos al siguiente apartado de esta memoria, donde se explica la técnica seguida para evitar colisionar con los obstáculos. Si aún así se necesita profundizar más cualquiera de los aspectos del algoritmo de **cálculo de distancias entre Poli-Esferas**, de nuevo se puede encontrar una amplia explicación en Bibliografía.

3.3.4- Evitación de obstáculos

El cometido del apartado anterior no es otro sino el de realizar los cálculos previos a este. La posición de los obstáculos la conocíamos, y ahora gracias a la detección de obstáculos sabemos si el globo dirigible va a colisionar con ellos. Lo que debemos hacer en este apartado es por un lado encontrar el primer obstáculo con el que va a colisionar el robot y después utilizar los cálculos realizados para mandar al robot a una posición de no colisión.

Como ya tenemos calculada la MTD de cada obstáculo y su posición, lo que vamos a hacer es recorrer el vector de obstáculos buscando aquel que cumpla 4 condiciones; ser uno de los que colisionan con el robot, ser el más cercano al robot, estar por delante del robot y estar antes del destino del robot. Con esto tenemos el obstáculo que debemos evitar.

Siguiendo con el algoritmo de cálculo de distancias entre Poli-Esferas, podemos utilizar los datos obtenidos para calcular cuál será el punto de máxima penetración entre el origen y la biesfera del obstáculo. Y teniendo este punto, lo único que deberemos hacer para evitar la colisión será introducir un nuevo destino intermedio desplazando este punto de máxima penetración la distancia necesaria en el eje del origen proyectado. Sin embargo, en caso de no ser suficiente el margen de error, podemos añadir de forma opcional un porcentaje como umbral de confianza.

$$C_{max\_penetracion} = C_s + \lambda \cdot (C_f - C_s)$$

$$C_{punto\_intermedio} = C_{max\_penetracion} - [umbral(\%)] \cdot MTD \cdot O^{\perp} \quad \textbf{(8)}$$

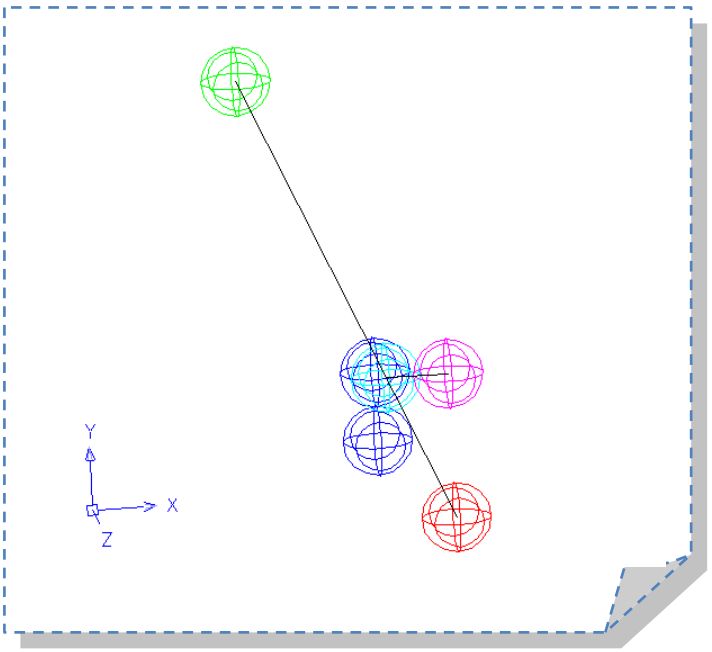

Figura 18: Cálculo del punto intermedio

En la figura anterior las esferas representan el volumen que ocupa cada uno de los objetos del mundo. La esfera roja envuelve por completo al robot y su centro será nuestro punto origen. La esfera verde es el destino. Las dos esferas azul oscuro representan una biesfera que engloba al obstáculo a evitar. Mientras que la esfera azul claro es el punto de máxima penetración calculado. En rosa tenemos el nuevo punto intermedio que asegura la no colisión, es decir, el nuevo destino que deberá seguir el robot.

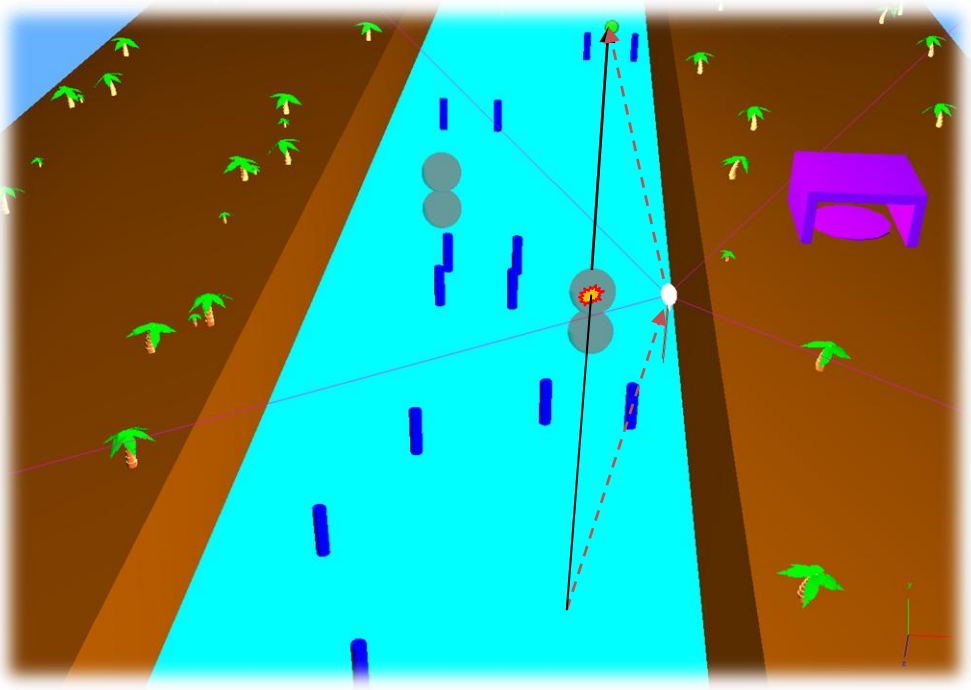

Figura 19: Screenshot de la ejecución del robot evitando un obstáculo

Ya que la gestión de los destinos del globo dirigible se hace mediante una pila, cuando para evitar un obstáculo se calcula un nuevo destino lo único que hace falta es añadir este punto intermedio al tope de la pila. Así a la siguiente iteración el destino habrá cambiado y se empezará a evitar el obstáculo. Una vez alcanzado este nuevo destino, para seguir el circuito se procederá igual que con el resto de destinos, bastará con eliminarlo de la pila. El cálculo del punto intermedio de no colisión puede ocasionarnos 2 problemas importantes.

Por un lado se encuentra la cuestión de que desplazar el punto de máxima penetración de manera que quede en la superficie de la biesfera puede no ser suficiente para evitar la colisión si tenemos en cuenta la inercia que puede llevar el robot. Por esto, pueden aparecer más puntos intermedios de los deseados a lo largo de las distintas iteraciones que realiza el controlador. La manera de solucionar esto pasa por dos operaciones. Podemos añadir un porcentaje de confianza a la distancia que separamos el punto intermedio. Y también insertaremos el punto intermedio en la pila únicamente si se encuentra a una distancia lógica del anterior punto intermedio insertado en la pila. De esta manera se minimizarán los casos en los que aparecen demasiados puntos intermedios para evitar un obstáculo.

El segundo problema que se nos presenta es que si el obstáculo se encuentra muy próximo a la superficie, es posible que dependiendo de dónde se encuentren el globo dirigible y el destino, la mejor forma de evitar el obstáculo sea por debajo del mismo. En otro caso esta maniobra estaría permitida, pero tratándose de que el obstáculo se encuentre próximo a la superficie del agua no debe ser permitida este tipo de maniobra. Pues el globo no debería sumergirse ni chocar en ningún momento. Para evitarlo se añade un límite inferior a la altitud, y en caso de que el punto intermedio se halle por debajo, se calcula el punto opuesto para que el robot pase por encima del límite establecido.

Obviamente todo este procedimiento se realizará cada iteración sobre los datos recibidos de la parte de detección de obstáculos. Y de esta manera, eligiendo primero qué obstáculo es el primero a evitar, se calcula la posición que se debe alcanzar para no colisionar con él. Una vez alcanzada esta posición intermedia, el robot recupera la trayectoria hacia el destino original gracias a la gestión de los destinos en forma de pila.

4- Manual de usuario

4.1- Requisitos

Para la realización de este proyecto se ha utilizado un ordenador de sobremesa AMD Athlon 64 processor 3400+ a 2,4 GHz y con 512 MB de memoria RAM. EL sistema operativo ha sido Microsoft Windows XP Profesional v. 2002 con el ServicePack 3. Para ejecutar la simulación es indispensable el software de Cyberbotics Webots 5.1.0. En su defecto, será válida una versión compatible.

4.2- Modificaciones del entorno

Todas las modificaciones que tengan que ver con elementos del entorno, van a ser realizadas sobre el apartado visual de Webots. Sin embargo, aquellas que afecten a la tarea del robot se harán sobre el código del controlador. Esto quiere decir que si por ejemplo queremos añadir un elemento decorativo no tendremos que tocar para nada el código del controlador. Sin embargo, si el elemento que queremos añadir es un nuevo obstáculo, deberemos añadirlo tanto al entorno visual como al código del controlador. En este apartado se explicará el caso de modificar el entorno visual sin afectar al código del controlador.

Las ventanas de Webots que vamos a necesitar son la ventana principal del programa donde veremos el resultado y la ventana que contiene el Scene Tree donde cambiaremos las características del mundo VRML.

4.2.1- Añadir un elemento

Ya que el mundo virtual está creado utilizando VRML, podríamos editar el código en cualquier editor de texto siempre y cuando respetáramos la estructura de este lenguaje. Pero como en realidad es una adaptación de este lenguaje lo recomendable es utilizar el editor de Webots. Este editor nos dará las opciones aceptadas en cada caso y lo único que deberemos hacer es ir completando los campos.

En la ventana tenemos un botón que nos permite crear nodos nuevos eligiéndolos de una lista de nodos permitidos.

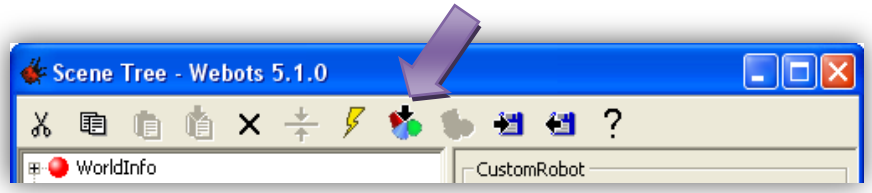

Figura 20: Botón Insert after

Con este botón vamos a crear los nodos principales y cada uno que creemos nos aparecerá en el Scene Tree en el lugar que nosotros hayamos elegido. Si pinchamos sobre el nuevo nodo, podremos darle un nombre a ese mismo nodo, lo que en VRML es un DEF. A la izquierda del nodo tenemos el símbolo + o – que nos permitirá expandir o contraer los campos. Si expandimos el nodo, podremos pinchar sobre cada uno de los campos que lo componen y en la parte derecha de la ventana completar los valores de los que deseemos.

Acompañados de un círculo rojo en lugar de azul veremos que van todos aquellos nodos susceptibles de contener otros nodos (a excepción del atributo CHILDREN, que aparece en azul pero puede contener otros nodos). En cambio en azul sólo aparecerán los atributos de los nodos que contienen valores.

Pongámonos en el caso de que queramos colocar un nuevo elemento decorativo en el mundo virtual. Por ejemplo una esfera que represente el Sol. El nodo que vamos a necesitar va a ser el nodo SOLID, así podremos aplicarle propiedades como la física. Entonces haremos clic sobre el botón “Insert after” y elegiremos el nodo SOLID.

Aparecerá debajo del nodo que tuviéramos seleccionado en ese momento. Y aparecerá comprimido, así que haremos clic sobre el + para expandir sus atributos.

Lo primero que haremos será definir el nodo con el nombre SOL y para eso lo escribiremos en la parte derecha de la ventana cuando lo tengamos seleccionado. Ahora le daremos una forma, así que iremos al atributo CHILDREN y haremos clic nuevamente en el botón "Insert after". Elegiremos el nodo SHAPE y aparecerá dentro de CHILDREN. El programa tiene un error de refresco y deberemos seleccionar un nodo diferente para ver los cambios. El nodo SHAPE tiene dos campos: APPEARENCE y GEOMETRY. El primero lo rellenaremos con un nodo APPEARENCE. El segundo con uno SPHERE. Este último acepta dos parámetros; el radio y el número de divisiones. Los rellenaremos con 1 y 20. Por el otro lado de los parámetros del nodo APPEARENCE sólo rellenaremos el MATERIAL con un nodo MATERIAL. Los atributos diffuseColor, emissiveColor y specularColor serán 1 1 0 para de esta forma tener una esfera amarilla.

Todos los elementos se crean por defecto en la posición (0, 0, 0). Por eso vamos a trasladar el objeto entero a un lugar mejor. Pincharemos el atributo TRANSLATION del sólido que estamos creando y pondremos sus valores a -2, 6, -8.

4.2.2- Modificar y eliminar un elemento

Como hemos visto los nodos contienen atributos. Y al igual que cuando creamos un nodo nuevo debemos ir completando esos atributos con valores, cuando los nodos ya están creados la manera de proceder es la misma. Cualquier valor que queramos modificar, lo haremos desplegando los atributos del nodo y pinchando sobre el atributo a cambiar. En la parte derecha de la ventana podremos poner el nuevo valor e inmediatamente aparecerá en la ventana principal el cambio realizado.

Webots permite transformar unos nodos en otros con el botón TRANSFORM, que está situado a la izquierda del botón "Insert after". Este botón lo que nos va a permitir es aprovechar los atributos en común entre el nodo existente y el nodo en que queremos que éste se transforme. Nos ahorrará tiempo en caso de que necesitemos realizar esta operación.

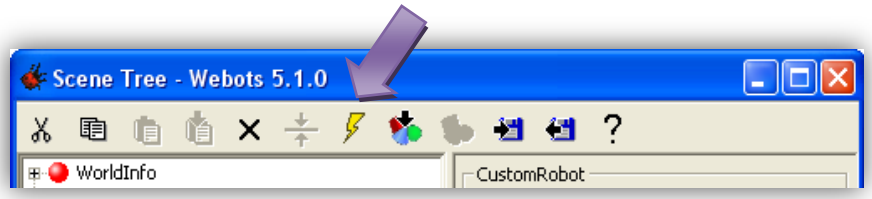

Figura 21: BotónTransform

También podemos realizar las típicas operaciones de Copiar, Cortar y Pegar. Pero lo interesante es que se nos permite hacerlo sobre los nodos enteros, incluidos los atributos.

Figura 22: Botones de cortar, copiar y pegar

Por último, si lo que queremos es eliminar un nodo completo sólo deberemos seleccionarlo y apretar el botón que contiene un aspa negra. Automáticamente desaparecerá del árbol y de la escena.

Figura 23: Botón eliminar nodo

4.3- Modificar el controlador

Necesitaremos tener en el directorio de trabajo las carpetas world y controllers. Dentro de la primera deberemos colocar el archivo "mundo_XVI.wbt". En la segunda, será necesario crear una carpeta con el nombre "c_mundo_XVI" que contenga los ficheros "Makefile", "pila.cpp", "punto.cpp" y "c_mundo_XVI.cpp". Aunque este paso es recomendable realizarlo desde el propio Webots con la opción de *Wizard > New robot controller*, ya que de esta forma es más sencillo y evita problemas de mala estructuración. Con esta estructura de archivos estaremos en condiciones de ejecutar Webots y compilar el código desde el editor de código. Una vez compilado nos aparecerán en esta última carpeta los archivos "c_mundo_XVI.o" y "c_mundo_XVI.exe". Ahora ya podremos lanzar la ejecución de la simulación desde la ventana principal de Webots.

Obviamente, cada vez que se realice un cambio en el código fuente del controlador, habrá que volver a compilar el código. Lo podemos hacer como se acaba de explicar desde la ventana del editor de código. Una vez compilado el código con los cambios, podremos lanzar a ejecución el nuevo programa tal y como se ha comentado.

4.3.1- Cambiar de controlador

En un principio para utilizar este proyecto no es necesaria esta operación, pues el mundo ya viene con el controlador necesario asociado al robot. Pero en cualquier caso, se nos puede presentar la necesidad de cambiar el controlador del robot. Los pasos a seguir son los siguientes. En primer lugar es necesario crear el controlador, como ya se ha explicado anteriormente en esta memoria. Una vez el controlador ya existe, sólo hay que expandir los atributos del robot en el Scene Tree y buscar "controller". En la parte derecha de la ventana tenemos un botón con tres puntos suspensivos que nos muestra una ventana donde elegir uno de los controladores permitidos por Webots. Si todo es correcto aparecerá el que buscamos y sólo con seleccionarlo quedará asociado al robot.

4.3.2- Añadir un elemento

En ocasiones no sólo vamos a modificar aspectos visuales del entorno, si no que nos va a interesar que el elemento que introduzcamos sea un elemento con el que el robot interactúe. Vamos a describir la forma de añadir los objetos más importantes. Uno de ellos son los destinos del robot. Para que el robot realice los cálculos necesarios para avanzar de un destino a otro, éstos deben ser almacenados. Para poder gestionarlos más fácilmente, los destinos que debe seguir el zepelín se almacenan en una pila. Para añadir un nuevo destino tenemos que añadir un nuevo elemento a la pila en el lugar que deseemos con la posición del destino a seguir por el robot:

```
stack.push(Punto(pos_X, pos_Y, pos_Z));
```

Es muy importante dejar los 3 últimos puntos de la pila sin modificar, pues son los establecidos para la maniobra de docking.

Otro tipo de objeto que nos puede interesar añadir son los obstáculos. Los obstáculos también quedan almacenados en memoria y son almacenados en forma de vector, pues no importa el orden de almacenamiento. Si queremos añadir uno podemos añadir la

posición de su centro al vector de obstáculos teniendo en cuenta que deberemos incrementar la constante "NUM_OBS" en una unidad.

```
#define NUM_OBS 2
float Obstaculos[NUM_OBS][3]= {
                                        0.95, 2.0, 2.0,
                                        -0.6, 2.2, 0.0
                                };
```

4.3.3- Modificar y eliminar un elemento

Al igual que podemos querer añadir elementos que no sólo afectan al aspecto visual, si no que también afectan a la parte del controlador, también nos puede interesar por cualquier motivo el modificar los elementos ya existentes o incluso eliminarlos por completo.

La forma de proceder es muy similar a la del apartado anterior. Se trata de localizar en el código el objeto a modificar, y cambiar directamente sobre el código los valores que se desee. Por ejemplo para cambiar de posición el primer destino del robot, lo que haríamos sería ir a la pila donde están todos almacenados y cambiar los valores de pos_X, pos_Y y pos_Z al último elemento añadido a la pila. Así la próxima vez que compilemos el código, los nuevos valores ya serán los que siga el zepelín.

En caso de querer modificar las posiciones de los obstáculos no debemos cambiar el valor de la constante NUM_OBS como hacíamos antes. Únicamente cambiaremos los valores del obstáculo que queremos mover. Para cambiar el radio de las esferas que envuelven a los objetos del cálculo de colisiones lo haremos directamente sobre el campo de la estructura de la esfera. Este es uno de los motivos por los que se planteará más adelante una propuesta de ampliación de este proyecto.

Visto cómo modificar los elementos, falta saber la manera de eliminarlos. Gracias al método escogido para almacenar dichos elementos su eliminación va a ser tan sencilla como suprimir del código las posiciones del elemento. Es decir, en el caso de querer eliminar un destino, lo localizaremos en la pila y suprimiremos la línea de código que lo inserta en la pila. De esta forma cuando se vuelva a compilar el código, ese destino no existirá para el robot. En el caso de los obstáculos es muy similar, pues están almacenados en un vector y si suprimimos la línea de código que lo almacena sucederá como en el caso anterior.

4.4- Manejo manual del robot

Como ya se ha comentado en alguna ocasión a lo largo de esta memoria, al robot volador se le han implementado 2 modos de vuelo. El primero y más importante es el modo automático. Este modo como ya se ha visto es capaz de hacer que el globo dirigible siga un circuito marcado por diferentes destino y además evitando los posibles obstáculos que pueda haber en su trayectoria sin perder esta misma. Sin embargo el modo manual consiste en la posibilidad de desactivar este "piloto automático" y manejar con el teclado todos los motores del globo.

La forma de activar y desactivar el modo manual es pulsando la tecla "P". En cualquier momento de la ejecución de la simulación se puede pulsar esta tecla y cambiará de modo. Por defecto comienza el modo automático, por lo tanto pulsar la tecla "P" significa cambiar al modo manual y volver a pulsarla activaría de nuevo el modo automático.

Para manejar los 3 motores del robot se utilizará el teclado como ya se ha dicho. Concretamente activaremos el motor 1 con las teclas "↑ / ↓". La primera hará que el globo avance y la segunda lo hará retroceder. El motor 2 lo controlamos con las flechas "A / Z", que harán que el robot suba y baje respectivamente. Finalmente controlamos el giro del robot con el motor 3 con la ayuda de las teclas "← / →", que giran la cola del robot hacia la derecha y hacia la izquierda. Se ha tenido en cuenta la visión del usuario y pulsar la tecla "←" hace que el robot se mueva hacia la izquierda, lo que significa que la cola del globo se mueve hacia la derecha.

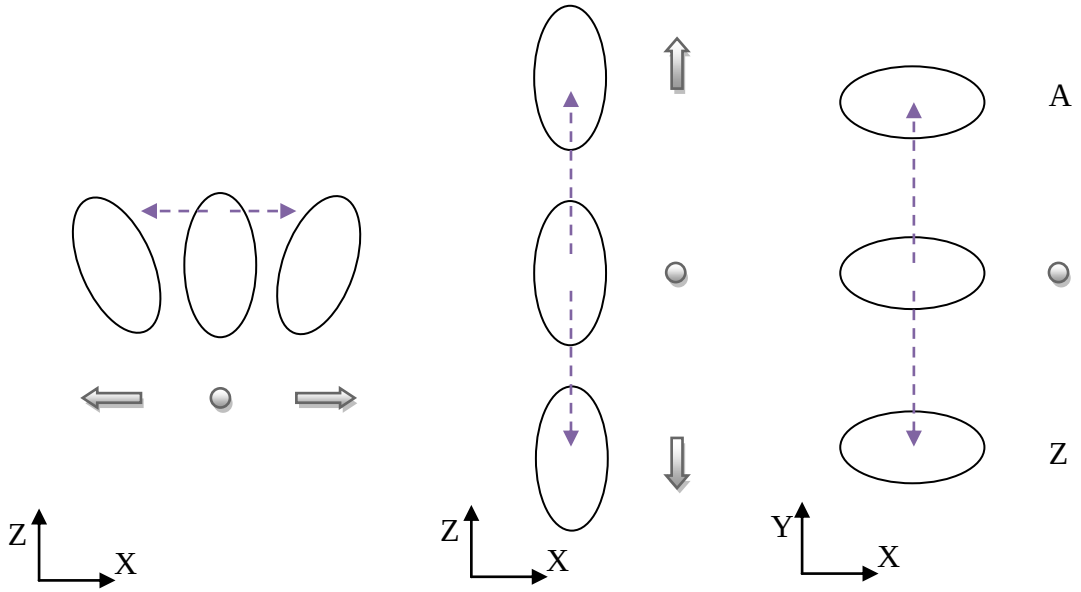

Figura 24: Resumen de los movimientos que realiza cada tecla

5- Conclusiones

El objetivo principal del proyecto era conseguir crear la simulación de un robot volador autónomo. Como el programa a utilizar era Webots y el proyectando no tenía experiencia con ese software, lo primero fue familiarizarse con él. Como apoyo se tienen 2 documentos: la guía de referencia y el manual de usuario. El programa también tiene varios ejemplos que se fueron consultando y realizando progresivamente añadiendo características a los robots creados.

Finalmente cuando ya se tenía cierta soltura con el programa se comenzó a crear el mundo virtual. En este caso sí que tenía experiencia con VRML y no supuso una gran dificultad modelar los objetos como era necesario. Por lo tanto era cuestión de aprender el funcionamiento de los nuevos nodos que incorpora Webots para los robots.

Con el mundo creado se añadió el modelo de ejemplo “blimp_asl2”, un globo dirigible de la EPFL. Y se creó un nuevo controlador incluyendo partes útiles del controlador original. En este nuevo controlador es donde se comenzó a trabajar con la parte de programación de este proyecto final de carrera.

Lo que se pretendía era en una primera parte controlar el robot de forma manual con el teclado. Para ello se estudió el tipo de controlador más adecuado a cada una de las circunstancias y motores del robot. Más adelante cuando el robot era manejable se incorporó el seguimiento de destinos prefijados. Es decir, el robot viajaba de forma correcta y automáticamente por un circuito marcado. Finalmente se añadió una parte muy importante de código encargada de detectar y evitar obstáculos mientras se seguía el circuito marcado. Para la realización de esta parte ha sido esencial el estudio de la tesis doctoral de Enrique Bernabeu Soler, director de este proyecto.

5.1- Posibles ampliaciones futuras

Como en todo proyecto existen una serie de cuestiones dignas de ser estudiadas para una posible ampliación. A continuación se hablará de las que quizá deberían considerarse prioritarias, ya sea por la mejora que suponen al proyecto o bien por su utilidad posterior.

Existen datos necesarios para la ejecución de la simulación que quedan almacenados en memoria para ser utilizados por los cálculos internos. Estos datos como pueden ser los destinos que debe seguir el robot o los obstáculos que aparecen en el entorno se almacenan como parte del código. Esto hace que para realizar pequeños cambios en las posiciones de estos elementos sea necesario volver a compilar el controlador. Debido a las limitaciones de Webots y del proyecto en sí, no parece viable crear una interfaz gráfica que permita al usuario la introducción de estos datos, pero sí que sería interesante modificar el código fuente para realizar la lectura de estos datos por fichero.

Otro aspecto que sería interesante modificar es el tamaño de las esferas envolventes para la detección y evitación de los obstáculos. Este radio ha sido el mismo durante todo el proyecto y por eso no se ha tenido en cuenta el poder modificarlo independientemente según el objeto al que pertenezca. Una posible e interesante ampliación sería añadir al vector de obstáculos además de la posición, el radio. Y más tarde en el algoritmo que hace uso de él, utilizar este nuevo valor almacenado en lugar del fijo que se utiliza actualmente.

También hay un aspecto menos importante que es el del paso de control automático a manual y viceversa. En estos momento lo que se hace es controlar el teclado y ver si la tecla “P” es pulsada. En este caso se cambia de modo con un flag de control. El inconveniente es que debido al tiempo de muestreo que es de 32 ms, puede suceder que en una pulsación pase un número par de iteraciones. En ese caso no se cambiaría de modo por la forma en que se ha programado el cambio.

Por último también se podría estudiar el caso de obstáculos en puntos críticos de la trayectoria del robot. Estos pueden ser por ejemplo obstáculos muy cercanos al destino o incluso muy cercanos entre sí. Habría que estudiar en qué medida afectaría a la trayectoria del robot y cómo se podría solucionar.

Tabla de ilustraciones

Bibliografía

[Ber98] Bernabeu Soler, Enrique Jorge (1998). *Planificación de movimientos libres de colisión en sistemas robotizados mediante la aplicación de la transformada de Hough.* (Tesis doctoral – Universidad Politécnica de Valencia).

[ZGeta06] Zufferey, J.-C., Guanella, A., Beyeler, A. and Floreano, D. (2006) *Flying over the Reality Gap: From Simulated to Real Indoor Airships.* [pdf] Autonomous Robots, 21(3) pp. 243-254.
<http://lis.epfl.ch/index.html?content=research/projects/BioinspiredFlyingRobots/>
Consulta: [06 noviembre 2009]

User Guide. [pdf] copyright © (2005) Cyberbotics Ltd. All rights reserved.
<www.cyberbotics.com>

Reference Manual. [pdf] copyright © (2005) Cyberbotics Ltd. All rights reserved.
<www.cyberbotics.com>

Apéndice A: Contenido del CD

1. Memoria.docx - *este fichero*
2. Memoria.pdf - *este fichero*
3. Memoria.doc - *este fichero*
4. Memòria valencià.docx - *este fichero valenciano*
5. Memòria valencià.pdf - *este fichero valenciano*
6. Memòria valencià.doc - *este fichero valenciano*
7. documentos - *archivos de consulta imprescindible*
    a. guide.pdf
    b. reference.pdf
    c. Zufferey_Guanella_Beyeler_Floreano_AURO_2006.pdf
8. versiones antiguas - *versiones de aprendizaje*
    a. worlds
    b. controllers
9. version final - *versión final estable*
    a. worlds
    b. controllers
    c. version FINAL.avi
10. archivos - *archivos utilizados de gran ayuda*
    a. calculos manuales.nb
    b. evitando obstaculo.dwg
    c. físicas.cc
11. Licencia GPLv3.txt - *licencia del proyecto*

Apéndice B: Función de detección de colisiones

```
void detecta_colision(Esfera ori, Esfera dest, Biesfera* obs){
  Punto intermedio;
  float mtdcalculado=0.0, lambda=0.0, lambda1=0.0, lambda2=0.0;
  int mtd=0;

  /*******************************/
  /*  Diferencias de MINKOSWSKI  */
  /*******************************/
  Esfera m0,m1,m2,m3;

  m0.c = ori.c - (obs->c0);
  m1.c = ori.c - (obs->c1);
  m2.c = dest.c - (obs->c0);
  m3.c = dest.c - (obs->c1);

  m0.r = ori.r + (obs->r0);
  m1.r = ori.r + (obs->r1);
  m2.r = dest.r + (obs->r0);
  m3.r = dest.r + (obs->r1);

  /*******************************/
  /*      calculo de LAMBDAS     */
  /*******************************/
  float l1,l2,l;
  float a, a1, a2;
  a1= (m2.c - m0.c)*(m0.c);
  a2= (m3.c - m0.c)*(m2.c - m0.c);
  a= a1 * a2;

  /*************************************/
  float b, b1, b2;
  b1= (m3.c - m0.c)*(m0.c);
  b2= modulo_sin_raiz( m0.c,m2.c );
  b= b1 * b2;

  /*************************************/
  float c,c1,c2;
  c1= modulo_sin_raiz( m0.c,m3.c );
  c2= modulo_sin_raiz( m0.c,m2.c );
  c= c1*c2;

  /*************************************/
  float d, d1;
  d1= (m3.c - m0.c)*(m2.c - m0.c);
  d= d1 * d1;

  /*********************/
  /*        l1         */
    l1 =  (a-b) / (c-d);
  /*********************/

  /*************************************/
  //   cambios para l2
  /*************************************/
  a1= (m3.c - m0.c)*(m0.c);
  a2= (m3.c - m0.c)*(m2.c - m0.c);
  a= a1 * a2;

  /*************************************/
  b1= (m2.c - m0.c)*(m0.c);
  b2= modulo_sin_raiz( m0.c,m3.c );
  b= b1 * b2;

  /*********************/
  /*        l2         */
    l2 =  (a-b) / (c-d);
  /*********************/

  /*********************/
  /*        l          */
    l = l1 + l2;
  /*********************/
```

```
// conociendo las lambdas, determinamos que mtd debemos aplicar
if(l<=1){
  if(l2>=0){
    if(l1>=0){
      mtd=1;
    }else{
      mtd=2;
    }
  }else{
    if(l1>=0){
      mtd=3;
    }else{
      mtd=4;
    }
  }
} else{
  if(l2>=0){
    if(l1>=0){
      mtd=5;
    }else{
      mtd=6;
    }
  }else{
    mtd=7;
  }
}

// en caso de que afecte a la otra triesfera
float l1bi=0.0, l2bi=0.0, lbi=0.0;

if(mtd==3 || mtd==7){
  // cambio de lambdas
  l1bi=-l2;
  l2bi=l1+l2;
  lbi=l1bi+l2bi;

  if(lbi<=1){
  if(l2bi>=0){
    if(l1bi>=0){
      mtd=8;
    }else{
      mtd=0;
    }
  }else{
    if(l1bi>=0){
      mtd=9;
    }else{
      mtd=0;
    }
  }
} else{
  if(l2bi>=0){
    if(l1bi>=0){
      mtd=10;
    }else{
      mtd=0;
    }
  }else{
    mtd=11;
  }
}
}
```

```
// segun el mtd aplicaremos unos calculos u otros
Triesfera tri;
Biesfera bi;
Punto cero, proyección, proyeccion1, proyeccion2;
float d0, mtd1=0.0, mtd2=0.0, radio1, radio2;
int tipo=0;       //0->nada, 1->esfera, 2->biesfera, 3->triesfera

switch(mtd){

  case 0:
    robot_console_printf("\n\n\t¡¡¡¡ERROR EN LOS CALCULOS!!!!\t\n\n");
    break;
  case 1:
    tipo=3;
    //S032
    tri.r0=m0.r;
    tri.r1=m3.r;
    tri.r2=m2.r;
    tri.c0=m0.c;
    tri.c1=m3.c;
    tri.c2=m2.c;
    proyeccion= (tri.c0 + (((tri.c1-tri.c0)^l1) + ((tri.c2-tri.c0)^l2)));
    d0=modulo_vector(cero,proyeccion);
    mtdcalculado=d0-(tri.r0+((tri.r1-tri.r0)*l1)+((tri.r2-tri.r0)*l2));
    obs->radio=(tri.r0+((tri.r1-tri.r0)*l1)+((tri.r2-tri.r0)*l2));
    obs->lambda=l;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 2:
    tipo=2;
    //S02
    bi.r0=m0.r;
    bi.r1=m2.r;
    bi.c0=m0.c;
    bi.c1=m2.c;
    lambda=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion= (bi.c0 + ((bi.c1-bi.c0)^lambda));
    if(lambda<0.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda>1.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c1)-bi.r1;
      obs->radio=bi.r1;
    }else{
      mtdcalculado=modulo_vector(cero,proyeccion) - (bi.r0 + ((bi.r1-bi.r0)*lambda));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda));
    }
    obs->lambda=lambda;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 3:
    break;
  case 4:
    tipo=2;
    //S02
    bi.r0=m0.r;
    bi.r1=m2.r;
    bi.c0=m0.c;
    bi.c1=m2.c;
    lambda2=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion2= (bi.c0 + ((bi.c1-bi.c0)^lambda2));
    if(lambda2<0.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c0)-bi.r0;
      radio2=bi.r0;
    }else if(lambda2>1.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c1)-bi.r1;
      radio2=bi.r1;
    }else{
      mtd2=modulo_vector(cero,proyeccion2) - (bi.r0 + ((bi.r1-bi.r0)*lambda2));
```

```
      radio2=(bi.r0 + ((bi.r1-bi.r0)*lambda2));
    }
    //S01 --> S10
    bi.r0=m1.r;
    bi.r1=m0.r;
    bi.c0=m1.c;
    bi.c1=m0.c;
    lambda1=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion1= (bi.c0 + ((bi.c1-bi.c0)^lambda1));
    if(lambda1<0.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c0)-bi.r0;
      radio1=bi.r0;
    }else if(lambda1>1.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c1)-bi.r1;
      radio1=bi.r1;
    }else{
      mtd1=modulo_vector(cero,proyeccion1) - (bi.r0 + ((bi.r1-bi.r0)*lambda1));
      radio1=(bi.r0 + ((bi.r1-bi.r0)*lambda1));
    }
    if(mtd1<=mtd2){
      mtdcalculado=mtd1;
      obs->lambda=lambda1;
      obs->mtd=mtdcalculado;
      obs->proy=proyeccion1;
      obs->radio=radio1;
    }else{
      mtdcalculado=mtd2;
      obs->lambda=lambda2;
      obs->mtd=mtdcalculado;
      obs->proy=proyeccion2;
      obs->radio=radio2;
    }
  case 5:
    tipo=2;
    //S32 --> S23
    bi.r0=m2.r;
    bi.r1=m3.r;
    bi.c0=m2.c;
    bi.c1=m3.c;
    lambda=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion= (bi.c0 + ((bi.c1-bi.c0)^lambda));
    if(lambda<0.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda>1.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c1)-bi.r1;
      obs->radio=bi.r1;
    }else{
      mtdcalculado=modulo_vector(cero,proyeccion) - (bi.r0 + ((bi.r1-bi.r0)*lambda));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda));
    }
    obs->lambda=lambda;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 6:
    tipo=2;
    //S02
    bi.r0=m0.r;
    bi.r1=m2.r;
    bi.c0=m0.c;
    bi.c1=m2.c;
    lambda2=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion2= (bi.c0 + ((bi.c1-bi.c0)^lambda2));
    if(lambda2<0.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda2>1.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c1)-bi.r1;
```

```
      obs->radio=bi.r1;
    }else{
      mtd2=modulo_vector(cero,proyeccion2) - (bi.r0 + ((bi.r1-bi.r0)*lambda2));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda2));
    }
    //S32 --> S23
    bi.r0=m2.r;
    bi.r1=m3.r;
    bi.c0=m2.c;
    bi.c1=m3.c;
    lambda1=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion1= (bi.c0 + ((bi.c1-bi.c0)^lambda1));
    if(lambda<0.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda1>1.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c1)-bi.r1;
      obs->radio=bi.r1;
    }else{
      mtd1=modulo_vector(cero,proyeccion1) - (bi.r0 + ((bi.r1-bi.r0)*lambda1));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda1));
    }
    if(mtd1<=mtd2){
      mtdcalculado=mtd1;
      obs->lambda=lambda1;
      obs->mtd=mtdcalculado;
      obs->proy=proyeccion1;
      obs->radio=radio1;
    }else{
      mtdcalculado=mtd2;
      obs->lambda=lambda2;
      obs->mtd=mtdcalculado;
      obs->proy=proyeccion2;
      obs->radio=radio2;
    }
    break;
  case 7:
    break;
  case 8:
    tipo=3;
    //S013
    tri.r0=m0.r;
    tri.r1=m1.r;
    tri.r2=m3.r;
    tri.c0=m0.c;
    tri.c1=m1.c;
    tri.c2=m3.c;
    proyeccion= (tri.c0 + (((tri.c1-tri.c0)^l1bi) + ((tri.c2-tri.c0)^l2bi)));
    d0=modulo_vector(cero,proyeccion);
    mtdcalculado=d0-(tri.r0+((tri.r1-tri.r0)*l1bi)+((tri.r2-tri.r0)*l2bi));
    obs->radio=(tri.r0+((tri.r1-tri.r0)*l1bi)+((tri.r2-tri.r0)*l2bi));
    obs->lambda=lbi;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 9:
    tipo=2;
    //S01 --> S10
    bi.r0=m1.r;
    bi.r1=m0.r;
    bi.c0=m1.c;
    bi.c1=m0.c;
    lambda=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion= (bi.c0 + ((bi.c1-bi.c0)^lambda));
    if(lambda<0.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda>1.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c1)-bi.r1;
      obs->radio=bi.r1;
    }else{
```

```
      mtdcalculado=(modulo_vector(cero,proyeccion)) - (bi.r0 + ((bi.r1-bi.r0)*lambda));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda));
    }
    obs->lambda=lambda;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 10:
    tipo=2;
    //S13
    bi.r0=m1.r;
    bi.r1=m3.r;
    bi.c0=m1.c;
    bi.c1=m3.c;
    lambda=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion= (bi.c0 + ((bi.c1-bi.c0)^lambda));
    if(lambda<0.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c0)-bi.r0;
      obs->radio=bi.r0;
    }else if(lambda>1.0){
      tipo=1;
      mtdcalculado=modulo_vector(cero,bi.c1)-bi.r1;
      obs->radio=bi.r1;
    }else{
      mtdcalculado=modulo_vector(cero,proyeccion) - (bi.r0 + ((bi.r1-bi.r0)*lambda));
      obs->radio=(bi.r0 + ((bi.r1-bi.r0)*lambda));
    }
    obs->lambda=lambda;
    obs->mtd=mtdcalculado;
    obs->proy=proyeccion;
    break;
  case 11:
    tipo=2;
    //S01 --> S10
    bi.r0=m1.r;
    bi.r1=m0.r;
    bi.c0=m1.c;
    bi.c1=m0.c;
    lambda2=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion2= (bi.c0 + ((bi.c1-bi.c0)^lambda2));
    if(lambda2<0.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c0)-bi.r0;
      radio2=bi.r0;
    }else if(lambda2>1.0){
      tipo=1;
      mtd2=modulo_vector(cero,bi.c1)-bi.r1;
      radio2=bi.r1;
    }else{
      mtd2=modulo_vector(cero,proyeccion2) - (bi.r0 + ((bi.r1-bi.r0)*lambda2));
      radio2=(bi.r0 + ((bi.r1-bi.r0)*lambda2));
    }
    //S13
    bi.r0=m1.r;
    bi.r1=m3.r;
    bi.c0=m1.c;
    bi.c1=m3.c;
    lambda1=-( (bi.c0 * (bi.c1-bi.c0)) / (modulo_sin_raiz( bi.c0,bi.c1 )) );
    proyeccion1= (bi.c0 + ((bi.c1-bi.c0)^lambda1));
    if(lambda1<0.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c0)-bi.r0;
      radio1=bi.r0;
    }else if(lambda1>1.0){
      tipo=1;
      mtd1=modulo_vector(cero,bi.c1)-bi.r1;
      radio1=bi.r1;
    }else{
      mtd1=modulo_vector(cero,proyeccion1) - (bi.r0 + ((bi.r1-bi.r0)*lambda1));
      radio1=(bi.r0 + ((bi.r1-bi.r0)*lambda1));
    }
    if(mtd1<=mtd2){
      mtdcalculado=mtd1;
      obs->lambda=lambda1;
```

```
          obs->mtd=mtdcalculado;
          obs->proy=proyeccion1;
          obs->radio=radio1;
        }else{
          mtdcalculado=mtd2;
          obs->lambda=lambda2;
          obs->mtd=mtdcalculado;
          obs->proy=proyeccion2;
          obs->radio=radio2;
        }
        break;
      default:
        break;
    }
}
```

Apéndice C: Código de evitación de colisiones

```
   // buscamos el mas cercano que colisionara (lambda mas pequeña con mtd negativo)
         minlambda=10000;
         for(int k=0; k<NUM_OBS; k++){
           if(
               obs[k].mtd < 0.0              // habra colision
               && obs[k].lambda < minlambda  // es la mas cercana
               && obs[k].lambda>0.0          // esta por delante del robot
               && obs[k].lambda<1.0          // esta antes del destino (podemos dejar margen)
             ){
             minlambda=obs[k].lambda;
             ilambda=k;
           }
         }

         // SI DISTANCIA <= 0 HAY COLISION
         if(obs[ilambda].mtd < 0.0){
           // determinar posicion de maxima penetracion
           Punto maxpenetracion;
           maxpenetracion= origen.c + ((destino.c - origen.c)^obs[ilambda].lambda);
           p_intermedio = maxpenetracion -
((normalizar(cero,obs[ilambda].proy))^((obs[ilambda].mtd)*1.05));

           // para evitar un globo submarino
           if(p_intermedio.y < 1.8){
             robot_console_printf("Evitando inmersion!!");
             float mtd_nueva = 0.0;
             mtd_nueva = obs[ilambda].mtd + (obs[ilambda].radio*2.0);
             if(mtd_nueva > 0)
               mtd_nueva = -mtd_nueva;
             p_intermedio = maxpenetracion -
((normalizar(obs[ilambda].proy,cero))^((mtd_nueva)*1.05));
           }

           // control de altitud
           if(p_intermedio.y < 1.6){
             p_intermedio.y = 1.6;
           }
           if(p_intermedio.y > 4.0){
             p_intermedio.y = 4.0;
           }

           // gestionar que no haya demasiados puntos intermedios
           if(stack.elementos()>2 && fabs(stack.top().z-p_intermedio.z)<1.5){
             stack.pop();
           }

           stack.push(p_intermedio);

           robot_console_printf("cuidadiTOOOOOOORRRR!!\n");

         }  else{

           //robot_console_printf("\n");

         }
```