UNIVERSIDAD CARLOS III DE MADRID

Departamento de Ingeniería de Sistemas y Automática

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

DISEÑO DE UN ENTORNO DE DESARROLLO BASADO EN MODELOS PARA ROBOTS MINI-HUMANOIDES

Autor: Marcos Arjonilla Viñarás

Tutor: Alberto Jardón Huete

Director: Félix Rodríguez Cañadillas

Leganés, Madrid
Septiembre de 2013

Agradecimientos

Agradecerles en primer lugar a mis padres, mi novia, mi hermano y mis amigos. Por haberme aguantado estos últimos meses, en los que he tenido que trabajar duro para conseguir el objetivo que me plantee hace ya varios años. Muchas gracias por vuestra paciencia y apoyo incondicional, os quiero.

En segundo lugar a Félix por el tiempo y esfuerzo que ha invertido en ayudarme. A mi tutor Alberto, y a las demás personas de la Asociación de Robótica que me han ayudado, Juan y Franklin. Muchas gracias sin vosotros no hubiese sido posible este desarrollo, que espero resulte muy útil dentro de la Asociación.

Por ultimo a mis compañeros, con los que tan buenos momentos he compartido durante estos años en la Universidad. Por lo mucho que me han ayudado y animado a conseguir este objetivo. Nunca os olvidare espero que sigamos manteniendo nuestra amistad y que nos acompañe la suerte en esta nueva etapa que comienza.

Muchas gracias a tod@s.

ÍNDICE

1. Introducción. 1

1.1. Introducción a la robótica mini-humanoide. 4

1.2. Descripción general del proyecto. 5

2. Objetivos 6

2.1. Estudio de los entornos de desarrollo actuales 6

2.2. Exploración del software disponible para el desarrollo del entorno 6

2.3. Creación del entorno de desarrollo.. 6

2.4. Creación de un modelo a modo de ejemplo para probar su funcionamiento 6

2.5. Generar el código en el entorno de desarrollo para el robot real....................................... 7

3. Línea de investigación de robots mini-humanoides Uc3m. 8

3.1. Robots mini-humanoides utilizados 9

3.1.1. Plataforma Bioloid Premium.. 9

3.1.1.1. Controlador CM-530 10

3.1.1.2. Servomotores AX-12... 12

3.1.1.3. Mejoras realizadas sobre la plataforma comercial 13

3.1.2. Kit RoboNova1 14

3.1.2.1. Servomotores HSR-8498HB 15

3.1.2.2. Controlador MR-C3024 15

3.2. Concurso CEABOT... 17

3.3. Entorno de pruebas.. 19

3.4. Justificación de este desarrollo dentro de la línea de investigación. 20

4. Estado del Arte. 21

4.1. Estudio de los entornos de desarrollo del mercado 21

4.1.1. Webots.. 22

4.1.2. Microsoft Robotics Developer Studio 4 23

4.1.3. SimRobot... 24

4.1.4. OpenHRP3 24

4.1.5. Marilou Robotics Studio 26

4.2. Modelado basado en componentes.. 27

4.2.1. Fundamentos del modelado de sistemas basado en componentes............................. 27

4.2.2. Concepto de componente 28

4.2.3. Etapas del modelado de sistemas basado en componentes 29
4.2.4. Beneficios e inconvenientes del modelado basado en componentes......................... 30
4.2.5. Ejemplos de diferentes sectores que utilizan modelado basado en componentes 31

5. Software utilizado en el desarrollo.. 33

5.1. Matlab. 33
5.2. Simulink 33
5.2.1. Stateflow 34
5.2.2. Simulink Coder 35
5.2.3. Embedded Coder Support Package for Arduino 36
5.3. MiddleWare de comunicación YARP (Yet Another Robot Platform)............................ 37
5.4. OpenRAVE 39
5.5. Blender 40

6. Desarrollo del proyecto. 43

6.1. Entorno de desarrollo 43
6.1.1. Controlador basado en una Maquina de Estados.. 44
6.1.1.1. Desarrollo teórico de la máquina de estados Mealy para el control del mini-humanoide... 45
6.1.1.2. Implementación de la máquina de estados dentro de StateFlow. 51
6.1.2. Entorno de simulación. 56
6.1.2.1. Creación de escenarios para el entorno de simulación 59
6.1.2.2. Incorporación de las propiedades físicas al entorno de simulación 68
6.1.2.3. Incorporación de sensores al robot Bioloid simulado 71
6.1.3. Middleware de comunicación 74
6.1.3.1. Comunicación YARP - OpenRave - C++ 75
6.1.3.2. Comunicación YARP – OpenRave - Matlab 80
6.1.3.3. Comunicación YARP – OpenRave – Simulink 83
6.1.4. Modelo final del controlador.. 85
6.1.5. Proceso de generación de código para la placa controladora 95
6.1.5.1. Proceso de generación de código C/C++ con Simulink Coder 95
6.1.5.2. Proceso de generación de un ejecutable para Arduino UNO 98

7. Conclusiones y trabajos futuros 102

7.1. Conclusiones del desarrollo 102
7.2. Trabajos futuros 104

8. Referencias 105

ANEXOS 108

Índice de figuras

Figura 1.1. Isaac Asimov durante el proceso creativo 1
Figura 1.2. Evolución histórica de los robots humanoides 2
Figura 1.3. Robot ASIMO 3
Figura 1.4. Logotipo de la Asociación de Robótica 4
Figura 3.1. Logotipo RoboticsLab 8
Figura 3.2. Kit Bioloid Premium 9
Figura 3.3. Placa controladora CM-530 10
Figura 3.4. Ventana de desarrollo RoboPlus Task 11
Figura 3.5. Ventana de desarrollo RoboPlus Motion 11
Figura 3.6. Ventana de desarrollo RoboPlus Manager 11
Figura 3.7. Recorrido angular de los servomotores AX-12 12
Figura 3.8. Conexión y asignación de IDs servomotores AX-12 12
Figura 3.9. Robot Bioloid custom 13
Figura 3.10. Placa controladora CM-900 14
Figura 3.11. Kit RoboNova 1 14
Figura 3.12. Servomotor HSR-8498HB 15
Figura 3.13. Esquema placa controladora MR-C3024 15
Figura 3.14. Entorno de desarrollo RoboBasic para RoboNova 16
Figura 3.15. Logotipo campeonato CEABOT 2013 17
Figura 3.16. Carrera de Obstáculos CEABOT 18
Figura 3.17. Prueba de escaleras CEABOT 18
Figura 3.18. Prueba de Sumo CEABOT 19
Figura 4.1. Entorno de desarrollo Webots 22
Figura 4.2. Entorno de desarrollo Microsoft Robotics Developer Studio 4 23
Figura 4.3. Entorno de desarrollo SimRobot 24
Figura 4.4. Entorno de desarrollo OpenHRP3 25
Figura 4.5. Entorno de desarrollo Marilou Robotics Studio 26
Figura 4.6. Proceso de desarrollo de aplicaciones basadas en componentes 30
Figura 5.1. Entorno de desarrollo Matlab/Simulink 34
Figura 5.2. Placa de pruebas Arduino UNO 36
Figura 5.3. Creación de servidor YARP 38

Figura 5.4. Entorno de simulación OpenRAVE 40
Figura 5.5. Pantalla de inicio de Blender........ 41
Figura 6.1. Esquema del modelado basado en componentes 43
Figura 6.2. Representación de la Máquina de Estados de Mealy 50
Figura 6.3. Bloque de Máquina de Estados de Mealy en StateFlow 51
Figura 6.4. Configuración de la Máquina de Estados en StateFlow 52
Figura 6.5. Configuración Entradas/Salidas de la Máquina de Estados en StateFlow ... 53
Figura 6.6. Máquina de Estados implementada en StateFlow........ 55
Figura 6.7. Configuración del Solver dentro de StateFlow 56
Figura 6.8. Extracto de código del archivo bioloid_robot.xml........ 57
Figura 6.9. Archivo bioloid_robot.xml ejetudo en OpenRave 58
Figura 6.10. Extracto del código Bioloid_Arm_Right.xml 58
Figura 6.11. Código para incorporar el robot al entorno de simulación final 59
Figura 6.12. Normativa para el escenario de la competición CEABOT 60
Figura 6.13. Modelo del suelo creado en Blender........ 61
Figura 6.14. Modelo de las paredes creado en Blender........ 62
Figura 6.15. Modelo de un obstáculo creado en Blender 63
Figura 6.16. Diseño de los 6 escenarios de ejemplo de la competición CEABOT 64
Figura 6.17. Preferencia a seleccionar para habilitar la exportación como .wrl........ 64
Figura 6.18. Archivo CEABOT.xml 65
Figura 6.19. Referencia a la textura por defecto en el archivo .wrl........ 66
Figura 6.20. Modo correcto de referenciar texturas en el archivo .wrl 66
Figura 6.21. Ejecución del archivo CEABOT.xml dentro de OpenRave........ 66
Figura 6.22. Extracto del archivo bioloid.env.xml, referencia al escenario 67
Figura 6.23. Extracto del archivo bioloid.xml donde se referencia el escenario........ 67
Figura 6.24. Entorno de simulación ejecutando el archivo bioloid.xml final 67
Figura 6.25. Entorno de simulación sin propiedades fisicas 68
Figura 6.26. Declaración de cuerpos estáticos y dinámicos dentro de OpenRave 69
Figura 6.27. Definicion de propiedades fisicas dentro del archivo bioloid.env.xml...... 70
Figura 6.28. Simulador incluyendo las leyes fisicas 70
Figura 6.29. Robot con sensores Laser de 3D 71
Figura 6.30. Declaración de sensores Laser 3D 72
Figura 6.31. Robot con sensores Laser de 2D 72

Figura 6.32. Definición de los tres sensores Laser de 2D ... 73
Figura 6.33. Simulador final con los sensores integrados y funcionando ... 74
Figura 6.34. Creación de servidor YARP ... 75
Figura 6.35. Proyecto ASIBOT de la Asociación de Robótica ... 76
Figura 6.36. Distribución de las IDs de los servomotores en el modelo de Bioloid ... 76
Figura 6.37. Inicialización del CartesianServer para 23 motores ... 77
Figura 6.38. Puertos creados para cada uno de los sensores ... 77
Figura 6.39. Apertura del simulador a través del CartesianServer ... 78
Figura 6.40. Ejemplo testRemoteRaveBot modificado para Bioloid ... 78
Figura 6.41. Test de movimientos desde el ejemplo creado en C++ ... 79
Figura 6.42. Conexión entre puerto readsensors y puerto del sensor frontal ... 79
Figura 6.43. Lectura de los sensores en el puerto readsensors ... 80
Figura 6.44. Código .m para conectarse con el remote_controlboard ... 80
Figura 6.45. Consola de Matlab mostrando librerías cargadas y robot disponible ... 81
Figura 6.46. Método utilizado en Matlab para mover los servomotores ... 81
Figura 6.47. Test de movimientos creado en Matlab. ... 81
Figura 6.48. Importación de puertos y botellas YARP ... 82
Figura 6.49. Código .m para la lectura del sensor frontal ... 82
Figura 6.50. Lectura de los sensores en Matlab ... 83
Figura 6.51. Consola de comandos Linux-Matlab ... 83
Figura 6.52. Bloques y cabeceras de las Matlab-functions utilizadas ... 84
Figura 6.53. Valores de los sensores a medida que se acercan a un obstáculo ... 85
Figura 6.54. Distribución de las IDs de los motores en RoboPlus. ... 86
Figura 6.55. Asignación de IDs de nuestro modelo y el de RoboPlus ... 86
Figura 6.56. Obtención del ángulo objetivo ... 87
Figura 6.57. Obtención de la velocidad angular ... 87
Figura 6.58. Modelo final de Simulink ... 90
Figura 6.59. Bloque openyarp dentro del modelo final ... 91
Figura 6.60. Bloque readlaserfront dentro del modelo final ... 91
Figura 6.61. Conexiones de la máquina de estados con el resto del modelo ... 92
Figura 6.62. Procesos dentro de la máquina de estados funcionando ... 93
Figura 6.63. Bloque sendmovements dentro del modelo final ... 94
Figura 6.64. Bloque de parada de la simulación dentro del modelo final ... 94

Figura 6.65. Máquina de estados dentro del archivo stateflowtrans.mdl 95
Figura 6.66. Configurar Solver para generación de código.. 96
Figura 6.67. Elección de código C/C++ para su generación 96
Figura 6.68. Proceso de generación desde la consola de comandos de Matlab 97
Figura 6.69. Carpeta con los archivos generados 97
Figura 6.70. Archivos staflowtrans.h y .c generados 98
Figura 6.71. Librería para Arduino UNO de Simulink 98
Figura 6.72. Desarrollo HIL con Arduino y Simulink 99
Figura 6.73. Configuración del puerto serie en Matlab.. 100
Figura 6.74. Conexión de la máquina de estados con Arduino UNO 100
Figura 6.75. Configuración de los parámetros para crear el ejecutable 101

Índice de Tablas

Tabla 6.1. Comparativa Máquina de estados de Moore y Mealy 44
Tabla 6.2. Entradas de la Máquina de Estados de Mealy 46
Tabla 6.3. Salidas de la Máquina de estados de Mealy 47
Tabla 6.4. Estados de la Máquina de estados de Mealy 47
Tabla 6.5. Calculo de la Máquina de estados de Mealy 48
Tabla 6.6. Tabla de transiciones del Estado 1 49
Tabla 6.7. Tabla de transiciones del Estado 2 49
Tabla 6.8. Tabla de transiciones del Estado 3 50
Tabla 6.9. Tabla de transiciones del Estado 4 50
Tabla 6.10. Secuencia para andar hacia delante 88
Tabla 6.11. Secuencia para andar hacia la izquierda 88
Tabla 6.12. Secuencia para andar hacia la derecha 89
Tabla 6.13. Datos importados a Matlab desde Excel en formato .csv 89

1. Introducción

Actualmente el mundo de la robótica se encuentra en auge, siendo la ciencia donde la tecnología desarrolla todo su potencial al tratarse de un campo en continua evolución basado en la investigación, donde existen múltiples posibilidades y variantes. En estos momentos todavía queda mucho por hacer y mejorar ya que se trata de una tecnología en continuo desarrollo, pero durante los últimos años se han producido grandes mejoras gracias al trabajo de muchos investigadores.

La robótica combina múltiples disciplinas tecnológicas como son: la electrónica, la informática, la inteligencia artificial, la mecánica, la ingeniería de control y la física entre otras. El término robot fue acuñado por primera vez en la obra RUR (Robots Universales Rossum), escrita por Karel Capek en 1920, basándose en la palabra checa "Robota", que significa trabajos forzados. De ahí nace su funcionalidad principal que es la de hacernos la vida más fácil.

En aquel momento la robótica era simplemente ciencia ficción, pero muchos de los puntos descritos en aquel inicio han ido poco a poco convirtiéndose en realidad. Dentro de la ciencia ficción se definieron tres leyes que han continuado vigentes hasta nuestros días, siendo las máximas que todo robot ha de cumplir. Dichas leyes, son un conjunto de normas escritas por Isaac Asimov, y aparecieron por primera vez en el relato Runaround (1942) [1].

Figura 1.1. Isaac Asimov durante el proceso creativo

Las tres leyes de la robótica establecen lo siguiente:

I. Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.

II. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la 1ª Ley.

III. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1ª o la 2ª Ley.

Una vez introducido el concepto de robótica y sus orígenes, pasaremos a describir la rama de esta ciencia en la que se centra el objeto de este proyecto, que son los robots Humanoides o Androides. Por definición este tipo de robots intentan reproducir total o parcialmente la forma y el comportamiento cinemático del ser humano. Uno de los aspectos más complejos de controlar en estos robots, y en el que se centran la mayoría de estudios dentro de este campo, es el de la locomoción bípeda y cómo coordinarla con un proceso en tiempo real para mantener el equilibrio del robot.

Actualmente, hay en desarrollo bastantes proyectos sobre robótica humanoide, tanto a nivel de investigación dentro de las diferentes universidades y otros centros cuya actividad se basa en I+D+I, como a nivel industrial ya que muchas empresas han comenzado a interesarse por esta rama de la robótica que sin duda representa actualmente un mercado emergente. En la figura 1.2. se muestra la evolución que han sufrido este tipo de robots durante los últimos años.

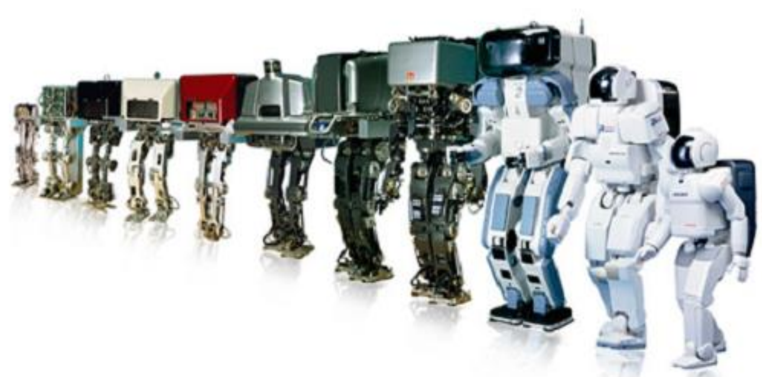


Figura 1.2. Evolución histórica de los robots humanoides

Dentro de este campo, el robot más destacado, es el ASIMO (Advanced Step in Innovative Mobility) el mostrado en la figura 1.3. y fue creado por la empresa Honda en el año 2000 y que representa de manera notable su gran inversión en I+D+I. ASIMO es el resultado del compromiso adquirido por la empresa en los años 80, cuando comenzó a trabajar en este campo y creó su primer robot bípedo. Desde entonces, la empresa ha invertido una gran cantidad de dinero representando una de las mayores inversiones en este campo, motivo por el que su aporte ha sido muy notable realizando grandes avances. ASIMO es considerado el humanoide más avanzado del mundo, siendo un referente para toda la industria.

Su última versión fue presentada en noviembre de 2011, se trata de un prototipo totalmente renovado el cual integra una nueva tecnología de control del comportamiento autónomo, con una inteligencia mejorada y una gran capacidad física para adaptarse a diferentes situaciones. Es por tanto un sistema muy complejo, que incorpora gran cantidad de sensores y actuadores, el conjunto ha de ser controlado en base a diferentes comportamientos programados, según los estímulos recibidos [2].

Figura 1.3. Robot ASIMO

Tras el ejemplo mostrado anteriormente, cabe destacar que actualmente los robots humanoides son cada vez más evolucionados, lo que supone un aumento en la complejidad de controlar todos los sistemas que en ellos se integran, para ayudar a percibir e interactuar con el entorno de manera autónoma. Lo que ha provocado que la dificultad en los algoritmos de control se incremente de forma drástica. Además a medida que la ciencia evoluciona, se tiende a que cada vez el comportamiento de este tipo de robots sea más parecido al comportamiento humano, aumentando su dificultad, debido a la gran complejidad que supone imitar nuestro comportamiento y forma de interactuar con el entorno que nos rodea.

1.1. Introducción a la robótica mini-humanoide

Dentro de la Asociación de Robótica de la Universidad Carlos III cuyo logo se puede observar en la figura 1.4., existe un gran interés en los sistemas robóticos basados en morfologías de tipo humanoide, encontrándose actualmente muchos proyectos en desarrollo. Concretamente este proyecto se ha desarrollado dentro del Grupo de Robótica de Mini Humanoides, este grupo de trabajo surgió dentro de la Asociación de Robótica de la Universidad Carlos III de Madrid, durante el año 2006, con el objetivo de que los alumnos tuvieran la posibilidad de acercarse a esta rama de la robótica durante su formación universitaria. Otro de los objetivos que se marcó este proyecto en sus inicios, fue el de participar en concursos de robótica mini-humanoide, de forma que los alumnos pudieran comprobar cuan óptimos eran sus desarrollos y a su vez compararlos con otros desarrollos realizados en el mismo ámbito, facilitando el aprendizaje por parte del equipo y acelerando la evolución del proyecto.

Figura 1.4. Logotipo de la Asociación de Robótica

Dicho proyecto con el tiempo se ha convertido en la puerta de acceso al ámbito de los robots mini-humanoides para todos aquellos alumnos interesados. Permitiendo continuar con su formación de una manera practica en los diferentes ámbitos que este proyecto engloba como son, la Instrumentación Electrónica, los Sistemas de Control Inteligente, la Ingeniería del Software y la Dinámica de Sistemas Mecánicos entre otros. El proyecto ha ido incorporando año a año nuevas mejoras y miembros al equipo, demostrando un gran compromiso y evolución dentro de este campo, lo que supone que la acogida del proyecto ha sido buena entre la comunidad universitaria y por tanto se le augura un gran futuro dentro de la Asociación de Robótica de la Universidad Carlos III de Madrid, así como en las competiciones que se participe [3].

1.2. Descripción general del proyecto

Con este proyecto se pretende crear un entorno de desarrollo para facilitar la definición de los algoritmos de control utilizados en los robots mini-humanoides. Las principales características que este entorno de desarrollo debe cumplir son las de tener un diseño modular que permita la reutilización de los elementos software, la trazabilidad en las pruebas y la generación sencilla de algoritmos.

Con el desarrollo de este proyecto además se pretende, que los algoritmos de control creados para los robots mini-humanoides puedan ser probados en un simulador antes de ser incorporados al procesador real. Con esto se consigue reducir los posibles costes ocasionados por fallos en los algoritmos diseñados, otorgando a los componentes del equipo la posibilidad de realizar pruebas sin necesidad de utilizar el robot real.

Tras lo expuesto anteriormente, se decidió crear el entorno de desarrollo y simulación que tiene como objeto este proyecto, con la vista puesta en dotarle de una gran flexibilidad y robustez de cara a ser modificable, además de intuitivo y fácil de utilizar. El diseño ha de ser reutilizable con independencia de las mejoras realizas en el sistema físico utilizado actualmente o incluso en el caso de que se sustituyera dicho sistema por completo en el futuro.

2. *Objetivos*

Dentro del proyecto se defino como objetivo principal la creación de un entorno de desarrollo para robots mini-humanoides, cuyas características principales fuesen la flexibilidad en los elementos software y trazabilidad en las pruebas con los elementos utilizados, para que en el futuro, se puedan introducir todos los cambios y mejoras que se deseasen dentro del modelo utilizado, o incluso se cambie el modelo por completo. De este objetivo principal derivan los siguientes sub-objetivos que ha habido que ir cumpliendo hasta obtener el entorno de simulación final:

2.1. Estudio de los entornos de desarrollo actuales

Estudiar las diferentes alternativas que el mercado actual ofrece en cuanto a entornos de desarrollo, dichos entornos deberían incluir simuladores para sistemas robóticos. El estudio de estos entornos de desarrollo, nos será de ayuda a la hora de determinar cuál es el entorno, o el diseño de entorno de desarrollo que más se ajusta a la plataforma mini-humanoide utilizada.

2.2. Exploración del software disponible para el desarrollo del entorno

Estudiar las diferentes alternativas para el modelado de escenarios 3D que existen y que tengan las características necesarias para su futura integración en el entorno de simulación elegido. Escoger un software para la creación de modelos de controlador que cumpla con las características de ser modificable, modular y capaz de generar código en C, C++ o tipo Arduino. Por ultimo seleccionar un middleware que capaz de comunicar y transmitir datos entre el entorno de simulación y el software de creación de modelos de control elegidos.

2.3. Creación del entorno de desarrollo

Una vez seleccionados todos los componentes del sistema se debía implementar dicho sistema (entorno de desarrollo), cumpliendo las premisas marcada en cuanto a modularidad y flexibilidad a la hora de ser modificado o mejorado en futuros desarrollos, cumpliendo de esta forma el principal objetivo del proyecto.

2.4. Creación de un modelo a modo de ejemplo para probar su funcionamiento

Crear un sistema de control para el modelo físico introducido en el simulador, para después conectarlo con el entorno de simulación, en el que se encontrará representado un entorno realista. Para la conexión entre ambos módulos, se hará uso de un middleware de comunicación encargado del envío y recepción de datos, asegurando el sincronismo y buen funcionamiento del sistema completo.

2.5. Generar el código en el entorno de desarrollo para el robot real

Una vez probada la funcionalidad para la que había sido diseñado el sistema, será necesario descargar todo el código de control generado y probado en el entorno de simulación, a la placa controladora definitiva que se instalará de manera física dentro del robot real.

3. *Línea de investigación de robots mini-humanoides Uc3m*

En el seno de la Asociación de Robótica de la Universidad Carlos III de Madrid, se encuentra actualmente en activo y por tanto en continuo desarrollo, el proyecto sobre robótica de mini-humanoides, dicho proyecto representa una buena alternativa para que los alumnos interesados pongan en práctica los conocimientos adquiridos durante su carrera universitaria, en mi caso Grado en Electrónica Industrial y Automática o bien aquellos alumnos que cursen el Máster Universitario en Robótica y Automatización ambos impartidos en la Universidad Carlos III de Madrid.

Para el desarrollo de este proyecto, se utilizan los recursos e instalaciones del grupo de investigación RoboticsLab [4], el cual apoya a la Asociación de Robótica dentro de la Universidad y cuyo logo se puede observar en la figura 3.1. La asociación cuenta actualmente con varias plataformas mini-humanoides, entre las que destacan los kits Bioloid y Robonova. Dichos kits serán analizados en detalle a lo largo de este capítulo, siendo el modelo elegido para el desarrollo de este proyecto el Bioloid, se hará un análisis más detallado, en el que se incluirá el software que hasta este momento se estaba utilizando en su programación.

Figura 3.1. Logotipo RoboticsLab

Los objetivos de este proyecto dentro de la asociación de robótica son claros y muy dirigidos a la investigación, entre ellos destacan los siguientes campos relacionados con la robótica humanoide: las diferentes técnicas de control de estabilidad que permitan a los robots mini-humanoides caminar de forma bípeda con un alto grado de seguridad en sus movimientos, la integración de diferentes tipos de sensores a bordo del robot para detectar y analizar el entorno, la investigación de algoritmos que permitan al robot interactuar con el entorno de una manera totalmente autónoma. Con el desarrollo de este proyecto dentro de la Asociación de robótica, se pretende que todas las líneas de investigación en las que se está trabajando, sean puestas a prueba dentro del entorno de simulación y desarrollo creado.

3.1. Robots mini-humanoides utilizados

Dentro del proyecto para mini-humanoides de la Asociación de Robótica de la universidad Carlos III se dispone de varios kits comerciales de mini-humanoides, en concreto se dispone de tres kits Bioloid Premium y de seis kits Robonova1. A continuación se realizará un análisis detallado de estos dos kits en el que se mostrarán sus principales características, siendo más detallado el análisis del kit Bioloid, al ser sobre el que se centrará el desarrollo realizado a lo largo de este proyecto.

3.1.1. Plataforma Bioloid Premium

En nuestro caso el kit elegido ha sido el Bioloid Premium [5] ya que por cuestiones de presupuesto, dentro de su gama era el más adecuado siguiendo sus especificaciones. Por otro lado al tratarse de un robot en el que se van a realizar múltiples modificaciones tanto a nivel hardware como software no se consideró necesario adquirir un kit superior, ya que su precio es más elevado y aunque a priori podía parecer más apropiado, el kit Premium posee la flexibilidad necesaria a la hora de integrar nuevos sensores y actuadores y la robustez suficiente a la hora de encarar las diferentes pruebas a realizar.

Figura 3.2. Kit Bioloid Premium

El kit Premium es adecuado para construir robots avanzados de hasta 18 grados de libertad como los son los robots mini-humanoides. Se trata de un kit adecuado para aprendizaje, hobby, investigación y/o competición. El kit está compuesto por bloques constructivos que el usuario puede unir con tornillos a su gusto, otorgando la posibilidad de crear diferentes robots no solo mini-humanoides sino robots que imitan diferentes animales, robots que se mueven sobre ruedas, etc. Todos los componentes montados formando un mini-humanoide como el mostrado en la figura 3.2., tiene una altura de unos 34 cm incluyendo la cabeza y un peso de 1,9 Kg.

La plataforma robótica se basas en una tecnología inteligente servocontrolada en serie que permite la retroalimentación, además del control sensorial de la posición, velocidad, temperatura, corriente y tensión de cada uno de sus servomotores. Gracias a sus actuadores Dynamixel AX-12A controlados a través de una placa CM-530 programable con su software propietario RoboPlus. Los componentes mencionados anteriormente serán analizados en detalle a continuación.

3.1.1.1. Controlador CM-530

La placa CM-530 mostrada en la figura 3.3., es un dispositivo controlador para los servomotores AX-12A y al que además se le pueden integrar sensores como por ejemplo los sensores de proximidad AX-S1. Incluye conexión para poder descargar un programa desde el PC que controle los movimientos del robot. Además se pueden usar sus botones como dispositivos de entrada de ordenes o bien utilizar dispositivos emisores/receptores para control remoto.

Figura 3.3. Placa controladora CM-530

La comunicación con el PC se realiza conectando el “serial cable connecting jack” del CM-530 al puerto serie del PC usando un cable serie que permita la comunicación entre el PC y el CM-530. En caso de utilizar un ordenador que no disponga de puerto serie, existe la posibilidad de usar uno de los puertos USB con el adaptador USB2Dynamixel.

Para programar la CM-530 se nos ofrece la posibilidad de utilizar el software RoboPlus. Es un software gratuito pensado únicamente para Windows característica que restringe sus posibilidades para usuarios de Linux, o iOS. El software RoboPlus, nos permite programar cada una de las diferentes variantes físicas que se pueden realizar con cualquiera de los kits ofrecidos por Robotis, pero solo para ellos, no se puede utilizar en robots de otras marcas como por ejemplo el robot mini-humanoide RoboNova del que también se disponen en la asociación.

Para programar el robot Bioloid se utilizan tres de los múltiples recursos que ofrece el software RoboPlus. Los tres recursos que hemos de utilizar se analizarán en detalle a continuación:

RoboPlus Task: este programa es el que utilizaremos para desarrollar el algoritmo de control que marcara su comportamiento, normalmente basaremos este comportamiento en los estímulos recibidos (sensores) y produciremos la respuesta deseada (actuadores). Todo este control del comportamiento deseado se almacenará en un fichero con extensión .tsk.

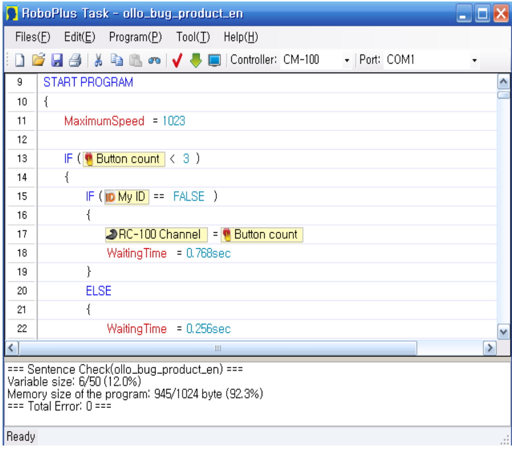


Figura 3.4. Ventana de desarrollo RoboPlus Task

RoboPlus Motion: Este programa sirve para programar el movimiento de nuestro robot. Otorgando los valores indicados a los 18 grados de libertad (servomotores) de los que el robot está dotado. Es el encargado de mover los motores y sincronizarlos. Se programa el movimiento del robot paso a paso, se mueve el robot a una posición de inicio, se graba esta posición y luego se mueve a una segunda posición y se vuelve a grabar. El interpolado entre las posiciones, se realiza automáticamente moviéndose suavemente de una posición a la siguiente, estos movimientos son almacenados en un fichero con extensión .mtn.

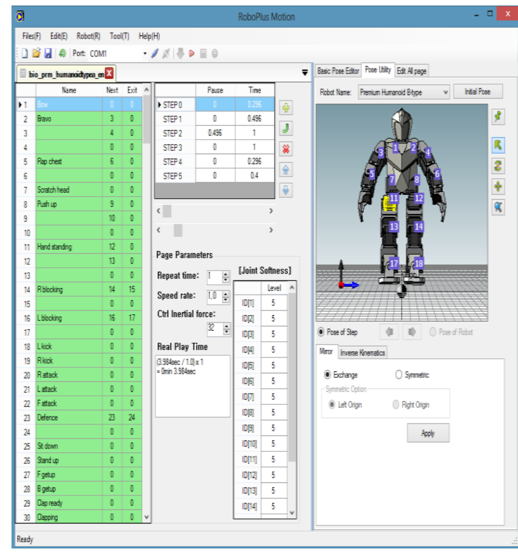


Figura 3.5. Ventana de desarrollo RoboPlus Motion

RoboPlus Manager: Este programa se utiliza para configurar los motores, los sensores y además para actualizar el firmware de la placa controladora. Dichas acciones se tienen que realizar de forma periódica, para prevenir que se desconfigure el sistema por completo teniendo que comenzar de nuevo por el principio. Por ejemplo se utiliza para asignar las IDs a cada uno de los servomotores y la velocidad a la que se transmitirán los datos.

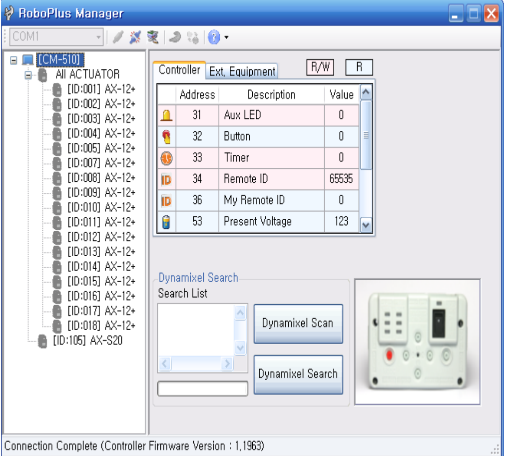


Figura 3.6. Ventana de desarrollo RoboPlus Manager

3.1.1.2. Servomotores AX-12

El AX-12A es un servomotor utilizado por Robotics, que actúa como las articulaciones del robot. Se puede controlar su posición en intervalos de 0.29º gracias a sus encoders digitales de 1024 pulsos por lo que se tendrá un total de unos 300º, aunque en ocasiones debido a las limitaciones de la estructura este ángulo puede verse reducido. También hay que tener en cuenta como se muestra en la figura 3.7., que existen unos valores de ángulos no permitidos por la diferencia entre los 300º máximos a controlar y los 360º que caracterizan una vuelta completa.

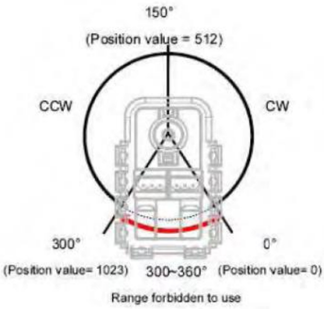


Figura 3.7. Recorrido angular de los servomotores AX-12

Adicionalmente este tipo de servomotores tienen las funciones de monitorizar tanto la temperatura como la carga, o la posición en la que se encuentran. A cada uno de los servomotores se les ha de asignar un ID diferente para distinguirlos de manera inequívoca cuando hay múltiples conectados al controlador CM-530. EL usuario puede cambiar el ID de los servomotores a su gusto teniendo en cuenta que ha de ser un número del 0 al 33 al ser estos los valores permitidos. En la figura 3.8. se puede observar cómo se conectan los AX-12A a la placa controladora CM-530 en serie, a través de dos cables de alimentación y uno de control que será el encargado de transmitir las ordenes.

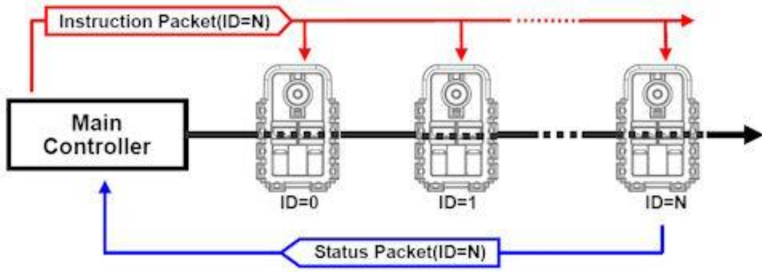


Figura 3.8. Conexión y asignación de IDs servomotores AX-12

3.1.1.3. Mejoras realizadas sobre la plataforma comercial

Tras todas las características mostradas anteriormente, cabe destacar que nuestro robot basado en la plataforma comercial Bioloid Premium Kit, se aleja mucho de esta tras todas las modificaciones y mejoras realizadas. Gracias a estas mejoras, se le ha dotado de unas capacidades muy superiores a la que tenía de serie, lo que le permitirá realizar con mayor capacidad las pruebas, a las que será sometido durante la competición CEABOT entre otras.

Las mejoras realizadas sobre el robot son varias y de muy diversa índole como se puede observar en la figura 3.9., a continuación se mostrarán algunas de ellas aunque actualmente se encuentras muchas otras en desarrollo, por lo que el robot final será casi completamente nuevo a excepción de los componentes básicos como son el chasis y los servomotores.

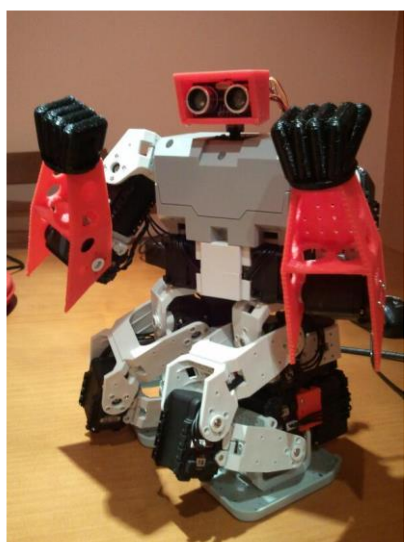


Figura 3.9. Robot Bioloid custom

Hasta hace algún tiempo se utilizaba una placa controladora CM-530 del fabricante Robotics, dicha controladora ha sido sustituida por una CM-900 [6], ya que con la antigua se reducían mucho las posibilidades de integrar nuevos sensores o actuadores, tenía una menor capacidad computacional y además no soportaba algunos lenguajes de programación de alto nivel. Por contra la placa actual soporta la programación en un lenguaje de más alto nivel utilizado por Arduino, además de permitir su programación en otros lenguajes como C y C++. Esta nueva placa se caracteriza por una mayor capacidad computacional y una gran flexibilidad a la hora de incorporar nuevos sensores y actuadores con nuevos puertos y buses de comunicación como se muestra en figura 3.10.

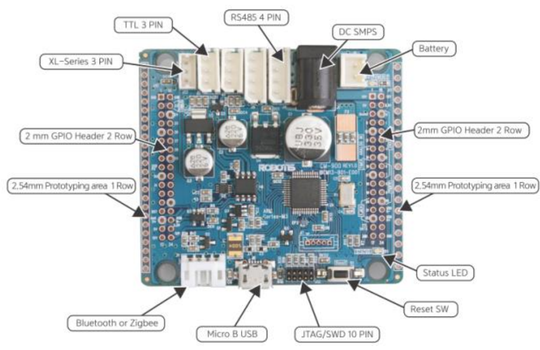


Figura 3.10. Placa controladora CM-900

La nueva placa incorporada en el robot Bioloid, se caracteriza por integrar un microprocesador ARM Cortex-M3 (32 bits), en concreto se trata de un STM32F103C8, con 8 timers, 10 entradas analógicas que incluyen ADC, una memoria flash de 64 KB, una memoria SRAM de 20 KB y un reloj interno con una frecuencia máxima de 72 MHz. Además de todas las características comentadas incluye muchos más puertos de comunicación de muy diversa índole, lo que nos permitirá la incorporación de nuevos sensores y actuadores, además de una mejora sustancial en las posibilidades de comunicación con el robot.

3.1.2. Kit RoboNova1

El robot mini-humanoide está construido tomando como base el kit comercial Robonova-1[7] mostrado en la figura 3.11. Con este kit se incluyen todas las piezas de aluminio anodizado y plástico rígido necesarias para construir el robot mini-humanoide. La estructura se compone de manera análoga al Kit Bioloid de cabeza, tronco, dos extremidades superiores y dos extremidades inferiores, imitando la apariencia de un ser humano. El robot una vez montado se caracteriza por tener una altura de unos 30,5 cm y un peso de 1,3 Kg.

Figura 3.11. Kit RoboNova 1

3.1.2.1. Servomotores HSR-8498HB

En el kit se incluyen 16 servomotores digitales HSR-8498HB de Hitec como los mostrados en la figura 3.12, que se encargaran de proporcionan el movimiento rotacional a las articulaciones representadas por sus 16 grados de libertad. Además, estos servos tienen un rango de giro de 180º y un par de fuerza superior a los 7,4 Kg/cm que son trasmitidos por engranajes de carbonita, más resistentes y duraderos que los de nailon.

Figura 3.12. Servomotor HSR-8498HB

Los servos HSR-8498HB han sido diseñados específicamente para este robot e incluyen una análogamente a los incluidos en el kit Bioloid “Feedback Motion” o movimiento por realimentación, permitiendo leer desde el controlador la posición real en la que se encuentra el servomotor, además incluye una opción por la que podemos colocar el robot manualmente en cualquier posición, para luego leerla y guardar la situación en la que se encuentran los 16 servos dentro de un programa.

3.1.2.2. Controlador MR-C3024

El kit incorpora como sistema integrado de control la placa controladora MRC-3024 mostrada en la figura 3.13. gobernada por un micro-controlador Atmel ATMega 128, y que cuenta con elevado número de puertos para controlar hasta 24 dispositivos como servomotores, además gracias a sus más de 40 puertas de entrada/salida digitales permite incorporar un gran número de sensores como sensores de distancia, giróscopos, displays LCD, sensores de infrarrojos, etc.

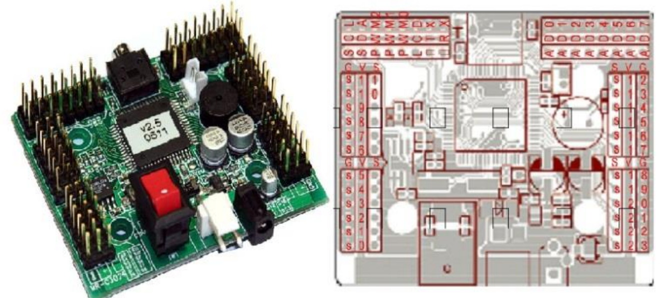


Figura 3.13. Esquema placa controladora MR-C3024

Además la placa cuenta con un altavoz capaz de generar tonos a diferentes frecuencias y un conector al que se le puede conectar un diodo LED. También incluye 64 Kbyte de memoria flash para almacenar los programas, que permiten que una vez que se han descargado, Además, la tarjeta es capaz de gestionar comunicaciones inalámbricas como por ejemplo señales IR.

El software necesario para programar el robot viene incluido en el kit Robonova-1, además alternativamente es ofrecido libremente por el fabricante en sus páginas Web. El software del robot Robonova está basado en el lenguaje de programación de robots RoboBasic mostrado en la figura 3.14. Se trata de lenguaje básico, pero muy especializado y orientado a la programación de robots. Este lenguaje incluye gran cantidad de comandos específicos para controlar las funciones del robot, por lo que facilita y simplifican el proceso de programar el control de mini-humanoide.

El kit Robonova-1 incluye un total de tres entornos para la programación del humanoide: RoboBasic, RoboRemocon y RoboScript. Estos entornos para la programación son altamente intuitivos y complementarios entre sí. Aunque el entorno de programación RoboBasic, es la herramienta principal para la programación, ya que incluye opciones para poder compilar y descargar los programas creados al robot, además de otras herramientas para ajustar y configurar los servomotores y otros componentes del robot.

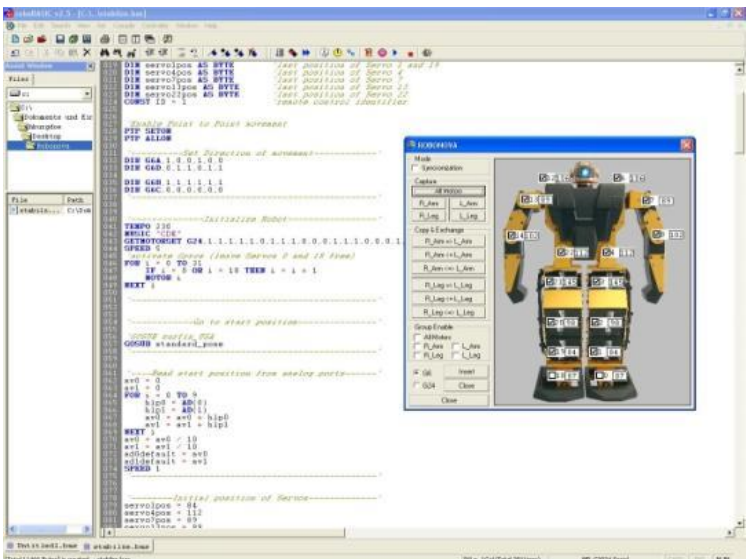


Figura 3.14. Entorno de desarrollo RoboBasic para RoboNova

3.2. Concurso CEABOT

Como se analizó en la introducción a este proyecto, otro de los objetivos en los que se centra la línea de investigación, es la participación en campeonatos de robótica mini-humanoide. Dentro de esta línea cabe destacar la competición CEABOT, cuyo logo se puede observar en la figura 3.15. La participación en este campeonato fomenta el aprendizaje comparar y compartir conocimientos con otros alumnos e investigadores del campo de la robótica humanoide.

La competición CEABOT es organizada por el Comité Español de Automática de manera anual, sus orígenes datan de manera análoga al proyecto de mini-humanoides del año 2006. El equipo de la Universidad Carlos III ha asistido a dicha competición desde sus inicios, completando buenas actuaciones, hasta que gracias al gran compromiso y evolución demostrando dentro de este campo, en el año 2012 se alzaron con el Subcampeonato de España en CEABOT, recompensando el gran trabajo realizado desde sus inicios.

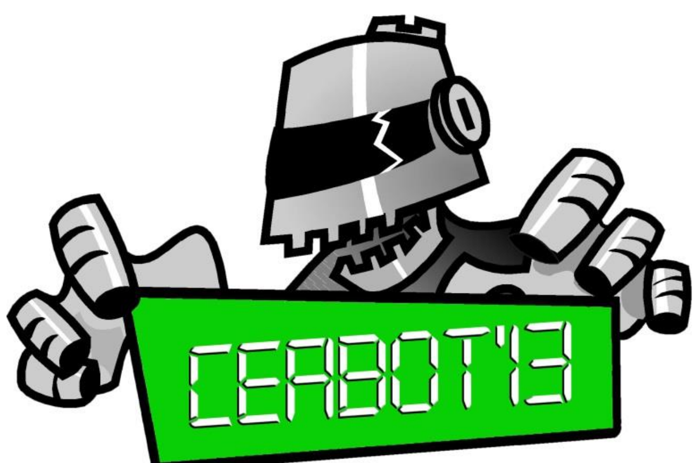


Figura 3.15. Logotipo campeonato CEABOT 2013

A continuación, se analizarán cada una las pruebas de las que consta el campeonato CEABOT 2013 [8], toda la información ampliada sobre estas pruebas y sobre las bases del concurso se puede encontrar dentro del Anexo 1. El desarrollo de este proyecto, se ha basado en la prueba "carrera de obstáculos" aunque se pretende que funcione como un entorno de desarrollo general, en el que una vez creado no resulte difícil introducir las demás pruebas.

Prueba 1: Carrera de Obstáculos

Dentro de esta prueba los robots deben ir desde un extremo del campo al otro, y regresar al punto de partida. El robot tiene que andar esquivando los obstáculos, sin tirarlos y sin variar su posición.

Figura 3.16. Carrera de Obstáculos CEABOT

Los robots saldrán desde la zona central, dentro de la zona de la zona de salida, como se observa en la figura 3.16. Han de cruzar todo el campo donde se sitúan los obstáculos, hasta la zona de llegada parcial y una vez allí, darse la vuelta de forma autónoma para realizar el mismo proceso de vuelta. Se puntuara en función del tiempo transcurrido, y el número de penalizaciones.

Prueba 2: Escalera

Dentro de esta prueba, se modificará el escenario anterior, del que serán retirados los obstáculos, para añadir una escalera como la mostrada en la figura 3.17.

Figura 3.17. Prueba de escaleras CEABOT

Los robots deberán alcanzar la zona de llegada, superando todos los escalones de subida y de bajada caminando, no está permitido ningún tipo de salto o acrobacia para facilitar la subida. La escalera solo se recorrerá en ambos sentidos, puntuándose la habilidad de cada robot para superar la escalera de forma autónoma, sin caerse o desviarse. Además también se tendrá en cuenta el tiempo empleado.

Prueba 3: Lucha de Sumo

En esta prueba se enfrentarán cara a cara dos robots de dos equipos diferentes, se realizara dentro de un nuevo escenario como el mostrado en la figura 3.18., y que incluye el área de combate con forma circular.

Figura 3.18. Prueba de Sumo CEABOT

Para obtener puntos, llamados puntos Yuhkoh. Se valora el comportamiento competitivo del robot, pudiéndose penalizar actitudes pasivas o inmóviles con el fin de que ambos robot desplieguen todas sus habilidades durante el combate. El combate se dividirá en asaltos de 2 minutos.

3.3. Entorno de pruebas

Dentro de las instalaciones asignadas a la Asociación de Robótica, se dispone de un entorno de pruebas que representa el entorno real de la competición CEABOT. En este entorno de pruebas se dispone de un escenario similar al de la “Carrera de obstáculos”. Además dentro de este mismo escenario se encuentra dibujado un círculo central, para realizar ensayos de la prueba “Lucha de Sumo”, y por ultimo también se dispone de un escenario que reproduce la prueba de “Subida de escaleras”.

3.4. Justificación de este desarrollo dentro de la línea de investigación

Tras el estudio detallado del estado en que se encuentra actualmente el proyecto de mini-humanoides, dentro de la Asociación de Robótica de la Universidad Carlos III de Madrid, en la que se disponía de todas las herramientas necesarias para realizar un buen papel dentro de la competición CEABOT. Se decidió implementar un entorno de desarrollo de algoritmos de alto nivel, que incorporase un simulador. De esta manera se podrían desarrollar y probar nuevos algoritmos de una manera sencilla.

Se consideró necesario desarrollar este entorno ya que sin duda ayudaría mucho al desarrollo de nuevos modelos de control, y lo más importante por la versatilidad que aporta al poderse instalarse en cualquier equipo con Linux, facilitando que más alumnos puedan trabajar en paralelo sin necesidad de acudir al laboratorio y sin que tengan que disponer del sistema físico para probar sus nuevos desarrollos.

Por último, gracias a este desarrollo, se facilitará la toma de decisiones de inversión en hardware para el futuro, ya que antes de realizar dicha inversión se podrá simular el funcionamiento del robot real una vez incluido, determinando si la inversión será necesaria al mejorar las capacidades o no.

4. Estado del Arte

Este capítulo se dividirá en dos vertientes principales, en primer lugar se analizará la situación actual de los entornos de desarrollo que existen en el mercado para desarrollar algoritmos de control de robots, utilizados para evaluar el funcionamiento del robot previamente a ser usado en el robot real, evitando de esta manera fallos, o desajustes que puedan causar daños en el robot permitiendo ahorrar costes en el proceso de diseño.

En segundo lugar se analizará detalladamente, el estado actual de la técnica de diseño y desarrollo “modelado basado en componentes”, que otorgan una mayor modularidad y flexibilidad a los sistemas implementados. Dicho modelo de desarrollo se encuentran actualmente en auge en muchas y muy variadas industrias, sirviendo de inspiración para el desarrollo de este proyecto.

4.1.Estudio de los entornos de desarrollo del mercado

En la actualidad se dispone de una gran variedad de entornos de simulación y desarrollo que permiten realizar simulaciones de robots en ambiente tridimensionales, poniendo a nuestra disposición el uso de sensores y actuadores con los que compondremos nuestro robot y podremos interactuar con el entorno en el que realizaremos la simulación. Cabe destacar que los entornos en los que se realizará la simulación debe ser creados por el usuario, además de representar de una manera realista el entorno real donde finalmente actuará el robot, dichos entornos han de estar caracterizados por las diferentes fuerzas físicas que encontramos en el mundo real como la gravedad.

Los simuladores que encontramos actualmente en el mercado utilizan en su gran mayoría interfaces gráficas para poder construir nuestros robots y poder modelar el entorno con el que interactuarán. Los simuladores que encontramos actualmente en el mercado utilizan en su gran mayoría interfaces gráficas para poder construir nuestros robots y poder modelar el entorno con el que interactuarán. Otra característica muy importante de los entornos de simulación es el lenguajes de programación que soportan para la creación de controladores, normalmente utilizan leguajes como C, C++, PYTON o Java.

A continuación se presentarán diferentes simuladores comerciales que podemos encontrar actualmente en el mercado, aunque dentro del mercado existe un gran número de ellos se analizarán solo algunos, ya que son los que se han considerado más relevantes dentro de este campo en la actualidad.

4.1.1. Webots

Se trata de un simulador que además viene acompañado de un paquete completo de software profesional para modelar, programar y simular robots móviles entre los que se incluyen los robots mini-humanoides como se muestra en la figura 4.1. Con este programa, el usuario es capaz de diseñar tanto el sistema físico de los diferentes robots, como un entorno virtual en tres dimensiones, eligiendo entre las múltiples opciones de las diferentes propiedades para cada objeto, como son la forma, el color, la textura, y todas sus propiedades físicas como masa, fricción, etc.

Además de todas las propiedades mostradas anteriormente, se ofrecen varias opciones para equipar los robots ya que se incluyen librerías con un amplio número de sensores y actuadores, tales como cámaras, servomotores, ruedas motrices, sensores de distancia, sensores de contacto y fuerza, emisores y receptores de señales de radiofrecuencia, etc.

En cuanto a los controladores de los robots se pueden programar tanto en el propio simulador como en un entorno de desarrollo externo, siendo soportados múltiples lenguajes como C, C++ o Java entre otros. Una vez programado el control pude ser testeado su comportamiento en los diferentes mundos creados incluyendo unas leyes físicas muy realistas, para más tarde incorporar dichos programas a los controladores finales dentro de los robots físicos reales [9].

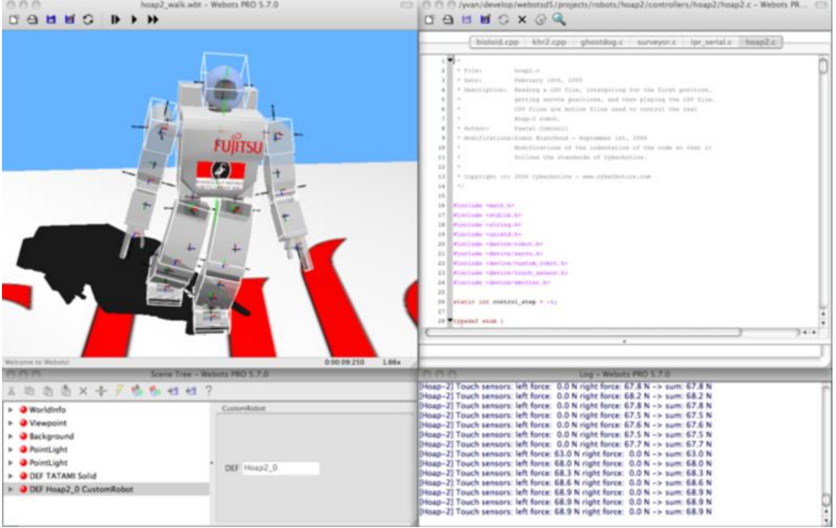


Figura 4.1. Entorno de desarrollo Webots

4.1.2. Microsoft Robotics Developer Studio 4

Este simulador fue desarrollado por Microsoft [10], y se basa en la metodología de programación .NET, se trata de un entorno de desarrollo altamente escalable diseñado para la creación, programación y simulación de diferentes plataformas robóticas entre las que se incluye la creación de mini-humanoides. Entre sus diferentes entornos cabe destacar los siguientes: Descentralización de Servicios de Software (DSS), Lenguaje de Programación Visual (VPL), Concurrencia en Tiempo de ejecución y Coordinación (CCR) y por ultimo su Entorno de simulación visual (VSE)

El simulador es multi-plataforma por lo que permite la programación bajo varios lenguajes como son Visual Basic .NET, Visual C#, JScript, IronPython y también otros lenguajes proporcionados por terceros siempre que se adecuen a su arquitectura de servicios.

Gracias a este simulador el desarrollador puede acceder fácilmente a los diferentes sensores y actuadores de sus robots, ya que el simulador le proporciona una librería de implementación concurrente basada en el lenguaje .NET. Otra característica importante de este simulador es la de permitir la comunicación entre módulos. A continuación se muestra la figura 4.2., en la que se puede observar la interfaz de usuario con la que el diseñador interactuará en el desarrollo de la plataforma.

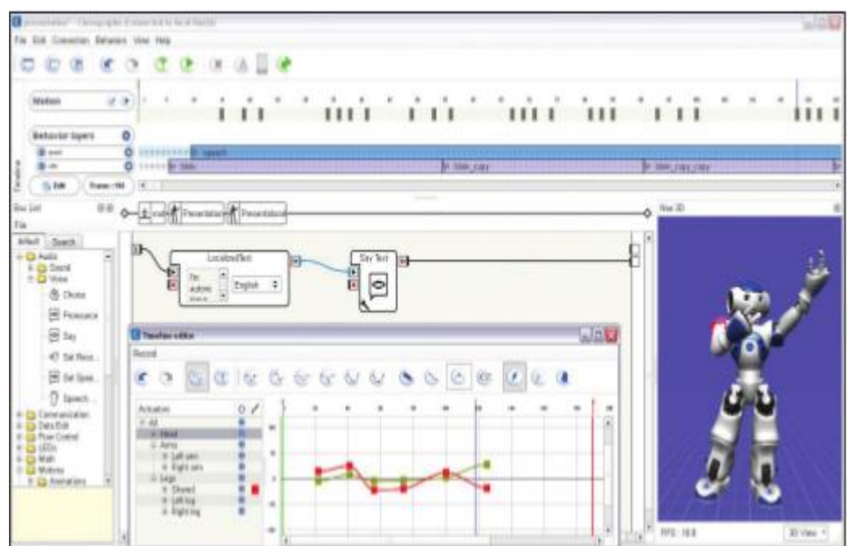


Figura 4.2. Entorno de desarrollo Microsoft Robotics Developer Studio 4

4.1.3. SimRobot

La principal característica de este simulador es la de que se defino para poder soportar todo tipo de robots móviles entre los que se incluyen los robots mini-humanoides que son sobre los que se centra el desarrollo de este proyecto. El lenguaje sobre el que se basan todos los modelos utilizados dentro de este simulador es el lenguaje XML, con los que los usuarios son capaces de especificar cualquier tipo de robot y también se pueden crear sus propios ambientes o escenarios sin la necesidad de utilizar otros lenguajes de programación [11].

Dentro de los modelos XML se pueden incluir todas las partes del cuerpo del robot, además de múltiples actuadores y múltiples sensores permitiendo de esta manera que el usuario tenga mucha libertad a la hora de desarrollar un robot. El motor utilizado para simular la dinámica de los cuerpos o solidos rígidos es Open Dynamics Engine (ODE), mientras que para la visualización de las imágenes dentro del simulador se basada en el motor OpenGL. Dentro de la figura 4.3., se puede observar el entorno de simulación que incluye los dos motores anteriormente mencionados.

Este simulador fue desarrollado por la Universidad de Bremen y el Centro de investigación de inteligencia artificial alemán. Actualmente se está utilizando mucho por toda Europa para la investigación en el campo de los robots autónomos.

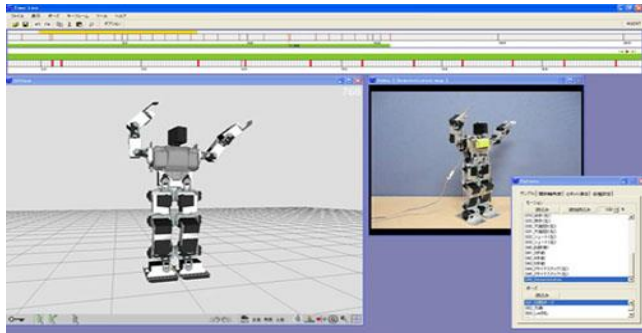


Figura 4.3. Entorno de desarrollo SimRobot

4.1.4. OpenHRP3

Open Architecture Humanoid Robotics Plataform version 3 [12], se trata de un simulador para robots y en el que se puede desarrollar software, permitiendo a los usuarios inspeccionar y modificar el modelo original del robot con el que se trabaja además de programar el bucle de control pudiendo testearlo a través de una simulación dinámica. Conjuntamente con OpenHRP3 se proporcionan diversas bibliotecas de cálculo que pueden ser utilizadas en el desarrollo de software para robótica.

Este simulador se encuadra dentro de un programa de investigación y desarrollo impulsado por el Ministerio de Economía e Industria de Japón, además forma parte, del proyecto denominado "Distributed component type robot simulator", que se está llevando a cabo por la asociación "Cooperation of Next Generation Robots". En su desarrollo también colaboran entre otros la Universidad de Tokyo y el instituto japonés "National Institute of Advanced Indrustrial Science and Technology (AIST)".

Los modelos del robot, además de los diferentes entornos se almacenan y distribuyen en formato VRML. Dentro del entorno existen otras funciones para la implementación de drivers de control dentro del "Controller", funciones que soportan la dinámica de los diferentes mecanismos y también una función de visualización para el proceso de simulación. Los dos módulos fundamentales para el desarrollador son el de planificación del movimientos "MotionPlanner", en el que se generan trayectorias evitando colisiones y el de generación de patrones para los diferentes pasos "PatternGenerator" en el que se generan movimientos estables para la locomoción del robot basándose en el control del ZMP.

El principal inconveniente que muestra este simulador es que su desarrollo se encuentra muy centrado en su propio robot "HRP3", por lo que resulta complicada su utilización para plataformas diferentes. A continuación, se muestra la figura 4.4. en la que se puede observar la interfaz de usuario que caracteriza a este simulador y donde además se observa el humanoide "HRP3", dentro de un escenario con una estructura tubular.

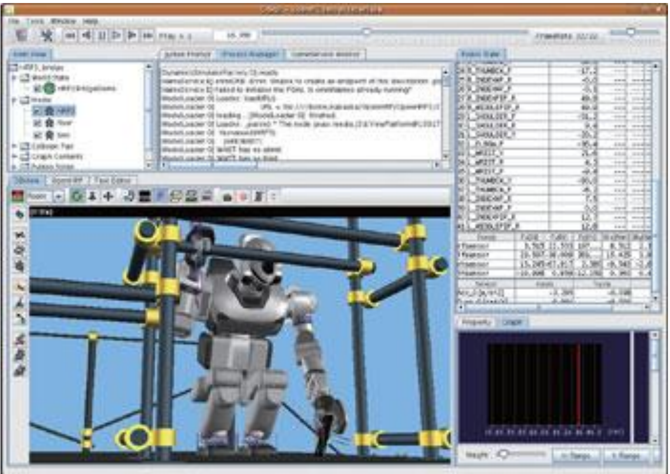


Figura 4.4. Entorno de desarrollo OpenHRP3

4.1.5. Marilou Robotics Studio

Este entorno de simulación, permite de manera análoga al resto de simuladores analizados, modelar las diferentes piezas de las que se compondrá el modelo del robot definitivo. Además permite crear ambiente o escenarios de simulación en 3D, para que los robots móviles, mini-humanoides, brazos articulados y robots paralelos puedan funcionar en condiciones reales. Como se puede observar en la figura 4.5., incluye las leyes físicas que experimentaran dentro del mundo real tras ser implementados físicamente [13].

Es por tanto un ambiente gráfico muy realista, también permite crear la jerarquía necesaria para que al construir los diferentes elementos como pueden ser cajas, esferas, cilindros, entre otros elementos geométricos se puedan ensamblar entre ellos para posteriormente probar dichos ensamblajes de forma simple, también permite la implementación de formas más complejas como son triángulo de malla, geometrías y formas convexas, etc. Cabe destacar las diferentes variantes que el usuario puede escoger a la hora de realizar una simulación ya que esta puede realizarse en tiempo real o bien un tipo de simulación acelerada en la que el tiempo puede verse reducido consiguiendo reducir el tiempo de las pruebas.

Este simulador soporta la programación en diferentes lenguajes como son C, C++ y C#. Tanto en el sistema operativo Windows como en Linux, lo que lo dota de una gran flexibilidad. Dentro de su última versión además de los lenguajes de programación indicados anteriormente se pueden utilizar otros como Visual Bacic.Net, incorporando además otras características de compatibilidad con entornos como Matlab, Java o Intempora RT-Mapas, por lo que sin duda representa una buena alternativa dentro del mundo de los simuladores que habría que tener en cuenta si en el futuro se pretendiera cambiar de simulador.

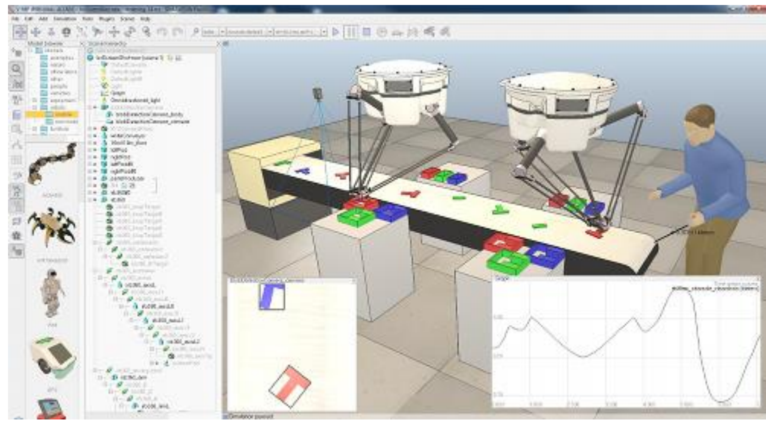


Figura 4.5. Entorno de desarrollo Marilou Robotics Studio

4.2. *Modelado basado en componentes*

Actualmente este tipo de diseño es utilizado en la mayoría de sectores industriales y en el desarrollo de software entre otros, esto se debe a las grandes ventajas que presenta frente a otro tipo de procesos de modelado y diseño. Esto no quiere decir que no presente inconvenientes que provoquen que en determinadas situaciones no sea aplicable.

Hoy en día la complejidad y el tamaño de las diferentes aplicaciones crece rápidamente siendo demandados productos de muy alta calidad, que implican una gran complejidad en su desarrollo, y siempre en el menor tiempo posible. Para ser capaces de cumplir con las expectativas es imprescindible desarrollar aplicaciones, en las que se asegure su reutilización, para su futura integración en nuevos productos. Este es el objetivo que persigue el modelado basado en componentes. Además gracias a este modelo de desarrollo se logra reducir costes, tiempo y esfuerzos de desarrollo del software, a la vez que ayuda a mejorar la fiabilidad, flexibilidad.

A lo largo de este apartado se realizará un estudio en detalle de esta metodología de desarrollo, comenzando por la definición de que es un componente, para posteriormente abordar la metodología de desarrollo y diseño de aplicación, así como sus características fundamentales que tiene que cumplir un modelo de componentes y por último sus ventajas e inconvenientes. Además se analizarán a modo de ejemplo algunos de los procesos en los que se ha optado por este tipo de implementación.

4.2.1. *Fundamentos del modelado de sistemas basado en componentes*

Los profesionales del mundo de la ingeniería del software, utilizan actualmente la forma en que otras ingenierías llevan resolviendo sus problemas desde hace casi un siglo, como pronostico Chambers [14], se estaba convirtiendo en una necesidad el importar a este ámbito el concepto de componente como bloque constructivo. Estos componentes software son diseñados de tal forma que se puedan utilizar en multitud de aplicaciones sin tener que ser modificados, permitiendo la creación de nuevos componentes y aplicaciones a partir de otros preexistentes.

Sin embargo, el diseño software sigue siendo una actividad muy complicada, ya que es muy difícil que las nuevas aplicaciones se construyan solamente a base del ensamblaje de componentes prexistentes. La naturaleza del software es muy distinta a la de otras ingenierías de producción, al encontrarse siempre sujeta a las características concretas del hardware en el que se utilizara de forma definitiva, por lo que aplicaciones óptimas para determinados sistemas, pueden no serlo para otros, llegando incluso a no poder ser utilizadas en estos. De este hecho derivan las tres perspectivas a analizar que se describen dentro del siguiente apartado, en el que además se estudiará el concepto de componente en sí mismo.

4.2.2. Concepto de componente

La definición de un componente software es la siguiente: "Un componente es una unidad binaria de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio."

Tras la definición mostrada en el párrafo anterior, se ha de analizar el concepto desde tres perspectivas diferentes, que lo definirán de una manera completa. Estas tres perspectivas fueron definidas por Brown [15], y son totalmente necesarias para poder comparar los distintos modelos de componentes:

- **Perspectiva de empaquetamiento:** el componente es considerado como una unidad de empaquetamiento y distribución. Este concepto se encarga de la organización, ya que se centra en la identificación de un conjunto de elementos que pueden ser reutilizados como una unidad. Enfatiza por tanto en la reutilización. Normalmente un componente empaquetado, es aquél que se distribuye en formato binario aunque no tiene por qué tratarse de un único fichero binario.
- **Perspectiva de servicio:** el componente es considerado como una entidad que ofrece sus servicios a los posibles clientes. Para el diseño y la implementación de aplicaciones se ha de comprender de qué manera se van a intercambiar las peticiones entre los diferentes componentes que colaboran dentro del servicio solicitado. Los servicios se agrupan en interfaces, que especifican los servicios ofrecidos por cada componente, representando la única manera de acceder a dichos servicios, representan por tanto, una visión lógica del componente.

- **Perspectiva de integridad**: esta perspectiva se centra en identificar los límites de cada componente, detectando sus limitaciones y la posibilidad de sustituirlo por otro. Define a los componentes como cápsulas de implementación, que encierran todo el software que mantiene la integridad de los datos manipulados, de manera independiente a la implementación de otros componentes. De forma que todo el componente puede ser reemplazado como una única unidad.

Por último es importante definir su tamaño, dado que los componentes también son unidades de sustitución, su granularidad ha de adaptarse a su propósito y al rol que va a desempeñar en la aplicación, maximizando su uso y su reutilización. Además su funcionalidad ha de estar bien definida, entre las que destacan los componentes de interfaz gráfica, componentes de entrada/salida, componentes de control, componentes de coordinación, componentes de análisis de datos o los componentes de almacenamiento.

4.2.3. Etapas del modelado de sistemas basado en componentes

Tradicionalmente dentro de la ingeniería del software, se ha seguido un enfoque descendente (top-down) para el desarrollo de sistemas. Sin embargo, el desarrollo basado en componentes se basa en un enfoque ascendente (bottom-up), en el que los productos finales se implementan mediante el ensamblaje y la integración de componentes software preexistentes. Existen dos formas básicas para realizar un diseño basado en componentes, bien utilizando componentes que han sido desarrollados por terceros, con lo que únicamente se han de seleccionar y ensamblar correctamente, o bien desarrollar de manera modular los componentes que formaran la aplicación. En función del enfoque escogido, el ciclo de desarrollo puede variar ligeramente. Normalmente, independientemente de la opción escogida como se muestra en la figura 4.6., el ciclo que se ha de seguir para el desarrollo de este tipo de aplicaciones es el definido como COTS por Carney [16] cuyas directrices se muestran a continuación:

- Selección y evaluación de los componentes software, que deberán satisfacer las necesidades del usuario y ajustarse al esquema de diseño de la aplicación.
- Adaptación de los componentes cuando sea necesario
- Ensamblaje de los componentes como parte de la solución final.
- Evolución del sistema si el usuario lo requiere.

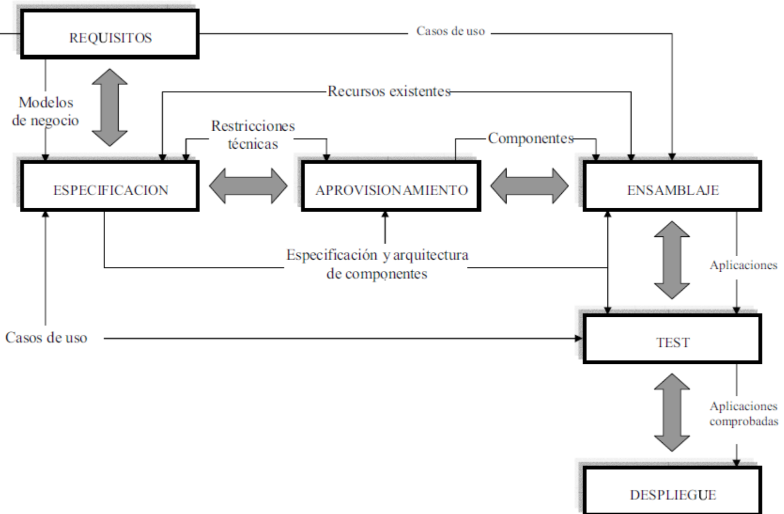


Figura 4.6. Proceso de desarrollo de aplicaciones basadas en componentes

4.2.4. Beneficios e inconvenientes del modelado basado en componentes

Entre sus principales **beneficios** cabe destacar los descritos a continuación:

- **Funcionalidad mejorada:** las aplicaciones diseñadas siguiendo este modelo, se caracterizan por su alta calidad, al estar sujetas a un proceso de mejora continua, resultando sencillo sustituir un componente por otro durante todo su ciclo de vida.
- **Reutilización del software:** los componentes que se utilizan para implementar una determinada aplicación, pueden ser reutilizados en la implementación de otras aplicaciones, realizando pequeños cambios de adaptación.
- **Simplifica las pruebas:** el tiempo de pruebas y verificación se ve ampliamente reducido, al componer las aplicaciones con componentes ampliamente probados previamente.
- **Simplifica el mantenimiento del sistema:** el mantenimiento es mucho más reducido que en aplicaciones diseñadas con otros sistemas, al tratarse de sistemas mucho más modulares que facilitan la detección y aislamiento de errores, pudiendo reparar el componente afectado o sustituirlo por uno nuevo de manera sencilla.
- **Mayor calidad:** este beneficio está directamente relacionado con todo lo expuesto en los cuatro puntos anteriores, que convierte este tipo de aplicaciones en sistemas de muy alta calidad.

- **Ciclos de desarrollo más cortos:** una vez los componentes han sido estandarizados y probados de manera aislada, resulta muy sencilla la implementación de nuevas aplicaciones, por lo que el tiempo de desarrollo se ve reducido de manera notable.
- **Mejor ROI:** el retorno sobre la inversión, es mucho mejor que en otros tipos de desarrollos ya que el ciclo de vida es mucho más elevado, además de la ventaja frente a otros sistemas, al poder reciclar los componentes para la implementación de nuevas aplicaciones**.**

Los **inconvenientes** derivados de este tipo de desarrollos son los siguientes:

- **Genera mucho tiempo de desarrollo inicial:** en el caso de tener que implementar cada uno de los componentes por primera vez, sin poder utilizar componentes ya implementados puede resultar muy costoso, al tratarse de componentes que han de ser muy flexibles, reutilizables y robustos.
- **Genera mucho trabajo adicional:** de los atributos mencionados en el punto anterior, deriva la necesidad de realizar muchas pruebas para asegurar su funcionamiento, además de su compatibilidad con los demás componentes con los que se utilizará en el futuro.

4.2.5. Ejemplos de diferentes sectores que utilizan modelado basado en componentes

En primer lugar se muestra un ejemplo del desarrollo de software basado en componentes, como se muestra en la publicación [17]. La reutilización de componentes software es un proceso inspirado en la manera en que se producen y ensamblan componentes en la ingeniería de sistemas físicos. La aplicación de este concepto al desarrollo de software no es nueva. Las librerías de subrutinas especializadas, comúnmente utilizadas desde la década de los setenta, representan uno de los primeros intentos por reutilizar software. Aunque actualmente es un modelo muy extendido dentro de esta industria.

Dentro de la industria de la automatización industrial, también se encuentra en auge el desarrollo basado en componentes como se muestra en la publicación [18]. Al igual que en otras ingenierías, actualmente se trata de aplicar la tecnología de componentes a fin de reducir los costos y plazos de desarrollo, aunque en las aplicaciones de tiempo real su aplicación presenta problemas que no están aún resueltos. La tecnología de componentes en sistemas de tiempo real está siendo requerida por la industria de automatización, de potencia y equipamientos eléctricos,

aviación, automoción, etc. Empresas líderes en estos sectores como son ABB o Boeing hacen grandes inversiones en su desarrollo, obteniendo resultados prometedores en cuanto al mantenimiento y gestión de la evolución de sus productos, pero basándose hasta ahora en soluciones propias que no han llegado a imponerse con carácter general.

Además de la industria de la automatización industrial, otras industrias dedicadas al control de procesos también están incorporando este tipo de desarrollos dentro de sus nuevos proyectos como se muestra en la publicación [19]. La reducción en los precios de los microprocesadores, entre otros avances tecnológicos, ha permitido realizar grandes avances en el desarrollo de software comercial (herramientas "Comercial Off The Shelf", COTS), así como en el equipamiento de control. Las prestaciones y la calidad que ofrecen los controladores lógicos programables (PLCs) han mejorado de forma espectacular. También se han producido grandes mejoras en el campo de la instrumentación como, aumentando la oferta de sensores y actuadores inteligentes.

Tras los ejemplos expuestos anteriormente, aunque existen muchos otros que no han sido expuestos, cabe destacar que este tipo de diseño se está introduciendo en la mayoría de sectores industriales, por sus grandes beneficios frente a modelos de desarrollo clásicos. Tras este estudio, se consideró muy interesante trasladar este modelo al mundo de la robótica, en concreto al desarrollo que dio lugar a este proyecto, dentro del ámbito de los entornos de desarrollo y simulación de robots mini-humanoides, y aunque a priori su integración resulta complicada, facilitara en gran medida los desarrollos futuros dentro de este ámbito.

5. Software utilizado en el desarrollo

Dentro de este capítulo, se describirá todo el software utilizado a lo largo del proyecto, para el diseño de cada uno de los componentes que formarán el entorno de desarrollo.

5.1.Matlab.

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es una herramienta de la empresa Mathworks [20] de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Mac OS X.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y también con otros dispositivos hardware. Es un software usado en universidades y centros de investigación y desarrollo, así como en la industria.

La versión utilizada en el desarrollo de este proyecto, es la versión R2012a, dentro del acuerdo de licencias para la docencia, que pone a disposición de los estudiantes la Universidad Carlos III de Madrid.

5.2.Simulink

Se decidió utilizar Simulink, para el diseño del controlador principal ya que en él se pueden incorporar modelos de sistemas físicos reales que serán posteriormente fáciles de modificar y ampliar cumpliendo con los objetivos marcados al inicio del proyecto.

Simulink [21] es un entorno de programación visual, que funciona sobre el entorno de programación Matlab. Se trata de un entorno de programación de más alto nivel de abstracción que el lenguaje interpretado Matlab. En Simulink se genera archivos con extensión .mdl que deriva de la palabra modelo.

Se trata por tanto de una herramienta de simulación de modelos o sistemas, con cierto grado de abstracción de los fenómenos físicos involucrados en los mismos como por ejemplo dentro del campo de la Ingeniería Electrónica para el procesamiento digital de señales (DSP), también es un software muy utilizado en Ingeniería de Control y Robótica siendo el punto en el que destaca sobre el resto de competidores y uno de los puntos por los que ha sido elegido.

Además dentro de Simulink tenemos la opción de incorporar funciones definidas en lenguaje .m creadas en Matlab, como las funciones de comunicación que integraremos dentro de nuestro modelo. Simulink permite la utilización de sus múltiples librerías a la hora de crear modelos como se muestra en la figura 5.1, donde se puede observar el entorno de desarrollo de Matlab y de Simulink conjuntamente.

Dichas librerías nos facilitaran el trabajo a la hora de crear la máquina de estados que representará el bucle central de nuestro modelo y que se realizará gracias a la librería de Stateflow. También se hará uso de otras librerías para una vez probado el modelo del controlador en el simulador conectarlo con el controlador real y descargar dicho código generado para que funcione de forma autónoma en el robot real. Dichas librerías serán analizadas con más detalle a continuación y son las siguientes: Simulink Coder, y dentro de esta ultima la librería Embedded Coder Support Package for Arduino.

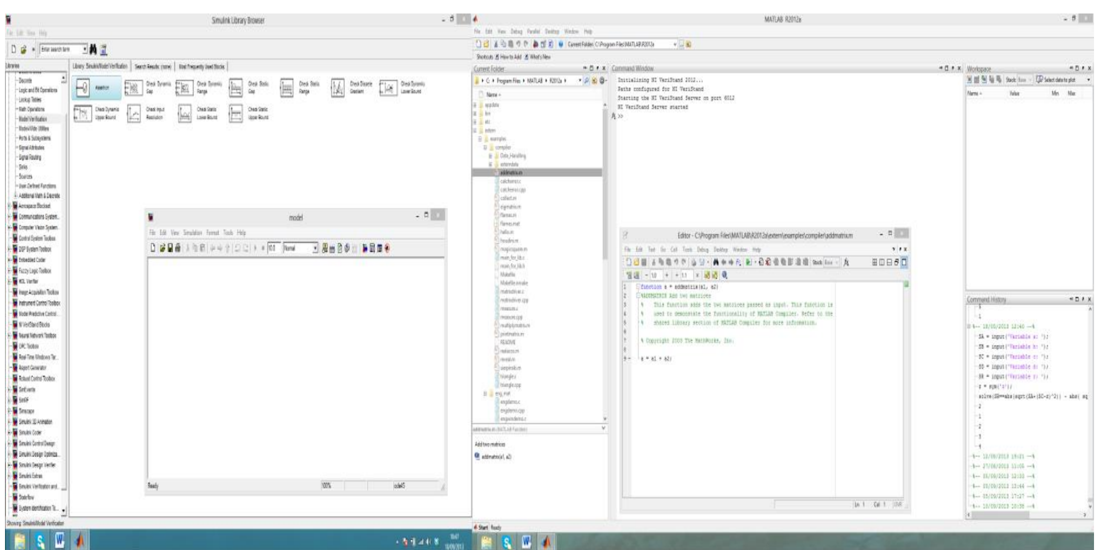


Figura 5.1. Entorno de desarrollo Matlab/Simulink

5.2.1. Stateflow

La librería de Stateflow [22] nos proporciona un entorno para modelar y simular lógica de decisión combinatoria y secuencial basado en máquinas de estado y diagramas de flujo. Permite combinar representaciones gráficas y tabulares, lo que incluye diagramas de transición de estado, diagramas de flujo, tablas de transición de estado y tablas de verdad, con el fin de modelar la forma en que el sistema reaccionará ante los eventos en función de las condiciones de tiempo y señales de entrada.

Es utilizado en aplicaciones de control, planificación de tareas y gestión de fallos, sus principales características son las siguientes:

- Entorno de modelado, componentes gráficos y motor de simulación para modelar y simular lógica compleja
- Semántica de ejecución determinista con jerarquía, paralelismo, operadores temporales y eventos
- Diagramas de estado, tablas de transición de estado y matrices de transición de estado que representan máquinas de estado finito
- Diagramas de flujo, funciones de MATLAB y tablas de verdad para representar algoritmos
- Animación de diagramas de estado, registro de actividades de estado, registro de datos y depuración integrada para analizar el diseño y detectar errores en tiempo de ejecución
- Comprobaciones estáticas y en tiempo de ejecución para detectar conflictos de transición, problemas cíclicos, incoherencias de estado, infracciones de rango de datos y condiciones de desbordamiento
- Máquinas de estado finito de Mealy y Moore.

En nuestro caso se ha utilizado para crear una máquina de estados de Mealy, que se describirá en detalle en el apartado del desarrollo del proyecto.

5.2.2. Simulink Coder

Con este Toolbox agregamos la funcionalidad de generar y ejecutar código C y C++ desde diagramas y modelos de Simulink, también gráficos de Stateflow y funciones de Matlab. El código fuente generado se puede utilizar para aplicaciones en tiempo real, tales como la simulación de modelos dinámicos MIL (model in the loop), el prototipado rápido SIL (software in the loop), y otras pruebas basadas en el soporte físico final HIL (hardware in the loop).

Gracias a esta herramienta se tiene la posibilidad de ajustar y controlar el código generado utilizando Simulink, o ejecutar e interactuar con el código fuera de Matlab y Simulink [23].

5.2.3. Embedded Coder Support Package for Arduino

Mathworks ofrece la posibilidad de integrar código en diferentes modelos hardware de bajo costo, tales como Arduino, LEGO MINDSTORMS NXT y Raspberry Pi. De esta manera se pueden diseñar algoritmos en Simulink para sistemas de control, robótica, procesamiento de audio y aplicaciones de visión por computador viéndolos actuar con su hardware. En nuestro caso se utilizará para generar código para las placas de Arduino, ya sea para la placa Arduino Uno o bien para la placa CM-900 que son las utilizadas en el desarrollo [24].

Arduino es una plataforma de creación de prototipos electrónicos de código abierto de bajo coste cuya fortaleza reside en la flexibilidad, además de la capacidad del hardware y el software fácil de usar.

Si se tiene instalado Simulink Coder se pueden utilizar los modelos de Arduino Duemilanove, Arduino Uno que es el mostrado en la figura 5.2. y Arduino Mega. Además nos permite el desarrollo de bloques de driver personalizados que gracias a su procesador Atmel de ATmega nos permite ejecutar bucles de control de hasta 25 Hz aunque no en tiempo real.

Figura 5.2. Placa de pruebas Arduino UNO

El Support Package Embedded Coder para Arduino nos permite crear aplicaciones de Simulink que se ejecutarán autónomamente sobre la placa de Arduino. El modelo de Simulink se convierte automáticamente en código C legible, compilado y descargable directamente en la placa de Arduino. Una vez realizada la generación del código C para Arduino podremos:

- Ver el código generado
- Modificar y volver a utilizar el código generado si es necesario
- El uso del PIL mode (processor in the loop)

Dotando a nuestro código de una gran trazabilidad, además dentro del paquete de ayuda se incluyen los siguientes bloques de drivers que nos facilitarán mucho la interconexión y el manejo de datos:

- Digital I/O
- Analog I/O
- Serial Write/Read.

Además de algunos ejemplos que también son de gran ayuda como los modelos:

- PIL
- Comunicación serie con un host
- Uso de Stateflow

5.3.MiddleWare de comunicación YARP (Yet Another Robot Platform)

Se trata de un conjunto de bibliotecas, protocolos y herramientas para mantener los módulos y dispositivos conectados entre sí a través del protocolo TCP/IP. Se trata de un producto middleware, ya que en ningún momento está controlando el sistema sino solamente enviando datos entre diferentes dispositivos que serán los encargados de actuar en función de los datos recibidos, por tanto no es un sistema operativo.

Para robótica humanoide es un producto muy útil y ampliamente utilizado, YARP [25] es un intento para hacer más estable y duradero el software robótico. Todo ello sin comprometer la capacidad de modificar y evolucionar el hardware y el software. YARP ayuda a organizar las comunicaciones entre los sensores, el procesador y los actuadores fomentando la flexibilidad del sistema. De esta manera permite que la evolución del sistema sea gradual y más fácil de llevar a cabo. El modelo YARP de comunicación es neutro para su transporte, de modo que el flujo de datos no depende de los detalles de las redes subyacentes y protocolos en uso.

YARP además es un producto de software libre, de código abierto, escrito por y para investigadores en robótica. Particularmente para la robótica humanoide es muy útil ya que dentro de esta rama, nos encontramos con muchos dispositivos hardware complicados de controlar. También se utiliza una gran variedad de software, por lo que YARP facilita mucho la tarea de implementar sistemas robustos. Gracias a las características anteriormente descritas, resulta un software muy atractivo y útil a la hora de realizar comunicaciones entre los diferentes componentes, de un sistema complejo.

Los componentes de YARP se pueden dividir en:

- libYARP_OS – se trata de la interfaz con el sistema operativo, se encarga de apoyar la fácil transmisión de datos a través de varios hilos e incluso a través de varias máquinas. YARP está escrito para ser utilizado en Linux, Windows, Mac OS X o Solaris. Además YARP utiliza la biblioteca de código abierto ACE (Adaptive Communication Environment), que es compatible con una amplia gama de entornos, y de la que YARP hereda esta virtud de portabilidad. Por ultimo cabe destacar que YARP está escrito casi por completo en C + +.
- libYARP_sig – es la encargada de las tareas de procesamiento de señales comunes (visual, auditiva, etc.) además permite interconectar fácilmente con otras bibliotecas de uso general, como por ejemplo con OpenCV.
- libYARP_dev – por ultimo esta librería crea una interfaz con otros dispositivos comunes que se utilizan dentro del marco de la robótica: framegrabbers, cámaras digitales, paneles de control de motores, etc.

Estos componentes se instalan por separado. Pero el componente básico que es libYARP_OS, debe estar instalado a nuestra disposición antes de poder utilizar los otros componentes. Para utilizar YARP en primer lugar siempre hay que crear un servidor como se muestra en la figura 5.3., que será el que se encargue de gestionar las conexiones entre los diferentes puertos, que representara cada uno de los dispositivos a conectar.

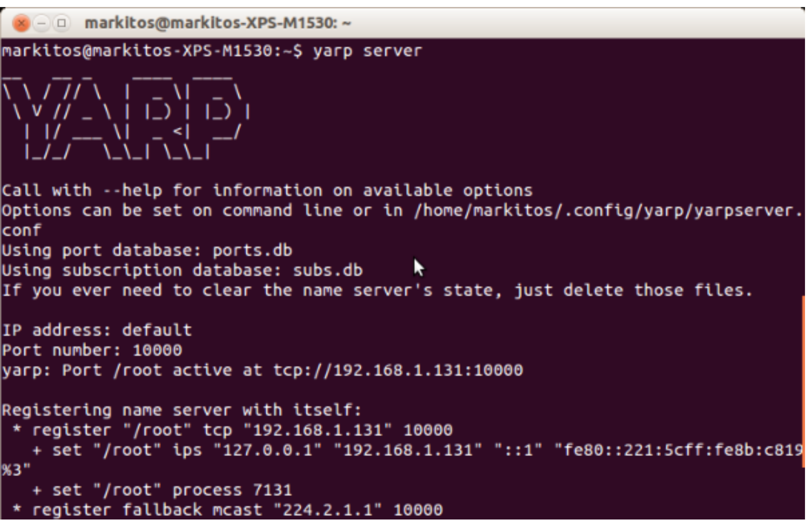


Figura 5.3. Creación de servidor YARP

5.4.OpenRAVE

El simulador OpenRAVE [26] fue concebido para aplicaciones de robots autónomos, ya que se caracteriza por su perfecta integración de los modelos 3D dentro del entorno de simulación, facilitando la visualización, y la planificación de las secuencias de comandos para el control de dichos robots. Gracias a su arquitectura de plugins permite a los usuarios que escriban sus controladores personalizados y amplíen su funcionalidad en diferentes entornos de desarrollo.

Haciendo uso de los múltiples plugins que el mercado ofrece OpenRAVE, podemos basar el control en algoritmos externos en los que se incluirá la planificación de tareas, el control de todos los movimientos del robot y también cualquier tipo de subsistema de detección con el que percibir el entorno que rodea al sistema en cuestión todo ello realizado dinámicamente en tiempo de ejecución.

El simulador OpenRAVE, fue desarrollado por Rosen Diankov en “the Quality of Life Technology Center” de la Universidad americana de “Carnegie Mellon Robotics Institute”. El proyecto nació en el año 2006, en sus comienzos fue solo una reescritura completa, definiendo nuevos conceptos del simulador RAVE que hasta aquel momento utilizaban, pero más tarde termino por definirse su propia arquitectura surgiendo un nuevo concepto de simulador, el cual comenzó a ser apoyado y utilizado por muchos investigadores del campo de la robótica a lo largo de todo el mundo, incluyéndose nuevas mejoras tras los múltiples desarrollos realizados sobre su base. En concreto la versión utilizada para el desarrollo de este proyecto es la versión 0.8.2., ya que es la última totalmente probado y estable que se puede encontrar actualmente.

Los usuarios de OpenRAVE centra su desarrollo en la utilización del lenguaje de programación basado en etiquetas XML, aunque otros lenguajes son también soportado, un ejemplo de estos lenguajes sería C o C++. Gracias a dichos lenguajes el desarrollador puede centrar sus esfuerzos en la implementación de modelos de los robots a utilizar. Ya que no han de gestionar de forma explícita los detalles relacionados con la cinemática y la dinámica del robot, o la detección de colisiones. Todo ello se incluirá en los diferentes escenarios (enviroments), que se incluirán en paralelo al modelo.

La arquitectura de OpenRAVE proporciona una interfaz flexible que puede ser combinada con otros paquetes populares tales como Robot Player o ROS, gracias a su diseño centrado en la planificación del movimiento autónomo a más alto nivel utilizando secuencias de comandos en lugar de planificar el de control basado en protocolos de mensajes a bajo nivel.

Además OpenRAVE se sustenta en una potente red en la que a través de secuencias de comandos sencillos se puede controlar y vigilar los movimientos del robot, apoyándose en los sensores y actuadores incluidos en los modelos XML que ejecuta. Los robots y el entorno o escenario son manejados durante el flujo de ejecución marcándoles un tiempo de refresco (timestamp) que una vez transcurrido refrescara la información, ya sea la recibida a través de comandos y en la que se basa para la realización de movimientos, o bien la generada por los sensores incluidos en el modelo. Para el control de este flujo de datos de entrada/salida se hace uso de lenguajes de programación como Python, Octave o Matlab.

Su principal ventaja y por la que ha sido elegido para el desarrollo de este es proyecto es su gran versatilidad y su arquitecturas abierta basada en componente, que permite que la comunidad de usuarios e investigadores del sector de la robótica compartan, comparen y mejoren tanto los algoritmos como los modelos creados para este entorno, este proyecto representa un buen ejemplo ya que el modelo que se muestra en la figura 5.4. a continuación y que será mejorado incluyéndole nuevos componentes durante el desarrollo del proyecto es heredado de otros desarrollos realizados con anterioridad.

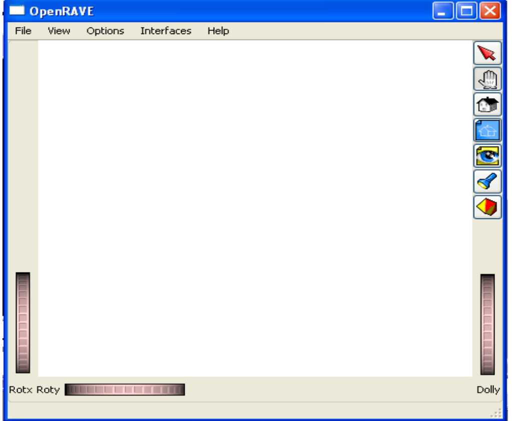


Figura 5.4. Entorno de simulación OpenRAVE

5.5. Blender

La versión de Blender [27] utilizada en el desarrollo de este proyecto es la versión 2.67b de la que se muestra en la figura 5.5. su pantalla de inicio. Dicha versión es la más actualizada en la que se han eliminado muchos de los bugs encontrados en versiones anteriores, cabe destacar que Blender es un programa de diseño y modelado de objetos tanto en 3D como en 2D, además es una plataforma abierta de software libre.

Blender tiene una interfaz muy inusual y altamente optimizada para la producción de gráficos en 3D. Esto puede ser un poco confuso para un usuario nuevo, pero a largo plazo se convierte en una herramienta muy útil y fácil de utilizar con un gran abanico de posibilidades.

Al inicio del proyecto se planteó la necesidad de modelar el entorno o escenario en el que se realizaban las pruebas de la carrera de obstáculos durante la competición universitaria CEABOT 2013. En este momento se presentaron varias opciones como fueron SolidWorks, Catia V5, y Blender entre otros entornos de diseño y modelado 3D. Finalmente se optó por Blender frente al resto de programas por su alta versatilidad a la hora de exportar en diferentes formatos, su rápido proceso de aprendizaje y por último la facilidad de conseguirlo y utilizarlo al tratarse de una plataforma abierta de software libre.

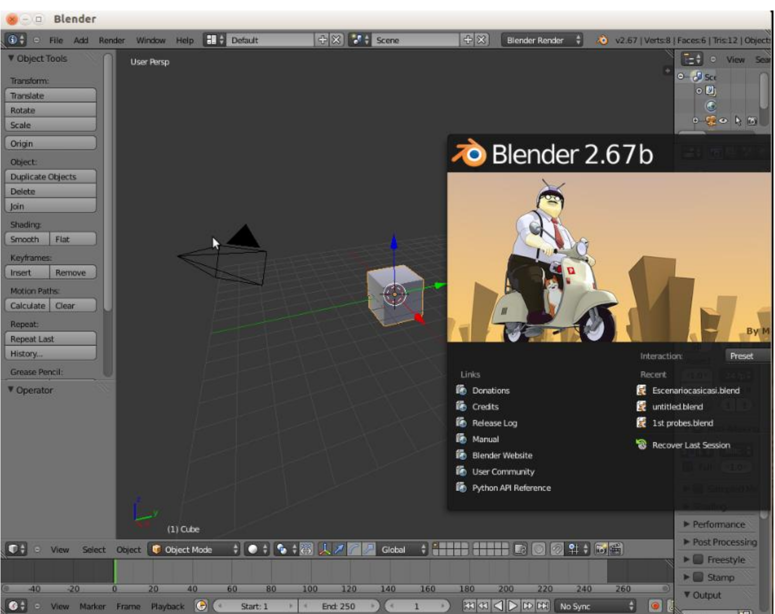


Figura 5.5. Pantalla de inicio de Blender

Este programa de software libre fue concebido por primera vez en diciembre de 1993 y se convirtió en un producto final para su uso por parte de la comunidad de desarrolladores en agosto de 1994, se trata de una aplicación integrada que permite la creación de una amplia gama de contenido en 2D y 3D.

Blender proporciona un amplio espectro de modelado, texturizado, iluminación, animación y la funcionalidad de post-procesamiento de vídeo en un solo paquete. Gracias a su arquitectura abierta, Blender proporciona interoperabilidad entre diversas plataformas, extensibilidad, todo en un tamaño pequeño, y con un flujo de trabajo totalmente integrado sin un coste computacional elevado. Blender es uno de los entornos de desarrollo de aplicaciones gráficas 3D en abierto más populares del mundo.

Está dirigido a profesionales dentro del campo del diseño gráfico así como a profesionales de los medios de comunicación y artistas distribuidos por todo el mundo, Blender puede ser usado para diversas actividades dentro del diseño 3D y 2D como son las de crear visualizaciones 3D e imágenes fijas, videos para su futura difusión, etc. la reciente incorporación de un motor 3D en tiempo real ha permitido la creación de contenido 3D interactivo para su reproducción independiente.

Fue originalmente desarrollado por la empresa "Not a Number" (NaN), y desde ese momento hasta la actualidad Blender ha continuado con la idea original de ser Software Libre, su código fuente se encuentra disponible bajo la licencia GNU GPL. Mientras que actualmente es la Blender Foundation en los Países Bajos la encargada de coordinar su continuo desarrollo.

Su mayor evolución se produjo entre 2008 y 2010, ya que la mayoría de las piezas clave de Blender fueron reescritas para mejorar notablemente sus funciones de flujo de trabajo y su interfaz de usuario. El resultado de este trabajo produjo la versión 2.5 del software.

Entre sus principales características cabe destacar las expuestas a continuación: representación de imágenes y post-procesado, suite de creación totalmente integrado, ofreciendo una amplia gama de herramientas esenciales para la creación del contenido 3D, incluyendo modelado, mapeado UV, texturizado, rigging, skinning, animación, simulación de partículas, scripting, renderizado, composición, post-producción, y creación de video juegos.

Se trata de un software multiplataforma, concebido con una interfaz gráfica OpenGL ya que esta es uniforme en muchas de las plataformas utilizadas en la actualidad y es fácilmente personalizable gracias a los scripts de python. Además está preparado para usarse en todas las versiones actuales de Windows (XP, Vista, 7,8), también en Linux, y en muchos otros sistemas operativos como son OS X, FreeBSD, Sun, etc.

Su arquitectura 3D de alta calidad permite la creación rápida y eficiente haciendo gala de un alto flujo de trabajo, además del pequeño tamaño del ejecutable, lo que lo convierte en fácilmente distribuible. Por último es muy valorable el apoyo de la comunidad de usuarios en los foros para preguntas.

6. Desarrollo del proyecto

Dentro de este apartado se describirá en primer lugar, todo el proceso seguido hasta conseguir el entorno de desarrollo basado en modelos para robots mini-humanoides, que tiene como objetivo este proyecto. En concreto se describirán todos los componentes creados, como han de ser utilizados y los procesos que habrá que seguir a la hora de utilizarlos.

En segundo lugar se describirá la manera de generar el código necesario, a partir de los controladores diseñados dentro del entorno. Para posteriormente descargarlo al controlador real y que este sea capaz de reproducir el comportamiento simulado, en el robot real.

6.1. Entorno de desarrollo

El entorno de desarrollo ha sido realizado siguiendo la línea descrita en el capítulo Estado del Arte, "Modelado basado en componentes". A medida que se vaya avanzando en el desarrollo de este capítulo, se realizará una descripción detallada de los componentes utilizados mostrados en la figura 6.1. (Controlador creado con Matlab-Simulink, simulador creado con OpenRave y por ultimo Middleware de comunicación implementado en YARP) y los procesos utilizados, para por ultimo describir la solución final, analizando los resultados obtenidos.

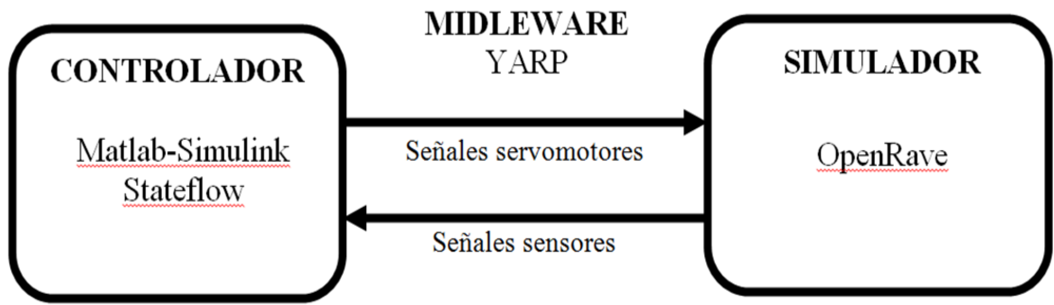


Figura 6.1. Esquema del modelado basado en componentes

Como se puede observar en la figura 6.1. el entorno de desarrollo creado, será totalmente modular, ya que se divide en tres módulos bien diferenciados entre si y totalmente independientes, por tanto en el futuro se podrán sustituir dichos módulos si se considera oportuno, o bien modificarlos de manera independiente, sin que esto afecte a los demás, cumpliendo con el modelo de componentes, seguido a la hora de desarrollar el entorno.

6.1.1. Controlador basado en una Maquina de Estados

Se va a utilizar, una máquina de estados sobre la que basar el control del mini-humanoide. Aunque existen varios tipos de máquinas de estados el estudio se centrará sobre los dos tipos que es posible utilizar dentro de Stateflow al ser el software elegido para su implementación. Dichos dos tipos son las máquinas de estados de Mealy y Moore comparadas en la tabla 6.1. Son circuitos síncronos. Los circuitos síncronos están compuestos por un circuito digital en el cual sus partes se encuentran sincronizadas por una señal de reloj.

En un circuito síncrono, los cambios producidos en los diferentes niveles lógicos son simultáneos. Estos cambios o transiciones se realizan después de un cambio de nivel o flanco de subida de una señal cuadrada y periódica llamada reloj (CLK). Por tanto la entrada a cada elemento de almacenamiento debe alcanzar su valor final antes de la llegada del siguiente flanco de la señal de reloj, por lo que el comportamiento de un circuito ha de ser totalmente predecible. Normalmente se requiere de un cierto retardo para cada una de las operaciones lógicas, por lo que existe una velocidad máxima de respuesta para cada sistema síncrono.

Máquina de Mealy	Máquina de Moore
La salida depende del estado actual y de las entradas	La salida depende sólo del estado actual
Por lo general, tienen menos número de estados	El número de estados es mayor o igual a la máquina de Mealy
Es menos estable	Es mas estable
Para probar un circuito, primero se hace el cambio en la entrada X y después se da el pulso de reloj	Para probar un circuito, primero se da el pulso de reloj y después se hace el cambio en la entrada X
Las salidas se encuentran en la arista	Las salidas se encuentran dentro del estado
1/1, 1/0, 0/0, S_1, S_2, 0/1, 1/0, S_0, 0/0	1, 1, 0, S_1 1, S_2 0, 0, 1, S_0 0, 0

Tabla 6.1. Comparativa Máquina de estados de Moore y Mealy

- **La máquina de Mealy,** es una máquina de estados finita, donde las salidas vienen determinadas por el estado actual y la entrada que se está produciendo en ese momento concreto. Esto significa que en el diagrama de estados se ha de incluir una señal de salida por cada una de las aristas de transición. Por ejemplo, en la transición de un estado 1 a un estado 2, si la entrada es cero la salida puede ser uno, y se debe poner sobre la arista la etiqueta “0/1”.

- **Una máquina de Moore,** en contraste, la salida de una máquina de estados finitos de Moore, depende solo del estado actual y no de la entrada actual. Por lo tanto, los estados de una máquina de Moore son la unión de los estados de la máquina de Mealy y el producto cartesiano de estos estados y el alfabeto de entrada.

6.1.1.1. Desarrollo teórico de la máquina de estados Mealy para el control del mini-humanoide

Se ha optado por la utilización de una máquina de estados de Mealy, en vez de una máquina de estados de Moore, ya que la principal característica de este tipo de máquinas que las diferencia y fortalece frente a las de tipo Mealy, para esta aplicación es la de que; el siguiente estado, depende no solo del estado actual, sino también de las entradas que se están recibiendo, en los puertos de dicha máquina de estados. A continuación se muestra las principales características que definen este tipo de máquinas secuenciales, para posteriormente analizar y desarrollar nuestra propia máquina de estados siguiendo las premisas dadas.

Una máquina de Mealy es un conjunto de elementos 6 (S, S0, Σ, Λ, T, G), que consiste de:

- Un conjunto finito de estados (S)
- Un estado inicial S0 el cual es un elemento de (S)
- Un conjunto finito llamado alfabeto de entrada (Σ)
- Un conjunto finito llamado alfabeto de salida (Λ)
- Una función de transición ($T : S \times \Sigma \rightarrow S$)
- Una función de salida ($G : S \times \Sigma \rightarrow \Lambda$)

Se decidido basar el núcleo central del control del robot mini-humanoide sobre una máquina de estados, ya que además de cumplir con la misión requerida, tiene una característica muy importante que es la escalabilidad o trazabilidad. Gracias a dicha característica resulta muy sencillo modificarla de cara a cumplir las diferentes misiones que se proponen dentro de la competición CEABOT ya sea incorporando nuevos estados o diferentes sensores (entradas).

En primer lugar se analiza el problema en cuestión. Los objetivos de nuestra máquina de estados que posteriormente se traducirán a (Entradas/Salidas/Estados) son los siguientes.

- El robot ha de andar recto en caso de que no encuentre ningún obstáculo en su camino.

- El robot ha de andar hacia la derecha en caso de que encuentre un obstáculo situado a su izquierda que le obstruya el camino, o bien en el caso que encuentre un obstáculo enfrente y tenga más espacio para evitarlo por la derecha que por la izquierda.
- El robot ha de andar hacia la izquierda en caso de que encuentre un obstáculo situado a su derecha que le obstruya el camino, o bien en el caso que encuentre un obstáculo enfrente y tenga más espacio para evitarlo por la izquierda que por la derecha.
- Una vez analizado el problema a resolver se pasa a las definiciones de los estados, entradas y salidas que se utilizarán en la implementación de la máquina de estados de Mealy, y por último se mostrar la tabla de transiciones donde se analizarán con detalle los resultados obtenidos, previamente a su implementación dentro de Stateflow.

Entradas

Las entradas estarán representadas tanto en el simulador OpenRave, como en el robot real por tres sensores de proximidad situados uno en el frente del robot y los otros dos a la izquierda y la derecha del solidarios al cuerpo, respectivamente. En la tabla 6.2. que se muestra a continuación, vendrán representados por la terna (u,v,w). Además en dicha tabla se muestra el significado en función de la terna recibida.

Entradas			
w	v	u	Significado
0	0	0	No hay obstáculos
0	0	1	Obstáculo enfrente
0	1	0	Obstáculo izquierda
0	1	1	Obstáculo enfrente izquierda
1	0	0	Obstáculo derecha
1	0	1	Obstáculo enfrente derecha
1	1	0	Obstáculos izquierda y derecha
1	1	1	Obstáculo izquierda derecha y enfrente

Tabla 6.2. Entradas de la Máquina de Estados de Mealy

Cabe destacar que el valor binario que mostrado en las entradas viene determinado en función de si:

- **“1”** → El valor recibido desde el sensor supera un determinado umbral y por tanto existe riesgo de colisión.
- **“0”** →El valor recibido desde el sensor no supera el umbral y por tanto no existe riesgo de colisión.

Salidas

Las salidas del sistema determinarán las órdenes que hemos de enviar a los servomotores ya sea en la simulación dentro de OpenRave o bien en el robot real para que se produzca el movimiento deseado. En la tabla 6.3., vendrán representados por los valores binarios (S_0, S_1). Además en dicha tabla se muestran los diferentes significados, en función de los valores enviados a la salida.

Salidas		
S_1	S_0	Significado
0	0	Parar
0	1	Andar a la izquierda
1	0	Andar a la derecha
1	1	Andar recto

Tabla 6.3. Salidas de la Máquina de estados de Mealy

Cabe destacar que dichas salidas serán posteriormente traducidas a una matriz de 18 posiciones con los valores necesarios para generar los movimientos deseados, dichos valores estarán a su vez representados por dos valores diferentes. El primero de ellos será la posición del servomotor deseada y el segundo la velocidad a la que ha de moverse hasta alcanzarla.

Estados

Los estados del sistema implementado, son el resultados de las entradas y salidas en cada instante durante la ejecución, mientras se encuentre activo uno de los estados el sistema realizará la acción asociada a ellos tanto en la simulación dentro de OpenRave como en el robot real. En la tabla 6.4., vendrán representados por los valores binarios (Q_0, Q_1). Además en dicha tabla se muestra la descripción de la acción que se encuentra realizando el robot mientras esta activo el estado indicado.

Estado		
Q_1	Q_0	Acción
0	0	Estado 0 --> Parado / en preparación / calibración
0	1	Estado 1 --> Girando hacia la izquierda
1	0	Estado 2 --> Girando hacia la derecha
1	1	Estado 3 --> Andando recto

Tabla 6.4. Estados de la Máquina de estados de Mealy

Cabe destacar que los estados se encuentran activos durante un tiempo determinado por el usuario en función de las necesidades del sistema. Dicho tiempo lo marcara el reloj interno del sistema, pudiendo ser en función de si se trata del robot dentro del entorno de simulación o del robot real, el reloj interno del ordenador en el que se esté llevando a cabo la simulación o el reloj de la placa en la que se haya descargado el código y que comande el robot real.

Tabla de transiciones

Dentro de este punto se analizarán las salidas que han de producirse además del estado al que se pasara en función de las entradas que se reciban de los tres sensores situados en el robot y teniendo en cuenta el estado actual que se encuentra activo.

Estado		Entradas			Estado siguiente		Salidas	
Q_1	Q_0	w	v	u	Q_1^+	Q_0^+	S_1	S_0
0	0	0	0	0	1	1	1	1
0	0	0	0	1	0	1	0	1
0	0	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	0	0	1	0	1
0	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	0	1
0	0	1	1	1	0	1	0	1
0	1	0	0	0	1	1	1	1
0	1	0	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1	0
0	1	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0	1
0	1	1	1	0	0	1	0	1
0	1	1	1	1	0	1	0	1
1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	0	0	1	0	1
1	0	1	0	1	0	1	0	1
1	0	1	1	0	1	0	1	0
1	0	1	1	1	1	0	1	0
1	1	0	0	0	1	1	1	1
1	1	0	0	1	0	1	0	1
1	1	0	1	0	1	0	1	0
1	1	0	1	1	1	0	1	0
1	1	1	0	0	0	1	0	1
1	1	1	0	1	0	1	0	1
1	1	1	1	0	1	0	1	0
1	1	1	1	1	1	0	1	0

Tabla 6.5.Calculo de la Máquina de estados de Mealy

Tras un análisis exhaustivo de la tabla 6.5., se pasó al cálculo y posterior simplificación de las funciones de transición dando resultado a los valores que se hace necesario representar en cada una de las transiciones entre los diferentes estados. En concreto para cada una de las transiciones se analizan las entradas recibidas y las salidas que se han de enviar al cambiar de un estado a otro o bien al mantenerse en el mismo estado.

Dichas transiciones se representarán en función del estado del que provienen y del estado al que van según la siguiente nomenclatura de entradas y salidas (u,v,w, S_0, S_1). En este caso también se está utilizando un código binario de (1, 0), pero para simplificar y reducir el número de transiciones necesarias, en caso de que alguno de los valores de entrada produjera la misma salida y la transición sea al mismo estado independientemente de su valor se ha sustituido por un valor (x) que representa tanto a 1 como a 0.

	Estado		Entradas			Estado siguiente		Salidas		Transición
	Q_1	Q_0	w	v	u	Q_1^+	Q_0^+	S_1	S_0	w v u / S_1 S_2
E	0	0	0	0	0	1	1	1	1	0 0 0 / 1 1
S	0	0	0	0	1	0	1	0	1	0 0 1 / 0 1
T	0	0	0	1	0	1	0	1	0	0 1 X / 1 0
A	0	0	0	1	1	1	0	1	0	
D	0	0	1	0	0	0	1	0	1	1 X X / 0 1
O	0	0	1	0	1	0	1	0	1	
	0	0	1	1	0	0	1	0	1	
1	0	0	1	1	1	0	1	0	1	

Tabla 6.6. Tabla de transiciones del Estado 1

	Estado		Entradas			Estado siguiente		Salidas		Transición
	Q_1	Q_0	w	v	u	Q_1^+	Q_0^+	S_1	S_0	w v u / S_1 S_2
E	0	1	0	0	0	1	1	1	1	0 0 0 / 1 1
S	0	1	0	0	1	0	1	0	1	0 0 1 / 0 1
T	0	1	0	1	0	1	0	1	0	0 1 X / 1 0
A	0	1	0	1	1	1	0	1	0	
D	0	1	1	0	0	0	1	0	1	1 X X / 0 1
O	0	1	1	0	1	0	1	0	1	
	0	1	1	1	0	0	1	0	1	
2	0	1	1	1	1	0	1	0	1	

Tabla 6.7. Tabla de transiciones del Estado 2

	Estado		Entradas			Estado siguiente		Salidas		Transición
	Q_1	Q_0	w	v	u	Q_1^+	Q_0^+	S_1	S_0	w v u / S_1 S_2
E	1	0	0	0	0	1	1	1	1	0 0 0 / 1 1
S	1	0	0	0	1	1	0	1	0	0 0 1 / 1 0
T	1	0	0	1	0	1	0	1	0	0 1 X / 1 0
A	1	0	0	1	1	1	0	1	0	
D	1	0	1	0	0	0	1	0	1	1 0 X / 0 1
O	1	0	1	0	1	0	1	0	1	
	1	0	1	1	0	1	0	1	0	1 1 X / 1 0
3	1	0	1	1	1	1	0	1	0	

Tabla 6.8. Tabla de transiciones del Estado 3

	Estado		Entradas			Estado siguiente		Salidas		Transición
	Q_1	Q_0	w	v	u	Q_1^+	Q_0^+	S_1	S_0	w v u / S_1 S_2
E	1	1	0	0	0	1	1	1	1	0 0 0 / 1 1
S	1	1	0	0	1	0	1	0	1	0 0 1 / 0 1
T	1	1	0	1	0	1	0	1	0	0 1 X / 1 0
A	1	1	0	1	1	1	0	1	0	
D	1	1	1	0	0	0	1	0	1	1 0 X / 0 1
O	1	1	1	0	1	0	1	0	1	
	1	1	1	1	0	1	0	1	0	1 1 X / 1 0
4	1	1	1	1	1	1	0	1	0	

Tabla 6.9. Tabla de transiciones del Estado 4

Una vez definidos todos los valores de las transiciones entre estados así como el estado inicial y el estado al que ha de pasar, se realizó un boceto con la representación de la máquina de estados completa para posteriormente representarla dentro del módulo de Simulink, con la Toolbox Stateflow. La máquina de estados definitiva es la que se muestra en la figura 6.2.

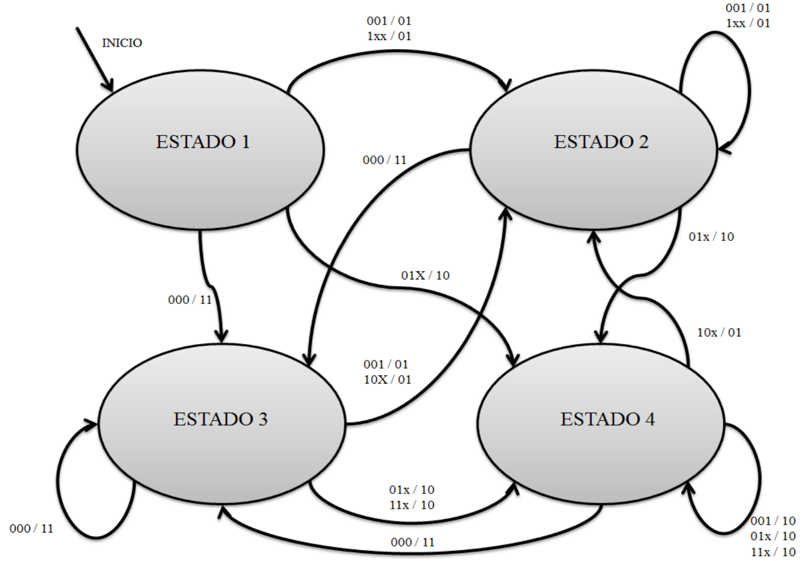


Figura 6.2. Representación de la Máquina de Estados de Mealy

6.1.1.2. Implementación de la máquina de estados dentro de StateFlow.

En este apartado se va a describir cómo se ha implementado la máquina de estados dentro del modelo de Simulink utilizando los bloques de la Toolbox Stateflow, además de cómo se han de configurar los parámetros dentro de este bloque para que cumpla con las necesidades descritas en los puntos anteriores.

En primer lugar se creó un modelo de Simulink en el que se insertó un bloque de Stateflow, llamado "Chart" o gráfico. Aunque también cabía la posibilidad de basar el diseño de nuestra máquina de estados sobre una "Truth Table" o tabla de verdad en la que simplemente se debían insertar los parámetros de la tabla de transiciones. Finalmente se optó por el grafico, al dotar al modelo de un carácter mucho más visual e intuitivo, pensando siempre en su trazabilidad de cara a futuras modificaciones. En la figura 6.3. se encuentra situado este bloque dentro de las librerías de Simulink en paralelo a como se integró en el modelo final, mostrándose ya con las entradas y salidas configuradas.

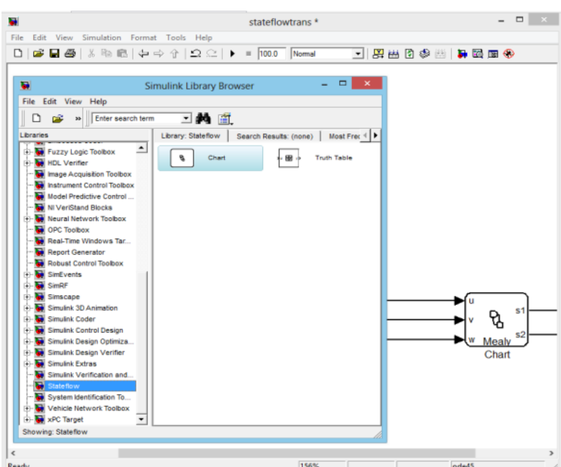


Figura 6.3. Bloque de Máquina de Estados de Mealy en StateFlow

Una vez integrado el bloque de la máquina de estados dentro del modelo de Simulink hubo que configurar varios puntos para que se adaptase al diseño realizado sobre papel basándose en el tutorial [28]. En primer lugar hubo que elegir la configuración de los parámetros descritos a continuación, además dicha configuración elegida se muestra en la figura 6.4. expuesta tras la siguiente explicación.

"State Machine Type": el tipo elegido fue la máquina de estado Mealy por todo lo expuesto en el análisis realizado en el punto anterior, las opciones que soporta este bloque además de la máquina de Mealy son la máquina de Moore, y la máquina clásica en la que el usuario define todos los aspectos desde cero, pero complica mucho el obtener un resultado adecuado.

"Update Method": en este caso elegimos el método de refresco come "inherited" o heredado ya que como la máquina de estados se integra dentro de un modelo de Simulink en el que existen dependencias con otros bloques como son los de comunicación, etc., no se podía configurar con uno delos otros métodos al causar incoherencias durante la ejecución. Las otras opciones que se podrían haber utilizado en caso de no existir dependencias con el resto del modelo son las de "contineous" o "discrete", en castellano continuo o discreto.

"Sample Time": el tiempo de muestreo en este caso fue elegido como 100ms para que fuse lo suficientemente pequeño como para que el robot no colisionase mientras estaba realizando alguno de los movimientos.

Además de los tres parámetros, existían varios más que fueron configurados como se muestra en la figura 6.4., para que soportarse el tipo de datos que le íbamos a transmitir desde Simulink, así como el orden de ejecución deseado por el usuario y evitando que se produjeran eventos de "overflow" o desbordamiento.

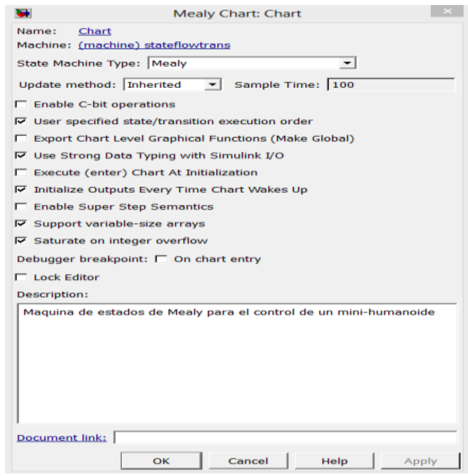


Figura 6.4. Configuración de la Máquina de Estados en StateFlow

Una vez configurado el bloque en el que se implementó la máquina de estados se pasó a incluir dentro del "Model Explorer" las entradas y salidas del bloque como se muestra en la figura 6.5., en concreto se configuraron las entradas (u, v, w) y las salidas (s1, s2) de manera análoga a las que se definieron durante el proceso de cálculo.

El tipo de datos configurado para las entradas como para las salidas es de tipo “double”, porque aunque no resulta necesario al estar trabajando en binario todos los datos, pudiendo utilizar datos tipo “integer” o incluso datos de tipo “boolean”, se consideró mejor configurarlos como “doublé” de cara a permitir futuras modificaciones en los datos que se recibirán en las entradas o los datos que se enviara a la salida.

Además dentro de la figura 6.5., también se pueden observar los diferentes estados que posee la máquina de estados, aunque estos fueron definidos directamente en el diagrama que se mostrará a continuación también pueden ser modificados desde esta ventana “Model Explorer”, pinchando sobre ellos.

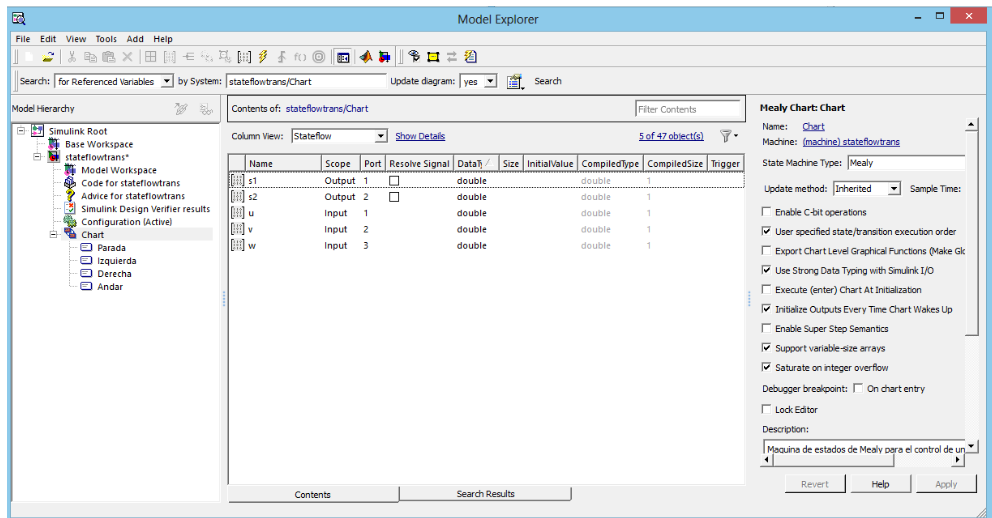


Figura 6.5. Configuración Entradas/Salidas de la Máquina de Estados en StateFlow

Una vez los parámetros necesarios para que la máquina de estados, se encontraran configurados para adaptarse a las necesidades analizadas en el apartado anterior. Era el momento de proceder a la implementación de la máquina de Mealy. Los bloques y transiciones necesarios para su desarrollo se describen a continuación:

“State”: este bloque fue utilizado para definir cada uno de los cuatro estados de los que consta la máquina de estados implementada, a este bloque se le pueden asociar funciones además de transiciones de entrada y de salida. En la figura 6.6., se puede apreciar su representación, concretamente en cada una de las cajas (Parada, Izquierda, Derecha, Andar). Para incluir más estados como los mencionados anteriormente basta con pinchar sobre el primero de los iconos de la columna izquierda mostrados en la figura 6.6. y arrastrarlo al diagrama.

“Default transition”: este bloque representaría cada una de las flechas que realizan las transiciones entre un estado y otro o entre los diferentes bloques de unión que se analizarán posteriormente. Dentro de estas transiciones se han de definir las entradas que las provocan así como las salidas que han de producir. Para incorporar nuevas transiciones simplemente habrá que arrastra el bloque situado en tercer lugar en la columna izquierda al diagrama y unir sus extremos entre los bloques que se quiera realizar la transición. En concreto para la máquina de estados de Mealy se definen de la siguiente manera:

- **Entradas**: han de definirse entre corchetes y solo pueden darse de una en una por lo que se hace necesario el uso de bloques de unión. Un ejemplo sacado de la figura 6.6., sería el siguiente: [u==0], en caso de que se cumpla esta condición el flujo de ejecución continuara por dicha transición, en caso contrario buscara un camino alternativo que ha de existir en el que sea [u==1], se utilizan dos signos de igualdad al ser una condición que se ha de evaluar. Para nuestro ejemplo en concreto se han definido tres entradas que toman valores binarios y son (u, v, w).
- **Salidas**: han de definirse entre llaves y pueden darse en grupos de varias salidas separadas por comas. En este caso, siempre se dan en grupos de dos (S1, S2), cabe destacar que solo se producen salidas en la última transición antes de entrar a un nuevo estado ya que han de evaluarse las tres condiciones (entradas) antes de decidir cuál será la salida. Un ejemplo sacado de la imagen mostrada más adelante es el de {S1=1, S2=0} que indicara que el robot ha de andar hacia adelante al no existir ningún obstáculo que se interponga en su camino.

“Connective Junctions”: este bloque representa cada uno de los pequeños círculos que se muestran en el diagrama de la imagen y su misión es la de evaluar la siguiente condición (entrada) produciendo en función de su valor en nuestro caso 1 o 0 la continuación del flujo de ejecución por el camino determinado. Siempre ha de situarse entre tres transiciones como mínimo una de entrada y dos de salida. Para incorporar nuevas uniones como estas simplemente se ha de arrastrar el bloque situado en cuarta posición dentro de la columna de la izquierda y situarlo entre un mínimo de tres transiciones, una de entrada y dos de salida.

Cabe destacar que el proceso de implementación de la máquina de estados de Mealy es diferente al de resto de máquinas de estados y a su vez estas son diferentes entre ellas ya que la nomenclatura y forma de representación es diferente debido a sus diferentes capacidades.

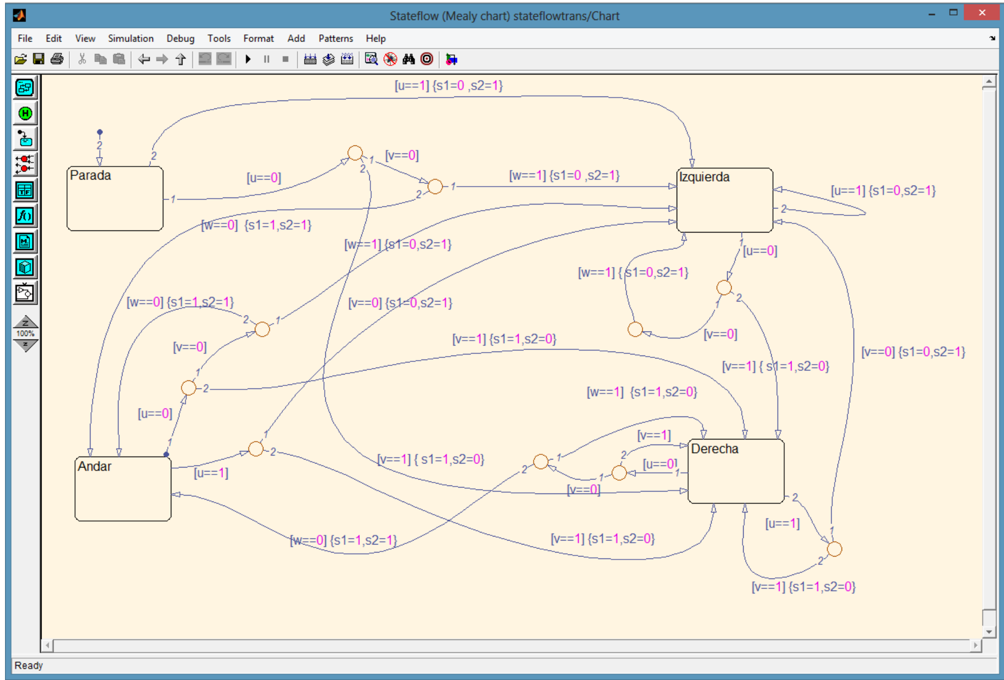


Figura 6.6. Máquina de Estados implementada en StateFlow

Por último se hubo de seleccionar un "Solver" o solucionador adecuado para el proceso de simulación al que nos enfrentaríamos posteriormente para lo cual se basó en el siguiente tutorial [29], en primer lugar explicar que un "Solver" es un componente del software Simulink. El cual determina el tiempo de la siguiente etapa de simulación aplicando un método numérico para resolver el conjunto de ecuaciones diferenciales ordinarias que representan el modelo. En el proceso de solucionar el problema del valor inicial, el "Solver" satisface además los requisitos de precisión que el usuario le especifique.

En concreto de los múltiples tipos de "Solvers" que existen dentro de Simulink se decidió utilizar uno de tipo "Variable-step" o paso variable para poder varían dinámicamente la duración del paso durante la simulación en función del error local, para lograr que se ejecute siempre dentro de las tolerancias especificadas. Con lo que se puede reducir el número total de pasos, y el tiempo de simulación requerido manteniendo el nivel de precisión especificado.

Dentro de las posibilidades ofrecidas para los "Solvers" de tipo "Variable-step" se optó por el de tipo ode45 con una tolerancia de 1µs lo cual representa un 0.1% del paso total de 100ms, siendo por tanto despreciable frente al total y otorgando una gran precisión a la simulación.

El “Solver” ode45: es un solucionador de Runge-Kutta (4,5), es un método de quinto orden que realiza una estimación de cuarto orden del error. Este solucionador utiliza también un interpolador “libre” de cuarto orden lo que le permite ser muy exacto.

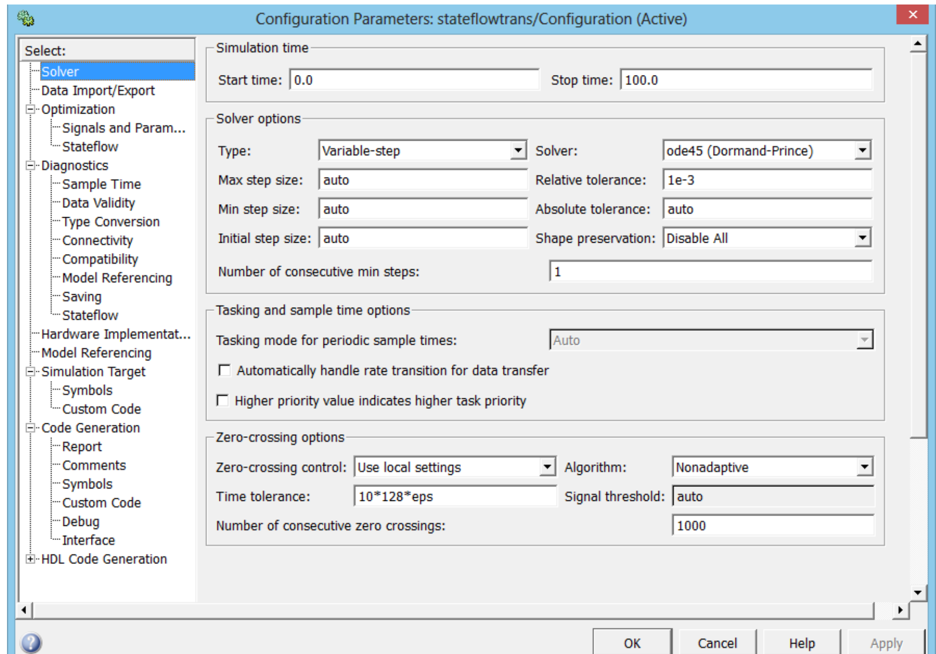


Figura 6.7. Configuración del Solver dentro de StateFlow

6.1.2. Entorno de simulación.

En este apartado se estudiará el entorno de simulación, realizando un análisis en profundidad de cuáles son los elementos que lo componen, cómo se han creado dichos elementos y cómo se podrían modificar, o añadir nuevos. Hasta que finalmente se mostrará el entorno de simulación definitivo que integra todos los elementos necesarios.

El entorno de simulación ha sido creado sobre el simulador OpenRave, el cual es programable a través de archivos XML [30], para el diseño del simulador se partió de un diseño existente dentro de la Asociación de Robótica, creado por Franklin y Gloria, el cual se puede descargar e instalar desde la web incluida en la referencia [31]. Incluye un modelo del robot Bioloid, con los servomotores funcionando, gracias a la etiqueta <manipulator> como se muestra en la figura 6.8., pero sin ningún sensor los cuales eran necesarios para este desarrollo, ni tampoco un escenario sobre el que simular las pruebas. Otro de los puntos que no incluía este robot y que hubo que incorporar, eran las propiedades físicas del entorno, por lo que no se encontraba sometido a ninguna fuerza externa.

```
- <Robot name="Humanoid_Robot_Bioloid">
    <!--<translation>0 0.8 0</translation>-->
    <translation>0 0.7 0</translation>
  - <KinBody file="Bioloid_Trunk.xml">
      <KinBody file="Bioloid_Arm_right.xml"/>
    - <Joint name="leftTorsoDummy" type="hinge" enable="false">
        <Body>Torso_dummy1</Body>
        <Body>waist</Body>
        <limits>0 0</limits>
      </Joint>
      <KinBody file="Bioloid_Arm_left.xml"/>
    - <Joint name="rightTorsoDummy" type="hinge" enable="false">
        <Body>Torso_dummy2</Body>
        <Body>waist</Body>
        <limits>0 0</limits>
      </Joint>
      <KinBody file="Bioloid_Leg_left.xml"/>
    - <Joint name="leftHipYawDummy" type="hinge" enable="false">
        <Body>hipDummyLeft</Body>
        <Body>waist</Body>
        <limits>0 0</limits>
      </Joint>
      <KinBody file="Bioloid_Leg_right.xml"/>
    - <Joint name="rightHipYawDummy" type="hinge" enable="false">
        <Body>hipDummyRight</Body>
        <Body>waist</Body>
        <limits>0 0</limits>
      </Joint>
    </KinBody>
  - <Manipulator name="leftArmManip">
      <base>Torso_dummy2</base>
      <effector>lPalm</effector>
      <direction>-1 0 0</direction>
    </Manipulator>
  - <Manipulator name="rightArmManip">
      <base>Torso_dummy1</base>
      <effector>rPalm</effector>
      <direction>1 0 0</direction>
    </Manipulator>
```

Figura 6.8. Extracto de código del archivo bioloid_robot.xml

El modelo recibido constaba de todos los archivos .wrl que definen las formas de cada una de las piezas que lo componen, como son brazos, piernas, torso, cabeza, etc. Y también de los archivos XML para poder simular el robot dentro de OpenRave. El principal archivo (bioloid_robot.xml), ya incluía todos los elementos unidos formando un robot completo como se muestra en la figura 6.9, en la que además se puede observar, la forma de ejecutarlo a través del comando “openrave bioloid_robot.xml” una vez dentro de la carpeta /bioloid-open/ y teniendo instalado el simulador OpenRave. Dentro de este archivo, se produce el ensamblaje de todos los elementos creados en archivos diferentes incluyéndolos como nuevos <kindbody> y uniéndolos entre sí a través de la etiqueta <joint> como se puede observar en la figura 6.8. mostrada anteriormente.

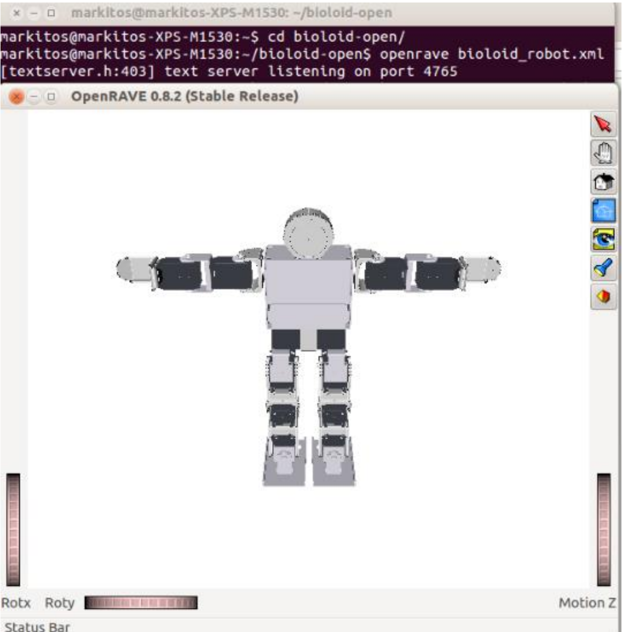


Figura 6.9. Archivo bioloid_robot.xml ejetudo en OpenRave

Un ejemplo de estos <kindbody> que forman el robot completo sería el archivo en el que se define el brazo derecho (Bioloid_Arm_Right.xml) mostrado en la figura 6.10., en estos archivos primarios es donde se referencian los modelos .wrl como <render> para que adquieran la forma creada en un programa de diseño 3D, como por ejemplo Blender.

```
− <KinBody name="rightArm">
    <modelsdir>models</modelsdir>
    <Body name="Torso_dummy1" type="dynamic"> </Body>
  − <Body name="ras" type="dynamic">
      <offsetfrom>Torso_dummy1</offsetfrom>
      <Translation>-0.076500 0.006 0.000000</Translation>
    − <Geom type="box" render="false">
        <Extents>0.01700 0.01250 0.02200</Extents>
      </Geom>
      <RotationAxis>1 0 0 -90</RotationAxis>
      <!--GIRAR -90° EL BRAZO-->
      <Translation>0.00 0.015 0.015</Translation>
      <!--MOVER EN EL EJE Y ^ Z-->
      <!--H O M B R O     D E R   C H O-->
    − <Geom type="trimesh">
        <RotationAxis>0 1 0 90</RotationAxis>
        <RotationAxis>0 0 1 90</RotationAxis>
        <Translation>0.08 0.03 0.022</Translation>
        <render>Bioloid/shoulder_left.wrl 0.01</render>
      </Geom>
    − <Mass type="box">
        <total>0.009</total>
        <inertia>0.003225 0 0 0 0.004119 0 0 0 0.002129</inertia>
      </Mass>
    </Body>
    <adjacent>Torso_dummy1 ras</adjacent>
  − <Joint name="rsp" type="hinge">
      <Body>Torso_dummy1</Body>
      <Body>ras</Body>
      <offsetfrom>ras</offsetfrom>
      <Anchor>0.017000 0.015000 0.000000</Anchor>
```

Figura 6.10. Extracto del código Bioloid_Arm_Right.xml

El robot completo se incorpora dentro del entorno de simulación analizado a lo largo de este apartado como se muestra en la figura 6.11. extraída del archivo bioloid.env.xml, en el que se crea el entorno completo de simulación al incorporar las etiquetas del lenguaje xml <enviroment>.

```
-<environment>
  -<Robot name="Bio_Robot" file="bioloid_robot.xml">
     <RotationAxis>1 0 0 90</RotationAxis>
     <translation>0 -0.5 -0.4</translation>
   </Robot>
```

Figura 6.11. Código para incorporar el robot al entorno de simulación final

Hasta este momento, tras los expuesto anteriormente solo se disponía del modelo del robot, por tanto se tenían que crear los escenarios, además teniendo en cuenta que hasta este momento no estaban incorporadas las leyes físicas dentro del simulador, había que añadirlas para que este fuera un simulador realista. Por ultimo añadirle al robot los sensores que nos diesen la información del entorno, de esta forma se aseguraría que el robot, tuviese un comportamiento análogo al que tendría en el mundo real.

De los puntos analizados en el párrafo anterior, surge la organización de este apartado dentro del proyecto, que por tanto se dividirá en tres sub apartados hasta que finalmente se obtenga el simulador realista deseado.

6.1.2.1. Creación de escenarios para el entorno de simulación

Para la creación de escenarios 3D, que simulen los escenarios a los que se enfrentará el robot mini-humanoide durante la competición CEABOT, se ha utilizado el entorno de desarrollo Blender. Tras instalar Blender 2.5 en su versión para Linux, se comenzó por estudiar su manual [32] y visualizar algunos de los video tutoriales que se ofrecen online, con lo que se logró, familiarizarse con en el entorno antes de comenzar a modelar el escenario de la prueba “Carrera de obstáculos” de la competición CEABOT 2013.

El escenario a modelar consta de un plano inferior (suelo), cuatro planos laterales (paredes) y varios obstáculos, en total el día de la pruebas los jueces situaran 6 obstáculos distribuidos por el escenario en diferentes configuraciones. Las medidas indicadas en la normativa de la competición como se muestra en la figura 6.12. extraída de la normativa del campeonato CEABOT 2013, expuesto en el Anexo 1, son las siguientes:

- Suelo (2,5 x 2 metros)
- Paredes (2,5 x 2 x 0,5 metros)
- Obstáculos (0,2 x 0,2 x 0,5 metros)

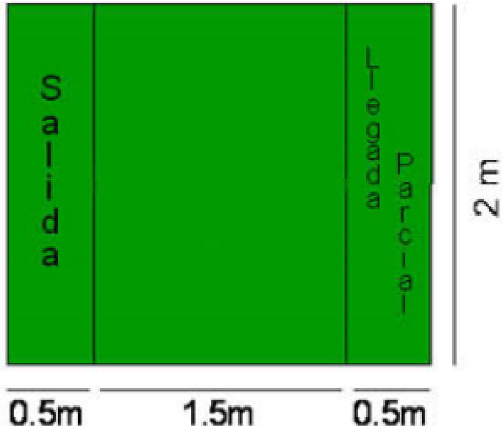


Figura 6.12. Normativa para el escenario de la competición CEABOT

Una vez el escenario era conocido se comenzó a modelarlo, en principio se comenzó realizando un cubo de base rectangular, posteriormente se eliminó la cara superior obteniendo de esta manera un cubo vacío, en el que ya se disponía del suelo y de las paredes. Una vez realizado, se separó en dos piezas dicho cubo, la primera de estas piezas era la base del cubo que representaría el suelo y la segunda las paredes.

En este momento se comenzó a añadir las texturas necesarias para que adquiriesen el aspecto del escenario original, fue entonces cuando surgió el primero de los problemas ya que no se podía insertar una textura diferente a las paredes, de la insertada en el suelo al tratarse originalmente de una misma pieza. En ese momento se decidió modelar cada pieza por separado, dotando al diseño de una mayor modularidad y por tanto mayor versatilidad a la hora de crear nuevos escenarios. Finalmente el modelado se dividió en tres piezas diferentes.

Suelo

El desarrollo del suelo se puede dividir en dos partes, la primera de ellas sería algo trivial al tratarse simplemente de crear un plano y posteriormente dimensionarlo para que tenga el tamaño deseado. Una vez realizado se pasó al diseño e implementación de la textura que acompañará al escenario dándole el aspecto deseado, lo que constituye la segunda parte del desarrollo de esta pieza.

Durante esta segunda fase del diseño, se dibujó haciendo uso del programa de Windows Paint un rectángulo de (2500 x 2000 pixeles), para que encajase perfectamente con el diseño realizado en Blender y de esta manera fuese sencillo trabajar sobre él, al ser una escala de 1mm por pixel. Además se añadieron dos líneas blancas, que delimitan la parte del escenario donde se sitúan los obstáculos, de la parte donde se situará la salida y la llegada parcial. Una vez dibujada la textura se procedió con su exportación al modelo Blender como se puede observar en la figura 6.13. expuesta a continuación.

Se comenzó por seleccionar el plano y aplicarle un material liso sin ningún tipo de variación. También se seleccionó la opción “shadeless” o sin sombras, que elimina todos los efectos de luz sobre el plano dejando la textura uniforme y facilitando el manejo del objeto por parte del ordenador, ya que al no tener que manejar efectos de luz, se aceleraran de manera notable los procesados futuros.

Una vez definido el material se pasó a importar la textura anteriormente mencionada, dicha textura podía tener varios formatos diferentes aceptados todos ellos por Blender, como son jpg, png o bmp, pero el más óptimo a la hora de realizar trabajos futuros sobre la textura importada, una vez ubicado dentro del simulador OpenRave, era el formato .tif, por sus características.

El proceso de importación se basó en el modo UV de Blender, siendo este el método más sencillo, al permitir mapear las coordenadas en las que se quiere ajustar la imagen, para hacerla coincidir con la geometría definitiva en 2D.

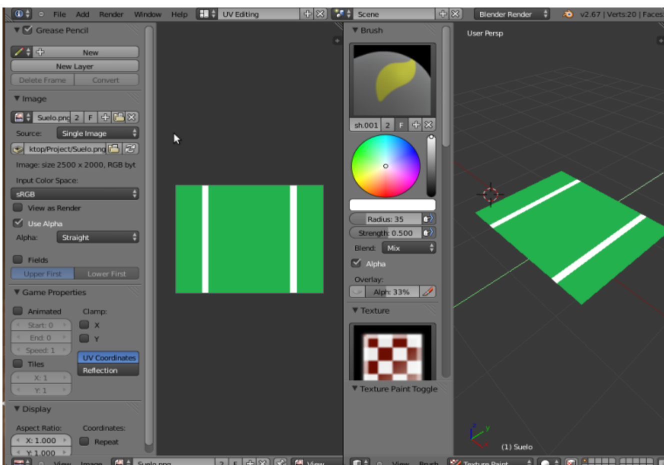


Figura 6.13. Modelo del suelo creado en Blender

Paredes

Las paredes representaban un problema al tener que realizar cuatro planos separados, que posteriormente formasen un conjunto perfectamente ensamblado, por lo que se escogió otra opción diferente a la de su modelado individual, en la que se produjo el conjunto de las cuatro paredes como una sola pieza. Para ello se insertó un cubo rectangular y se hizo coincidir en medidas con el suelo previamente realizado (2,5 x 2 metros), además se le dio una altura de 0,5m en concordancia con lo expuesto en el reglamento.

Una vez definido el cubo se eliminaron la cara superior e inferior, obteniéndose la figura deseada mostrada en la figura 6.14., que representa las cuatro paredes en una sola pieza. Una vez definida se le aplico un material liso y sin efectos de luz de manera análoga al suelo, posteriormente en vez de seleccionar una imagen como textura, se le asignó un color gris estándar para contrastar con la textura aplicada al suelo.

Una vez terminado el modelo de la pieza se ubicó de manera que fuese totalmente coincidente con el suelo, a lo largo de sus vértices y aristas, tomando el aspecto de una sola pieza con diferentes texturas, que imita perfectamente el escenario real en cuanto a medidas y a apariencia, con la salvedad de los obstáculos que serán incorporados a posteriori.

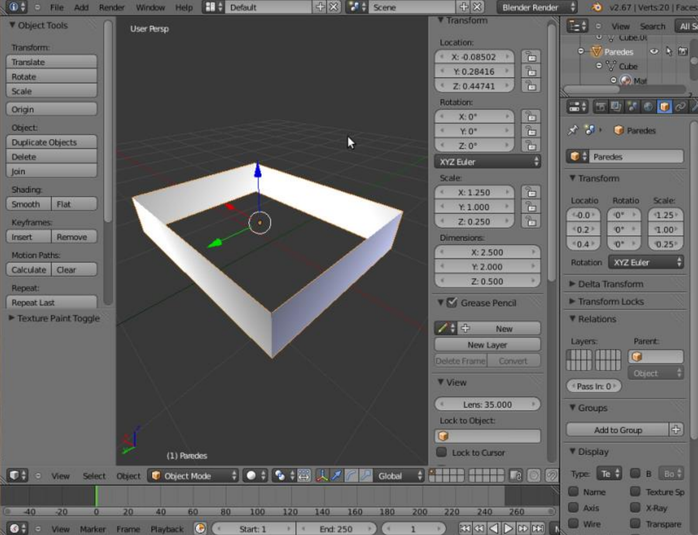


Figura 6.14. Modelo de las paredes creado en Blender

Obstáculos

Por último se diseñó uno de los obstáculos, insertando un cubo y redimensionándolo hasta el tamaño adecuado descrito en la normativa de la competición (20 x 20 x 50 centímetros), el material y textura utilizados en dicho obstáculo, son los mismos seleccionados para las paredes como se puede observar en la figura 6.15., manteniendo sus propiedades para diferenciarse del suelo, pero integrándose perfectamente en consonancia con las paredes del escenario.

Una vez se encontraba definido uno de los obstáculos, se copió para formar cinco nuevos obstáculos del mismo tamaño y características, con lo que ya se podían simular todo tipo de escenarios, donde realizar las pruebas necesarias de cara a la competición, en la que el escenario no es conocido a priori sino que son los jueces los que sitúan dichos obstáculos antes del inicio de la prueba.

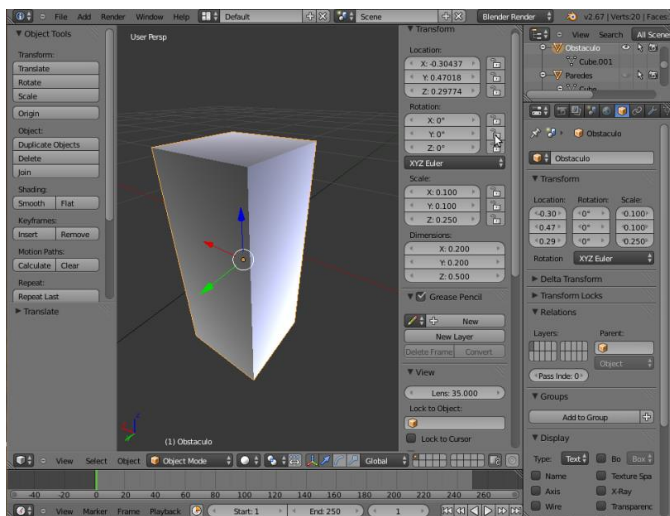


Figura 6.15. Modelo de un obstáculo creado en Blender

Conjunto final

Una vez definidos todos los elementos del escenario y aplicadas todas las texturas, se procedió colocando los obstáculos en las 6 configuraciones diferentes como se puede observar en la figura 6.16.. que se recogen a modo de ejemplo, dentro de la normativa de la competición CEABOT incluida como Anexo 1, ya que en ellas se muestran casos bastante variados, ante los que se puede enfrentar el robot durante el desarrollo de la competición.

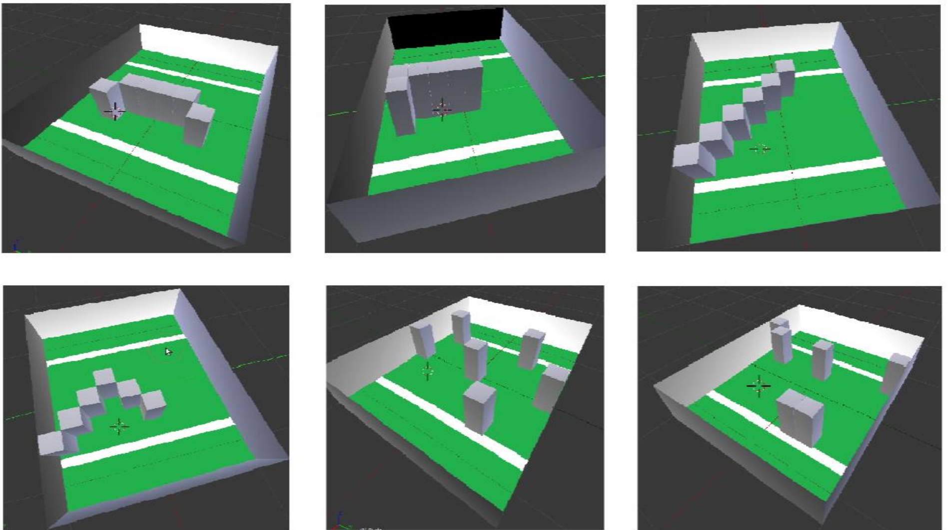


Figura 6.16. Diseño de los 6 escenarios de ejemplo de la competición CEABOT

Importación del escenario a OpenRave

Una vez implementados los escenarios en Blender, era necesario importarlos al entorno de simulación OpenRave, en el que se realizarán las pruebas del controlador definido en Matlab-Simulink a través de una conexión YARP.

La primera de las acciones a realizar era la de exportarlo en formato .wrl o .iv desde Blender, al ser los formatos de modelos que soporta OpenRave, en este caso se optó por el formato .wrl, ya que Blender solo soportaba la exportación en formato .iv hasta la versión 4.4.x y en nuestro caso se estaba utilizando la versión 4.6.2. Hubo que seleccionar como preferencias de usuario dentro de Blender la opción de exportar en dicho formato, como se muestra en la figura 6.17. expuesta a continuación.

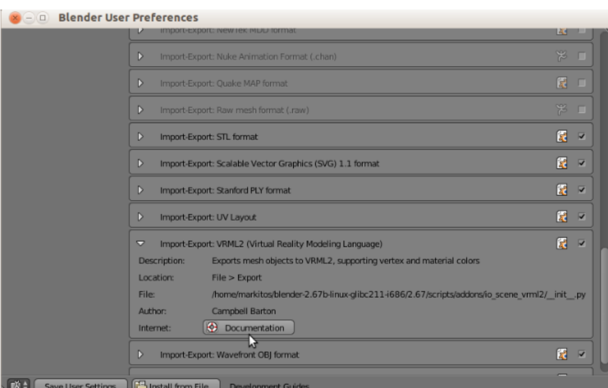


Figura 6.17. Preferencia a seleccionar para habilitar la exportación como .wrl

Una vez exportado en formato .wrl, se procedió a crear un nuevo archivo XML (CEABOT.xml), que es el mostrado en la figura 6.18., en dicho archivo, se incluyó el escenario creado, para ello hubo que definir un nuevo <Kindboody>. Además de configurar varios parámetros hasta conseguir que el archivo .wrl realizara la función de escenario, para la cual había sido diseñado.

```
-<KinBody name="escenario">
   <!--modelsdir>models</modelsdir-->
 -<Body type="static">
   -<Geom type="trimesh">
      <!--render> Blender/escenario.wrl</render-->
      <render>models/Bioloid/Escenarioconf2.wrl</render>
    -<!--
        <collision>models/Bioloid/escenario.wrl</collision>
      -->
      <extents> 1 1 0.001</extents>
      <!--<Translation>0.40000        -0.3000  0.7</Translation>-->
      <Translation>0 -0.3 0.9</Translation>
      <RotationAxis>0 1 0 -90</RotationAxis>
    </Geom>
  </Body>
</KinBody>
```

Figura 6.18. Archivo CEABOT.xml

Los parámetros que hubo que configurar una vez creado el archivo XML fueron, en primer lugar hubo que incluir como <render> el escenario creado en Blender, para ello se ha de incluir la ruta en la que se encuentra, en este caso /models/Bioloid, seguido del nombre del archivo .wrl. En este caso se hace referencia al escenario con la segunda configuración, pero se podría referenciar cualquiera de las 6 configuraciones de ejemplo, cambiando simplemente el digito del final por el de la configuración deseada. Además se configuro como un escenario “static” para que no se viese afectado por la gravedad incluida en el simulador, con una geometría “trimesh”. Por último se configuraron los parámetros de rotación y translación hasta obtener la ubicación deseada.

Una vez incluido y configurado dentro del archivo XML, surgió el problema con la textura .tif que se creó para el suelo, ya que no se mostraba correctamente dentro del escenario en OpenRave como cabía esperar, el problema era que no se encontraba bien referenciada dentro del archivo .wrl. Ya que la forma en que Blender referencia por defecto las texturas dentro de los archivos .wrl, mostrada en la figura 6.19., no era correcta para ejecutarla dentro del entorno de simulación OpenRave.

```
# 'Suelo'
Shape {
        appearance Appearance {
                texture ImageTexture {
                        url [ "/home/markitos/Desktop/Project1/Suelo.png" "Suelo.png" "/home/markitos/Desktop/Project1/
Suelo.png" ]
                }
```

Figura 6.19. Referencia a la textura por defecto en el archivo .wrl

Por lo que hubo que modificar el archivo .wrl desde el editor de texto, cambiando la referencia como se muestra en la figura 6.20. expuesta a continuación, dicha forma de referenciar la textura es a modo de ejecutable, incluyéndola en la misma carpeta que se encuentran los escenarios Bioloid-Open/models/Bioloid, y en general todos los modelos .wrl que utiliza OpenRave, para cada una de las partes del robot.

```
# 'Suelo'
Shape {
        appearance Appearance {
                texture ImageTexture {
                        url ["./Suelo.tif"]
                }
        }
```

Figura 6.20. Modo correcto de referenciar texturas en el archivo .wrl

Una vez realizados estos cambios ya se visualizaba correctamente el escenario dentro de OpenRave, para lanzar el simulador incluyendo el escenario, nos hemos de situar dentro del directorio /Bioloid-Open, e introducir el comando (openrave CEABOT.xml), lo que lanzara el escenario dentro del simulador, en este caso se trata del escenario con la segunda configuración de ejemplo, como se muestra en la figura 6.21. expuesta a continuación.

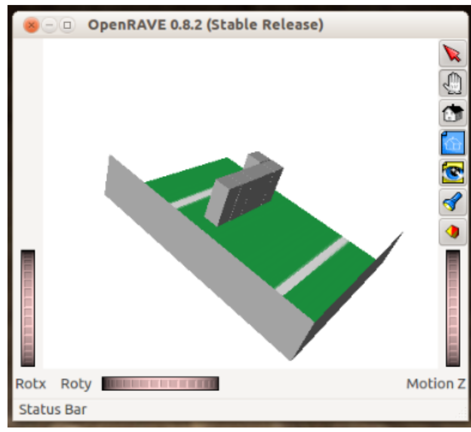


Figura 6.21. Ejecución del archivo CEABOT.xml dentro de OpenRave

Por ultimo para poder mostrar el escenario conjuntamente con el robot, hubo que referenciarlo desde el archivo XML (Bioloid.env.xml), como se muestra en la figura 6.22. mostrada a continuación, ya que es dicho archivo, el que se encarga de crear un entorno de simulación, incluyendo el robot y el escenario, haciendo uso de la etiqueta <enviroment> de apertura y </enviroment> para su cierre.

```
-<KinBody name="floor" file="CEABOT.xml">
  -<Body type="static">
     <RotationAxis>1 0 0 90</RotationAxis>
     <Translation>0 0 -0.015</Translation>
   -<Geom type="box">
       <extents>2 0.005 2 </extents>
       <diffuseColor>.3 1 .3</diffuseColor>
       <ambientColor>0.3 1 0.3</ambientColor>
     </Geom>
   </Body>
 </KinBody>
```

Figura 6.22. Extracto del archivo bioloid.env.xml, referencia al escenario

Finalmente, una vez configurado dentro del archivo bioloid.env.xml, la referencia al escenario, con sus respectivos colores de ambiente, sus propiedades de translación y rotación y su tipo de geometría como “box”, de tipo “static”. Se pudo hacer la referencia simple al archivo CEABOT.xml como se muestra en la figura 6.23. dentro del que será el simulador final, bioloid.xml.

```
-<Robot name="Humanoid_Robot_Bioloid">
  -<KinBody file="Bioloid_Trunk.xml">
     <KinBody file="CEABOT.xml"/>
     <KinBody file="Bioloid_Arm_right.xml"/>
```

Figura 6.23. Extracto del archivo bioloid.xml donde se referencia el escenario

En el archivo del simulador final bioloid.xml, en que se incluyen ya los sensores y actuadores y que se lanzará haciendo uso del CartesianServer encargado de manejar todos estos elementos a través del comando “cartesianServer - - env bioloid-open/bioloid.xml - - numMotors 23”. Finalmente se obtuvo el resultado deseado como se muestra en la figura 6.24. expuesta a continuación.

Figura 6.24. Entorno de simulación ejecutando el archivo bioloid.xml final

6.1.2.2. Incorporación de las propiedades físicas al entorno de simulación

Para que el entorno de simulación sea totalmente realista, se han de incluir las propiedades físicas que se encuentran el mundo, ya que de otra manera las simulaciones realizadas no tendrían valor, y cuando se descargase el código generado al controlador real el robot no actuaría como cabría esperar. Por tanto en el desarrollo de este apartado se abordara esta problemática y la manera de resolverla dentro del entorno de simulación.

Cuando se introdujo por primera vez el robot dentro del entorno de simulación en el que ya se había colocado el escenario, se observaron varios comportamientos anómalos, que había que corregir. En primer lugar el robot flotaba como se muestra en la figura 6.25., este hecho se debía a la falta de propiedades físicas que caracterizasen el simulador como un entorno real.

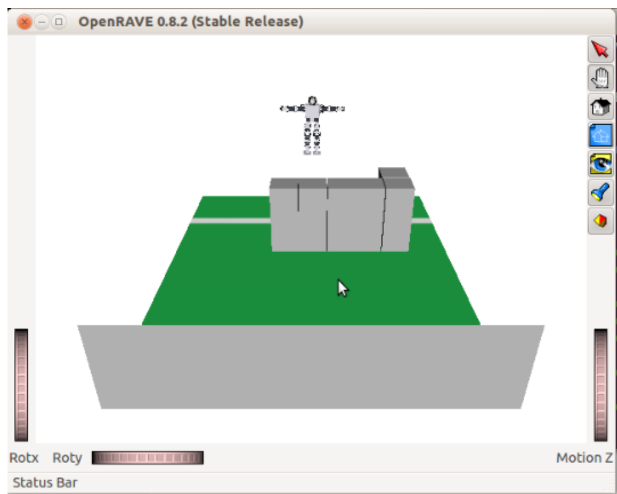


Figura 6.25. Entorno de simulación sin propiedades fisicas

El primer punto a analizar dentro de este apartado era la forma en que se definían los objetos (<body>) dentro del simulador OpenRave, ya que existen dos posibilidades, con la primera de ellas declarando los cuerpos (<body>), introducidos como “static”, dichos cuerpos no se ven afectados por las fuerzas como la gravedad, ni por ningún otro tipo de fuerza por lo que no se puede dotar a estos objetos de ninguna clase de movimiento. Mientras que si los objetos son declarados como de tipo “dynamic”, se ven afectados por las fuerzas introducidas y además, se les pueden aplicar otro tipo de fuerzas, como las generadas por los servomotores para dotarlos de movimiento.

Del párrafo anterior deriva la necesidad de declarar dentro del simulador objetos de diferentes tipos, en concreto el robot será declarado como “dynamic”, por la necesidad de moverlo durante la simulación, mientras que el escenario será declarado como “static”, para que permanezca totalmente inmóvil y pueda ser recorrido por el robot sin hundirse.

Ambos tipos de declaración se pueden observar en la figura 6.26. donde se muestra por un lado la declaración del escenario, dentro del código del archivo bioloid.env.xml, como “static”, para que no se vea afectado por ningún tipo de fuerza. Y por otro lado la declaración dentro del archivo Bioloid_Arm_Right.xml, en el que se declaran como “dynamic” todas las piezas que componen el brazo derecho del robot, para que se pueda mover adecuadamente dentro del simulador. De manera análoga a la declaración del brazo, todo el resto de pieza que forman el robot, como son las piernas, la cabeza, etc. Han sido declaradas como “dynamic”.

```
- <KinBody name="floor" file="CEABOT.xml">
  - <Body type="static">
      <RotationAxis>1 0 0 90</RotationAxis>
      <Translation>0 0 -0.015</Translation>
    - <Geom type="box">
        <extents>2 0.005 2 </extents>
        <diffuseColor>.3 1 .3</diffuseColor>
        <ambientColor>0.3 1 0.3</ambientColor>
      </Geom>
    </Body>
  </KinBody>
```

```
- <KinBody name="rightArm">
    <modelsdir>models</modelsdir>
    <Body name="Torso_dummy1" type="dynamic"> </Body>
  - <Body name="ras" type="dynamic">
      <offsetfrom>Torso_dummy1</offsetfrom>
      <Translation>-0.076500 0.006 0.000000</Translation>
    - <Geom type="box" render="false">
        <Extents>0.01700 0.01250 0.02200</Extents>
      </Geom>
      <RotationAxis>1 0 0 -90</RotationAxis>
      <!--GIRAR -90° EL BRAZO-->
      <Translation>0.00 0.015 0.015</Translation>
      <!--MOVER EN EL EJE Y ^ Z-->
```

Figura 6.26. Declaración de cuerpos estáticos y dinámicos dentro de OpenRave

Una vez definidos los cuerpos con su tipo correcto, basándose en su funcionalidad, se pasó a incorporar las leyes físicas necesarias para que el simulador reprodujese la realidad que el robot se encontraría en la naturaleza. Dicha realidad aunque puede variar ligeramente en función del lugar del mundo en el que nos encontremos, es una fuerza de atracción generada por la tierra, que provoca una aceleración de 9,8m/s² en los cuerpos que se encuentran sobre ella.

Para reproducir esta fuerza se incluyó dentro del entorno <enviroment>, un motor físico <physicsengine>, que simula las fuerzas existentes en la naturaleza y al que se le pueden asignar diferentes tipos y parámetros para configurarlo.

En nuestro caso concreto se optó por incluir un motor físico de tipo “ode” (Open Dynamics Engine) [33], el cual está definido en librerías de C/C++, este motor físico simula las leyes dinámicas del solido rígido, y además es capaz de detectar las colisiones entre los diferentes cuerpos dentro del simulador.

Para introducir el motor físico hay que tener en cuenta que tiene que situarse dentro de las etiquetas <enviroment>, por tanto ha sido introducido en el archivo bioloir.env.xml, las propiedades <odeproperties>, que hubo que configurar hasta obtener el comportamiento deseado fueron las mostradas en la figura 6.27. mostrada a continuación.

```
- <physicsengine type="ode">
  - <odeproperties>
      <friction>0.5</friction>
      <gravity>0 0 -9.8</gravity>
      <selfcollision>1</selfcollision>
      <dcontactapprox>1</dcontactapprox>
      <cfm>0.1</cfm>
      <erp>0.9</erp>
    </odeproperties>
  </physicsengine>
```

Figura 6.27. Definicion de propiedades fisicas dentro del archivo bioloid.env.xml

De las propiedades mostradas en la figura 6.27., cabe destacar la propiedad <gravity> que como se puede observar fue configurada para que se aplicase una acelaración negativa de -9,8m/s², en el eje z. Además de las propiedades de >selfcollision> y <dcontactapprox> que se encargan de detectar y prevenir las colisiones entre objetos. Una vez configurados todos los parametros mostrados anteriormente, se consiguio que el simulador representase de una manera realista el escenario de la competición CEABOT, incluyendo las fuerzas que actuan en un entorno real como se muestra en la figura 6.28. mostrad a continuación en la que se puede observar, como se apoya el robot sobre el suelo del escenario sin traspasarlo.

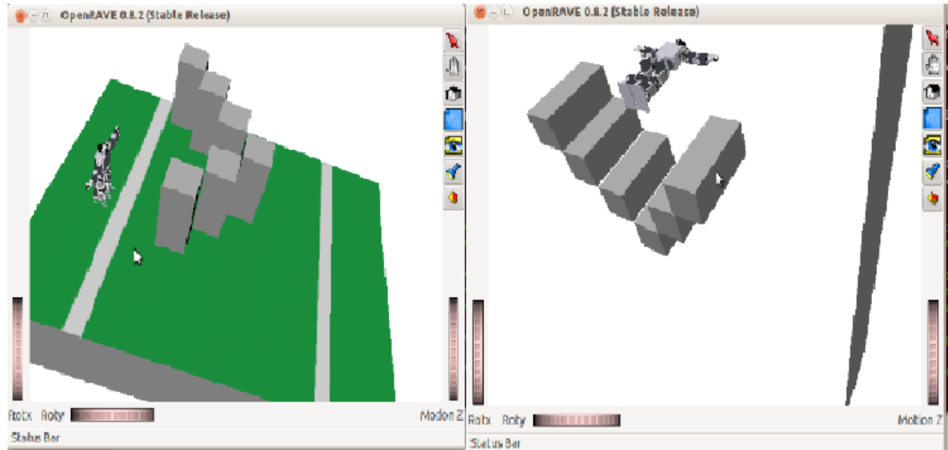


Figura 6.28. Simulador incluyendo las leyes fisicas

6.1.2.3. Incorporación de sensores al robot Bioloid simulado

Una vez se encontraba creado el entorno de simulación completo, en el que se incluía el robot con sus servomotores, además del escenario de la competición CEABOT y todo ello incluyendo las leyes físicas que imitaban las fuerzas presentes en la naturaleza. Solo faltaba la forma en que íbamos a interactuar con el entorno, para conocer donde nos encontrábamos y como era el escenario que nos rodeaba.

Era por tanto el momento de integrar los sensores al robot. Se comenzó realizando un estudio de los sensores implementados para a OpenRave [34], dichos sensores eran varios, entre los que se encontraban cámaras, sensores laser, encoders digitales, IMUs, etc. Además existe la posibilidad de crear tus propios sensores personalizados, para ello habrá que crear de manera análoga a como se han creado cada parte que componen el robot, su forma física <body> y su unión al resto del robot <joint>, y una vez creado dotarle de los atributos deseados en función del tipo de sensor.

En nuestro caso se han integrado 3 sensores laser, ya que solo se necesitaba detectar el lugar en el que se encuentran los obstáculos y a que distancia exacta están situados, por tanto con estos sensores era suficiente, pero siguiendo los pasos que se describen a continuación se podrán integrar todos los sensores deseados, del tipo que sea necesario para las diferentes pruebas de las que consta el campeonato.

Existían dos tipos de sensores laser, uno que detectaba en 3D y otro en 2D, cada uno de ellos con sus propias características. Se probó con ambos tipos de sensores, en primer lugar se optó por los sensores 3D, situando tres de ellos en el robot como se muestra en la figura 6.29., uno para la parte central, y uno más por cada uno de los laterales (izquierda y derecha)

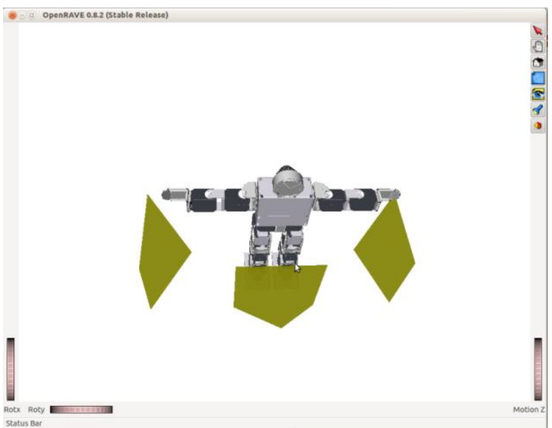


Figura 6.29. Robot con sensores Laser de 3D

Para conseguir la configuración mostrada, hubo que modificar las propiedades de <rotationaxis> y <translation> hasta conseguir la ubicación deseada, una vez unido al torso del robot gracias a la etiqueta <link>. Una vez conseguida la ubicación deseada, se configuro su rango máximo <maxrange> su tiempo de muestreo, ancho, alto, color, etc. como se puede observar en la figura 6.30. expuesta a continuación.

```
- <AttachedSensor name="myflashlaserleft">
    <link>Torso_dummy1</link>
    <translation>-0.2 -0.2 0</translation>
    <rotationaxis>0 1 0 -90</rotationaxis>
  - <sensor type="BaseFlashLidar3D">
      <maxrange>5</maxrange>
      <scantime>0.2</scantime>
      <KK>32 24 32 24</KK>
      <width>64</width>
      <height>48</height>
      <color>1 1 0</color>
    </sensor>
  </AttachedSensor>
```

Figura 6.30. Declaración de sensores Laser 3D

Tras probar con los sensores laser de 3D, se decidio que no resultaba necesario obtener informacion en 3D, sino que solo era necesario obtener datos en 2D, para conocer la distancia a la que se encontraba cada uno de los obstaculos por la derecha, izquierda y frente, basandonos en el tiempo de vuelo de la señal laser. Fue entonces cuando se cambiaron los sensores laser 3D por sensores laser 2D, como se muestra en la figura 6.31., donde se muestra el robot con dichos sensores integrados.

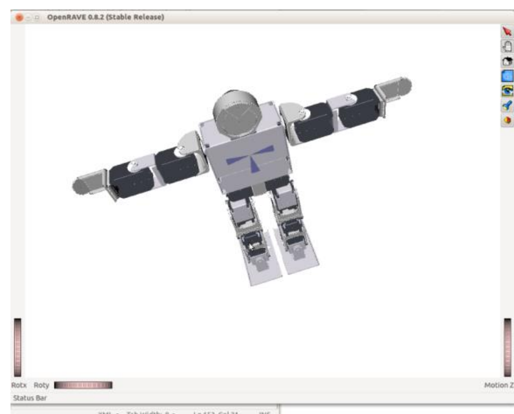


Figura 6.31. Robot con sensores Laser de 2D

Una vez se habían integrado los tres sensores en el robot dentro del archivo bioloid.xml, ya que será el que se ejecute finalmente, representando al simulador completo. Hubo que configurar sus parámetros para obtener el resultado deseado. En primer lugar se situaron los sensores unidos al torso del robot, todos ellos en el mismo punto, utilizando las etiquetas <link> y <translation> como se puede observar en la figura 6.32., obteniéndose un único punto focal desde el que parten los tres rayos.

```
− <AttachedSensor name="mylaserfront">
    <link>Torso_dummy1</link>
    <translation>0 0.06 0.08</translation>
    <rotationaxis>0 0 1 90</rotationaxis>
    <rotationaxis>1 0 0 -90</rotationaxis>
  − <sensor type="BaseLaser2D" args="">
      <minangle>179.5</minangle>
      <maxangle>180.5</maxangle>
      <resolution>0.05</resolution>
      <maxrange>5</maxrange>
      <scantime>10</scantime>
    </sensor>
  </AttachedSensor>
− <AttachedSensor name="mylaserleft">
    <link>Torso_dummy1</link>
    <translation>0 0.06 0.08</translation>
    <rotationaxis>0 0 1 90</rotationaxis>
  − <sensor type="BaseLaser2D" args="">
      <minangle>-90.5</minangle>
      <maxangle>-89.5</maxangle>
      <resolution>0.05</resolution>
      <maxrange>5</maxrange>
      <scantime>10</scantime>
    </sensor>
  </AttachedSensor>
− <AttachedSensor name="mylaserright">
    <link>Torso_dummy1</link>
    <translation>0 0.06 0.08</translation>
    <rotationaxis>0 0 1 90</rotationaxis>
  − <sensor type="BaseLaser2D" args="">
      <minangle>89.5</minangle>
      <maxangle>90.5</maxangle>
      <resolution>0.05</resolution>
      <maxrange>5</maxrange>
      <scantime>10</scantime>
    </sensor>
  </AttachedSensor>
</Robot>
```

Figura 6.32. Definición de los tres sensores Laser de 2D

Una vez se encontraban integrados y colocados los sensores dentro del robot, al situarse sus respectivas declaraciones entre las etiquetas <robot>, hubo que configurar la dirección a la que se deseaba que apuntaran, haciendo uso de las etiquetas <rotationaxis>. De esta manera se consiguió que cada sensor apuntase en la dirección deseada y a la altura apropiada (derecha, izquierda, frente).

En este momento se fijó su ángulo máximo y mínimo (<maxangle>, <minangle>), que en este caso se optó por tener tan solo 1° grado, al no necesitar conocer otro dato más que el tiempo de vuelo, no resultaba necesaria una amplitud mayor. Una vez configurados se pasó a determinar su resolución, tiempo de muestreo y rango máximo (<resolution>, <maxrange>, <minrange>), quedando finalmente definidos para cumplir con las especificaciones deseadas como se puede observar en la figura 6.32. expuesta anteriormente.

Tras todos las configuraciones analizadas anteriormente, finalmente se lanzó el simulador obteniéndose el resultado que se muestra en la figura 6.33., siendo este resultado el deseado ya que como se observa, los tres sensores representados por los rayos y puntos en azul, apuntan en las direcciones deseadas, con un rango mínimo y una resolución muy alta detectando la distancia a los obstáculos de manera muy precisa, una vez el rayo emitido choca con el obstáculo y regresa (tiempo de vuelo).

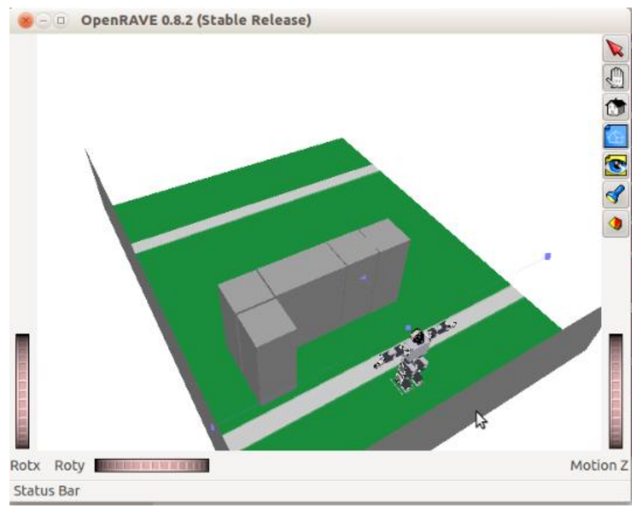


Figura 6.33. Simulador final con los sensores integrados y funcionando

6.1.3. Middleware de comunicación

En este apartado, se describirá como se ha resuelto el problema de la comunicación entre los dos componentes principales (controlador, simulador). Una vez ambos componentes estaban diseñados e implementados, surgió el problema de realizar un intercambio de datos entre ellos, ya que dentro del simulador, se producía la lectura de los sensores, que iban a determinar la forma de actuar dentro del control (máquina de estados), la cual en función de la salida que produjera debería enviar un valor u otro a los actuadores dentro del simulador.

Para realizar este intercambio de datos, se decidió utilizar YARP, ya que con él se podían crear puertos tipo TCP/IP, de ambos componentes e intercambiar datos entre ellos. YARP, permite crear puertos de lectura, puertos de escritura o bien puertos de lectura/escritura. Para que estos puertos puedan ser creados se ha de crear en primer lugar un servidor (YARP server) como se muestra en la figura 6.34., que se encargue de realizar todas las comunicaciones entre los diferentes puertos.

```
markitos@markitos-XPS-M1530: ~
markitos@markitos-XPS-M1530:~$ yarp server

Call with --help for information on available options
Options can be set on command line or in /home/markitos/.config/yarp/yarpserver.
conf
Using port database: ports.db
Using subscription database: subs.db
If you ever need to clear the name server's state, just delete those files.

IP address: default
Port number: 10000
yarp: Port /root active at tcp://192.168.1.131:10000

Registering name server with itself:
 * register "/root" tcp "192.168.1.131" 10000
   + set "/root" ips "127.0.0.1" "192.168.1.131" "::1" "fe80::221:5cff:fe8b:c819
%3"
   + set "/root" process 7131
 * register fallback mcast "224.2.1.1" 10000
```

Figura 6.34. Creación de servidor YARP

Una vez creados los diferentes puertos se han de crear "bottles" que se encargarán de encapsular los datos que se deseen enviar, realizando esta operación de transmisión de datos de una manera simple entre los diferentes puertos creados, enviando varios datos de una sola vez sin necesidad de ir enviando cada dato de manera individual.

Una vez descrito el proceso de comunicación que se lleva a cabo dentro de la comunicación del middleware YARP, se dividirá el estudio de dicha comunicación en dos vertientes, la primera de ellas será la parte que comprende al simulador en OpenRave, en la que se estudiará la forma de recibir los datos del controlador, así como la forma de enviar los datos a través de YARP a ese controlador. En segundo lugar se estudiara la parte que comprende al controlador en Matlab-Simulink, en la que se analizará cómo se envían los datos a los servomotores y como se reciben y se procesan los datos de los sensores para que sean interpretados por la máquina de estados.

6.1.3.1. Comunicación YARP - OpenRave - C++

Para realizar la comunicación entre el simulador y el middleware, se utilizó una herramienta creada con YARP, por Juan uno de los investigadores, dentro de la Asociación de Robótica, dicha herramienta, fue diseñada para el proyecto ASIBOT [35], mostrado en la figura 6.35. y desarrollado dentro de esta Asociación, por lo que para utilizar la herramienta, en primer lugar se ha de instalar todo el software diseñado para ASIBOT, desde el enlace mostrado en la referencia [36], en concreto dentro de todo el desarrollo que se instala, solo utilizaremos el CartesianServer para realizar las comunicaciones con OpenRave, dentro de nuestro proyecto.

Figura 6.35. Proyecto ASIBOT de la Asociación de Robótica

Una vez instalado el CartesianServer en nuestro equipo, hubo que realizar algunos cambios para que se adaptase a nuestro robot mini-humanoide, para realizar los cambios adecuados nos basamos en la información proporcionada en el enlace de la referencia [37], donde se detalla cómo realizar cambios para que este servidor YARP se adapte a todo tipo de robots.

Para el control de los servomotores, no hubo que realizar ningún cambio ya que dentro del CartesianServer, se contempla la opción de manejar varios <joint>, que en nuestro caso son los declarados como <manipulator> dentro del archivo bioloid.xml. La distribución de los servomotores en nuestro modelo de robot Bioloid es la mostrada en la figura 6.36., por lo que aunque el robot solo posee 18 servomotores, uno por cada uno de sus grados de libertad, al estar distribuidas las IDs de la manera mostrada a continuación, se debía iniciar el servidor para 23 motores. Esto no supuso un problema ya que el CartesianServer soporta un máximo de 33 motores.

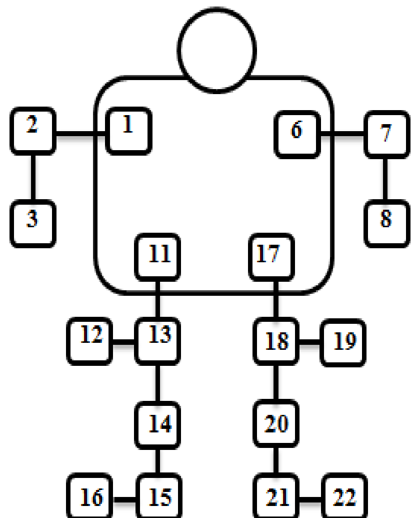


Figura 6.36. Distribución de las IDs de los servomotores en el modelo de Bioloid

Para controlar los movimientos de los 18 servomotores del robot, el CartesianServer crea las 23 instancias dentro de su servidor todas ellas englobadas dentro de la clase “control board”, como se muestra en la figura 6.37., siendo controlable su ángulo objetivo (posición del servomotor a la que se desea llegar), así como la velocidad a la que dicho servomotor debe moverse hasta esa posición objetivo. Para enviar este comando el CartesianServer crea los puertos tipo YARP movjCartesianServer, como se observa en la figura 6.37., gracias a él además podemos conocer el estado en que se encuentran dichos servomotores.

```
yarp: created wrapper <controlboard>. See C++ class ServerControlBoard for docum
entation.
[CartesianBot] success: Acquired robot interfaces.
Server control board starting
[CartesianServer] success: Acquired cartesian interface.
yarp: Port /ravebot/movjCartesianServer/rpc:o active at tcp://192.168.1.131:1000
9
yarp: Port /ravebot/movjCartesianServer/command:o active at tcp://192.168.1.131:
10010
yarp: Port /ravebot/movjCartesianServer/state:i active at tcp://192.168.1.131:10
011
```

Figura 6.37. Inicialización del CartesianServer para 23 motores

Una vez resuelta la forma de comunicarse con los servomotores, hubo de resolverse la lectura de los sensores, para lo cual hubo que modificar la clase Sensor dentro del servidor, ya que hasta este momento no se habían incorporado sensores laser de 2D, dentro del servidor. Dentro del servidor existía la opción de leer cámaras, encoders, sensores laser de 3D, etc. Pero no se había trabajado con sensores de 2D, por lo que basándonos en los sensores 3D definidos, se modificaron sus atributos hasta obtener la lectura de los sensores 2D deseada.

Para dicha lectura de los sensores, se crean dentro del servidor, tres puertos de lectura (lasersensorright, lasersensorleft, lasersensorfront), como se muestra en la figura 6.38. en concordancia con los nombres asignados a cada uno de los sensores, dentro del archivo bioloid.xml en el que fueron definidos.

```
yarp: Port /ravebot/Humanoid_Robot_Bioloid_mylaserfront/depth:o active at tcp://
192.168.1.131:10022
Sensor 1 name: Humanoid_Robot_Bioloid_mylaserleft
Sensor 1 supports ST_Laser.
yarp: Port /ravebot/Humanoid_Robot_Bioloid_mylaserleft/depth:o active at tcp://1
92.168.1.131:10023
Sensor 2 name: Humanoid_Robot_Bioloid_mylaserright
Sensor 2 supports ST_Laser.
yarp: Port /ravebot/Humanoid_Robot_Bioloid_mylaserright/depth:o active at tcp://
192.168.1.131:10025
```

Figura 6.38. Puertos creados para cada uno de los sensores

Para testear el buen funcionamiento de todos los elementos dentro del simulador, se ha de ejecutar dicho simulador a través del comando que conecta el archivo bioloid.xml con el CartesianServer “cartesianserver - -env bioloid-open/bioloid - -numMotors 23”, gracias a dicho comando, se lanza el simulador, como se muestra en la figura 6.39., creándose los puertos de lectura y escritura, con los que se trabajara en los diferentes sensores y actuadores del robot simulado.

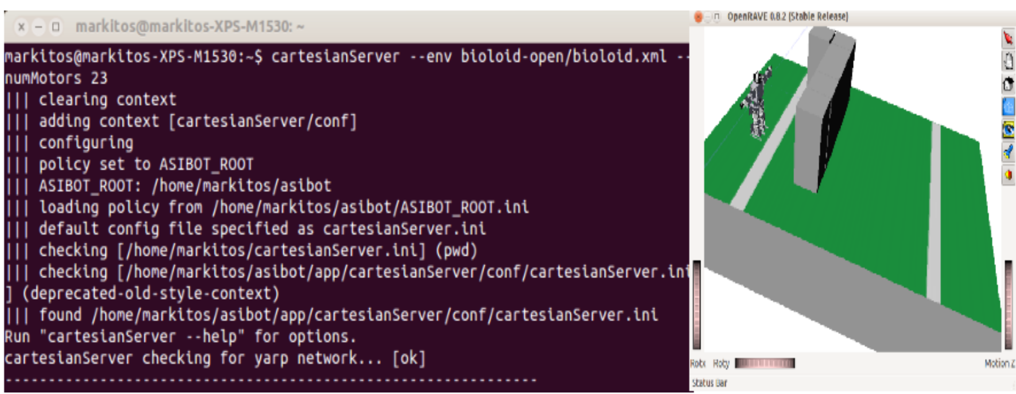


Figura 6.39. Apertura del simulador a través del CartesianServer

A modo de prueba, se comenzó enviando velocidades y posiciones a algunos de los servomotores con un pequeño ejemplo creado en C++, dicho ejemplo mostrado en la figura 6.40. es el “testRemoteRaveBot.cpp”, dicho ejemplo se encontraba dentro de los instalados conjuntamente con ASIBOT, y fue modificado para que actuase de manera correcta con nuestro robot, y con los motores deseados. En concreto con este ejemplo se pretendía mover el brazo derecho y la pierna derecha a modo de demostración. Por lo que se han usado los servomotores (1, 2, 3 para el brazo y el 13, 14, 15 para la pierna). Se les han definido los ángulos objetivos además de la velocidad a la que deben de alcanzarlos como se puede observar a continuación.

```
printf("test positionMove(14,90)\n");
pos->positionMove(14, 90);

printf("Delaying 5 seconds...\n");
Time::delay(5);

IEncoders *enc;
ok = dd.view(enc);

IVelocityControl *vel;
ok = dd.view(vel);
vel->setVelocityMode();

printf("test velocityMove(1,10)\n");
vel->velocityMove(1,10);
```

Figura 6.40. Ejemplo testRemoteRaveBot modificado para Bioloid

Los comandos enviados desde el programa creado en C++ mostrados en la figura 6.40. anteriormente, provocan el movimiento en el robot y además muestran por pantalla dentro de la consola de comandos, cuales son los valores que se están enviando como se puede observar en la figura 6.41.. Por tanto gracias a este ejemplo se pudo comprobar la funcionalidad del simulador, y la correcta comunicación con el servidor, en la que el servidor crea un puerto de lectura, desde el que maneja los datos de posición y velocidad recibidos, en función de la ID del servomotor, dentro del rango deseado.

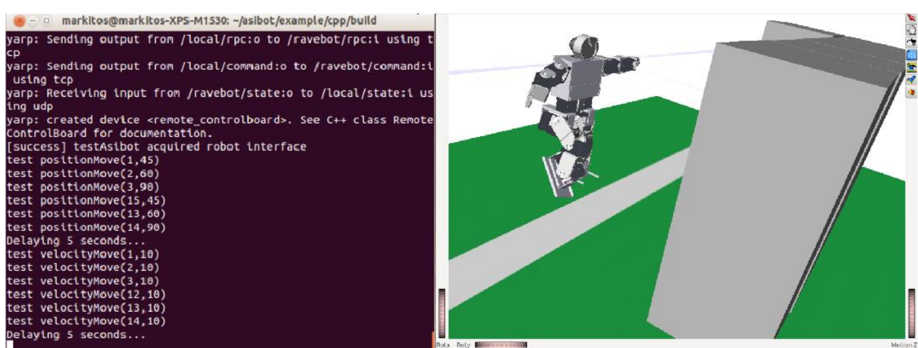


Figura 6.41. Test de movimientos desde el ejemplo creado en C++

Además de los servomotores también se testeo el buen funcionamiento de los sensores, para lo cual se lanzó en primer lugar el CartesianServer, modificado y ligado a nuestro archivo bioloid.xml como se mostró anteriormente. Una vez lanzado se creó un puerto de lectura denominado "readsensors", a través del comando (yarp read /readsensor). Una vez creado se conectó con el puerto de escritura de uno de los sensores, en este caso con el de sensor situado en la parte frontal para comprobar su funcionamiento. Todo este proceso se puede observar en la figura 6.42.

```
markitos@markitos-XPS-M1530: ~
markitos@markitos-XPS-M1530:~$ yarp connect /ravebot/Humanoid_Robot_Bioloid_myla
serfront/depth:o /readsensors
Success: Added connection from /ravebot/Humanoid_Robot_Bioloid_mylaserfront/dept
h:o to tcp://readsensors
markitos@markitos-XPS-M1530:~$
yarp: Sending output from /ravebot/Humanoid_Robot_Bioloid_mylaserfront/depth:o t
o /readsensors using tcp
```

Figura 6.42. Conexión entre puerto readsensors y puerto del sensor frontal

Conectando estos dos puertos como se muestra en la figura 6.42. se consiguió obtener la lectura del sensor en una nueva consola de comandos, que es en la que se creó el puerto de lectura (readsensors), como se muestra en la figura 6.43., por tanto esta operación se deberá repetir para cada uno de los sensores, en caso de que se desee obtener su lectura por pantalla.

```
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
50 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167
 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250
67 0 0 250 167 0 0 250 167 0 0 250 167 0 1 0 0}
```

Figura 6.43. Lectura de los sensores en el puerto readsensors

6.1.3.2. Comunicación YARP – OpenRave - Matlab

Una vez resuelta la comunicación con el simulador, se debía resolver la parte en la que se comunica a través de YARP, el controlador, para de esta manera poder intercambiar datos entre el simulador y el control. Con lo que se conseguirá la funcionalidad deseada del entorno de desarrollo y cada uno de sus compontes interactuando con el resto.

En primer lugar se comprobó de manera análoga a la mostrada en el apartado anterior para C++, la funcionalidad de la comunicación para enviar datos a los servomotores y recibir datos desde los sensores a través de YARP, con el simulador, pero esta vez desde Matlab, como se describe en los siguientes párrafos.

En primer lugar se ha de cargar las librerías de YARP dentro de Matlab, este paso es estrictamente necesario ya que sin él no se podrá crear la comunicación. Para cargar dichas librerías se ha de incluir dentro del programa el comando “LoadYarp”, como se puede observar dentro de la figura 6.44., en la que también se muestra el código a través del que se comprueba que la comunicación está realmente activa, dicha conexión se realiza con el “remote_controlboard”, mencionado en el apartado anterior.

```
LoadYarp;

options = yarp.Property;
options.put('device','remote_controlboard');
options.put('remote','/ravebot');
options.put('local','/matlab');
dd = yarp.PolyDriver(options);

if isequal(dd.isValid,1)
    disp '[success] robot available';
else

disp '[warning] robot NOT available, does it exist?';
end
```

Figura 6.44. Código .m para conectarse con el remote_controlboard

Como se observa en la figura 6.45., al iniciar el programa .m mostrado anteriormente, se cargan correctamente las librerías y se conecta con el robot simulado. Aunque aparece un warning al no encontrarse activo ningún programa “testRaveBot”. Una vez comprobado que la conexión esta activa, en el siguiente párrafo se analizara el ejemplo “testRaveBot”, creado para ASIBOT y modificado para funcionar de manera correcta con nuestro robot Bioloid simulado.

```
>> loadyarp1
WARNING: requires a running instance of RaveBot (i.e. testRaveBo
Yarp library loaded and initialized
[success] robot available
>> |
```

Figura 6.45. Consola de Matlab mostrando librerías cargadas y robot disponible

Una vez cargadas las librerías y comprobada la comunicación, se comenzó a comunicar datos a los servomotores a través de las funciones mostradas en la figura 6.46., las cuales han sido extraídas del ejemplo "testRaveBot" y modificadas para nuestro robot dentro del programa denominado "onemovement" en concreto, utilizando los mismos métodos y clases utilizadas para el ejemplo de C++, analizado en el apartado anterior pero en Matlab.

```
disp 'test positionMove(15,45) -> moves motor 14 (start count at mot
pos.positionMove(15,45);

disp 'test delay(5)';
yarp.Time.delay(5);

vel.setVelocityMode();  % use the object to set the device to veloci
disp 'test velocityMove(13,10) -> moves motor 13 (start count at mot
vel.velocityMove(13,10);
```

Figura 6.46. Método utilizado en Matlab para mover los servomotores

Para comprobar que esta comunicación funcionaba correctamente se ha de lanzar el CaresianServer ligado a nuestro archivo bioloid.xml, de manera análoga a la expuesta en el apartado anterior. Tras iniciar el simulador, se movió la pierna derecha a modo de ejemplo, en concreto los servos utilizados en este ejemplo fueron (13, 14 y 15). Levantando de esta manera la pierna como se puede observar en la figura 6.47. En la que además se muestra la salida obtenida en la consola de comandos de Matlab, donde se muestran los ángulos objetivo y las velocidades a las que se han de alcanzar dichos ángulos, comprobando que la comunicación es correcta.

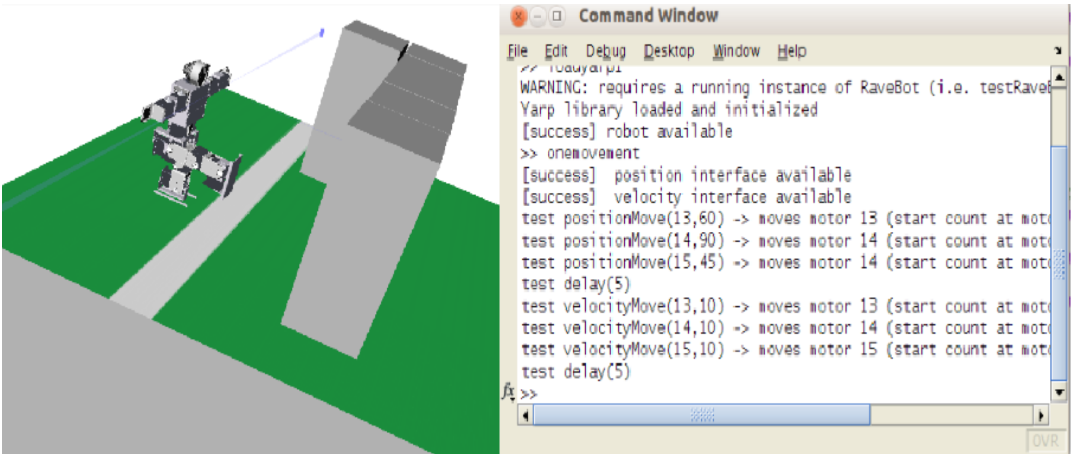


Figura 6.47. Test de movimientos creado en Matlab.

Una vez éramos capaces de enviar órdenes a los servomotores desde Matlab, el siguiente paso siguiendo la misma línea mostrada anteriormente, en el ejemplo creado en C++, era leer la información enviada desde los sensores integrados en el simulador OpenRave. Para lo cual una vez lanzado el simulador desde el CartesianServer y creados los puertos de escritura analizados en el apartado anterior, quedaba crear un puerto de lectura dentro de Matlab desde el que poder leer la información de los sensores.

Antes de crear el puerto de lectura, dentro del mismo código se importan todos los puertos YARP activos, además de las botellas con los datos de la comunicación, como se observa en la figura 6.48., ya que sin importarlos no seriamos capaces de trabajar con ellos dentro de Matlab.

```
import yarp.Port;
import yarp.Bottle;
```

Figura 6.48. Importación de puertos y botellas YARP

Una vez disponemos de todos los elementos para trabajar con ellos dentro de Matlab, procedemos a crear el puerto de lectura, en el que se recibirán los datos de los sensores en forma de botella. Al conectar dichos puertos, con el comando mostrado en la figura 6.49., además se comprueba que la conexión sea correcta, para evitar posibles pérdidas de datos, mostrando un mensaje por pantalla en el que se indica si la conexión es correcta o no, además de los datos recibidos.

```
port.open('/matlab/read');

if (yarp.Network.connect('/ravebot/Humanoid_Robot_Bioloid_mylaserfront,
disp('Frontsensor OK');
else
disp('Frontsensor not connect');
end
b=Bottle;
while(~done)
  port.read(b);

  %disp(b);
  disp(b.toString_c());

  if (strcmp(b.toString, 'quit'))
    done=1;
  end
end

port.close;
```

Figura 6.49. Código .m para la lectura del sensor frontal

A continuación se puede comprobar en la figura 6.50. que la lectura se realiza correctamente, ya que se muestran dentro de la consola de Matlab, los datos enviados desde el sensor frontal, de manera análoga a como se obtenían anteriormente en la consola de comandos, cuando se utilizaba el ejemplo creado en C++.

```
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
```

Figura 6.50. Lectura de los sensores en Matlab

Por ultimo cabe destacar que existe la posibilidad de comprobar si los la comunicación entre Matlab-YARP-OpenRave es correcta de una forma alternativa. Al haber iniciado Matlab desde la consola de comandos de Linux, esta registra todos los eventos que ocurre dentro del programa, registrando por tanto las comunicaciones externas que se producen, tanto de lectura como de escritura, como se puede comprobar en la figura 6.51., donde se muestra una captura de pantalla de la consola de comandos desde la que se abrió Matlab.

```
markitos@markitos-XPS-M1530: ~
markitos@markitos-XPS-M1530:~$ ./matlab
/usr/local/MATLAB/R2012a/bin/util/oscheck.sh: line 177: /lib/libc.so.6: No such
file or directory
yarp: Port /matlab/rpc:o active at tcp://192.168.1.131:10019
yarp: Port /matlab/command:o active at tcp://192.168.1.131:10020
yarp: Port /matlab/state:i active at tcp://192.168.1.131:10021
yarp: Sending output from /matlab/rpc:o to /ravebot/rpc:i using tcp
yarp: Sending output from /matlab/command:o to /ravebot/command:i using tcp
yarp: Receiving input from /ravebot/state:o to /matlab/state:i using udp
yarp: created device <remote_controlboard>. See C++ class RemoteControlBoard for
 documentation.
yarp: Port /matlab/read active at tcp://192.168.1.131:10004
yarp: Receiving input from /ravebot/Humanoid_Robot_Bioloid_mylaserleft/depth:o t
o /matlab/read using tcp
yarp: Receiving input from /ravebot/Humanoid_Robot_Bioloid_mylaserright/depth:o
to /matlab/read using tcp
yarp: Receiving input from /ravebot/Humanoid_Robot_Bioloid_mylaserfront/depth:o
to /matlab/read using tcp
```

Figura 6.51. Consola de comandos Linux-Matlab

6.1.3.3.Comunicación YARP – OpenRave – Simulink

En este punto se analizará la solución creada, para comunicar el simulador en OpenRave con la máquina de estados creada en Simulink utilizando la librería de Stateflow. A la hora de resolver esta cuestión, se presentaban dos posibilidades, que eran cada una de las analizadas en los dos apartados anteriores.

En primer lugar se podría utilizar una “s-function” [38], dentro del modelo de Simulink para utilizar el código creado en C++ o Matlab, mostrado anteriormente o bien utilizar dentro del modelo de Simulink, “matlab-functions” [39] en las que se introdujera el código de las funciones en Matlab anteriormente analizadas. Finalmente se optó por la segunda opción, ya que resulta más sencillo y optimo insertar “matlab-functions” que “s-functions” dentro de los modelos de Simulink, obteniéndose un modelo más modular y sencillo siguiendo la línea de desarrollo de todo el proyecto.

Una vez decidida la solución a implementar, se comenzó por definir cada una de las funciones necesarias para comunicar de manera correcta la máquina de estados creada en stateflow, con Openrave. Como ya se analizó en el apartado donde se describió la máquina de estados, esta trabajaba con valores binarios, en concreto trabajaba con los valores binarios de entrada (u, v, w) y los valores de salida (s1, s2). Al no coincidir estos valores con los recibidos desde los sensores ni tampoco con los que se habían de enviar a los servomotores, hubo que analizar la información manejada por YARP, para de esta manera adaptarla a los valores binarios de la máquina de estados, como se analizará en el siguiente apartado.

Las funciones utilizadas para la comunicación dentro del modelo son: en primer lugar una función que se encargara de cargar las librerías de YARP dentro del modelo de Simulink, otras tres dedicadas a cada uno de los sensores, que además se encargaran de interpretar los datos de entrada y determinar los valores binarios que serán enviados a la máquina de estados y por ultimo una función que se encargara de enviar los valores indicados a cada uno de los servomotores, según la información recibida desde las salidas de la máquina de estados. En la figura 6.52. se muestran las cabeceras de dichas funciones, acompañadas de su representación como Matlab-function dentro del modelo final, que será analizado con detalle dentro del siguiente apartado, en donde también se describirá cada una de las funciones mostradas.

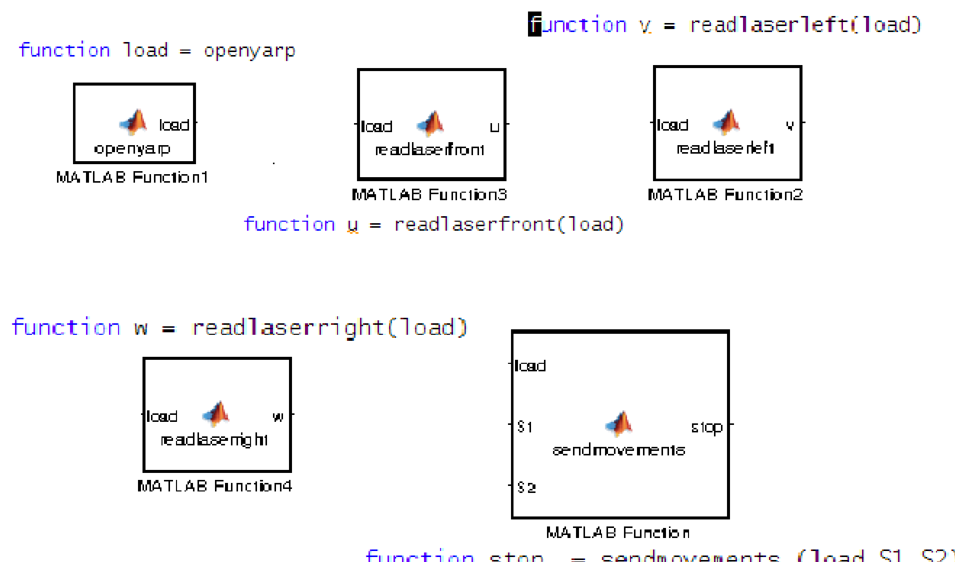


Figura 6.52. Bloques y cabeceras de las Matlab-functions utilizadas

6.1.4. Modelo final del controlador

En este apartado se analizará el modelo definitivo del controlador creado en Simulink, incluyendo las comunicaciones, la máquina de estados y demás bloques. Antes de realizar el análisis citado, se realizará una descripción detallada de cómo se han de interpretar los datos recibidos desde los sensores del simulador y cuáles serán los datos a enviar a los servomotores, en función de las secuencias extraídas del software RoboPlus.

Los datos recibidos desde los sensores simulados dentro de OpenRave, son recibidos en forma de botellas YARP, las cuales se componen de varios datos. En primer lugar [mat] y [dec] cuyo significado no es relevante para el desarrollo, ya que se trata de simples identificadores. En segundo lugar envían cinco datos entre paréntesis loas cuales representan la posición en la que se encuentra el sensor situado, además de otras propiedades como su rango, etc. Por último los datos que nos interesan que son los 88 datos recibidos entre llaves a continuación.

Como se observa en la figura 6.53., en la que se van capturando los datos enviados por los sensores a medida que se acerca el robot a un obstáculo, sus valores se van incrementando hasta el valor máximo de (0, 0, 250, 167), que será el valor umbral que tomaremos a la hora de transformar estos valores recibidos, en valores binarios para enviárselos a la máquina de estados. Ya que una vez estos valores llegan a 250 el robot se encuentra bastante cerca y si siguiese avanzando podría colisionar.

```
[mat] [dec] (4 88 8 21 1) {0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 88 162 3 11}
[mat] [dec] (4 88 8 21 1) {0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 101 32 116 99}
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
50 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167
 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250
67 0 0 250 167 0 0 250 167 0 0 250 167 0 1 0 0}
[mat] [dec] (4 88 8 21 1) {0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0
50 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167
 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250 167 0 0 250
67 0 0 250 167 0 0 250 167 0 0 250 167 0 0 0 0}
```

Figura 6.53. Valores de los sensores a medida que se acercan a un obstáculo

Una vez analizados los datos recibidos desde los sensores, hay que determinar los movimientos que debe realizar el robot en función de ellos como andar hacia delante, hacia la derecha y hacia la izquierda, para ello se extrajeron del programa RoboPlus, las secuencias a través de las que se realizaban dichos movimientos. Al extraer de dicho programa las secuencias de movimientos, nos encontramos con tres problemas, el primero de ellos era que los servos, para los que se encontraban escritas las secuencias tenían unas IDs diferentes a las que nosotros le habíamos asignado a nuestro modelo, como se puede observar en la imagen 6.54. donde se muestra la asignación de IDs que tenía el robot Bioloid dentro de RoboPlus.

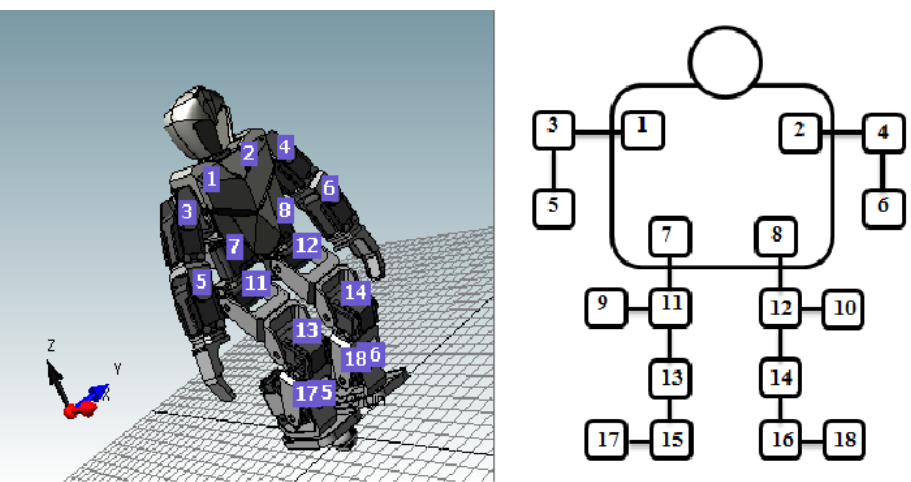


Figura 6.54. Distribución de las IDs de los motores en RoboPlus

Por lo tanto hubo que reordenar la tabla de datos extraída desde RoboPlus, dentro del archivo de Excel creado hasta que los ángulos deseados para los servomotores, coincidiesen con las IDs de nuestro modelo. En concreto los cambios que hubo que hacer dentro de dicho archivo, son los que modifican las IDs como se muestra en la figura 6.55., donde se pueden observar los valores de IDs asignados dentro de RoboPlus y los asignados dentro de nuestro modelo.

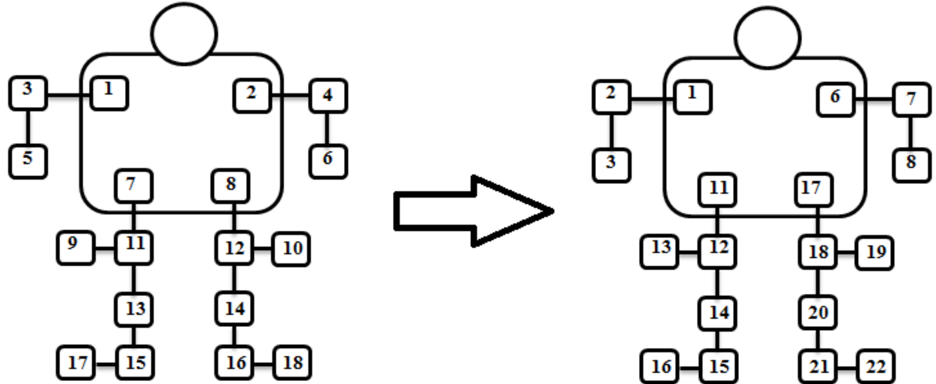


Figura 6.55. Asignación de IDs de nuestro modelo y el de RoboPlus

Una vez resuelto este problema pasamos a solucionar el segundo de los tres encontrados, este problema consistía en que en lugar de determinar el ángulo objetivo en grados como lo hacíamos nosotros, determinaban dicho ángulo en función de la cuenta del encoder de cada uno de los servomotores, por lo que hubo que transformar dicha cuenta a grados aplicando la fórmula que se muestra en la figura 6.56. dentro de Excel y teniendo en cuenta que el recorrido máximo de cada uno de estos servomotores es de 300°, siendo la cuenta del encoder dentro de este rango de 0 a1023 pulsos.

$$Angulo\ objetivo^0 = \frac{\text{numero de pulsos x } 300^0}{1024 \text{ pulsos}}$$

Formula en Excel → = (A1*300)/1024

Figura 6.56. Obtención del ángulo objetivo

El último de los problemas encontrados fue que dentro de RoboPlus, en vez de determinar la velocidad a la que tenían que moverse los servomotores desde el ángulo actual al ángulo deseado, se determinaba el tiempo en el que dicho ángulo debía variar, por lo que hubo que transformar dicho tiempo a velocidad angular introduciendo la formula mostrada en la figura 6.57. dentro de Excel, para conseguir el movimiento deseado.

$$(\theta - \theta_0)^0 \ \frac{\pi}{180^0} = \omega \ {}^{rad}/_{s} \ x \ (t - t_0)s$$

$$\omega = \frac{\pi \ x \ (\theta - \theta_0)}{180 \ x \ (t - t_0)} \ {}^{rad}/_{s}$$

Formula en Excel → =(PI()*ABS(A1-E1))/((180)*F1)

Figura 6.57. Obtención de la velocidad angular

Una vez transformados los datos siguiendo los pasos descritos anteriormente se obtuvieron las secuencias mostradas a continuación para los diferentes movimientos, en concreto el movimiento de andar de frente será la secuencia mostrada en la Tabla 6.10., el de andar hacia la derecha la secuencia mostrada en la tabla 6.11. y el de andar hacia la izquierda la secuencia mostrada en la tabla 6.12.

Secuencia para andar hacia delante						
IDs motores	Posición inicial (º)	Paso 1 (º)	Paso 2 (º)	Paso 3 (º)	Paso 4 (º)	Velocidad angular (rad / s)
1	49,51	58,89	86,13	78,52	51,27	1,31
2	211,52	220,90	248,14	240,53	213,28	1,31
3	81,74	81,74	81,74	81,74	81,74	0,00
4	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00	0,00
6	135,35	135,35	135,35	135,35	135,35	0,00
7	164,36	164,36	164,36	164,36	164,36	0,00
8	104,88	104,88	104,88	104,88	104,88	0,00
9	0,00	0,00	0,00	0,00	0,00	0,00
10	0,00	0,00	0,00	0,00	0,00	0,00
11	195,12	195,12	195,12	195,12	195,12	0,00
12	150,29	144,43	148,54	136,23	148,54	1,31
13	152,93	163,48	151,17	155,27	151,17	1,31
14	88,48	98,14	103,71	87,60	87,01	1,09
15	193,36	212,11	212,70	201,56	196,00	1,96
16	72,07	78,22	72,66	57,42	72,66	0,44
17	225,29	242,29	227,05	221,48	227,05	1,31
18	181,93	185,45	196,58	196,00	179,88	1,53
19	101,95	103,71	119,82	114,26	103,13	0,87
20	150,29	156,15	148,54	140,33	148,54	1,31
21	152,93	159,38	151,17	143,55	151,17	1,31
22	217,97	217,97	217,97	217,97	217,97	0,00

Tabla 6.10. Secuencia para andar hacia delante

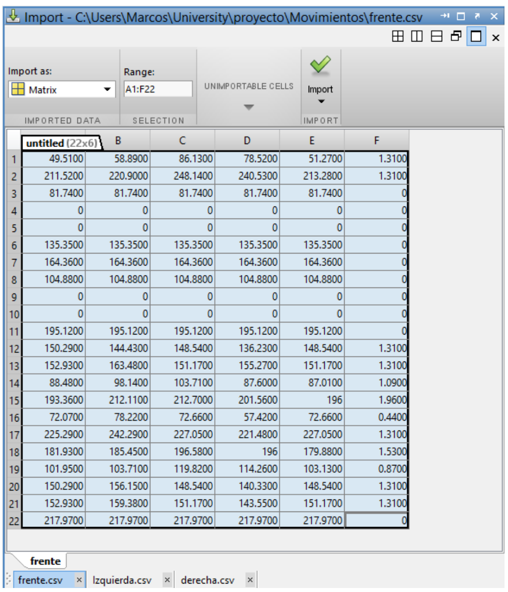
Secuencia para andar hacia la izquierda						
IDs motores	Posición inicial (º)	Paso 1 (º)	Paso 2 (º)	Paso 3 (º)	Paso 4 (º)	Velocidad angular (rad / s)
1	68,85	68,85	68,85	68,85	68,85	0,00
2	230,86	230,86	230,86	230,86	230,86	0,00
3	81,74	81,74	81,74	81,74	81,74	0,00
4	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00	0,00
6	135,35	135,35	135,35	135,35	135,35	0,00
7	164,36	164,36	164,36	164,36	164,36	0,00
8	104,88	113,67	122,75	113,67	104,88	0,00
9	0,00	0,00	0,00	0,00	0,00	0,00
10	0,00	0,00	0,00	0,00	0,00	0,00
11	195,12	186,04	176,95	186,04	195,12	0,00
12	150,59	145,02	148,54	135,35	148,54	1,53
13	152,93	164,36	151,17	154,69	151,17	1,31
14	100,20	105,18	99,61	91,99	99,90	0,22
15	199,51	207,71	200,10	194,53	199,80	0,22
16	62,99	78,81	70,31	57,71	70,31	5,45
17	228,81	241,99	229,39	220,90	229,39	0,44
18	189,26	186,33	188,96	193,95	189,55	0,22
19	110,45	105,76	110,74	113,38	110,16	0,22
20	150,59	156,74	148,54	139,45	148,54	1,53
21	152,93	160,25	151,17	142,97	151,17	1,31
22	217,97	217,97	217,97	217,97	217,97	0,00

Tabla 6.11. Secuencia para andar hacia la izquierda

Secuencia para andar hacia la derecha						
IDs motores	Posición inicial (º)	Paso 1 (º)	Paso 2 (º)	Paso 3 (º)	Paso 4 (º)	Velocidad angular (rad / s)
1	68,85	68,85	68,85	68,85	68,85	0,00
2	230,86	230,86	230,86	230,86	230,86	0,00
3	81,74	81,74	81,74	81,74	81,74	0,00
4	0,00	0,00	0,00	0,00	0,00	0,00
5	0,00	0,00	0,00	0,00	0,00	0,00
6	135,35	135,35	135,35	135,35	135,35	0,00
7	164,36	164,36	164,36	164,36	164,36	0,00
8	122,75	113,67	104,88	113,67	122,75	0,00
9	0,00	0,00	0,00	0,00	0,00	0,00
10	0,00	0,00	0,00	0,00	0,00	0,00
11	176,95	186,04	195,12	186,04	176,95	0,00
12	150,29	145,02	148,54	135,35	148,54	1,31
13	152,93	164,36	151,17	154,69	151,17	1,31
14	100,49	105,18	99,90	91,99	99,61	0,65
15	200,39	207,71	199,80	194,53	200,10	0,22
16	70,61	78,81	70,31	57,71	70,31	0,22
17	228,81	241,99	229,39	220,90	229,39	0,44
18	189,55	186,33	189,55	193,95	188,96	0,44
19	111,62	105,76	110,16	113,38	110,74	0,65
20	150,29	156,74	148,54	139,45	148,54	1,31
21	152,93	160,25	151,17	142,97	151,17	1,31
22	217,97	217,97	217,97	217,97	217,97	0,00

Tabla 6.12. Secuencia para andar hacia la derecha

Todos los datos mostrados en las tres tablas anteriores, son exportados desde Excel, en formato .csv, para que puedan ser importados dentro de Matlab como se muestra en la Tabla 6.13. Una vez importados, Matlab los transformará a formato .dat, para más tarde utilizarlos dentro de la función “sendmovements”, la cual se encargará de enviar dichos datos a través de YARP a cada uno de los servomotores del simulador OpenRave.



	A	B	C	D	E	F
1	49.5100	58.8900	86.1300	78.5200	51.2700	1.3100
2	211.5200	220.9000	248.1400	240.5300	213.2800	1.3100
3	81.7400	81.7400	81.7400	81.7400	81.7400	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	135.3500	135.3500	135.3500	135.3500	135.3500	0
7	164.3600	164.3600	164.3600	164.3600	164.3600	0
8	104.8800	104.8800	104.8800	104.8800	104.8800	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	195.1200	195.1200	195.1200	195.1200	195.1200	0
12	150.2900	144.4300	148.5400	136.2300	148.5400	1.3100
13	152.9300	163.4800	151.1700	155.2700	151.1700	1.3100
14	88.4800	98.1400	103.7100	87.6000	87.0100	1.0900
15	193.3600	212.1100	212.7000	201.5600	196	1.9600
16	72.0700	78.2200	72.6600	57.4200	72.6600	0.4400
17	225.2900	242.2900	227.0500	221.4800	227.0500	1.3100
18	181.9300	185.4500	196.5800	196	179.8800	1.5300
19	101.9500	103.7100	119.8200	114.2600	103.1300	0.8700
20	150.2900	156.1500	148.5400	140.3300	148.5400	1.3100
21	152.9300	159.3800	151.1700	143.5500	151.1700	1.3100
22	217.9700	217.9700	217.9700	217.9700	217.9700	0

Tabla 6.13. Datos importados a Matlab desde Excel en formato .csv

Una vez determinadas las secuencias de movimientos y la interpretación de los datos recibidos desde los sensores del simulador, ya podíamos pasar a diseñar la lógica del control final, para conseguir que todos los elementos descritos anteriormente funcionasen como un conjunto, de una manera fiable y robusta, para lo cual se creó el modelo mostrado en la figura 6.58., del que se irán analizando en detalle cada uno de los bloques a lo largo de este apartado.

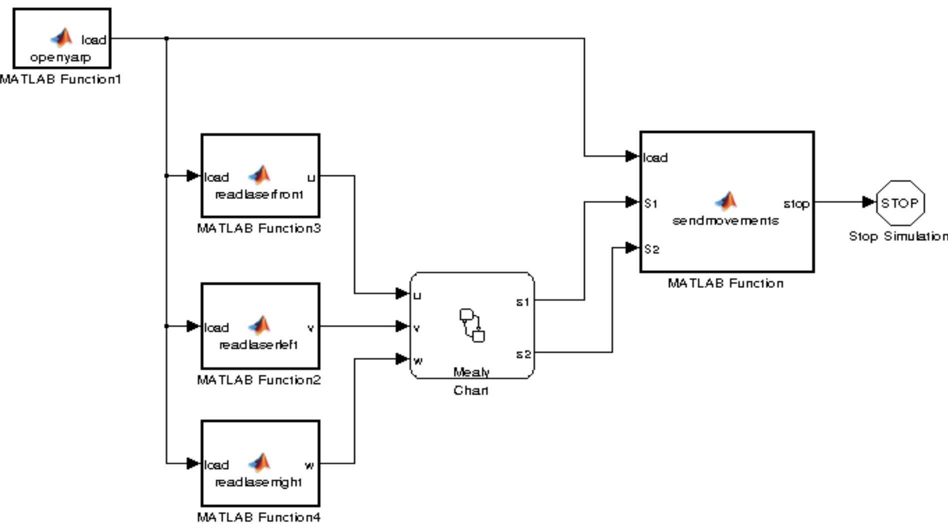


Figura 6.58. Modelo final de Simulink

Como se observa en la figura 6.58. el modelo consta de un total de 7 bloques, entre los que se incluyen 5 Matlab-functions, una máquina de estados de Mealy, creada en Stateflow y por ultimo un bloque de parada que se activara cuando sea necesario o en caso de que se produzca algún fallo dentro de la simulación del modelo. Dentro de este apartado, se irán describiendo cada uno de los bloques de izquierda a derecha, comenzando por el bloque de “openyarp” y terminando en el bloque de stop.

El bloque “openyarp”, será el encargado como se puede observar en la figura 6.59. de cargar las librerías de YARP, y crear las conexiones necesarias para que podamos enviar datos al “remote_controlboard”, en caso de que todo el proceso sea correcto, incluyendo la creación del objeto dd, que será el encargado de manejar todas las interacciones con YARP, se cargará un valor “1” en su variable de salida “load”, lo que permitirá que el resto de bloques del modelo puedan comenzar su ejecución.

load
openyarp
MATLAB Function1

Editor - Block: final/MATLAB Function1

```
function load = openyarp

LoadYarp;
options = yarp.Property;
options.put('device','remote_controlboard');
options.put('remote','/ravebot');
options.put('local','/matlab');
dd = yarp.PolyDriver(options);

if isequal(dd.isValid,1)
load=1;
end
```

Figura 6.59. Bloque openyarp dentro del modelo final

En segundo lugar se analizarán los bloques de Matlab-function, utilizados para leer los datos de los sensores y acondicionar la señal recibida, que será transformada a una señal binaria apropiada para transferir los datos a la máquina de estados. Se analizará con detalle solo uno de los tres que será el del sensor frontal, ya que los demás poseen una programación análoga pero cambiando determinados parámetros característicos de cada uno de ellos como por ejemplo la salida (u, v, w).

El bloque "readlaserfront", dentro de esta bloque como se observa en la figura 6.60.. En primer lugar se importan los puertos y botellas de las librerías YARP, para que se pueda trabajar con ellos, en segundo lugar se crea un puerto de lectura (/matlab/read), que posteriormente se conecta con el puerto YARP del sensor frontal dentro de OpenRave. En este punto se realiza una comprobación de que realmente exista la conexión entre ambos puerto y esté funcionando, en caso de que todo se encuentre correctamente conectado, se mantiene la variable de entrada (load) a "1", dando acceso al bucle while y en caso contrario se pone dicha variable a "0", finalizando la simulación.

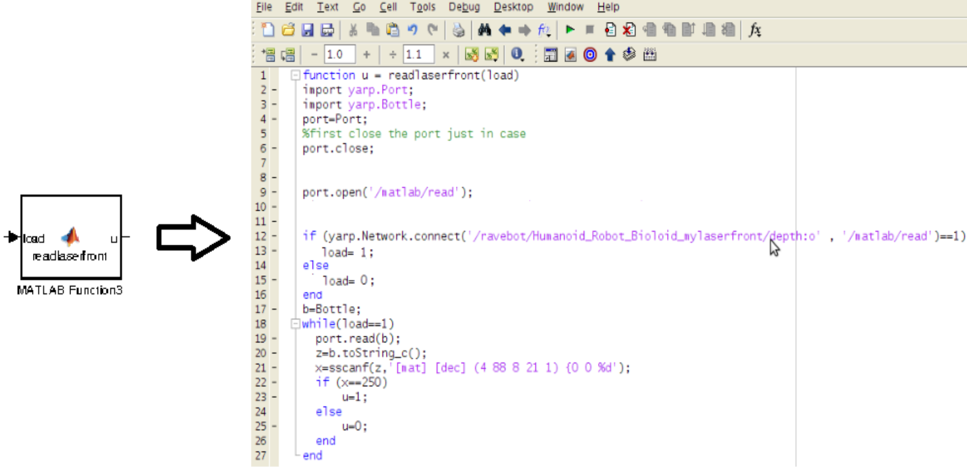


Figura 6.60. Bloque readlaserfront dentro del modelo final

Dentro del bucle while que se encontrará activo mientras la variable "load" se encuentre a "1", como se muestra en la imagen 6.61., se crea un puerto para la lectura de la botella "b", en la que se encuentran integrados todos los datos enviados por el sensor en OpenRave. En segundo lugar se almacena dentro de la variable "z", todos los datos que contiene la botella "b", pero transformados a tipo string, para después analizar estos datos y en función de si su valor es igual o no al valor umbral, que determina si hay riesgo de colisión, se le asigna a la variable de salida "u" un valor 0 o 1 que se enviará a la máquina de estados, significando "0" que no hay peligro de colisión y "1" que si existe riesgo.

Las salida de cada uno de los sensores (u, v, w), se conecta a la máquina de estados analizada anteriormente, en la que se encuentra implementada la lógica secuencial, que se encargará de enviar las salidas correspondientes (S1, S2) al siguiente bloque, en el que se interpretarán dichas salidas enviando la secuencia indicada al simulador OpenRave a través de YARP. Todas las conexiones realizadas a y desde la máquina de estados se muestran en la figura xxx.

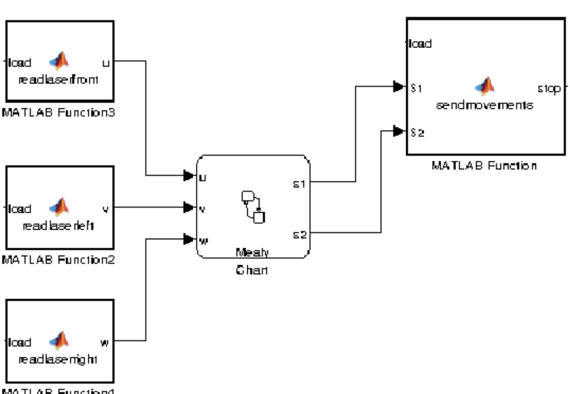


Figura 6.61. Conexiones de la máquina de estados con el resto del modelo

A continuación se muestra la figura 6.62., donde se puede observar la máquina de estados funcionando, en concreto se muestran las transiciones y estados activos, en algunos de los procesos más relevantes dentro de su funcionamiento, como son el proceso de andar de frente, o los procesos en los que se ha detectado un obstáculo y por tanto se produce una transición. Los estados y transiciones activos en cada momento son los que se observan resaltados en color azul.

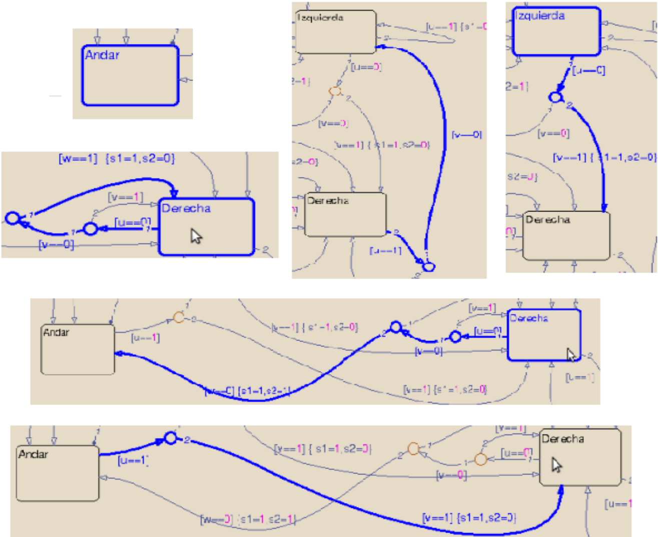


Figura 6.62. Procesos dentro de la máquina de estados funcionando

Las salidas de la máquina de estados, se conectan al bloque “sendmovements”, que será el encargado de interpretar los datos recibidos y en función de ellos enviar la secuencia correcta al simulador. Todas las secuencias serán cargadas desde los archivos .dat importados a Matlab, analizados anteriormente dentro de este apartado.

En primer lugar dentro de esta función se crean los objetos (pos, vel), que serán los encargados de enviar los ángulos objetivo y la velocidad a la que se han de mover los servomotores, para alcanzar dichos ángulos en el tiempo adecuado. Uno vez creados como se observa en la figura 6.63. se inicia el bucle while, siempre que la variable “load”, se encuentre a “1”. Dentro de este bucle en primer lugar se analizan las entradas (S1, S2) recibidas desde la máquina de estados, asignando un valor a la variable “x” en función de ellas, que se encargará de ejecutar el caso apropiado dentro de la siguiente estructura switch case.

Dentro de esta estructura switch case, será donde se envíen los valores indicadas para cada una de las secuencias, en función del valor “x” anteriormente analizado. En concreto se recorrerán cada uno de los ficheros .dat haciendo uso del método la burbuja, implementado con dos bucles for. Gracias a este método se puede recorrer el archivo insertando los datos apropiados dentro de (PositionMode, VelocityMove) de manera que se insertarán los datos de la columna 1 a la 5 para los 22 servos y los datos de la columna 6 en la que se indican las velocidades también para los 22 servos. Los datos a los servomotores dentro de OpenRave serán transmitidos a través de YARP.

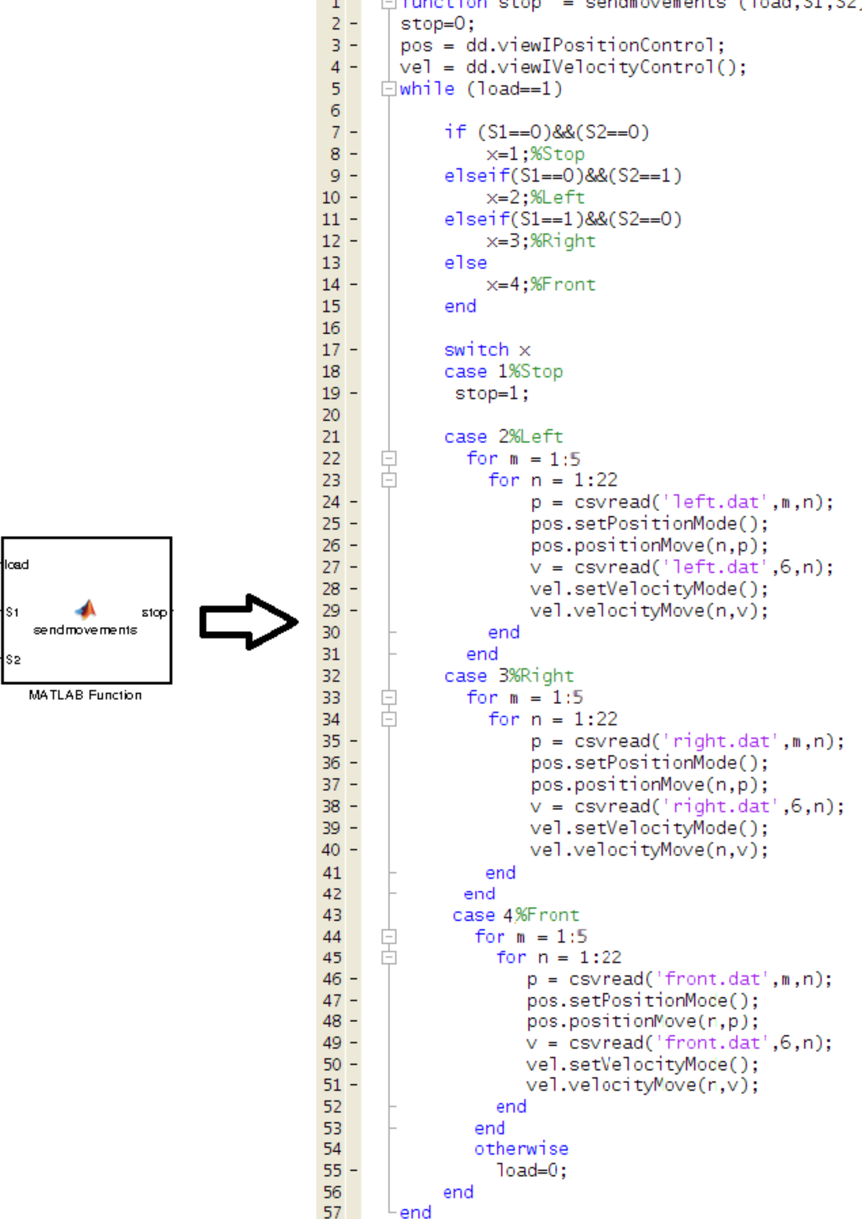


Figura 6.63. Bloque sendmovements dentro del modelo final

Por ultimo cabe destacar que en el caso de que la máquina de estados envié un valor para S1=0 y S2=0, se modificará el valor de la variable de salida “stop”, asignándola un valor uno que activará el bloque mostrado en la figura 6.64., y que parara la simulación siguiendo las especificaciones del diseño del control.

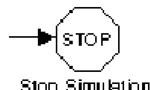


Figura 6.64. Bloque de parada de la simulación dentro del modelo final

6.1.5. Proceso de generación de código para la placa controladora

En este apartado se indicarán los procesos que existen dentro de Simulink, para generar a partir del modelo creado, código en C/C++ utilizando la herramienta Simulink Coder, o bien generar un ejecutable de tipo Arduino, en el caso de que se utilice finalmente una placa Arduino Uno, o bien la CM-900, ya que soporta este tipo de código. Por tanto este apartado se dividirá en dos sub-apartados en los que se explicarán ambas procesos.

6.1.5.1. Proceso de generación de código C/C++ con Simulink Coder

El proceso para generar código en C/C++, resulta sencillo siguiendo los pasos descritos dentro del tutorial [40], en pocos pasos se obtendrá código C/C++ de nuestro modelo creado en Simulink gracias a la herramienta que el proveedor Mathworks nos proporciona. Esta herramienta es la ToolBox Simulink Coder, en este apartado se ha generado código C del modelo de máquina de estados (stateflowtrans.mdl) mostrado en la figura 6.65., a modo de ejemplo.

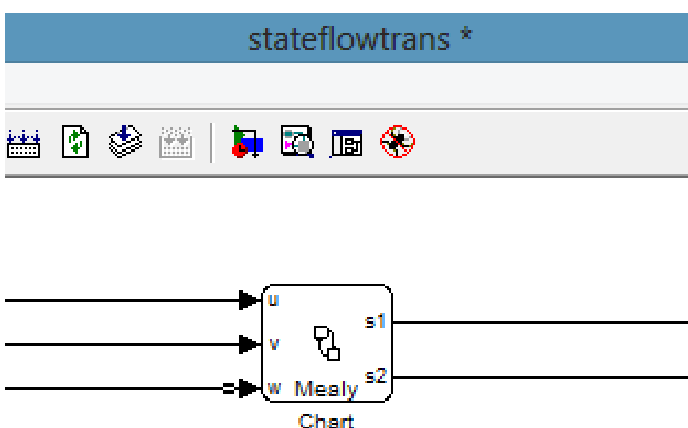


Figura 6.65. Máquina de estados dentro del archivo stateflowtrans.mdl

En primer lugar dentro de las opciones de la herramienta Simulink Coder, a las que se accede siguiendo dentro de la ventana del modelo la ruta (tools/Code Generation/Options…), se ha de modificar el tipo de Solver elegido para que sea uno de tipo fixed-step, ya que son los únicos soportados por dicha herramienta, por tanto se ha de sustituir el Solver elegido para la máquina de estados por otro con la misma funcionalidad pero en lugar de Variable-step que sea Fixed-step, en este caso se ha utilizado el ode4 como se observa en la figura 6.66., al ser el que utiliza los mismos métodos (Runge-Kutta) que el ode45, configurado en primer lugar, pero con un periodo invariable.

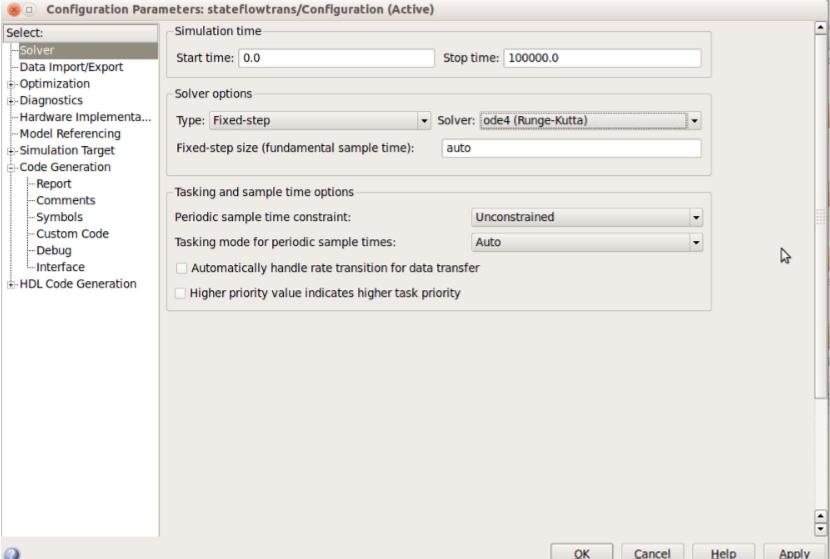


Figura 6.66. Configurar Solver para generación de código

Una vez seleccionado el Solver adecuado, simplemente habría que seleccionar como se muestra en la figura 6.67. el tipo de que código que queremos obtener C o C++ y comenzar la generación. Para comenzar dicha generación habrá que pulsar el botón de Build Model, dentro de la ruta de la ventana de Simulink (tools/Code Generation/Build Model).

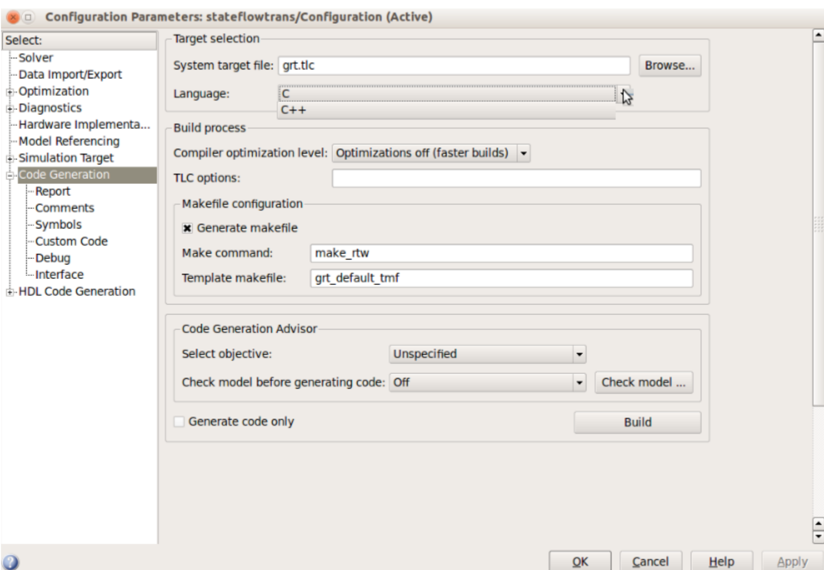


Figura 6.67. Elección de código C/C++ para su generación

En este momento se inicia la generación de código, realizándose varios procesos de manera automática como se puede observar dentro de la consola de Matlab, mostrada en la figura 6.68.

```
### Starting build procedure for model: stateflowtrans
### Generating code into build folder: /home/markitos/stateflowtrans_grt_rtw
### Invoking Target Language Compiler on stateflowtrans.rtw
### Using System Target File: /usr/local/MATLAB/R2012a/rtw/c/grt/grt.tlc
### Loading TLC function libraries
.....
### Initial pass through model to cache user defined code
..
### Caching model source code
...................................
### Writing source file stateflowtrans.c
### Writing header file stateflowtrans_private.h
.
### Writing header file stateflowtrans.h
### Writing header file stateflowtrans_types.h
### Writing header file rtwtypes.h
.
### Writing header file rtmodel.h
### Writing source file stateflowtrans_data.c
### Writing header file rt_nonfinite.h
.
### Writing source file rt_nonfinite.c
### Writing header file rt_defines.h
.
### Writing header file rtGetInf.h
### Writing source file rtGetInf.c
### Writing header file rtGetNaN.h
.
### Writing source file rtGetNaN.c
### TLC code generation complete.
### Creating project marker file: rtw_proj.tmw
.
### Processing Template Makefile: /usr/local/MATLAB/R2012a/rtw/c/grt/grt_unix.tmf
### Creating stateflowtrans.mk from /usr/local/MATLAB/R2012a/rtw/c/grt/grt_unix.tmf
### Building stateflowtrans: /usr/local/MATLAB/R2012a/bin/glnx86/gmake -f stateflowtrans.mk  GENERATE_REPORT=0
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -c -ansi -pedantic -fwrapv   -DUSE_RTMODEL -O0 -ffloat-store -fPIC -m32 -DMODEL=stateflowtrans -DRT -DNUMST=3 -DTID01EQ=0 -DI
gcc -m32  -o ../stateflowtrans rtGetInf.o rtGetNaN.o rt_logging.o rt_nonfinite.o stateflowtrans_data.o stateflowtrans.o grt_main
### Created executable: ../stateflowtrans
### Successful completion of build procedure for model: stateflowtrans
```

Figura 6.68. . Proceso de generación desde la consola de comandos de Matlab

Una vez finalizado el proceso de generación de código, ya disponemos de todo este código C, con sus respectivos ficheros de cabeceras .h, sus ficheros .c y algunos componentes más que generan como los ficheros .o, etc., como se muestra en la figura 6.69., donde se pueden observar todos los archivos generados dentro de la carpeta “stateflowtrans_gft_rtw”.

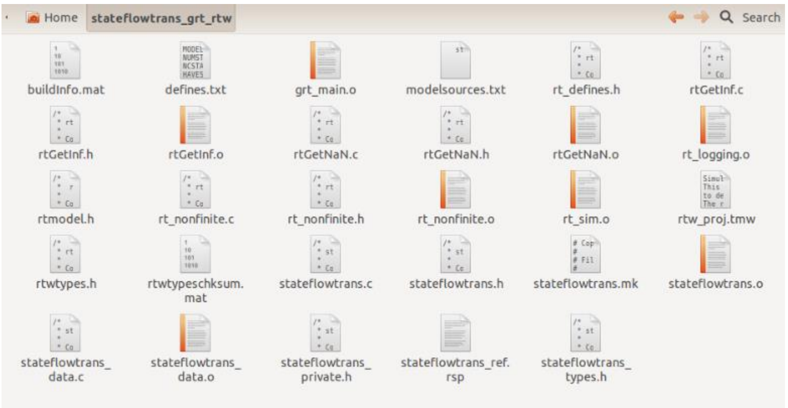


Figura 6.69. Carpeta con los archivos generados

Los archivos que nos servirán para reproducir el comportamiento de la máquina de estados con un programa en C, serán los archivos de staflowtrans con las extensiones .h y .c. El contenido de dichos archivos se puede observar en la figura 6.70., donde se comprueba que el código obtenido es código C.

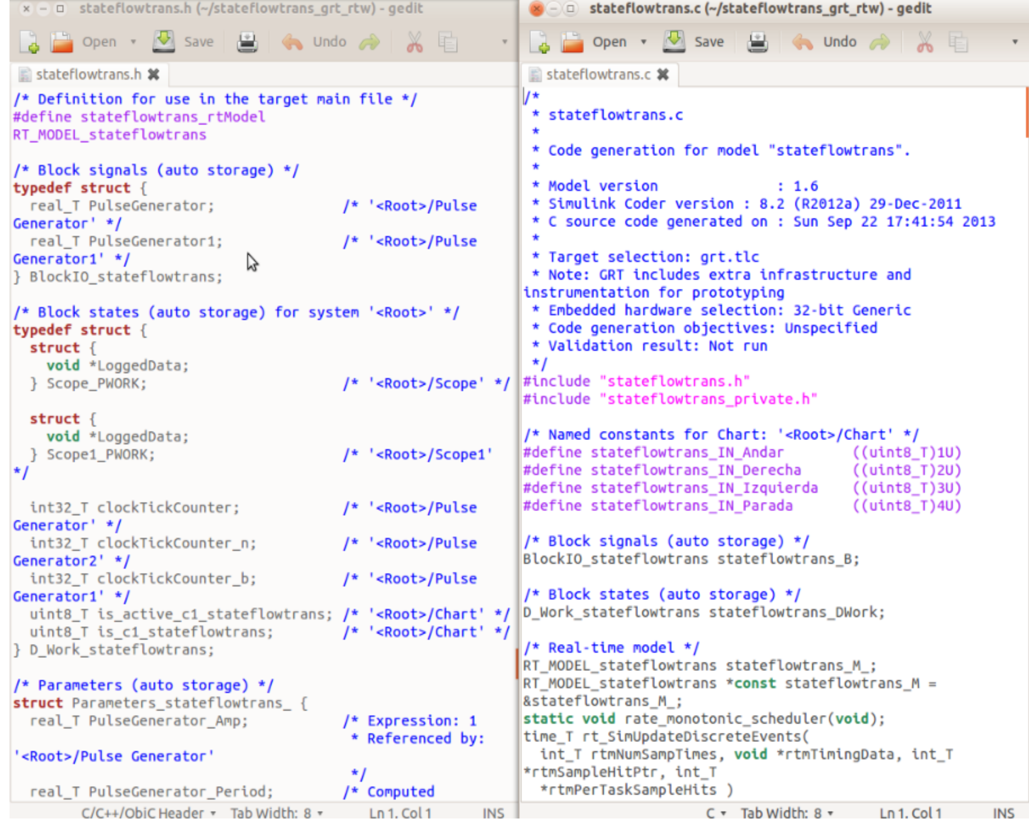


Figura 6.70. Archivos staflowtrans.h y .c generados

6.1.5.2. Proceso de generación de un ejecutable para Arduino UNO

Al instalar el pack de soporte para Arduino, dentro de Simulink se incluye una nueva librería, mostrada en la figura 6.71., en la que disponemos de bloques para conectar los puertos de entrada/salida digital, PWM, entrada/salida analógica, comunicación serie, etc.

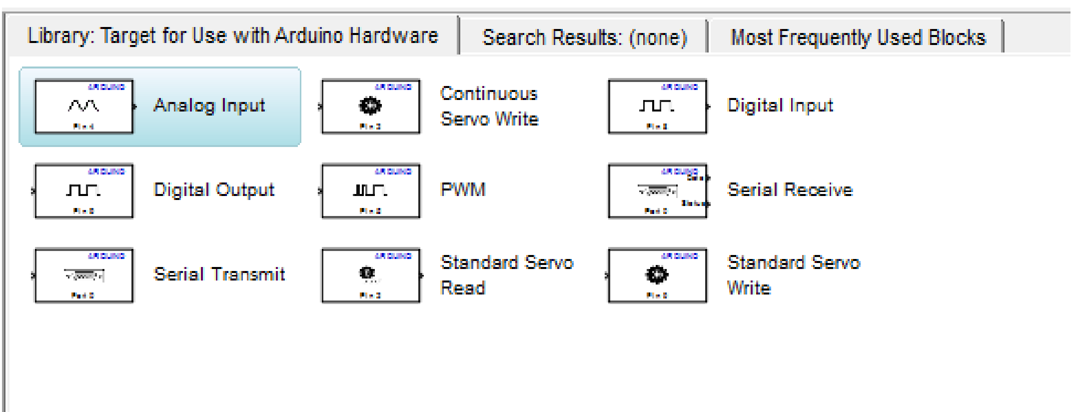


Figura 6.71. Librería para Arduino UNO de Simulink

Gracias a esta librería, se podrá conectar nuestra placa de Arduino UNO con Simulink, pudiendo trabajar directamente con los datos recibidos a través de sus pines de entrada y enviar datos a sus pines de salida, de este modo podremos realizar todas las pruebas que deseemos, siguiendo el modelo de desarrollo HIL (Hardware In the Loop) como se puede observar en la figura 6.72. La conexión entre la placa de Arduino y el equipo donde se esté trabajando con Matlab se realizará a través del puerto serie de la placa Arduino UNO y uno de los USB del ordenador.

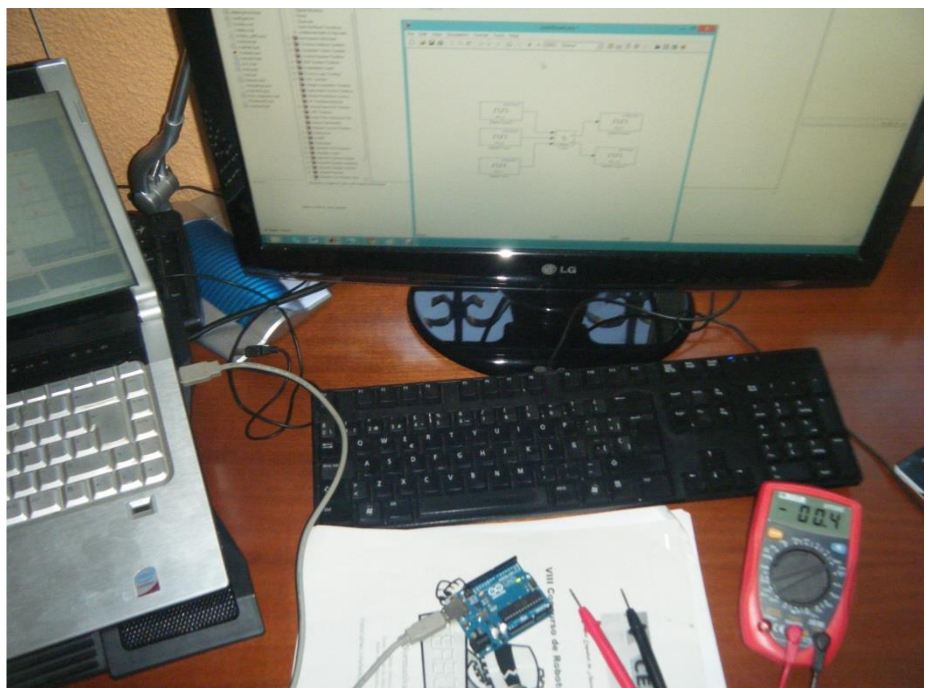


Figura 6.72. Desarrollo HIL con Arduino y Simulink

Para conectar el modelo con la placa real, hubo que configurar el puerto serie a través del que se conectaba la placa Arduino UNO como se muestra en la figura 6.73., en nuestro caso el puerto en concreto en el que se encontraba conectada era el puerto COM 8. Una vez conectada y configurada, tras marcarle los puertos de entrada y salida se comprobó que realmente se estaban comunicando y funcionando correctamente.

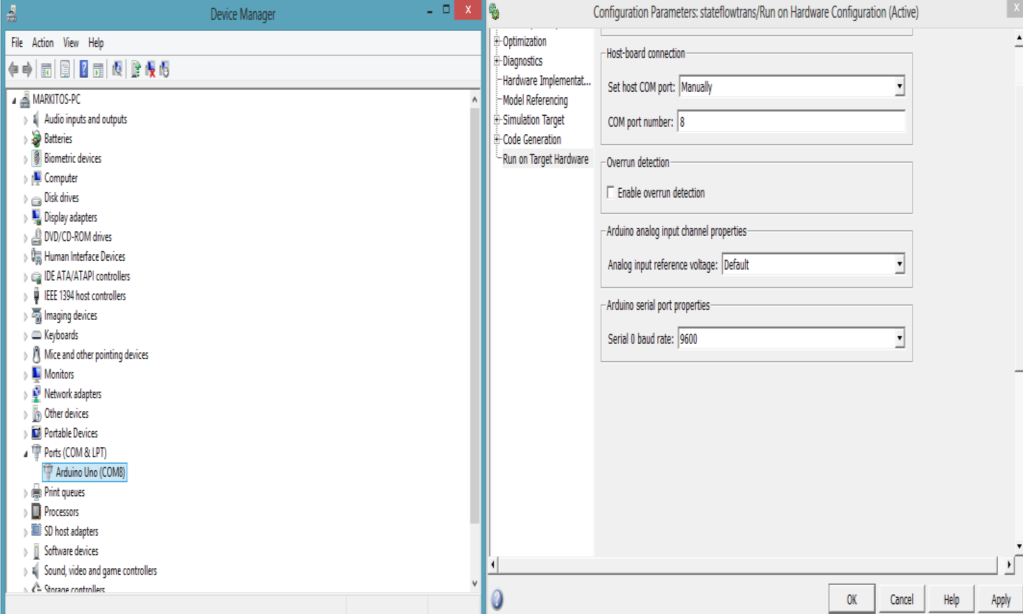


Figura 6.73. Configuración del puerto serie en Matlab

Una vez probado el funcionamiento del modelo, con el proceso descrito anteriormente se podrá generar un archivo ejecutable con código tipo Arduino, para descargarlo en la placa y conectarlo a los dispositivos finales, en nuestro caso los sensores y actuadores de nuestro robot. A modo de ejemplo se han conectado a nuestra máquina de estados (stateflowtrans), tres entradas digitales y dos salidas digitales configurándolas en diferentes puertos de la placa, como se muestra en el modelo mostrado en la figura 6.74.

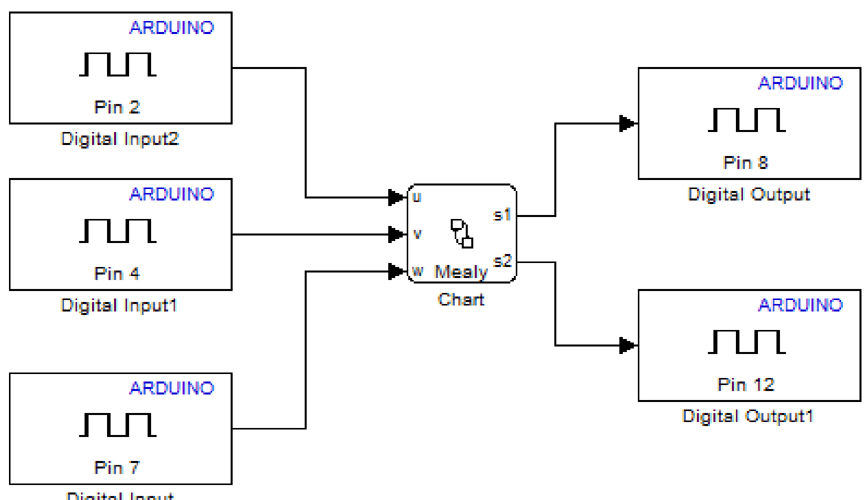


Figura 6.74. Conexión de la máquina de estados con Arduino UNO

Para descargar el ejecutable a nuestra placa, se han de seguir los pasos descritos a continuación, los cuales han sido extraídos del tutorial [41], tras lo cual se podrá desconectar de nuestro ordenador la placa Arduino UNO y funcionará de una manera totalmente autónoma.

- En primer lugar hemos de ir a tools > Run on Target Hardware > Prepare to Run. Esta acción preparar nuestra aplicación para funcionar. Además de cambiar los parámetros de configuración de modelo, para que se adecuen al hardware real.
- E la ventana que se abre, hemos de establecer el hardware de destino en nuestro caso Arduino UNO, como se observa en la figura 6.75., y además el puerto en el que se encuentra, en nuestro caso el puerto COM 8.

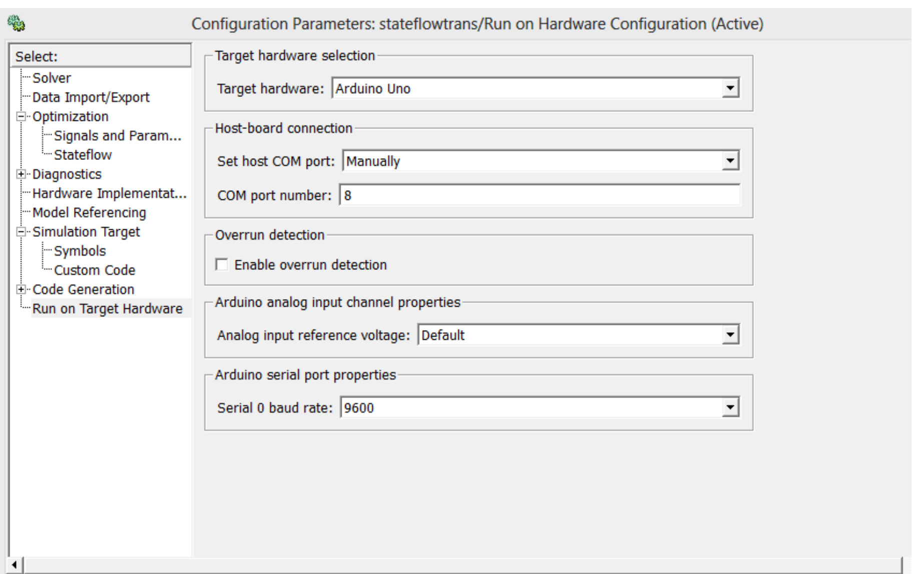


Figura 6.75. Configuración de los parámetros para crear el ejecutable

- Por ultimo regresamos al modelo y dentro de Tools > Run on Target Hardware pinchamos en Run. Esta acción descargará automáticamente la aplicación a la placa y se ejecuta el modelo en el hardware Arduino.

Una vez finalizado el proceso, se desconectará la placa del ordenador y la siguiente vez que se le conecte la alimentación ya ejecutará por defecto la aplicación descargada, obteniéndose el resultado deseado, al ser una aplicación totalmente autónoma capaz de trabajar en le controladora del robot real.

7. Conclusiones y trabajos futuros

El desarrollo de este proyecto ha supuesto para mí un gran reto, ya que cuando comencé a trabajar en él hace ya varios meses, mis conocimientos sobre robótica humanoide eran bastante limitados y he conseguido gracias a él, adquirir un elevado nivel dentro de este ámbito, el cual resulta apasionante y lleno de alternativas, fomentando el aprendizaje de una manera divertida pero a la vez profesional.

Además de los nuevos conocimientos adquiridos sobre robótica humanoide, durante el desarrollo de este proyecto he potenciado y desarrollado muchos otros, como son mis conocimientos sobre el sistema operativo Linux y los múltiples software utilizados durante su desarrollo. Lo cual me hace sentir orgulloso ya que sin duda representará un gran aporte a mi futuro profesional y personal.

Los objetivos marcados para este proyecto eran bastante ambiciosos, y se han conseguido superar, demostrando el buen hacer y el compromiso, que siempre ha de ir acompañado por duro trabajo, como dijo Thomas A. Edison *"Genius is one percent inspiration and ninety-nine percent perspiration",* lo que denota que sin un duro trabajo por detrás nada es posible.

Durante el desarrollo de este proyecto, también ha habido muchos momentos en los que las cosas no salían y resultaba difícil mantener la moral, pero incluso en esos momentos viéndolo desde otra perspectiva una vez concluido, se aprendieron conceptos nuevos o métodos para determinar dónde estaba el fallo y finalmente terminar resolviéndolo. Esto me lleva a otra gran frase del genio Thomas A Edison, la cual me inspiro en los momentos bajos *"I have not failed. I´ve just found 10000 ways that won´t work".*

Tras esta visión de lo que ha supuesto el proyecto para mi persona y de cómo finalmente se han conseguido superar los objetivos marcados, pasaré a realizar un pequeño repaso al desarrollo mostrado a lo largo de esta memoria y finalizaré con los trabajos que se podrán realizar en el futuro.

7.1. Conclusiones del desarrollo

En primer lugar, se pretendió desde el comienzo de este proyecto dotarle de modularidad y que fuese fácil de ampliar y mejorar. Lo cual conseguirá mantenerlo funcionando y renovándose de una manera continua durante mucho tiempo.

El proyecto comenzó realizando un estudio de la ciencia de la Robótica Humanoide, para después centrarse en los principales desarrollos que se pueden encontrar actualmente dentro de este campo, lo cual aportó una visión general de en qué consiste esta área, en la que se aplican muchas de las tecnologías más punteras que se pueden encontrar actualmente.

Tras este breve estudio, se pasó a analizar los recursos disponibles dentro de la asociación de Robótica de la Universidad Carlos III de Madrid, centrándose en los materiales concretos para el desarrollo de este proyecto, como son los kits comerciales Bioloid Premium y Robonova. Para posteriormente analizar de manera detallada en qué consistía la competición CEABOT, ya que el desarrollo de este proyecto iría muy orientado a las pruebas que se realizan durante su transcurso.

Una vez analizados los puntos mencionados en el apartado anterior, se pasó a analizar las opciones que ofrecía el mercado de este tipo de entornos de desarrollo y simulación para robots. Lo que nos aportó una visión de cuáles eran los puntos fuertes y débiles de cada uno de ellos, inspirándonos para que nuestro diseño fuera lo más flexible y robusto posible.

En este punto se estudiaron los métodos punteros actualmente dentro del desarrollo de software, lo que nos llevó a seguir los cánones marcados por el tipo de desarrollo basado en componentes, ya que sus grandes beneficios una vez implementado el producto se ajustaban mucho a los objetivos marcados.

Partiendo de una idea clara de lo que se pretendía, tras los estudios mencionados en los párrafos anteriores, se pasó a elegir el software a utilizar, siempre teniendo en cuenta las principales directrices que debía cumplir el desarrollo que se iba a realizar. Entre las que destacaban la necesidad de implementar un entorno fiable, flexible, robusto, fácil de manejar y con una gran capacidad de ser modificado y ampliado.

Fue entonces cuando se comenzó con el desarrollo que se dividiría en tres componentes principales, el primero de ellos sería el entorno de simulación en el que como se ha mostrado a lo largo de la memoria, cumple perfectamente con su misión ya que en él se pueden incorporar todo tipo de robots de diferentes características, todo tipo de escenarios y además sus simulaciones son muy realistas.

El segundo de los tres componentes que compondrán el sistema completo, es el controlador. En este punto se optó por Simulink ya que es un software muy utilizado en la industria, muy robusto y capaz de generar modelos que funcionen en tiempo real. Además gracias a él se puede trabajar con diferentes lenguajes de programación y se pueden reutilizar los modelos creados, ampliándolos o modificándolos según las exigencias de la aplicación en concreto.

El último de los componentes, deriva de la necesidad de comunicar los dos componentes anteriores. Para lo que se ha utilizado YARP, cumpliendo con las expectativas y generando una comunicación transparente a través del protocolo TCP/IP fácilmente configurable y ampliable, siguiendo la línea general marcada para el desarrollo.

Tras el ensamblaje de los componentes analizados anteriormente, se ha conseguido un entorno de desarrollo de algoritmos de alto nivel para robots mini-humanoides, el cual podrá ser modificado y ampliado, siguiendo la descripción de los trabajos futuros a continuación.

7.2. Trabajos futuros

Los trabajos futuros han de centrarse en la competición CEABOT, ya que fue el punto de partida marcado, por lo que dichos trabajos han de ir dirigidos a mejorar la funcionalidad del robot durante su celebración. En concreto dichos trabajos serán divididos en dos puntos que se describirán en los siguientes párrafos.

El primero de los puntos será el simulador, en el que se podrán añadir nuevos escenarios, como el de la prueba de la escalera o el de la lucha de sumo. Además de los escenarios, dentro del simulador se podrán incluir nuevos sensores al modelo de Bioloid ya existente, como cámaras, IMUs, etc., para aumentar su capacidad de percepción y por tanto su comportamiento autónomo. Por último se podrá crear un nuevo modelo para el robot Robonova, el cual será integrado en el simulador de manera análoga a como se ha realizado en este proyecto para el Bioloid.

El segundo de los puntos en el que podrán centrarse los trabajos futuros, será el de crear los algoritmos dentro de Simulink, modificando la máquina de estados creada en Stateflow para la demás pruebas de la competición CEABOT, como son la prueba de escaleras o la prueba de sumo. Para ello se deberán añadir nuevos estados, entradas y salidas dependiendo de los nuevos sensores integrados en el simulador, y del comportamiento deseado, para lo cual además se deberán importar nuevas librerías de movimientos.

8. Referencias

[1] Las tres leyes de la robótica. Disponible online (Sep. 2013): http://es.wikipedia.org/wiki/Tres_leyes_de_la_rob%C3%B3tica

[2] Robot ASIMO. Disponible online (Sep. 2013): http://asimo.honda.com/

[3] Asociación de Robótica Universidad Carlos III de Madrid. Disponible online (Sep. 2013): http://asrob.uc3m.es/index.php/Mini-Humanoide

[4] RoboticsLab Universidad Carlos III de Madrid. Disponible online (Sep. 2013): http://roboticslab.uc3m.es/roboticslab/

[5] Robot Bioloid. Disponible online (Sep. 2013): http://www.robotis.com/xe/bioloid_en

[6] Controlador CM-900. Disponible online (Sep. 2013): http://support.robotis.com/en/product/auxdevice/controller/cm-900_manual.htm

[7] Robot Rovonova. Disponible online (Sep. 2013): http://www.robonova.de/store/home.php

[8] Competición CEABOT. Disponible online (Sep. 2013): http://www.ceautomatica.es/sites/default/files/upload/10/CEABOT/index.htm

[9] Webots. Disponible online (Sep. 2013): http://www.cyberbotics.com/overview

[10] Microsoft Robotics Developer Studio 4. Disponible online (Sep. 2013): http://www.microsoft.com/robotics/

[11] SimRobot. Disponible online (Sep. 2013): http://www.informatik.uni-bremen.de/simrobot/index_e.htm

[12] OpenHRP3. Disponible online (Sep. 2013): http://www.openrtp.jp/openhrp3/en/

[13] Marilou Robotics Studio. Disponible online (Sep. 2013): http://www.robotic-lab.com/blog/2007/06/14/marilou-robotics-studio-simulador-robotico/

[14] Chambers, C.: «Towards reusable, extensible components». ACM Computing Surveys, 1996, p. 192.

[15] Brown, A.: Building Systems from Pieces with Component-Based Software Engineering. Volume Constructing Superior Software, Sams, 1999. Capítulo 6.

[16] Carney, D. y Long, F.: «What do you mean by COTS? Finally, a usefull answer». IEEE Software, 2000, pp. 83–86.

[17] Desarrollo de Software Basado en Componentes. Disponible online (Sep. 2013): http://webdelprofesor.ula.ve/ingenieria/jonas/Productos/Publicaciones/Congresos/CAC03%20Desarrollo%20de%20componentes.pdf

[18] Metodología de desarrollo de aplicaciones basadas en componentes para automatización industrial. Disponible online (Sep. 2013): http://www.ceautomatica.es/old/actividades/jornadas/XXIV/documentos/tire/146.pdf

[19] Modelado basado en componentes de sistemas distribuidos de control industrial. Disponible online (Sep. 2013): http://www.disa.bi.ehu.es/mmarcos/publications/downloads/JJAA05.pdf

[20] Matlab. Disponible online (Sep. 2013): http://www.mathworks.es/products/matlab/

[21] Simulink. Disponible online (Sep. 2013): http://www.mathworks.es/products/simulink/

[22] Stateflow. Disponible online (Sep. 2013): http://www.mathworks.es/products/stateflow/

[23] Simulink Coder. Disponible online (Sep. 2013): http://www.mathworks.es/products/simulink-coder/

[24] Embedded Coder. Disponible online (Sep. 2013): http://www.mathworks.es/products/embedded-coder/

[25] YARP. Disponible online (Sep. 2013): http://eris.liralab.it/yarp/

[26] OpenRAVE. Disponible online (Sep. 2013): http://openrave.org/

[27] Blender. Disponible online (Sep. 2013): http://es.wikibooks.org/wiki/Blender_3D:_novato_a_profesional

[28] Manual de Stateflow para crear Máquinas de Estado de Mealy. Disponible online (Sep. 2013): http://www.mathworks.es/es/help/stateflow/ug/creating-mealy-and-moore-charts.html

[29] Solver Matlab-Simulink. Disponible online (Sep. 2013): http://www.mathworks.es/es/help/simulink/ug/choosing-a-solver.html

[30] Modelo Robot Bioloid. Disponible online (Sep. 2013): https://code.google.com/p/bioloid-open/

[31] Manual de Blender en castellano. Disponible online (Sep. 2013): http://www.futureworkss.com/tecnologicos/informatica/tutoriales/Manual_de_Blender.pdf

[32] Formatos XML para OpenRave. Disponible online (Sep. 2013): http://openrave.programmingvision.com/wiki/index.php/Format:XML

[33] ODE (Open Dynamics Engine) . Disponible online (Sep. 2013): http://en.wikipedia.org/wiki/Open_Dynamics_Engine

[34] Sensores en OpenRave. Disponible online (Sep. 2013): http://openrave.org/docs/latest_stable/openravepy/examples.showsensors/

[35] ASIBOT. Disponible online (Sep. 2013): http://roboticslab.uc3m.es/roboticslab/robot.php?id_robot=3

[36] Install ASIBOT simulation and basic control. Disponible online (Sep. 2013): http://roboticslab.sourceforge.net/asibot/install_on_ubuntu.html

[37] Modificar CartesianServer para que se adapte a las necesidades creadas. Disponible online (Sep. 2013): http://wdaiki.icub.org/iCub_documentation/icub_anyrobot_cartesian_interface.html

[38] Simulink s-function. Disponible online (Sep. 2013): http://www.mathworks.es/es/help/simulink/sfg/what-is-an-s-function.html

[39] Simulink Matlab-function. Disponible online (Sep. 2013): http://www.mathworks.es/es/help/simulink/ug/creating-an-example-model-that-uses-a-matlab-function-block.html

[40] Tutorial Simulink Coder. Disponible online (Sep. 2013): http://www.mathworks.es/products/simulink-coder/webinars.html

[41] Tutorial para descargar el código desde Simulink a Arduino. Disponible online (Sep. 2013): http://www.mathworks.es/es/help/simulink/ug/create-and-run-an-application-on-arduino-hardware.html

ANEXOS

Comité Español de Automática

VIII Concurso de Robots Humanoides

Normativa

Guillem Alenyà
Juan C. García Sánchez
Aleix Rull Sanahuja

concurso.ceabot@gmail.com

Terrassa, Septiembre 2013

NORMATIVA GENERAL

Objetivo.

El objetivo del concurso es mostrar las habilidades que cada robot humanoide posee mediante el desarrollo de varias pruebas que serán realizadas por separado. El número y contenido de las pruebas puede variar en cada convocatoria, según recoge la normativa específica de cada edición.

Equipos.

Los equipos podrán estar formados por un máximo de tres personas, no pudiendo pertenecer una misma persona a equipos distintos. En cada equipo habrá una persona que hará de **portavoz** representando al equipo y a quien se le informará tanto de los posibles cambios en las bases como de las decisiones de los jueces durante el transcurso del concurso.

El **portavoz** de cada equipo será la persona encargada de depositar y poner en marcha el robot durante el desarrollo de las pruebas. No se podrá cambiar de portavoz durante la competición a no ser que exista una causa de fuerza mayor que lo justifique.

Cada equipo se identifica por el robot o robots que haya inscrito oficialmente, Cada equipo no puede inscribir más de un robot por cada prueba. Por tanto, el número máximo de robots por cada equipo será menor o igual al de pruebas en cada edición. En caso de mayor número de robots se inscribirán como equipos distintos. Los robots de un equipo tienen que tener una estructura base común.

Inscripción.

Los participantes deberán inscribirse por email (concurso.ceabot@gmail.com) indicando el nombre, email y teléfono de cada uno de los miembros del equipo. Así mismo, indicarán tanto quién va a ejercer de responsable, datos el centro y/o universidad por el que se presentan como las características técnicas de cuantos robots vayan a usar.

Bases.

Estas normas se tienen como fundamentales y se han de respetar. La participación en el "Concurso de Robots Humanoides" implica la **total aceptación** de estas bases. Se dividen en dos secciones: la primera: NORMATIVA GENERAL, es aplicable en cada edición y rige los aspectos comunes; la segunda: NORMATIVA REGULADORA DE LAS PRUEBAS: se revisa casa año para ajustarla a las nuevas pruebas propuestas.

Cambio de reglas.

Estas bases pueden ser modificadas por la Organización. Ésta comunicará a los equipos toda modificación que se pudiese realizar con suficiente antelación. En caso de haber cambios, éstos se comunicarán a los portavoces de los equipos tan pronto como sea posible, y siempre antes de la realización de las pruebas.

Los jueces.

Los jueces se encargarán de tomar todas las decisiones oportunas durante el transcurso de la competición, referentes a descalificaciones, ganadores o pruebas nulas. Es por tanto, de ellos la última palabra en la interpretación de estas bases.

Expulsión de la competición.

En casos extremos, los jueces se reservan el derecho a expulsar de la competición a quienes se crean merecedores de dicha penalización.

Objeciones.

El portavoz del equipo puede presentar sus objeciones a los jueces, en caso de tener cualquier duda en la interpretación de las normas. Sólo se admitirán objeciones antes de que comience cada prueba. Si estas objeciones dan como lugar una modificación en el reglamento, los jueces informarán inmediatamente al resto de portavoces.

Presentación oficial.

La presentación oficial es obligatoria. La no asistencia implica la descalificación del equipo. Se realizará durante la celebración del concurso, y siempre antes de que comience la competición al principio de cada prueba. En este acto se realizarán los sorteos de grupo, así como la explicación del desarrollo de la competición, se repasan brevemente las reglas y se aprovechará también para dejar los robots en la mesa de los jueces.

Excepciones.

En caso de que ocurra cualquier circunstancia no contemplada en los artículos anteriores de la prueba, los jueces adoptarán la decisión oportuna.

Robots.

Los robots han de tener una constitución antropomórfica, es decir dos piernas un tronco y dos brazos articulados. La altura máxima es de 50 cm. y la máxima longitud del pie de 11 cm. Se entiende por altura máxima la distancia desde el suelo a la parte más alta del robot cuando éste se encuentra completamente estirado. Se entiende por máxima longitud del pie a la distancia entre sus dos puntos más alejados. En la Figura 1 se puede ver algunos ejemplos de esto. El peso máximo permitido es de 3 Kg. Estas restricciones no son aplicables para la prueba libre.

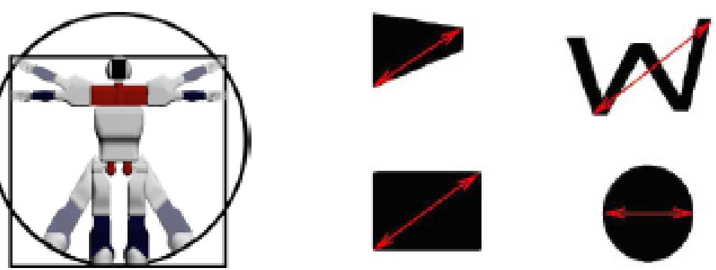


Figura 1: Ejemplo de Humanoide antropomórfico, y máxima longitud del pie.

Locomoción.

El modo de locomoción deberá ser andar o correr a dos patas, no pudiéndose utilizar ruedas, patines o similares.

Autonomía.

Cada robot debe ser completamente autónomo a nivel de locomoción, sensorización y procesamiento. Actuadores, sensores, energía y procesado deben estar incorporados en el robot, debiendo éste tomar sus propias decisiones.

No se podrá dar ninguna instrucción directa o indirectamente al robot tras encenderlo. Además, tras encender el robot, éste deberá esperar 5 segundos antes de realizar cualquier movimiento.

Modificaciones sobre el robot.

- Las pruebas son llevadas a cabo por un robot humanoide por cada equipo.
- Los equipos podrán utilizar un robot diferente para cada prueba, siempre y cuando en la inscripción se hayan inscrito todos los robots que presenta en el equipo. Por tanto, los puntos para cada prueba se imputan al equipo.
- Sólo se podrá cambiar de robot durante una prueba, en caso de incapacitación del primer robot utilizado para dicha prueba y con el consentimiento de los jueces.
- El código del robot no podrá ser modificado una vez haya comenzado cada una de las pruebas de la competición. Para ello, los robots deberán permanecer en la mesa de los jueces hasta el momento de su participación. Previa solicitud a los jueces se podrán hacer arreglos sencillos sobre sensores o servos dañados en la realización de las pruebas.

Seguridad.

El robot no puede poseer ningún elemento que suponga un peligro para él, los otros robots, el campo de pruebas o las personas.

Reparto de puntuación entre pruebas

El reparto de la puntuación entre las pruebas de habilidad/movilidad y sumo se hará al 70% / 30% de la siguiente forma:

- La prueba de movilidad supone un 35% (Prueba 1) del total de puntos posibles.
- La prueba de habilidad supone un 35% (Prueba 2) del total de puntos posibles.
- Por lo tanto, la prueba de sumo supone un 30% de los puntos totales.

Prueba 1: Carrera de obstáculos

Artículo 1.1. Objetivo.

Los robots irán desde un extremo del campo al otro, y vuelta al punto de partida, caminando de cara. El robot deberá esquivar los obstáculos, sin tirarlos ni desplazarlos de su posición.

Los robots saldrán desde la zona central, situada en la Zona de Salida, debiendo llegar a la Zona de Llegada Parcial. Una vez allí, el robot deberá darse la vuelta de forma autónoma e iniciar el proceso de vuelta, una vez haya traspasado por completo la línea de la Zona de Llegada Parcial.

Para puntuar se tendrá en cuenta tanto el tiempo transcurrido, como la distancia, y también el número de penalizaciones.

Los jueces decidirán la posición de los obstáculos a esquivar, antes de cada uno de los dos intentos de los que disponen los equipos. La configuración de los obstáculos, será igual para todos los equipos en cada intento. En el apéndice, se pueden observar algunas configuraciones posibles y algunos casos especiales a tener en cuenta en la programación del robot.

Habrá un máximo de 6 obstáculos paralelepípedos rectangulares, cuyas medidas serán de 20x20x50cm. Estarán hechos de cartón rígido o material similar y serán en la medida de lo posible inamovibles por un robot. Los obstáculos estarán dispuestos en el escenario dejando un hueco de paso adecuado de al menos 50 cm.

Artículo 1.2. Campo.

El campo de pruebas es una superficie nivelada, plana y rígida, de 2.5 m. de largo por 2 m. de ancho, formado típicamente por tableros de conglomerado y revestimiento de melamina o pintados. El color de la superficie será verde y homogéneo en la medida de lo posible (Pantone Code: 16C606 (R:22:G:198:B:6)). El campo está dividido por líneas blancas en tres zonas como se aprecia la figura 2. Alrededor del campo habrá una pared de 50 cm. de altura de un color distinto al del campo, preferiblemente blanco.

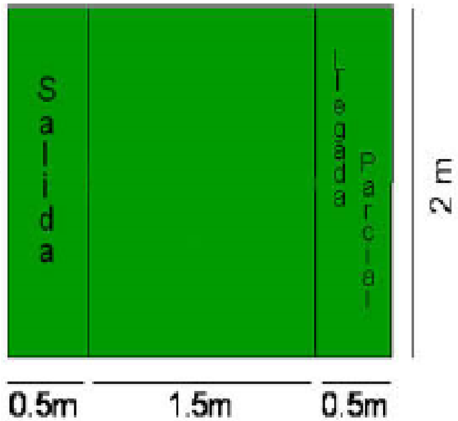


Figura 2: Esquema del Campo durante la Prueba 1.

Es posible que el campo esté compuesto por tableros ensamblados entre sí. En la medida de lo posible las uniones entre tableros se realizarán con machihembrado o centradores para que no presenten escalones y desniveles entre ellos.

El campo estará iluminado con luz artificial de interior, que será lo más uniforme posible. Se debe tener en cuenta que durante la realización de las pruebas puede haber flashes procedentes de cámaras fotográficas de público o prensa y alterar posibles sistemas de visión a bordo de los robots.

Durante la realización de las pruebas se advertirá al público para que se abstenga el uso de flashes procedentes de cámaras fotográficas pero esta circunstancia no podrá ser controlada durante el transcurso de la competición.

Artículo 1.3. Tiempo máximo.

El tiempo máximo para esta prueba es de 5 minutos por parcial. El tiempo comenzará a contar cuando el robot, después de realizar la pausa de 5 segundos, se ponga en movimiento. Se considera que el robot ha terminado un parcial cuando haya cruzado **completamente** la línea de final de ese tramo.

Artículo 1.4. Manipulación del robot y penalizaciones.

Durante el transcurso de las pruebas solo podrá manipular el robot el jurado y solo lo podrá hacer en el caso que el robot caiga o bien que el robot desplace o se quede bloqueado por un obstáculo. Por cada manipulación se obtendrá una penalización. La recolocación del robot tras una caída se realizará en la misma posición donde se ha producido la caída pudiendo reorientar el robot en ±45º respecto a la dirección de caída.

Artículo 1.5. Puntuación.

Para calcular la puntuación se tendrá en cuenta la distancia recorrida por el robot, el tiempo necesario y las penalizaciones. La filosofía detrás de la manera de calcular la puntuación es que cuanto más lejos llegue el robot en el menor tiempo posible y con la mínima intervención humana, más alta será la puntuación.

La puntuación se calcula de forma independiente, y de la misma manera, para los recorridos de ida y vuelta. La puntuación total es la suma de los dos recorridos. La puntuación para un recorrido nunca puede ser negativa y afectar así a la puntuación del otro recorrido.

Los jueces sólo medirán el tiempo T(s), la distancia recorrida d(cm) y el número de penalizaciones, obteniendo la puntuación de la siguiente forma:

$$P = \frac{(T_{MAX}(s) - T(s)) \cdot k_T}{T_{MAX}(s)} + \frac{d(cm)}{d_{MAX}(cm)} \cdot k_D - (2 \cdot pen)^{k_P}$$

donde:

- $T_{MAX}(s)$ es el tiempo máximo para realizar el recorrido en segundos (300 s).
- $T(s)$ es el tiempo que ha necesitado el robot para completar el recorrido en segundos. Si el robot no completa el recorrido este valor será igual al tiempo máximo.
- $d(cm)$ es la distancia a la parte posterior del pie recorrida por el robot en centímetros dentro del tiempo permitido. Si el robot llega al final del recorrido antes de que se agote el tiempo este valor será igual a la distancia máxima.
- $d_{MAX}(cm)$ es la distancia de un recorrido en centímetros (150 cm).
- *pen* es el número de penalizaciones que ha realizado el robot en un recorrido
- k_T, k_D y k_P son las constantes de tiempo, distancia y penalizaciones que permiten escalar de forma adecuada las puntuaciones.

El factor de las penalizaciones se calcula como una potencia para permitir que un robot estable que requiera pocas intervenciones humanas no sea penalizado en exceso, pero robots más inestables y que requieran continua intervención humana tengan una gran penalización. Este factor mide la autonomía real del robot.

Los factores de distancia y tiempo son lineales y valoran la agilidad del robot para superar los obstáculos. Cuanto más lejos llegue un robot y con el menor tiempo, más puntuación tendrá.

Las constantes se han obtenido con los siguientes criterios:

- La puntuación por tiempo y por distancia son del mismo orden.
- En general, para una penalización por cada tercio de circuito, la puntuación final, sin tener en cuenta el tiempo, será baja, pudiendo llegar a ser nula.

Con estos criterios, los valores de las constantes utilizadas son:

- $k_T = 7.5$
- $k_D = 7.5$
- $k_P = 1.35$

Se puede dar el caso que un robot que haya recorrido más distancia quede por detrás de otro que haya completado una menor parte del circuito pero que tenga menos penalizaciones, si la diferencia de distancia no es muy grande (<1/3 del circuito).

La puntuación obtenida con la ecuación anterior sólo sirve para generar la clasificación de la prueba de la forma más justa según las capacidades del robot. **La puntuación final** será asignada según la clasificación como se muestra en la siguiente tabla:

Puesto	Puntos
1	35
2	29
3	25
4	21
5	17
6	13
7	9
8	5
9	2

A nivel de ejemplo, se presentan los siguientes casos, asumiendo el mismo tiempo para todos ellos, y como quedaría la clasificación:

Caso	Distancia	Penal.
A	0.5 m	0
B	0.5 m	1
C	0.5 m	2
D	0.5 m	3
E	1 m	0
F	1 m	1
G	1 m	2
H	1 m	3
I	1.5 m	0
J	1.5 m	1
K	1.5 m	2
L	1.5 m	3

La clasificación en este caso quedaría de la siguiente manera:

Puesto	Caso
1	I
2	E
3	J
4	A
5	F
6	K
7	B
8	G
9	L
10	C
11	H
12	D

En los casos I, J, K y L la clasificación podría cambiar porque en estos casos se ha completado el circuito y se tendría que tener en cuenta la puntuación por tiempo.

Prueba 2: Escalera

Artículo 2.1. Objetivo.

En esta prueba, se añade una escalera al escenario de la primera prueba, una vez retirados los obstáculos.

Para superar la prueba, los robots deberán alcanzar la Zona de Llegada indicada por el juez, superando una serie de escalones de subida y de bajada.

Se puntuará tanto el número de escalones superados como el tiempo empleado.

Los robots deberán subir y bajar la escalera caminando, no siendo permitido ningún tipo de salto o acrobacia. La escalera solo se recorre en un sentido, siendo éste elegido por los jueces. Se finalizará la prueba cuando se haya sobrepasado totalmente la línea de Salida o Llegada Parcial, según el sentido de comienzo de la prueba indicado por los jueces. Se puntuará la habilidad de superar la escalera de forma autónoma, valorando el que el robot supere escalones sin que caiga o se desvíe y penalizando cualquier intervención por parte del portavoz del equipo.

Artículo 2.2. Campo.

Los jueces decidirán la colocación de la escalera antes de cada ronda de intentos, teniendo en cuenta que uno de los extremos deberá cubrir completamente una de las líneas sin sobrepasarla. De este modo, quedarán por el otro extremo 20 cm. hasta la línea de Salida o Llegada Parcial.

Las escaleras tendrán unos escalones de 3 cm. de altura y de longitud 25, 15 y 50 cm. según se indica en la figura 3. Además las escaleras, tendrán un ancho de 99 cm.).

Normativa CEABOT 2013 Pág. 8

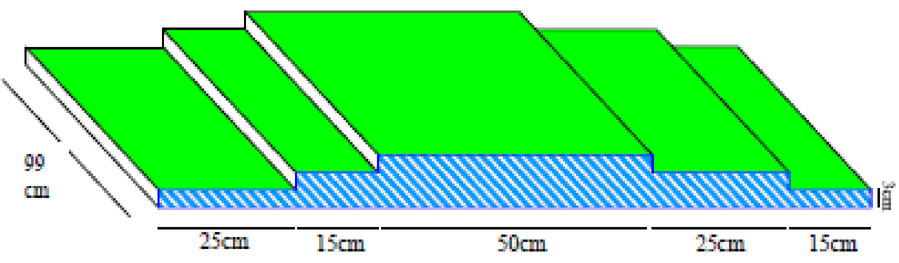


Figura 3: Esquema de las escaleras.

Artículo 2.3. Tiempo máximo.

El tiempo máximo de cada carrera es de 5 minutos. El tiempo comenzará a contar cuando el robot, después de realizar la pausa de 5 segundos, se ponga en movimiento. Se considera que el robot ha terminado un parcial cuando haya cruzado **completamente** la línea de final de ese tramo.

Artículo 2.4. Manipulación del robot y penalizaciones.

En caso que el robot caiga y no pueda levantarse, los jueces serán los únicos que podrán colocar el robot en el peldaño desde el que se ha caído o el siguiente según, le indique el portavoz del grupo del robot, obteniendo una penalización.

Si el robot toca con mano/brazo la superficie de la escalera será penalizado aunque no llegue a caer. Nótese que si el robot, cae pero se recupera y vuelve a quedarse de pie en el mismo escalón del que se cayó, no será penalizado.

Artículo 2.5. Puntuación.

En primer lugar se tendrán en cuenta el número de penalizaciones, de menor a mayor. En caso de empate a penalizaciones, se tendrá en cuenta, primero el número de escalones superadas y a igualdad de estos, el tiempo para decidir las posiciones finales de los equipos.

Se puntuará, por una parte teniendo en cuenta el orden de llegada a la zona Llegada Parcial. La puntuación se muestra en el cuadro 2. Si un robot no llega a la Llegada Parcial, solo recibirá puntos por los escalones superados.

Posición Llegada parcial	Puntos
1º	10
2º	8
3º	7
4º	6
5º	5
6º	4
7º	3
8º	2
9º	1

Nº escalones superados	Puntos
1	3
2	5
3	10
4	18
5	22
6	25

Cuadro 2: Puntuaciones para la segunda prueba.

Por tanto, el robot que complete **libre de penalizaciones** los 6 escalones en menos tiempo que sus oponentes, recibirá 35 puntos.

Prueba 3: Lucha (Sumo)

Articulo 3.1. Objetivo de la prueba.

En la prueba luchan dos robots de dos equipos diferentes, dentro del Área de Combate según las normas que a continuación se expondrán, para obtener puntos efectivos (llamados puntos Yuhkoh). Se valora el comportamiento competitivo del robot, pudiéndose penalizar actitudes pasivas e inmóviles.

Artículo 3.2. Definición del área de combate.

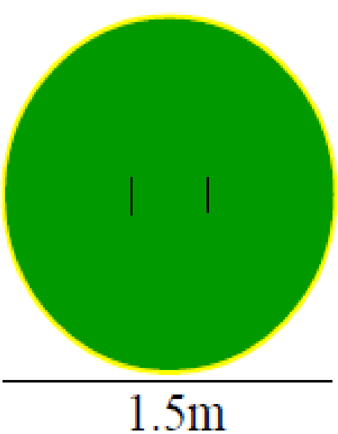
Se denomina Área de Combate a la tarima de juego (Ring). Cualquier espacio fuera del Área de Combate se denomina Área Exterior, que deberá ser de al menos 0.5 m. alrededor del Ring (Figura 3).

Artículo 3.3. Ring de sumo.

El Ring será circular, de color verde, homogéneo en la medida de lo posible y de 150 cm. de diámetro. Señalando el límite exterior del Ring, habrá una línea blanca o amarilla circular de 5 cm. de ancho y no habrán paredes ni otros obstáculos a menos de 0.5m del límite exterior..

En el centro del Ring habrá dos líneas paralelas separadas 20 cm., llamadas líneas Shikiri. Las líneas Shikiri serán de color negro o blanco de 2 cm. de ancho y 20 cm. de largo. Estas líneas marcarán las posiciones iníciales de los robots.

El campo estará iluminado con luz artificial de interior, que será lo más uniforme posible.



1.5m

Figura 3. Estructura del Ring

Artículo 3.4. Combates de sumo.

Los combates consistirán en 3 asaltos de 2 minutos cada uno. Entre asalto y asalto habrá un tiempo máximo de 1 minuto.

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

Ganará un asalto el que consiga más puntos Yuhkoh durante ese asalto o el que consiga 5 o más puntos Yuhkoh.

Ganará el combate el que haya ganado más asaltos. Es decir, los combates serán al mejor de 3 asaltos. En caso de empate a asaltos se desempatará con la suma total de los puntos Yuhkoh conseguidos. En caso de empatar a suma de puntos Yuhkoh se hará un asalto más.

Artículo 3.5. Rutina del combate.

Siguiendo las indicaciones de los jueces, los equipos se saludarán en el Área Exterior. A continuación, sólo entrará en el Área de Combate el portavoz del equipo, situando el robot centrado detrás de la línea Shikiri.

Cuando el juez lo indique, se activarán los robots, que deberán permanecer parados durante 5 segundos. Tras dicha pausa, comenzará el asalto.

Únicamente se podrá acceder dentro del Área de Combate cuando el asalto esté parado y/o den permiso los jueces. Cuando el árbitro dé por finalizado el combate, los dos portavoces de equipo retirarán los robots del Área de Combate.

Artículo 3.6. Puntos Yuhkoh.

Se otorgarán puntos Yuhkoh al robot de un equipo cuando:

- El robot contrario toca el espacio fuera del Ring, 1 punto.
- El robot contrario toca con alguna mano el suelo sin caer, 1 punto.
- El robot contrario cae al suelo por sí mismo, 1 punto.
- El robot contrario cae al suelo tras lanzar un ataque, 1 punto.
- Por tumbar al robot contrario dentro o fuera del Ring, 2 puntos.

Cuando se cumplan varias condiciones, sólo se otorgarán los puntos de una de ellas, siendo ésta la de mayor puntuación.

En caso de empate, como por ejemplo, caer los dos robots a la vez sin haberse producido un ataque, no se otorgará ningún punto. En caso que un robot inicie un ataque y se caiga, pero también caiga el robot contrincante, el robot que inició el ataque recibirá 2 puntos y el otro robot 1 punto. En este mismo caso si el robot atacante se levanta por si solo del suelo, recibirá 2 puntos y el otro ninguno.

Artículo 3.7. Parada del combate.

Cada asalto durará 2 minutos, pudiéndose parar y reanudarse hasta agotar el tiempo, cuando:

- Cuando el juez otorgue algún punto Yuhkoh.
- Los dos robots permanezcan 30 seg. sin moverse.
- Los dos robots permanezcan 30 seg. sin tocarse.
- Los dos robots permanezcan 30 seg. empujándose pero sin que el movimiento favorezca a ninguno de los equipos.

Para reanudar el combate, tras cada asalto o tras una pausa, se colocarán de nuevo los robots en las líneas Shikiri.

Artículo 3.8. La organización del concurso.

El número de equipos por grupo y las clasificaciones que dan derecho a pasar a la fase final, se decidirán en función del número de inscritos, y se comunicarán antes del comienzo de la prueba.

Los grupos y turnos de combate se sortearán antes del comienzo de la prueba. Dependiendo del número de rondas que se realicen (fases previas, octavos, cuartos, etc.) se podrán establecer cabezas de serie, de forma que sea mayor la competitividad.

Artículo 3.9. Puntuación.

La puntuación de las pruebas eliminatorias, será como sigue:

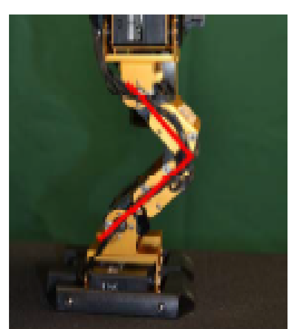
Posición	Puntos
1º	30
2º	25
3º	21
4º	18
5º	15
6º	12
7º	9
8º	6
9º	3

Cuadro 3: Puntuación de la tercera prueba

En caso de quedar campeón en la prueba de sumo, el robot recibirá 30 puntos. Los robots calificados por debajo del noveno no reciben puntos en esta prueba.

Artículo 3.10. Descalificación de robots.

Para fomentar la competición, los robots no podrán permanecer con las rodillas dobladas más de 90º excepto para movimientos puntuales de ataque o para andar, movimientos que no podrán superar los 10s. De esta forma será más sencillo poder derribar al contrincante y los combates serán más dinámicos y divertidos (Figura 4).



Bioloid con un ángulo superior a 90° RoboNova con un ángulo igual a 90°

Figura 4. Angulo de rodilla.

Los robots también podrán ser descalificados por actitudes pasivas, inmóviles o programaciones aleatorias de ataques en el caso que el jurado lo detecte o que uno de los portavoces de alguno de los robots combatientes lo comunique al jurado al final de cualquier asalto. La comprobación de estas actitudes se realizará mediante el test de pasividad. Si es el portavoz de un equipo quien sugiere al jurado la realización de la prueba al robot oponente, el jurado podrá rechazar dicha petición si no encuentra razones suficientes para su ejecución.

El **test de pasividad** consiste en colocar un señuelo (caja de medidas aproximadas 25x25x50cm) delante del robot a 20cm, como si fuera un oponente al inicio de un asalto. El robot deberá tocar el señuelo en un tiempo máximo de 30 segundos (después de los 5 segundos de espera pertinentes). El test podrá repetirse en dos ocasiones. En caso de no superar el test en ninguno de los dos intentos, el robot se considerará no válido para la competición y será descalificado. Esta regla de descalificación pretende motivar la movilidad, el uso de sensores y la creación de algoritmos inteligentes de sumo.

PRUEBA LIBRE -EXHIBICIÓN

El objetivo de esta prueba es que los equipos que lo deseen puedan demostrar las habilidades y capacidades que han sido capaces de programar en su robot humanoide, bien porque sean diferentes de aquellas demostradas en las pruebas regladas, o bien por que los robots excedan alguno de los parámetros de tamaño y peso especificados en la normativa. El tema o exhibición será libre y la única limitación será el tiempo, que no superará los 5 minutos. Será el equipo inscrito en esta prueba el responsable de la infraestructura necesaria para el desarrollo de la misma.

Para participar en la modalidad de prueba libre será preciso en el momento de la inscripción notificarlo expresamente, especificando el tipo de robot (dimensiones y breve descripción del mismo si no es un modelo comercial) y habilidades o pruebas que van a realizar. Durante la ejecución del concurso, el jurado asignará un horario para la realización de estas exhibiciones en función del número de inscripciones.

Para la prueba libre-exhibición, no es necesario cumplir las especificaciones de dimensiones y peso.

EQUIPO GANADOR

Tras completar cada prueba el equipo del jurado presentará el recuento de puntos otorgados a cada equipo en cada prueba y la posición global.

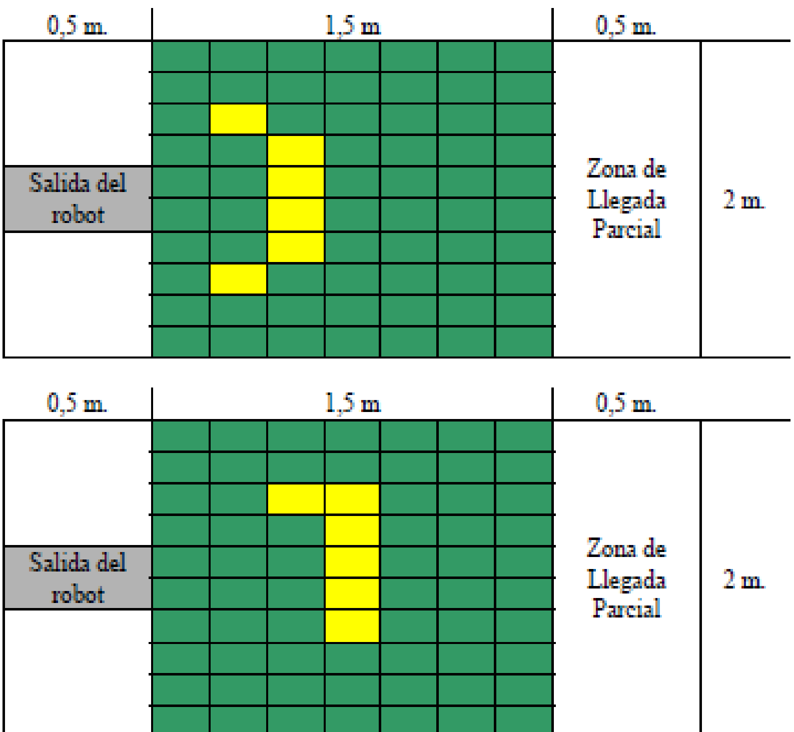
Tras completar las tres pruebas, el jurado presentará una tabla mostrando los puntos obtenidos por cada equipo en cada una de las pruebas, junto con la clasificación global. El equipo con más puntos, será el ganador

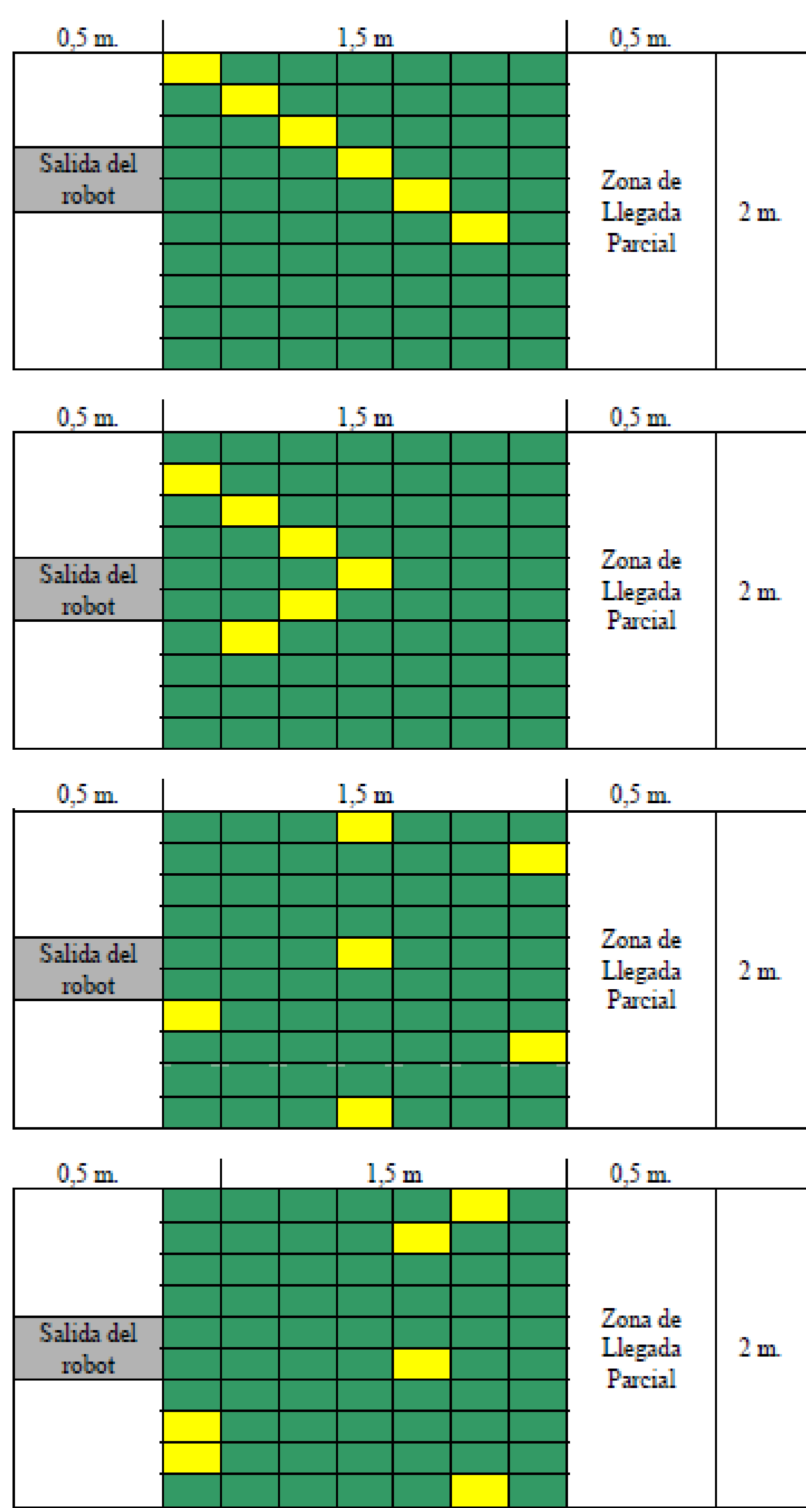
La organización se reserva el derecho de otorgar un reconocimiento al equipo que haya causado mejor impresión en la realización de la prueba libre.

Apéndice: Configuraciones de ejemplo para la carrera de obstáculos

Tal y como se indica en la normativa de la prueba, los jueces elegirán una combinación de obstáculos, antes de cada intento. Los participantes, cogerán el robot (que deberá estar encima de la mesa del jurado) y sin ningún tipo de comando externo (es decir, no se puede usar un mando a distancia o similar) activarán el robot. Una vez finalizado el intento, el robot volverá a colocarse encima de la mesa de los jueces. Cuando acabe la primera tanda de intentos, los jueces modificarán la posición de los obstáculos y se procederá a la siguiente tanda.

A continuación se muestran posibles configuraciones del campo, a tener en cuenta por los participantes, los jueces tienen plena capacidad para distribuir los obstáculos según crean conveniente sin necesidad de utilizar alguna de las configuraciones que se exponen a continuación. Las celdas en amarillo indican la localización aproximada de un obstáculo.




0,5 m.
1,5 m
0,5 m.
Salida del robot
Zona de Llegada Parcial
2 m.
0,5 m.
1,5 m
0,5 m.
Salida del robot
Zona de Llegada Parcial
2 m.
0,5 m.
1,5 m
0,5 m.
Salida del robot
Zona de Llegada Parcial
2 m.
0,5 m.
1,5 m
0,5 m.
Salida del robot
Zona de Llegada Parcial
2 m.
Normativa CEABOT 2013
Pág. 16