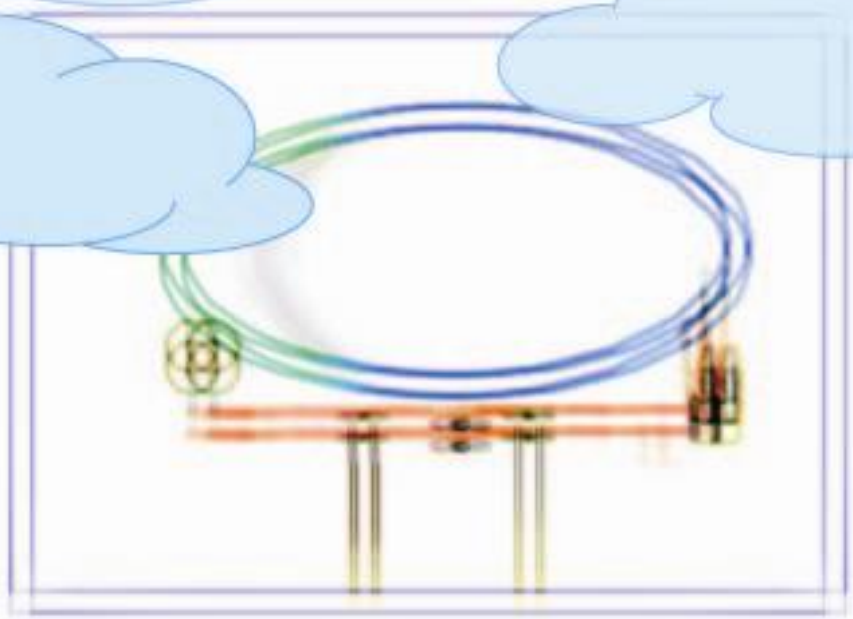

PFC - Simulació d'un robot volador autònom mitjançant la utilització de l'eina Webots

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Hector Díaz Bachiller

Dirigit per:
Enrique Jorge Bernabeu Soler

Curs 2009 - 2010

A tots els quals van desitjar veure aquest projecte finalitzat.

Gràcies.

Índex

1- Introducció

1.1- Descripció dels objectius del projecte

La finalitat d'aquest projecte final de carrera és aconseguir simular el comportament d'un robot volador utilitzant el programari comercial de Cyberbotics, Webots. Concretament la seua versió 5.1.0.

Com robot volador es va escollir una espècie de globus dirigible, llevat que no porta timó, de grandària reduïda. Com veurem més endavant, tots els moviments del robot es controlen mitjançant hèlixs. Aquest zeppelin és un robot construït en la École Polytechnique Fédérale de Lausanne (EPFL). Existeix un model virtual sota llicència GPL2 o superior per a Webots que inclou components físiques i que és el punt de partida d'aquest projecte.

Tenint com base aquest model virtual, que té en compte factors necessaris com el fregament o la inèrcia, anem a construir un entorn pel qual el globus dirigible haurà de volar de forma autònoma. El vol es realitzarà seguint un circuit fixat per endavant, i acabarà en un espai destinat per al repòs del robot. Aconseguir que el robot es desplace pels punts desitjats permetrà un major control i per tant, les aplicacions futures seran molt interessants (vigilància forestal, fotografia aèria, transport aeri automàtic, etc.).

Al llarg de tot el recorregut que el robot ha de realitzar en l'escenari creat per a la simulació, també s'afegiran obstacles. Una part molt important d'aquest projecte va a ser que el globus haurà d'evitar la col·lisió amb qualsevol tipus d'obstacle fix. Per a així aconseguir realment un robot volador autònom, que no precise de cap tipus d'interacció externa per a realitzar la seua comesa. La posició dels obstacles serà coneguda en tot moment pel robot i aquest haurà de ser capaç d'evitar tots aquells que es troben en la seua trajectòria.

Per tant necessitem crear un món virtual en Webots que continga obstacles. A més s'introduirà el model del globus dirigible en aquest món. I amb tot açò es treballarà en el controlador del robot que actuarà sobre els motors per a manejar-lo de forma automàtica.

1.2- Eines utilitzades

Com ja s'ha comentat, el programa principal amb el qual es va a treballar és la versió 5.1.0 de Webots. Però a part d'aquest programari, també s'ha precisat l'ús de diferents programes. La majoria d'ells habituals en l'àmbit científic (Matlab, Mathematica, Autodesk Map 5…) i altres més utilitzats en qualsevol altre àmbit (Bloc de notes, Excel, etc.).

L'equip utilitzat sempre ha estat un AMD Athlon 64 Processor 3400+ a 2,4GHz i amb 512 MB de memòria RAM. I el sistema operatiu sobre el qual hem executat la principal eina de treball, Webots, ha estat Microsoft Windows XP Professional SP3.

1.2.1- Webots

Webots és un entorn de desenvolupament usat per a modelar, programar i simular robots mòbils. Amb Webots l'usuari pot dissenyar complexes configuracions de robots, amb un o més robots, iguals o diferents, en un entorn compartit. Les propietats de cada objecte, com forma, color, teixidura, massa, fricció, etc., les tria l'usuari. Per a equipar cada robot, es disposa d'un ampli ventall de sensors i actuadors simulats.

A més, els controladors dels robots poden ser programats amb el propi entorn de desenvolupament integrat (IDE) o amb entorns de desenvolupament de tercers. El comportament del robot pot ser testejat en mons amb física realista. I els programes dels controladors es poden transferir a robots reals existents en el mercat.

Webots és usat en prop de 700 universitats i centres d'investigació al voltant de tot el món. Ha estat desenvolupat pel Swiss Federal Institute of Technology en Lausana. Testejat, documentat i mantingut durant més de 10 anys. El temps de desenvolupament que assoleix estalviar pot ser molt elevat.

Aquest programari té versions per a Mac OS X, Linux i Windows. I com ja hem comentat per a la realització d'aquest PFC hem utilitzat la versió 5.1.0 de Webots que corre baix el sistema operatiu Windows XP. En aquesta versió la interfície gràfica es divideix en 4 pantalles que podem modificar, veure o ocultar al nostre gust. Però aquests aspectes els tractarem més endavant.

1.2.2- Altres

Encara que la part més important del desenvolupament d'aquest projecte s'ha realitzat amb Webots, com en qualsevol altra empresa científica ha estat necessària la utilització d'altres eines de treball menys especialitzades però no per això prescindibles.

Potser les més útils hagen estat tant Matlab com Mathematica, ambdues eines han permès realitzar de forma manual els càlculs complexos que porta a terme el programa de forma automàtica. Així doncs, quan ha estat necessari corroborar els resultats obtinguts pel programa, s'han utilitzat aquestes eines per a comprovar que els resultats eren correctes.

Com en moltes ocasions els resultats obtinguts eren desmanegats i difícils de visualitzar a primera vista, ha estat realment útil en aquest aspecte disposar d'una eina de dibuix en 3D. Autodesk Map 5 ha estat el programa instal·lat en l'ordinador que ens ha permès realitzar aquest tipus de dibuixos de forma senzilla. D'aquesta manera, dades complexes com posicions d'esferes en l'espai tridimensional, trajectòries, etc., resultaven molt més senzills de tractar quan els mostràvem dibuixats amb aquesta eina.

A part d'aquests programes, també s'han utilitzat uns altres molt més estesos en el seu ús com poden ser Microsoft Office Excel o el Bloc de notes del propi sistema operatiu. El primer d'ells ha estat especialment útil per a dibuixar gràfiques amb les dades obtingudes del programa i així poder ajustar paràmetres d'una forma ràpida i eficaç. El Bloc de notes ha facilitat la programació amb la seua funció de reemplaçar text.

1.3- Contingut de la memòria

En aquesta memòria s'explica pas a pas el desenvolupament del projecte final de carrera. I per a això s'ha dividit el contingut en 5 apartats principals, intentant així que la informació quede el més ordenada possible i siga fàcil de seguir.

El primer apartat és una introducció que parla de l'objectiu del projecte i les eines que han estat necessàries per a realitzar-lo. El punt actual s'inclou dins d'aquest apartat dedicat a la introducció.

En un segon apartat parlarem de la principal eina utilitzada, el programari de Cyberbotics Webots 5.1.0. Parlarem de les característiques d'aquesta versió, sobretot de com es divideix la seua interfície gràfica i el més important, com l'hem organitzat en el nostre projecte.

El punt més important serà el qual contempla la tercera part d'aquesta memòria. Anem a descriure l'entorn de simulació. Què és el que necessitem, com és el món que hem creat per a les nostres simulacions, com és el robot que anem a programar i com s'ha dissenyat tot açò. A més de la part visual, parlarem del controlador que maneja al nostre robot volador. Descriurem com ha de ser l'entrada de dades, com s'ha assolit el moviment del robot, la detecció d'obstacles i la forma que s'eviten els mateixos.

Després d'haver explicat tot el treball realitzat, el següent apartat està enfocat a l'ús posterior d'aquest projecte. Doncs va a intentar donar tota la informació possible perquè un futur usuari o programador puga aprendre a utilitzar aquest projecte. S'exposaran els requisits per a utilitzar el projecte, la forma que es poden modificar aspectes de l'entorn com l'entrada de dades, i la forma de modificar el controlador del globus dirigible. A més, una breu explicació de com manejar el zeppelin de forma manual.

Finalment, s'indicaran una sèrie de conclusions acompanyades de possibles àmbits d'aplicació d'aquest projecte i també possibles millores o ampliacions que es podrien implementar en un futur.

2- Webots, eina de disseny i simulació

Aquest projecte utilitza Webots (http://www.cyberbotics.com), un programari comercial de simulació de robots mòbils desenvolupat per Cyberbotics Ltd. Encara que a data d'avui la versió 6.0 de Webots ha canviat la seua interfície gràfica, la llicència que es tenia per a treballar amb Webots era de la seua versió 5.1.0. Per tant tot el relacionat amb Webots a partir d'aquest punt serà fent referència a aquesta versió.

Quan executem el programa per primera vegada, hem d'establir quin volem que siga el directori de treball. Hem de triar una carpeta on tinguem privilegis i ell ens crearà una sèrie de carpetes que contenen arxius necessaris per al funcionament del nostre projecte. Aquestes carpetes són les següents: controllers, data, objects, plugins, worlds, transfer, resources, lib, includes i doc. De totes elles, anem a prestar especial atenció a controllers i worlds. Per la senzilla raó que quan creem un món nou, es guardarà en la carpeta worlds. I quan creem un controlador nou per al nostre robot, es guardarà en la carpeta controllers.

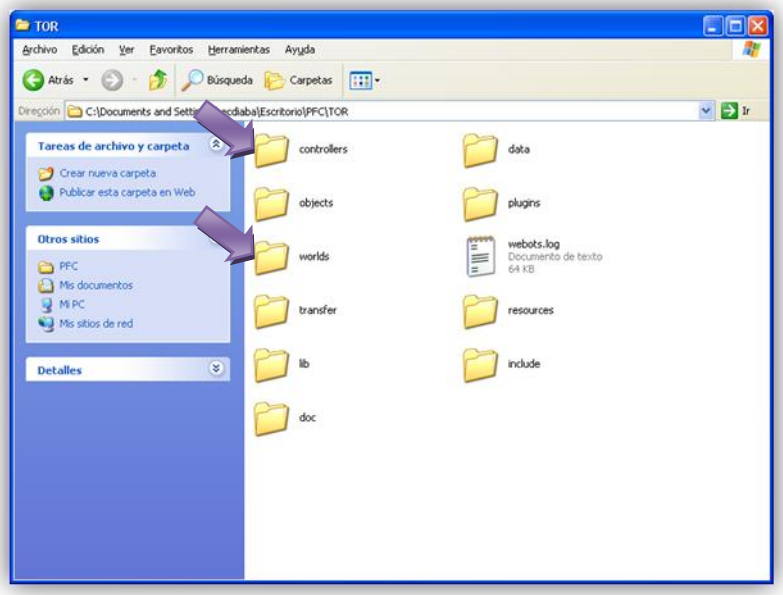

Figura 1: Directori de treball Webots 5.1.0

Com ja hem comentat abans en aquesta memòria, la interfície de Webots es divideix en 4 finestres que podem disposar on ens plaga, a més de modificar la seua grandària de forma independent. A continuació anem a tractar d'explicar la funció de cadascuna d'elles i com l'hem utilitzat per a aquest projecte en concret.

2.1- VRML

Virtual Reality Modeling Language (VRML) és el llenguatge de programació que s'utilitza en Webots per a modelar el món virtual on es simularà el comportament dels robots. Encara que cap dir que no tots els nodes i característiques d'aquest llenguatge estan disponibles.

Aquest llenguatge possibilita la descripció d'objectes 3D a partir de prototips basats en formes geomètriques bàsiques o estructures especificades a partir de vèrtex i arestes. A més es poden afegir característiques als objectes com color o materials.

Com he comentat, Webots no inclou tots els nodes de VRML97, però si que afig alguns molt importants per a treballar amb simulació de robots. Per exemple tenim el node CustomRobot, que és el qual hem utilitzat nosaltres per a crear el nostre robot volador. Aquest node ens va a permetre modelar un robot a força d'utilitzar altres nodes i a més afegir-li aspectes especials. Per exemple, al tractar-se d'un robot, anem a poder associar-li un controlador. Però aquest tema ho veurem més endavant en un apartat específic.

Nodes que aporta Webots i que podem utilitzar són el node DifferentialWheels per a crear robots terrestres amb rodes. O el node Servo, que ens va a permetre modelar robots amb servos i simular el comportament dels mateixos. Amb el node Camera podem afegir una càmera virtual i modificar les seues característiques de manera que obtindríem una imatge virtual del que seria la imatge real si muntàrem una càmera en el nostre robot. Podem afegir sensors de distància amb el node DistanceSensor, i fins i tot un receptor GPS afegint el node GPS al robot. Si volguérem establir algun tipus de comunicació, podem incloure en la simulació els nodes Emitter i Receiver. A més d'altres nodes com poden ser Charger, Gripper, Joint, LED, Light Sensor, Physics o TouchSensor, tenim el node Supervisor. Aquest node es pot utilitzar per a controlar el món i tots els robots que aquest conté.

El nostre món només va a contenir un robot del tipus CustomRobot. I encara que podríem afegir les físiques que proporciona Webots, les anem a programar en el controlador del propi robot per a poder-les personalitzar un poc més. No obstant això si que anem a utilitzar una característica dels nodes Solid molt interessant que és l'atribut boundingObject. Es tracta d'embolicar als objectes sòlids que formen el món amb una forma bàsica que ens va a permetre calcular col·lisions entre objectes. D'aquesta manera, afegirem un nivell més de realisme a la nostra simulació.

2.2- Scene Tree

Per a poder crear el món virtual sobre el qual treballarem, tenim una finestra on se'ns mostra el "Scene Tree" o arbre d'escena. Podem mostrar-la o ocultar-la fent clic en Windows>Scene Tree en la finestra principal (o Ctrl + T). Es tracta d'una finestra amb la jerarquització de nodes VRML que formen tot el món. En aquesta finestra modificarem el món afegint els nodes que siga necessari i modificant qualsevol dels seus atributs desplegant-los amb el botó "+" de cadascun d'ells.

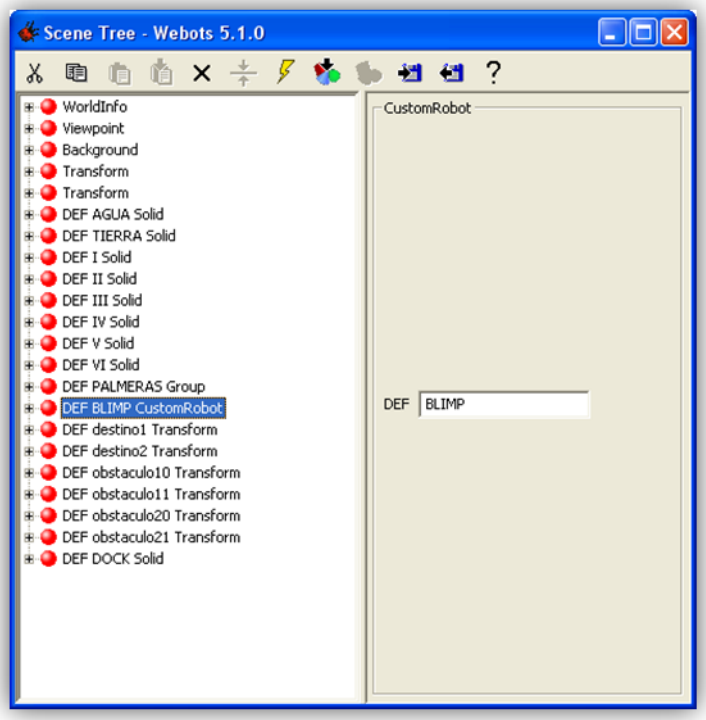

Figura 2: Finestra SceneTree de Webots 5.1.0

En la part superior d'aquesta finestra, tenim una sèrie de botons amb els quals manipular els elements de l'arbre d'escena. D'esquerra a dreta són: Tallar, copiar, pegar, pegar després del node actual, eliminar node, resetejar a l'última configuració guardada, transformar un node en un altre, inserir un nou node després de l'actual, crear un nou node, exportar, importar i l'ajuda.

L'arbre de la nostra escena conté els nodes típics de tot món virtual creat amb VRML: Worldinfo i Viewpoint. Amb ells proporcionem informació de com és el món i establim un punt de vista predeterminat. També tenim els nodes Background, 2 nodes Transform amb les llums, un sòlid per a representar l'aigua i altre per a la terra. Els sòlids definits com de l'I al VI representen les portes creades per a les proves del robot. Mentre que les palmeres són mers elements decoratius agrupats en un únic node. També tenim un CustomRobot definit com BLIMP, aquest és el nostre robot volador. Les dues destinacions són esferes de color verd i els obstacles estan representats com biesferes. Finalment, el sòlid DOCK és una estructura creada per al repòs del robot una vegada finalitza el seu recorregut.

2.3- Controladors

En Webots, d'una banda hem de modelar l'entorn de simulació creant un món com ja hem vist que podem fer. Però per altra banda hem de recordar que el gran potencial d'aquest programa resideix en la simulació de robots. Per això, una vegada creats tots els objectes del món i modelatge també el nostre robot, hem de programar les accions que es portaran a terme.

El que ens permet Webots és associar un programa (controlador) a un robot, de tal forma que podem rebre informació dels sensors que hem col·locat en el nostre robot i manar informació als actuadors que també li hem incorporat. Aquests controladors poden estar escrits en C, C++ o Java. En aquest projecte s'ha decidit treballar en C++.

Quan volem crear un controlador, hem de fer clic en Wizard>New robot controller. Llavors triem el llenguatge de programació que desitgem desenvolupar-lo i seguidament el nom que li anem a donar a aqueix nou controlador. Automàticament Webots crearà una carpeta amb el seu nom dins de la carpeta controllers del directori de treball.

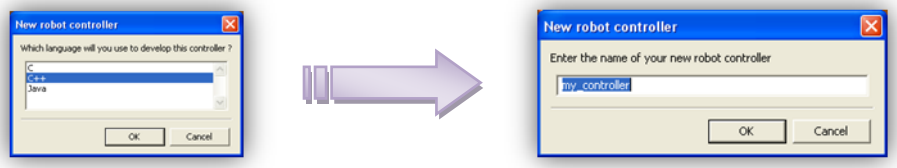

Figura 3: Creació d'un controlador

El següent pas és associar aqueix controlador a un robot en concret. Doncs podem tenir molts controladors en el nostre directori de treball, però només un serà el qual treballe sobre el nostre robot. Per a associar-lo hem d'anar a la finestra de l'arbre d'escena que ja hem vist i desplegar el node CustomRobot. Un dels seus atributs és "controller". Si premem sobre ell, en la part dreta de la finestra podem prémer sobre el botó dels punts suspensius per a triar el controlador a utilitzar.

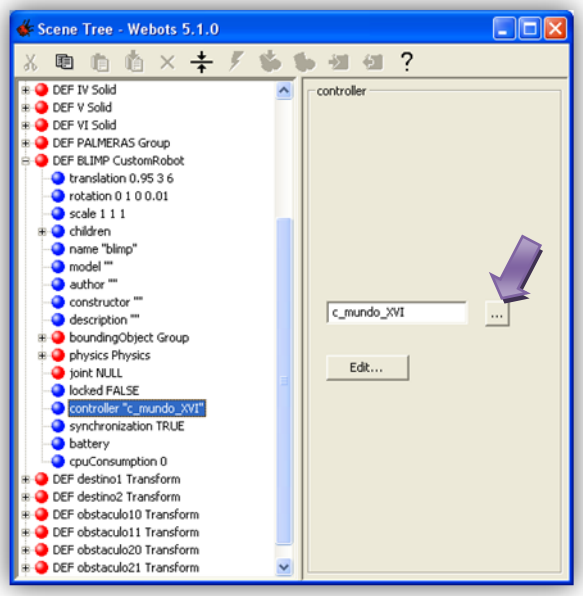

Figura 4: Associació d'un controlador a un CustomRobot

D'aquesta forma creem el controlador i ho associem al robot en qüestió, però Webots disposa d'una finestra més ideada per a poder editar el controlador. Açò vol dir que des del mateix programa anem a poder escriure el codi del controlador. Aquesta finestra d'edició de text podem mostrar-la des de Windows>Text editor, en la finestra principal.

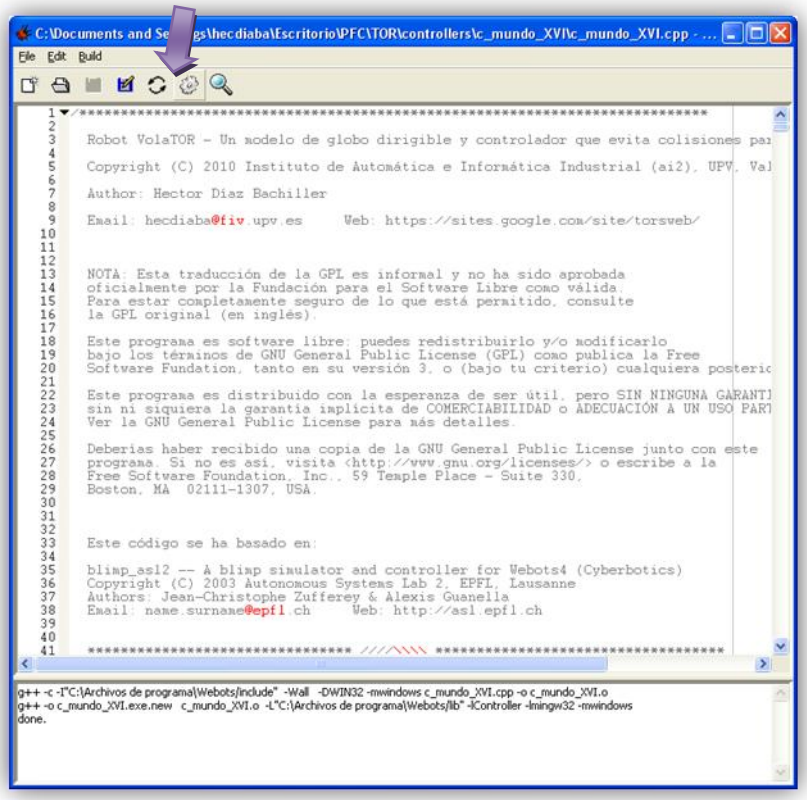

Figura 5: Finestra Text Editor de Webots 5.1.0.

La finestra es divideix en dues seccions: la primera i principal que és on s'escriu el codi, i una segona en la part inferior on es mostren els missatges de compilació. Es tracta d'una finestra amb les característiques típiques per a crear, obrir i guardar documents. Però a més d'aquestes funcionalitats, afig el botó de "revert". Aquest botó retorna el document obert a la seua última versió guardada. També disposa de l'opció "make", que realitza una compilació del codi com veem en la part inferior de la figura. Aquesta compilació crea en el directori del controlador un fitxer objecte i altre executable utilitzant el "makefile" que conté aqueixa carpeta.

Encara que per a la compilació dels controladors es poden utilitzar altres eines, per a aquest projecte s'ha decidit utilitzar l'opció de compilar des de Webots. Per això hem d'instal·lar en l'ordinador l'entorn MinGW. Aquest és un entorn de desenvolupament lliure basat en gcc, un compilador de codi obert per a C i C++. Inclou la utilitat de "make" utilitzada en Webots per a compilar amb el makefile del controlador. Està inclòs en el subdirectori devel de Webots per a Windows.

2.4- La finestra Log

Finalment, queda descriure la finestra Log. Com totes les altres, podem mostrar-la des de Windows>Log. És la finestra més simple, però no per això menys important, doncs ens va a permetre mostrar tots les dades que necessitem conèixer en temps d'execució. Des del codi podem mostrar missatges en aquesta finestra, doncs és l'eixida estàndard de les funcions que mostren dades. Tot el bolcat a aquesta finestra durant l'execució es guarda en un fitxer Webots.Log en el nostre directori de treball. D'aquesta manera podem seguir l'execució i conèixer les dades que necessitem en qualsevol moment.

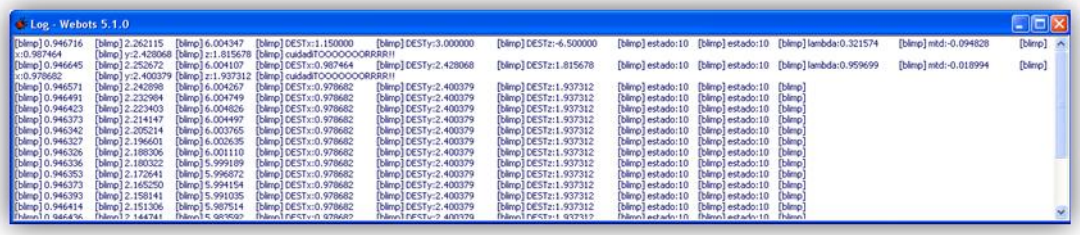

Figura 6: Finestra Log de Webots 5.1.0.

3- Entorn de simulació

3.1- Descripció de l'entorn

Al principi, l'entorn de simulació sobre el qual es pretenia treballar amb el robot anava a ser el més semblança possible a un circuit de carreres sobre un riu. No obstant això, a causa del tipus de robot volador escollit i que la creació de mons amplis es fa una miqueta desmanegada amb Webots, es va modificar aquest entorn donant lloc a un similar però amb característiques més adaptades al projecte.

L'entorn consisteix en un riu d'aigua amb les seues dues ribes de terra. En les ribes, i a manera d'ambientació, hi ha palmeres per a donar un toc més realista. A més, en la riba dreta del riu hi ha un lloc destinat al docking del globus.

Sobre l'aigua hi ha portes per les quals ha de passar el globus volador. Aquestes portes, que consisteixen en dos cilindres separats una certa distància, formen el circuit. De que la il·luminació siga correcta s'encarreguen dos punts de llum en el cel.

Per al desenvolupament del projecte s'ha optat per crear unes destinacions en aquest món que tenen forma d'esfera verda, i embolicar els obstacles amb una biesfera roja semitransparent. D'aquesta manera resulta més còmode treballar amb el robot.

El robot és un globus dirigible blanc que està suspès en l'aire i que després de realitzar el circuit marcat va a descansar a la zona violeta de docking.

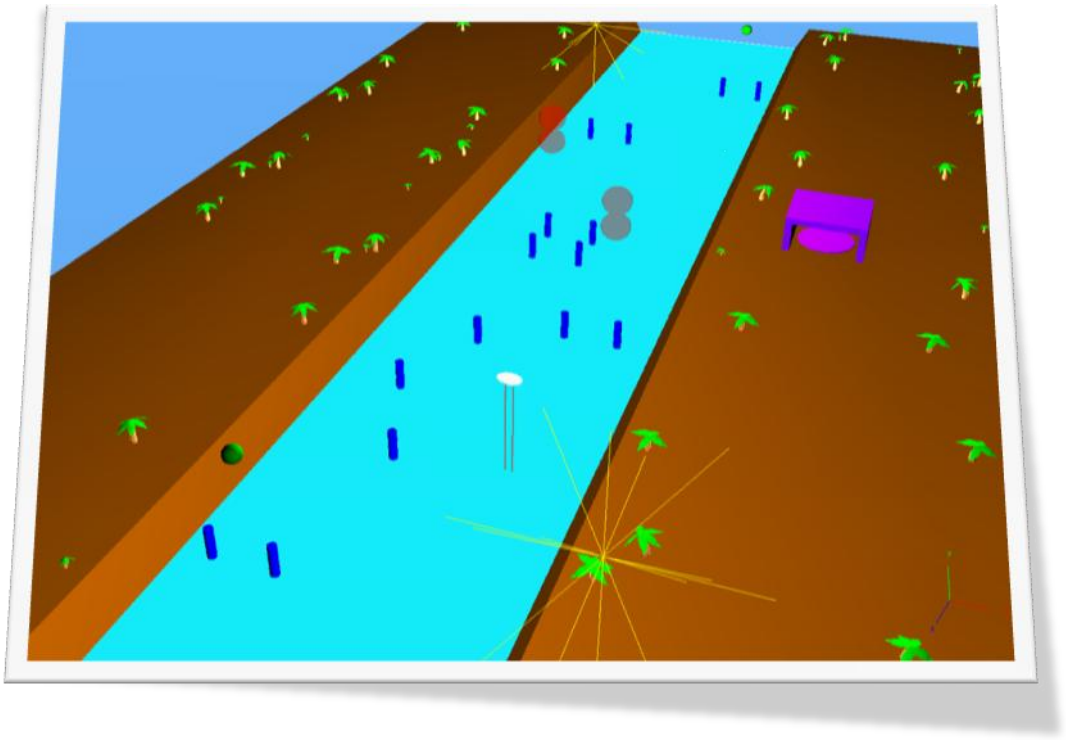

Figura 7: Vista inicial de l'entorn de simulació

3.2- Disseny dels diferents elements 3D.

A continuació es va a descriure com s'ha portat a terme la creació de l'entorn 3D descrit en l'apartat anterior. Explicant pas a pas de quins objectes i de quines característiques es compon cadascun dels elements que formen el món. Com ja hem vist, tot el relacionat amb l'entorn virtual es basarà en nodes pertanyents a un subgrup de nodes de VRML97 més alguns afegits per Webots.

A causa de la seua importància, parlarem en el primer punt dels components del món i en un segon apartat únicament del model del robot volador.

3.2.1- Disseny del món

El primer que anem a deixar clar van a ser els eixos de coordenades amb els quals anem a treballar. Es tracta d'un sistema dextrogir:

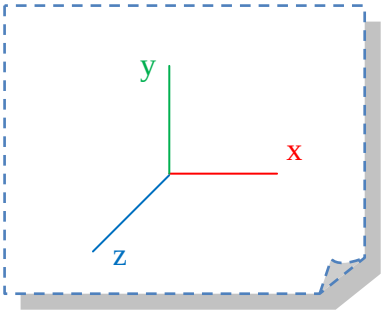

Figura 8: Sistema de coordenades

Per a començar, parlarem de la terra. Les ribes del riu s'han modelat mitjançant una extrusió de color marró la qual s'ha escalat segons el factor 2x2x2. La longitud de la extrusió és de 10 metres, i els punts que formen el crossSection són d'esquerra a dreta (-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 vindria a ser:

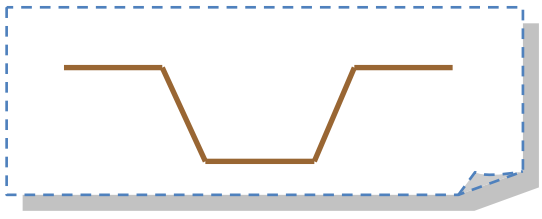

Figura 9: CrossSection de la terra

A aquest sòlid se li ha afegit un boundingObject. Ho forma un grup de dos Box, un per a cada riba, de la mateixa longitud que la extrusió. D'aquesta forma serà possible calcular les col·lisions amb les ribes, però no amb el fons de la llera (no ens interessa).

Per a seguir amb el riu, anem ara a comentar com s'ha creat l'aigua. També és un sòlid, però la seua forma és una única Box de color blau de 4x2x20 metres. De tal forma que l'efecte obtingut juntament amb les ribes de la terra és bastant real. Com

boundingObject s'ha utilitzat la mateixa Box utilitzant l'opció DEF de VRML97 per a definir nodes i reutilitzar-los.

En les ribes del riu, és a dir, en la terra trobem una sèrie de palmeres de diferents grandàries. Estan pensades com objecte decoratiu i no tenen incorporades col·lisions, doncs es tracta simplement de millorar l'aspecte del món. Aquestes palmeres han estat importades a partir d'un model VRML97 ja existent. Estan agrupades i se'ls han aplicat una sèrie de transformacions per a aconseguir aqueixa disposició en concret.

Amb el node Background ho que fem és definir el color del cel, que per al nostre cas serà blau. I amb 2 PointLight vam afegir la llum necessària per a il·luminar tots els objectes. La intensitat de la llum serà de 2 unitats per a la primera i 4 per a la segona. El color de la llum serà blanca per a ambdues (1,1,1), així com el ràdio que serà de 1000 unitats i sense atenuació. El node Viewpoint ve per defecte i bàsicament l'única cosa que fa és marcar quin és el punt de vista inicial. Nosaltres ho hem canviat a un més d'acord amb el món.

Com dèiem, la idea inicial era crear una espècie de circuit, i per a això hem creat una sèrie de portes per les quals el robot ha de passar. En concret són un total de 6 portes. No totes són iguals, però són molt similars. Les nombre 1, 2, 3 i 6 estan compostes per un grup de dos cilindres blaus. La separació entre ambdós cilindres és bastant ajustada a la grandària del globus dirigible. La grandària de cada cilindre és de 0.05 metres de ràdio i 0,4 m d'altura. La porta 4 és un grup de 4 cilindres amatents formant un quadrat, mentre que la porta 5 consta de 3 cilindres en zig-zag. Tots els cilindres que formen les portes són sòlids i també se'ls apliquen col·lisions. Encara que per al procés de desenvolupament aquest ha estat desactivat.

Les destinacions que se li marquen al zeppelin han estat modelats en el món com esferes verdes de 0,1 metres de ràdio amb 20 subdivisions. D'altra banda, els obstacles marcats per a evitar es mostren en el món com biesferes semitransparents de color roig. Cada obstacle està format per dues esferes de 0,2 metres de ràdio i 20 subdivisions. Per tant, el centre de l'esfera superior queda desplaçat (pel que fa al centre de l'obstacle) 0,2 metres positius en l'eix Y, mentre que l'esfera inferior ho farà en el sentit negatiu de l'eix Y.

Només queda descriure el dock, doncs el disseny del robot ho deixarem per a l'apartat següent. El dock és una estructura de color rosa on el robot anirà a descansar una vegada haja finalitzat el seu recorregut. Consta de 4 peces agrupades per a ser reutilitzades com boundingObject. En concret són 2 parets de 0.1x0.5x1.0 metres separades 1 metre de distància en l'eix X. Un sostre de 1.1x0.1x1.0 metres col·locat sobre les parets anteriors. I un sòl amb forma de cilindre de 0,4 metres de ràdio i 0,05 metres d'altura amatent en el centre de la part inferior de l'estructura.

3.2.2- Disseny del robot volador

Anem a veure una imatge del robot real i altra del model del globus dirigible utilitzat en Webots assenyalant les diferents parts de les quals es compon. D'aquesta forma podrem entendre de forma visual com és el robot i com s'ha modelat. Cap dir que aquest model és una modificació del robot blimp_asl2 que ve com exemple en Webots.

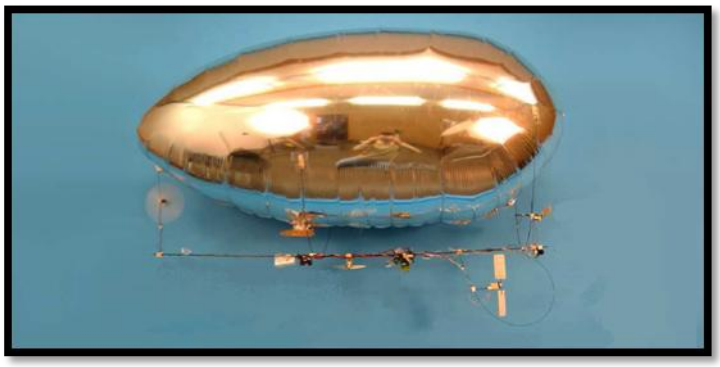

Figura 10: Robot volador real construït en la EPFL

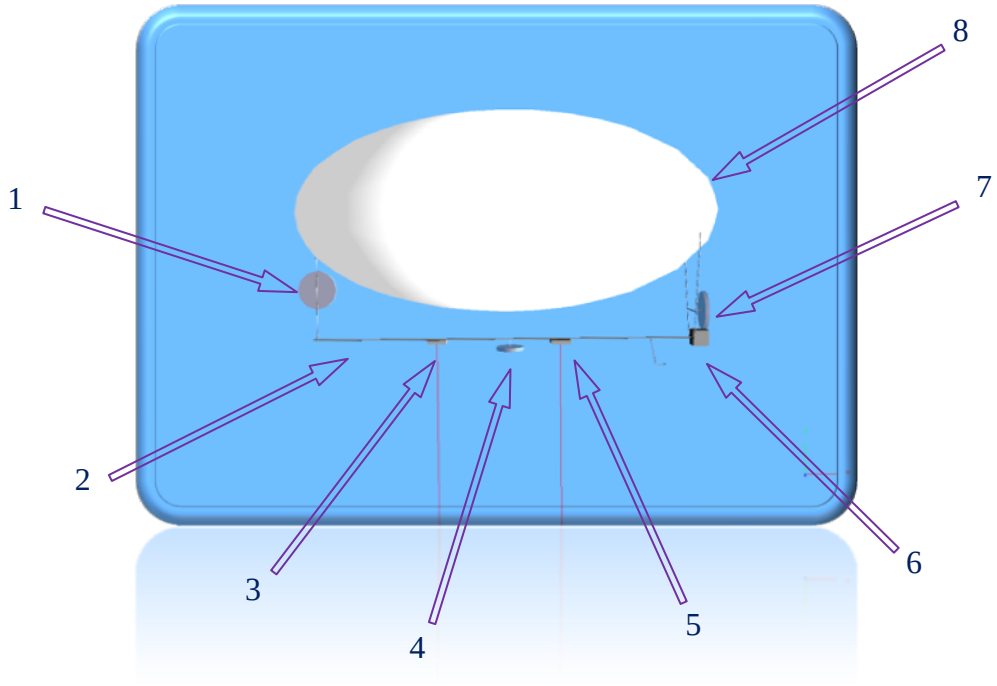

Figura 11: Model amb les parts del robot volador

A continuació es mostra la llista d'elements que podem veure en la imatge:

1.- Motor de YAW + hèlix
2.- Estructura del globus dirigible
3.- Bateries + sensor de distància
4.- Motor d'ascens/descens + hèlix + GPS
5.- Processador + sensor de distància
6.- Càmera 800x600 color
7.- Motor d'avanç/reculada + hèlix
8.- Globus d'heli

L'estructura principal (2) sobre la qual s'acobla la resta de components del robot està modelada a força de cilindres de color grisenc per a aconseguir l'efecte metàl·lic. Sobre aquesta, es subjecta el globus d'heli (8) modelat com una esfera blanca. La forma s'aconsegueix gràcies a un escalat una mica major al doble en l'eix X que en els eixos Y i Z.

Els 3 motors dels quals consta el robot per a realitzar els seus moviments s'han muntat sobre l'estructura i tenen forma de cilindre per a simular les hèlixs. El primer d'ells (1) permet la rotació del globus sobre el seu eix vertical, és a dir, amb aquest motor s'aconsegueix el que cridem yaw. S'ha muntat en la part del darrere. En la part central trobem el segon motor (4), que ens permetrà ascendir i descendir, i el receptor GPS. Finalment i en la part davantera està el tercer motor (7). És l'encarregat de fer avançar i retrocedir al robot.

A cadascun dels costats del segon motor trobem una caixa també de color metàl·lic. La qual està en la part del darrere correspon a les bateries del robot (3). En aqueixa posició s'ha afegit un sensor de distància per a controlar l'altitud del globus. D'altra banda, la caixa de la part davantera simula el microprocessador (5) i altre sensor de distància amb el mateix propòsit que l'anterior. Únicament falta parlar de la càmera (6) que es situa en la part frontal del robot. Per a modelar-la s'ha utilitzat altra caixa (aquesta vegada una mica major) d'un material similar a la resta.

3.3- Disseny del controlador

Encara que en Webots podem crear un món amb diversos robots i cadascun d'ells pot tenir associat un controlador, en el nostre projecte únicament existeix un robot i bastarà amb implementar un controlador per a ell.

Anem a dividir el disseny del controlador en 4 parts bé diferenciades. D'una banda parlarem de com s'ha dissenyat l'entrada de dades. És a dir, on i com es situen aqueixes dades que l'usuari ha d'introduir per a utilitzar aquest projecte. També tenim la part de controlar el moviment del robot, ja que el mateix pot treballar de forma automàtica o manual. El robot haurà de detectar els obstacles que li faran col·lisionar d'entre tots els obstacles que existisquen en el món. I finalment haurà de ser capaç d'evitar aquests obstacles de tal forma que assolisca arribar a les destinacions marcades sense col·lisionar en cap moment.

A part de tot l'esmentat, com ja vam parlar anteriorment, els càlculs físics del món es faran també en el controlador del robot. D'aquesta forma, podem controlar millor tots els aspectes relacionats amb el fregament, la inèrcia, etc. que per a un robot aeri cobren una major importància que si treballàrem amb robots terrestres.

3.3.1- Entrada de dades

Les dades que l'usuari ha d'introduir perquè el controlador puga treballar es divideixen en dues parts. Els corresponents al circuit que es desitja que realitze el globus dirigible i els quals pertanyen als obstacles que el globus ha d'esquivar.

Els primers són posicions tridimensionals del món que han d'estar en zones accessibles pel robot. Per exemple, una destinació no ha d'estar mai dins de l'espai que correspon a l'aigua. El robot evitaria submergir-se gràcies al tractament que se li ha donat als sensors de distància que controlen l'altitud, però és un cas que no hauria de donar-se. Amb aquestes posicions col·locades en forma de pila anem a marcar el circuit que volem que recórrega el robot. O el que és el mateix, el robot anirà de destinació a destinació seguint l'ordre que els col·loquem. Hem de tenir en compte que les 3 últimes posicions corresponen a la maniobra de docking i per tant no hauríem de modificar-les llevat que siga necessari.

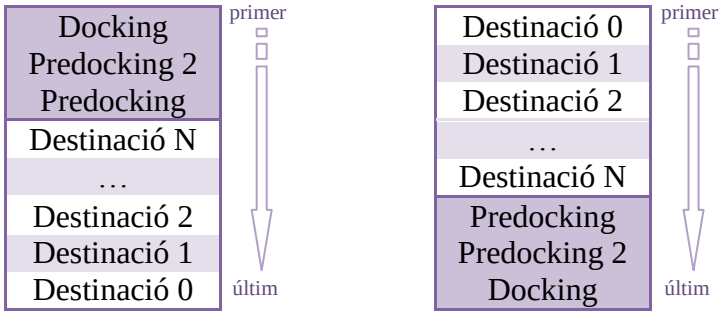

Figura 12: Ordre d'introducció en la pila (esquerra). Ordre d'extracció de la pila (dreta)

La introducció d'aquestes dades que marquen les destinacions del robot ha de fer-se directament sobre el codi del controlador. Una possible extensió d'aquest projecte seria adaptar el codi perquè la pila on es guarden les destinacions fora emplenada amb la lectura d'un fitxer d'entrada que els continguera. Però aquests temes els tractarem més endavant cap al final de la memòria.

També hem d'introduir les dades dels obstacles. Aquests són biesferes, i per tant la dada que introduirem per a cada biesfera serà el punt d'unió d'ambdues esferes, el que podríem cridar centre de l'obstacle. En aquesta ocasió la forma d'introduir-lo també serà directament en el codi del controlador i serà en forma de vector, doncs l'ordre no importa a l'hora d'emmagatzemar els obstacles. Com abans, es tracta d'una posició 3D. Però altra dada important que hem d'introduir sobre els obstacles és el ràdio de les seues esferes. Al principi el codi està preparat per a obstacles de mateix ràdio, encara que no seria complicat adaptar-lo perquè els obstacles pogueren tenir ràdios de diferent longitud.

3.3.2- El moviment del robot

El nostre robot consta de 3 motors com ja hem vist en la part del disseny del robot. Cadascun d'ells va a permetre al robot realitzar diferents moviments. Per a poder referir-nos més clarament a cadascun d'ells i evitar confusions els numerarem a partir d'ara de la següent manera. Quan fem referència al motor 1, estarem parlant del motor situat en la part davantera del robot (7 en la Figura 11). El motor 2 serà el de la zona baixa del robot (4 en la Figura 11). Mentre que parlarem del motor 3 quan ens referim al de la part del darrere (1 en la Figura 11).

Arribats a aquest punt hem de fer una distinció entre el control del robot de forma manual i el control de forma automàtica. Ja que el controlador del robot està preparat per a poder canviar d'una manera a l'altre prement la tecla "P". En els dos casos anem a acabar treballant en termes de l'acceleració que li apliquem als motors, però en el cas manual, aquesta la controlarà l'usuari al seu gust amb l'ajuda del teclat i en el cas automàtic serà el propi controlador el qual s'encarregue de calcular-la en tot moment.

Per a conèixer en profunditat el funcionament de la física existent en el controlador i causant del moviment del robot, es pot consultar el document original de Jean-Christophe Zufferey et al. "Flying over the Reality Gap: From Simulated to Real Indoor Airships", 2006 [ZGeta06]. Però de moment bastarà amb tenir constància que al tractar-se d'un robot aeri no anem a treballar amb velocitat com faríem en un robot terrestre sinó que la nostra eina per a realitzar els moviments serà l'acceleració dels motors. Així doncs, el que farem serà aplicar un multiplicador de l'acceleració a cada motor i d'aquesta forma incrementar o disminuir el valor final d'acceleració que se li aplica al motor en qüestió.

Hem de saber que el sistema de coordenades del globus és diferent al del món virtual de Webots. Encara que açò es soluciona en la programació i a efectes pràctics l'usuari no ho nota en cap moment. Ni tan sols quan el control del globus dirigible ho realitza ell.

Com la manera d'utilitzar el robot de forma manual s'explicarà en l'apartat 4.4, anem a explicar a continuació el funcionament automàtic del globus.

Pujar/Baixar

AL tractar-se d'un robot volador el primer que anem a haver de controlar és l'altitud a la qual es troba el robot. Podem conèixer-la gràcies a la posició que ens proporciona el node GPS. I així, sabent l'altitud a la qual es troba el globus, podem actuar sobre el motor 2 per a dur-lo a altra diferent, amb el que aconseguiríem un moviment de pujar o baixar el globus en l'espai. Encara que sempre hem de tenir en compte que si deixem d'actuar sobre el motor, el globus dirigible caurà per l'acció de l'acceleració gravitatòria que s'ha afegit a les físiques del món. Per tant, en tot moment (encara que el que desitgem siga deixar el robot immòbil en un punt) el motor 2 haurà de ser controlat i estar actuant.

Perquè el globus dirigible no es descontrole en cap moment, quan els sensors de distància situats en la part inferior del globus detecten que aquest es troba massa prop de la superfície, el motor 2 es posa en funcionament en una maniobra d'emergència que li proporciona més altitud. De la mateixa manera, però utilitzant les dades del GPS en lloc dels sensors de distància, quan el globus s'eleva fora del rang desitjat també s'inicia una maniobra automàtica d'emergència que ho estabilitza a una altitud correcta per sota d'aqueix llindar.

No obstant això mentre el globus es troba en la zona de maniobres desitjada, l'altitud és controlada pel motor 2 en funció de l'altitud actual i l'altitud de la pròxima destinació. Doncs les destinacions són posicions 3D i ens interessa arribar-los a amb la major exactitud possible, inclosa l'altitud.

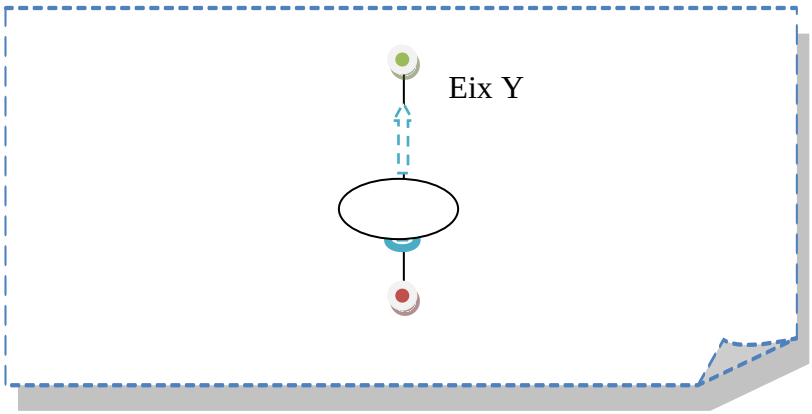

Figura 13: Moviment Pujar/Baixar del globus dirigible

Per tant, dins del controlador que s'encarrega de supervisar tot el relacionat amb el robot, necessitarem programar un "controlador" que actue sobre el motor 2. Però hem de distingir entre el controlador que li diem a Webots que assigne al nostre robot i una part de codi d'aquest controlador que s'encarregarà únicament d'actuar sobre el motor 2.

A continuació explicarem com, després d'estudiar el cas, el robot, la física associada, etc. hem escollit el tipus de controlador i les característiques que ho feien útil per al nostre projecte. La tasca d'aquest controlador és saber a cada moment quant s'ha d'actuar sobre el motor 2 perquè la nova altitud del globus s'arribe a de forma correcta en el període de temps correcte.

D'entre els tipus de controlador que podríem haver escollit, després de diverses proves hem optat per un controlador proporcional-derivatiu. La part integral d'un PID no ens aportava gens útil i per aquest motiu no l'hem incorporat. Els valors que s'apliquen a cadascuna dels guanys, varien segons l'estat en el qual es troba el robot. Açò és així degut al fet que no es pot aplicar la mateixa acció si per exemple l'única cosa que volem és mantenir-nos a una altitud fixa, que si a més hem d'estar en continu moviment. Aquests valors formen part de la programació del controlador i es poden veure en el codi del robot.

A més de la part de programació corresponent a controlar la potència, fa mancada canviar el sentit d'aquesta ja que el motor 2 està situat en el globus de manera que valors positius fan baixar al robot. Però açò és una simple qüestió de programació relacionada amb el disseny que està comentada en el codi i no implica sols que un canvi de signe a la variable que controla la potència subministrada.

Avançar/Retrocedir

Partim de la base que el robot es troba en una posició actual (origen) i ha d'arribar a una posició final (destinació). En cada iteració l'origen haurà canviat si hi ha hagut moviment, però la destinació romandrà sense canvis fins que s'arribe a. Per a començar, posarem com condició que ambdues posicions es troben en el mateix eix (paral·lel a l'eix Z del món), doncs el nostre món té profunditat en aqueix eix i és cap a on ens interessa que es realitze el moviment. I a més, el robot estarà orientat cap a la destinació. D'aquesta forma, posar en funcionament el motor 1 amb un multiplicador positiu significarà que el robot avança cap a avant alhora que es va acostant a la destinació (veure Figura 11). Si utilitzàrem un multiplicador negatiu, el robot es mouria cap a arrere, allunyant-se de la destinació. Ja que açò implica que el motor gire en sentit contrari i amb aquest l'hèlix que aporta el moviment.

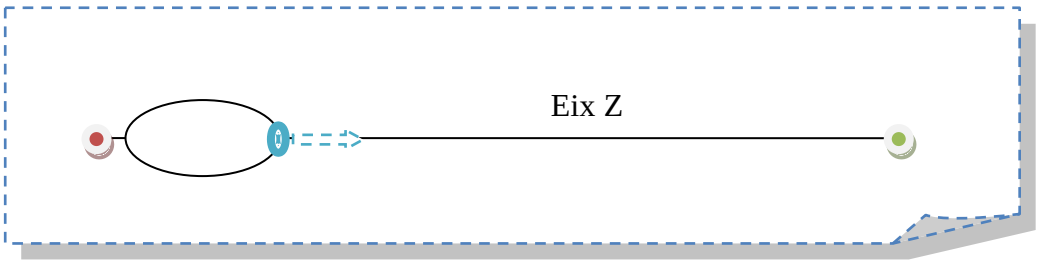

Figura 14: Moviment Avançar/Retrocedir del robot

Ja sabem com anem a començar a realitzar el moviment més senzill, però hem de tenir en compte que no ens val únicament amb arribar a la destinació. Si estiguérem treballant amb un vehicle terrestre el fregament actuaria en el nostre favor i al deixar de subministrar potència als motors, el vehicle pràcticament pararia en el lloc correcte. Però com sempre, al tractar-se d'un vehicle aeri el fregament actuarà en menor mesura i la inèrcia farà que el robot es desplace molt més allà de la destinació si l'única cosa que fem és deixar de subministrar potència als motors a l'arribar a la destinació. Açò no és viable per al nostre projecte i hem de controlar aquesta situació. Per a això el que farem serà incorporar un controlador proporcional-derivatiu al codi de manera que actue sobre la variable del multiplicador de potència. Així el que estarem fent és controlar la

potència que se li subministra al motor 1 tenint en compte la distància a la qual es troba el robot de la destinació. Quan el robot es trobe relativament lluny, la potència serà major i el robot es mourà més ràpidament. No obstant això, conforme el robot es vaja acostant a la destinació, la potència amb la qual s'actuarà sobre el motor 1 anirà disminuint-se fins al punt d'utilitzar fins i tot un multiplicador negatiu per a aconseguir que la velocitat del globus baixe fins a valors propers a zero en el punt de la destinació. D'aquesta forma, ens estem avançant al que succeiria i aconseguim actuar abans que el globus sobrepasse la destinació parant-lo en el lloc correcte.

Tot aquest funcionament ho realitza de nou un controlador, però aquesta vegada aplicat a la variable encarregada d'ajustar la potència del motor 1. Aquest controlador és del tipus proporcional-derivatiu, però l'ajustament de les variables implicades en el controlador és diferent al del motor 2. A més, igual que en el cas anterior, segons l'estat que es trobe el robot i en el qual haja d'actuar aquest controlador, els valors de les variables canviaran. Cal adonar-se que així com quan treballàvem amb el motor 2 era necessari que estiguera actuant de forma contínua, en aquest cas no va a ser necessari ja que no hi ha cap força gravitatòria ni de cap altre tipus que estiga actuant contínuament sobre el globus. Totes les forces que s'exerceixen en aquesta adreça provenen del propi moviment del robot com per exemple la inèrcia. El que es tracta d'explicar és que caldrà controlar forces com la inèrcia amb aquest motor, però no forces contínues com la gravetat.

Esquerra/Dreta

El nostre globus consta d'un tercer motor (motor 3) que li permet realitzar girs en l'espai. Però aquests girs tenen una característica molt important i que ha dut més temps controlar que els altres moviments. I és que estem parlant d'un motor que està situat en la cua del globus dirigible a manera de timó. Açò provoca que les forces exercides per l'hèlix d'aquest motor facen girar la part del darrere del globus. I amb això s'aconsegueix un gir característic molt diferent al que podem obtenir amb un robot terrestre que conste de rodes diferencials per exemple. Açò és degut al fet que l'eix de rotació deixa d'estar en el centre o en un dels laterals i passa a estar en la part davantera.

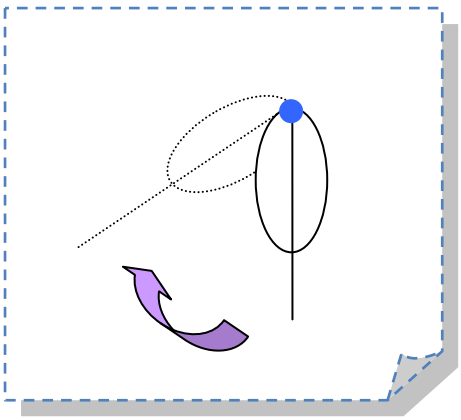

Figura 15: Moviment de rotació Esquerra/Dreta del robot

Realitzar aquest tipus de girs no és trivial i de nou és necessari aplicar un controlador sobre el multiplicador de la potència del motor 3 per assolir que els girs siguen correctes i alhora precisos.

El primer que es va intentar per a solucionar aquesta dificultat va anar en primer lloc cercar l'orientació correcta cap a la destinació amb l'ajuda d'uns paràmetres concrets de controlador PD i una vegada orientat el globus deixar d'actuar sobre el motor 3 per a començar un moviment rectilini. El problema és que el que s'aconseguia era un moviment molt artificial, en el sentit que els moviments es dividien en fases i a més la precisió era inacceptable degut sobretot a la inèrcia que seguia actuant sobre el robot.

Així que es va abordar el problema d'altra manera. El que s'ha fet ha estat aprofitar l'autòmat d'estats creat per al robot. En l'estat de "BUSCA_ANGULO" s'utilitza un controlador PD per a orientar el robot cap a la seua destinació igual que comentàvem anteriorment. I una vegada s'aconsegueix tenir orientat el globus cap a la destinació amb un error prou menut, es passa a l'estat "AVANZA".

En l'estat "AVANZA", a més de seguir controlant els dos moviments vists anteriorment, es comença a controlar el motor 3 d'una forma especial. Es fa amb un controlador PID, similar al de l'estat "BUSCA_ANGULO" només que aquesta vegada afegim la part integral del controlador per a obtenir el mínim error possible en el transitori. Però amb açò únicament no era suficient i l'opció per la qual es va optar va ser aplicar altre controlador PID però aquesta vegada sobre l'error comès en l'eix X. És a dir, amb un primer controlador actuem per a corregir l'angle del robot, i amb el segon obliguem a corregir-lo més o menys depenent del lluny que es trobe el globus dirigible de l'eix paral·lel a l'eix Z que ha d'arribar a el globus per a arribar a la destinació. De forma gràfica seria de la següent manera:

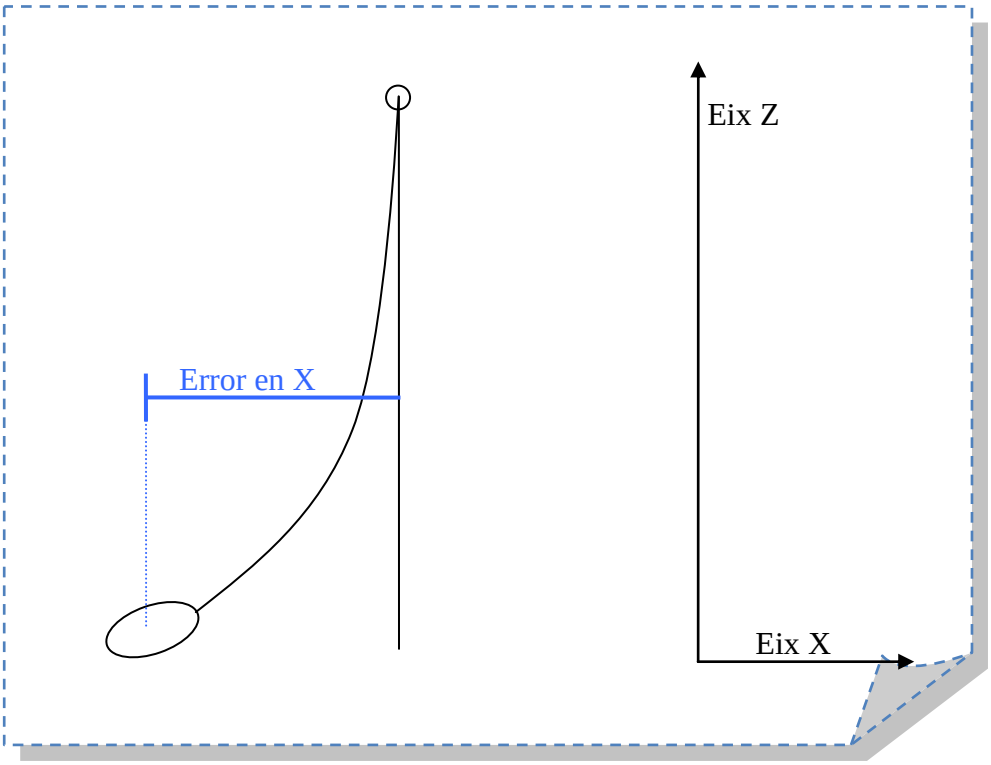

Figura 16: Trajectòria del robot

L'últim aspecte a tenir en compte en aquest cas és que com estem treballant amb angles de Euler hauríem de tenir especial atenció a tractar els casos d'angles positius i

negatius de la manera adequada. Per aquest motiu el codi estiga duplicat en aqueix aspecte, per a tenir en compte en quin cas dels dos es troba i com actuar.

Moviment conjunt

Ja hem explicat un a un els tipus de moviment que el robot pot realitzar i la forma de controlar-los per separat. Però açò només va ser el principi, doncs fins que no es va tenir controlada l'altitud, no es va començar a estudiar el cas de moure el robot en línia recta. I una vegada trobada la configuració adequada per a arribar a un punt situat en línia recta a una altura diferent, es va incorporar el controlador del gir. Primer simplement realitzar un gir cercant l'orientació correcta cap a la destinació, romanent el globus estàtic. Més tard afegint el moviment i el controlador explicat més amunt per a arribar a la posició de destinació seguint el seu eix paral·lel a l'eix Z. Arribat a aqueix punt funcionava, però no era possible controlar la inèrcia com era necessari i el robot arribava a la destinació amb massa error en l'eix X. La solució va vindre al tenir en compte, en el controlador del motor 1, la línia recta que uneix la posició actual amb la destinació en lloc de l'error en l'eix Z. El que podríem cridar la hipotenusa del triangle recte format per l'origen i la destinació. Tenint açò últim en compte és com s'han obtingut els resultats correctes i d'aquesta forma el robot realitza els moviments d'una destinació a un altre amb un error mínim.

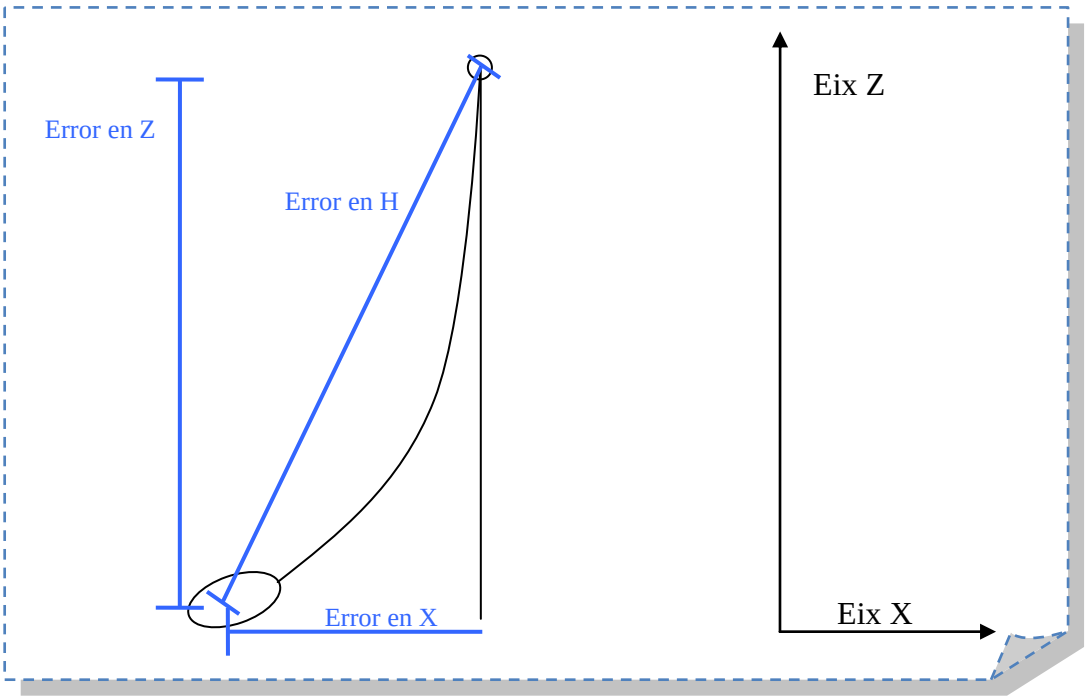

Figura 17: Moviment conjunt i errors en els eixos

3.3.3- Detecció de col·lisions

De moment tenim controlat tot el relacionat amb el moviment del robot. Ja hem vist com, i som capaços de manar al robot des d'un origen a qualsevol destinació. El globus dirigible arribarà a aqueix punt cometent un error mínim dins dels marges acceptables. Però anem a introduir un element més al projecte que són els obstacles.

Els obstacles els anem a dissenyar amb forma de cilindre, doncs recordem que la idea original era realitzar un circuit. Les portes dels circuits seran els nostres obstacles i per això s'ha triat el cilindre per a representar-les.

A més de modelar-les en l'entorn visual de Webots, hem d'incorporar-les al codi del controlador. I com els obstacles van a ser objectes coneguts per endavant, els anem a representar en memòria com un vector d'obstacles. Cada obstacle es representarà com un punt de l'espai 3D. Aquest punt que emmagatzemem és el punt que es troba en el centre del cilindre.

Cal fer un aclariment en aquest punt i és que s'ha considerat per a aquest projecte que tots els obstacles són d'igual grandària. Però una possible ampliació del projecte seria no només emmagatzemar el centre del cilindre, sinó també la seua grandària. D'aquesta forma estaríem permetent treballar més tard amb obstacles de diferent grandària.

Però anem a explicar la manera de detectar obstacles que hem implementat en el robot. Perquè com hem dit, ja som capaços de dur el globus dirigible d'un punt a un altre, però ara vam necessitar que detecte els obstacles que hi ha en la seua trajectòria per a saber si col·lisionarà amb ells o no. O el que és el mateix, detectar els obstacles, per a més tard generar automàticament trajectòries lliures de col·lisió. Per això s'ha creat una rutina que és cridada per a cadascun dels obstacles de l'entorn (recordem que tots els obstacles són coneguts i estan guardats en un vector). A la rutina "detecta_colision" se li passen com arguments l'origen, la destinació i una biesfera creada a partir de la posició de l'obstacle.

El funcionament de la rutina consisteix a calcular els valors que ens permetran saber si estant en aqueix origen i volent arribar a aqueixa destinació, col·lisionarà amb aqueix obstacle. La rutina té implementada la primera part de l'algorisme de càlcul de distàncies entre Poli-Esferes [Ber98]. La segona part d'aquest algorisme serà explicada més endavant en l'apartat 3.3.4- Evitació d'obstacles.

El que va a fer la rutina és calcular les diferències de Minkowski [Ber98] a partir de les 4 esferes que té com dades d'entrada; origen, destinació i les dues esferes que formen la bi-esfera de l'obstacle. Formant així una tetra-esfera o poliesfera de Minkowski on podem saber que si el punt origen està dins haurà col·lisió, i si està fora no l'haurà.

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

Per a conèixer si un punt està dins o no de la poliesfera de Minkowski serà necessari fer ús de la transformada de Hough [Ber98] ja que les rectes de Hough ens van a servir tant per açò com per saber quina és la distància que fa que el ràdio de l'esfera quede en la superfície de la poliesfera.

Una vegada coneixem els valors de la poliesfera de Minkowski, anem a treballar amb les 2 tri-esferes que la constitueixen si la dividim en dues, i també amb les distintes bi-

esferes que la formen. Per a conèixer la posició de l'origen O anem a redefinir-lo de la següent forma:

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

Per a simplificar direm que λO1 i λO2 són la posició relativa de l'origen respecte als centres d'una de les tri-esferes. Mentre que DO és la distància de l'origen al triangle que formen aquests centres.

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

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

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

(3)

Depenent dels valors de les lambdes i de la seua suma, anem a haver de realitzar uns càlculs o altres, doncs podem trobar-nos en el cas d'una tri-esfera, en el d'una bi-esfera, en el del mínim entre 2 bi-esferes o també pot afectar a l'altra tri-esfera i donar-se aquests mateixos casos. Però una vegada calculades λO1, λO2 i λO1 + λO2 ja sabem en quin cas estarem i podrem consultar les taules per a saber quins càlculs realitzar.

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 cas que els càlculs indiquen que s'ha de treballar amb l'altra triesfera, abans de res és imprescindible realitzar un menut ajustament en les lambdes i després es pot procedir a continuar amb els càlculs:

$$\lambda_{01} = -\lambda_{02}$$

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

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 tractar-se d'una triesfera només ens queda calcular DO a partir de la projecció de l'origen sobre la triesfera. Amb açò ja som capaços de calcular la MTD (Mínima Distància de Translació) necessària per a deixar el ràdio en la superfície 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 cas d'haver de calcular la distància a una biesfera hem de calcular una *lambda diferent a més de la projecció de l'origen sobre la biesfera. I amb açò i els ràdios dels centres de la biesfera ja podríem calcular la MTD. Cal tenir en compte que si λ és menor que 0 la MTD serà la distància al centre de la primera esfera menys la seua ràdio. I si és major que 1 estarem en el mateix cas però de la segona 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)

Arribats a aquest punt, pot ser que hàgem hagut de calcular la distància a una *tri-esfera, a una bi-esfera, a una esfera o a dues bi-esferes i calcular la mínima. Però en qualsevol cas la rutina retornarà la MTD calculada, les lambdes necessàries i la projecció de l'origen.

Com tot el procés de la rutina es fa per a cada obstacle, finalment només queda comprovar si la MTD és major que 0 (no haurà col·lisió amb aqueix obstacle) o si per contra la MTD és menor o igual a 0 significant açò que el robot col·lisionarà amb aqueix obstacle llevat que s'evite d'alguna manera. Amb açò arribem al següent apartat d'aquesta memòria, on s'explica la tècnica seguida per a evitar col·lisionar amb els obstacles. Si encara així es necessita aprofundir més qualsevol dels aspectes de l'algorisme de càlcul de distàncies entre Poli-Esferes, de nou es pot trobar una àmplia explicació en [Ber98].

3.3.4- Evitació d'obstacles

La comesa de l'apartat anterior no és altre sinó el de realitzar els càlculs previs a aquest. La posició dels obstacles la coneixíem, i ara gràcies a la detecció d'obstacles sabem si el globus dirigible va a col·lisionar amb ells. El que hem de fer en aquest apartat és d'una banda trobar el primer obstacle amb el qual va a col·lisionar el robot i després utilitzar els càlculs realitzats per a manar al robot a una posició de no col·lisió.

Com ja tenim calculada la MTD de cada obstacle i la seua posició, el que anem a fer és recórrer el vector d'obstacles cercant aquell que complisca 4 condicions; ser un dels que col·lisionen amb el robot, ser el més proper al robot, estar per davant del robot i estar abans de la destinació del robot. Amb açò tenim l'obstacle que hem d'evitar.

Seguint amb l'algorisme de càlcul de distàncies entre Poli-Esferes, podem utilitzar les dades obtingudes per a calcular quin serà el punt de màxima penetració entre l'origen i la biesfera de l'obstacle. I tenint aquest punt, l'única cosa que hauríem de fer per a evitar la col·lisió serà introduir una nova destinació intermedi desplaçant aquest punt de màxima penetració la distància necessària en l'eix de l'origen projectat. No obstant això, en cas de no ser suficient el marge d'error, podem afegir de forma opcional un percentatge com llindar de confiança.

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

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

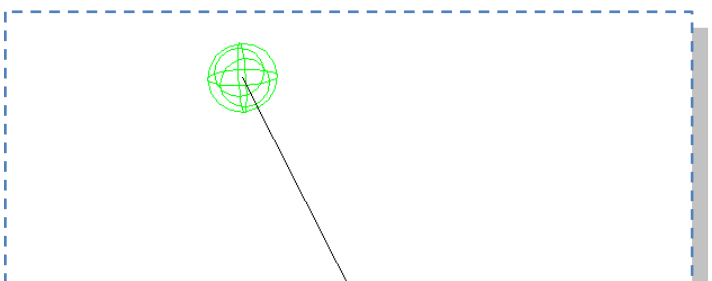

Figura 18: Càlcul del punt intermedi

En la figura anterior les esferes representen el volum que ocupa cadascun dels objectes del món. L'esfera roja embolica per complet al robot i el seu centre serà el nostre punt origen. L'esfera verda és la destinació. Les dues esferes blava fosc representen una biesfera que engloba a l'obstacle a evitar. Mentre que l'esfera blava clar és el punt de màxima penetració calculat. En rosa tenim el nou punt intermedi que assegura la no col·lisió, és a dir, la nova destinació que haurà de seguir el robot.

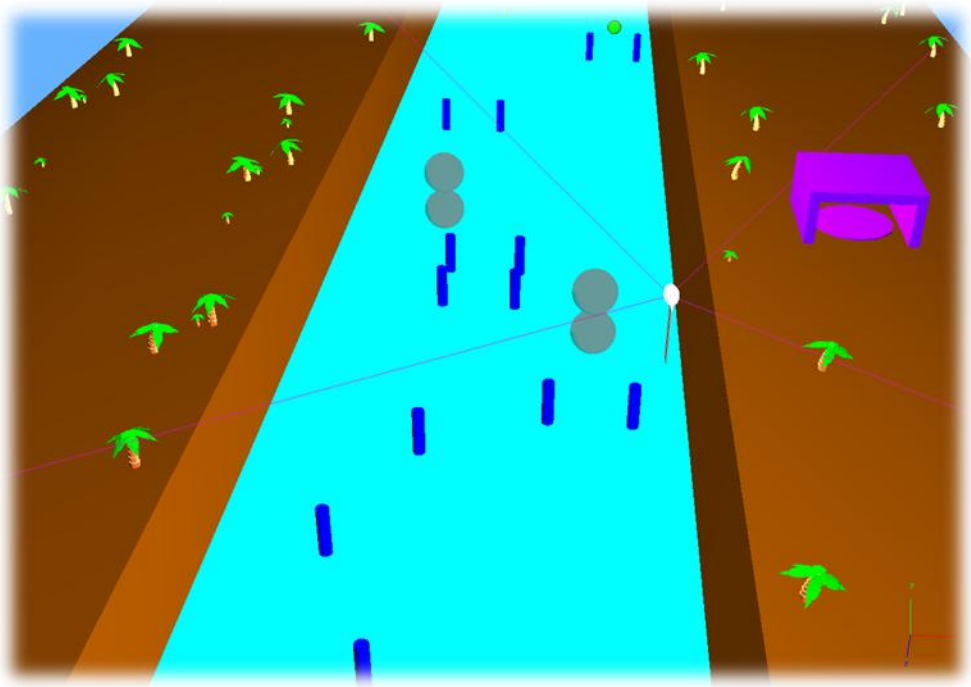

Figura 19: Screenshot de l'execució del robot evitant un obstacle

Ja que la gestió de les destinacions del globus dirigible es fa mitjançant una pila, quan per a evitar un obstacle es calcula una nova destinació l'única cosa que fa falta és

afegir aquest punt intermedi al topall de la pila. Així a la següent iteració la destinació haurà canviat i es començarà a evitar l'obstacle. Una vegada arribat aquesta nova destinació, per a seguir el circuit es procedirà igual que amb la resta de destinacions, bastarà amb eliminar-lo de la pila. El càlcul del punt intermedi de no col·lisió pot ocasionar-nos 2 problemes importants.

D'una banda es troba la qüestió que desplaçar el punt de màxima penetració de manera que quede en la superfície de la biesfera pot no ser suficient per a evitar la col·lisió si tenim en compte la inèrcia que pot dur el robot. Per açò, poden aparèixer més punts intermedis dels desitjats al llarg de les diferents iteracions que realitza el controlador. La manera de solucionar açò passa per dues operacions. Podem afegir un percentatge de confiança a la distància que separem el punt intermedi. I també inserirem el punt intermedi en la pila únicament si es troba a una distància lògica de l'anterior punt intermedi inserit en la pila. D'aquesta manera es minimitzaran els casos en els quals apareixen massa punts intermedis per a evitar un obstacle.

El segon problema que se'ns presenta és que si l'obstacle es troba molt pròxim a la superfície, és possible que depenent d'on es troben el globus dirigible i la destinació, la millor forma d'evitar l'obstacle siga per sota del mateix. En altre cas aquesta maniobra estaria permesa, però tractant-se que l'obstacle es trobe pròxim a la superfície de l'aigua no ha de ser permesa aquest tipus de maniobra. Doncs el globus no hauria de submergir-se ni xocar en cap moment. Per a evitar-lo s'afig un límit inferior a l'altitud, i en cas que el punt intermedi es trobe per sota, es calcula el punt oposat perquè el robot passada per sobre del límit establit.

Òbviament tot aquest procediment es realitzarà cada iteració sobre les dades rebudes de la part de detecció d'obstacles. I d'aquesta manera, triant primer quin obstacle és el primer a evitar, es calcula la posició que s'ha d'arribar a per a no col·lisionar amb ell. Una vegada arribada aquesta posició intermèdia, el robot recupera la trajectòria cap a la destinació original gràcies a la gestió de les destinacions en forma de pila.

4- Manual d'usuari

4.1- Requisits

Per a la realització d'aquest projecte s'ha utilitzat un ordinador de sobretaula AMD Athlon 64 processor 3400+ a 2,4 GHz i amb 512 MB de memòria RAM. EL sistema operatiu ha estat Microsoft Windows XP Professional v. 2002 amb el ServicePack 3. Per a executar la simulació és indispensable el programari de Cyberbotics Webots 5.1.0. En defecte d'això, serà vàlida una versió compatible.

4.2- Modificacions de l'entorn

Totes les modificacions que tinguen a veure amb elements de l'entorn, van a ser realitzades sobre l'apartat visual de Webots. No obstant això, aquelles que afecten a la tasca del robot es faran sobre el codi del controlador. Açò vol dir que si per exemple volem afegir un element decoratiu no hauríem de tocar per res el codi del controlador. No obstant això, si l'element que volem afegir és un nou obstacle, hauríem d'afegir-lo tant a l'entorn visual com al codi del controlador. En aquest apartat s'explicarà el cas de modificar l'entorn visual sense afectar al codi del controlador.

Les finestres de Webots que anem a necessitar són la finestra principal del programa on veurem el resultat i la finestra que conté el Scene Tree on canviarem les característiques del món VRML.

4.2.1- Afegir un element

Ja que el món virtual està creat utilitzant VRML, podríem editar el codi en qualsevol editor de text sempre que respectàrem l'estructura d'aquest llenguatge. Però com en realitat és una adaptació d'aquest llenguatge el recomanable és utilitzar l'editor de Webots. Aquest editor ens donarà les opcions acceptades en cada cas i l'única cosa que hauríem de fer és anar completant els camps.

En la finestra tenim un botó que ens permet crear nodes nous triant-los d'una llista de nodes permesos.

Figura 20: Botó Insert after

Amb aquest botó anem a crear els nodes principals i cadascun que creem ens apareixerà en el Scene Tree en el lloc que nosaltres hàgem triat. Si punxem sobre el nou node, podrem donar-li un nom a aqueix mateix node, el que en VRML és un DEF. A l'esquerra del node tenim el símbol + o – que ens permetrà expandir o contraure els camps. Si expandim el node, podrem punxar sobre cadascun dels camps que ho componen i en la part dreta de la finestra completar els valors dels quals desitgem.

Acompanyats d'un cercle roig en lloc de blau veurem que van tots aquells nodes susceptibles de contenir altres nodes (a excepció de l'atribut CHILDREN, que apareix en blau però pot contenir altres nodes). En canvi en blau només apareixeran els atributs dels nodes que contenen valors.

Posem-nos en el cas que vulguem col·locar un nou element decoratiu en el món virtual. Per exemple una esfera que represente el Sol. El node que anem a necessitar va a ser el node SOLID, així podrem aplicar-li propietats com la física. Llavors farem clic sobre el botó "Insert after" i triarem el node SOLID. Apareixerà sota node que

tinguérem seleccionat en aqueix moment. I apareixerà comprimit, així que farem clic sobre el + per a expandir els seus atributs.

El primer que farem serà definir el node amb el nom SOL i per a això ho escriurem en la part dreta de la finestra quan ho tinguem seleccionat. Ara li donarem una forma, així que anirem a l'atribut CHILDREN i farem clic novament en el botó "Insert after". Triarem el node SHAPE i apareixerà dins de CHILDREN. El programa té un error de refresc i hauríem de seleccionar un node diferent per a veure els canvis. El node SHAPE té dos camps: APPEARENCE i GEOMETRY. El primer ho emplenarem amb un node APPEARENCE. El segon amb un SPHERE. Aquest últim accepta dos paràmetres; el ràdio i el nombre de divisions. Els emplenarem amb 1 i 20. Per l'altre costat dels paràmetres del node APPEARENCE només emplenarem el MATERIAL amb un node MATERIAL. Els atributs diffuseColor, emissiveColor i specularColor seran 1 1 0 per a de aquesta forma tenir una esfera groga.

Tots els elements es creen per defecte en la posició (0, 0, 0). Per això anem a traslladar l'objecte sencer a un lloc millor. Punxarem l'atribut TRANSLATION del sòlid que estem creant i posarem els seus valors a -2, 6, -8.

4.2.2- Modificar i eliminar un element

Com hem vist els nodes contenen atributs. I igual que quan creem un node nou hem d'anar completant aqueixos atributs amb valors, quan els nodes ja estan creats la manera de conducta és la mateixa. Qualsevol valor que vulguem modificar, ho farem desplegant els atributs del node i punxant sobre l'atribut a canviar. En la part dreta de la finestra podrem posar el nou valor i immediatament apareixerà en la finestra principal el canvi realitzat.

Webots permet transformar uns nodes en uns altres amb el botó TRANSFORM, que està situat a l'esquerra del botó "Insert after". Aquest botó el que ens va a permetre és aprofitar els atributs en comú entre el node existent i el node que volem que aquest es transforme. Ens estalviarà temps en cas que necessitem realitzar aquesta operació.

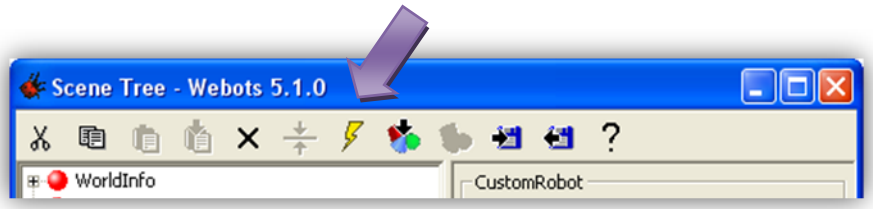

Figura 21: Botó Transform

També podem realitzar les típiques operacions de Copiar, Tallar i Pegar. Però l'interessant és que se'ns permet fer-lo sobre els nodes sencers, inclosos els atributs.

Figura 22: Botons de tallar, copiar i pegar

Finalment, si el que volem és eliminar un node complet només hauríem de seleccionar-lo i estrènyer el botó que conté un aspa negra. Automàticament desapareixerà de l'arbre i de l'escena.

Figura 23: Botó eliminar node

4.3- Modificar el controlador

Necessitarem tenir en el directori de treball les carpetes world i controllers. Dins de la primera hauríem de col·locar l'arxiu "mundo_XVI.wbt". En la segona, serà necessari crear una carpeta amb el nom "c_mundo_XVI" que continga els fitxers "Makefile", "pila.cpp", "punto.cpp" i "c_mundo_XVI.cpp". Encara que aquest pas és recomanable realitzar-lo des del propi Webots amb l'opció de Wizard > New robot controller, ja que d'aquesta forma és més senzill i evita problemes de dolenta estructuració. Amb aquesta estructura d'arxius estarem en condicions d'executar Webots i compilar el codi des de l'editor de codi. Una vegada compilat ens apareixeran en aquesta última carpeta els arxius "c_mundo_XVI.o" i "c_mundo_XVI.exe". Ara ja podrem llançar l'execució de la simulació des de la finestra principal de Webots.

Òbviament, cada vegada que es realitze un canvi en el codi font del controlador, caldrà tornar a compilar el codi. Ho podem fer com s'acaba d'explicar des de la finestra de l'editor de codi. Una vegada compilat el codi amb els canvis, podrem llançar a execució el nou programa tal com s'ha comentat.

4.3.1- Canviar de controlador

Al principi per a utilitzar aquest projecte no és necessària aquesta operació, doncs el món ja ve amb el controlador necessari associat al robot. Però en qualsevol cas, se'ns pot presentar la necessitat de canviar el controlador del robot. Els passos a seguir són els següents. En primer lloc és necessari crear el controlador, com ja s'ha explicat anteriorment en aquesta memòria. Una vegada el controlador ja existeix, només cal expandir els atributs del robot en el Scene Tree i cercar "controller". En la part dreta de la finestra tenim un botó amb tres punts suspensius que ens mostra una finestra on triar un dels controladors permesos per Webots. Si tot és correcte apareixerà el qual cerquem i només amb seleccionar-lo quedarà associat al robot.

4.3.2- Afegir un element

En ocasions no només anem a modificar aspectes visuals de l'entorn, si no que ens va a interessar que l'element que introduïm siga un element amb el qual el robot interactue. Anem a descriure la forma d'afegir els objectes més importants. Un d'ells són les destinacions del robot. Perquè el robot realitze els càlculs necessaris per a avançar d'una destinació a un altre, aquests han de ser emmagatzemats. Per a poder gestionar-los més fàcilment, les destinacions que ha de seguir el zeppelin s'emmagatzemen en una pila. Per a afegir una nova destinació hem d'afegir un nou element a la pila en el lloc que desitgem amb la posició de la destinació a seguir pel robot:

```
stack.push(Punt(posX, posY, posZ));
```

És molt important deixar els 3 últims punts de la pila sense modificar, doncs són els establits per a la maniobra de docking.

Altre tipus d'objecte que ens pot interessar afegir són els obstacles. Els obstacles també queden emmagatzemats en memòria i són emmagatzemats en forma de vector, doncs no importa l'ordre d'emmagatzematge. Si volem afegir un podem afegir la posició

del seu centre al vector d'obstacles tenint en compte que hauríem d'incrementar la constant "NUM_OBS" en una unitat.

```
#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 i eliminar un element

Igual que podem voler afegir elements que no només afecten a l'aspecte visual, si no que també afecten a la part del controlador, també ens pot interessar per qualsevol motiu el modificar els elements ja existents o fins i tot eliminar-los per complet.

La forma de conducta és molt similar a la de l'apartat anterior. Es tracta de localitzar en el codi l'objecte a modificar, i canviar directament sobre el codi els valors que es desitge. Per exemple per a canviar de posició la primera destinació del robot, el que faríem seria anar a la pila on estan tots emmagatzemats i canviar els valors de posX, posY i posZ a l'últim element afegit a la pila. Així la pròxima vegada que compilem el codi, els nous valors ja seran els quals seguisca el zeppelin.

En cas de voler modificar les posicions dels obstacles no hem de canviar el valor de la constant NUM_OBS com fèiem abans. Únicament canviarem els valors de l'obstacle que volem moure. Per a canviar el ràdio de les esferes que emboliquen als objectes del càlcul de col·lisions ho farem directament sobre el camp de l'estructura de l'esfera. Aquest és un dels motius pels quals es plantejarà més endavant una proposta d'ampliació d'aquest projecte.

Vist com modificar els elements, falta saber la manera d'eliminar-los. Gràcies al mètode escollit per a emmagatzemar aquests elements la seua eliminació va a ser tan senzilla com suprimir del codi les posicions de l'element. És a dir, en el cas de voler eliminar una destinació, ho localitzarem en la pila i suprimirem la línia de codi que ho insereix en la pila. D'aquesta forma quan es torne a compilar el codi, aqueixa destinació no existirà per al robot. En el cas dels obstacles és molt similar, doncs estan emmagatzemats en un vector i si suprimim la línia de codi que ho emmagatzema succeirà com en el cas anterior.

4.4- Maneig manual del robot

Com ja s'ha comentat en alguna ocasió al llarg d'aquesta memòria, al robot volador se li han implementat 2 maneres de vol. El primer i més important és la manera automàtica. Aquesta manera com ja s'ha vist és capaç de fer que el globus dirigible seguisca un circuit marcat per diferents destinació i a més evitant els possibles obstacles que puga haver en la seua trajectòria sense perdre aquesta mateixa. No obstant això la manera manual consisteix en la possibilitat de desactivar aquest "pilot automàtic" i manejar amb el teclat tots els motors del globus.

La forma d'activar i desactivar la manera manual és prement la tecla "P". En qualsevol moment de l'execució de la simulació es pot prémer aquesta tecla i canviarà de manera. Per defecte comença la manera automàtica, per tant prémer la tecla "P" significa canviar a la manera manual i tornar a prémer-la activaria de nou la manera automàtica.

Per a manejar els 3 motors del robot s'utilitzarà el teclat com ja s'ha dit. Concretament activarem el motor 1 amb les tecles "↑ / ↓". La primera farà que el globus avanç i la segona ho farà retrocedir. El motor 2 ho controlem amb les fletxes "A / Z", que faran que el robot puge i baixe respectivament. Finalment controlem el gir del robot amb el motor 3 amb l'ajuda de les tecles "← / →", que giren la cua del robot cap a la dreta i cap a l'esquerra. S'ha tingut en compte la visió de l'usuari i prémer la tecla "←" fa que el robot es moga cap a l'esquerra, el que significa que la cua del globus es mou cap a la dreta.

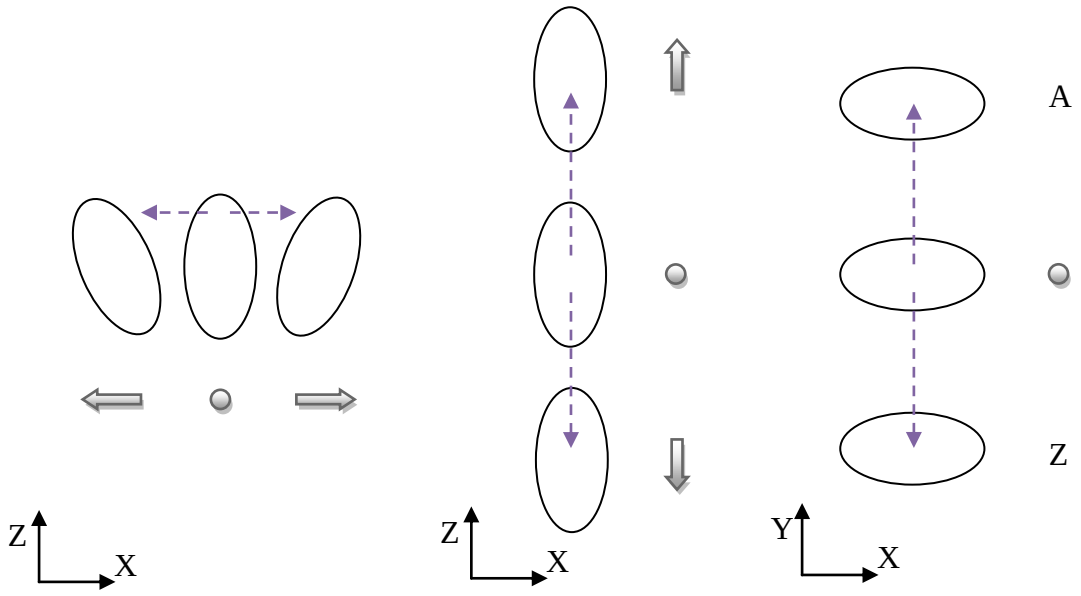

Figura 24: Resum dels moviments que realitza cada tecla

5- Conclusions

L'objectiu principal del projecte era aconseguir crear la simulació d'un robot volador autònom. Com el programa a utilitzar era Webots i el projectant no tenia experiència amb aqueix programari, el primer va anar familiaritzar-se amb ell. Com suport es tenen 2 documents: la guia de referència i el manual d'usuari. El programa també té diversos exemples que es van anar consultant i realitzant progressivament afegint característiques als robots creats.

Finalment quan ja es tenia certa soltesa amb el programa es va començar a crear el món virtual. En aquest cas si que tenia experiència amb VRML i no va suposar una gran dificultat modelar els objectes com era necessari. Per tant era qüestió d'aprendre el funcionament dels nous nodes que incorpora Webots per als robots.

Amb el món creat es va afegir el model d'exemple "blimp_asl2", un globus dirigible de la EPFL. I es va crear un nou controlador incloent parts útils del controlador original. En aquest nou controlador és on es va començar a treballar amb la part de programació d'aquest projecte final de carrera.

El que es pretenia era en una primera parteix controlar el robot de forma manual amb el teclat. Per a això es va estudiar el tipus de controlador més adequat a cadascuna de les circumstàncies i motors del robot. Més endavant quan el robot era manejable es va incorporar el seguiment de destinacions prefixades. És a dir, el robot viatjava de forma correcta i automàticament per un circuit marcat. Finalment es va afegir una part molt important de codi encarregada de detectar i evitar obstacles mentre es seguia el circuit marcat. Per a la realització d'aquesta part ha estat essencial l'estudi de la tesi doctoral d'Enrique Bernabeu Soler, director d'aquest projecte.

5.1- Possibles ampliacions futures

Com en tot projecte existeixen una sèrie de qüestions dignes de ser estudiades per a una possible ampliació. A continuació es parlarà de les quals potser haurien de considerar-se prioritàries, ja siga per la millora que suposen al projecte o bé per la seua utilitat posterior.

Existeixen dades necessàries per a l'execució de la simulació que queden emmagatzemats en memòria per a ser utilitzats pels càlculs interns. Aquestes dades com poden ser les destinacions que ha de seguir el robot o els obstacles que apareixen en l'entorn s'emmagatzemen com part del codi. Açò fa que per a realitzar menuts canvis en les posicions d'aquests elements siga necessari tornar a compilar el controlador. A causa de les limitacions de Webots i del projecte en si, no sembla viable crear una interfície gràfica que permeta a l'usuari la introducció d'aquestes dades, però sí que seria interessant modificar el codi font per a realitzar la lectura d'aquestes dades per fitxer.

Altre aspecte que seria interessant modificar és la grandària de les esferes evolupants per a la detecció i evitació dels obstacles. Aquest ràdio ha estat el mateix durant tot el projecte i per això no s'ha tingut en compte el poder modificar-lo independentment segons l'objecte al que pertanga. Una possible i interessant ampliació seria afegir al vector d'obstacles a més de la posició, el ràdio. I més tard en l'algorisme que fa ús d'ell, utilitzar aquest nou valor emmagatzemat en lloc del fix que s'utilitza actualment.

També hi ha un aspecte menys important que és el del pas de control automàtic a manual i viceversa. En aquests moment el que es fa és controlar el teclat i veure si la tecla "P" és premuda. En aquest cas es canvia de manera amb un flag de control. L'inconvenient és que a causa del temps de mostreig que és de 32 ms, pot succeir que en una pulsació passe un nombre parell d'iteracions. En aqueix cas no es canviaria de manera per la forma que s'ha programat el canvi.

Finalment també es podria estudiar el cas d'obstacles en punts crítics de la trajectòria del robot. Aquests poden ser per exemple obstacles molt propers a la destinació o fins i tot molt propers entre si. Caldria estudiar en quina mesura afectaria a la trajectòria del robot i com es podria solucionar.

Taula d'il·lustracions

Bibliografia

[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>

Apèndix A: Contingut del CD

1. Memoria.docx - *aquest fitxer castellà*
2. Memoria.pdf - *aquest fitxer castellà*
3. Memoria.doc - *aquest fitxer castellà*
4. Memòria valencià.docx - *aquest fitxer*
5. Memòria valencià.pdf - *aquest fitxer*
6. Memòria valencià.doc - *aquest fitxer*
7. documentos - *arxius de consulta imprescindible*
    a. guide.pdf
    b. reference.pdf
    c. Zufferey_Guanella_Beyeler_Floreano_AURO_2006.pdf
8. versiones antiguas - *versions d'aprenentatge*
    a. worlds
    b. controllers
9. version final - *versió final estable*
    a. worlds
    b. controllers
    c. version FINAL.avi
10. archivos - *arxius utilitzats de gran ajuda*
    a. calculos manuales.nb
    b. evitando obstaculo.dwg
    c. físicas.cc
11. Licencia GPLv3.txt - *llicència del projecte*

Apèndix B: Funció de detecció de col·lisions

```
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;
    }
  }
```

Apèndix C: Codi d'evitació de col·lisions

```
   // 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");

         }
```