DEPARTAMENT D'ENGINYERIA I CIÈNCIA DELS COMPUTADORS

Simulaciones Hiperrealistas para Robótica Educativa

TESIS DOCTORAL

Autor:
JAIME ALEMANY JULIÀ

Director:
ENRIC CERVERA I MATEU

Castelló, octubre de 2015

Resumen

Esta tesis se centra en el desarrollo de simulaciones que explotan las capacidades del hardware y del software actuales para ofrecer experiencias más didácticas y atractivas en robótica educativa. Para conseguir tales simulaciones he examinado las teorías del aprendizaje con entornos virtuales, extrayendo cuáles son los elementos clave que incrementan la motivación de los alumnos y mejoran el aprendizaje. Después he realizado un estudio sobre los simuladores robóticos actuales con el objetivo de determinar cuáles de ellos son los más adecuados para la incorporación de tales elementos clave, escogiendo un subconjunto de los simuladores más representativos del estado del arte. Los simuladores elegidos los he puesto a prueba durante el desarrollo de varios entornos de simulación con diversos propósitos educativos, los cuales van desde la implementación de algoritmos para sistemas inteligentes hasta las competiciones robóticas entre estudiantes o la programación online de robots remotos. Como conclusiones de esta investigación destaco las grandes diferencias que he encontrado entre los simuladores cuando se trata de construir mundos virtuales o introducir nuevas plataformas robóticas, decantándose la balanza claramente hacia los simuladores basados en motores de juegos. Tales simuladores, desde el punto de vista del docente desarrollador de entornos, permiten explotar el hardware actual y las tecnologías software más vanguardistas para obtener los mayores grados de realismo, facilitando la incorporación de los elementos clave que proporcionan experiencias didácticas y atractivas. En un marco donde la robótica educativa gana cada vez mayor importancia, resulta notable la falta de aprovechamiento tecnológico (tanto a nivel de hardware como de aplicaciones software) que en general muestran los simuladores de plataformas robóticas no basados en motores de juegos, los cuales dan la impresión de estar diseñados para fines más cercanos a la investigación que a la educación.

Agradecimientos

Me gustaría agradecer el apoyo de mi familia y la ayuda del doctor Enric Cervera del Departamento de Ingeniería y Ciencia de los Computadores de la Universitat Jaume I de Castelló de la Plana. Su paciencia y su colaboración durante los últimos años me ha permitido superar numerosas dificultades y llevar a cabo esta tesis.

Contenidos

Resumen i

Lista de figuras ix

Capítulo 1 1
INTRODUCCIÓN 1
1.1. Robots en educación 1
1.2. Videojuegos y simuladores en educación 4
1.3. Robots simulados vs robots reales 7
1.4. Esquema de la tesis 10

Capítulo 2 11
ESTADO DEL ARTE 11
2.1. Simuladores 12
2.2. Middlewares 23
2.3. Motores de videojuegos para el desarrollo de juegos serios 25

Capítulo 3 31
METODOLOGÍA Y HERRAMIENTAS 31
3.1. Revisión de las teorías sobre aprendizaje con entornos virtuales 33
3.2. Selección de simuladores para robótica educativa 34
3.3. Implementación de simulaciones didácticas y atractivas 34
3.4. Pruebas de rendimiento e integración 43

Capítulo 4 45
SIMULADOR PARA UN LABORATORIO DE ROBÓTICA. ANTECEDENTES..... 45
4.1. Contexto: Proyecto GUARDIANS y máster de Sistemas Inteligentes 45
4.2. Baja fidelidad general en los simuladores de 2008 46
4.3. Síntesis de mi experiencia con USARSim para UT2004 47
4.4. Desarrollo de un simulador con USARSim v.3.3 51
4.5. Conclusiones 86

Capítulo 5 87
SELECCIÓN DE SIMULADORES 87
5.1. Revisión de las teorías del aprendizaje con entornos virtuales 87
5.2. Criterios generales para la selección de simuladores 95
5.3. La inmersión en entornos virtuales 99
5.4. Selección de simuladores para robótica educativa 101

Capítulo 6 107
SIMULACIONES EN DOCENCIA SOBRE INTELIGENCIA ARTIFICIAL 107
6.1. Plataforma de software para Sistemas Inteligentes 107
6.2. Selección de un simulador para sistemas inteligentes 110
6.3. USARSim UDK v.1.2 en simulaciones para Sistemas Inteligentes 111
6.4. Conclusiones 137

Capítulo 7 139
SIMULACIÓN DE COMPETICIONES ROBÓTICAS 139
7.1. Sobre la enseñanza de la robótica en la UJI. 139
7.2. Marco de competiciones robóticas simuladas 143
7.3. Lucha de sumo. Entornos virtuales para competiciones robóticas 145
7.4. Lucha de boxeo con USARSim v.1.2 para UDK 164
7.5. Competiciones domésticas con USARSim 168

7.6. Resolución de un laberinto con USARSim 174
7.7. Liga de fútbol con USARSim v.1.2 para UDK 176
7.8. Entorno doméstico multicompetición. Otras características relevantes 178
7.9. Conclusiones 181

Capítulo 8 185
SIMULACIONES ONLINE 185
8.1. Introducción 185
8.2. Programación online de robots remotos 187
8.3. Entornos para simulación online 193
8.4. Pruebas experimentales de R^3PO con USARSim 208
8.5. Conclusiones 211

Capítulo 9 213
CONCLUSIONES 213
9.1. Resumen de conclusiones 213
9.2. Contribuciones 214
9.3. Publicaciones 214
9.4. Trabajo futuro 216

REFERENCIAS 217

Lista de figuras

Figura 1. Kits robóticos de carácter educativo 2
Figura 2. Número de estudiantes matriculados en Ingeniería Informática 3
Figura 3. Competiciones RoboCup. 3
Figura 4. Competiciones CEABOT de robots luchadores de sumo 4
Figura 5. Relación entre videojuegos y simuladores 5
Figura 6. Capturas de pantalla de los simuladores Stage y Gazebo 9
Figura 7. Capturas de pantalla del simulador MORS 13
Figura 8. Entorno para competiciones de lucha de sumo con Gazebo 2.2 14
Figura 9. Capturas de pantalla del simulador SARGE 15
Figura 10. Ensayos de agarre utilizando el módulo simplegrasping de OpenRAVE 16
Figura 11. Simulación de competiciones de fútbol con SimRobot 17
Figura 12. Robot NAO en un entorno altamente interactivo de USARSim 18
Figura 13. Simulaciones con Microsoft Robotics Developer Studio 19
Figura 14. Competición robótica simulada con Webots 7.4.1 20
Figura 15. Capturas de pantalla del simulador Marilou de anyKode 20
Figura 16. Capturas de pantalla del simulador simRobot Pro 21
Figura 17. Cronología de simuladores de plataformas robóticas 22
Figura 18. Principio de funcionamiento de RT Middleware 24
Figura 19. Capturas de pantalla de aplicaciones que usan CryENGINE 3 28
Figura 20. Capturas de pantalla correspondientes al motor Source de Valve 29
Figura 21. Capturas de pantalla de aplicaciones que usan el motor UT3 30
Figura 22. Capturas de pantalla de aplicaciones que usan el motor Unity 30
Figura 23. Metodología general para la obtención de simulaciones hiperrealistas 32
Figura 24. Atributos clave para obtener simulaciones didácticas y atractivas 33
Figura 25. Procedimiento general para el desarrollo de entornos de simulación 36
Figura 26. Proceso para la generación de un modelo 3D 38
Figura 27. Creación de un jarrón destructible mediante Nvidia Apex PhysX Lab 38
Figura 28. Flujo de archivos entre herramientas de desarrollo de simulaciones 40
Figura 29. Clases de objetos que componen un humanoide Nao 42
Figura 30. Idea principal del proyecto GUARDIANS 46

Figura 31. Fotografía del exterior del edificio TI del Campus Riu Sec de la UJI.....47
Figura 32. Vista exterior del edificio TI virtual y sus alrededores inmediatos.....48
Figura 33. Robot Erratic real y en simulación.....48
Figura 34. Modelos 3D para la simulación de los robots P2DX y Erratic.....49
Figura 35. Comportamiento del entorno del edificio TI y el robot Erratic 50
Figura 36. Partes en las que se divide el modelo virtual para el robot Erratic.....52
Figura 37. Obtención de un entorno de simulación con USARSim para UT2004.....53
Figura 38. Vistas del robot Erratic durante su modelado con 3D Studio Max 55
Figura 39. Buscador de texturas de Unreal Editor con texturas del robot Erratic 56
Figura 40. Sensor Laser en 3D Studio Max 57
Figura 41. Ordenador del robot Erratic con sus envolventes de colisión 58
Figura 42. Chasis del robot Erratic en el StaticMesh Browser de Unreal Editor 59
Figura 43. Sonar del robot Erratic en 3D Studio Max.....61
Figura 44. Entorno virtual con el edificio TI y su entorno inmediato 64
Figura 45. Plano de AutoCAD de la primera planta del edificio TI 65
Figura 46. Pared levantada en 3D Studio Max a partir de un plano en AutoCAD 67
Figura 47. Pared importada de 3D Studio max 67
Figura 48. Forma básica para crear un forjado o pavimento del edificio TI.....68
Figura 49. Techo de un despacho del edificio TI 69
Figura 50. Ventanas del edificio TI 70
Figura 51. Puertas del edificio TI 71
Figura 52. Texture Browser de Unreal Editor con texturas para simular vidrio.....72
Figura 53. Parámetros para el eje de giro de una puerta en Unreal Editor 74
Figura 54. Asignación de una malla como hoja de puerta automática 75
Figura 55. Grupo Mover de parámetros para una puerta automática 75
Figura 56. Grupo de parámetros Movement de un Mover.....76
Figura 57. Grupo de parámetros Events de un Mover.....76
Figura 58. Grupos de parámetros MoverSounds y Object de un Mover 76
Figura 59. Puerta de laboratorio con apertura mediante pulsador 77
Figura 60. Máquinas expendedoras de la planta baja del edificio TI de la UJI.....78
Figura 61. Combinación de luces con objetos autoiliminados 80
Figura 62. Propiedades gráficas de un objeto en Unreal Editor 3.....80
Figura 63. Pelota de fútbol en 3D Studio Max con textura realista 81
Figura 64. Pelota de fútbol en el StaticMesh Browser de Unreal Editor 3.....82
Figura 65. Vistas del objeto caramelo en 3D Studio Max.....83
Figura 66. Vistas del saco de boxeo en 3D Studio Max 84

Figura 67. Envolventes de colisión y articulaciones de un saco de boxeo 84
Figura 68. Aspecto en simulación del expendedor de juguetes.....86
Figura 69. Esquema de los resultados de aprendizaje 88
Figura 70. Relación entre resultados cognitivos y atributos de un simulador 92
Figura 71. Relación entre resultados procedimentales y atributos de un simulador .94
Figura 72. Relación entre resultados afectivos y atributos un simulador.....95
Figura 73. Plataforma de desarrollo de software basada en Python y USARSim..... 109
Figura 74. Robot humanoide NAO 110
Figura 75. Vista frontal de un laberinto para clasificación de imágenes..... 112
Figura 76. Vista global de un laberinto para clasificación de imágenes..... 112
Figura 77. Laberinto con las formas a reconocer y una fuente de distorsión..... 112
Figura 78. Comparativa entre modelos de humanoide NAO para USARSim 113
Figura 79. Tamaño del área captada según la distacia de observación 115
Figura 80. Tamaños de los símbolos captados según la distancia de observación.. 116
Figura 81. Minotauro recorriendo el laberinto 119
Figura 82. Templo díptero y elementos característicos del estilo jónico..... 117
Figura 83. Pedestal del interior templo con texturas simulando relieve 118
Figura 84. Modelo de capitel sin relieves. Efecto de un normal map 118
Figura 85. Tramo de muro exterior del laberinto..... 119
Figura 86. Normal map obtenido con NVIDIA Normal Map Filter 120
Figura 87. Bóveda celestial con luna y una fuente de luz direccional..... 122
Figura 88. Minotauro en su etapa final de desarrollo..... 125
Figura 89. UDK Particle System Editor mostrando un fuego..... 124
Figura 90. Características principales del robot humanoide NAO 125
Figura 91. Niveles de detalle de la cabeza del robot NAO 126
Figura 92. Comparativa de envolventes de colisión para un pié del robot NAO 127
Figura 93. Detalle de la mano del robot NAO 128
Figura 94. Secuencia de cierre de la mano de mi modelo de humanoide NAO..... 128
Figura 95. Grados de libertad y articulaciones del humanoide NAO 131
Figura 96. Robot NAO ejecutando un algoritmo de clasificación de imágenes 132
Figura 97. Evolución del error durante el entrenamiento de una red neuronal..... 133
Figura 98. Algoritmo para la extracción de una referencia en la imagen captada .. 134
Figura 99. Robot NAO implementando un algoritmo genético para gateo 136
Figura 100. Evolución de la función aptitud frente a generaciones..... 137
Figura 101. Competición de robots luchadores de sumo en la UJI 140
Figura 102. Evolución del porcentaje de estudiantes matriculados en robótica..... 140

Figura 103. Competición CEABOT'08 entre los equipos de la UJI y la UHU 141
Figura 104. Estudiantes de la asignatura robótica vs estudiantes de Informática... 142
Figura 105. RoboCup Soccer Standard Platform League..... 144
Figura 106. Combate de sumo simulado con Webots..... 146
Figura 107. Contorno de colisión del ring de sumo..... 147
Figura 108. Modelo de robot Pioneer 3AT con cámara y sensores de suelo..... 148
Figura 109. Sensores sonar para la detección de suelo en un Pioneer 3AT..... 150
Figura 110. Entorno Linux con Gazebo 2.2 mostrando un ring de sumo 151
Figura 111. Necesidad de sombras. Ave en dos espacios virtuales 153
Figura 112. Diferencias en los grupos de suavizado..... 156
Figura 113. Comparación entre modelos de robot Pioneer 3AT..... 156
Figura 114. Edificio TI de la UJI para simulaciones con USARSim v.1.2 para UDK.... 160
Figura 115. Detalle de un ring de sumo para simular competiciones robóticas..... 160
Figura 116. Pasillo de la primera planta del edificio TI. Versiones de USARSim 161
Figura 117. Bandera de la competición de sumo 162
Figura 118. Modelo de robot Pioneer 3AT incluido en USARSim..... 164
Figura 119. Ring para competiciones simuladas de boxeo con USARSim..... 164
Figura 120. Ring de boxeo durante un entrenamiento con un saco..... 165
Figura 121. Valla con material realista. 166
Figura 122. Texturas que componen el material de la valla del ring 166
Figura 123. Saco de boxeo visto en el editor de UDK y en 3D Studio Max 167
Figura 124. Entorno realista para simulación de competiciones robóticas 169
Figura 125. Edificio para simulaciones de entornos domésticos con USARSim..... 170
Figura 126. Planta baja de un entorno de simulación de tareas domésticas..... 171
Figura 127. Cocina para simulación de tareas cotidianas con USARSim..... 172
Figura 128. Lenguaje de programación Kismet de inteligencia artificial de UDK..... 172
Figura 129. Objetos que el robot ha de colocar dentro de unas cajas marcadas 173
Figura 130. Alienígena de juguete compuesto por cuerpos sólidos articulados..... 174
Figura 131. Vista general de un laberinto en base a paneles de madera 175
Figura 132. Elementos importantes de un entorno de simulación laberíntico..... 175
Figura 133. Campo de fútbol para simulación de una competición de fútbol..... 176
Figura 134. Dimensiones oficiales de un campo de fútbol para RoboCup..... 177
Figura 135. Campo de fútbol para humanoides NAO con USARSim..... 177
Figura 136. Animales que simulan existencia de vida en el entorno 179
Figura 137. Combinación de movimientos animados y simulados 180
Figura 138. Errores en grupos de suavizado en USARSim v1.2 para UDK..... 183

Figura 139. Vista general de R^{3}PO 188
Figura 140. Detalle de la interacción entre cliente y servidor de R^{3}PO 189
Figura 141. Programación remota con TurtleSim mediante R^{3}PO 189
Figura 142. Vista general de RPN desde el lado del usuario 190
Figura 143. Lista de los mejores registros del simulador TurtleSim de RPN 191
Figura 144. Estructura general de RPN 192
Figura 145. Vista general del entorno de ShanghAI Lectures 194
Figura 146. Versión USARSim del entorno para IA de ShanghAI Lectures 195
Figura 147. Gestor gráfico de IA de UDK (Kismet) 197
Figura 148. Robot NAO entrando en el sector del laberinto 199
Figura 149. Sound Cue Editor de UDK 200
Figura 150. Perspectiva del sector de la piscina con la red de sensores 200
Figura 151. Editor de fracturas de NVIDIA APEX 1.1 PhysX Lab 202
Figura 152. Evolución de un jarrón destructible en simulación con USARSim 202
Figura 153. Cocina utilizada en la competición HUMABOT Challenge 2014 204
Figura 154. El robot NAO apaga un fogón accionando un pulsador 204
Figura 155. Set de productos reales y simulados para HUMABOT Challenge 205
Figura 156. Un robot NAO coge un tomate y lo deja dentro de una cacerola 205
Figura 157. Robot NAO programado de manera online a través de RPN 206
Figura 158. Entornos domésticos de Webots 7 para HUMABOT Challenge 208
Figura 159. Programación remota del humanoide NAO con USARSim y R^{3}PO 210

Capítulo 1

INTRODUCCIÓN

En este capítulo expongo la motivación que impulsa esta tesis y sus principales contribuciones.

1.1. Robots en educación

En la última década el uso de robots en educación ha crecido significativamente en todos los niveles formativos, desde preescolar hasta las universidades pasando por los institutos de enseñanza secundaria [1]. Hoy es habitual el uso de robots para estimular el proceso de aprendizaje en cursos de introducción a la programación [2] o para reforzar principios de robótica del comportamiento en estudiantes de diferentes disciplinas (informática, ingeniería, psicología) con conocimientos de programación limitados [3]. Esta expansión de los robots desde los laboratorios hacia las aulas se debe a la conjunción de varios factores de naturaleza diversa, entre los que destacan fundamentalmente los tres que a continuación se detallan.

En primer lugar, es importante subrayar la capacidad de los robots para encarnar conceptos relacionados con la ciencia, la tecnología, la ingeniería y las matemáticas (STEM) de una manera muy comprensible. Esto resulta especialmente apreciado por los estudiantes e investigadores de áreas de informática como Sistemas Inteligentes [4] [5] [6]. Un claro ejemplo lo encontramos en enseñanzas sobre Robótica e Inteligencia Artificial. Aquí, ya sea en el paradigma clásico [7] o en la visión encarnada [8], las plataformas de robots móviles se han utilizado históricamente como puntos de referencia en la implementación de los sistemas inteligentes, especialmente aquéllos relacionados con la percepción y la acción [9] [10] [11]. Así, los robots proporcionan un modelo físico que permite demostrar de manera visual conceptos e ideas tradicionalmente enseñados utilizando abstracciones.

En segundo lugar, destaca la creciente disponibilidad de innovadoras plataformas robóticas con marcado carácter educativo y precios asequibles. Claros ejemplos de éstas son los humanoides Robonova 1 y los kits de Lego Mindstoms (ver Figura 1). De hecho, estos últimos kits han sido desarrollados de acuerdo con los principios educativos derivados de las teorías del desarrollo cognitivo de Jean Piaget [12] revisados por Seymour Papert [13] [14]. Este enfoque indica que en todo proceso de aprendizaje es el papel activo de quien aprende el que amplía su conocimiento a través de la manipulación y construcción de objetos. Esta filosofía sugiere que la tradicional construcción de kits es muy adecuada como herramienta de aprendizaje. Sin embargo, dar vida a un objeto por medio de la interacción con un ordenador personal hace posible desarrollar aplicaciones que van más allá de la idea original de los primeros que propusieron esta metodología [15].

Figura 1. Kits robóticos de carácter educativo. A la izquierda se muestra un kit de construcción Lego Mindstorms[1] y a la derecha un humanoide Robonova[2] 1

En tercer lugar, los robots poseen un intrínseco atractivo y una gran capacidad para captar la atención y el interés de los estudiantes [6] [16]. Estas cualidades han sido explotadas durante los últimos años en universidades de todo el mundo en un intento de compensar la decreciente tendencia global en el interés de los estudiantes por las carreras relacionadas con la ciencia, la tecnología, la ingeniería y las matemáticas [17] [18]. Esta tendencia puede observarse también en la menguante cantidad de alumnos que cursan carreras relacionadas con la informática [19], a pesar de los esfuerzos de la comunidad educativa por hacer estos estudios más atractivos. En este sentido son

[1] http://www.lego.com/en-us/mindstorms/build-a-robot/spik3r
[2] http://sysmagazine.com/posts/123533/

comunes las propuestas que respaldan el uso de robots en asignaturas de introducción a la programación [20] o la introducción de prácticas con robots autónomos móviles en asignaturas de robótica [6]. Un claro ejemplo de esta creciente falta de interés en los estudiantes por los contenidos STEM se muestra en la Figura 2, donde se observa perfectamente que desde el año 2005 ha descendido preocupantemente el número total de alumnos matriculados en Ingeniería Informática en la Universidad Jaume I de Castellón.

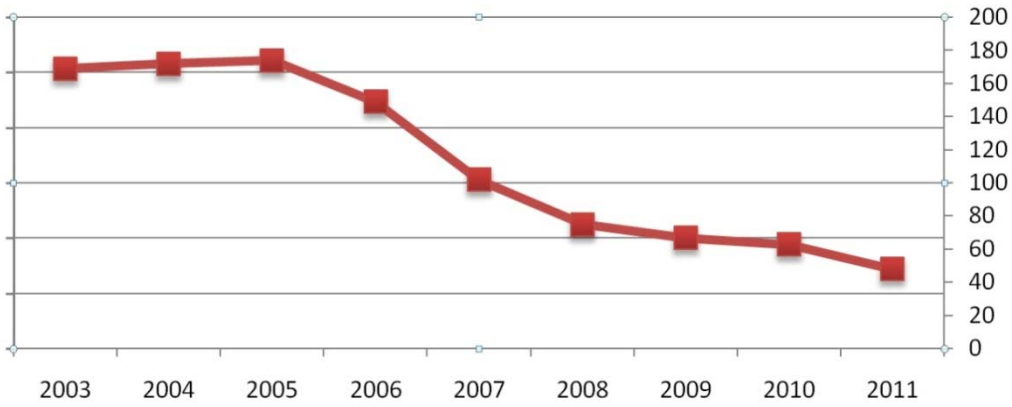

Figura 2. Número de estudiantes matriculados en el Grado en Ingeniería Informática en la Universidad Jaume I de Castellón entre los años 2003 y 2011

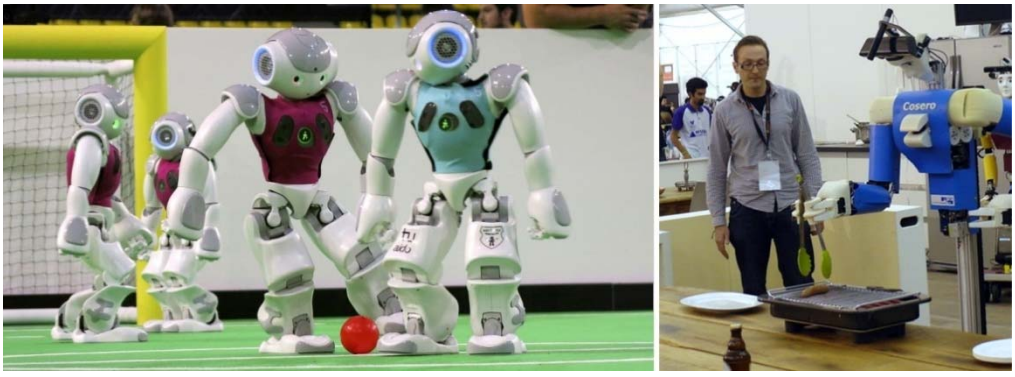

Figura 3. Competiciones RoboCup. A la izquierda se muestra un partido de fútbol entre humanoides NAO correspondiente a la RoboCup Soccer Standard Platform League[3] y a la derecha se observa una prueba de habilidad doméstica en el marco de la competición RoboCup @Home[4]

[3] http://www.timesofisrael.com/israel-iran-compete-in-robot-soccer-tourney/
[4] http://www.ais.uni-bonn.de/robocup.de/bilder_videos.html

Intentando invertir esta evolución en las preferencias de los estudiantes han surgido iniciativas muy interesantes, algunas de ellas en forma de competiciones robóticas como las mundialmente conocidas RoboCup (ver Figura 3) [21]. A nivel nacional resultan de especial interés las competiciones de robots luchadores de sumo como CEABOT [22], donde se enfrentan los estudiantes de diferentes universidades españolas (ver Figura 4). El impacto mediático de tales iniciativas ha incrementado el número de estudiantes atraídos por la robótica, pero no ha conseguido contrarrestar el menguante interés de los estudiantes por las carreras con contenidos STEM.

Figura 4. Competiciones CEABOT de robots luchadores de sumo. A la izquierda CEABOT 2007[5] entre los equipos UJI y UPV. A la derecha CEABOT 2008[6] con los equipos UJI y UHU

No obstante, hace 30 años la presencia de los robots en el sistema educativo era bien diferente. En la década de los 80 los simuladores eran la principal herramienta de trabajo en los laboratorios de robótica de la mayoría de las universidades debido a los elevados precios de los robots [23]. Desde entonces los simuladores robóticos y los robots reales han coexistido como probadas herramientas de trabajo para robótica educativa e investigación.

1.2. Videojuegos y simuladores en educación

El uso de mundos virtuales y juegos para aplicaciones serias ha emergido como una fuerza dominante en la formación, la educación y la simulación, incrementándose su popularidad en las dos últimas décadas [24]. Esto ha suscitado un mayor interés en los juegos y la simulación que ha impulsado el auge de organizaciones (por ejemplo,

[5] http://www.youtube.com/watch?v=biiI4g1Nmuk
[6] http://www.youtube.com/watch?v=Ljm1n8Edyns

North American Simulation and Gaming Association, International Simulation and Gaming Association), revistas de carácter científico (por ejemplo, Journal of Educational Multimedia and Hypermedia Special Issue on Learning and Teaching with Electronic Games) y otras iniciativas en torno al uso de juegos para aplicaciones serias (por ejemplo, Serious Games). Este auge se debe, fundamentalmente, a la capacidad de los mundos virtuales para captar la atención de las más jóvenes y tecnológicas generaciones hacia los altamente atrayentes entornos de juego [25] y a los positivos resultados de numerosas investigaciones acerca del uso de videojuegos en el ámbito académico, las cuales concluyen que éstos mejoran el aprendizaje, aumentando la motivación y el rendimiento de los estudiantes [26] [27] [28].

A pesar de este creciente interés, en muchas ocasiones parece existir confusión en la literatura acerca de qué constituye un juego y qué constituye una simulación [29]. ¿Son lo mismo o son diferentes? ¿Puede una simulación ser un juego y viceversa? Esto no quiere decir que las simulaciones no contengan elementos de los juegos ni que los juegos no incluyan elementos de simuladores, de hecho sí lo hacen [30] tal como se ilustra en la Figura 5. Los simuladores, al igual que los juegos, son interactivos y tienen el propósito de conseguir objetivos concretos dentro de un contexto determinado; sin embargo, las simulaciones realizan un intento serio de representar con precisión un fenómeno real [31] y, en general, incorporan modelos de procesos complejos determinados por algoritmos específicos [32].

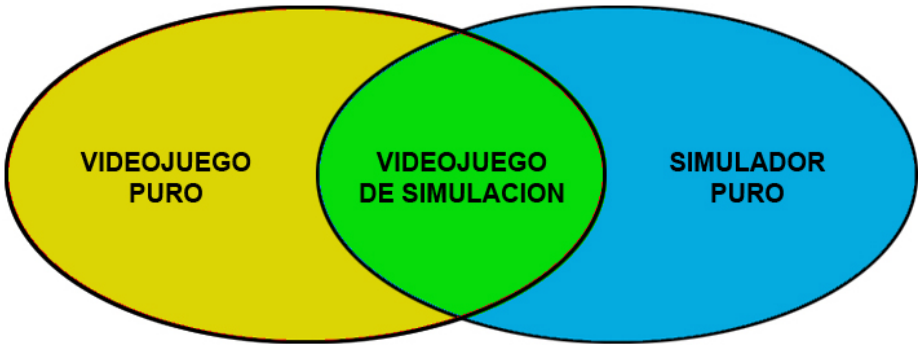

Figura 5. Relación entre videojuegos y simuladores. Los videojuegos y los simuladores comparten elementos comunes que constituyen la base de los videojuegos de simulación

Cualquier juego diseñado para propósitos educativos debería estar estrechamente ligado a objetivos de aprendizaje [33], de modo que la efectividad educativa de tal juego se podría entender como el grado de coincidencia entre los objetivos de

aprendizaje y los atributos del juego. En este sentido, existen numerosos estudios acerca de cuáles son las características principales de los juegos y qué papel tienen desde un punto de vista educativo. En 2001, Garris y Ahlers [34] citaron 39 atributos fundamentales característicos de los juegos, 12 de los cuales resultaron ser estadísticamente significativos e importantes para incrementar de la "sensación de juego" de las simulaciones. En un estudio complementario, Garris y sus colegas [35] analizan un subconjunto de estos atributos, los cuales fueron considerados como los elementos que convierten un videojuego en didáctico y atractivo. Estos atributos son fundamentalmente: fantasía, fidelidad, estimulación sensorial, desafío, misterio, evaluación y control [29].

- La fantasía es un atributo que capta la atención del alumno. Representa algo que se aparta de la vida real y evoca imágenes mentales de cosas que no existen [35] [36].
- La fidelidad es el grado de similitud entre el juego y el entorno que representa [31]. Según algunos estudios el grado de aprendizaje aumenta cuando se incrementa el nivel de fidelidad de un juego [37].
- La estimulación sensorial representa estímulos visuales, auditivos, táctiles e incluso olfativos [38] con el fin de distorsionar la percepción del usuario y facilitar la aceptación temporal una realidad alternativa.
- El desafío está directamente ligado a la dificultad que supone conseguir los objetivos de un juego o una simulación, de manera que la cantidad óptima de desafío es aquélla que genera y mantiene el interés y la motivación del usuario [39].
- Existe misterio cuando hay un hueco entre la información conocida y la desconocida. El misterio despierta la curiosidad del usuario por completar su información, incrementando su motivación [29].
- La evaluación es la medición del grado en que se alcanzan los objetivos dentro de un juego y es un atributo fundamental para impulsar la mejora y el progreso del usuario hacia tales objetivos. Esta medición puede lograrse tabulando comparaciones de los resultados obtenidos por diferentes usuarios o proporcionando realimentación durante el avance del juego (puntuación numérica, tiempo invertido, etc.). Esta realimentación informa inmediatamente al usuario acerca de su progreso y de los efectos positivos o negativos de sus decisiones [27].

- En el contexto de un videojuego educativo el control se refiere a la capacidad de los usuarios de influir sobre su entorno de aprendizaje [40]. Cuando a los estudiantes se les ofrece este control está demostrado que invierten más tiempo en su entorno de aprendizaje y son capaces de elaborar estrategias más complejas que cuando no tienen tal control [41].

En los simuladores el atributo de fidelidad es muy variable: los simuladores de baja fidelidad simplifican los sistemas para resaltar sus elementos clave, mientras que los de alta fidelidad [42] tratan de modelar los sistemas de la manera más realista posible resultando éstos más parecidos a los videojuegos [29]. Si tenemos en cuenta que algunas investigaciones demuestran mejoras en el aprendizaje cuando se introducen elementos de videojuegos en un simulador [43] puede deducirse que: la convergencia de los simuladores de alta fidelidad con juegos serios representa un área de gran potencial para cumplir con los requisitos cognitivos de aprendizaje con un alto grado de eficacia, aprovechando las ventajas de los contenidos basados en juegos para motivar y cautivar a los usuarios [25].

1.3. Robots simulados vs robots reales

La experiencia de años demuestra que tanto los robots como los simuladores son buenas herramientas con las que apoyar enseñanzas de sobre robótica, pero al igual que con cualquier recurso, su uso no ha de ser arbitrario sino que debe elegirse bien en qué momento utilizar cada una de ellas. Así, aunque los experimentos con robots reales resultan indispensables en muchas ocasiones, cabe destacar la funcionalidad y practicidad que las simulaciones pueden aportar en multitud de ocasiones, pues el uso de simuladores suele venir acompañado de una reducción en el coste de la actividad formativa [44].

Con el auge de los kits robóticos a precios asequibles se ha multiplicado el número de robots en los laboratorios de robótica para ser usados en investigación o durante las sesiones prácticas con alumnos y, con ello, el esfuerzo necesario para mantener una flota de robots en condiciones operativas. Esto supone un sobrecoste adicional en horas de preparación de material difícilmente asumible por el personal docente e investigador [1]. Los robots protagonistas de los ensayos requieren ser configurados una y otra vez para adaptarse a los avances consecuencia de los mismos. Son usuales las continuas operaciones de montaje, reparación o sustitución de sensores y

actuadores de todo tipo que primero hay que adquirir si no se dispone de ellos, con el consiguiente coste en tiempo y dinero. Además, los espacios necesarios para tales experimentos no siempre están disponibles y, cuando lo están, no tardan en requerir multitud de retoques para ajustarse a las cambiantes condiciones de la investigación en curso [42].

Con la llegada del vídeo a Internet y las redes sociales se abren muchas oportunidades para difundir las experiencias sobre la enseñanza en robótica y las competiciones, las cuales están siendo explotadas a través de cursos online con buenos resultados [45]. Esta formación a distancia se apoya en conocidas herramientas como los laboratorios remotos [46] y los robots online, las cuales están alcanzando una nueva edad de oro gracias a middlewares multiplataforma [47] y la adopción de nuevos y potentes estándares World Wide Web [48]. Algunos autores piensan que tales plataformas incrementarán sustancialmente la productividad de la comunidad investigadora en robótica, convirtiéndose en recursos didácticos de un valor incalculable, ya que permiten el acceso a sofisticados robots por el simple coste de una conexión a Internet [49]. A este respecto, la utilización de simuladores en buena parte de los ensayos agiliza el trabajo, dadas las dificultades que ofrecen los robots reales para llevarlos al punto de partida cuando llegan a una situación irreversible a centenares o miles de kilómetros de distancia.

Al igual que los robots reales, los simuladores ofrecen la posibilidad de organizar competiciones entre estudiantes para estimular su interés, la participación y el trabajo en equipo. Esto ha sido explotado con éxito durante años por organismos como el National Institute of Standards and Technology (NIST) promoviendo gran variedad de competiciones de robots virtuales a nivel mundial como RoboCup Rescue[7] [50] o Virtual Manufacturing and Automation Competition[8] [51].

Como se ha comentado, una buena solución para los problemas relacionados con el uso de robots reales consiste en trasladar los ensayos a un mundo virtual donde siempre esté todo disponible con un coste en tiempo y dinero reducido. Con tal propósito se han desarrollado numerosos simuladores para robots tanto en modo 2D (Stage [52]) como 3D (Gazebo [53], Simbad [54], Webots [55]) pero la mayoría de ellos no consigue captar durante mucho tiempo el interés del usuario debido,

[7] http://robotarenas.nist.gov/competitions.htm
[8] http://www.icra2013.org/?page_id=231

principalmente, a la falta de realismo [42] y otros motivos de diversa índole que serán tratados en capítulos posteriores de este trabajo. La calidad gráfica del entorno virtual resulta muchas veces demasiado baja para que el ensayo pueda ser seguido con facilidad y resulte convincente para el investigador. Como ejemplos, en la Figura 6 pueden observarse dos capturas de pantalla correspondientes los simuladores Stage y Gazebo, ambos parte del proyecto Player/Stage [56]. La imagen 2D de la izquierda corresponde al simulador Stage y en ella se distinguen tres robots moviéndose por el aulario de la Escuela Superior de Tecnología y Ciencias Experimentales de la Universidad Jaume I de Castellón. La imagen 3D de la derecha corresponde al simulador Gazebo y muestra un modelo del robot Pioneer2 AT de ActivMEDIA Robotics [57] que intenta reconocer algunos objetos y formas geométricas valiéndose de una videocámara y de un sensor láser. En el caso del simulador 2D Stage resulta evidente la escasez de realimentación que este entorno de mapa de bits devuelve al usuario, mientras que en el caso del simulador 3D Gazebo (una de las primera versiones con baja fidelidad gráfica) se observa una imagen muy poco realista lejos de poder sustituir ensayos de algoritmos de visión con robots reales.

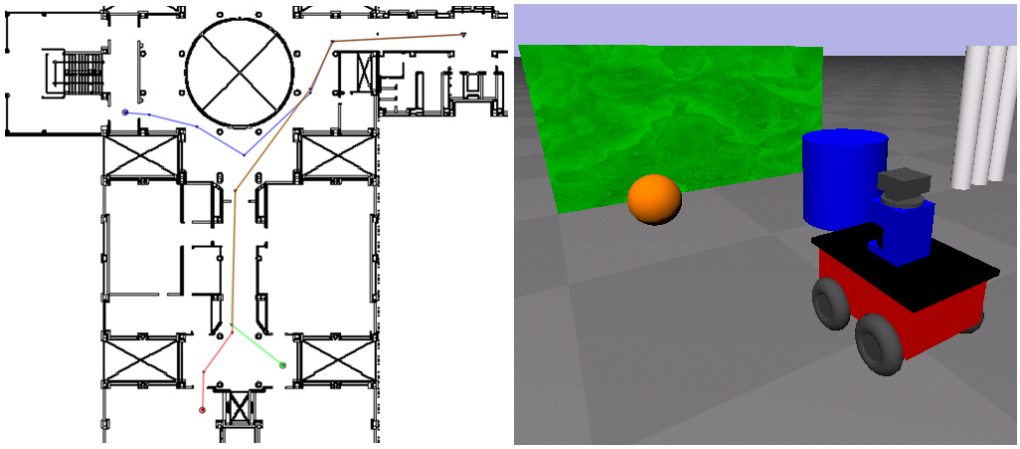

Figura 6. Capturas de pantalla de los simuladores Stage (imagen 2D de la izquierda) y Gazebo[9] (imagen 3D de la derecha) ambos parte del proyecto Player/Stage

Esta general falta de idoneidad en los simuladores ha llevado a muchos investigadores a desarrollar su propio simulador para hacer frente a las necesidades de sus ensayos [25]. No obstante, algunos autores apuntan que existen suficientes simuladores y motores gráficos y de física para que el investigador centre sus esfuerzos en elegir un

[9] http://playerstage.sourceforge.net/gazebo/gazebo.html

simulador en lugar de crear el suyo propio [58]. Así pues, el problema se reduce la mayoría de las veces a encontrar el simulador que mejor se adapte a las necesidades del personal docente e investigador.

En conclusión, el objetivo de esta investigación es llevar a cabo un estudio de las capacidades de los simuladores actuales que termine en la obtención de un entorno de simulación de robots hiperrealista que contenga elementos clave de videojuegos para facilitar la competición entre estudiantes, estimulando su interés por el mundo de la ciencia y la tecnología. Para maximizar su utilidad y rango de aplicación, el simulador obtenido ha de permitir su integración en una plataforma online dedicada a la enseñanza de robótica a distancia.

1.4. Esquema de la tesis

El resto de esta tesis está organizado del siguiente modo:

- El **capítulo 2** expone el estado del arte en simuladores y middlewares, describiendo sus características fundamentales.
- En el **capítulo 3** se explica la metodología empleada para la obtención de los objetivos de esta tesis, así como las herramientas en las que me he apoyado.
- Mis experiencias e impresiones desarrollando entornos y modelos de robots para el simulador USARSim v3.3 se recogen en el **capítulo 4**.
- El **capítulo 5** recoge el proceso de selección de los simuladores que se perfilan más adecuados para alcanzar los objetivos de esta tesis.
- En el **capítulo 6** se estudian las posibilidades que ofrece USARSim v1.2 para UDK. Se pone a prueba este simulador generando entornos y robots orientados a la enseñanza de conocimientos relacionados con la inteligencia artificial.
- El **capítulo 7** está dedicado a la simulación de competiciones robóticas. Aquí se detallan varios entornos y modelos de robots que he creado para facilitar las competiciones entre estudiantes. Las exigentes características propias de los entornos de competición resultan un campo de pruebas perfecto en el que se ponen de manifiesto las diferentes capacidades de los simuladores bajo estudio.
- En el **capítulo 8** se estudia el grado de adecuación de los simuladores en diferentes contextos de formación a distancia.

Capítulo 2

ESTADO DEL ARTE

Para conocer el estado actual del tema en estudio es necesario adquirir una visión general de los simuladores existentes para robótica educativa. También es interesante tener en cuenta que los videojuegos modernos se desarrollan sobre potentes motores de renderizado y de física que proporcionan gráficos y simulación de alta calidad. Tales motores pueden ser ejecutados por GPUs (unidades de procesamiento de gráficos) en multitud de plataformas como ordenadores personales, consolas o incluso dispositivos móviles. Tomando esto en consideración y dada la existente motivación para crear juegos serios y simuladores inmersivos de alta fidelidad, una opción de desarrollo obvia es utilizar motores gráficos y de física de videojuegos modernos [25]. Por este motivo, dedico una parte este capítulo al estudio de los motores de videojuegos disponibles para aplicaciones serias.

En el pasado los simuladores de robótica generalmente se desarrollaban de manera específica para la propia línea de robots de una empresa o para un propósito concreto. En los últimos años, sin embargo, la mejora de los motores gráficos y de física así como la multiplicación de la capacidad de procesamiento y renderizado de los equipos informáticos han impulsado una nueva generación de simuladores que combinan precisos cálculos de física con detalladas representaciones gráficas en tiempo real [59]. Estos simuladores tienen la suficiente flexibilidad potencial para simular cualquier tipo de plataforma robótica, incluyendo los robots autónomos móviles sobre los que se apoya la implementación de gran cantidad de algoritmos característicos de la informática y la inteligencia artificial [9] [10] [11].

Como se verá en apartados posteriores, los simuladores de robótica ya no constituyen una clase en sí mismos, ya que para asegurar la portabilidad del código del simulador hasta las plataformas robóticas del mundo real (como suele ser el objetivo final) es frecuente que tales plataformas ejecuten middlewares específicos. Por middleware se entiende aquel software que asiste a una aplicación para interactuar o comunicarse

con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos, facilitando la terea del programador. El rendimiento, el comportamiento y el funcionamiento interno de estos middlewares deben tenerse en cuenta cuando se comparan simuladores, pues se trata de factores que afectan a la fidelidad de la simulación [59].

2.1. Simuladores

En la actualidad existen muchos simuladores para robots autónomos móviles. La mayoría de ellos suelen combinar la simulación del movimiento de un robot (o conjunto de robots) con la información aportada por sus sensores y pueden clasificarse según muchos criterios: tipos de robots que pueden ser simulados, tipo de simulación (cinemática o dinámica, 2D o 3D), número de robots que pueden ser simulados simultáneamente, plataformas disponibles (Windows, Linux, iOS,...), tipo de licencia, etc.

Dado el objetivo de la presente investigación he centrado mi atención en los simuladores generales, los cuales se caracterizan por incorporar diversidad de plataformas robóticas (ATRV, vehículo genérico de 4 ruedas, robots aéreos,...), sensores (cámaras, escáner láser, GPS, odometría,...) y actuadores (controladores de velocidad, controladores de articulaciones, brazos, pinzas,...). Además, generalmente este tipo de simuladores también permite la introducción de robots, sensores y actuadores personalizados para facilitar su adaptación a las necesidades de cada usuario. Por lo contrario, quedan fuera de este estudio los simuladores que no ofrecen un alto nivel de fidelidad física y gráfica o no permiten desarrollos de simulaciones personalizadas por el usuario. A continuación se ofrece una visión global de los simuladores generales más utilizados por la comunidad científica.

2.1.1. Simuladores open source

Las aplicaciones comerciales que no liberan su código fuente ofrecen serias dificultades al usuario investigador cuando surgen problemas de falta de adaptación del software a la investigación en curso. Este hecho unido a la mencionada falta de idoneidad de los simuladores comerciales para ser utilizados en el ámbito de la educación y la investigación, constituye uno de los principales motivos por los que en los últimos 15 años han emergido decenas de simuladores y middlewares open source

destinados a la educación y la investigación en general, tal como apunta Jeff Craighead en su página de SARGE[10]. Se trata, en su mayoría, de iniciativas de desarrollo encabezadas fundamentalmente por personal docente e investigador de universidades de todo el mundo. A continuación se describen las aportaciones más relevantes.

Blender[11] [60] es un avanzado paquete de modelado y animación 3D que ofrece renderizado de alta calidad (ver Figura 7) y una gran variedad de tecnologías útiles para simulación, tales como cálculo de cinemática inversa, simulación de dinámica basada en Bullet [61], scripting (basado en Python) y un potente motor de juego. Desde hace algún tiempo se está intentando aprovechar la tecnología de Blender para crear simuladores de robots móviles (Blender for Robotics[12]) como OpenRobots, el cual ha evolucionado recientemente a MORSE (Modular Open Robots Simulator Engine) [62] [63]. MORSE admite middlewares open source comunes como ROS, YARP, Pocolibs y MOOS y, además, ofrece un canal de comunicación genérico basado en socket. MORSE está en continuo desarrollo en el laboratorio francés LAAS del CNRS francés, muestra de ello es que su última versión (MORSE 1.3) ha sido puesta a disposición de la comunidad científica en mayo de 2015. Por contra, MORSE solo soporta Linux como sistema operativo.

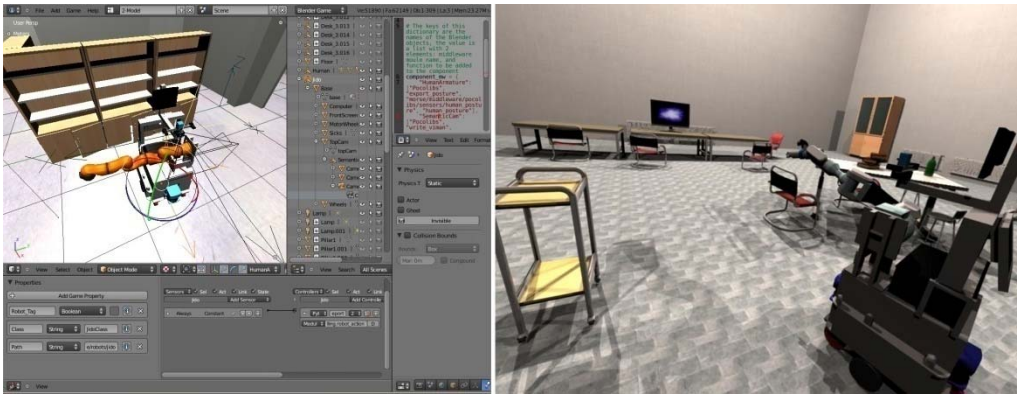

Figura 7. Capturas de pantalla del simulador MORSE[13]. A la izquierda se muestra el interfaz gráfico de usuario (Blender) y a la derecha se observa con detalle un entorno de simulación

[10] http://sarge.sourceforge.net/page11/page11.html
[11] http://www.blender.org/about
[12] http://wiki.blender.org/index.php/Community:Science/Robotics
[13] http://www.openrobots.org/morse/doc/latest/what_is_morse.html

El proyecto Player[14] crea software libre que para la investigación en robótica ofreciendo probablemente el servidor para el control de dispositivos robot más utilizado del mundo. Player ha sido desarrollado por un equipo internacional de investigadores en robótica conjuntamente con los simuladores Stage y Gazebo [52]. Stage es un simulador de robots bidimensional diseñado principalmente para espacios interiores, proporcionando una simulación simplificada para grandes grupos o enjambres de robots que se mueven por un mundo descrito como bitmap (imagen de mapa de bits) [56] [64]. Los robots están modelados como una base móvil con sensores usando un lenguaje descriptivo. Los dispositivos se comunican en Stage del mismo modo que lo hace el hardware real, permitiendo utilizar exactamente el mismo código tanto en simulación como en ensayos con robots reales [65]. Este simulador se puede utilizar como una aplicación independiente, biblioteca de C ++, o un plug-in para Player. Esto no garantiza, sin embargo, que las simulaciones tengan una alta fidelidad de simulación física. La última versión de este simulador es Stage 4.1.1 de enero de 2012. Aunque Player puede considerarse un middleware y Stage un simulador que quedaría fuera de los intereses de esta tesis por ser 2D y de baja fidelidad, he considerado oportuno incluir su descripción en este apartado por su estrecha relación con Gazebo.

Figura 8. Captura de pantalla de un entorno para competiciones de lucha de sumo con Gazebo 2.2

Gazebo [53] es un simulador 3D diseñado para grupos reducidos de robots que ofrece cálculos de física de alta calidad basados en ODE (Open Dynamics Engine) [66], aunque

[14] http://playerstage.sourceforge.net/

también permite utilizar los motores de física Bullet, Simbody y DART. Gazebo presenta una interfaz estándar de Player, además de su propia interfaz nativa. Los controladores desarrollados para el simulador Stage generalmente se pueden utilizar con Gazebo sin modificación (y viceversa). Gazebo permite el uso de plug-ins que expanden de forma notoria sus capacidades con nuevos sensores o robots personalizados. Gazebo[15] 6.0 ofrece gráficos de calidad media-alta para sistemas operativos Ubuntu, manteniendo una alta compatibilidad con ROS. En la Figura 8 puede observarse una captura de pantalla de este simulador en la que se muestran dos robots Pioneer P3AT que modelé conjuntamente con el ring de sumo virtual para competiciones entre estudiantes.

SARGE [67] [68] (Search And Rescue Game Environment) es un simulador desarrollado por Jeffrey Craighead en el departamento de ciencia de los computadores de la universidad de Florida para entrenar a las fuerzas del orden en el uso de la robótica en operaciones de búsqueda y rescate. El Propósito de J. Craighead era crear un simulador que permitiera evitar los mencionados problemas de idoneidad de los simuladores robóticos del momento (2007) en cuanto a complejidad de uso, dificultad de desarrollo y baja fidelidad del motor de física. SARGE utiliza el motor de juegos Unity, permitiendo obtener simulaciones de alta fidelidad gráfica (ver Figura 9) y física basadas estas últimas en PhysX[16] [61] de Nvidia, actualmente considerado uno de los motores de física más potentes. La programación se puede realizar con JavaScript clone, C, C++ y Boo. La última versión SARGE 0.3 de 2010 puede ser ejecutada en Windows y Mac OS.

Figura 9. Capturas de pantalla del simulador SARGE. La imagen de la izquierda muestra la plataforma robótica iRobot Packbot Scout sobre un terreno de hierba. En la imagen de la derecha se observa el robot SeaRAI en un entorno acuático [67].

[15] http://gazebosim.org/

[16] http://developer.nvidia.com/gameworks-physx-overview

OpenRAVE[17] (Open Robotics and Automation Virtual Environment) proporciona un entorno para las pruebas, el desarrollo y la implementación de algoritmos de planificación de movimiento en aplicaciones de robótica del mundo real [69] [70]. Su principal aplicación se centra en la planificación de agarre de objetos con pinzas o manos de manipuladores de robots humanoides como muestra la Figura 10. OpenRAVE también ofrece capacidad de simulación y planificación a otras estructuras software como Player o ROS. Una de las ventajas de OpenRAVE es que con su sistema de plug-ins puede conectarse a prácticamente todo, ya sea un controlador, un planificador, un motor de simulación o un robot real. OpenRAVE fue creado en el Instituto de Robótica de la Universidad Carnagie Mellon por Rosen Diankow. La última versión de OpenRAVE es la 0.9.0 de marzo de 2013 y soporta diversos lenguajes de programación como C++ o Python.

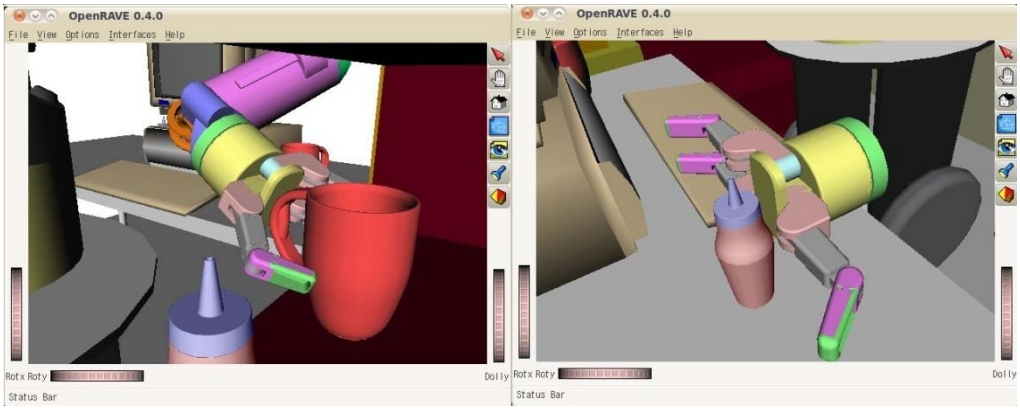

Figura 10. Ejemplos de ensayos de agarre utilizando el módulo simplegrasping de OpenRAVE [69]

SimRobot[18] [71] es un simulador 3D de alta fidelidad (ver Figura 11) con gráficos basados en OpenGL[19] y cálculos de física basados en ODE [61], el cual ha sido desarrollado en el Centro Alemán de Investigación en Inteligencia Artificial de la Universidad de Bremen. Actualmente su uso principal se encuentra en la competición RoboCup simulada, aunque sus posibilidades de uso van mucho más allá. A diferencia de otros simuladores de robótica, SimRobot no está diseñado entono a una interacción servidor/cliente, permitiendo pausar las simulaciones o ejecutarlas paso a paso con el consiguiente valor añadido a la hora de depurar código [72]. Como la mayor parte de simuladores generales, SimRobot permite la introducción de robots,

[17] http://openrave.org/docs/latest_stable/
[18] http://www.informatik.uni-bremen.de/simrobot/index_e.htm
[19] http://en.wikipedia.org/wiki/OpenGL

sensores y actuadores personalizados. La versión más reciente de SimRobot permite su instalación en Linux, Windows y Mac OS.

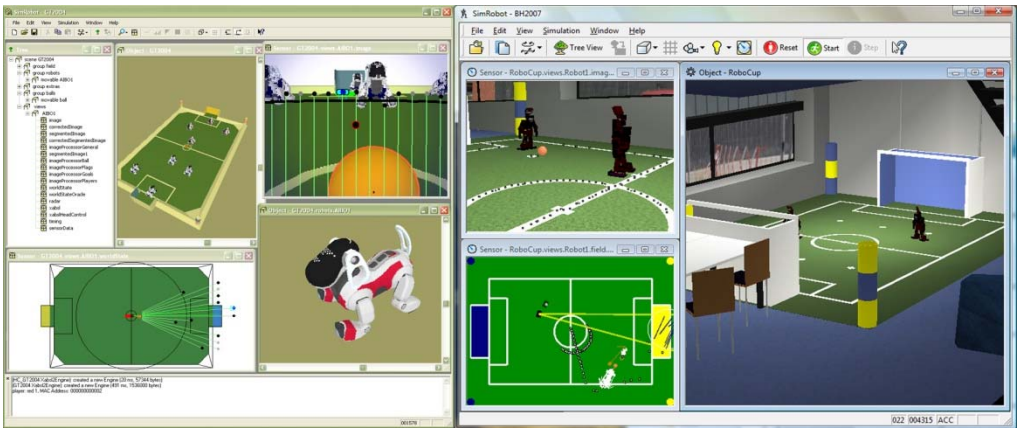

Figura 11. Simulación de competiciones de fútbol con SimRobot. A la izquierda, interfaz de usuario de SimRobot simulando el German Team 2004 de robots Aibo [72]. A la derecha, simulación del equipo de humanoides B-Human [71]

USARSim (Unified System for Automation and Robot Simulation) [73] [74] [75] es un simulador 3D de alta fidelidad para robots que utiliza el potente motor gráfico y de simulación Unreal Engine[20] 3 del exitoso videojuego Unreal Tournament[21] 3 de Epic Games. Originalmente concebido en 2002 en la Universidad Carnegie Mellon, USARSim fue desarrollado a partir de 2005 por el National Institute of Standards & Technology (NIST, USA) por Stephen Balakirsky principalmente. UDK (Unreal Development Kit) es un paquete de herramientas open source que ofrece la potencia de Unreal Engine 3 (que utiliza NVIDIA PhysX para los cálculos de física) de forma gratuita, incorporando un editor de niveles (UDK Editor) en tiempo real. UDK Editor permite al usuario importar modelos desarrollados con herramientas CAD populares como Maya [76] [77] y 3D Studio Max [78] [79]. Tales modelos pueden utilizarse posteriormente como elementos del entorno o para la creación de nuevos robots y sensores. Unreal Engine 3 ofrece un lenguaje de scripts (UnrealScript) orientado a objetos que permite personalizar la simulación, definiendo nuevos robots y sensores o modificando el comportamiento de los ya existentes entre muchas otras opciones. Unreal Engine 3 utiliza una arquitectura cliente-servidor para soportar múltiples jugadores y, por tanto, varios robots controlados desde diferentes ordenadores.

[20] http://en.wikipedia.org/wiki/Unreal_Engine#Unreal_Engine_3
[21] http://en.wikipedia.org/wiki/Unreal_Tournament_3

Unreal Engine 3 está diseñado para DirectX (versiones 9-11 para Windows y Xbox 360), así como sistemas que usan OpenGL, incluyendo PlayStation 3, Mac OS X, iOS, Android y Stage 3D para Adobe Flash Player 11. La última versión de este simulador es USARSim[22] UDK v1.4 de noviembre de 2013. En la Figura 12 se puede observar la riqueza gráfica y funcional de USARSim en un entorno con objetos de comportamiento físico diverso que desarrollé para las sesiones prácticas de un curso de robótica online.

Figura 12. Robot NAO durante una simulación con USARSim en un entorno altamente interactivo

2.1.2. Simuladores comerciales

En la actualidad existen numerosos simuladores comerciales disponibles en el mercado, estando destinados muchos de ellos a cubrir necesidades del sector industrial o de la propia línea de robots de una empresa [59]. Al igual que ocurre con cualquier otro software comercial, una de las desventajas de estos simuladores es que no son open source.

Sin embargo, entre estos simuladores podemos encontrar algunos que despiertan especial interés en el campo de la educación y la investigación en robótica. Se trata, principalmente, de simuladores generales de alta fidelidad que permiten al usuario introducir sus propias plataformas robóticas y programar sus comportamientos. A continuación describo los simuladores comerciales que considero más interesantes en el campo de la educación y la investigación en robótica.

[22] http://sourceforge.net/projects/usarsim/files/usarsim-UDK/

Microsoft Robotics Developer Studio[23] (MRDS) [80] [81] es un simulador comercial en 3D que ofrece gráficos y cálculos de física de alta calidad (ver Figura 13). Para ello utiliza el motor de física PhysX de Nvidia. MRDS facilita el control remoto de robots (reales o simulados) utilizando HTML y JavaScript. Al tratarse de un producto Microsoft, MRDS recomienda el uso de Microsoft Visual Studio 2010 para la programación con C#, la cual se complementa con un lenguaje gráfico (VPL, Visual Programming Language) que simplifica la programación con un intuitivo pincha y arrastra de bloques funcionales. La última versión de este software, MRDS 4.0.261.0 de junio de 2012, solo puede ejecutarse en Windows 7, aunque permite el control de robots con otros sistemas operativos a través de un canal de comunicación tal como Bluetooth o WiFi.

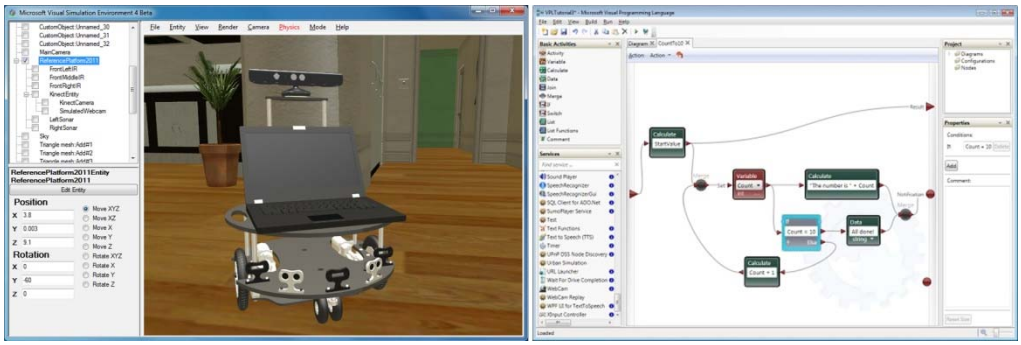

Figura 13. Capturas de pantalla de una simulación con Microsoft Robotics Developer Studio. A la izquierda se observa la simulación en sí, mientras que a la derecha se puede ver una programación con el lenguaje gráfico VPL[24]

Webots [55] es un simulador 3D de alta fidelidad (ver Figura 14) con cálculos de física basados en ODE que proporciona interfaz de programación para numerosos lenguajes como C, C++, Java, Python o MATLAB incorporando un compilador C. Webots está muy extendido en la comunidad educativa, ofreciendo buena conectividad con extendidos middlewares como ROS, software CAD para importación de modelos u otros simuladores como Choregraphe de Aldebarán Robotics. Webots además de multitud de plataformas robóticas y sensores, incorpora un editor de entornos y de robots de fácil uso que le permite adaptarse con facilidad a las necesidades del usuario. Actualmente este simulador se halla en continuo desarrollo

[23] http://msdn.microsoft.com/en-us/library/dd939239.aspx
[24] http://msdn.microsoft.com/en-us/library/bb483024.aspx

y su última versión (Webots[25] 8.3.0 de octubre de 2015) está disponible para entornos Linux, Windows y Mac OS.

Figura 14. Competición robótica simulada con Webots 7.4.1. Dos robots P3AT se enfrentan en un combate de sumo

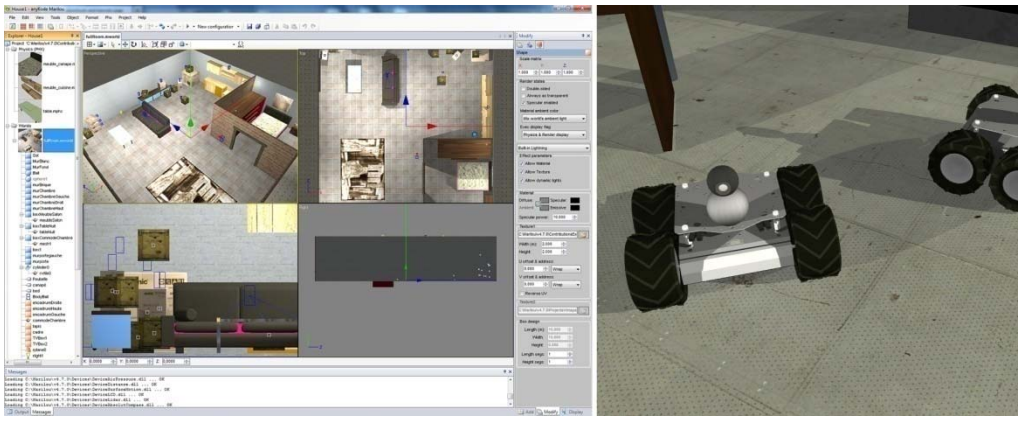

Figura 15. Capturas de pantalla del simulador Marilou de anyKode[26]. La imagen de la izquierda muestra el aspecto del interfaz gráfico de usuario, mientras que la imagen de la derecha muestra en detalle la calidad gráfica del simulador

anyKode Marilou[27] modeling and simulation environment es un conjunto de programas de simulación robótica 3D de alta fidelidad gráfica (ver Figura 15) y física que incluye un entorno para el diseño de robots. La simulación de la física incluye detección de colisiones de geometrías complejas en base a superficies formadas por

[25] http://www.cyberbotics.com/
[26] http://www.anykode.com/downloads.php
[27] http://www.anykode.com/

triángulos, lo que facilita la introducción de modelos 3D creados con programas de CAD. Aunque existen versiones Beta más actuales, la última versión estable oficial es la Marilou 2010, la cual está disponible para sistemas operativos Windows y Linux y admite programación mediante multitud de lenguajes, entre ellos C/C++, J# o VB#.

robotSim[28] Pro es un simulador 3D de alta fidelidad gráfica y física (basada esta última en ODE) pensado para simplificar el diseño y las pruebas de algoritmos de control y comportamiento de robots. robotSim Pro cuenta con un editor de 3D para crear o modificar todo tipo de entornos y objetos que se complementa con robotBuilder que añade la capacidad de crear y modificar modelos de robots (ver Figura 16). robotSim proporciona una interfaz gráfica para una rápida simulación utilizando programas de control de LabVIEW Robotics, permitiendo importar desde otros lenguajes como C/C++ o VHDL. Desde hace unos años ambas aplicaciones de Cogmation Robotics se distribuyen como un módulo de LabVIEW Robotics 2012 bajo el nombre de LabVIEW Robotics Simulator.

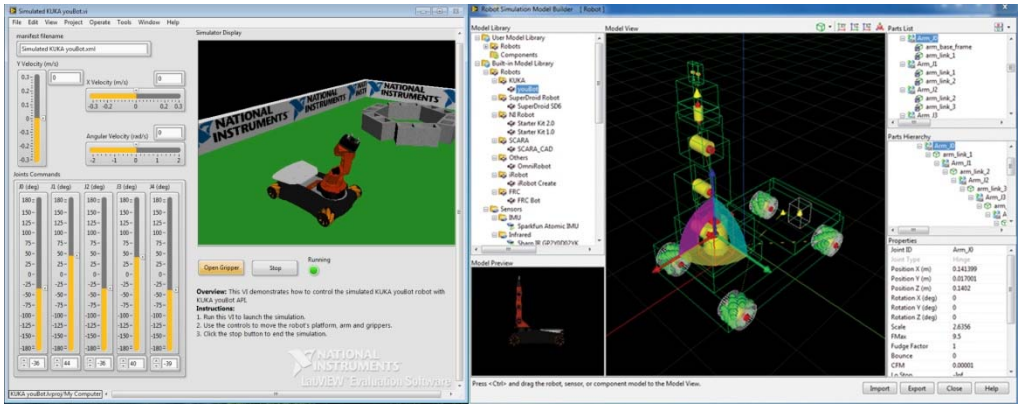

Figura 16. Capturas de pantalla de simRobot Pro. A la izquierda se puede ver una simulación del robot KUKA youBot con LabVIEW Robotics Simulator[29]. La imagen de la derecha corresponde al LabVIEW Robotics Simulator Model Builder

La Figura 17 muestra la cronología y evolución de los simuladores robóticos más interesantes para los objetivos de esta tesis. En los últimos quince años han surgido otros simuladores destacables en el campo de la robótica educativa, pero algunos de ellos ya no se encuentran bajo desarrollo y por este motivo no los he incluido en este estudio.

[28] http://sine.ni.com/nips/cds/view/p/lang/es/nid/209028
[29] http://www.ni.com/white-paper/14673/en/#toc5

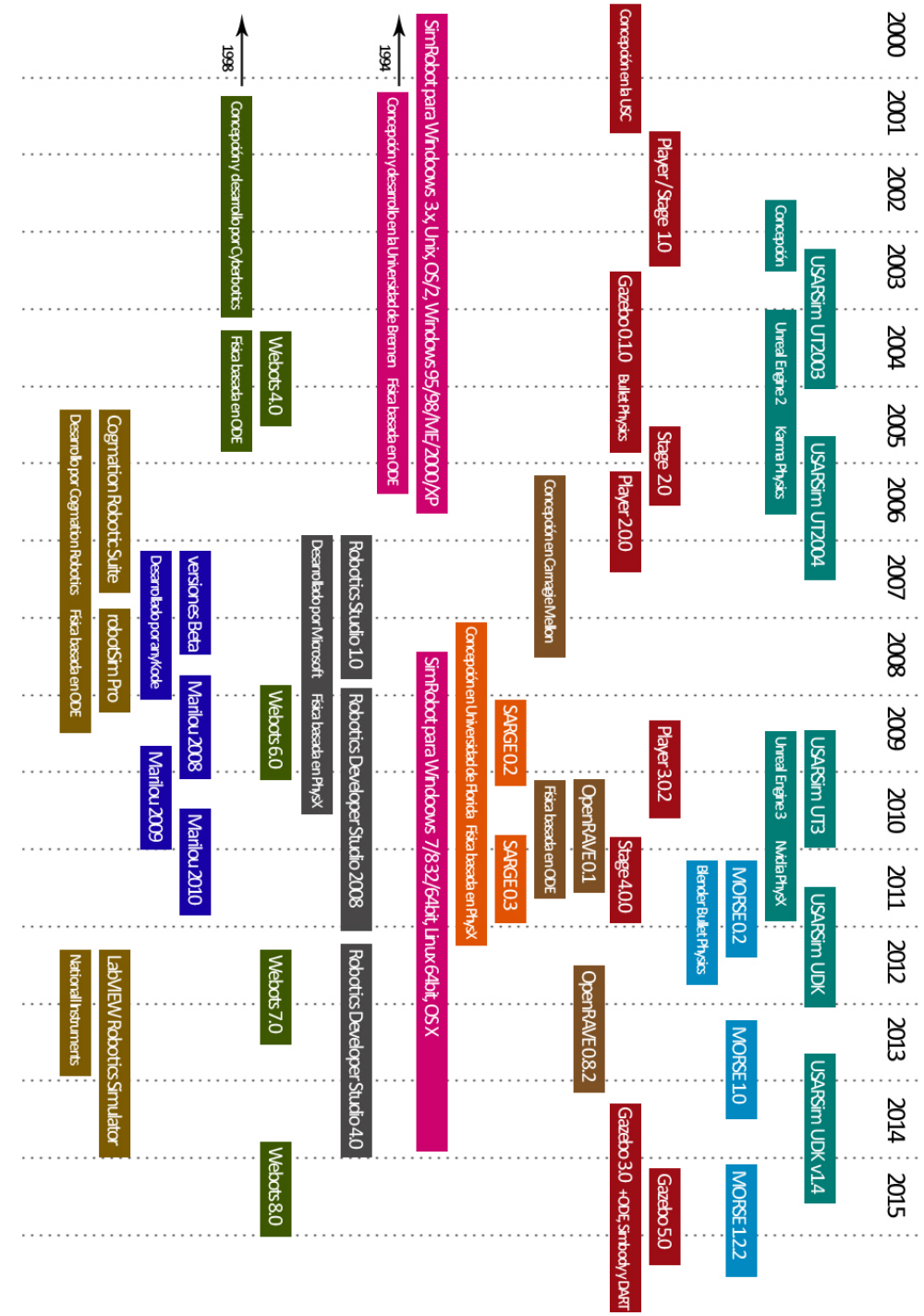

Figura 17. Cronología de los simuladores de robótica más destacables de los últimos quince años

2.2. Middlewares

ROS[30] (Robot Operating System) [82] es actualmente uno de los más populares middlewares para sistemas robóticos. Se trata de un set de librerías y herramientas software de ayuda a la creación de aplicaciones robot. ROS ha experimentado un meteórico auge en los últimos años, probablemente gracias a que permite a sus usuarios tomar parte activa en el desarrollo del sistema. Aunque la empresa Robotics Equipment Corporation[31] lo ha portado a Windows, ROS solo es oficialmente compatible con sistemas basados en UNIX incluyendo Mac OS. Uno de los puntos fuertes de ROS es que resulta fácil de usar con otros simuladores y middlewares, permitiendo incorporar sus librerías. ROS también proporciona funciones para serialización y transporte entre diferentes programas (nodos), los cuales pueden estar ubicados en diferentes computadoras y escritos en distintos lenguajes de programación (C++, Python, MATLAB, Java,…). ROS está diseñado para ser un sistema parcialmente en tiempo real debido a que uno de los principales robots sobre los que se ha desarrollado es el PR2, destinado a situaciones que incluyen interacción hombre-máquina en tiempo real. ROS, proporciona abstracción de hardware, lo que permite que los algoritmos de control desarrollados durante las simulaciones sean directamente aplicables a robots reales sin modificaciones ROS Jade[32] lanzado en mayo de 2015 es la última versión, aunque todavía se recomienda el uso de la anterior versión (ROS Indigo) por ser más estable.

YARP (Yet Another Robot Plattform), es un conjunto de librerías para desacoplar limpiamente los dispositivos de la arquitectura de software, favoreciendo así la evolución incremental de la misma [83]. El ánimo de YARP es proporcionar una base de software robot estable y de larga duración, al tiempo que permita frecuentes cambios de sensores, actuadores, procesadores y redes. YARP abstrae el mecanismo de transporte de los componentes de software, permitiendo su ejecución en cualquier máquina. Permite el uso compartido de memoria para la comunicación local, y TCP / IP, UDP, y de multidifusión para la comunicación en red. YARP interactúa bien con código C / C ++ y se puede utilizar con varios otros lenguajes de programación. La última versión estable de YARP es la 2.3.64 de julio de 2014 y puede ejecutarse en Windows, Linux y Mac OSX.

[30] http://www.ros.org/
[31] http://www.servicerobotics.eu/index.php?id=37
[32] http://wiki.ros.org/Distributions

RT Middleware (Robotics Technology Middleware) [84] es una plataforma software que permite construir sistemas robot mediante la combinación de los módulos de software de los elementos funcionales del robot (elemento funcional RT). RT Middleware es tan solo un estándar software para la plataforma robótica, siendo OpenRTM-aist[33] su principal implementación. OpenRTM-aist está siendo desarrollado y distribuido por el Instituto Nacional de Ciencia y Tecnología Industrial Avanzada de Japón, país donde esta plataforma es muy popular. La última versión de OpenRTM-aist es la 1.1.0 de mayo de 2012, la cual permite su instalación tanto en sistemas Windows como Linux y soporta programación con C++, Python y Java. La Figura 18 muestra de forma esquemática el principio de funcionamiento de este software.

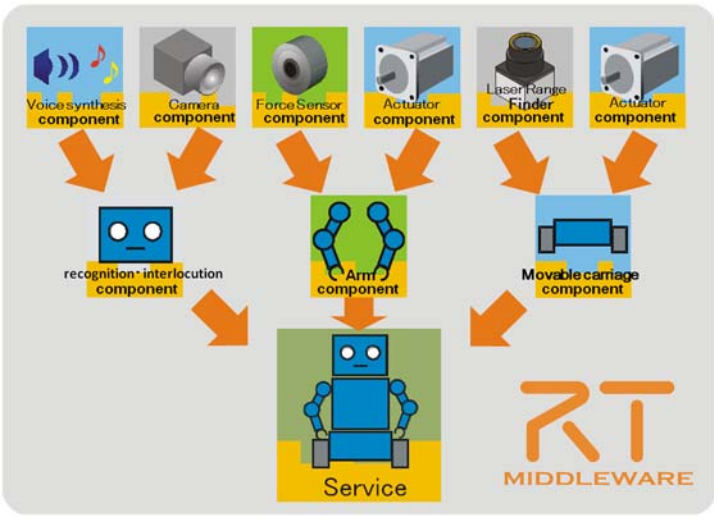

Figura 18. Principio de funcionamiento de RT Middleware[34]. El sistema robot se obtiene conectando múltiples componentes que pueden estar a su vez agrupados según su función en el sistema

MRPT[35] (Mobile Robot Programming Toolkit) [85] es un conjunto de aplicaciones y librerías C++ que proporciona estructuras de datos y algoritmos para tareas comunes de investigación robótica. MRPT está orientado principalmente a facilitar funciones de SLAM (Simultaneous Localization And Mapping), visión por ordenador y algoritmos de planificación de movimientos. Esta herramienta se encuentra en continuo

[33] http://www.openrtm.org/
[34] http://openrtm.org/openrtm/en/content/rt-middleware-overview
[35] http://www.mrpt.org/

desarrollo por el laboratorio MAPIR de la Universidad de Málaga y su actual versión 1.2.2 de septiembre de 2014 permite su instalación en entornos Windows y Linux.

OROCOS es el acrónimo de Open RObot COntrol Software [86]. El proyecto Orocos tiene como objetivo la creación de un marco de software de propósito general, gratuito y modular para el control de robots y máquinas. El proyecto Orocos está compuesto fundamentalmente por 4 bibliotecas C++: Real-Time Toolkit (RTT), Kinematics and Dynamics Library (KDL), Bayesian Filtering Library (BFL) y Orocos Component Library (OCL) [87]. RTT no es una aplicación en sí misma, sino que proporciona infraestructura y funciones para construir aplicaciones robóticas en C++. OCL proporciona componentes de control listos para usar. KDL es una biblioteca C++ que permite calcular cadenas cinemáticas en tiempo real. BFL facilita la inferencia en redes bayesianas dinámicas. En las versiones más actuales, RTT y OCL se han fusionado en Orocos Toolchain[36] v.2.x.

2.3. Motores de videojuegos para el desarrollo de juegos serios

Los videojuegos de carácter didáctico (llamados juegos serios) representan el estado del arte en la convergencia de la tecnología de los juegos electrónicos con los principios de diseño educativos y pedagógicos. Los motores de los videojuegos modernos proporcionan avanzadas tecnologías de iluminación, sombras, materiales y efectos especiales, confiriendo al desarrollador el potencial suficiente para ofrecer al usuario la misma calidad perceptual como si estuviera presente en la escena real [88]. A pesar del valor de los contenidos de alta fidelidad para cautivar a los estudiantes y proporcionar entornos de formación realistas, la creación de videojuegos con elevados niveles de realismo visual y funcional es un proceso complejo, lento y costoso. Por consiguiente, los motores de juegos comerciales, que proporcionan entornos de desarrollo y recursos para crear más rápidamente mundos virtuales de alta fidelidad, se utilizan cada vez más tanto para aplicaciones educativas como de entretenimiento [88].

Un motor de juego [89] es un conjunto de herramientas software con el propósito de simplificar las tareas de desarrollo de un juego. Los motores de juego suelen proporcionar abstracción de plataforma, permitiendo que un mismo juego pueda ejecutarse en diferentes plataformas (videoconsolas, ordenadores, teléfonos móviles

[36] http://www.orocos.org/wiki/orocos/toolchain

y navegadores web principalmente) con pocas o ninguna modificación en su código fuente. Es habitual que los motores de juegos ofrezcan un editor de niveles y herramientas para modificar el comportamiento del juego. Así, por ejemplo, los usuarios pueden simplemente crear nuevos niveles, objetos o personajes añadiéndolos a un juego, o generar juegos completamente nuevos alterando el código del motor.

2.3.1. Criterios de selección de motores de juegos para aplicaciones didácticas

De lo expuesto hasta ahora puede deducirse que el estado del arte para juegos serios es un reflejo de los juegos de ocio. Sin embargo, los requisitos técnicos de los juegos serios son con frecuencia más diversos y amplios que sus homólogos de entretenimiento. Esto provoca con mucha frecuencia que los creadores de juegos serios recurran al desarrollo propio, o a encargos a medida en empresas especializadas (como por ejemplo TOTEMlearning[37]), debido a las dificultades existentes para que los desarrolladores de motores de videojuegos comprendan y den respuesta a las necesidades del diseño pedagógico.

Con el ánimo de dar respuesta a las necesidades técnicas para el desarrollo de juegos serios se han realizado numerosos estudios en la última década [44] [29]. Son de especial interés para este trabajo las investigaciones centradas en hallar las características clave que debe reunir un motor de juego para que resulte adecuado como base para el desarrollo de aplicaciones didácticas y entornos virtuales para investigación.

En general, existe consenso entre los investigadores acerca de cuáles son los criterios a seguir a la hora de comparar motores de juego según su capacidad para desarrollar recursos académicos. Aunque la terminología puede variar según la fuente consultada, estos criterios se pueden definir como: fidelidad audiovisual, fidelidad funcional, componibilidad, accesibilidad, conectividad y heterogeneidad [88]. A continuación describo estos criterios de forma resumida, pues esta información la amplio en el capítulo 5 de esta tesis, donde me centro en la selección de simuladores.

[37] http://www.totemlearning.com/

La fidelidad audiovisual se refiere al grado de similitud gráfica y auditiva entre un entorno virtual y su correspondencia en el mundo real. Un alto grado de fidelidad audiovisual se considera deseable ya sea para mejorar la transferencia de conocimientos desde un juego al mundo real [90] como para atraer la atención de los usuarios a través de una experiencia inmersiva [91].

La fidelidad funcional de un entorno virtual indica el grado en que éste se comporta de manera realista. La fidelidad funcional depende del comportamiento físico, de la interactividad y de la inteligencia que muestren los objetos, mecanismos y personajes presentes en un mundo virtual.

Accesibilidad. A diferencia de los juegos de ocio, los juegos serios van a menudo dirigidos a individuos que no acostumbran a jugar a videojuegos, con poco interés en la tecnología o escaso conocimiento de las interfaces de usuario. Del mismo modo, los desarrolladores de juegos serios son en muchas ocasiones personal docente explorando nuevas formas de transferir conocimientos [25]. Por lo tanto, la capacidad del motor de juego para soportar desarrolladores y usuarios con experiencia limitada es un aspecto importante.

La heterogeneidad se refiere a las posibilidades multiplataforma hardware y software que ofrece una aplicación. Los videojuegos se desarrollan para multitud de plataformas como PC, Microsoft Xbox, Nintendo Wii o Sony PlayStation y para diversos sistemas operativos como Windows, Linux, iOS, Android o Windows Phone, intentando alcanzar la mayor cantidad posible de usuarios potenciales. En el campo de los juegos serios las capacidades multiplataforma resultan de vital importancia debido a que el sector de la población al que van dirigidos está compuesto en gran parte por no jugadores con hardware y sistemas operativos diversos.

Componibilidad. Los juegos serios a menudo tratan de modelar escenarios y situaciones reales o adaptar los datos del mundo real para su uso en juegos con unos gastos mínimos de desarrollo. El término componibilidad en este contexto se utiliza para describir tanto la reutilización de contenido creado dentro de un motor de juego como la capacidad del mismo para importar y utilizar recursos desarrollados con otras aplicaciones.

Conectividad. El soporte para elementos sociales y comunidades de gran escala está adquiriendo cada vez más reconocimiento en los videojuegos como mecanismo para

aumentar la captación de nuevos jugadores y el juego a largo plazo. Es habitual que los motores de juego actuales permitan conexiones cliente-servidor, facilitando la conjunción de gran cantidad de usuarios en un mismo entorno virtual.

2.3.2. Motores de juegos interesantes para robótica

Existen actualmente en el mercado más de 100 de motores[38] de videojuegos, muchos de los cuales podrían utilizarse para juegos serios y en concreto para simuladores de robótica. A continuación expongo brevemente aquellos motores que según algunos estudios [88] [25] resultan especialmente interesantes por satisfacer los criterios identificados en el apartado anterior.

CryENGINE 3 [92] reúne una serie de características que son útiles para la creación de juegos serios inmersivos y realistas, tales como un editor de niveles en tiempo real, texturas de relieve, iluminación dinámica, potente motor de física, shaders[39], sombras, soporte para múltiples núcleos, sistema de animación de personajes, estrategias de búsqueda de rutas (pathfinding [93]), sonidos dinámicos y música interactiva. Este motor está disponible para PC, PS3 y Xbox360. La Figura 19 muestra la fidelidad gráfica que puede obtenerse con este motor de juego, tanto para aplicaciones educativas como de entretenimiento.

Figura 19. Capturas de pantalla de aplicaciones que usan CryENGINE 3. La imagen de la izquierda corresponde al juego: Ryse Son of Rome[40] de Crytek. La imagen de la derecha pertenece a una simulación de construcción de edificios con The Situation Engine [94]

[38] http://en.wikipedia.org/wiki/List_of_game_engines
[39] http://developer.valvesoftware.com/wiki/Shader
[40] http://www.crytek.com/games/ryse/overview

Source Engine[41] de Valve proporciona características interesantes como animación de personajes, un potente editor de niveles, inteligencia artificial avanzada, shaders, estrategias de búsqueda de rutas y navegación, iluminación dinámica, sonidos dinámicos, emisores de partículas y efectos ambientales como lluvia o niebla. Este motor está disponible para PC y consolas como la Xbox360. La Figura 20 muestra dos capturas de pantalla de aplicaciones desarrolladas con el motor Source.

Figura 20. Capturas de pantalla correspondientes al motor Source de Valve. La imagen de la izquierda está obtenida del juego HALF-LIFE 2[42], mientras que la de la derecha pertenece a una simulación para formación quirúrgica [95]

El motor Unreal Engine 3, así como el paquete de software UDK, ya ha sido descrito parcialmente en el apartado destinado a simuladores open source. No obstante, tal descripción está centrada en su relación con USARSim, motivo por el cual considero importante resaltar algunas características de este motor aún no tratadas tales como su iluminación dinámica de alta calidad, sus avanzados sistemas de partículas y editor de materiales y su versátil editor gráfico de inteligencia artificial. Unreal Engine 3 y UDK ofrecen una amplia gama de tecnologías, herramientas de creación de contenidos y la infraestructura de apoyo necesaria para desarrollar juegos de muy alto nivel [88].

La Figura 21 muestra el aspecto de un juego y una simulación obtenidos ambos con el motor de juego Unreal Engine 3. En la simulación (imagen de la derecha) un robot Guardian de Robotnik dotado de un brazo manipulador agarra una maleta sospechosa que ha sido localizada por un robot aéreo de vigilancia.

[41] http://developer.valvesoftware.com/wiki/Source_Engine_Features
[42] http://www.electricblueskies.com/category/pc-gaming-new/half-life-2/

Figura 21. Capturas de pantalla con el motor UT3. La imagen de la izquierda pertenece al juego Unreal Tournament 3. La imagen de la derecha corresponde a una simulación de búsqueda, localización y procesamiento de objetos sospechosos con USARSim

Unity 3 [96] es un motor de juego que utiliza cálculos de física de alta fidelidad basados en PhysX, renderizado basado en shader y otras tecnologías interesantes como manejo de niveles de detalle para modelos 3D y descarte de oclusiones (occlusion culling [97]). Unity permite la creación de juegos para diferentes plataformas, tales como PC, Mac, Nintendo Wii, iPhone y sistemas basados en Android. La Figura 22 muestra el aspecto de un juego y una simulación obtenidos ambos con el motor de juego Unity 3.

Figura 22. Capturas de pantalla del motor Unity. La imagen de la izquierda está obtenida del juego Game of Thrones: Seven Kingdoms[43]. La imagen de la derecha corresponde al simulador SARGE[44]

[43] http://www.forbes.com/sites/erikkain/2012/07/02/bigpoint-announces-game-of-thrones-seven-kingdoms-comic-con-trailer-on-the-way/

[44] http://sourceforge.net/projects/sarge/

Capítulo 3

METODOLOGÍA Y HERRAMIENTAS

El procedimiento seguido para alcanzar los objetivos de esta investigación consta de varias fases que a continuación se resumen:

- Revisión de las teorías del aprendizaje mediante entornos virtuales (videojuegos y simuladores).
- Selección de los simuladores para robótica más interesantes desde un punto de vista educativo, aplicando criterios basados en teorías del aprendizaje.
- Implementación de simulaciones didácticas y atractivas. Desarrollo de entornos de simulación, modelos de robots y sensores.
    - Diseño de modelos 3D. Modelado del entorno 3D, robots, sensores, actuadores y objetos interactivos.
    - Exportación e importación de modelos, texturas y animaciones.
    - Construcción de un entorno virtual.
    - Programación de la inteligencia artificial del entorno.
    - Definición de robots, sensores y actuadores.
- Pruebas de rendimiento y capacidad de integración en sistemas de gestión de aprendizaje online.

El diagrama de flujo de la Figura 23 muestra de forma esquemática las operaciones realizadas para llevar a cabo esta investigación, relacionando cada operación con las herramientas empleadas para su ejecución. Este diagrama de flujo indica, también, el procedimiento de pruebas con diferentes simuladores seguido para implementar simulaciones hiperrealistas acordes a contrastados criterios pedagógicos, basados en teorías del aprendizaje.

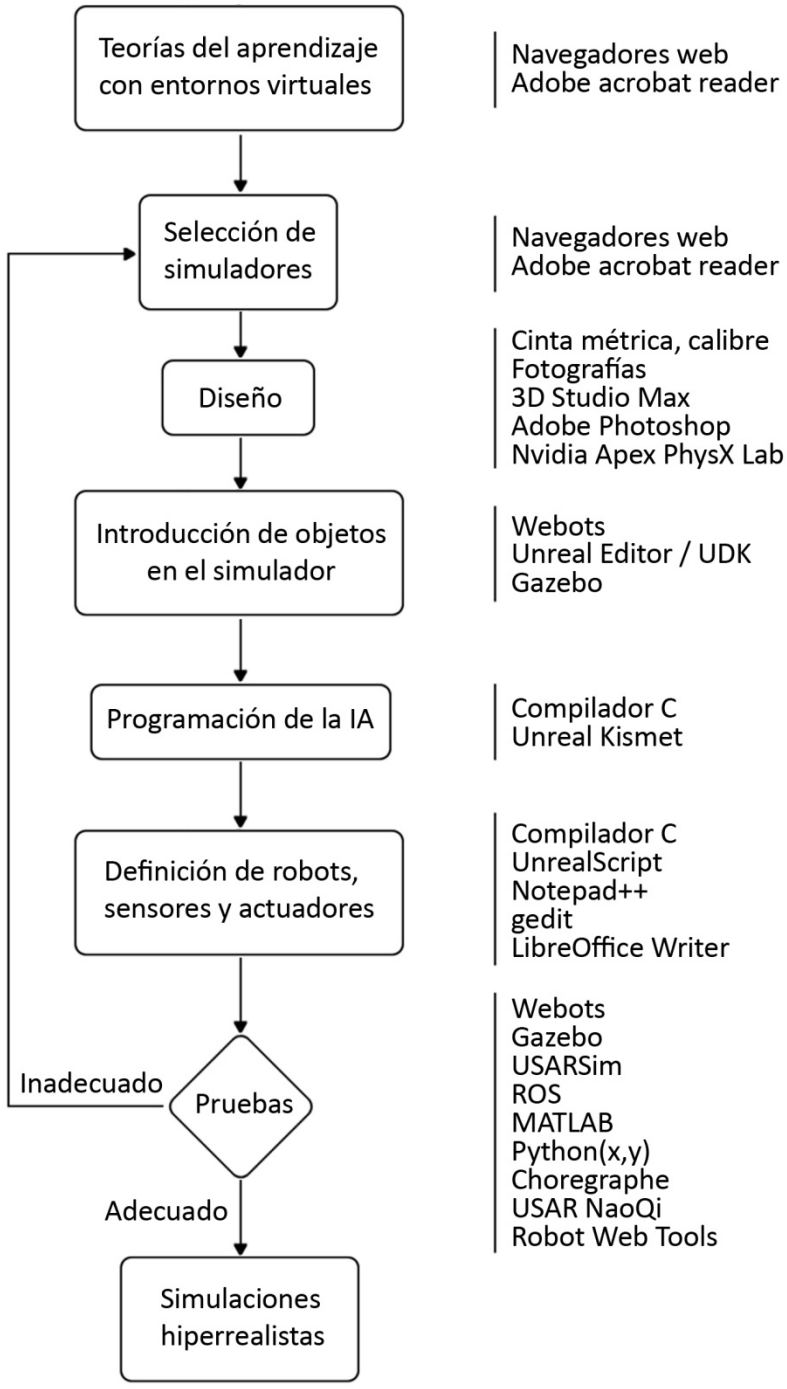

Figura 23. Metodología general para la obtención de simulaciones hiperrealistas en robótica educativa

3.1. Revisión de las teorías sobre aprendizaje con entornos virtuales

Los objetivos generales de la formación basada en entornos virtuales (simuladores o videojuegos) son acelerar el aprendizaje y asegurar la transferencia de lo aprendido (conocimientos, habilidades y actitudes) a las tareas del mundo real [44]. Aunque existen numerosos estudios sobre el aprendizaje mediante entornos virtuales [26] [98] [28] [99], no son tantos los trabajos destinados a conocer qué componentes (o atributos) de los videojuegos o simuladores tienen una influencia contrastada sobre los resultados de aprendizaje [29].

En esta tesis, se revisan las teorías que relacionan los atributos de los videojuegos y simuladores con los resultados de aprendizaje, buscando información útil que sirva de base para la obtención simulaciones didácticas y atractivas.

Las principales herramientas utilizadas en esta fase son navegadores web y visualizadores de archivos en formato *.pdf*.

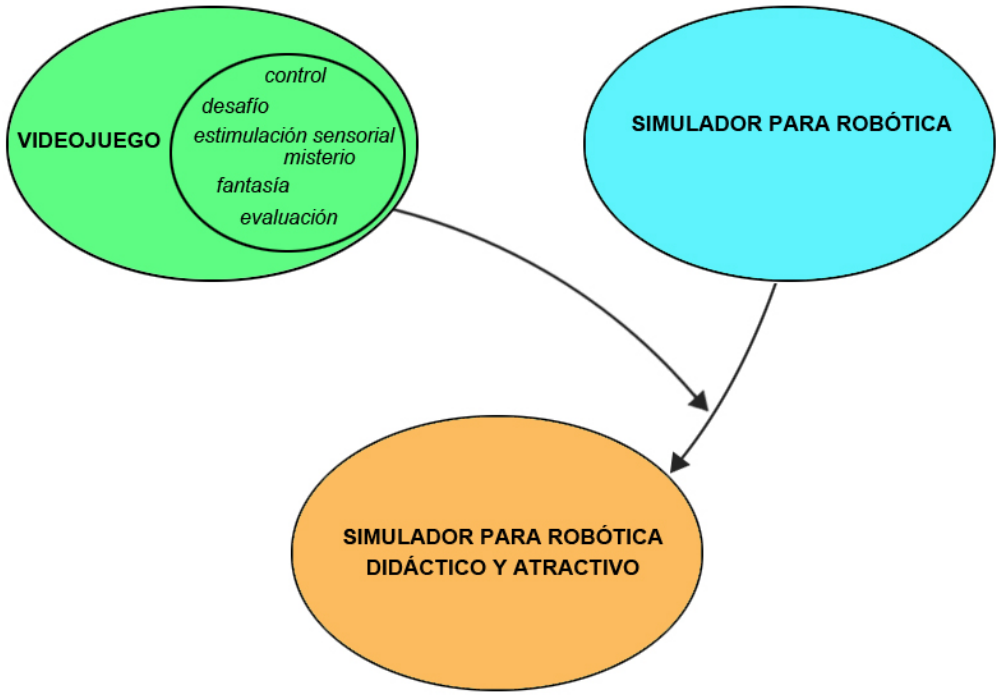

Figura 24. Atributos de los videojuegos importantes para obtener simulaciones didácticas que capten el interés de los estudiantes

3.2. Selección de simuladores para robótica educativa

Tal como indican algunos autores, el diseño de videojuegos didácticos (juegos serios) y simuladores es una tarea compleja en la que resulta crucial una acertada selección inicial del software a emplear [25]. Así, es primordial llevar a cabo un estudio comparativo de las capacidades de los actuales simuladores de sistemas robóticos que determine cuáles son, a priori, los más convenientes para los objetivos de esta tesis.

Los simuladores más interesantes para robótica educativa los he escogido combinando principios de teorías del aprendizaje con criterios generales para la comparación de simuladores y motores de juegos. Afortunadamente, existen varios estudios que analizan y comparan tanto simuladores robóticos como motores de videojuegos orientados a aplicaciones educativas [88] [58] [59]. En esta selección de simuladores también han tenido influencia mis propias experiencias personales en la creación de simulaciones de alta fidelidad [42]. Como resultado he obtenido un conjunto inicial compuesto por tres simuladores que han pasado a una fase de comprobación empírica, en la cual se han puesto a prueba sus capacidades para la implementación de simulaciones de alto valor pedagógico.

En esta fase de la investigación he empleado navegadores web y visualizadores de archivos en formato *.pdf* como principales herramientas de trabajo.

3.3. Implementación de simulaciones didácticas y atractivas

Una vez elegidos los simuladores más interesantes, he procedido al desarrollo de varios entornos de simulación orientados a satisfacer distintas necesidades de la docencia en ingeniería informática principalmente. Se trata de un total de 9 entornos virtuales, algunos de ellos de gran tamaño y con propósitos múltiples, creados mediante diferentes simuladores y diseñados para facilitar la implementación de diversos algoritmos relacionados con la inteligencia artificial y la competición entre estudiantes fundamentalmente.

Cada uno de los mencionados entornos está compuesto, en cuanto a modelos 3D, por elementos estáticos, robots, objetos interactivos y entes animados fundamentalmente. Los elementos estáticos son aquellos objetos inmóviles sin ningún tipo de comportamiento físico asociado (muebles pesados, elementos estructurales, bóveda celestial, etc.). Los robots, por el contrario, están compuestos

por numerosas partes móviles (piernas, ruedas, etc.), sensores (cámaras, micrófonos, sensores de distancia, etc.) y actuadores (brazos, pinzas, etc.) con funcionamientos y comportamientos físicos específicos. Los objetos interactivos son elementos normalmente inertes que a diferencia del entorno estático reaccionan ante las acciones o fuerzas aplicadas por un robot, ya que poseen un comportamiento físico definido (libros, puertas, muñecos, etc.). Entes animados son todos aquellos objetos, máquinas o animales, normalmente compuestos por multitud de piezas unidas mediante articulaciones, que se mueven dentro el entorno virtual simulando tener un funcionamiento o vida inteligente.

Además, aparte de los modelos 3D, existen otros tipos de elementos indispensables como la inteligencia artificial del propio entorno, sonidos, mensajes de texto para el usuario, etc. que ayudan a crear una atmósfera adecuada para el aprendizaje, tal como se detalla en capítulos posteriores de esta tesis.

La Figura 25 muestra una vista general del procedimiento que he seguido para la implementación de las simulaciones, resaltando las herramientas software empleadas y el flujo de información entre ellas. Los recursos que he generado mediante diferentes aplicaciones de diseño asistido por ordenador los he importado desde cada simulador, engrosando su biblioteca de recursos y quedando disponibles para su posterior uso. Después, mediante el editor de niveles de cada simulador he construido diferentes entornos que he dotado de cierta inteligencia e interactividad haciendo uso de los distintos editores de inteligencia artificial que ofrecen los simuladores.

El desarrollo de estos entornos obedece a un triple objetivo. Primero, durante la construcción del entorno y programación de clases de objetos, se evalúan las capacidades del simulador y su facilidad de uso. Segundo, con el entorno terminado y los modelos de robots introducidos, se procede la programación de comportamientos con diferentes lenguajes (C, Python, etc.), utilizando con frecuencia middlewares y equipos remotos, lo que permite ver la facilidad de integración del simulador con el software habitualmente empleado por los estudiantes de ingeniería informática. Tercero, si el simulador resulta útil y adecuado se pone a disposición de los alumnos para sus sesiones prácticas, con lo que se obtiene una realimentación adicional por su parte y un beneficio para la comunidad educativa.

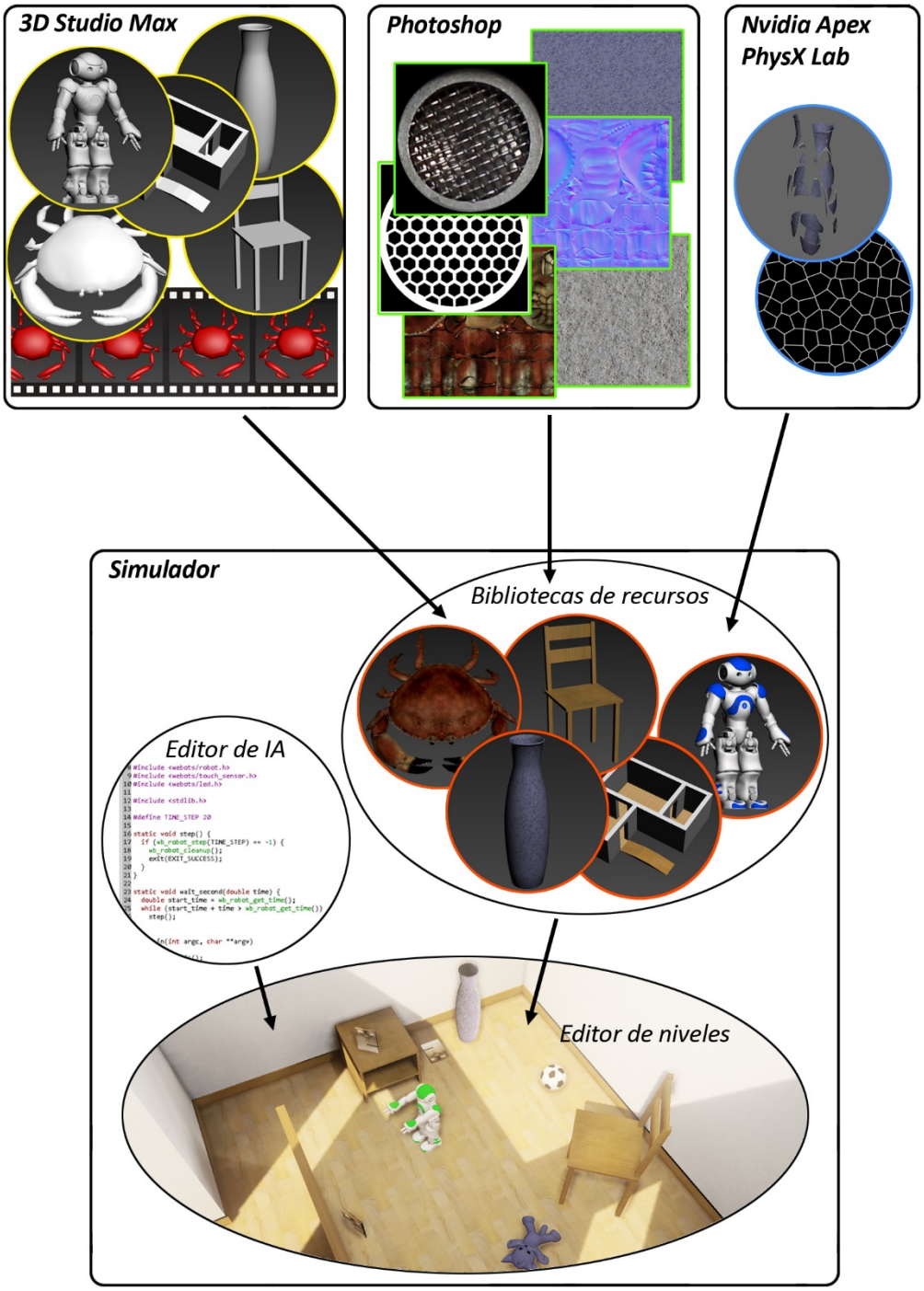

Figura 25. Procedimiento general para el diseño de modelos 3D y la creación de entornos de simulación

3.3.1. Diseño de modelos 3D

Esta parte del trabajo incluye el modelado 3D de edificios, mobiliario, robots, actuadores, sensores y objetos variados, así como la creación de las animaciones necesarias. Se trata de una gran cantidad de modelos 3D, algunos de ellos indispensables para poder ejecutar cualquier simulación (terreno, robots, etc.) y otros necesarios bien para incrementar la funcionalidad del entorno o su realismo audiovisual. Hay que tener presente que el grado de realismo y la capacidad para producir experiencias inmersivas aumentan con el número de objetos presentes en la escena [44].

Tal como se justifica en capítulos posteriores de esta tesis, a medida que aumenta el número de modelos de alta fidelidad gráfica presentes en una escena resulta conveniente la utilización de niveles de detalle reducido para los objetos que se encuentran más alejados del punto de vista del usuario [100] [101] [102]. Esto recomienda la generación de varios modelos con distintos niveles de detalle, al menos para los objetos con más polígonos, lo cual deriva en un aumento del número de modelos 3D a desarrollar.

Para llevar a cabo esta fase he utilizado planos y he efectuado mediciones para definir la forma y dimensiones de los modelos 3D. Además, para dar a los modelos el material (color, textura, opacidad, reflexión, etc.) adecuado, he tomado fotografías y he buscado texturas por la red, las cuales a su vez han tenido que ser acondicionadas a las necesidades de cada modelo. Estas tareas las he llevado a cabo utilizando aplicaciones informáticas de modelado y animación como 3D Studio Max [79] [78] y programas retoque digital de imágenes como Adobe Photoshop [103], con sus correspondientes plugins específicos para cada tipo de operación. En este sentido, han resultado de especial utilidad el plugin ActorX[45] y NVIDIA Normal Map Filter[46].

ActorX es un plugin de Epic Games[47] para aplicaciones de diseño en 3D como Maya[48] [76] [77] y 3D Studio Max[49] que permite exportar objetos dotados de esqueleto (con

[45] http://udn.epicgames.com/Two/ActorX.html
[46] http://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop
[47] http://epicgames.com/about
[48] http://www.autodesk.com/products/maya/overview
[49] http://www.autodesk.com/products/3ds-max/overview

articulaciones) y sus animaciones en formato *.psk* y *.psa* respectivamente (ver Figura 28).

NVIDIA Normal Map Filter es un plugin de NVIDIA[50] para Adobe Photoshop que facilita la creación de texturas de relieve de tipo normal map a partir del nivel de brillo de una imagen.

La Figura 26 muestra una captura de pantalla de 3D Studio Max donde se ilustra de manera gráfica el proceso de generación de modelos 3D a partir de datos geométricos y materiales. El robot de la imagen es un humanoide NAO de Aldebaran Robotics[51].

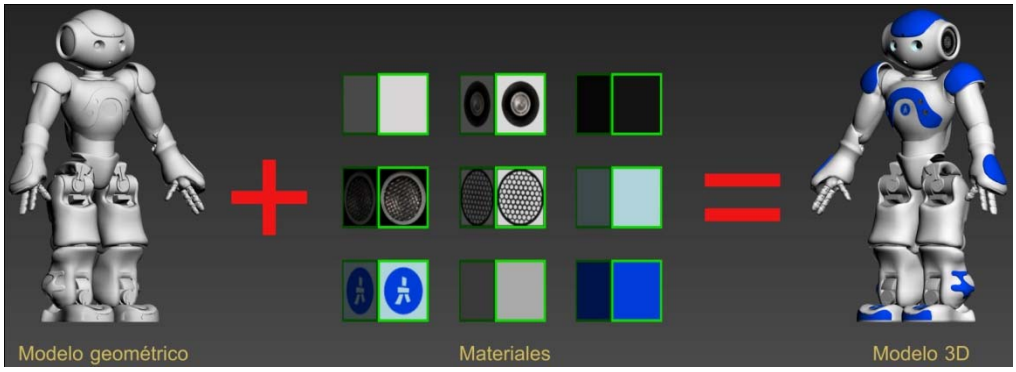

Figura 26. Proceso para la generación de un modelo 3D a partir de datos geométricos y materiales

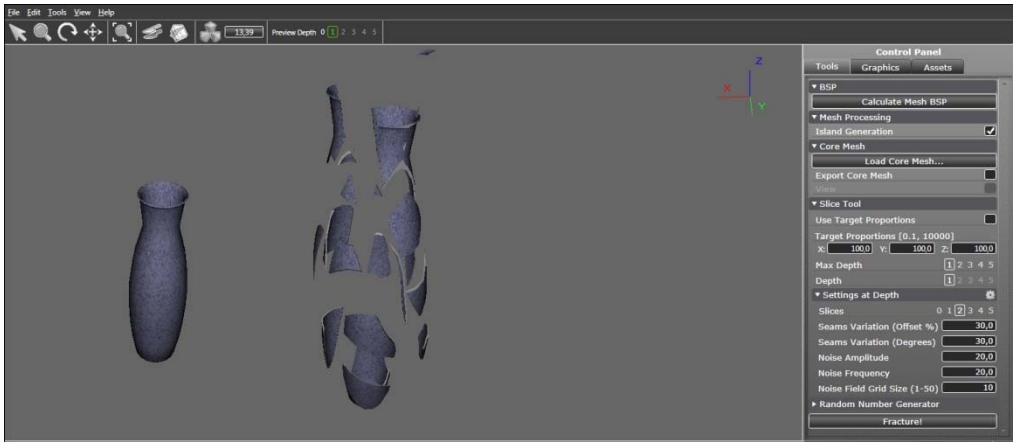

Figura 27. Captura de pantalla de la creación de un jarrón destructible con Nvidia Apex PhysX Lab

[50] http://www.nvidia.com/object/about-nvidia.html
[51] http://doc.aldebaran.com/1-14/family/nao_h25/index_h25.html

En el caso de los objetos y estructuras destructibles también he recurrido a aplicaciones software como Nvidia Apex PhysX Lab[52] para el diseño de patrones aleatorios de fractura fundamentalmente (ver Figura 27). Con este software es posible definir varios niveles de fractura, las texturas que aparecen en los fragmentos desprendidos y núcleos indestructibles tales como armaduras de hormigón, etc.

3.3.2. Exportación e importación de modelos, texturas y animaciones

Los modelos 3D y sus propiedades (texturas, envolventes de colisión, articulaciones, animaciones,...) creados mediante aplicaciones CAD deben ser reconocibles por los editores de modelos y niveles de cada simulador para poder ser utilizados. Esto requiere exportar los modelos obtenidos en diferentes formatos en función de su naturaleza (elementos estáticos, móviles, articulados, animados, etc.) y la del simulador destino.

La Figura 28 muestra el flujo de archivos y formatos entre las aplicaciones software que he empleado para la creación de los mencionados entornos de simulación. Los 4 rectángulos representan las principales aplicaciones software que he usado. Los círculos simbolizan los tipos de recursos generados mediante cada aplicación. Las flechas indican el flujo de información entre aplicaciones y el texto que las acompaña corresponde al formato empleado para transferir tal información.

Es destacable el hecho que cada editor de niveles y modelos (objetos, robots, sensores, actuadores, etc.) requiere que éstos estén guardados en un formato o formatos específicos diferentes para poder ser importados sin perder información relevante. Esto supone un incremento importante del volumen de archivos a manejar cuando se pretende utilizar un mismo recurso (modelo 3D, textura, animación, fractura, etc.) en distintos simuladores.

La necesidad de un elevado número de modelos apuntada en el apartado anterior unida a la gran diversidad de formatos necesarios para las operaciones de exportación e importación entre aplicaciones software, ha requerido manejar una gran cantidad de información que adjunto en un DVD organizado por entornos y tipos de recursos.

[52] http://physxinfo.com/wiki/PhysXLab

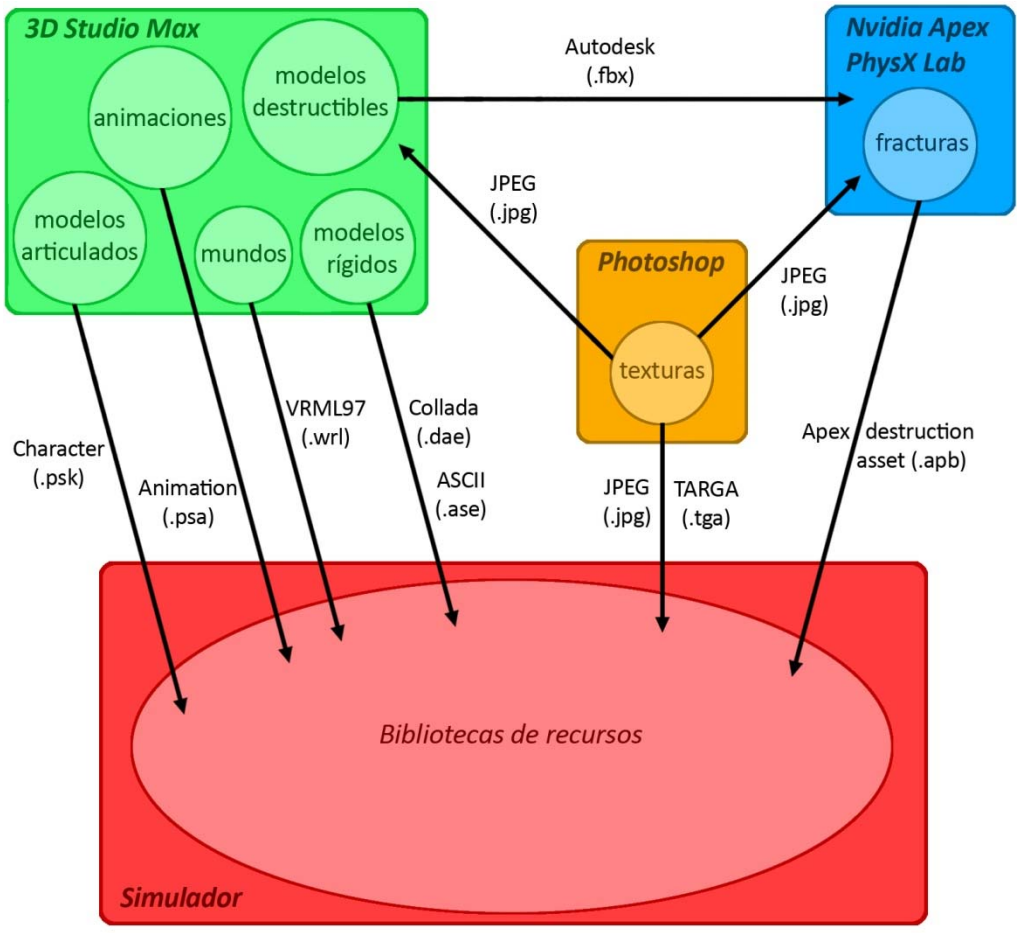

Figura 28. Flujo de archivos entre las principales herramientas para el desarrollo de simulaciones

3.3.3. Construcción de un entorno virtual

Una vez importados todos los modelos 3D, sus texturas y sus animaciones, y guardados en las bases de datos de cada simulador, he utilizado diferentes editores de niveles para crear entornos virtuales con distintos simuladores. En general, aunque existen otras maneras proceder, cada simulador ofrece la posibilidad de crear mundos virtuales a través de un editor de niveles. Estos editores de niveles permiten desarrollar entornos bien por adición de elementos presentes en una biblioteca de recursos del propio simulador (ver Figura 25) o bien creando in situ formas geométricas que darán lugar a objetos, robots, estructuras, etc. Normalmente, los objetos que pueden conseguirse mediante el uso de las herramientas de dibujo del

propio editor de niveles suelen ser geométricamente simples, motivo por el que acostumbran a utilizarse para crear el suelo, la bóveda celestial o estructuras sencillas. Aquellos objetos complejos por sus texturas, formas geométricas, animaciones o comportamiento físico es recomendable crearlos mediante aplicaciones CAD externas. Es por este motivo que los simuladores permiten la importación de objetos desarrollados con software de diseño 3D y animación como Blender[53] [60], 3D Studio Max o Maya. Entre este software yo he elegido 3D Studio Max fundamentalmente por tratarse de la aplicación más recomendada y con mayor compatibilidad de formatos.

3.3.4. Programación de la inteligencia artificial del entorno

Las competiciones entre estudiantes sugieren la incorporación de algoritmos que detecten si alguno de los contrincantes ha alcanzado el objetivo planteado y, por tanto, el juego ha terminado y hay un ganador. Del mismo modo, cuando un entorno de simulación va dirigido a estudiantes para la realización de sus ejercicios prácticos, resulta conveniente introducir indicaciones (visuales o auditivas) que les recuerden cuáles son los objetivos de la simulación y den realimentación de los logros alcanzados, errores cometidos o estado general de la simulación [29]. Además, la presencia en los entornos virtuales de objetos o personajes que muestran interactividad o un comportamiento aparentemente inteligente ayuda a incrementar el grado de control de los estudiantes sobre el entorno de aprendizaje, facilitando experiencias más pedagógicas [44].

Aunque es fácil encontrar entornos de simulación desprovistos de inteligencia artificial, tal como se justifica detalladamente más adelante en esta tesis éste es un componente muy importante para generar experiencias didácticas y atractivas. Por este motivo, he utilizado los editores de inteligencia artificial disponibles en cada simulador para incrementar la interactividad y funcionalidad tanto de los entornos como de los objetos presentes en ellos. En este aspecto los simuladores que he utilizado ofrecen opciones muy distintas que van desde la programación mediante bloques funcionales hasta el uso de instrucciones en lenguaje C.

[53] http://www.blender.org/about

3.3.5. Definición de clases

Los robots, sus sensores, sus actuadores o incluso sus materiales deben estar descritos por clases que los definan completamente para poder ser utilizados en los distintos simuladores. Como es de esperar, cada simulador exige que el desarrollador de entornos defina sus clases de objetos empleando diferentes métodos y lenguajes de programación. Sin embargo, a pesar de las aparentes diferencias de forma, es común que exista al menos un archivo que defina cada clase de objeto. Estos archivos describen al simulador qué partes componen cada objeto, cuál es la relación entre ellas, cuál es la función que desempeñan, etc.

Aunque el procedimiento de definición clases varía en función del simulador y de los objetos en cuestión, a modo de ejemplo muestro en la Figura 29 las clases que componen un robot humanoide Nao en condiciones de simulación con USARSim v1.2 para UDK. En este caso la clase robot NaoV32 puede equiparse con diferentes sensores (sonar, cámara, acelerómetros, etc.) en función de las necesidades de la simulación.

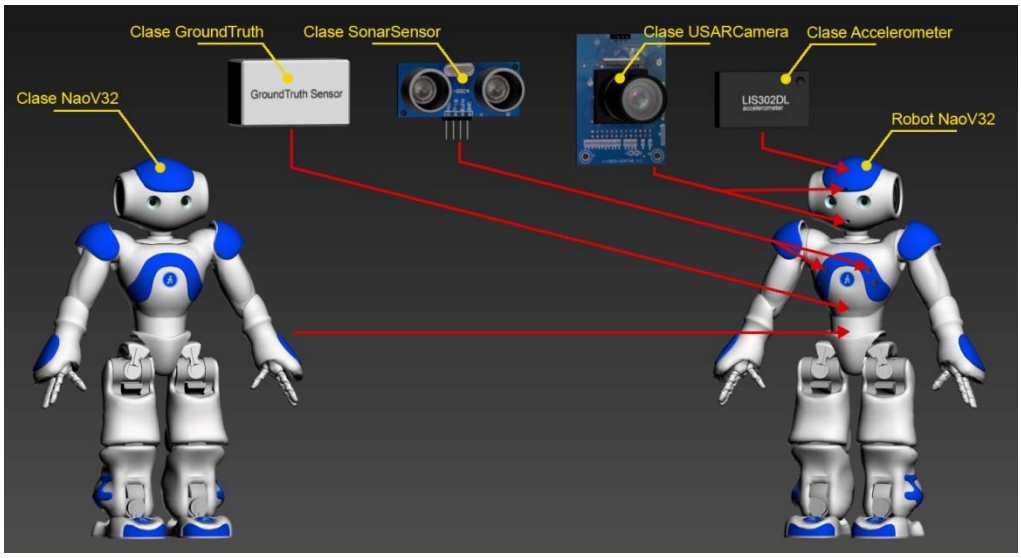

Figura 29. Clases de objetos que componen un humanoide Nao en condiciones de simulación con USARSim v1.2 para UDK

Esta parte del trabajo la he realizado utilizando editores de texto como gedit, Notepad++ y LibreOffice Writer para escribir código en UnrealScript, Python o C.

3.4. Pruebas de rendimiento e integración en sistemas de gestión de aprendizaje

Las pruebas de rendimiento que he realizado consisten en unos ensayos simples donde se comprueba si la simulación se ejecuta correctamente y en tiempo real. En estos ensayos se lanza un robot en un entorno, se programa un comportamiento para tal robot y se observa si todo funciona según lo esperado, de manera fluida y sin mensajes de error o advertencia. En esta parte del trabajo es fundamental la buena programación de la inteligencia artificial del entorno y la compatibilidad entre las diferentes versiones de simuladores, middlewares y entornos de programación.

La fluidez con la que se ejecutan las simulaciones depende en buena medida de las características y capacidades de los equipos informáticos empleados. Es por ello que el buen funcionamiento de las simulaciones los he comprobado en varios equipos informáticos con diferentes configuraciones. Con estos ensayos he observado el grado de optimización de los modelos de robots y entornos en cuanto a la carga física y gráfica que introducen. En algunos casos el propio simulador ofrece lecturas de la velocidad de ejecución y en otros he medido esta velocidad en fotogramas por segundo (fps), tomando como buenas las lecturas equivalentes a ejecución en tiempo real o superior a 30 fps[54] [104] [105].

Para terminar, he comprobado la facilidad que ofrecen los simuladores a la hora de preparar recursos educativos online. Los entornos que he considerado más interesantes para esta tesis los he puesto a disposición de los estudiantes de robótica bien integrándolos en un sistema de gestión de aprendizaje online o bien permitiendo la descarga de tales entornos a los equipos participantes en una competición con robots humanoides[55] [106].

[54] http://www.tweakguides.com/Graphics_5.html
[55] http://www.irs.uji.es/humabot/

Capítulo 4

SIMULADOR PARA UN LABORATORIO DE ROBÓTICA. ANTECEDENTES.

En este apartado se detallan mis experiencias de hace más de 7 años en desarrollo de entornos de simulación para robótica educativa las cuales, sin duda, han ejercido gran influencia en mi valoración de los simuladores bajo estudio.

4.1. Contexto: Robots autónomos móviles en proyecto GUARDIANS y máster de Sistemas Inteligentes

En 2008, el Laboratorio de Robótica Inteligente de la Universidad Jaume I de Castellón contaba con media docena de robots Erratic de Videre Design entre otros, los cuales deberían cubrir las necesidades que surgieran en investigación y docencia. En cuanto a la formación, personal docente vinculado a este laboratorio determinó que resultaba necesario disponer de un simulador para llevar a cabo sesiones prácticas con alumnos de un máster en Sistemas Inteligentes. Los resultados obtenidos durante las sesiones de simulación deberían permitir, además, ser contrastados con los datos recogidos en los ensayos con robots reales, de manera que si el simulador resultaba fiable sería instalado en los ordenadores de las aulas de informática disponibles para tales sesiones. Esto facilitaría la tarea docente, permitiendo a los alumnos poner a prueba sus algoritmos de control de manera simultánea sin necesidad de adquirir más robots o disponer de grandes espacios para ello.

En lo referente a la investigación, en 2008 personal investigador ligado al Laboratorio de Robótica Inteligente de la UJI estaba trabajando en el proyecto *GUARDIANS* (Group of Unmanned Assistant Robots Deployed In Aggregative Navigation by Scent) [107] [108]. Se trata de un proyecto europeo en el que trabajaba el equipo de la Universidad Jaume I en colaboración con universidades y empresas de España Alemania, Reino Unido, Bélgica, Portugal y Turquía. Los guardians son un enjambre de robots

autónomos que realizan tareas de navegación y búsqueda sobre un terreno urbano potencialmente peligroso. La idea principal del proyecto es que los robots puedan ayudar a los seres humanos en las peligrosas situaciones que afrontan los equipos de emergencia como los bomberos. Los robots involucrados deben ejecutar algoritmos destinados a esquivar las trayectorias de otros robots y de los seres humanos, mientras se mantienen en la cercanía de un ser humano. El modelo de robot elegido para formar tales enjambres en la Universidad Jaume I fue el Erratic. Aunque las simulaciones del proyecto GUARDIANS se estaban implementando con Player/Stage en modo 2D, se consideró interesante simular los entornos en 3D. De este modo las simulaciones ganarían en realismo incorporando los efectos que las nubes de humo tienen sobre la visión de las cámaras de los robots.

Figura 30. Idea principal del proyecto GUARDIANS. La imagen de la izquierda corresponde a una recreación en 3D donde un enjambre de robots colabora con un bombero en tareas de búsqueda y rescate. La imagen de la derecha pertenece a un experimento real llevado a cabo en los pasillos del edificio TI de la UJI

4.2. Baja fidelidad general en los simuladores de 2008

Después de un estudio de las posibilidades y características de los simuladores para robótica de alta fidelidad física y gráfica disponibles en aquel momento, observé que ninguno de ellos incorporaba el modelo Erratic mencionado, por lo que sería necesario adoptar un simulador que permitiera introducir esta plataforma robótica. Aquí cabe destacar que pocos eran en 2008 los simuladores para robótica capaces de ofrecer unos gráficos parecidos a los que hoy consideramos de alta fidelidad. Apenas

Webots 5 y USARSim para UT2004 (Unreal Tournament 2004) alcanzaban el nivel buscado, siendo Webots 5 la opción menos atractiva por tratarse un simulador comercial con su coste económico asociado. Así, llegué a la conclusión que la mejor opción podría ser USARSim v.3.3, pues su motor de juego, Unreal Engine 2, ofrecía gráficos de muy alta calidad y precisos cálculos de física basados en Karma Physics[56] a muy bajo coste. USASim v.3.3 era un simulador open source que requería una licencia del exitoso videojuego FPS (First Person Shooter) Unreal Tournament 2004 para su instalación, lo que reducía su precio a menos de 50,00€. Este bajo coste lo convertía en adecuado para ser instalado en multitud de equipos para las sesiones prácticas de los alumnos de las asignaturas de robótica. Además, USARSim presentaba el atractivo adicional de ser utilizado para famosas competiciones robóticas como las mencionadas RoboCup Rescue [50] y VMAC [51].

4.3. Síntesis de mi experiencia con USARSim para UT2004

USARSim v.3.3 para UT2004 fue tomado como base para realizar un entorno de simulación personalizado y adaptado a las necesidades del personal docente e investigador del Departamento de Ciencia de los Computadores de la Universitat Jaume I de Castellón. Como se explica más adelante, a las plataformas robóticas incluidas en USARSim v.3.3 añadí el robot Erratic de Videre Design y como entorno virtual introduje el edificio TI de la Universitat Jaume I de Castellón (ver Figura 31 y Figura 32) que alberga principalmente seminarios, despachos y laboratorios [42].

Figura 31. Fotografía del exterior del edificio TI en el Campus Riu Sec de la Universidad Jaume I de Castellón

[56] http://udn.epicgames.com/Two/KarmaReference.html

Figura 32. Vista exterior del edificio TI virtual con sus alrededores inmediatos y la biblioteca

La elección del edificio TI de la UJI se debió a que en sus laboratorios se realizan numerosos ensayos con robots y, además, sus pasillos son utilizados con frecuencia para probar algoritmos de navegación y mapeado como los empleados en el proyecto GUARDIANS (ver Figura 30).

El entorno de simulación que resultó del trabajo con USARSim v.3.3 superó con creces las necesidades identificadas por el personal docente e investigador del Departamento de Ingeniería y Ciencia de los Computadores de la Universidad Jaume I de Castellón. En la Figura 33 se puede apreciar el elevado grado de similitud entre el edificio real y el entorno simulado.

Figura 33. A la izquierda una imagen real del robot Erratic en un pasillo del edificio TI de la UJI. A la derecha una imagen del simulador

El modelo de robot Erratic obtenido para USARSim v.3.3 permitió simular dicho robot con una fidelidad gráfica y física elevada. La Figura 34 muestra la sensible diferencia en nivel de detalle entre mi modelo del robot Erratic introducido y el modelo Pioneer 2DX (P2DX) que incorpora USARSim v.3.3. Dicha diferencia de detalle se puede observar cuantificada en triángulos en la Tabla 1. Tal como explico más adelante, la cantidad de polígonos que debían conformar el modelo del Robot Erratic sería el adecuado para las capacidades del hardware del momento, observándose que los robots que incorporaba USARSim v.3.3 estaban bastante por debajo de tal cantidad, incluso para simular grupos de robots.

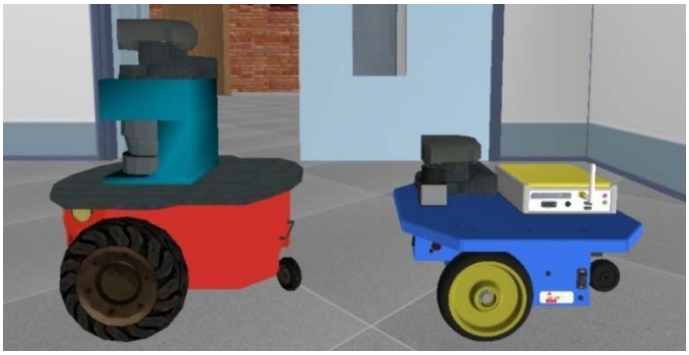

Figura 34. Modelos 3D para la simulación de los robots P2DX (izquierda) y Erratic (derecha)

Robot Part	**P2DX**	**ERA**
Body	449	17569
Computer	0	4690
Laser	224	2968
Left wheel	200	11524
Right wheel	200	11524
Rear Wheel	200	4124
TOTAL	1273	52399

Tabla 1. Detalle de los modelos virtuales de los robots Pioneer 2DX y Erratic medido en triángulos

La Figura 35 compara el comportamiento del entorno de simulación (medido en fotogramas por segundo) utilizando dos equipos informáticos con capacidades muy distintas. El primero es un ordenador portátil con un procesador Intel Core Duo T2300 1.66 GHz, 2 Gb de RAM, y una tarjeta gráfica Nvidia 7400 Go, mientras que el segundo es un ordenador de escritorio con un procesador Intel Core2 Quad Q9300 2500GHz, 4 Gb de RAM y una Nvidia GeForce 7950 GTX. Realicé tres pruebas de rendimiento en

cada equipo: una con solo el entorno, otra con el entorno y el robot P2DX y una última con el entorno y el robot Erratic. Como puede observarse, la simulación se ejecuta a una velocidad de 35 fps en el peor de los casos, lo que se considera un comportamiento fluido al superar los 30 fps[57] [104] [105]. De hecho, es importante remarcar que a pesar del alto nivel de detalle del robot Erratic, ninguno de los dos equipos bajo análisis experimentó un decremento en la fluidez de ejecución de la simulación al introducir el robot. Esto pone de manifiesto que tanto el número de polígonos y como la física del robot virtual Erratic se eligieron de manera correcta, ofreciendo alta fidelidad visual y física y una ejecución en tiempo real muy fluida.

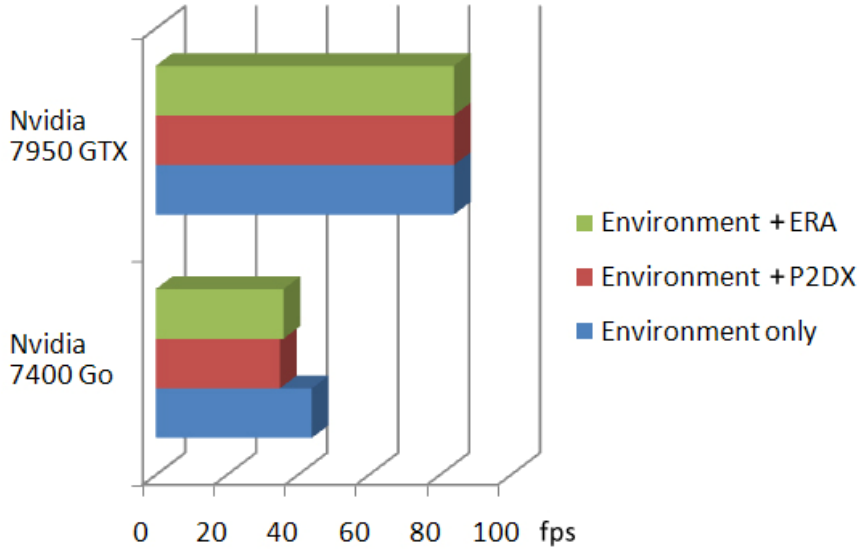

Figura 35. Comparativa de comportamiento del entorno del edificio TI y el robot Erratic (ERA) en simulación con diferentes equipos

USARSim v.3.3 resultó un buen simulador que permitió al personal docente e investigador del Laboratorio de Robótica Inteligente de la UJI reproducir en un entorno virtual diversidad de ensayos con la plataforma robótica Erratic. A su alto grado de fidelidad gráfica y física se unió la versatilidad de su editor de niveles UnrealEd 3 y la facilidad para generar nuevas clases de robots utlizando un lenguaje de scripts llamado *UnrealScript*. Sus posibilidades de programar algoritmos de control mediante middlewares como ROS o MATLAB USARSim Toolbox resultaron

[57] http://www.tweakguides.com/Graphics_5.html

también de especial utilidad por tratarse de aplicaciones bien conocidas en el ámbito de la informática [42].

En los apartados siguientes expongo el proceso seguido para la obtención de simulaciones personalizadas mediante USARSim v.3.3, detallando todas las operaciones que realicé para introducir la nueva plataforma robótica y el edificio TI, desde el punto de vista del diseño CAD y de la programación de nuevas clases con UnrealScript.

4.4. Desarrollo de un simulador con USARSim v.3.3

El proceso necesario para trasladar ensayos con robots a un plano virtual consta de varias fases que, a su vez, dependen del tipo de elemento con el que estemos tratando, ya sea éste un robot, un objeto inanimado o el entorno estático. Por este motivo, expongo el desarrollo de cada tipo de objeto de manera diferenciada.

La Figura 37 muestra el procedimiento que seguí para la obtención de un entorno de simulación totalmente personalizado para USARSim v.3.3. Aunque en los apartados siguientes describo de manera concreta y detallada cómo resolví cada etapa, es interesante tener una visión global de las operaciones que fueron necesarias y las herramientas que ayudaron a ejecutarlas. Así pues, en términos generales, partí de planos y medias para crear modelos 3D de objetos y robots. Estos modelos fueron decorados con texturas que obtuve mediante fotografías en su mayoría y les asigné un contorno para detección de colisiones. Todos estos objetos y robots en 3D fueron tratados con UT2004 para crear una base de datos con información relevante de los mismos y un mundo virtual. Después, programé las clases que definen los robots y las partes que los constituyen (ver Figura 36).

Una vez obtenido el entorno de simulación, utilicé MATLAB y su USARSim ToolBox como entorno de programación para realizar pruebas con algoritmos de comportamiento para los robots. Se trata de ensayos en los que comprobé tanto la fluidez de ejecución del simulador en diferentes equipos como la credibilidad del flujo de información recogida por los sensores del robot. En esta etapa del desarrollo del simulador la capacidad de MATLAB para generar bases de datos con tal información resultó de gran ayuda.

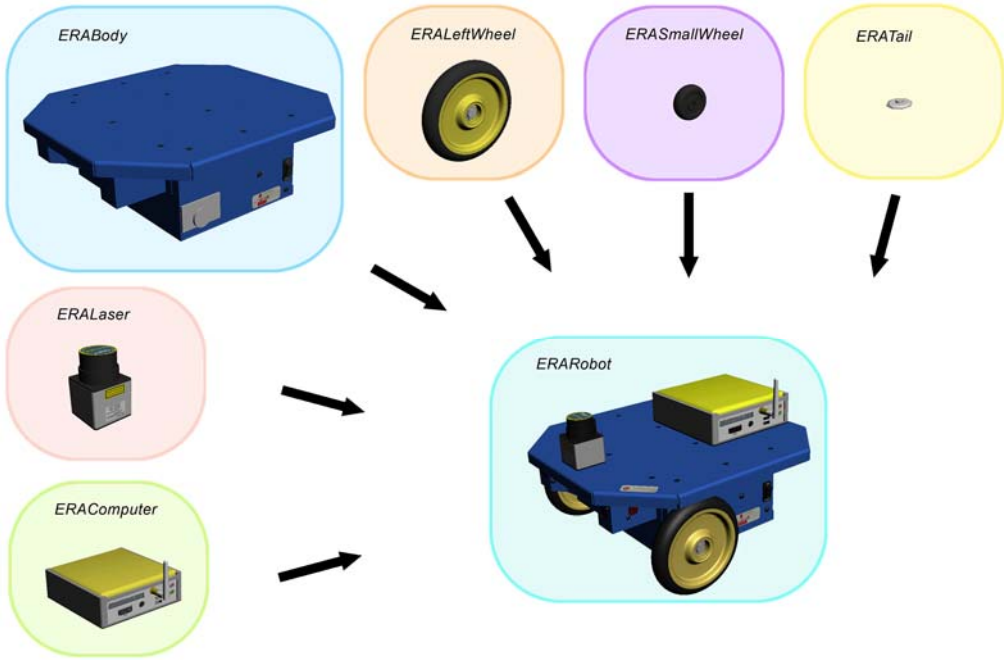

Figura 36. Partes en las que se divide mi modelo virtual para el robot Erratic de Videre Design

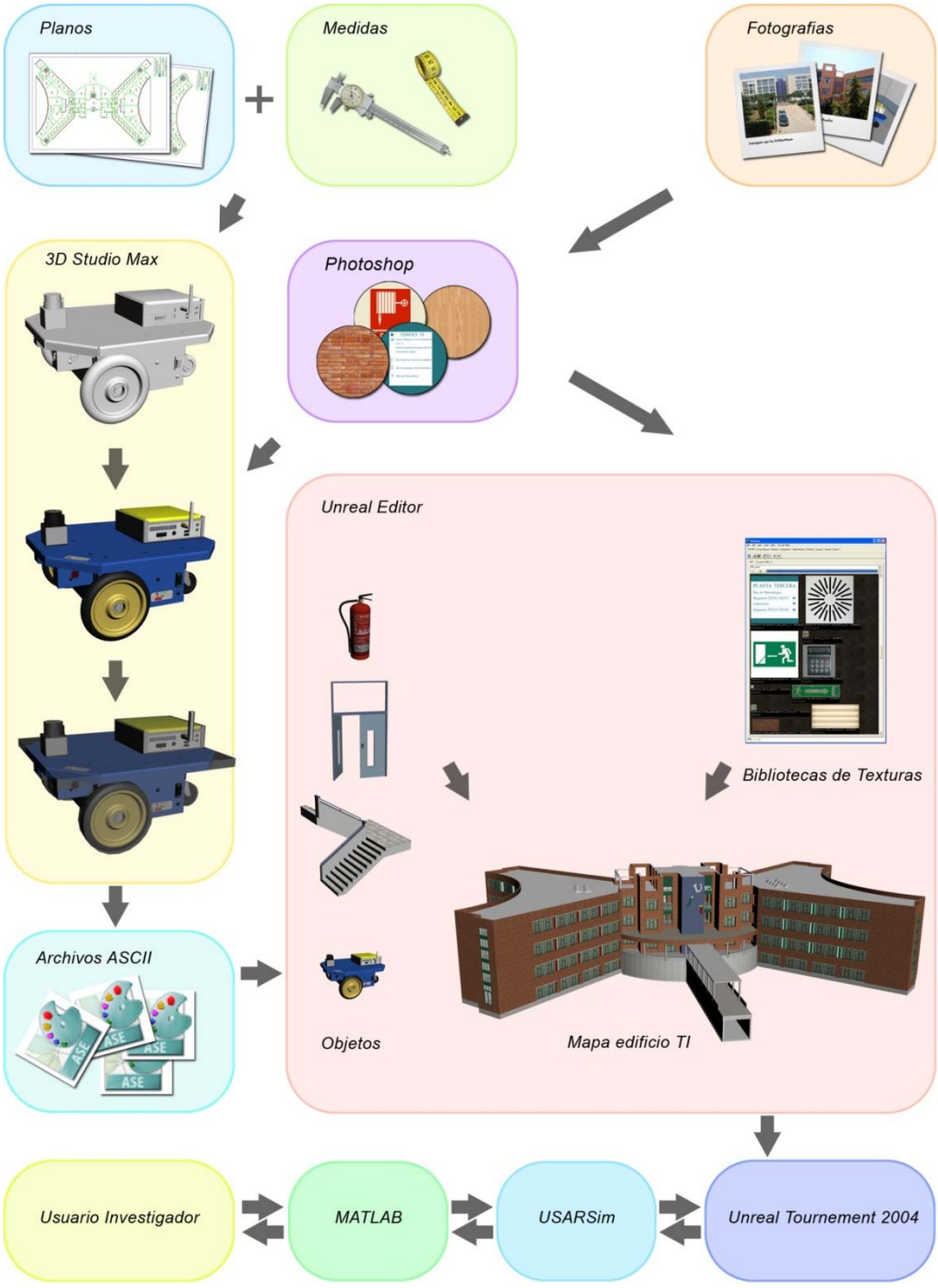

Figura 37. Procedimiento para la obtención de un entorno completo de simulación con USARSim para UT2004

4.4.1. Creación del robot virtual Erratic

La creación del modelo virtual para el robot Erratic consta de varias fases que describo a continuación y que se pueden resumir en: recopilación de mediciones y cotas, toma de fotografías, elaboración de un modelo 3D, desarrollo de materiales multitextura, creación de envolventes de colisión, recopilación de información en una biblioteca de recursos, definición de clases y edición de archivos de configuración.

Planos y medidas

Para obtener el modelo virtual del robot Erratic fue necesario contar con el conjunto de cotas que determinan las dimensiones de todos sus elementos. En este sentido, procedí a la recolección de todos los documentos que contienen ese tipo de información. Sin embargo, en los documentos consultados hallé todas las dimensiones que definen la geometría del robot, de modo que realicé mediciones sobre uno de los robots del Laboratorio de Robótica Inteligente de la UJI para obtener las cotas restantes.

Fotografías

Una vez conseguidas todas las dimensiones que dan la forma al modelo 3D del robot, hay que centrarse en las texturas que otorgan color y apariencia real a los materiales que componen tal modelo. Con este objetivo tomé fotografías de las diferentes partes del robot real, hasta conseguir una representación visual de cada uno de los materiales que lo integran.

Las fotografías las traté con Adobe Photoshop CS3 hasta obtener unas texturas realistas. A su vez, estas texturas debían cumplir dos requisitos extra para poder ser utilizadas por UnrealEd 3 (editor de mapas de Unreal Tournament 2004) y, por tanto, con USARSim. En primer lugar, las texturas debían tener un formato de 2^xx 2^y píxeles. Es decir, tanto la dimensión horizontal como la vertical debían ser potencias de 2. En segundo lugar, estas texturas debían guardarse en formato de archivo TARGA (*.tga) de 24 bits. La Figura 39 muestra el navegador de recursos de Unreal Editor con un grupo de texturas ya importadas en este formato y listas para su uso en USARSim.

Robot 3D

Con las dimensiones y las texturas del robot empezó la fase de desarrollo del modelo virtual utilizando 3D Studio Max. En esta fase es importante un buen trabajo que resuelva el modelo 3D utilizando pocos polígonos. En la Figura 38 puede observarse una captura de pantalla de tal aplicación CAD con el modelo robot Erratic ya completado.

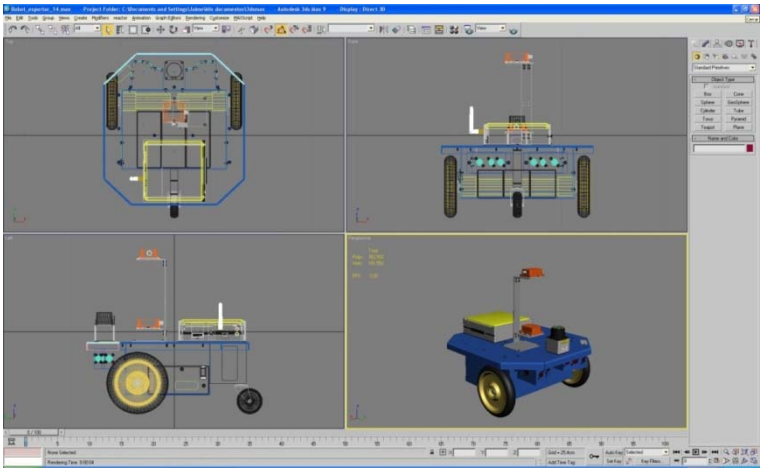

Figura 38. Vistas 2D y perspectiva del robot Erratic durante su modelado con 3D Studio Max

En primer lugar construí el modelo 3D del robot a partir de sus dimensiones, tratando que éste resultase lo más parecido posible al real sin perder velocidad de computación durante las simulaciones. Esto lo llevé a cabo aproximando las formas del robot a conjuntos de primitivas geométricas, las cuales deben estar formadas por un número de polígonos o caras que no resulte demasiado alto y, por tanto, no consuma gran cantidad de tiempo de cómputo. Llegado este momento, el robot 3D estaba compuesto por varios conjuntos de elementos que había que separar o agrupar formando entidades independientes en función del tipo de objeto que constituyen. Por ejemplo, un sonar, un láser, una rueda, el chasis,… son entidades distintas que hay que tratar de forma diferenciada. En general cualquier elemento que tenga una funcionalidad propia o pueda describir movimiento relativo al resto debe considerarse independiente. Es importante indicar que el editor de mapas UnrealEd 3 requiere que los objetos importados de otros programas CAD estén compuestos por una sola malla con todas sus texturas distribuidas en las tres dimensiones. Además, estas texturas

deben estar debidamente agrupadas y almacenadas en un archivo con extensión *.utx (ver Figura 39).

En segundo lugar, cada uno de estos elementos independientes lo transformé en una malla (*mesh* en 3D Studio Max) única. Esta operación elimina toda información geométrica previa y convierte el objeto 3D en un conjunto de polígonos definidos por caras triangulares. Como ya he indicado, cuanto mayor es el número de triángulos, mayor es el parecido del modelo con el objeto real, pero mayor resulta la carga computacional. Por tanto, es necesario llegar a un grado de simplificación de la realidad que resuelva el compromiso entre calidad gráfica y velocidad de simulación. Esto no siempre resulta sencillo, debido a la subjetividad con la que los humanos percibimos los objetos, es decir, no hay una ecuación matemática que indique qué densidad de polígonos ha de tener un objeto 3D que va a ser observado a una determinada distancia. En cambio, como guía suele tomarse la cantidad total de polígonos que el motor gráfico permita representar de manera fluida (por encima de 30 fps) con el hardware disponible. En 2008 esta cantidad se tomó de unos 120.000 polígonos y los ordenadores para las simulaciones se dotaron de tarjetas gráficas con el chip Nvidia GT 440 cuyas capacidades estaban por encima del trabajo a realizar. Teniendo en cuenta estas limitaciones, el modelo del robot lo limité a unos 50.000 polígonos para que pudiera ser observado de cerca sin perder apenas detalle, dejando otros 70.000 polígonos para el entorno del edificio. Con el fin de poder simular conjuntos de robots de manera fluida, creé una segunda versión del robot Erratic con 18.000 polígonos.

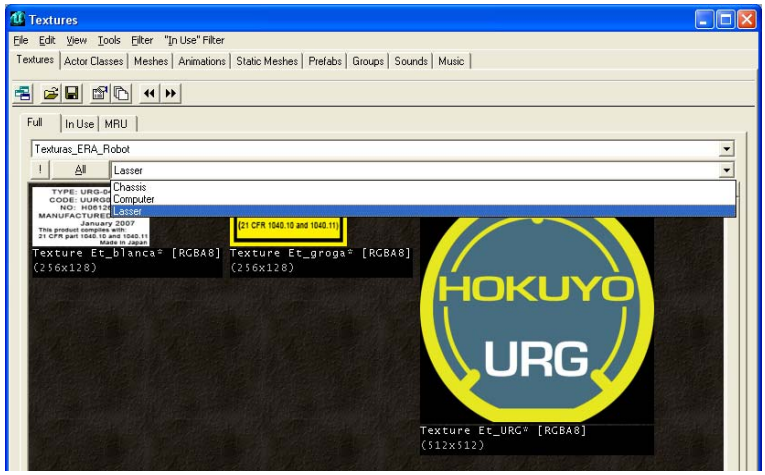

Figura 39. Buscador de texturas de Unreal Editor mostrando algunas texturas del robot Erratic

Material multitextura

A continuación, con las partes del robot reducidas a mallas, procedí a dotarlas de las sus texturas correspondientes. La mayoría de las partes que componen el robot presentan más de un color o textura y, en cambio, sólo pueden constituir una única malla. Esto requiere utilizar los llamados *materiales multi-subobjeto* cuando trabajamos con 3D Studio Max, los cuales pueden estar formados por tantas texturas como se desee. En la Figura 40 se muestra el sensor laser del robot Erratic con un material compuesto por cinco texturas.

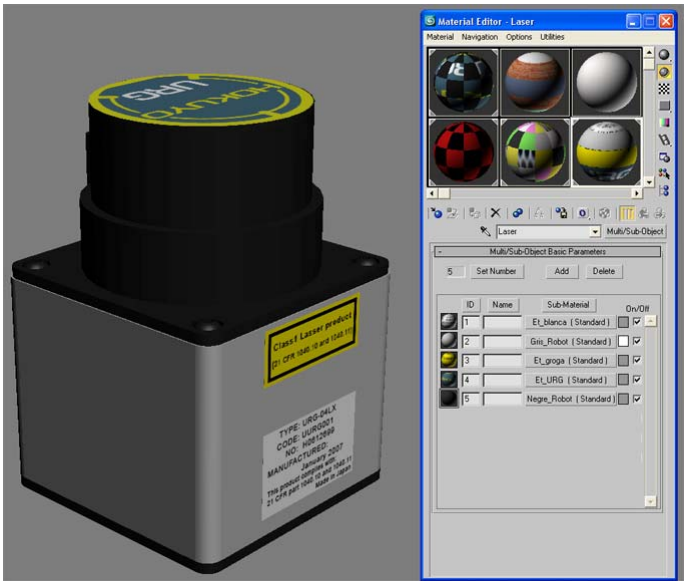

Figura 40. Sensor Laser en 3D Studio Max con el editor de materiales desplegado a la derecha

Así pues, elaboré un material adecuado para cada elemento independiente (malla) del robot y definí su distribución sobre el mismo mediante un mapa de texturas en los ejes U, V y W, tal como requiere el UnrealEd 3 para la correcta importación de texturas.

Colisiones

Una vez terminado el aspecto físico de los elementos que componen el robot, debe dotarse a los mismos de una envolvente que defina su contorno de colisión. De lo contrario, el robot atravesaría todas las superficies que se encontrara en su camino, el cual sería en caída libre, pues atravesaría también el suelo.

UnrealEd 3 permite asignar envolventes de colisión tanto a los objetos creados con este programa como a los importados en formato ASCII (*.*ASE*). El problema que reviste utilizar UnrealEd 3 para definir las colisiones de objetos importados, es que sus envolventes asociadas bien deben coincidir con la malla del objeto, o bien han de seguir geometrías muy simples (cubos, cilindros, esferas,..). En el primer caso, el tiempo de cómputo de una envolvente de estas características se verá incrementado considerablemente por contener un elevado número de triángulos o caras. En el segundo caso, obtendremos un alto rendimiento computacional, pero no resultará fácil ajustar una sola forma geométrica simple como envolvente del objeto en cuestión. En la Figura 41 puede observarse cómo un objeto simple como el ordenador del robot Erratic requirió varias primitivas (tres hexaedros) para definir aceptablemente su contorno de colisión.

Figura 41. Ordenador del robot Erratic con sus envolventes de colisión marcadas aristas verdes

Llegados a este punto, decidí utilizar 3D Studio Max para definir el perímetro de colisión de los elementos creados con este programa, ya que trabajando con 3D Studio Max, la asignación de este tipo de envolventes resulta muy sencillo debido a dos motivos fundamentalmente. En primer lugar, y dado que el objeto se está creando ya con 3D Studio Max, no hace falta cambiar de aplicación y, además, las envolventes se pueden crear como cualquier otra forma de las que componen el robot. Incluso puede hacerse uso de las mismas primitivas (formas simples) que se utilizaron para crear el propio objeto cuya envolvente queremos definir, las cuales ya tienen la forma y ubicación adecuadas. En segundo lugar, 3D Studio Max permite exportar envolventes de colisión junto con el objeto modelado. Para ello solamente es necesario ir al menú de propiedades de las primitivas que definen tales envolventes y nombrarlas correctamente (ver Tabla 2). Una vez hecho esto, se selecciona la malla del objeto y

las de la envolvente y se exporta el conjunto como archivo ASCII con el nombre que se desee.

Nombre de la envolvente	Primitiva	Ejemplo
MCDBX_Nombre	Caja (Box)	MCDBX_ComputerChassis
MCDSP_Nombre	Esfera (Esphere)	MCDSP_ERAWheel
MCDCY_Nombre	Cilindro (Cylinder)	MCDCY_Columna
MCDCX_Nombre	Malla convexa	MCDCX_Extintor

Tabla 2. Formatos para nombrar envolventes de colisión con compatibilidad para UnrealEd 3

Biblioteca de recursos de Unreal Editor 3

Con todas las partes del robot completamente modeladas en 3D Studio Max y exportadas a formato ASCII, el siguiente paso es crear un archivo que permita almacenarlas de forma que puedan ser accedidas y comprendidas por el motor de Unreal Tournament 2004. Para conseguir tal objetivo, desde el explorador de Static Mesh (malla estática) de UnrealEd, importé el archivo *.ASE* correspondiente a cada parte del robot, definiendo un grupo adecuado para la misma (chasis, cámara, ruedas,…). Después, una vez importados y distribuidos en grupos todos los elementos que forman el robot, guardé un archivo de Static Mesh en formato *.usx*.

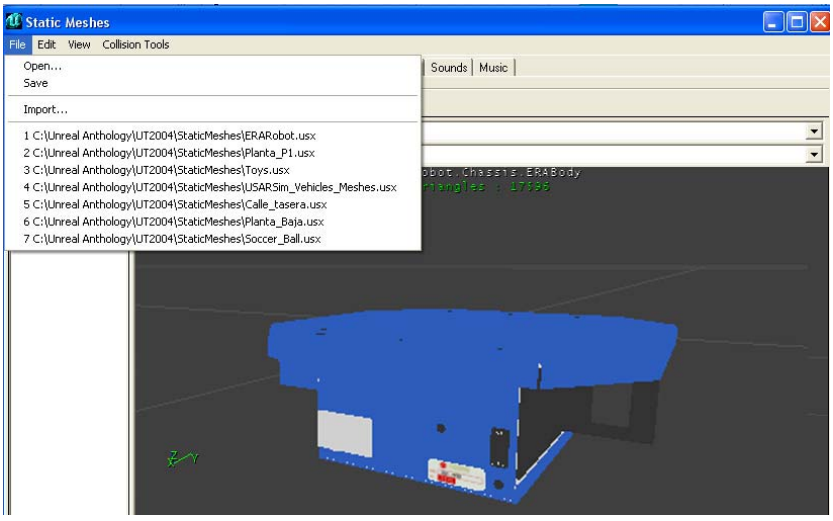

Figura 42. Navegador de mallas estáticas de UnrealEd3 visualizando el chasis del robot Erratic

Siguiendo este procedimiento obtuve el archivo *ERARobot.usx*, el cual contiene toda la información del aspecto y contorno de colisión del robot necesaria para llevar a cabo las simulaciones con USARSim. En la Figura 42 puede observarse el chasis del robot ERA guardado en el grupo "chassis" del archivo antes mencionado.

Definición de clases

Una vez obtenido el modelo 3D del robot y almacenado éste en un archivo **.usx*, el siguiente paso consiste en generar una serie de ficheros **.uc* de UnrealScript. Con estos archivos se han de definir las clases correspondientes a cada parte del robot. En el caso del robot Erratic fue necesario disponer del árbol de clases mostrado en la Tabla 3.

USARBot Classes	USARModels Classes	USARMisPkg Classes
ERA.uc	ERAComputerBody.uc	ERAComputer.uc
ERARangeSensor.uc	ERALasser.uc	ERAComputerBody.uc
ERASonarSensor.uc	ERALTire.uc	
USAR_ERALTire.uc	ERARTire.uc	
USAR_ERARTire.uc	ERASmallTire.uc	
USAR_ERASmallTire.uc		

Tabla 3. Árbol de clases necesario para definir todas las partes que componen el robot Erratic

El archivo de UnrealScript *ERA.uc* define el robot ERA como perteneciente a la clase SkidSteeredRobot utilizada por USARSim para simular los robots cuyo giro se produce mediante un cambio en la velocidad de rotación de las ruedas motrices. Estos robots no poseen ruedas direccionales para cambiar su orientación. Tal es el caso de los robots P2AT y P2DX, ambos incluidos en USARSim. La clase *ERA.uc* contiene toda la información acerca del robot Erratic de Videre Design relativa al comportamiento físico y dinámico del mismo.

USAR_ERALTire.uc define la rueda motriz izquierda del robot ERA como extensión de la clase *BulldogTire.uc* utilizada por USARSim para simular robots como el P2DX o el P2AT. En este archivo se ajustan, además, los parámetros que definen el comportamiento físico de la rueda y se le asocia la malla ERARobot.Wheels.ERALeftWheel, es decir, la malla ERALeftWheel contenida en el grupo Wheels del archivo de malla estática *ERARobot.utx*.

USAR_ERARTire.uc es similar a la clase anterior, pero en este caso se asocia a la rueda motriz derecha la malla estática ERARobot.Wheels.ERAReftWheel. El comportamiento de ambas ruedas, derecha e izquierda es el mismo, pero cambia su posición respecto al eje de coordenadas vertical en 180 grados.

USAR_ERASmallTire.uc define la rueda loca trasera del robot ERA (castwheel) como extensión de la clase *BulldogTire.uc* utilizada por USARSim para simular robots como el P2DX. En este caso, asigné nuevos parámetros físicos y la malla estática ERARobot.Wheels.ERASmallWheel.

ERARangeSensor.uc es similar al archivo *RangeSensor.uc* de USARsim, con el nombre modificado para permitir introducir cambios que sólo afectan a los sensores del robot ERA. *RangeSensor.uc* es una extensión de la clase Sensor utilizada como base para la programación de todos los tipos de sensores de los robots de USARSim. RangeSensor, a su vez, se utiliza como base para sensores del tipo sonar o laser.

ERASonarSensor.uc define el funcionamiento y escala del sonar del robot ERA. En la Figura 43 se observa el aspecto del sensor sonar definido para el robot Erratic.

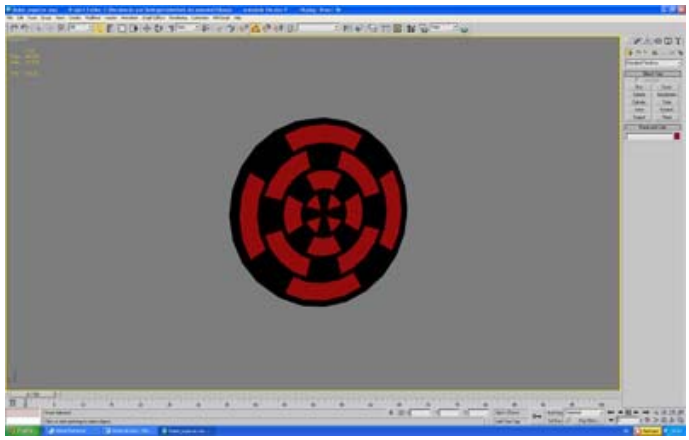

Figura 43. Vista de 3D Studio Max con el modelo de sonar del robot Erratic (ERASonarSensor)

ERALTire.uc define la rueda motriz izquierda del robot ERA como extensión de la clase USAR_ERALTire y ajusta todos sus parámetros físicos de simulación.

ERARTire.uc define la rueda motriz derecha del robot ERA como extensión de la clase USAR_ERARTire y ajusta todos sus parámetros físicos de simulación.

ERASmallTire.uc define la rueda loca del robot ERA como extensión de la clase USAR_ERASmallTire y ajusta todos sus parámetros físicos de simulación.

ERAComputerBody.uc define el cuerpo del ordenador del robot ERA como extensión de la clase KDPart utilizada por USARSim como base para todas las partes del robots que no sean ruedas. En este caso, asigné los parámetros físicos del cuerpo del ordenador y asocié al mismo la malla estática ERARobot.Computer.Computer.

ERALasser.uc define el sensor láser del robot ERA como extensión de la clase RangeScanner utilizada por USARSim como base para simular los sensores laser. En este archivo se asigna la malla ERARobot.Lasser.Lasser al láser del robot ERA y se ajusta su escala para que tenga el tamaño adecuado.

ERAComputerBody.uc define el cuerpo del ordenador del robot ERA como extensión de la clase MisPkgLinkInfo utilizada por USARSim para simular una gran diversidad de accesorios (o partes de los mismos) que pueda llevar montados el robot. En este caso, asigné a la clase accesorio ERAComputerBody el cuerpo del ordenador detallado por la clase USARModels de igual nombre.

ERAComputer.uc declara el ordenador del robot ERA como accesorio. Más adelante, en el archivo *USARMisPkg.ini* se especifica que ERAComputer es un accesorio compuesto por un solo elemento llamado ERAComputerBody.

Archivos de configuración

Una vez todas estas clases definidas y guardadas en su carpeta correspondiente, las compilé ejecutando *make.bat* en cada una de las citadas carpetas. De este modo, quedaron definidas todas las partes del robot tanto en aspecto físico como en cinemática y funcionamiento. Pero, para que el robot pueda ser simulado como tal, es necesario indicarle a USARSim cómo se interconectan entre sí todos sus elementos. Con este fin, introduje modificaciones en los archivos de configuración *USARBot.ini* y *USARMisPkg.ini* de la carpeta System de Unreal Tournament 2004.

La dinámica de los ensayos con robots suele requerir el uso de una serie de dispositivos, en su mayoría opcionales, que deben ir montados en aquéllos. Es decir, cada experimento requiere que el robot incorpore una serie de accesorios tales como

una cámara, un brazo robot, una pinza, etc. Estos dispositivos opcionales se incluyen en USARSim dentro del grupo USARMisPkg.

Como he comentado anteriormente en este capítulo, la carpeta USARMisPkg contiene otra con el nombre "classes" donde se definen todas las clases para los accesorios de los robots.

Por otro lado, la carpeta System contiene el archivo *USARMisPkg.ini*, en el cual se indica cuántas partes componen cada accesorio y cómo se interconectan entre sí. Con el fin de que el ordenador del robot ERA pudiera ser eliminado del robot según conveniencia, se añadí las siguientes líneas en el archivo *USARMisPkg.ini*:

```
;-----------------------------------------------------------------------------------------------------------
; Computer mission package used for the ERA
;-----------------------------------------------------------------------------------------------------------
[USARMisPkg.ERAComputer]
Links = (LinkNumber = 1, LinkClass = Class'USARMisPkg.ERAComputerBody', DrawScale3D =
(X=1.0, Y=1.0, Z=1.0), ParentLinkNumber = -1, SelfMount = "A")
;-----------------------------------------------------------------------------------------------------------
; Computer Links used for the ERA
;-----------------------------------------------------------------------------------------------------------
[USARMisPkg.ERAComputerBody]
MountPoints = (Name = "A", JointType = "Revolute", Location = (X= 0.0, Y=0.0, Z=0.0),
Orientation=(X = 1.5707963267948966192313216916398, Y = 0, Z = 0))
MountPoints=(Name="B",JointType= "Revolute", Location = (X= -0.0, Y = 0.0, Z= -0.0),
Orientation=(X=1.5707963267948966192313216916398,Y=0,Z=0))
MaxSpeed=0.1745
MaxTorque=20
MinRange=0.0
MaxRange=3.14159
;-----------------------------------------------------------------------------------------------------------
```

De esta manera, el ordenador del robot ERA (ERAComputer) quedó definido como una parte opcional del robot compuesta por un único cuerpo (ERAComputerBody), el cual puede girar alrededor de su eje z.

Una vez declaradas todas las partes del robot ERA, sólo queda añadirlo a la lista de aquellos disponibles para ser simulados con USARSim. Este objetivo requiere

modificar el archivo *USARBot.ini* ubicado en la carpeta System, insertando un bloque de líneas que contengan, como mínimo, la siguiente información:

1. Peso del robot.
2. Carga máxima del robot.
3. Partes que componen el robot, incluyendo los accesorios si los hubiera.
4. Localización de los puntos de unión entre diferentes partes.
5. Tipo de articulación que restringe el movimiento relativo entre elementos adyacentes.

Con tal objetivo, introduje las modificaciones pertinentes en el archivo *USARBot.ini*, quedando el robot Erratic en condiciones de ser utilizado para realizar simulaciones con USARSim.

4.4.2. Creación del edificio TI virtual y su entorno

Tal como ya he indicado anteriormente, el entorno para realizar los ensayos simulados corresponde a los laboratorios de investigación y alrededores del Departamento de Ingeniería y Ciencia de los Computadores de la Universidad Jaume I. Esto abarca buena parte de la planta baja y el primer piso del edificio TI de esta universidad, por lo que decidí trasladar al plano virtual el edificio TI entero con sus cuatro plantas y su entorno inmediato para facilitar posibles ensayos en el exterior. En la Figura 44 se puede observar una vista del edificio TI centrada en su fachada posterior.

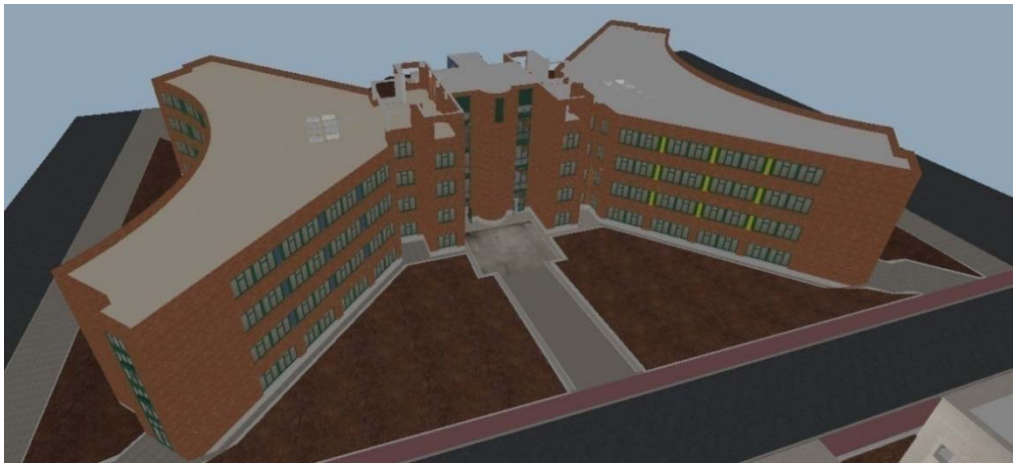

Figura 44. Vista global de un mundo virtual que contiene el edificio TI y su entorno inmediato

Con el objetivo de crear el edificio virtual seguí un procedimiento similar al utilizado para el robot ERA. En cambio, existen ciertas diferencias que es importante destacar.

Planos y medidas

Para obtener el modelo virtual del edificio TI dispuse de los planos del mismo en formato **.dwg* de AutoCAD (ver Figura 45). Aunque esto facilitó mucho las cosas, fue necesario realizar muchas mediciones, sobretodo en cuanto a alturas de ventanas y algunos elementos que no se detallan en los planos. La mayoría de estos elementos forman parte del mobiliario del interior del edificio como por ejemplo: tablones de anuncios, papeleras, máquinas expendedoras, bancos, carteles informativos, planos de evacuación, etc. Otros, en cambio, podrían incluirse como partes de instalaciones del edificio como pulsadores variados (luz, ascensor, alarma contra-incendios), sensores (detectores de humo o presencia), lámparas, bocas de incendio, extintores, etc.

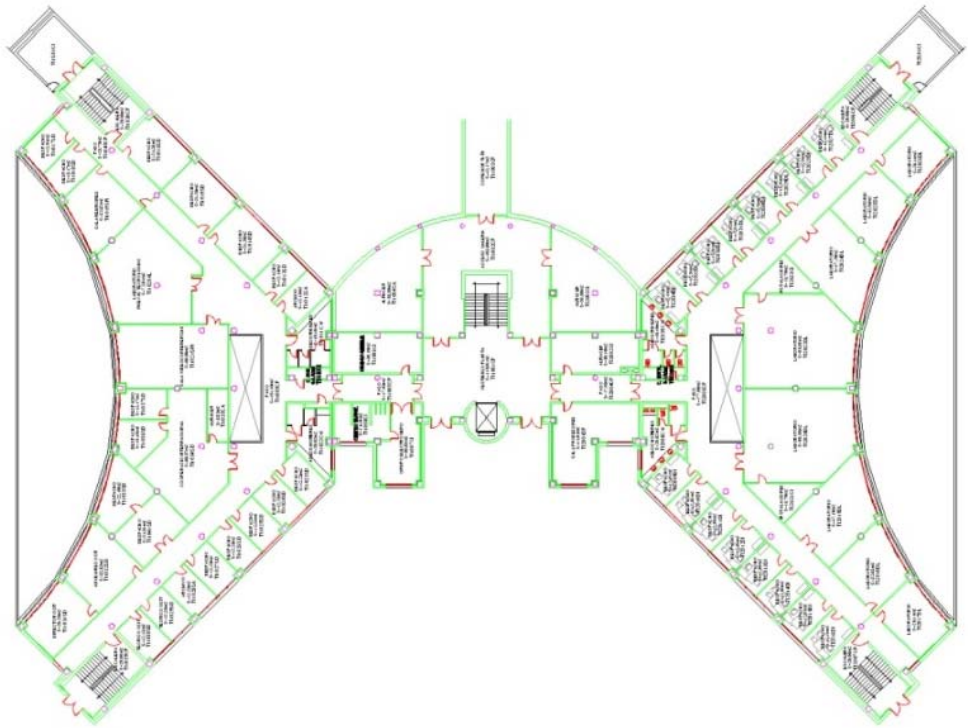

Figura 45. Plano de AutoCAD correspondiente a la primera planta del edificio TI de la UJI

Fotografías

La amplia diversidad de los materiales y colores que componen los elementos que forman el edificio TI y su entorno me obligó a realizar muchas fotografías para generar las texturas necesarias. Así, tomé fotos digitales de las paredes, techos, suelos, puertas, aceras, cemento, asfalto, carteles, lámparas, barandillas, columnas, rendijas de ventilación, sensores, etc.

Las fotografías las traté con Adobe Photoshop CS3 hasta obtener las texturas que darían apariencia realista a los materiales, de manera similar a la llevada a cabo para el robot ERA.

Edificio TI y entorno inmediato. Modelos estáticos

Con las dimensiones y las texturas del edificio y su entorno cercano empiecé la fase de desarrollo del modelo virtual utilizando 3D Studio Max.

La construcción del edificio TI virtual llevó muchas horas trabajo debido a la gran cantidad de elementos que lo componen. La diversidad de elementos que forman parte del edifico y de sus alrededores hace necesaria una explicación diferenciada del procedimiento seguido para su correcta introducción en el simulador.

Paredes

Como elementos principales de la estructura del edificio se encuentran las paredes, el forjado, el pavimento y el techo de cada planta. Estos elementos los creé a partir de los planos de AutoCAD del edificio. Así, desde 3D Studio Max importé los planos de cada planta, los superpuse a la distancia entre plantas y construí cada pared de manera que su forma fuera lo más simple posible (ver Figura 46). Es decir, intenté modelar cada pared a partir de un rectángulo extruido, con lo cual su aspecto óptimo sería el de una caja estrecha y muy alta. Esta manera de proceder conlleva una serie de ventajas importantes y algunos inconvenientes menores que es interesante resaltar.

Por una parte, tratar con paredes de forma hexaédrica (caja o box) supone mayor facilidad para aplicarle los mapas de texturas y un contorno de colisión. Los mapas de texturas permiten adaptarse a determinadas formas geométricas simples como un hexaedro, cilindro, esfera o un plano. Entre las envolventes de colisión exportables a

UnrealEd, las que mayor rendimiento dan son aquellas basadas en combinaciones de figuras geométricas como las anteriores. Además, si exportamos un único hexaedro como pared, existe la ventaja adicional que el propio UnrealEd permite asociarle automáticamente una caja como contorno de colisiones. Aunque esto último no siempre funciona como se desea, dependiendo de la orientación de la pared.

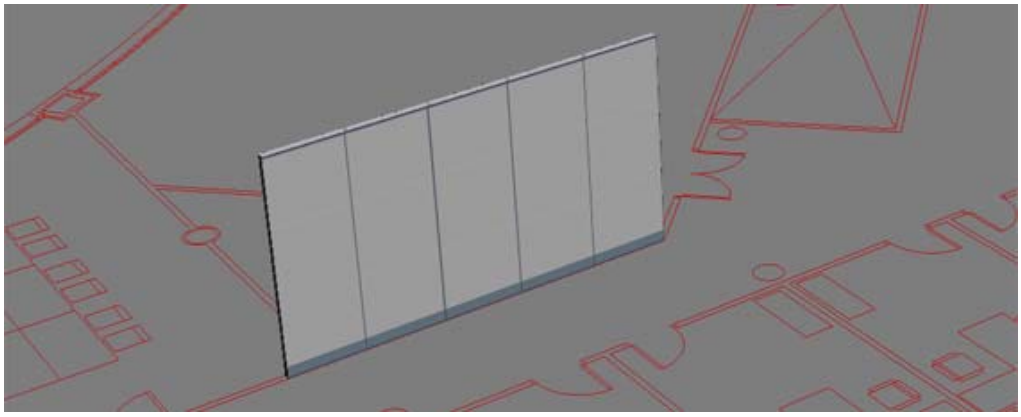

Figura 46. Pared levantada en 3D Studio Max a partir de un plano en AutoCAD del edificio TI

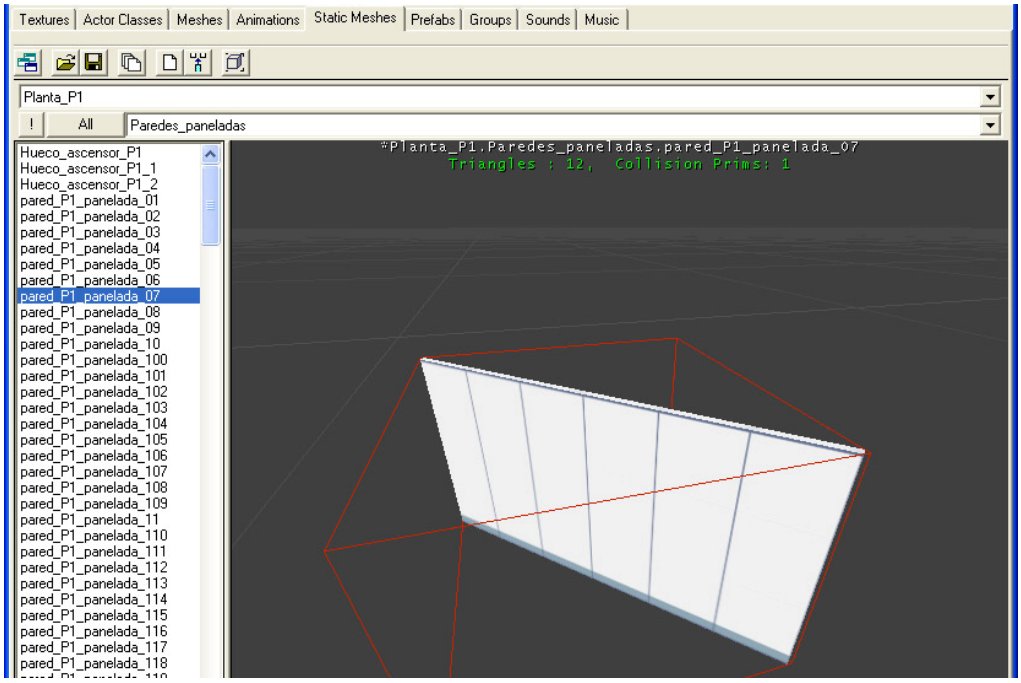

Figura 47. StaticMesh Browser de UnrealEd 3 mostrando una pared importada de 3D Studio max

En la Figura 47 puede observarse una pared del edificio TI tal como se muestra en el navegador de mallas estáticas de UnrealEd 3 después de ser importada en formato ASCII desde 3D Studio Max. Se trata de la pared número 7 perteneciente al primer piso del edificio TI.

Por otra parte, dividir todas paredes en hexaedros supone multiplicar un número de objetos a exportar ya de por sí muy elevado. Sólo hay que ver el plano de una planta para darse cuenta que el número de paredes supera las 200.

Pavimentos, techos y forjados

A partir de los planos de cada planta edificio, construí tanto los forjados como los pavimentos y los techos. En general, esta tarea resultó más sencilla que la introducción de las paredes, pues el número de elementos por planta es mucho menor.

Básicamente, para obtener la formas 3D de estos elementos, redibujé el contorno exterior de cada planta, tracé las líneas que definen los huecos entre diferentes alturas y proyecté la línea resultante en el eje Z de coordenadas, formando en unos casos un forjado y en otros un revestimiento cerámico (ver Figura 48).

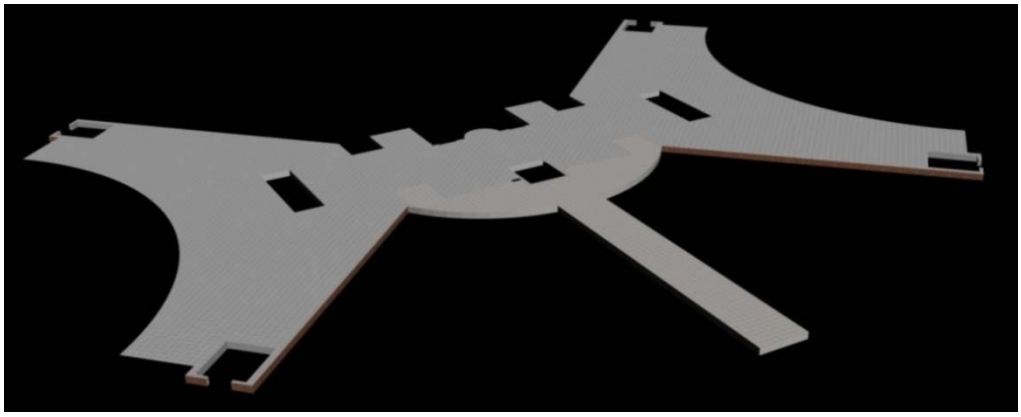

Figura 48. Forma básica para crear un forjado o pavimento del edificio TI en 3D Studio Max

En cuanto a los techos, éstos los modelé a partir de las fotografías tomadas en su día, respetando tanto su estructura de paneles como las lámparas fluorescentes, difusores y rejillas de aire acondicionado para conseguir un aspecto realista. La Figura 49 muestra un techo suspendido registrable muy usual en los despachos del edificio TI de la UJI.

Figura 49. Techo suspendido registrable de un despacho del edificio TI con rejillas y fluorescentes

Columnas

Las columnas cilíndricas que forman parte de la estructura del edificio las modelé aproximándolas a cilindros con un número de caras suficiente para que su aspecto resultase adecuado pero sin llegar a representar una carga gráfica excesiva.

Ventanas

En el Edificio TI hay varios tipos de ventanas que introduje de manera diferenciada para agilizar su exportación a UnrealEd. En general, las ventanas se encuentran en lugares donde no van a producirse choques o contactos directos con el robot o, como mínimo, si estos contactos se materializan, también habrá colisión simultánea con el muro que sostiene dicha ventana. Esto supone una ventaja considerable a la hora de tratar este tipo de objetos, ya que no requieren que se les asocie envolvente de colisión alguna.

Con las ventanas exteriores de los laboratorios, seminarios y despachos seguí un procedimiento distinto al utilizado para obtener el resto de ventanas. Esto es debido a la presencia de persianas (ver Figura 50 izquierda), las cuales modelé como un plano con una textura que imita estos mecanismos. Esta manera de proceder supone exportar a formato ASCII una a una todas las ventanas a causa de dificultades con los mapas UVW que determinan la colocación de las texturas en los objetos. La explicación es sencilla: todos los objetos cuya textura sea un color más o menos uniforme (como la textura verde de la carpintería de aluminio de una ventana, ver

Figura 50 derecha) pueden ser exportados como un conjunto, de manera que al ser importados desde UnrealEd continúan conservando su textura aparentemente intacta; en cambio, esto no sucede de igual manera cuando la textura, a pesar de ser la misma para todos los objetos exportados, no es uniforme.

De lo descrito en el párrafo anterior se deriva que las ventanas dotadas únicamente de marco, bisagras, manecilla y cristal resultan más sencillas de exportar a formato ASCII porque en muchos casos se pueden incluir varias en una sola malla y tratarlas en grupo, lo que reduce notablemente el número de objetos a tratar.

Figura 50. A la izquierda una ventana con lamas y a la derecha una ventana del hall de la segunda planta del edificio TI de la UJI

Considero muy importante indicar que los objetos exportados desde 3D Studio Max a UnrealEd sin envolvente de colisión presentan ciertas peculiaridades durante una simulación: por una parte, no tienen capacidad para impactar con el robot objeto de la simulación, por lo que son atravesados por el mismo sin interacción alguna; por otra parte, sí son detectados por los sensores que incorpora el robot y tienen capacidad para colisionar con cualquier otro ente móvil, salvo éste. Esto permite ahorrarnos las envolventes de colisión para las ventanas y demás objetos no alcanzables por el robot y, por tanto, poder exportarlos en grupo.

Puertas en general

A pesar de su aparente sencillez, las puertas constituyen elementos difíciles de tratar en un mundo virtual. Es decir, su forma suele ser simple (parecida a un hexaedro) al igual que sus texturas, pero su posición y posibilidades de movimiento derivan en la necesidad de tomar ciertas decisiones previas a su colocación en el mundo virtual.

En el edificio TI he diferenciado 3 tipos de puertas: las inmóviles o estáticas, las que pueden ser empujadas por el robot y las que se abren automáticamente cuando el robot se acerca. Las primeras se pueden procesar como si se tratara de paredes. Las segundas y las terceras, en cambio, deben considerarse objetos móviles con las dificultades que ello conlleva.

Las puertas de los pasillos, laboratorios y seminarios del edificio TI (ver Figura 51) presentan un cristal en la parte central de la hoja. Si se considera una de estas puertas como fija (inmóvil), puede utilizarse la siguiente simplificación: modelar la hoja sin el cristal, la manecilla, la cerradura y las bisagras como elementos independientes que pueden exportarse en grupos homogéneos a UnrealEd. Así, cada elemento tiene una textura y una envolvente de colisión simple y, además, se puede crear el cristal desde UnrealEd como un elemento independiente e inmóvil. En cambio, si una puerta con cristal debe ser móvil, se ha de modelar todo el conjunto como una malla con todas sus texturas debidamente colocadas y con sus envolventes de colisión; además, UnrealEd sólo permite importar objetos con texturas basadas en imágenes en formato TARGA *.tga y sin datos de opacidad o reflexión.

Figura 51. A la izquierda una puerta de un pasillo y a la derecha una puerta de un laboratorio

De lo explicado hasta ahora se desprenden varias preguntas: ¿Qué ocurre con los cristales o materiales no opacos? ¿Se pueden exportar desde 3D Studio Max o cualquier otra aplicación de diseño 3D? Pues bien, para exportar con éxito objetos con partes de cristal siguí el siguiente procedimiento: tomé el nombre de una textura cristal de la biblioteca de UnrealEd y, desde 3D Studio Max, asigné una textura cualquiera con dicho nombre a la parte transparente de la puerta en cuestión; cuando importé esta puerta desde UnrealEd, lo hice con la textura transparente seleccionada en el navegador de texturas para provocar que UnrealEd asigne por defecto esta textura si no encuentra la que requiere la superficie que se está importando. De este

modo pueden obtenerse objetos móviles con partes transparentes o semiopacas. La Figura 52 muestra algunas de las posibilidades que ofrece UnrealEd en cuanto a texturas para simular cristales.

Figura 52. Texture browser de Unreal Editor mostrando algunas texturas para simular cristal

Una vez importadas en UnrealEd, las puertas móviles deben tratarse de forma muy diferente a las fijas. Puesto que hay dos tipos de puertas móviles y son muy diferentes entre sí, las describo por separado.

Puertas empujables

Las puertas que sólo se mueven cuando un robot las empuja deben introducirse en el mapa en calidad de Actor en lugar de StaticMesh (como es el caso de las paredes, y otros elementos sin movimiento). Hecho esto, el siguiente paso consiste en ajustar los parámetros que describen la cinemática de cada puerta de este tipo presente en el mapa. Esto se realiza accediendo a las KActor Properties (propiedades para el motor de física Karma), donde las variables más importantes son:

KarmaParams

bHighDetailOnly: Algunos objetos introducidos en calidad de Actor solo usarán Karma si este parámetro toma el valor True. Además, para evitar que las máquinas lentas lo cambien a False intentando

incrementar su rendimiento en la simulación, es necesario cambiar en el archivo default.ini las siguientes líneas

;----------
[Engine.LevelInfo]
PhysicsDetailLevel=PDL_Medium
;----------

por estas otras

;----------
[Engine.LevelInfo]
PhysicsDetailLevel=PDL_High
;----------

KActorGravScale: Factor que define en qué grado se ve un objeto afectado por la gravedad durante su movimiento.

KAngularDamping: Factor que define en qué grado se atenúa el movimiento de giro de un objeto debido a fricciones con el aire.

KBuoyancy: Factor que define el grado de elasticidad de los choques de un objeto. Cuanto mayor es este número, más rebota el objeto al colisionar con otros.

KLinearDamping: Factor que define en qué grado se atenúa el movimiento de avance rectilíneo de un objeto debido a fricciones con el aire.

KMass: Factor que define en qué grado afecta la masa de un objeto a su cinética.

KarmaParamsCollision

KFriction: Parámetro que define el coeficiente de rozamiento entre el Actor y el suelo u otras superficies.

KRestitution: Factor que define en qué grado un objeto es capaz de absorber un golpe. El valor 0.0 hace que el comportamiento del objeto al recibir un golpe sea parecido al de un bloque de piedra.

Adicionalmente, es importante saber que si un objeto sin envolvente de colisión debe chocar contra otros, es necesario dar el valor True al parámetro UseSimpleKarmaCollision desde el StaticMesh Browser, de lo contrario, el programa buscará tal envolvente sin encontrarla y el choque no se producirá. Cuando se define

una envolvente de esta manera, existe la posibilidad de ajustarla a formas geométricas simples como una caja o una línea ajustando el parámetro UseSimpleBoxCollision o UseSimplelinCollision como verdadero o falso.

A continuación, es necesario definir las restricciones en el movimiento de las puertas. En el edificio TI no existen puertas correderas, sólo hay puertas que giran respecto a un eje. Por tanto, para cada una de éstas puertas definí un eje de giro como el mostrado en color naranja en la parte derecha de la Figura 51. En UnrealEd los ejes de giro como el mencionado reciben el nombre de KHinge y se añaden al mapa seleccionando desde el Actor Classes Browser la opción KHinge del grupo Kconstraint dentro del directorio KActor.

Con el eje en su lugar, procedí a asignarle la hoja de una puerta como objeto cuyas posibilidades de movimiento se verían restringidas por aquél. Para ello, accedí a las propiedades del eje, configurando el parámetro KConstraintActor1 de manera que señalase la hoja de la puerta en cuestión tal como se muestra en la Figura 53.

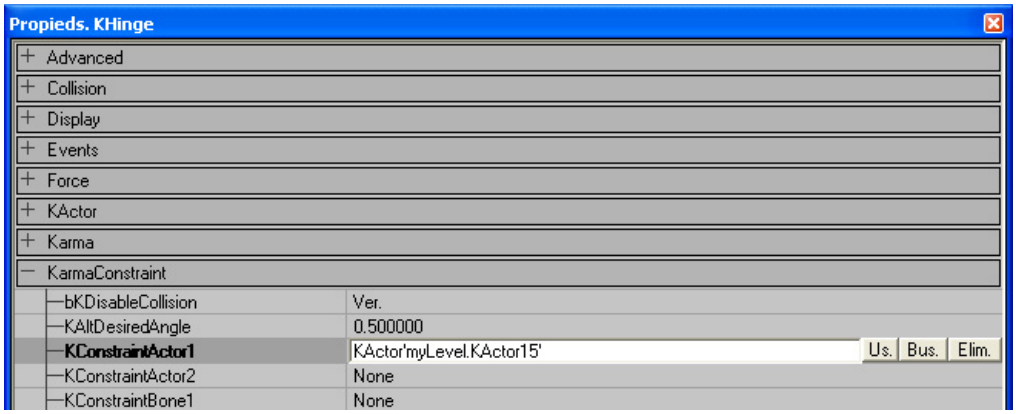

Figura 53. Parámetros para el eje de giro (KHinge) de la hoja de una puerta en Unreal Editor

Puertas automáticas

Para conseguir puertas que se abren automáticamente al acercarse un robot seguí un procedimiento totalmente distinto al anterior. En primer lugar, introduje un elemento móvil (llamado "Mover" en UnrealEd 3) en el mapa del edificio TI y le asigné la malla de la hoja móvil de la puerta en su parámetro StaticMesh (ver Figura 54). A continuación, es necesario definir si el Mover realiza un movimiento lineal (puerta corrediza) o circular (puerta abatible) desde el parámetro KeyNum del Grupo Mover de las propiedades del Mover. Así, partiendo de la puerta en posición de cerrada

(KeyNum 0), seleccioné KeyNum 1 y coloqué la hoja de la puerta en posición totalmente abierta, de manera que con sólo dos posiciones (NumKeys 2) quedó determinada su trayectoria. En la Figura 55 se pueden ver cómo quedaron definidos estos parámetros de la puerta móvil.

Mesh	None
OverlayMaterial	None
OverlayTimer	0.000000
⊞ PrePivot	(X=-17.949999,Y=-1.000000,Z=0.000000)
ScaleGlow	0.600000
⊞ Skins	...
StaticMesh	StaticMesh'Planta_P1.Puertas.Puerta_P1_lab_diario' Elim. Us. ...
Style	STY_Normal
Texture	Texture'Engine.S_Actor'
UV2Mode	UVM_MacroTexture
UV2Texture	None

Figura 54. Asignación de una malla (Puerta_P1_lab_diario) como hoja de puerta automática

Mover	
AntiPortalTag	None
bDamageTriggered	Falso
bDynamicLightMover	Falso
bOscillatingLoop	Falso
BrushRaytraceKey	0
bSlave	Falso
bToggleDirection	Ver.
bTriggerOnceOnly	Falso
BumpEvent	None
BumpType	BT_PlayerBump
bUseShortestRotation	Falso
bUseTriggered	Falso
DamageThreshold	0.000000
DelayTime	0.000000
EncroachDamage	0
KeyNum	1
MoverEncroachType	ME_IgnoreWhenEncroach
MoverGlideType	MV_GlideByTime
MoveTime	1.000000
NumKeys	2
OtherTime	0.000000
PlayerBumpEvent	None
StayOpenTime	4.000000
WorldRaytraceKey	0

Figura 55. Grupo Mover de los parámetros utilizados para configurar el movimiento de una puerta automática

Para determinar el comportamiento de las puertas automáticas también fue necesario ajustar diversos parámetros de las propiedades del Mover tal como se muestra en la Figura 56, la Figura 57 y la Figura 58.

Movement	
AttachTag	None
bBounce	Falso
bFixedRotationDir	Falso
bHardAttach	Falso
bIgnoreEncroachers	Falso
bIgnoreTerminalVelocity	Falso
bOrientToVelocity	Falso
bRotateToDesired	Falso
Buoyancy	0.000000
DesiredRotation	(Pitch=0,Yaw=0,Roll=0)
Location	(X=-8487.000000,Y=-10192.000000,Z=1275.000000)
Mass	100.000000
Physics	PHYS_MovingBrush
Rotation	(Pitch=0,Yaw=-21504,Roll=0)
RotationRate	(Pitch=0,Yaw=0,Roll=0)
Velocity	(X=0.000000,Y=0.000000,Z=0.000000)

Figura 56. Grupo de parámetros Movement de un Mover. Estos parámetros definen el comportamiento físico del Mover

Events	
Event	None
ExcludeTag	...
Tag	Puerta_diario

Figura 57. Grupo de parámetros Events de un Mover. Estos parámetros definen posibles eventos que pueden activar un movimiento

MoverSounds	
ClosedSound	None
ClosingSound	None
LoopSound	None
MoveAmbientSound	None
OpenedSound	None
OpeningSound	None
Object	
Group	None
InitialState	TriggerOpenTimed
Name	Mover1

Figura 58. Grupos de parámetros MoverSounds y Object de un Mover. Estos parámetros definen los sonidos que emite un Mover durante sus movimientos

Adicionalmente, programé algunas puertas para que un robot las pudiera abrir accionando un pulsador (ver Figura 59). Para ello añadí unos interruptores y definí convenientemente el parámetro Event de la puerta en cuestión para que apuntara al nombre del interruptor. Cuando un robot se acerca a escasos centímetros del pulsador, éste actúa activando la apertura o cierre de la puerta. De esta manera el usuario gana control sobre la configuración del entorno de simulación.

Figura 59. Puerta de laboratorio con apertura mediante pulsador situado en la esquina inferior izquierda de la imagen

Escaleras

Las escaleras las obtuvieron utilizando un método similar al de cualquier pared, pero en este caso sólo tuve en cuenta el contorno de colisión para los primeros escalones que conducen a una planta superior. Esta simplificación está justificada por el hecho que los robots disponibles en el Departamento de Ingeniería y Ciencia de los Computadores de la Universidad Jaume I de Castellón no tienen capacidad para subir o bajar escaleras. Si estas condiciones cambiaran con la necesidad de simular un robot capaz de moverse por las escaleras, sería sencillo corregir esto bien añadiendo un BlockingVolume para cada escalón desde el propio UnrealEd o bien añadiendo un contorno de colisión adecuado desde 3D Studio Max y volviendo a exportar las nuevas escaleras. Esta última opción tiene la ventaja que al ser reimportadas las nuevas escaleras desde UnrealEd aparecerían en su lugar correspondiente de forma automática y con todas sus propiedades idénticas a las que tenían las viejas, salvo que en este caso incorporarían una envolvente de colisión distinta.

Entorno exterior

Una vez terminado el edificio TI virtual, decidí incluir su entorno cercano para conferir a las simulaciones un aspecto agradable, de manera que el edificio no pareciera estar flotando en el espacio exterior. Con tal propósito, modelé el paisaje que rodea el edificio de forma aproximada, pero cuidando que aquello que se observa por las ventanas y puertas acristaladas o abiertas coincidiera con la realidad. De este modo, construí un entorno realista compuesto básicamente por un terreno exterior de tierra, calles, el edificio de la biblioteca de la Universidad Jaume I y un cielo azul. Como se trata de elementos inmóviles, el procedimiento fue similar al seguido para la creación

de las paredes, techos y suelos, pero en este caso no definí envolventes de colisión para simplificar el trabajo y mejorar la velocidad de la simulación. Si con el tiempo fuera necesario simular el robot en el exterior del edificio, estas envolventes de colisión se podrían añadir fácilmente.

Objetos variados

En el interior del edificio se encuentran una serie de elementos de diversa índole que consideré conveniente incluir en el entorno virtual para incrementar el grado de realismo de las simulaciones. Este es el caso de algunas barandillas, partes visibles de la instalación contra incendios, carteles de distintos tipos, máquinas expendedoras, lámparas, etc. El carácter estático de todos estos elementos sugería darles el mismo trato que a las paredes, pudiendo omitirse la creación de envolventes de colisión para los objetos no alcanzables por los robots. La Figura 60 muestra las máquinas expendedoras del hall de la planta baja del edificio TI.

Figura 60. Máquinas expendedoras situadas en el hall de la planta baja del edificio TI de la UJI

4.4.3. Luces

El realismo de un entorno virtual depende en gran medida del tipo de iluminación que se ha utilizado en su creación. Para entender cómo resolví la iluminación en el entorno virtual del edificio TI es necesario saber que la cantidad de luz que refleja la superficie de un objeto proviene de dos fuentes: la luz propia del objeto (si la tiene) y la luz exterior que llega al mismo.

Cuando se introduce un nuevo objeto en el entorno virtual, en ausencia de fuentes de luz, éste queda con todas sus superficies negras sin que se distinga textura o color alguno. Para solucionar tal situación, al principio, introduje luces externas que iluminaran todas las superficies a su alcance intentando conseguir un aspecto natural. El resultado obtenido, en cambio, no fue el esperado. Las superficies cercanas a las fuentes de luz quedaron de color casi blanco, es decir, quemadas o sobreexpuestas, mientras que los objetos lejanos a aquéllas apenas se distinguían en la oscuridad.

El siguiente paso fue multiplicar el número de luces externas, dividiendo su potencia con la esperanza que desaparecieran las zonas sobreexpuestas y las subexpuestas. El resultado de este nuevo intento de iluminar no fue muy diferente al anterior, pues, aunque en menor medida, continuaba existiendo multitud de superficies hiperiluminadas e infrailuminadas. Como conclusión extraje que sólo si el número de luces externas tendía a infinito desaparecería aquel problema, a la vez que emergía uno nuevo. El nuevo problema consistía en la creciente atenuación de sombras a medida que se incrementan las fuentes de luz y, por tanto, el número de ángulos desde los que ésta alcanza un objeto. Si desaparecen las sombras, también lo hace el aspecto realista del entorno virtual. Además, las sombras son absolutamente necesarias para distinguir el punto de contacto de los objetos con el suelo.

Paralelamente a los experimentos con fuentes de luz externas, realicé pruebas de autoiluminación con algunos objetos. De estas pruebas concluí que tal tipo de iluminación sólo produce efecto sobre los colores del objeto autoiluminado, viéndose éstos como colores planos más oscuros o claros, pero sin brillos ni sombras sobre ninguna de sus caras y, por tanto, sin aspecto tridimensional. Esto es así debido a que para obtener tanto superficies que degraden de colores claros a oscuros como sombras proyectadas por objetos opacos, es necesario definir, al menos, un punto desde el cual se proyecta luz sobre nuestro mundo virtual.

De todo lo anterior deduje la necesidad de dotar a cada objeto con cierto grado de luz propia que, por una parte, garantice un mínimo de iluminación en todas sus caras y, por otra, minimice la cantidad de luces exteriores requeridas para conferirle un aspecto natural. De lo contrario, el motor gráfico de Unreal Tournement 2004 requería una iluminación de una complejidad muy alta con resultados poco realistas. En la Figura 61 puede observarse la calidad de la iluminación obtenida de esta manera.

Figura 61. Combinación de luces con objetos autoiliminados. El punto de luz (bombilla) tiene como misión la proyección de sombras para incrementar el efecto tridimensional de la escena

La Figura 62 muestra algunas de las propiedades gráficas de un objeto en Unreal Editor 3. Los parámetros AmbientGlow y ScaleGlow están entre los más importantes a la hora de iluminar con este software. El primero se refiere a la cantidad de luz que incide en todas las caras uno objeto, mientras que el segundo expresa cuánto se ve afectado dicho objeto al ser alcanzado por una fuente de luz cercana. El parámetro AmbientGlow lo ajusté a valor 70 para conseguir la cantidad de autoiluminación mínima citada anteriormente. El parámetro ScaleGlow depende mucho de la luz exterior que se disponga y, en general, valores entre 0.4 y 0.7 han dado buen resultado tanto para el interior del edificio TI como para su entorno exterior.

Display	
AmbientGlow	70
AntiPortal	None
bAcceptsProjectors	Ver.
bAlwaysFaceCamera	Falso
bDeferRendering	Ver.
bDisableSorting	Falso
bShadowCast	Ver.
bStaticLighting	Ver.
bUnlit	Falso
bUseDynamicLights	Ver.
bUseLightingFromBase	Falso
ScaleGlow	0.600000

Figura 62. Propiedades gráficas de un objeto en Unreal Editor 3. El parámetro AmbientGlow define el nivel de autoiluminación

4.4.4. Objetos inanimados móviles

Son numerosos los experimentos en los que un robot debe reconocer un objeto, perseguirlo, agarrarlo o incluso manipularlo con algún propósito predefinido. Para satisfacer estas necesidades modelé algunos objetos inanimados pero móviles, es decir, con física. Se trata de artilugios con geometrías y comportamientos físicos simples que pueden servir como base para el desarrollo de otros más complejos o más útiles, en función de las necesidades de cada ensayo. De este modo, introduje en el entorno una pelota de fútbol, un caramelo, un saco de boxeo y un dispensador de objetos accionable por robots. Estos objetos los detallan a continuación de manera individualizada debido a sus diferencias en cuanto a propiedades y comportamiento físico.

Pelota

Para empezar con algo simple, modelé una pelota de fútbol, de modo que los robots de las simulaciones tuvieran algún objeto que pudieran empujar de un lado a otro. Con tal propósito tomé el diámetro de una pelota real y su esquema de colores, de modo que su aspecto resultase lo más real posible (ver Figura 63). Después, desde 3D Studio Max di forma y color a la misma, sin olvidar asociarle una envolvente de colisión esférica. Hecho esto, exporté el conjunto en formato ASCII, generando el archivo *soccer_ball.ase* necesario para incorporar la pelota a la biblioteca de recursos de Unreal Editor 3.

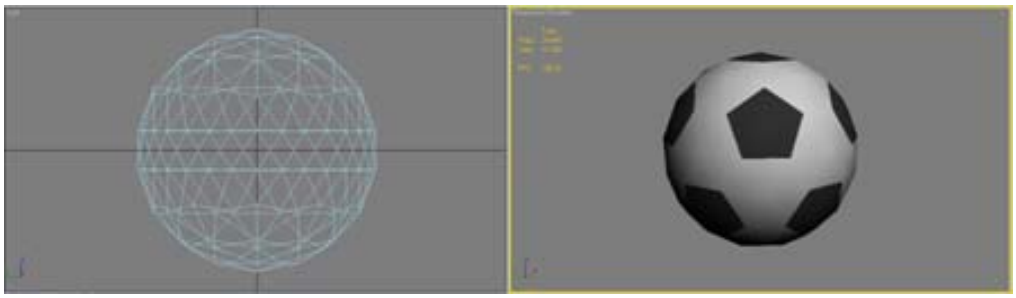

Figura 63. Pelota de fútbol en 3D Studio Max con textura realista. La textura permite saber si la pelota está girando o simplemente desliza por el suelo

El siguiente paso en la generación de objetos inanimados móviles fue crear desde UnrealEd un archivo (en este caso *Toys.utx*) en el que guardarlos todos. Después, desde el StaticMesh Browser importé la pelota como "Soccer_Ball" (ver Figura 64).

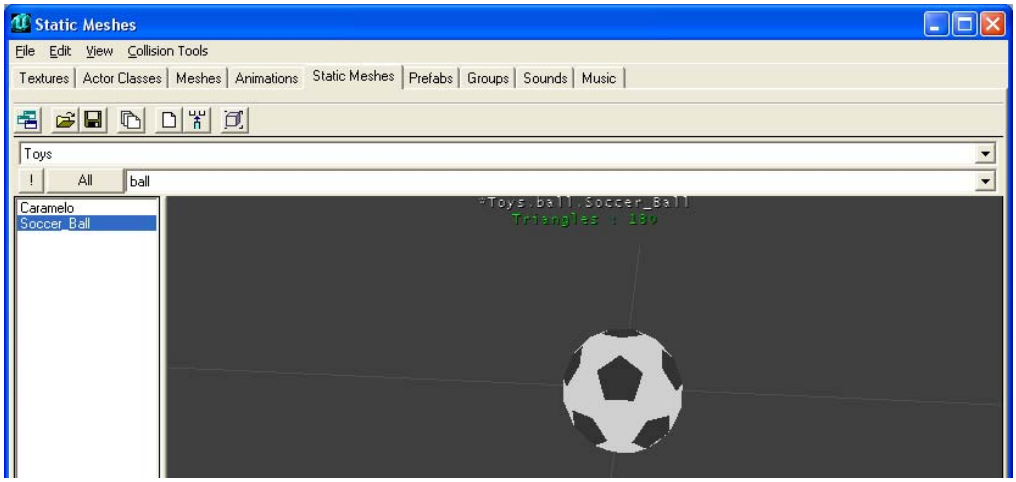

Figura 64. Pelota de fútbol (SoccerBall) mostrada en el StaticMesh Browser de Unreal Editor 3

Por último, inserté la pelota en el laboratorio virtual en calidad de Actor, ya que no se trata de un objeto inamovible. Hecho esto, ajusté los parámetros físicos de la pelota para que su comportamiento en simulación fuese realista (ver Tabla 4). La explicación de los KarmaParams se encuentra en el apartado concerniente a las puertas empujables.

Grupo de parámetros	Parámetro	Valor
KarmaParams	bHighDetailOnly	False
	KActorGravScale	1.0
	KAngularDamping	0.7
	KBuoyancy	0.2
	KLinearDamping	0.7
	KMass	0.2
KarmaParamsCollision	KFriction	0.7
	KRestitution	1.0

Tabla 4. Valores de los parámetros que rigen el comportamiento físico de la pelota con Karma Physics

Caramelo

El siguiente objeto que introduje en orden de simplicidad es un “caramelo”, el cual está compuesto por un cuerpo cilíndrico alargado terminado en dos tapas circulares de diámetro ligeramente superior. En este caso, escogí sus dimensiones de manera que un robot dotado con pinza pudiera agarrarlo en toda su longitud como si se tratase de una lata de refresco (ver Figura 65).

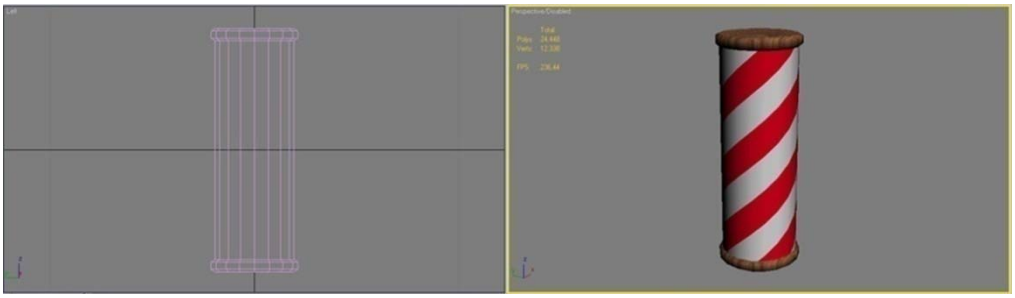

Figura 65. Vista de alzado y perspectiva del objeto caramelo durante su edición en 3D Studio Max

Igual que en el caso de la pelota, elaboré con Photoshop una textura que permite saber si el objeto está girando alrededor de alguno de sus ejes. La textura de bandas blancas y rojas helicoidales cumple a la perfección con este cometido y sirvió para darle su nombre de "caramelo". El tratamiento del caramelo es análogo al de la pelota exceptuando los parámetros que definen su comportamiento físico, los cuales resumo en la Tabla 5.

Grupo de parámetros	Parámetro	Valor
KarmaParams	bHighDetailOnly	False
	KActorGravScale	0.6
	KAngularDamping	0.1
	KBuoyancy	0.8
	KLinearDamping	0.2
	KMass	0.2
KarmaParamsCollision	KFriction	0.6
	KRestitution	1.0

Tabla 5. Valores de los parámetros que rigen el comportamiento físico del caramelo con Karma Physics

Saco de Boxeo

Con el objetivo de explotar las capacidades de simulación de USARSim, decidí crear un saco de boxeo. De esta manera puede simularse el movimiento de un péndulo al ser golpeado por un robot, lo cual introduce restricciones en el movimiento del objeto inanimado móvil, abriendo un gran abanico de posibilidades para futuros ensayos. Puse dos sacos diferentes pesos y comportamientos físicos para poder experimentar con Unreal Editor 3. En la Figura 66 puede observarse el saco de boxeo con su soporte durante su diseño en 3D Studio Max.

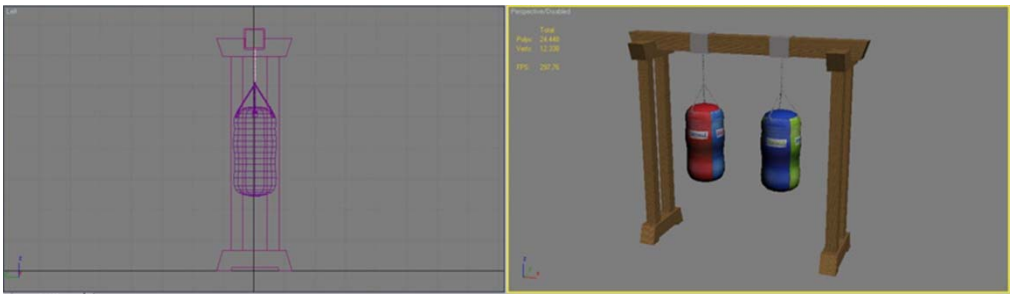

Figura 66. Vista de alzado y perspectiva del saco de boxeo durante su edición con 3D Studio Max

A diferencia de los objetos anteriores, el saco de boxeo no está compuesto por un solo elemento, sino por tres: el saco, las cadenas y el soporte. Sin embargo, su proceso de creación no difiere mucho al de los objetos anteriores. Así, desde Photoshop creé texturas de lona, madera y metal para el saco, el soporte y las cadenas, respectivamente. Después, desarrollé estos tres elementos en un mismo archivo de 3D Studio Max, eligiendo las envolventes de colisión adecuadas para exportar cada elemento por separado generando los archivos `Estructura_Punching.ase`, `Chain.ase`, `Punching_Bag.ase` y `Punching_Bag2.ase`.

La Figura 67 muestra las envolventes de colisión utilizadas para cada parte del saco de boxeo. Obsérvese como las partes que no se prevé que vayan a chocar con un robot no poseen envolvente (travesero superior de la estructura de madera).

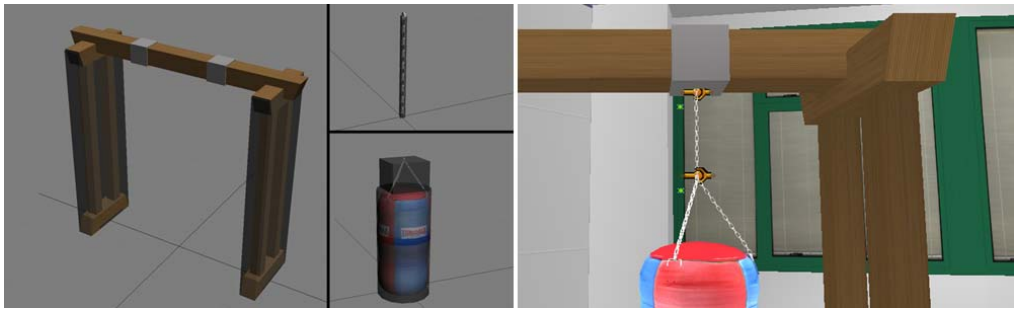

Figura 67. A la izquierda envolventes de colisión de los componentes del saco de boxeo y a la derecha sus articulaciones

Una vez en la biblioteca de recursos de Unreal Editor 3, inserté los objetos en un rincón del laboratorio virtual, quedando a disposición de posibles simulaciones. Mientras que las cadenas y el saco las introduje en calidad de Actor, el soporte lo definí como StaticMesh, dado que se supone inmóvil. Por tanto, sólo hubo que ajustar

los parámetros físicos del tramo de cadena inicial (Chain) y los del conjunto llamado Punching_Bag (ver Tabla 6).

Elemento	Grupo de parámetros	Parámetro	Valor
Chain	**KarmaParams**	bHighDetailOnly	False
		KActorGravScale	1.0
		KAngularDamping	0.2
		KBuoyancy	0.0
		KLinearDamping	0.2
		KMass	0.1
	KarmaParamsCollision	KFriction	0.0
		KRestitution	0.0
Punching_Bag	**KarmaParams**	bHighDetailOnly	False
		KActorGravScale	1.0
		KAngularDamping	0.8
		KBuoyancy	0.0
		KLinearDamping	1.0
		KMass	2.0
	KarmaParamsCollision	KFriction	0.0
		KRestitution	0.0

Tabla 6. Valores de los parámetros que rigen el comportamiento físico de los elementos del saco de boxeo con Karma Physics

Posteriormente, añadí las restricciones de movimiento de las partes móviles del conjunto como son el primer tramo de cadena (Chain) y el saco con el segundo tramo de cadenas (Punching_Bag), tal como se muestra en la Figura 67 a la derecha de la imagen. Estas restricciones las implementé mediante dos articulaciones cónicas llamadas ("KBSJoint" en Unreal Editor 3) de manera análoga al procedimiento seguido con las puertas.

Expendedor de juguetes

Con el objetivo de permitir que un usuario sea capaz de elegir con qué tipo de geometría quiere interactuar y para además introducir mecanismos simples en el simulador diseñé un expendedor de juguetes (ver Figura 68). Éste consiste en una rampa que termina en una superficie horizontal donde se colocan dos objetos, los cuales pueden ser forzados a caer por la pendiente mediante dos manivelas situadas en los laterales de la rampa. Este expendedor también podría servir como rampa o base para desarrollar otros mecanismos para el simulador.

Figura 68. Aspecto en simulación del expendedor de juguetes con el caramelo y la pelota

A la parte fija del mecanismo le di aspecto de madera y una envolvente de colisión compuesta por varios hexaedros, mientras que a las manivelas móviles les conferí aspecto de metal con recubrimiento de goma en sus extremos y una envolvente de colisión formada por dos hexaedros en tales extremos. Con ello conseguí un aspecto sencillo y de uso intuitivo. Para permitir el accionamiento de las manivelas fue necesario añadir dos ejes de giro como los usados para simular las bisagras de las puertas.

4.4.5. Funciones MATLAB

Con todos los elementos necesarios para realizar ensayos virtuales terminados, el siguiente paso fue utilizar el paquete de funciones MATLAB USARSim Toolbox para establecer un interfaz flexible entre el usuario y USARSim. Tales funciones, guardadas en *M-files* de MATLAB, sirvieron como base para controlar el robot virtual Erratic, navegar y almacenar los datos que sus sensores recogieron durante las pruebas del simulador, quedando a discreción de cada usuario implementar funciones específicas para cada ensayo particular.

4.5. Conclusiones

Después de trabajar una gran cantidad de horas con USARSim v.3.3 para UT2004, la impresión general que me dio este simulador fue muy buena. Destaca la alta fidelidad audiovisual y funcional, así como las facilidades que ofrece Unreal Editor 3 para crear escenarios virtuales o introducir nuevas plataformas robot. Como punto en contra, cabe mencionar que iluminar con Unreal Tournament 2004 no resulta sencillo, requiriendo numerosas iteraciones de prueba/error hasta alcanzar un nivel de iluminación adecuado para la escena.

Capítulo 5

SELECCIÓN DE SIMULADORES

En este capítulo reviso las teorías del aprendizaje mediante entornos virtuales y los criterios técnicos de selección de simuladores para determinar cuáles son los más interesantes para conseguir simulaciones didácticas y atractivas. Dada su importancia, las teorías del aprendizaje y los criterios técnicos de selección de simuladores los he complementado con un estudio sobre las características que deben reunir los entornos virtuales para facilitar experiencias inmersivas que mantengan el interés del usuario.

5.1. Revisión de las teorías del aprendizaje con entornos virtuales

Revisando las principales teorías del aprendizaje con entornos virtuales se puede establecer la base para conseguir simulaciones robóticas que favorezcan la motivación y el interés de los usuarios. Los alumnos motivados son entusiastas y disfrutan de lo que están haciendo, ellos se esfuerzan y persisten en sus tareas impulsados por su propia voluntad en lugar de fuerzas externas [35]. Skinner y Belmont [109] señalaron que, si bien los alumnos motivados son fáciles de reconocer, son difíciles de encontrar.

En este apartado primero repaso los conceptos fundamentales de las principales teorías del aprendizaje con un enfoque dirigido hacia los resultados de aprendizaje, después hago una revisión de la literatura acerca de los atributos de los simuladores importantes para el aprendizaje y la motivación, y finalmente expongo las relaciones entre tales atributos y los resultados de aprendizaje. Conocidas las relaciones entre los atributos de los simuladores y los resultados de aprendizaje, el siguiente paso es encontrar qué simuladores facilitan en mayor medida la incorporación de tales atributos.

5.1.1. Teorías del aprendizaje

Los investigadores conceptualizan el aprendizaje como un proceso. Kolb [110] define el aprendizaje como el proceso de transformar la experiencia en conocimiento y que este proceso se compone de cuatro fases (experiencia concreta, observación reflexiva, conceptualización abstracta y experimentación activa). Otros autores se centran en los aspectos conscientes e inconscientes del aprendizaje, argumentando que si bien el aprendizaje en general se produce como resultado de situaciones de la vida real, también puede derivarse de simulaciones o escenarios imaginados. [111]. Para los intereses de esta tesis resultan especialmente importantes los trabajos que relacionan determinados atributos de los video juegos educativos y simuladores con resultados de aprendizaje concretos.

Según Kraiger, Ford y Salas [112] los resultados de aprendizaje se dividen en tres grupos: resultados cognitivos, resultados basados en habilidades y resultados afectivos (ver Figura 69).

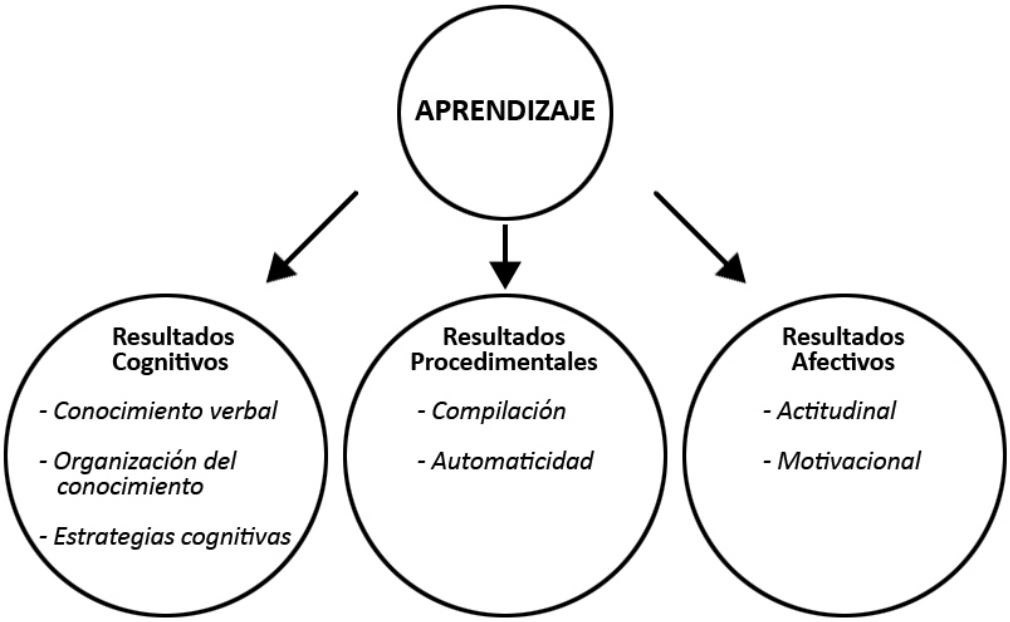

Figura 69. Resultados de aprendizaje. Esquema adaptado del texto de Kraiger, Ford y Salas [112]

Los resultados de aprendizaje cognitivos son aquellos relacionados con la adquisición de conocimientos, la organización de los mismos y la generación de estrategias. Primero, los alumnos deben adquirir conocimiento. Concretamente, los alumnos ganan tres tipos de conocimiento: conocimiento declarativo (conocimiento acerca del

qué), conocimiento procedimental (conocimiento acerca del cómo) y conocimiento estratégico o tácito (conocimiento acerca del cuál, cuándo y por qué). Una vez los alumnos han adquirido el conocimiento deben organizarlo. Esto conlleva la agrupación de fragmentos de información en modelos mentales, los cuales son guardados en la memoria de largo plazo. Finalmente, utilizando los conocimientos previamente adquiridos y organizados, los alumnos son capaces de desarrollar y aplicar estrategias que impliquen tales conocimientos.

Posteriormente al desarrollo de resultados cognitivos, los alumnos progresan hacia la adquisición de habilidades técnicas o motoras. Después de una fase de compilación de habilidades, los alumnos ganan capacidad para automatizar los procedimientos aprendidos mediante comportamientos fluidos y para modificar o adaptar esos procedimientos según lo requieran nuevas situaciones.

Por último, el aprendizaje también da forma a los sentimientos del individuo. Kraiger, Ford y Salas clasifican estos resultados de aprendizaje en dos categorías: aquéllos relacionados con cambios en la actitud del alumno y aquéllos relacionados con cambios en la motivación. Los resultados afectivos (o actitudinales) incluyen individualismo creativo, crecimiento interior, autoconciencia y cambio en los valores del alumno. Los resultados motivacionales incluyen disposición de motivación, autoeficacia y establecimiento de objetivos o metas.

5.1.2. Atributos importantes en juegos serios y simuladores

Cualquier juego diseñado para propósitos educativos debería estar estrechamente ligado a objetivos de aprendizaje [33], de modo que la efectividad educativa de tal juego se podría entender como el grado de coincidencia entre los objetivos de aprendizaje y los atributos del juego. Es decir, para maximizar la eficacia educativa de un video juego hay que dotarlo de aquellos atributos que favorezcan los resultados de aprendizaje buscados. En este sentido, existen numerosos estudios acerca de cuáles son las características principales de los juegos y qué papel tienen desde un punto de vista educativo.

En 2002, Garris, Ahlers y Driskell [35] revisaron la literatura existente deduciendo que numerosos autores estaban utilizando diferentes términos y aproximaciones para referirse a similares características de los juegos. Garris y sus colegas en su estudio concluyeron que, salvando el hecho que las simulaciones pretenden representar la

realidad y los juegos no, simuladores y juegos pueden considerarse similares a la hora de decidir cuáles son las características fundamentales que los convierten en didácticos y atractivos. De hecho, Garris y sus colegas basaron sus conclusiones en estudios y ensayos empíricos llevados a cabo tanto con juegos serios como con simuladores de manera indistinta. Estas características las englobaron en seis categorías o dimensiones: fantasía, reglas/objetivos, estimulación sensorial, desafío, misterio y control.

En 2009 Wilson y otros realizaron un estudio sobre las características de los juegos serios y los resultados de aprendizaje [29]. Tal estudio revisó, entre otros, los trabajos de Garris, Ahlers y Driskell, concluyendo que las características importantes para el aprendizaje y la motivación son principalmente: fantasía, fidelidad, estimulación sensorial, desafío, misterio, evaluación y control. En este estudio se incluye la fidelidad y la evaluación como atributos importantes, separando este último de las reglas/objetivos a diferencia de Garris y sus colegas. Dada la importancia de la fidelidad en el mundo de las simulaciones, he considerado adecuado tomar como base el trabajo de Wilson y sus colegas.

La *fantasía* representa algo que se aparta de la vida real y evoca imágenes mentales de cosas que no existen [35] [36]. La fantasía permite que el usuario interaccione sin miedo de consecuencias propias del mundo real y, además, facilita que el usuario se sienta inmerso en el entorno virtual [39] [35]. Algunas investigaciones sobre fantasía sugieren que los alumnos se sienten más interesados que con las técnicas de clase tradicionales o los juegos no inmersivos, siendo capaces de aprender más rápidamente que cuando usan formatos basados en papel [113] [41] [114].

La *fidelidad* es el grado de similitud física y psicológica entre el juego y el entorno que representa [31]. La fidelidad psicológica se presenta cuando un entorno virtual provoca que sus usuarios experimenten los mismos procesos cognitivos para completar una tarea que si estuvieran realizando tal tarea en el mundo real.

La *estimulación sensorial* incluye estímulos visuales, auditivos, táctiles e incluso olfativos [38] con el fin de distorsionar la percepción del usuario y facilitar la aceptación temporal una realidad alternativa. Estos estímulos pueden utilizarse para favorecer la fantasía o como realimentación para el usuario [115].

El *desafío* está directamente ligado a la dificultad que supone conseguir los objetivos de un juego o una simulación, de manera que la cantidad óptima de desafío es aquélla que genera y mantiene el interés y la motivación del usuario [39]. Algunos estudios demuestran que el desafío está relacionado con la motivación [116]. La motivación se mantiene creando incertidumbre en la consecución de los objetivos.

Existe *misterio* cuando hay un hueco entre la información conocida y la desconocida. El usuario puede saber que cierta información existe, pero no la conoce todavía. La adición de ciertos elementos en un juego puede incrementar el misterio: novedad, complejidad, inconsistencia, sorpresa, información incompleta [36] y la incapacidad para predecir el futuro [35]. El misterio despierta la curiosidad del usuario por completar su información, incrementando su motivación [29]. Malone y Lepper señalaron que la curiosidad es uno de los principales factores que impulsan el aprendizaje [36].

La *evaluación* es la medición del grado en que se alcanzan los objetivos dentro de un juego o simulación y es un atributo fundamental para impulsar la mejora y el progreso del usuario hacia tales objetivos. Esta medición puede lograrse tabulando comparaciones de los resultados obtenidos por diferentes usuarios o proporcionando realimentación durante el avance del juego (puntuación numérica, tiempo invertido, etc.). Esta realimentación informa inmediatamente al usuario acerca de su progreso y de los efectos positivos o negativos de sus decisiones [27]. Para que los usuarios puedan mejorar su rendimiento en el juego o incluso su aprendizaje, es vital que aprecien la conexión entre sus acciones y sus resultados. Esta conexión puede establecerse a través de tres formas de evaluación: evaluación de terminación, evaluación en curso y evaluación del profesor [117].

En el contexto de un videojuego educativo el *control* se refiere a la capacidad de los usuarios de influir sobre su entorno de aprendizaje [40]. Cuando a los estudiantes se les ofrece este control está demostrado que invierten más tiempo en su entorno de aprendizaje y son capaces de elaborar estrategias más complejas que cuando no tienen tal control [41].

5.1.3. Relación entre los atributos de juegos serios y los resultados de aprendizaje

A continuación expongo cuáles son los efectos de estos atributos fantasía, fidelidad, estimulación sensorial, misterio, evaluación y control sobre los resultados de aprendizaje.

Resultados de aprendizaje cognitivos

Respecto a la relación entre los resultados de aprendizaje cognitivos con los atributos de los juegos serios, éstos aumentan conforme lo hace la estimulación sensorial [118], el misterio [119], la evaluación [120] y el control [121].

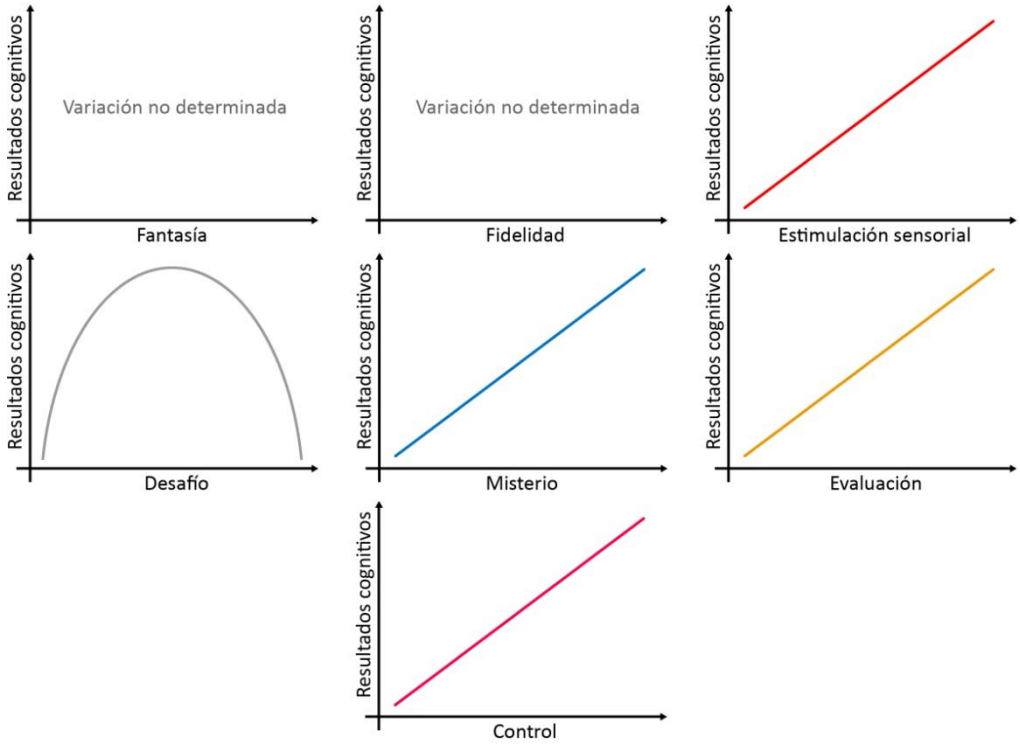

Figura 70. Relación entre los resultados de aprendizaje cognitivos y los atributos de los juegos serios o simuladores

La Figura 70. Relación entre los resultados de aprendizaje cognitivos y los atributos de los juegos serios o simuladores muestra gráficamente la variación de los resultados de aprendizaje cognitivos con los atributos importantes en los juegos serios y simuladores.

La relación entre el nivel de fantasía introducido en un entorno virtual y los resultados de aprendizaje cognitivos derivados no está clara. Muchos autores confirman que los alumnos mejoran sus resultados cognitivos al introducir la fantasía en los juegos serios, pero quizá el aumento de resultados cognitivos se deba al conocido incremento de motivación (resultado de aprendizaje afectivo) que los contenidos fantásticos provocan en los usuarios [39].

Existe un probada relación positiva entre el desafío y los resultados de aprendizaje cognitivos [26]. No obstante, algunos autores proponen que el nivel de desafío debe mantenerse dentro de un intervalo de manera que el juego o la simulación proponga un reto difícil pero alcanzable [29], siendo la incertidumbre en la consecución de los objetivos lo que mueve el interés del usuario [35] [122]. Por lo que la relación entre el desafío y los resultados de aprendizaje cognitivos es una U invertida.

Aunque existen numerosos estudios acerca de la fidelidad en los juegos serios y simuladores [123] [124] [44], parece que no está claro qué nivel de fidelidad es el adecuado para alcanzar los resultados de aprendizaje cognitivos.

Resultados de aprendizaje procedimentales

Existe escasa investigación acerca de la relación entre los resultados de aprendizaje procedimentales con los atributos de los juegos serios. No obstante, tras revisar la literatura existente, en 2009 Wilson y sus colegas [29] propusieron una serie de relaciones, todavía por confirmar empíricamente, las cuales describo a continuación. Respecto a los resultados de aprendizaje procedimentales, éstos se incrementarán en la medida que aumenta el nivel de fantasía. En lo concerniente a fidelidad concluyeron que en la medida que la fidelidad física y psicológica aumentan, el usuario aumentará su aprendizaje procedimental dado que será necesario utilizar similares procesos físicos y cognitivos para ejecutar la tarea real y la simulada. Sin embargo, superado cierto nivel de fidelidad incrementos en éste atributo no supondrán mayores beneficios para el aprendizaje. En lo referente al nivel de control, propusieron la cantidad de control en manos del usuario afectará positivamente a su aprendizaje procedimental.

La Figura 71 muestra gráficamente la variación de los resultados de aprendizaje procedimentales con los atributos importantes en los juegos serios y simuladores.

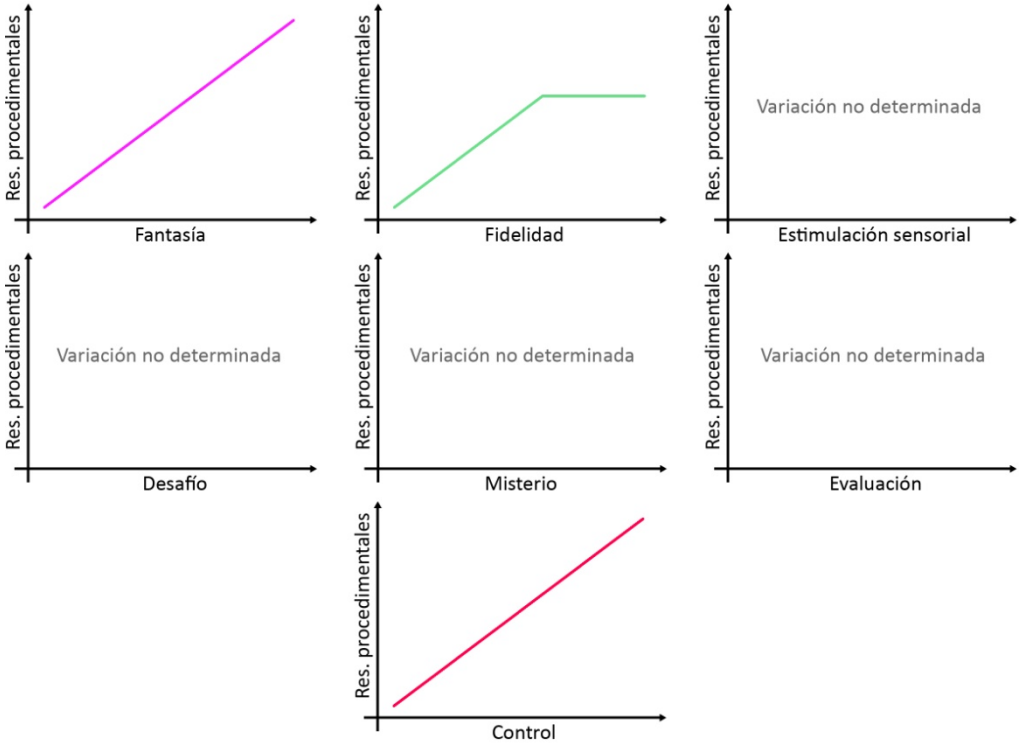

Figura 71. Relación entre los resultados de aprendizaje procedimentales y los atributos de los juegos serios o simuladores

Resultados de aprendizaje afectivos

Respecto a la relación entre los resultados de aprendizaje cognitivos con los atributos de los juegos serios, éstos aumentan conforme lo hace el nivel de fantasía [34] [114], fidelidad [123] [125] [29], estimulación sensorial [34] [122], el misterio [125] [29], la evaluación [34] [122] y el control [34] [123] [126]. En cuanto la relación entre el nivel de desafío y los resultados de aprendizaje afectivos, se espera que tenga forma de U invertida, pues muy poco o demasiado desafío conducirán a un descenso en la motivación [29]. La Figura 72 muestra gráficamente la variación de los resultados de aprendizaje afectivos con los atributos importantes en los juegos serios y simuladores.

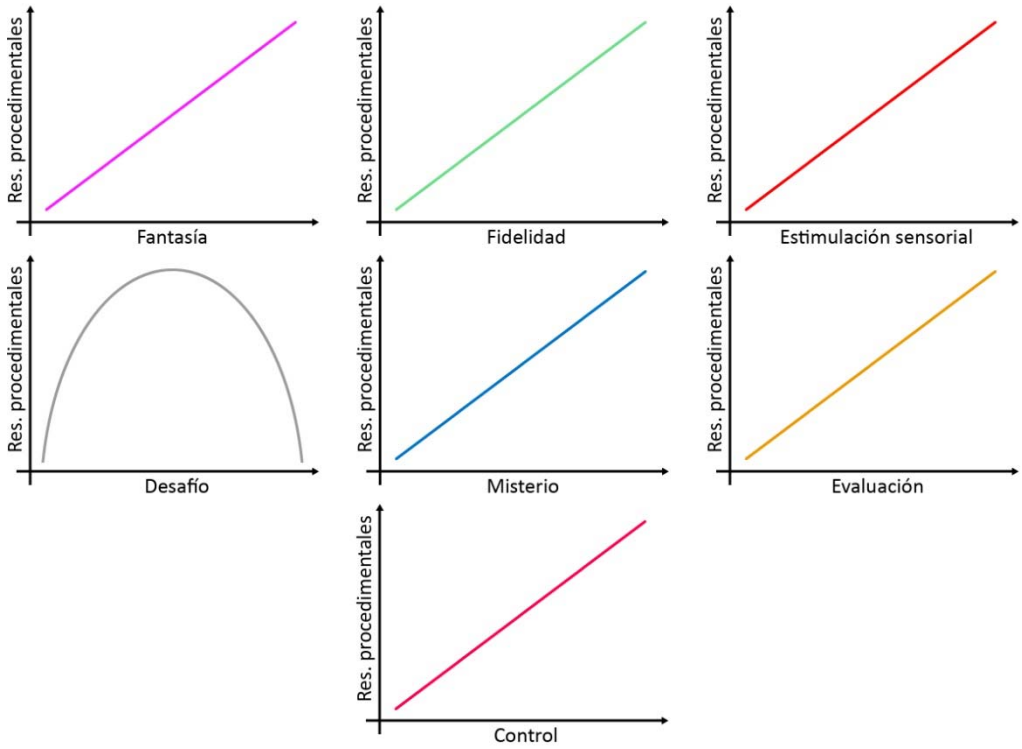

Figura 72. Relación entre los resultados de aprendizaje afectivos y los atributos de los juegos serios o simuladores

5.2. Criterios generales para la selección de simuladores

En este apartado reviso los criterios generales para la selección de simuladores robóticos y motores de juegos para aplicaciones serias. La finalidad de tal estudio es de determinar qué simuladores tienen mayor capacidad para incorporar los atributos importantes para la obtención de simulaciones didácticas y atractivas.

Un repaso de la literatura existente pone de manifiesto que muchos autores utilizan términos diferentes para referirse a características similares y, además, tanto la selección de simuladores como la selección de motores para juegos serios se rigen por criterios parecidos. Así, por ejemplo, algunos estudios comparan los simuladores para robótica en función de su fidelidad (gráfica y funcional), capacidad multiplataforma o accesibilidad del código [59]. Otros autores hablan de fidelidad (gráfica y funcional), facilidad de desarrollo y coste [58]. Otros se fijan en características como simulación 3D, facilidad de uso, capacidad multiplataforma, motor de física (fidelidad funcional), accesibilidad del código y coste [54].

En 2012 Panagiotis Petridis y otros [88] realizaron un estudio sobre la selección de motores de juegos para aplicaciones serias, tomando como criterios la fidelidad (audiovisual y funcional), la accesibilidad (facilidad de uso y desarrollo, coste, documentación), la heterogeneidad (capacidad multiplataforma), la componibilidad (capacidad para reutilizar recursos propios e importar material creado con otro software) y la conectividad.

Dado que los criterios para la comparación de simuladores y motores de juego para aplicaciones serias son similares y que parte de los simuladores bajo estudio están basados en motores de juegos, considero que el enfoque de Petridis y sus colegas es el que más se ajusta a los intereses de esta tesis. A continuación describo los criterios que he tomado para la selección de simuladores.

La fidelidad audiovisual es el grado en que la simulación se ve, se escucha y se siente como el entorno operativo en términos de representación visual, controles y sonido [127]. Existen numerosos motivos por los que se considera deseable un alto grado de fidelidad audiovisual. Por una parte, cuando hay necesidad de transferir el conocimiento aprendido dentro del juego (o simulador) a situaciones del mundo real, cuanto más similitud haya entre el espacio real y el virtual, más eficaz es probable que sea la transferencia de aprendizaje [128]. Por otra parte, la inmersión cognitiva a través de contenidos de alta fidelidad es un eficaz y contrastado mecanismo para atraer la atención de los estudiantes. De hecho, el concepto "inmersión" es muy común en el ámbito de los juegos serios, donde, en general, se consideran positivos todos aquellos componentes de un entorno virtual que conduzcan a una experiencia inmersiva, tales como fidelidad audiovisual, fidelidad funcional, gafas o interfaz para realidad virtual [129], etc. Por lo contrario, rupturas de coherencia, como las provocadas por una baja velocidad de fotogramas (FPS < 30) o discontinuidades espaciales en el entorno virtual se demuestra que tienen un importante impacto negativo en la inmersión [130]. Para obtener una fidelidad audiovisual alta es necesario que el simulador cuente con un potente motor de renderizado y avanzadas técnicas de animación y audio. Los motores gráficos actuales permiten avanzadas técnicas de renderizado en texturizado (materiales multitextura, bump mapping [131], procedural texturing [132],...), iluminación (light mapping[58], per-vertex lighting, per-pixel lighting[59],...), sombreado (shadow mapping [133], shadow volume [134],...)

[58] http://en.wikipedia.org/wiki/Lightmap
[59] http://en.wikipedia.org/wiki/Per-pixel_lighting

y efectos especiales (reflejos, sistemas de partículas, billboarding [135], destellos de lentes). En cuanto a técnicas de animación actuales pueden destacarse: cinemática directa e inversa, animación por fotogramas clave, animación de mallas articuladas, mesh morphing [136], animation blending [137]). En la actualidad se utilizan diferentes procedimientos que permiten ofrecer alta fidelidad auditiva tales como técnicas de sonido 3D o flujo continuo de sonido.

La *fidelidad funcional* se define como el grado en que la simulación se comporta como el equipo operacional al reaccionar a las tareas ejecutadas por el alumno [138]. La fidelidad funcional es importante para incrementar el realismo de las simulaciones, así como el interés y la atención de los usuarios [139] [140]. En cuanto a la fidelidad funcional, es interesante que los simuladores den soporte para lenguajes de scripts y diversas técnicas de IA, tales como la detección de colisiones, cálculo de trayectorias o toma de decisiones. Del mismo modo, resulta de vital importancia un motor de física potente capaz de cálculos de dinámica de cuerpos rígidos o blandos, cuerpos articulados, vehículos, sistemas de partículas y simulación de fluidos volumétricos y telas o ropa.

Accesibilidad. Los juegos serios y los simuladores van a menudo dirigidos a individuos con poco interés en la tecnología o escaso conocimiento de las interfaces de usuario. Algunos estudios (Jarvis 2009) han demostrado que la interacción convencional mediante teclado y ratón en un mundo con múltiples grados de libertad puede resultar inicialmente abrumadora para ese tipo de usuarios. Como ejemplos interfaces que tratan de simplificar las interacciones de los usuarios en entornos virtuales con elevados grados de libertad pueden citarse los dispositivos de interacción Wii Remote controller [141] y Kinect [142], basados en acelerómetros y visión respectivamente. Asimismo, los creadores de entornos de simulación pueden ser investigadores o personal docente explorando un nuevo medio, en lugar de desarrolladores de entornos virtuales tradicionales que buscan un contenido educativo. Por lo tanto, la capacidad del simulador soportar desarrolladores y usuarios con experiencia limitada es un aspecto importante. Así pues, es importante que los simuladores ofrezcan paquetes completos de desarrollo con licencias gratuitas para usos no comerciales y acceso a código fuente, facilitando una adecuada curva de aprendizaje apoyada por abundante documentación técnica y tutoriales tanto en webs oficiales como en comunidades de usuarios.

Heterogeneidad. La heterogeneidad se refiere a las posibilidades multiplataforma hardware y software que ofrece una aplicación. Desde hace años los desarrolladores de videojuegos se toman muy enserio esta característica fundamental, lanzando sus juegos para multitud de plataformas como PC, Microsoft Xbox, Nintendo Wii o Sony PlayStation, intentando alcanzar la mayor cantidad posible de usuarios potenciales. En el campo de los juegos serios y simuladores las capacidades multiplataforma de los mismos resultan más decisivas aún si cabe debido a que el sector de la población al que van dirigidos está compuesto en gran parte por no jugadores con hardware y sistemas operativos diversos.

Componibilidad. Los simuladores a menudo tratan de modelar escenarios y situaciones del mundo real, o adaptar los datos del mundo real para su uso con unos gastos mínimos de desarrollo. El término componibilidad en este contexto se utiliza para describir tanto la reutilización de contenido creado dentro de un simulador como la capacidad del mismo para importar y utilizar datos de otras fuentes. Esto supone un gran reto debido principalmente a los avances en tecnología de hardware, la diversidad de formatos y tecnologías existentes para crear objetos o animaciones y a la elevada densidad de modelos 3d que caracteriza los mundos virtuales de alta fidelidad. Así, el imparable avance tecnológico impulsa de manera incesante la potencia de los equipos informáticos, ofreciendo crecientes capacidades de renderizado y simulación física a precios cada vez más asequibles. Esto permite, y exige a su vez, a los desarrolladores una constante creación de contenidos (modelos 3D, texturas, animaciones,...) con mayores niveles de fidelidad visual y funcional. Este fenómeno es muy usual (casi universal) y emerge de inmediato con una simple observación de los modelos de cualquier saga de videojuegos o simuladores. Así, por ejemplo, si en 2005 los modelos de los personajes principales del conocido shooter en primera persona Battlefield 2 estaban compuestos por unos 9000 triángulos (personaje con su equipo), en 2011 sus homólogos del posterior lanzamiento Battlefield 3 contaban con 15000 (un 67% más), y dos años después la resolución de los personajes del Battlefield 4 rondaban los 23000 triángulos (más de un 50% de incremento sobre el lanzamiento anterior). A este respecto, tampoco son inusuales los lanzamientos de actualizaciones y paquetes de expansión donde se aprecian importantes incrementos en el nivel de detalle de los contenidos con el objetivo de retrasar la obsolescencia de un videojuego, un claro ejemplo son los paquetes de vehículos en alta definición del exitoso juego Worl of Tanks. Cuando se intenta obtener un nivel de fidelidad elevado en un entorno virtual no es suficiente con crear

modelos precisos de los objetos y caracteres que pueblan tal mundo, especialmente si se trata de un mundo amplio con innumerables modelos presentes en cada escena. En estos casos, para que el hardware sea capaz de ejecutar la simulación en tiempo real se suele recurrir al uso de varios niveles de detalle o LOD (del inglés Level Of Detail) para cada modelo 3D correspondientes a las diferentes distancias desde las que puede ser observado por el usuario. Así, los objetos más cercanos al usuario se cargan con el LOD más rico en detalle, mientras aquéllos más alejados se corresponden con su LOD más pobre, existiendo con frecuencia varios escalones de LODs intermedios que se utilizan progresivamente conforme el usuario se acerca o aleja de tales objetos. Cada LOD suele estar compuesto por, al menos, un modelo 3D, una textura y un contorno de colisión diferentes, los cuales deben ser creados por los desarrolladores del entorno virtual. Por último, cabe destacar la gran diversidad de tecnologías y formatos existentes para crear y almacenar modelos 3D, texturas o animaciones. Así, mientras algunos motores gráficos tratan el aspecto visual de los materiales que componen los objetos como una simple textura, otros usan más de 4 texturas (pudiendo superar la docena) para definir el color base, el color del brillo, el relieve, la opacidad, la humedad, reflejos, distorsiones, etc., complicando más aún si cabe el intercambio de recursos entre aplicaciones.

Conectividad. En la actualidad son comunes los cursos online y los contenidos online de asignaturas que se cursan principalmente de forma presencial. También están adquiriendo cada vez más popularidad entre la comunidad científica los laboratorios de robótica online que permiten a estudiantes e investigadores acceder a plataformas robóticas de todo a través de un simple navegador web [49]. Así pues, resulta de gran relevancia la capacidad de los simuladores para integrarse en sistemas online de gestión de aprendizaje, permitir simulaciones remotas u ofrecer conexiones cliente-servidor que faciliten la conjunción de varios usuarios en un mismo entorno virtual.

5.3. La inmersión en entornos virtuales

La *inmersión* se refiere al grado en que un individuo se siente absorbido o absorto por una determinada experiencia [91]. La inmersión puede contribuir a la cantidad de información adquirida, habilidades desarrolladas y la posterior transferencia de conocimientos a entornos reales [44].

En 1994, Bob Witmer y Michael Singer [143] concluyeron que el grado de inmersión experimentado en un juego de ordenador puede quedar determinado por varios factores, incluyendo: (a) factores de control, o el grado en que las acciones tomadas para controlar o manipular el entorno resultan inmediatas y naturales; (b) factores sensoriales, en referencia a la calidad, riqueza y variedad de información que se presenta a los sentidos; (c) factores de distracción, o la medida en que el usuario queda aislado del entorno físico externo; y (d) factores de realismo, incluyendo el detalle, textura y realismo de la escena. La Tabla 7 detalla los factores que Bob Witmer y Michael Singer consideraron importantes para la obtención de experiencias inmersivas [91].

Factores de Control	**Factores Sensoriales**	**Factores de Distracción**	**Factores de Realismo**
o Grado de control o Inmediatez de control o Anticipación de eventos o Modo de control o Modificabilidad del entorno físico	o Modalidad sensorial o Riqueza ambiental o Presentación multimodal o Consistencia de la información multimodal o Grado de percepción del movimiento o Búsqueda activa	o Aislamiento o Atención selectiva o Conciencia del interfaz	o Realismo de la escena o Información de acorde al mundo objetivo o Significatividad de la experiencia o Ansiedad o desorientación a la finalización

Tabla 7. Factores importantes para la obtención de experiencias inmersivas con entornos virtuales

Dado el interés de esta tesis en la obtención de simulaciones inmersivas, he prestado especial interés a los factores de la Tabla 7 que son de aplicabilidad a los simuladores robóticos. De este modo, he considerado importante que los simuladores escogidos en este apartado faciliten la incorporación de los factores que favorecen la inmersión.

En cuanto a los factores de control, he valorado positivamente los simuladores que permiten ofrecer al usuario control sobre la configuración de la simulación (por

ejemplo, punto de lanzamiento del robot), ejecución fluida en tiempo real, interfaz de usuario sencillo (por ejemplo, movimiento de la cámara basado en teclas WASD y orientación con ratón) y un entorno altamente interactivo (por ejemplo, mecanismos accionables u objetos destructibles).

En lo referente a factores sensoriales, he considerado importante que los simuladores permitan generar entornos de gran riqueza para los sentidos como por ejemplo el sonido del viento, canto de pájaros o ruidos de maquinaria, movimiento de las nubes o humo, cursos de agua. Las modalidades y coherencia de la información sensitiva también es importante, por ejemplo, no es lo mismo ver cómo un robot golpea una caja que ver y oír también ese golpe. Esto es, el sonido de impacto de los objetos es importante. Aunque a veces no se le otorga importancia, es vital percibir el movimiento relativo de unos objetos respecto a otros, por ello las simulaciones deben contar con gran cantidad de texturas que eviten la imposibilidad de saber si algo se está moviendo (solo hay que pensar si una rueda lisa sin radios sobre un suelo sin textura ni discontinuidades gira, desliza o ni siquiera se mueve). La facilidad para modificar el punto de observación de la simulación es también un factor relevante para que el usuario pueda seguir la evolución de la misma.

En cuanto a los factores relacionados con el realismo, he tomado como buenos los simuladores capaces de generar entornos realistas con gran densidad de objetos de alta fidelidad gráfica. Del mismo modo, la fidelidad funcional del simulador también la he valorado muy positivamente, tomando como buenos aquellos simuladores que permiten incluir tanto objetos como mecanismos o personajes que se comportan de manera aparentemente real.

Los factores de distracción no los he tenido en consideración para la selección de simuladores porque los simuladores robóticos bajo estudio no presentan apenas diferencias entre ellos en este campo.

5.4. Selección de simuladores para robótica educativa

Como se ha indicado en el apartado de objetivos del capítulo 1, el estudio de los simuladores disponibles para robótica tiene como finalidad seleccionar aquéllos más interesantes para ser utilizados en el marco de la investigación y la educación. Del

análisis de los simuladores expuestos en el capítulo 2 extraigo que éstos pueden clasificarse en tres tipos fundamentales:

- Simuladores gratuitos open source basados en motores de juegos.
- Simuladores gratuitos open source no basados en motores de juegos.
- Simuladores comerciales no open source.

Con el fin que esta tesis resulte los más amplia y general posible, no he elegido un único simulador como aquél que podría considerarse óptimo. En su lugar, he seleccionado un simulador de cada tipo fundamental para posteriormente someterlo a prueba en la creación de escenarios destinados a ensayos de diversa índole, en los cuales se han puesto de manifiesto sus puntos fuertes y sus debilidades.

5.4.1. Selección de un simulador robótico gratuito no basado en motor de juego

La Tabla 8 muestra una comparativa de los simuladores open source gratuitos no basados en motores de juegos. Esta tabla recoge únicamente los simuladores robóticos expuestos en el capítulo 2 (estado del arte) de esta tesis, por su importancia o interés en el campo de la robótica educativa. Del análisis de esta tabla he concluido que el simulador Gazebo es el más conveniente de su sector.

Gazebo 2.2 es un simulador estrechamente vinculado a ROS, uno de los middlewares robóticos de mayor proyección en la actualidad y ambos han experimentado una meteórica expansión en los últimos años. Gazebo se distribuye en solitario o conjuntamente con algunas versiones de ROS, resultando especialmente interesante la versión Gazebo 2.2 por su compatibilidad con ROS Indigo y Ubuntu 13.10 Saucy Salamander. Gazebo 2.2 dispone de numerosas plataformas robóticas listas para usar en simulaciones, así como una biblioteca con numerosos objetos con los que construir un entorno. Tanto la biblioteca de objetos como la de robots están en continuo crecimiento gracias a las aportaciones desinteresadas de la amplia comunidad de desarrollo de este simulador. Aunque si nivel de realismo gráfico es algo inferior al de Webots o USARSim, resulta de gran calidad comparado con la fidelidad visual que ofrecen el resto de simuladores de robótica gratuitos no basados en motores de juego. Se trata, pues, de una opción muy interesante cuando se busca trabajar con software totalmente libre.

		Gazebo	OpenRAVE	SimRobot
Fidelidad audiovisual	**Renderizado**	Alto	Alto	Alto
	Animación	Baja	Baja	Baja
	Sonido	Bajo	Bajo	Bajo
Fidelidad funcional	**Programabilidad**	Media	Alta	Media
	Técnicas de IA	Bajo	Bajo	Bajo
	Física	Media	Media	Media
Componibilidad	**Soporte para aplicaciones CAD**	Medio	Medio	Medio
	Disponibilidad de contenidos	Alta	Baja	Media
Heterogeneidad	**Multiplataforma hardware**	Baja	Baja	Baja
	Multiplataforma software	Media	Media	Alta
Accesibilidad	**Documentación y soporte**	Medio	Bajo	Bajo
	Coste	Gratuito	Gratuito	Gratuito
Conectividad	**General**	Alta	Media	Media

Tabla 8. Comparación de simuladores open source gratuitos no basados en motores de video juegos

5.4.2. Selección de un simulador robótico gratuito basado en motor de juego

La Tabla 9 muestra una comparativa de los simuladores open source gratuitos basados en motores de juegos. Esta tabla recoge únicamente los simuladores robóticos expuestos en el capítulo 2 de esta tesis, por su importancia o interés en el campo de la robótica educativa. Del análisis de esta tabla he concluido que el simulador USARSim es el más conveniente de su sector.

USARSim para UDK 1.2 es un simulador que altísima fidelidad gráfica y física capaz de generar mundos virtuales de excepcional realismo gracias a su motor de juego Unreal Engine 3. UDK incorpora un editor de mundos con una enorme versatilidad y capacidad que permite trabajar de manera muy fluida generando toda clase de recursos propios de juegos de última generación. Como ejemplos de recursos interesantes para simulaciones realistas se pueden citar: animaciones de máquinas trabajando de manera cíclica en un entorno industrial o personas paseando por la escena bajo simulación; sonidos ambientales de todo tipo (máquinas, animales, personas, megafonía,...); sistemas de partículas como fuego o agua con sus efectos

sobre la visón del entorno (refracción, distorsión,...); objetos de tela que reaccionan a la acción del viento ondeando, los cuales incluso se pueden romper; terreno con relieve realista; cúpula celestial con texturas de nubes en movimiento; materiales animados para simular pantallas de electrodomésticos o fluidos en movimiento como agua o lava; entorno destruible; materiales que reflejan el entorno perfectos para introducir agua, espejos o superficies pulidas en general; sonidos de impacto para objetos que son golpeados o destruidos; posibilidad de proporcionar mensajes escritos en pantalla o auditivos; etc.

		MORSE (Blender)	USARSim (UE3)	SARGE (Unity)
Fidelidad audiovisual	**Renderizado**	Alto	Muy alto	Muy alto
	Animación	Media	Alta	Alta
	Sonido	Alto	Muy alto	Muy alto
Fidelidad funcional	**Programabilidad**	Media	Media	Alta
	Técnicas de IA	Medio	Alto	Alto
	Física	Alta	Muy alta	Muy alta
Componibilidad	**Soporte para aplicaciones CAD**	Incorporado	Alto	Alto
	Disponibilidad de contenidos	Media	Muy alta	Media
Heterogeneidad	**Multiplataforma hardware**	Baja	Media	Media
	Multiplataforma software	Baja	Alta	Media
Accesibilidad	**Documentación y soporte**	Medio	Muy alto	Alto
	Coste	Gratuito	Gratuito	Gratuito
Conectividad	**General**	Media	Alta	Media

Tabla 9. Comparación de simuladores open source gratuitos basados en motores de video juegos

Además, USARSim posee un potente editor de IA que resulta de gran utilidad para dotar de inteligencia al simulador, de manera que el usuario reciba instrucciones para alcanzar los objetivos de la simulación, realimentación de los logros y errores cometidos durante la misma, puntuación obtenida, mensajes de voz o de texto, etc. Aunque USARSim solo puede ejecutarse en entornos Windows, esto no supone ningún problema debido al hecho que los usuarios pueden conectarse como clientes al servidor de la simulación desde equipos con otros sistemas operativos vía red local o Internet. La selección de USARSim UDK 1.2 queda, pues, ampliamente justificada si

además se tienen en cuenta las positivas experiencias personales con la versión de USARSim para UT2004.

5.4.3. Selección de un simulador robótico comercial

Webots 7 es probablemente el simulador comercial de mayor uso en investigación y educación. Sus ricos gráficos y sus cálculos de física resultan de un nivel suficientemente alto para que las nuevas generaciones de alumnos, acostumbrados a gráficos 3D de calidad, lo encuentren adecuado y atractivo. Webots dispone de un editor intuitivo y sencillo de usar, permitiendo generar sistemas robóticos y pequeños mundos virtuales de manera gráfica en un tiempo razonable. Además, en cuanto a los objetos que no forman parte del entorno estático, este editor permite definirlos de manera muy completa definiendo tanto los materiales como las envolventes de colisión, su peso y su centro de masas. El centro de masas es muy importante para obtener alta fidelidad en el comportamiento físico de los objetos y hay muy pocos simuladores de robótica que permitan tratarlo de una manera tan eficaz como Webots. Este simulador cuenta con gran cantidad de plataformas robóticas y sensores listos para usar que resultan de gran ayuda al usuario cuando llega el momento de configurar los robots según la necesidad de la investigación en curso. La característica multiplataforma de este simulador y su buena conectividad con los middlewares más populares de la actualidad lo convierten en una opción muy versátil y fiable. Si el número de licencias necesarias y su consecuente coste económico no es elevado, Webots 7 puede ser una buena elección.

La Tabla 10 muestra una comparativa de los simuladores robóticos comerciales. Esta tabla recoge únicamente los simuladores robóticos expuestos en el capítulo 2 de esta tesis, por su importancia o interés en el campo de la robótica educativa. Del análisis de esta tabla he concluido que el simulador Webots es el más conveniente de su sector.

		Webots	Microsoft RDS	anyKode	LabVIEW Robotics
Fidelidad audiovisual	**Renderizado**	Alto	Alto	Alto	Alto
	Animación	Bajo	Bajo	Bajo	Bajo
	Sonido	Bajo	Bajo	Bajo	Bajo
Fidelidad funcional	**Programabilidad**	Media	Alta	Media	Alta
	Técnicas de IA	Bajo	Bajo	Bajo	Bajo
	Física	Media	Media	Media	Media
Componibilidad	**Soporte para aplicaciones CAD**	Bajo	Medio	Medio	Medio
	Disponibilidad de contenidos	Alta	Media	Baja	Media
Heterogeneidad	**Multiplataforma hardware**	Baja	Baja	Baja	Media
	Multiplataforma software	Alta	Baja	Media	Media
Accesibilidad	**Documentación y soporte**	Alta	Media	Baja	Media
	Coste	No gratuito	No gratuito	No gratuito	No gratuito
Conectividad	**General**	Media	Media	Baja	Media

Tabla 10. Comparación de las capacidades de simuladores robóticos comerciales no open source

Capítulo 6

SIMULACIONES EN DOCENCIA SOBRE INTELIGENCIA ARTIFICIAL

Sistemas Inteligentes (SI) es el nombre del área de conocimiento del Currículum de Informática relacionada con la Inteligencia Artificial (IA), que trata de los problemas que son difíciles o poco prácticos de resolver con los algoritmos disponibles. El campo de los SI/IA prepara al estudiante para determinar cuándo un enfoque "inteligente" es apropiado para un problema dado, identificar el mecanismo de representación y razonamiento adecuado, ponerlo en práctica y evaluarlo.

6.1. Plataforma de software para Sistemas Inteligentes

Sistemas Inteligentes es un área relevante en el plan de estudios de Informática de la UJI. Por ello, personal del Departamento de Ingeniería y Ciencia de los Computadores de la UJI elaboró en 2013 una plataforma de desarrollo de software basado en Python, el cual aborda los principales temas de los sistemas inteligentes, proporcionando software listo para ser utilizado en el aula y el laboratorio.

El best-seller de Stuart Russell y Peter Norvig "Artificial Intelligence: a Modern Approach" (AIMA) [144] es el libro de texto líder en Inteligencia Artificial, siendo utilizado en más de 1200 universidades de más de 100 países. Existe gran cantidad de recursos online para este libro, incluyendo un amplio repositorio con el código adecuado para todos los algoritmos del libro en varios lenguajes, pudiéndose encontrar versiones en Java, Lisp y Python para la mayoría de algoritmos.

La elección de Python como base para esta plataforma de software se debe fundamentalmente a tres motivos. En primer lugar, Python es sencillo de usar y más fácil de leer que Lisp para alguien sin experiencia en ambos lenguajes, lo cual es importante para propósitos pedagógicos tal como apunta el propio Peter Norvig en

sus conclusiones sobre Python. En segundo lugar, el código de Python tiene mucha mayor similitud al pseudocódigo del libro que el correspondiente a Lisp, lo que reduce el tiempo necesario para que los estudiantes mapeen el pseudocódigo en Python. Por último, tal como apunta Peter Norvig, el código en Java resulta de una verbosidad elevada y dista mucho del pseudocódigo de su libro.

Para completar esta plataforma de software se decidió incorporar un simulador de robots realista de código abierto con capacidad para ofrecer una rica interacción con el estudiante durante la implementación de los conceptos y el análisis de los resultados. Tal como se justifica en el siguiente apartado, como simulador se eligió USARSim v.1.2 para UDK.

En un primer momento y para comprobar el grado de adecuación del simulador, desarrollé dos entornos virtuales que permitieron implementar diferentes técnicas de aprendizaje automático: por una parte, aprendizaje supervisado con redes neuronales para una tarea de clasificación de imágenes y, por otra parte, aproximación evolutiva a búsqueda multidimensional de parámetros para un controlador de gateo de un robot humanoide.

Como robot humanoide para las simulaciones se eligió el modelo NAO [145] de Aldebaran Robotics (ver Figura 74) debido principalmente a que el Departamento de Ingeniería y Ciencia de los Computadores de la UJI ya contaba con media docena en sus laboratorios y sus capacidades se conocían por encima de las necesidades de los ensayos.

Una vez determinado el lenguaje de programación, el simulador y el modelo de robot a usar, esta plataforma de desarrollo de software ha quedado compuesta por las siguientes aplicaciones: Python(x,y), NAOqi SDK, PyNAOqi, USARNaoQi, USARSim y UDK, tal como se muestra en la Figura 73.

Python(x,y)[60] [146] es una herramienta gratuita de desarrollo de software para ciencia e ingeniería que permite cálculos numéricos y análisis y visualización de datos fundamentalmente. Python(x,y) se basa en el lenguaje de programación Python y combina una interfaz gráfica de usuario con un entorno de desarrollo científico interactivo.

[60] http://python-xy.github.io/

NAOqi SDK[61] es básicamente un conjunto de librerías en C++ diseñadas para el control del robot NAO. PyNAOqi es un conjunto de librerías en lenguaje Python que utiliza una instalación de NAOqi SDK para controlar el robot NAO mediante programas de usuario escritos en Python.

USARNaoQi es una aplicación que permite lanzar un robot NAO en un entorno de USARSim y traduce los comandos de NAOqi SDK a comandos de USARSim.

El modo de trabajar con estas herramientas es sencillo. Primero se ejecuta USARSim generando un entorno para la simulación, después el usuario introduce un robot NAO en el entorno de USARSim ejecutando la aplicación USARNaoQi para posteriormente programar su comportamiento en lenguaje Python mediante el entorno de programación Python(x,y).

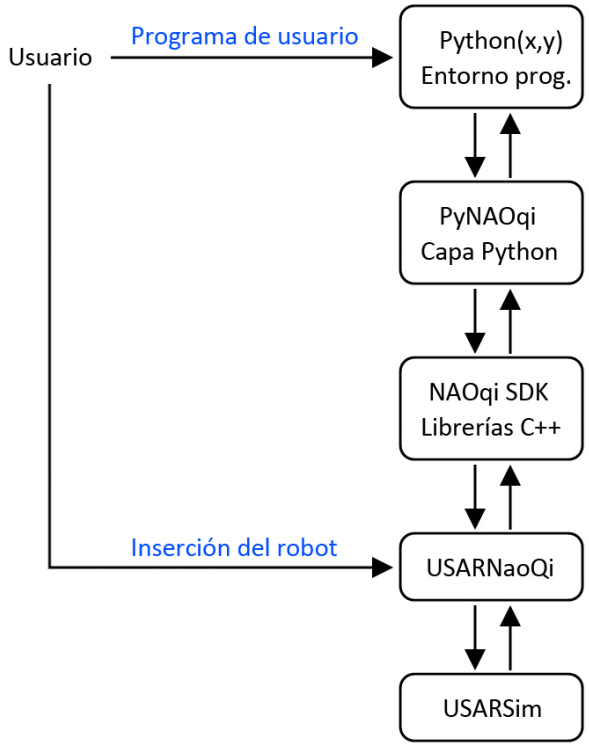

Figura 73. Plataforma de desarrollo de software basada en Python con simulación mediante USARSim

[61] http://doc.aldebaran.com/1-14/dev/naoqi/index.html

Como muestra del enfoque propuesto, se han desarrollado dos casos de estudio completos para el aula, consistentes en la implementación de dos temas de aprendizaje automático y de búsqueda, a saber, redes neuronales y algoritmos genéticos. Estos casos de estudio se detallan al final de este capítulo analizando los resultados obtenidos por alumnos de Sistemas Inteligentes de la UJI durante su implementación.

Figura 74. Robot humanoide NAO. Imagen extraída de la página web de Aldebaran Robotics[62]

6.2. Selección de un simulador para sistemas inteligentes

De los tres simuladores elegidos para desarrollar entornos virtuales para robótica educativa, decidí utilizar USARSim para UDK frente a Webots o Gazebo debido a una serie de motivos que expongo a continuación.

En primer lugar, una de las tareas del robot virtual consistiría en clasificar imágenes y en este sentido el motor gráfico de UDK ofrece mejor calidad y mayor riqueza de iluminación. Es decir, UDK permite iluminar de forma más parecida a la realidad y posee mayor cantidad de tipos de luz con los que hacerlo. Además, solamente con UDK sería posible simular el efecto que sobre la captación de una imagen produce la refracción de la luz al atravesar una columna de aire caliente en movimiento, humo, fuego, agua, etc.

En segundo lugar, USARSim 1.2 para UDK incorpora un modelo de robot NAO desarrollado en la Universidad de Ámsterdam [147]. Aunque el aspecto de tal modelo no es muy realista, su comportamiento físico sí lo es y, además, presenta la importante

[62] http://doc.aldebaran.com/2-1/home_nao.html

ventaja que utiliza una instalación local de NAOqi SDK para procesar comandos al robot NAO, lo que significa que exactamente el mismo código Python se puede utilizar para el robot real y el robot simulado.

6.3. USARSim UDK v.1.2 en simulaciones para Sistemas Inteligentes

Las características de los entornos de simulación aconsejaban el uso de USARSim 1.2 para UDK como se ha comentado en el apartado anterior, así que procedí a la elaboración de dos entornos de simulación con USARSim y a introducir un nuevo modelo para el robot NAO para mejorar el que ya incorpora este simulador.

6.3.1. Descripción general

Con el objetivo de implementar los algoritmos de aprendizaje supervisado con redes neuronales para tareas de clasificación de imágenes, construí un laberinto en el que un robot debía reconocer y clasificar una serie de imágenes para poder salir del mismo. Intentando explotar las capacidades de UDK dando mayor riqueza y realismo a las simulaciones, introduje distintas fuentes que provocaran cambios de luz o distorsionaran las imágenes a captar por los robots. Para evitar un silencio irreal y conseguir una experiencia más inmersiva añadí gran cantidad de sonidos ambientales que van des del sonido de un curso de agua, varios fuegos, viento hasta aves nocturnas y otros sonidos intrigantes. Este mundo virtual también cuenta con una gran bóveda celestial lejana y realista adornada con una enorme luna llena que ilumina toda la escena, proyectando las sombras de muros, columnas, ramas secas y resto de naturaleza muerta.

La Figura 75 muestra el laberinto que debe recorrer el robot NAO guiado por los símbolos que hay en las paredes desde la entrada (situada en la parte inferior derecha) hasta la salida (situada en la parte superior). En la Figura 76 puede verse este laberinto en toda su extensión con un robot NAO a mitad del recorrido. En la parte izquierda de la Figura 77 se muestra un pebetero con fuego que dificulta el reconocimiento de uno de los símbolos a clasificar por el robot, en la parte derecha de esta figura se pueden ver los cuatro símbolos que permitirán al robot conocer sus posibles direcciones de avance en su camino por el laberinto. El mencionado fuego consta de varios sistemas de partículas que se detallarán más adelante en este capítulo, introduciendo refracciones, luces de intensidad y ubicación variable, humo y proyección de partículas incandescentes.

Figura 75. Vista de frente del laberinto para algoritmos de clasificación de imágenes

Figura 76. Vista global del laberinto para algoritmos de clasificación de imágenes

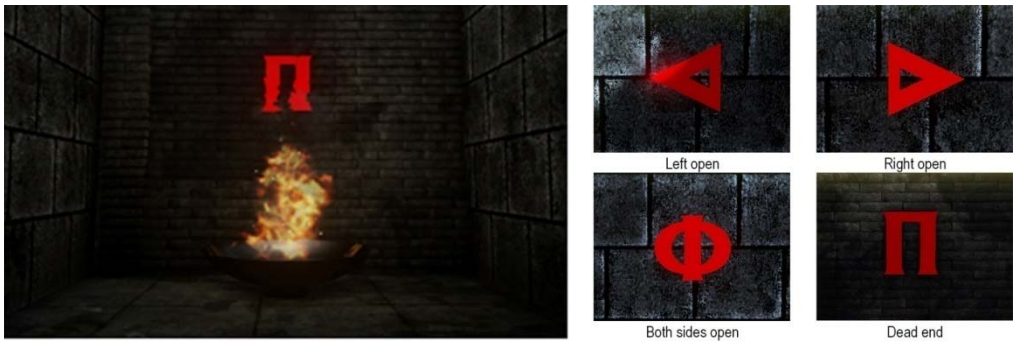

Figura 77. Entorno del laberinto con las formas a reconocer (derecha) y una fuente de distorsión dinámica de la imagen (izquierda)

En cuanto al entorno para pruebas de algoritmos de aproximación evolutiva a búsqueda multidimensional de parámetros para un controlador de gateo, técnicamente solo requería de un gran espacio abierto sin obstáculos, de modo que utilicé el mismo entorno eliminando el laberinto, el curso de agua y los árboles cercanos.

En lo referente al modelo del robot NAO, decidí modificar los archivos existentes en USARSim UDK v.1.2 e introducir un modelo mucho más exacto tanto gráfica como funcionalmente. En la Figura 78 puede observarse que el modelo de la izquierda (el que incorpora USARSim UDK v1.2) no tiene siquiera manos siendo su aspecto general bastante inexacto a pesar de estar formado por una importante cantidad de triángulos y vértices. El aspecto irreal del robot podría disminuir el grado de aceptación del usuario y con ello su motivación [44] por utilizar el simulador. El modelo de NAO del centro de la figura resulta mucho más parecido al robot real (a la derecha) con una cantidad similar de triángulos y vértices. Además, como se comenta en apartados siguientes de este capítulo, para cada parte del robot desarrollé varias versiones con diferentes niveles de detalle en orden descendente con el objetivo de poder simular de manera fluida grupos de robots sin que el usuario perciba pérdida de fidelidad gráfica.

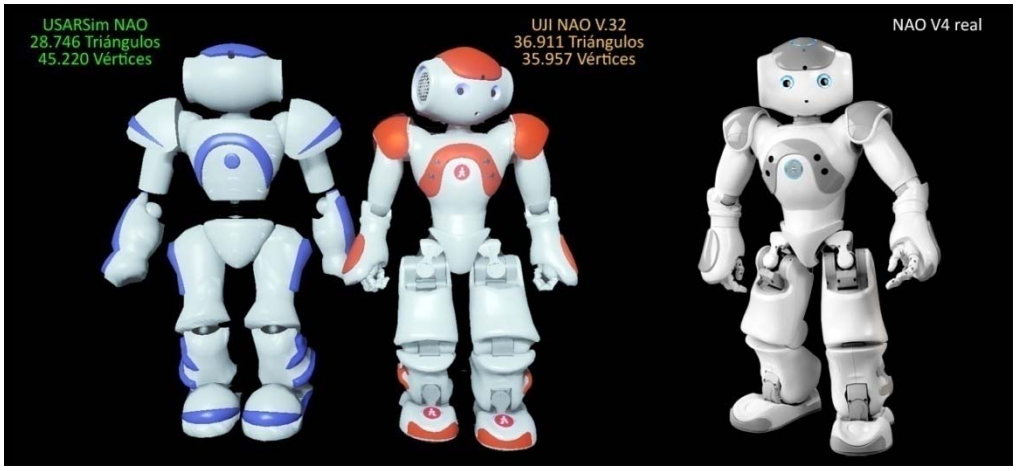

Figura 78. Comparativa entre el modelo de NAO de USARSim UDK v.1.2 (izquierda), mi modelo de NAO v.32 y un NAO v.4

En los puntos siguientes se detalla el procedimiento seguido para obtener tanto los mencionados entornos como el modelo de robot NAO en sus diferentes niveles de detalle según la distancia de observación.

6.3.2. Metodología

El procedimiento utilizado para la obtención del laberinto y del robot virtual, ya se ha indicado en el capítulo 3 y no difiere mucho del procedimiento ya detallado en el capítulo 4. Teniendo esto en cuenta, indico a continuación los pasos seguidos en la elaboración de este entorno y el robot virtual, intentando no dar información redundante.

6.3.3. Creación de un laberinto realista con USARSim UDK v.1.2

En el entorno del laberinto pueden distinguirse fundamentalmente los siguientes elementos: paredes del laberinto, muro exterior, suelo, templo, curso de agua, árboles, bóveda celestial, sonidos ambientales, minotauro, fuego, humo y terreno exterior. Para explicar la creación de estos objetos se han agrupado según su similitud funcional dentro del simulador.

Paredes del laberinto, templo, suelo y muro exterior

El laberinto para la prueba de algoritmos de clasificación de imágenes debe cumplir una serie de condiciones en cuanto a dimensiones para poder ser utilizado con el robot humanoide NAO. Como la estatura de este robot es de 58 centímetros, he procedido a levantar un entramado de paredes de un metro de altura con una separación de un metro para permitir la maniobra del robot. La textura utilizada para estas paredes se encuentra disponible en el editor de niveles de UDK, lo que ha facilitado la elaboración del material de aspecto pétreo adecuado. Es importante recalcar que el motor gráfico de UDK permite asignar a los modelos 3D materiales en lugar de simples texturas, consiguiendo efectos de mucho mayor realismo y posibilidades prácticamente infinitas. Los materiales están compuestos por varias texturas y por funciones que las modifican y combinan con capacidad para generar casi cualquier aspecto o efecto imaginable. Así, se puede crear, por ejemplo, una falsa impresión de relieve sobre superficies lisas, o incluso movimientos aleatorios para crear nubes, cursos de lava, agua y fluidos en general. Las texturas realistas son importantes para aumentar la percepción del movimiento del robot y de la cámara del usuario durante las simulaciones.

En las paredes del laberinto he colocado los símbolos que debe reconocer el robot durante las simulaciones para poder alcanzar la salida. Se trata de letras del alfabeto

griego que indican en qué dirección se puede seguir avanzando (ver Figura 77). Estos símbolos miden 17cm x 17cm y los he situado, en su mayoría, de manera que su centro quede a la altura de la cámara frontal del robot NAO (53 cm) para que resulten fáciles de encontrar y evitar efectos de perspectiva. Estas medidas las he escogido de manera que los símbolos puedan leerse perfectamente a distancias de observación entre 26cm y un metro (ver Figura 80), lo cual está en consonancia con las dimensiones del laberinto y con las capacidades del NAO. Es importante indicar que la cámara frontal de los robots NAO v.3.x (que son los disponibles en el Laboratorio de Robótica Inteligente de la UJI) tiene un campo de visión (FOV) horizontal de 47,8º y uno vertical de 36,8º (ver Figura 79) con una resolución VGA de 640 x 480 píxeles y una distancia mínima de observación recomendada de 30cm. Esto significa que a menos de 26cm de distancia del símbolo el NAO no puede verlo entero porque aquél sobrepasa los límites del área captable por su cámara frontal. Para distancias de observación entre 50 y 100 centímetros los símbolos ocupan entre el 20% y el 5% de la imagen captada por la cámara frontal del robot NAO tal como se puede deducir de la Figura 79, de la Ecuación 1 y la Ecuación 2

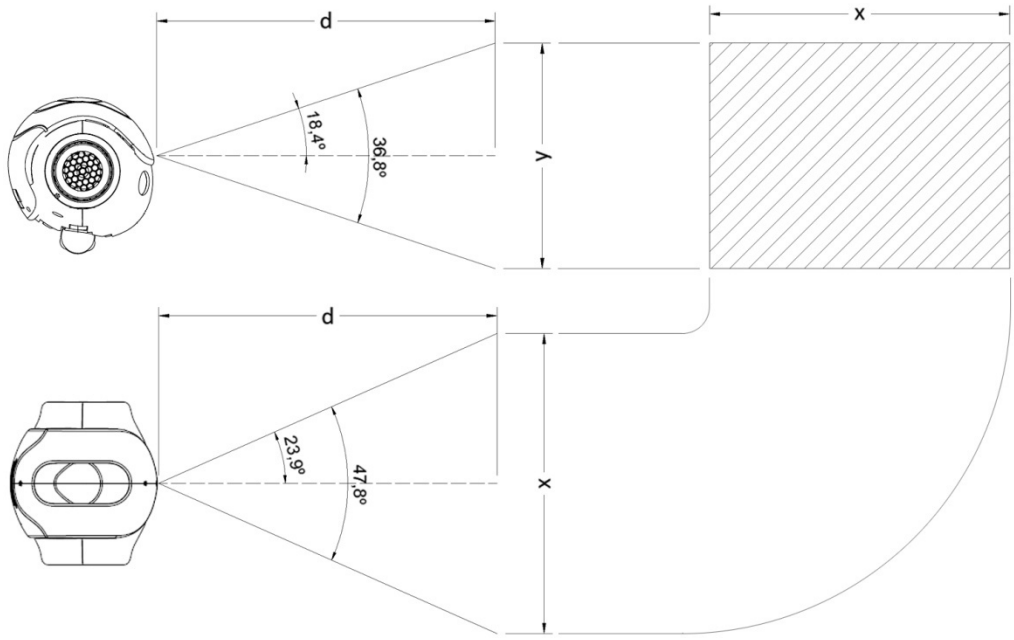

Figura 79. Campo de visión de la cámara frontal del robot NAO y tamaño del área captada a una distancia de observación "d"

$$A = x \cdot y = \left[2 \cdot d \cdot \tan\left({}^{47{,}8^{o}}/_{2}\right)\right] \cdot \left[2 \cdot d \cdot \tan\left({}^{36{,}8^{o}}/_{2}\right)\right]$$

Ecuación 1. Área de la imagen captada por la cámara frontal de un NAO v.3.x en función de la distancia de observación

$$R = \frac{17 \cdot 17}{A} * 100$$

Ecuación 2. Relación porcentual del área que ocupa el símbolo en la imagen captada por la cámara. En esta ecuación el área A debe introducirse en cm^2

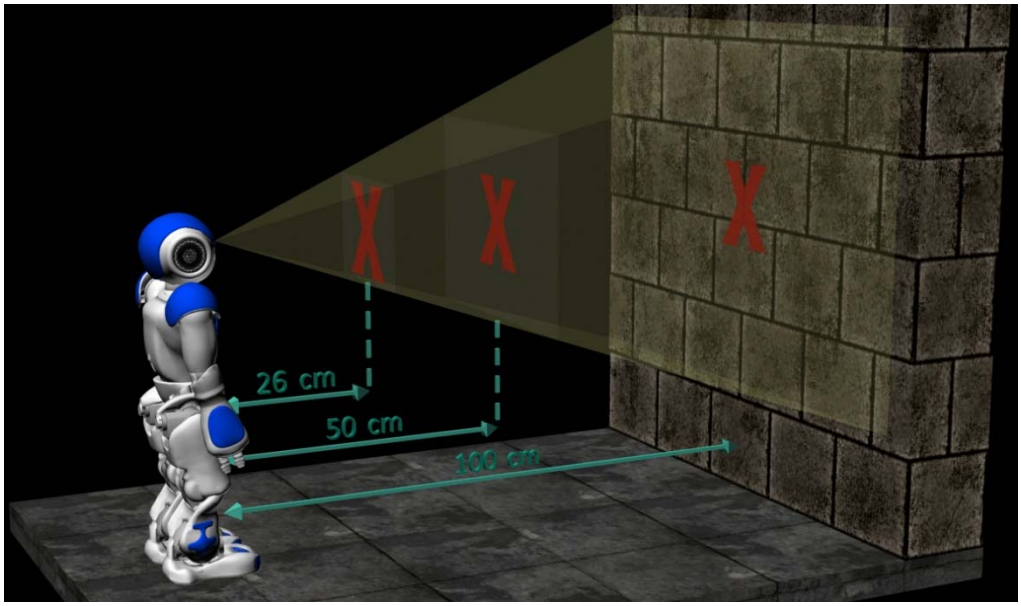

Figura 80. Relación de tamaños de los símbolos captados en función de la distancia de observación

El templo incluido en el mundo del laberinto tiene como función principal proporcionar al usuario una ambientación más rica y, por tanto, una experiencia más inmersiva [91]. No obstante, este templo lo he diseñado para que sirva de punto de salida a un minotauro (ver Figura 81) que recorre todo el laberinto arrasando todo a su paso transcurridos 10 minutos después de empezar la simulación. Con esto se consiguen cuatro objetivos importantes según las teorías del aprendizaje con entornos virtuales:

Objetivo 1. Ofrecer realimentación en tiempo real, indicando al usuario que el tiempo disponible para superar la prueba ha terminado. De esta manera se incrementa el grado de evaluación (medido en rapidez) de los algoritmos empleados por los alumnos.

Objetivo 2. Incrementar el grado de fantasía ofrecidos en la simulación para conseguir experiencias más inmersivas y atractivas.

Objetivo 3. Aumentar el misterio y con ello la motivación de los alumnos por pasar más tiempo con el simulador. Los alumnos conocen la presencia del minotauro, pero no saben qué papel juega hasta que finaliza el tiempo disponible para terminar la prueba.

Objetivo 4. Elevar el desafío ofrecido por la simulación, pues los algoritmos implementados por los alumnos deberán clasificar bien las imágenes y, además, actuar con rapidez.

Figura 81. Minotauro introducido en el simulador. La bestia recorre el laberinto en busca del robot humanoide transcurridos 10 minutos desde el inicio de la simulación

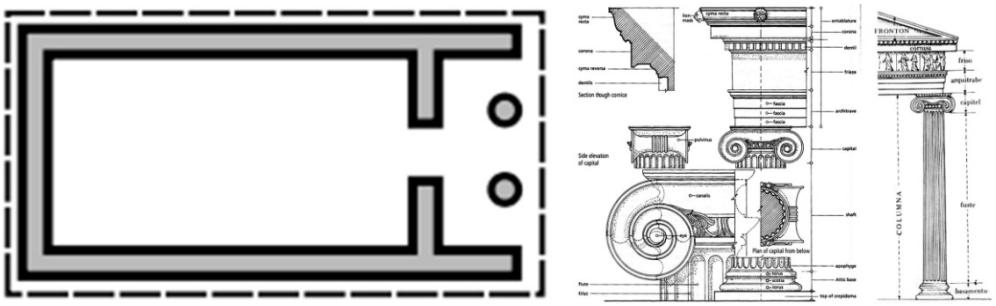

Figura 82. Planta de un templo díptero y elementos característicos del arte griego de estilo jónico

En cuanto a la creación del edificio en sí, éste lo he construido según planos de templos dípteros griegos de estilo jónico (ver Figura 82) para estar en consonancia

con la figura del minotauro. En el interior del templo he situado un pedestal con relieves que conmemoran la victoria de Teseo frente a la citada bestia (ver Figura 83) para evitar que su posible aparición no resulte "fuera de lugar" con sus negativas consecuencias en cuanto a inmersión.

Figura 83. Pedestal situado en el interior del templo con texturas simulando relieve. Capturas de pantalla tomadas desde varios puntos de vista

El diseño de los modelos 3D del conjunto del templo lo he realizado con 3D Studio Max partiendo de fotos, dibujos y algunas imágenes de películas ambientadas en mitología griega. Es importante destacar la enorme reducción de polígonos que he conseguido en los elementos que componen el edificio utilizando texturas del tipo *normal map* que simulan relieves de forma realista donde no hay más que superficies lisas (ver Figura 84). Como ejemplo de orden de magnitud puede citarse que cada tramo de muro exterior original está formado por 51060 triángulos, mientras que el muro utilizado en el simulador solo tiene 152, conservando un aspecto muy similar tal como puede apreciarse en la Figura 85.

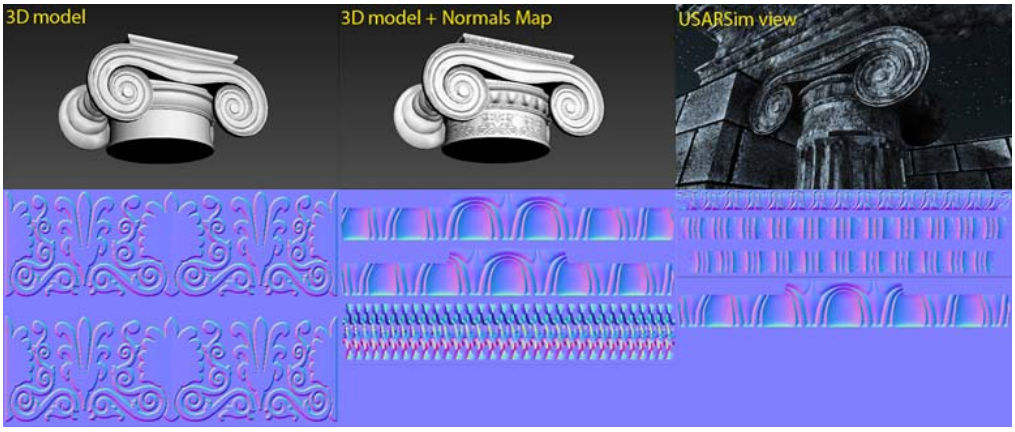

Figura 84. Modelo de capitel sin relieves (izquierda) y resultados visuales después de aplicar un normal map (abajo) y materiales pétreos en UDK

El procedimiento seguido para obtener las texturas de *normal map* ha sido diferente en función de los datos de partida. Así, partiendo de una imagen 2D he conseguido un efecto de relieve mediante una aplicación de NVIDIA para Adobe Photoshop llamada NVIDIA Normal Map Filter, la cual permite ajustar la profundidad y el grosor de las líneas de relieve (ver Figura 86). En cambio, partiendo de un modelo 3D altamente detallado y utilizando la opción *render to texture* de 3D Studio Max también se puede obtener una textura *normal map* que simule relieves sobre superficies lisas. Éste ha sido el procedimiento que he seguido para implementar los relieves de capiteles, columnas y muros exteriores del laberinto. El uso de estas texturas ha sido posible gracias a las excelentes capacidades del editor de materiales de UDK, el cual ofrece posibilidades casi inagotables a la hora de elaborar materiales para los modelos.

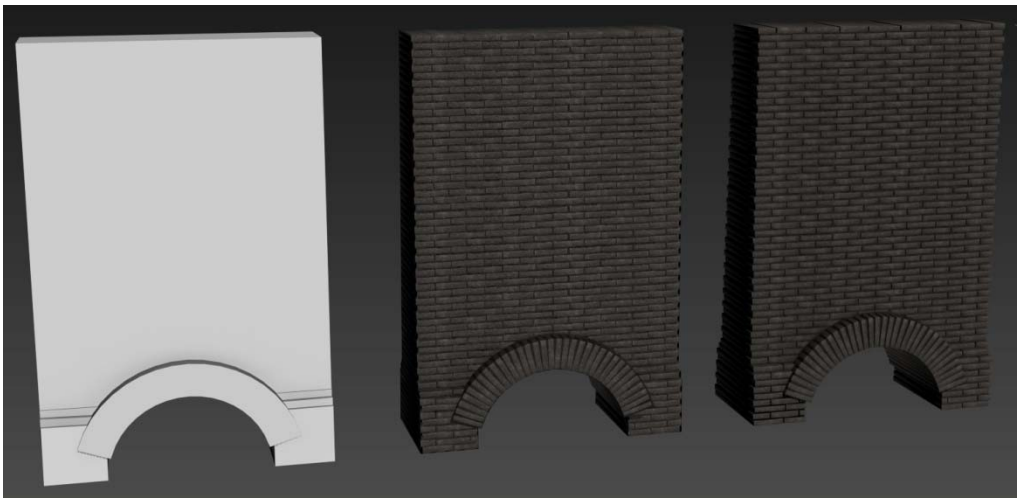

Figura 85. Tramo de muro exterior del laberinto. A la izquierda puede verse la forma geométrica básica a la que se le han aplicado las texturas de relieve figura central) para obtener un aspecto similar al de un muro compuesto por 51060 triángulos (figura de la derecha)

El cerramiento exterior que contiene el laberinto está formado por gruesos muros de sección trapezoidal de 2 metros de altura cuyo objetivo es aislar el recorrido del laberinto respecto al entorno exterior tanto funcional como estéticamente. Es decir, el robot no debe poder pasar a través de estas paredes que, además, ocultarán el despoblado terreno exterior al usuario del simulador. El muro lo diseñé en 3D Studio Max como una unión de pequeños poliedros alargados y planos para obtener un *normal map* que aplicado sobre superficies planas permita simular la presencia de cientos de piedras con forma alargada. Para no dar una sensación de patrón repetitivo

en el aspecto de estos muros y ganar en realismo, he hecho uso de una técnica que permite simular capas de suciedad aleatorias sobre cualquier superficie mediante texturas adicionales que se mezclan con las que conforman el material en estado nuevo. Se trata de un recurso muy utilizado en los videojuegos y relativamente fácil de implementar con el potente editor de materiales de UDK. Este efecto de suciedad puede observarse en la Figura 75 sobre los citados muros y en la Figura 80 sobre las baldosas del suelo.

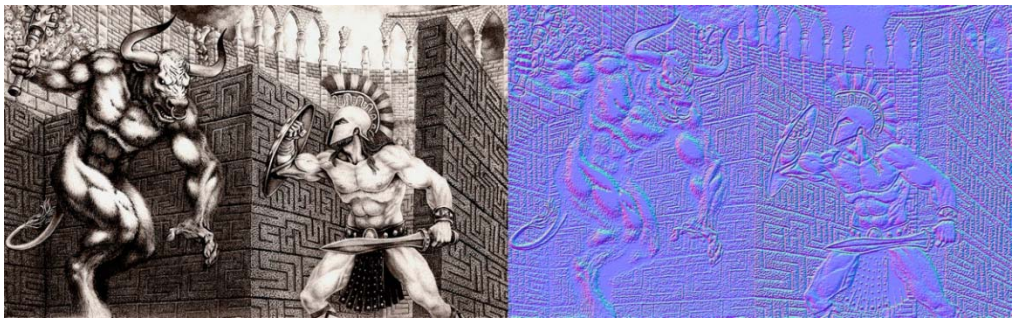

Figura 86. Normal map (a la derecha) obtenido con NVIDIA Normal Map Filter con Adobe Photoshop, partiendo de una imagen plana en escala de grises[63] (a la izquierda)

El suelo de todo el recinto que contiene el laberinto lo he creado a partir de un plano con un material que simula baldosas de piedra envejecida. Las texturas que componen dicho material se encuentran disponibles en las librerías del propio editor de niveles de UDK, hecho que ha facilitado el diseño del suelo. Solamente un curso de agua salvado por un pequeño puente de madera interrumpe la continuidad de este suelo. La función de este puente en forma de rampa es servir de dificultad para llegar a la puerta de salida en posibles simulaciones futuras.

En cuanto a envolventes de colisión, éstas las he exportado desde 3D Studio Max de manera similar a la explicada en el capítulo 5. Hay que subrayar que los editores de niveles UT2004 y UDK tratan las envolventes de colisión de manera muy parecida, motivo por el que no describo este aspecto de nuevo en este capítulo. Tan solo cabe destacar que ha cambiado la nomenclatura con la que deben nombrarse las envolventes para ser reconocidas como tales por el editor de UDK.

[63] http://streamfinancial.com.au/the-myth-of-theseus-and-the-minotaur-lessons-from-the-ancient-greeks/

Terreno exterior, bóveda celestial, iluminación y sonidos

Como simulador con motor de juego, USARSim para UDK permite la introducción de una realista bóveda celestial y un extenso terreno que evitan la sensación de estar flotando en el espacio, otorgando gran realismo a las simulaciones y facilitando la inmersión del usuario. Este efecto también puede potenciarse eliminando el incómodo silencio absoluto mediante otra de las capacidades típicas de los motores de juegos, los sonidos ambientales.

El terreno donde está emplazado el laberinto lo he creado de manera que resulte realista y a su vez no suponga una carga para la mayoría de tarjetas gráficas. Por una parte, se trata de un terreno plano muy extenso bordeado por formaciones montañosas que dificultan la percepción de los límites del entorno. Por otra parte, este terreno lo he diseñado con la mínima densidad de malla (técnica conocida como *low tessellation*) compatible con las citadas elevaciones y con la depresión necesaria para incluir un pequeño río o curso de agua. El terreno no lleva ninguna textura en concreto para evitar la poca realista sensación de patrón de textura repetitivo cuando la cámara se eleva unos metros por encima del laberinto. En su lugar, he procedido a pintar el terreno como si fuera un lienzo tomando texturas en lugar de colores aprovechando otra de las capacidades habituales de los motores de juegos. Sobre el terreno y los muros del laberinto he puesto vegetación muerta, árboles y arbustos secos fundamentalmente, buscando provocar una sensación de ambiente tétrico e inhóspito. Esta vegetación se halla disponible en una de las bibliotecas de recursos que UDK pone a disposición de los usuarios para agilizar la construcción de entornos virtuales.

El cielo de este entorno está formado por una bóveda celestial y una luna. La bóveda es una esfera enorme con un material sencillo basado en una textura que simula un cielo estrellado. El efecto realista de escena nocturna lo he obtenido mediante una iluminación direccional que simula proceder de la luna, proyectando sombras y haces de luz de un color blanco azulado (ver Figura 87). La luna es una gran esfera con un material formado por una textura lunar realista y dotado de un efecto de auto iluminación y resplandor, lo que afianza la impresión que es este astro el que ilumina toda la escena. Este entorno nocturno con luz suave de baja intensidad lo he escogido intencionadamente para poder comprobar el efecto que sobre el reconocimiento de los símbolos tiene una iluminación variable que no podría obtener si brillara el sol con fuerza. Los pebeteros con fuego introducen una luz variable en intensidad y posición,

además de humo y refracciones que ocultan parcialmente y deforman la imagen a captar por la cámara del robot tal como ya he mencionado.

Figura 87. Bóveda celestial con la luna iluminando el entorno con una fuente de luz direccional

El espacio intermedio entre el terreno y el cielo está ocupado por una ligera niebla azulada que incrementa el efecto de distancia con los elementos del entorno que se suponen lejanos tales como las montañas del borde exterior o la superficie de la bóveda celestial y la luna. Se trata éste de un recurso comúnmente utilizado en los videojuegos y, por tanto, sencillo de implementar en UDK. En la Figura 87 se puede observar el efecto de esta niebla sobre la nitidez de las montañas y el cielo bajo.

La ambientación la he completado añadiendo varios sonidos que simulan graznidos aves nocturnas, viento, fuego o un curso de agua. Como el entorno lo he preparado ejecutar la animación de un minotauro, también se puede escuchar de vez en cuando una especie de rugido desconcertante para alertar al usuario de manera que la posible aparición de tal bestia resulte más esperada, creíble y aceptada.

Minotauro

El minotauro que he incluido en este entorno lo he desarrollado en base a una maya dotada de un esqueleto ("SkeletalMesh" en UDK) que le permite ser animada. Este tipo de recurso es muy empleado para crear los personajes que pueblan los video juegos y lo expongo con más detalle en el capítulo 8 de esta tesis. Para incrementar la fidelidad audiovisual y funcional de las simulaciones he utilizado la herramienta *APEX Clothing*[64] (PhysX Clothing) [148] incluida en el plug-in NVIDIA PhysX[65] para 3D Studio Max. APEX Clothing utiliza técnicas vanguardistas que permiten simular tejidos y ropa de manera altamente realista. La rica interacción de la ropa con el cuerpo en movimiento de los personajes y con el viento eleva sustancialmente el grado de fidelidad de la escena.

La Figura 88 muestra la etapa final del desarrollo del minotauro con el plug-in NVIDIA PhysX para 3D Studio Max. A la derecha de la imagen se pueden ver los parámetros del modificador APEX Clothing aplicado a la maya del minotauro ya dotada de un esqueleto. Las líneas rojas sobre la ropa indican el grado de libertad del movimiento de la ropa expresada en distancia máxima de desplazamiento sobre su punto de reposo.

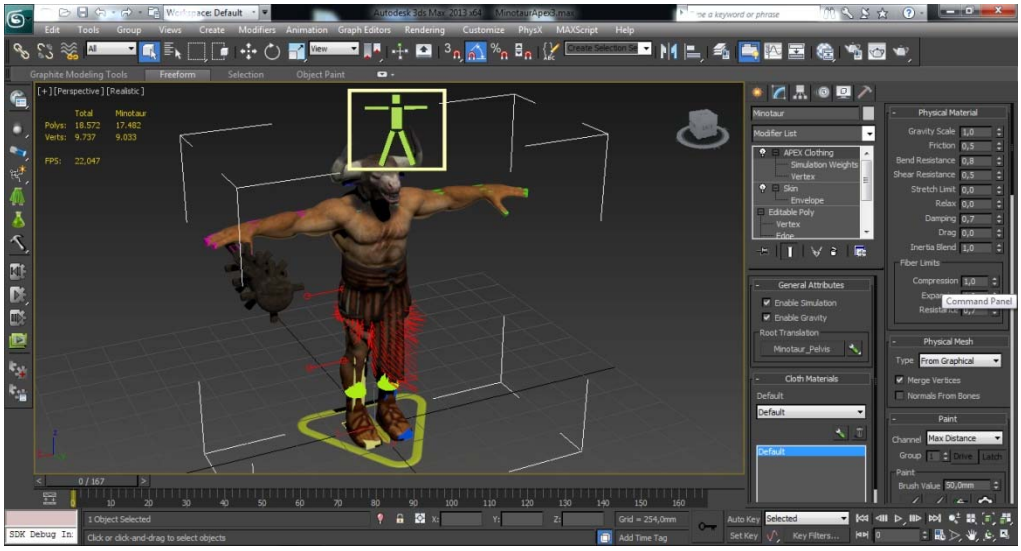

Figura 88. Creación de ropa realista mediante el modificador APEX Clothing para 3D Studio Max

[64] http://physxinfo.com/wiki/APEX_Clothing
[65] http://developer.nvidia.com/clothing

Sistemas de partículas

Como sistema de partículas se entiende toda fuente de emisión de partículas de cualquier tipo tales como chispas, llamas, humo, gotas de agua, fragmentos de objetos, burbujas, calor, gases, polvo, etc. Se trata de un recurso que suele aportar riqueza a los mundos virtuales tanto en forma elementos visuales como auditivos, pues la emisión de las partículas suele estar ligada a algún tipo de sonido o ruido.

Como ya he comentado, en el entorno del laberinto he introducido varios de estos sistemas conformando fuego y humo. Como orden de magnitud, puede citarse que un fuego consta de 5 sistemas de partículas (textura de llama, resplandor de llama, humo, chispas y refracción) definidos por funciones de carácter aleatorio que determinan desde su trayectoria hasta su tiempo de vida (ver Figura 89). La inclusión de los sistemas de partículas, en este caso, tiene una componente funcional muy importante, ya que tanto el fuego como el humo dificultan la lectura de los símbolos de navegación poniendo a prueba el algoritmo de clasificación de imágenes implementado. Éste es uno de los motivos por los que decidí utilizar USARSim para UDK, ya que los simuladores no basados en motores de juegos no suelen ofrecer este tipo de recursos como es el caso de Webots o Gazebo. Hay que advertir que los sistemas de partículas usados en gran cantidad pueden suponer una sobrecarga para muchas tarjetas gráficas, por lo que no conviene usarlos frívolamente.

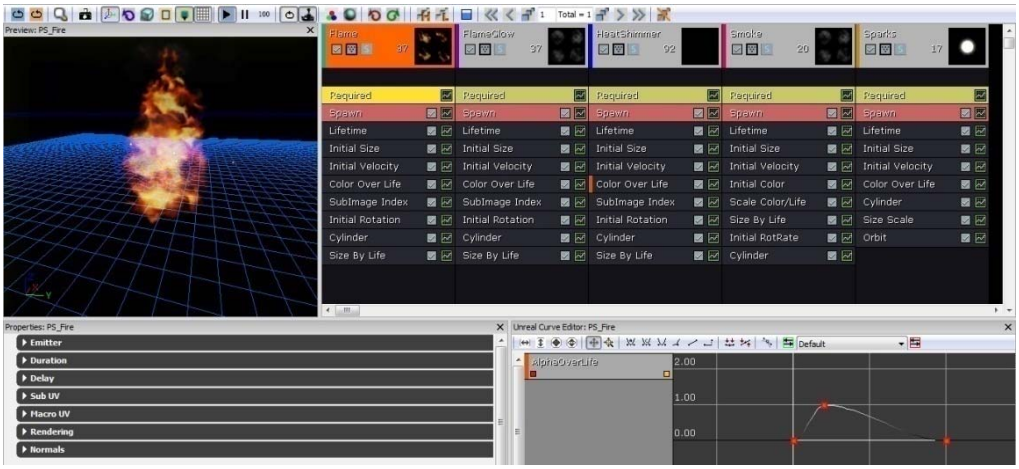

Figura 89. UDK Particle System Editor mostrando los cinco sistemas que componen un fuego

6.3.4. Modelo virtual del robot NAO

NAO es un robot humanoide autónomo programable desarrollado por Aldebaran Robotics, una empresa francesa con sede en París, con las siguientes características principales (ver Figura 90):

- Cuerpo con 25 grados de libertad cuyos elementos clave son los motores eléctricos y actuadores.
- Completa red de sensores que incluye 2 cámaras, 4 micrófonos, sensor de distancia sonar, 2 emisores y receptores infrarrojos, una placa de inercia, 9 sensores de contacto y 8 sensores de presión.
- Varios dispositivos de comunicación, incluyendo sintetizador de voz, luces LED, y 2 altavoces de alta fidelidad.
- CPU Intel Atom a 1,6 GHz ubicada en la cabeza que ejecuta Linux kernel y soporta middleware propietario de Aldebaran (NAOqi)
- Batería de 27,6 vatios-hora que proporciona al robot 1,5 o más horas de autonomía en función del uso.

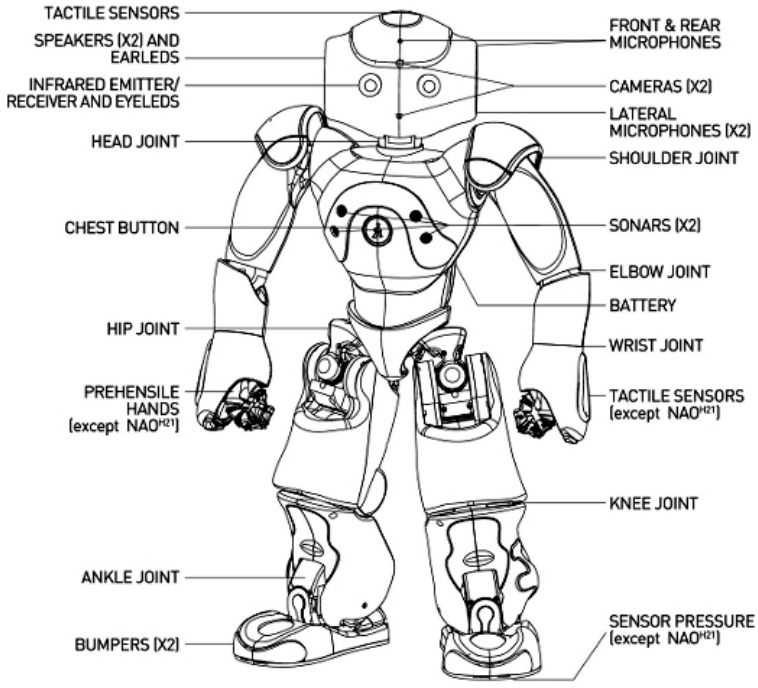

Figura 90. Características principales del robot humanoide NAO. Esquema procedente de la documentación web de Aldebaran Robotics

Tal como ya he comentado, USARSim para UDK incorpora un modelo de robot NAO que, aunque no es muy exacto visualmente, es una buena base para desarrollar un modelo más exacto y con mejor funcionalidad física.

En cuanto al modelo gráfico de NAO que he creado, éste lo he desarrollado en 4 versiones con diferentes niveles de detalle (LOD, Level Of Detail) para ofrecer una carga gráfica decreciente con la distancia de observación (ver Figura 91) para permitir la simulación fluida de grupos de robots. Estos niveles de detalle los he creado guardando una relación de triángulos del 75% (aproximadamente) entre niveles consecutivos de manera que el LOD_1 tiene un 25% menos de triángulos que su predecesor el LOD_0 (ver Ecuación 3). Esta suave reducción de polígonos dificulta que el usuario detecte el momento en que se carga un LOD u otro.

$$LOD_n = 0{,}75 * LOD_{n-1}$$

Ecuación 3. Relación entre la cantidad de triángulos que forman niveles de detalle consecutivos para favorecer transiciones suaves no detectables por el usuario y evitar el fenómeno *popping*

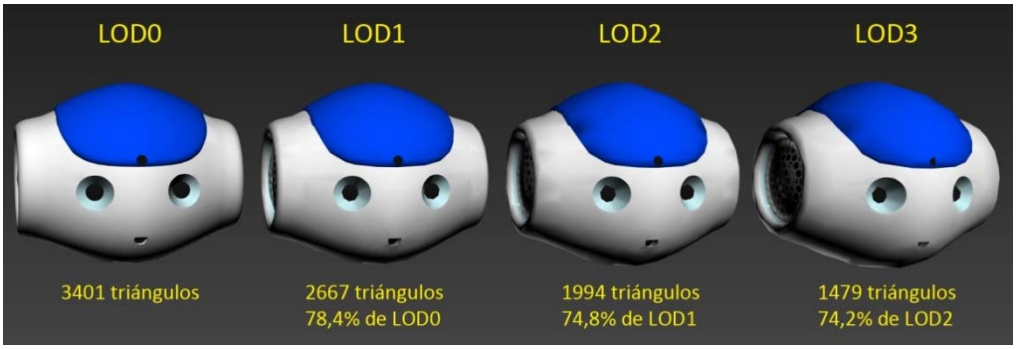

Figura 91. Niveles de detalle de la cabeza del robot NAO para cuatro intervalos de distancias de observación

Los niveles de detalle permiten que el usuario perciba la escena como si todos los objetos que la conforman fuesen de alta fidelidad gráfica [149]. Cuando observamos un objeto muy de cerca dentro de un entorno virtual suceden varias cosas: por una parte, ese objeto se ve con todo lujo de detalle pudiendo apreciarse si se trata de un modelo de alta o baja fidelidad gráfica; por otra parte, a medida que acercamos la cámara al mencionado objeto se reduce el campo visual y con ello la cantidad de objetos presentes en la imagen captada, lo que a su vez reduce la carga gráfica para el hardware utilizado. Así, cuando la cámara se aleja de los objetos y permite observar un gran número de ellos, se cargan los niveles de detalle bajos y cuando la cámara se

acerca y se reduce el número de objetos visibles, se cargan los niveles de detalle altos, lo que permite al hardware trabajar siempre con un número de triángulos razonable manteniendo una alta satisfacción por parte del usuario.

En lo referente a fidelidad funcional, el modelo de robot NAO que he introducido en USARSim incorpora importantes mejoras comparado con el modelo que viene incluido en el paquete del simulador. Se trata de un modelo más exacto, con mayores capacidades funcionales y con un comportamiento en simulación más parecido al del robot real, tal como se detalla a continuación.

En primer lugar, las envolventes de colisión de mi modelo de humanoide NAO las he trazado con mayor exactitud siguiendo más fielmente el contorno de todo el robot, permitiendo una fiel interacción con los objetos y el terreno. En la Figura 92 puede apreciarse la sensible diferencia en cuanto a envolventes de colisión entre el modelo de NAO que incorpora USARSim y el introducido por mí. Las envolventes de colisión de mi modelo son más exactas y, por tanto, requieren cálculos más complejos por parte del hardware. No obstante, después de hacer numerosas pruebas de rendimiento en cuanto a suavidad de ejecución con diferentes geometrías de colisión he llegado a la conclusión que el hardware actual permite el uso envolventes mucho más complejas que las que yo he utilizado sin perder fluidez de ejecución. El hardware que he utilizado para las pruebas de rendimiento es un portátil con un procesador Intel i5-2430M, 8Gb de RAM y tarjeta gráfica NVIDIA GeForce GT 520M.

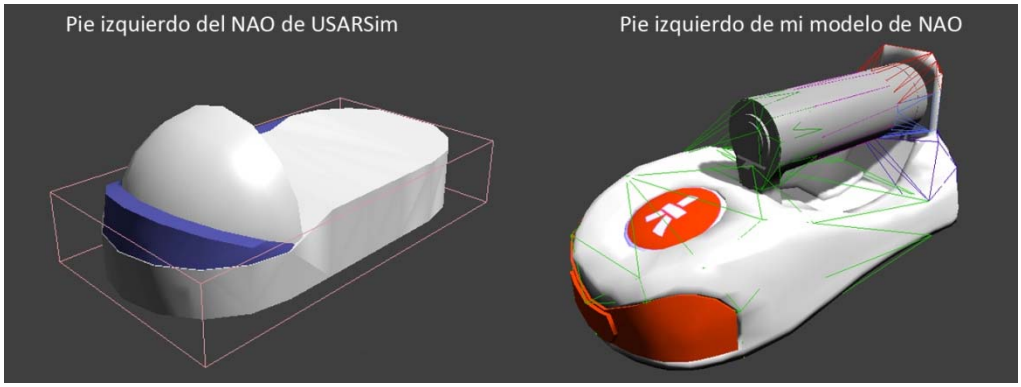

Figura 92. Comparativa de envolventes de colisión entre el pié del NAO de USARSim y el nuevo modelo introducido por mí

En segundo lugar, he separado antebrazo de la mano permitiendo el giro de la muñeca del robot NAO mediante una articulación tal como se corresponde al modelo real,

ganando así un grado de libertad en cada brazo. La Figura 93 muestra en su parte izquierda una vista en perspectiva de la mano del robot NAO donde puede apreciarse su posibilidad de giro respecto al antebrazo según el eje X del sistema de coordenadas indicado.

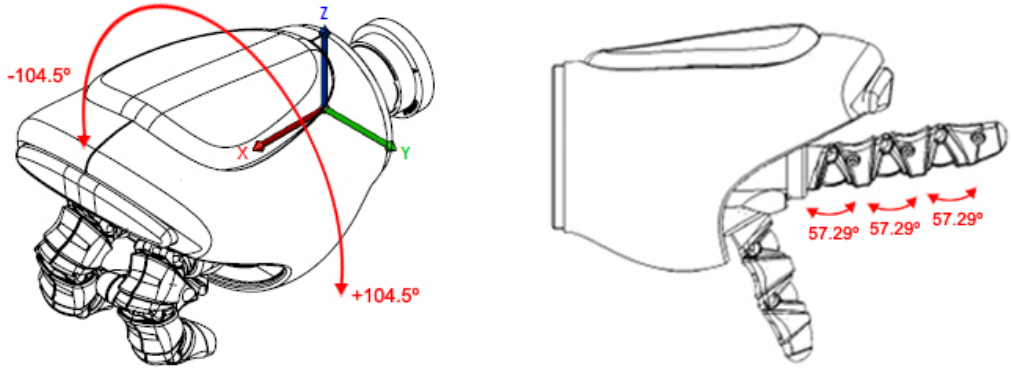

Figura 93. Detalle de la mano del robot NAO. Los grados de libertad son agarre y giro sobre X

En tercer lugar, he dotado al robot con capacidad para coger objetos mediante dedos prensiles acordes al modelo real, ampliando su número de grados de libertad hasta los 25 y con ello el abanico de posibles ensayos virtuales. Para ello he exportado de manera individual cada una de las falanges que componen cada dedo y he introducido articulaciones entre ellas según la documentación que se puede encontrar en la web de Aldebaran Robotics[66] (ver Figura 93). La Figura 94 expone una secuencia de cierre de una mano de mi modelo de robot NAO ejecutada en el editor de SkeletalMesh de UDK.

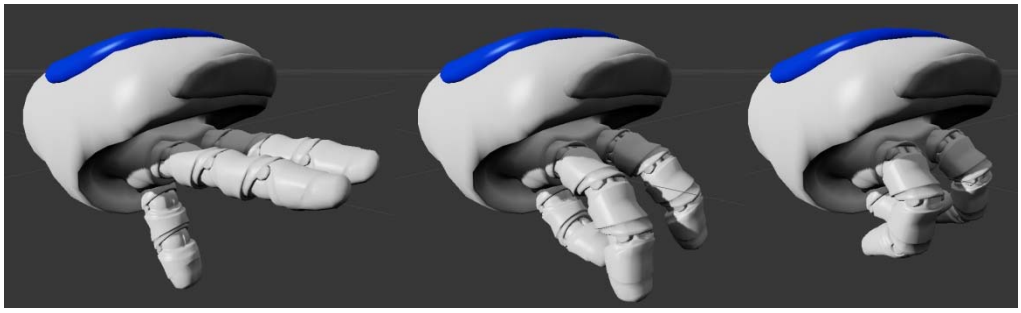

Figura 94. Secuencia de cierre de la mano de mi modelo de humanoide NAO en el editor de SkeletalMesh de UDK

[66] http://doc.aldebaran.com/1-14/family/nao_h25/links_h25.html

Como ya he indicado en el capítulo 5, los nuevos modelos de robots deben introducirse en USARSim a través de clases que los definan completamente, dejando constancia de las clases de robots generadas en el archivo *UDKUSAR.ini*. A la hora de crear una clase para el nuevo modelo virtual de NAO han surgido algunas dificultades basadas en la multitud de articulaciones presentes en los dedos del robot. Cada dedo superior consta de 3 articulaciones y cada pulgar tiene otras 2, lo que asciende a 8 articulaciones en cada mano. El robot real solo dispone de un mecanismo de apertura y cierre en cada mano, el cual se controla con un solo valor numérico, no pudiendo controlar la posición de cada falange por separado (el humanoide NAO solo tiene 25 grados de libertad). Las aplicaciones NAOqi y USARNaoQi que intervienen como middleware entre USARSim y las aplicaciones software para el control del robot (Choregraphe, PyNAOqi, etc.) trabajan con un grado de libertad para cada mano. Como los ensayos a realizar en el entorno del laberinto no requieren manipulación de objetos, he solucionado el problema creando varias clases de NAO, unas con las manos cerradas y otras con las manos abiertas con dedo inferior prensil. Si en el futuro surge la necesidad de simular ensayos donde este robot deba coger objetos con precisión valiéndose de las manos, sería necesario retomar el trabajo y seguir buscando nuevas soluciones en esa línea. Teniendo en cuenta que el nuevo modelo de NAO podría usarse para simular conjuntos de robots (trabajo cooperativo, competiciones virtuales entre alumnos como ligas de fútbol, etc.) y sería necesario distinguirlos, he creado diversas clases con funciones y colores diferentes (ver Tabla 11).

Nombre de la clase	**Descripción**
NaoBlue.uc	Modelo de NAO con dedo inferior prensil y color azul
NaoRed.uc	Modelo de NAO con dedo inferior prensil y color rojo
NaoV32Blue.uc	Modelo de NAO con la mano cerrada y color azul
NaoV32Red.uc	Modelo de NAO con la mano cerrada y color rojo
NaoV32Green.uc	Modelo de NAO con la mano cerrada y color verde
NaoV32Yellow.uc	Modelo de NAO con la mano cerrada y color amarillo
NaoV32Brown.uc	Modelo de NAO con la mano cerrada y color verde
NaoV32Red3p.uc	Modelo de NAO con una cámara posterior que enfoca al robot desde arriba para permitir vistas en tercera persona. Este modelo se describe con detalle en el capítulo 9 de esta tesis.

Tabla 11. Clases de robots humanoides NAO introducidas en el simulador USARSim para UDK

A continuación expongo unas líneas de código del archivo *NaoBlue.uc* en las que se muestra cómo he introducido las articulaciones de la muñeca y el dedo pulgar de la mano derecha del NAO. Los nombres escogidos para estas articulaciones *RWristYaw* y *RHand* se corresponden con aquéllos que utiliza el software de Aldebaran Robotics para este robot, de manera que resultan directamente utilizables. La Figura 95 (obtenida de la citada documentación web de Aldebaran Robotics) muestra todos los nombres de las articulaciones del robot NAO.

```
//-----------------------------------------------------------------------------------------------------------
        // Articulación de la muñeca derecha
//-----------------------------------------------------------------------------------------------------------
        Begin Object Class=RevoluteJoint Name=RWristYaw
                Parent=RLowerArm
                Child=RWrist
                InverseMeasureAngle=true
                Offset=(x=0.1365,y=0.0908,z=-0.076)
                LimitLow=-1.8151 // -104
                LimitHigh=1.8151 // 104
                Direction=(x=-3.14,y=1.571,z=-3.14)
                MaxForce=`MaxForceMotorType2
                Damping=`DampingMotorType2
        End Object
        Joints.Add(RWristYaw)
//-----------------------------------------------------------------------------------------------------------
        // Articulación del dedo inferior de la mano derecha
//-----------------------------------------------------------------------------------------------------------
        Begin Object Class=RevoluteJoint Name=RHand
                Parent=RWrist
                Child=RThumb
                Offset=(x=0.181,y=0.0908,z=-0.05)
                LimitLow=-1.8151 // -104
                LimitHigh=1.8151 // 104
                Direction=(x=1.571,y=-1.571,z=0)
                MaxForce=`MaxForceMotorType2
                Damping=`DampingMotorType2
        End Object
        Joints.Add(RHand)
//-----------------------------------------------------------------------------------------------------------
```

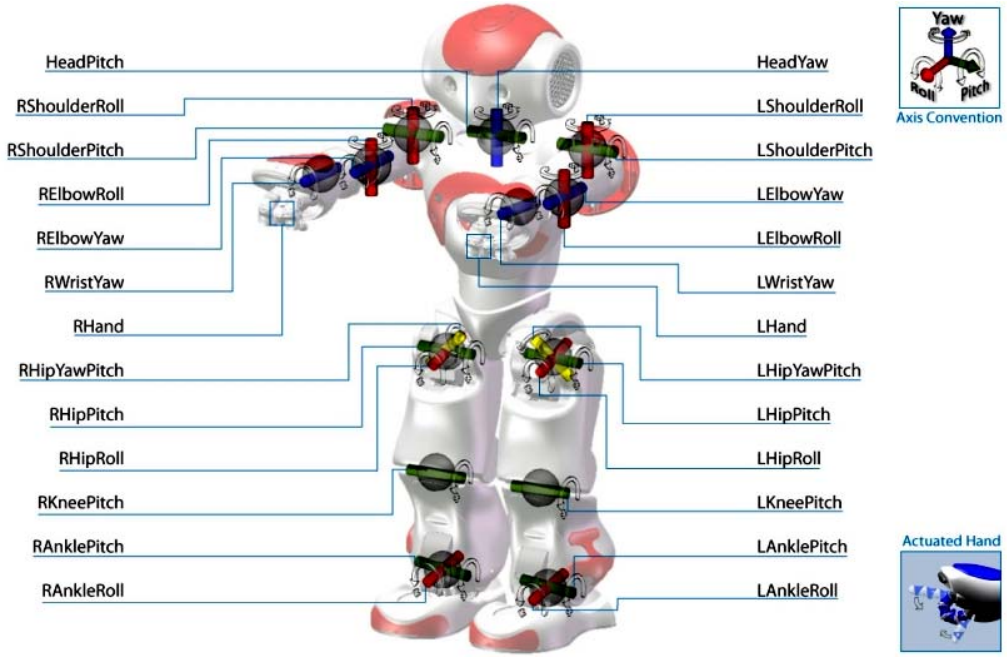

Figura 95. Grados de libertad y nombre de las articulaciones del robot humanoide NAO

6.3.5. Resultados en simulaciones de Sistemas Inteligentes con USARSim v.1.2 para UDK. Casos de estudio.

Los alumnos de robótica de la UJI han hecho uso de la descrita plataforma software durante sus sesiones prácticas, validando la buena adecuación tanto del lenguaje Python como del simulador, los entornos y el modelo de robot. A continuación expongo los resultados obtenidos durante la implementación en clase de dos casos de estudio sobre temas de Sistemas Inteligentes.

a) Caso de estudio sobre Aprendizaje Automático Avanzado: Redes Neuronales

En este ejercicio práctico, los alumnos de la asignatura Sistemas Inteligentes de tercer curso del Grado en Ingeniería Informática han puesto en práctica un clasificador de redes neuronales para un conjunto de puntos de referencia ubicados en un laberinto. Se trata de un caso de estudio donde un agente encarnado (robot NAO) dotado de una cámara de vídeo captura imágenes del entorno. La Figura 96 ilustra este caso de

estudio con la imagen captada por la cámara frontal del humanoide NAO ubicada arriba a la izquierda. El conjunto de datos formado por tales imágenes debe ser recogido para posteriormente entrenar la red neuronal con el algoritmo de propagación hacia atrás [150]. Después de un entrenamiento exitoso, el agente encarnado debe ser capaz de identificar los puntos de referencia y, en consecuencia, moverse hacia la salida del laberinto.

Tal como se observa en la Figura 77, cuatro tipos de puntos de referencia básicos se representan en color rojo brillante sobre las paredes del laberinto. Otros dos puntos de referencia adicionales señalan la entrada y la salida del mismo (ver Figura 75), pero estos dos puntos no tienen que ser aprendidos. Aunque el número de referencias es reducido, el problema resulta desafiante debido a las ya descritas variables condiciones de iluminación y otros efectos tales como humo que disminuye la visibilidad o refracciones provocadas por fuentes de calor.

Figura 96. Robot NAO ejecutando un algoritmo de clasificación de imágenes. El algoritmo ha sido implementado con Python y simulado con USARSim

Este ejercicio práctico es especialmente interesante desde el punto de vista educativo porque combina el procesamiento de imagen con el aprendizaje automático. Esto permite afirmar que este marco software combina perfectamente la visión, la robótica y el aprendizaje automático, proporcionando un multidisciplinario y enriquecedor enfoque para el estudiante. El entorno de simulación, por su parte, incorpora una notable cantidad de los elementos que facilitan simulaciones didácticas y atractivas.

Afortunadamente, existen poderosos paquetes de Python para el procesamiento de la visión que pueden facilitar la detección una referencia en la imagen captada por una cámara. La Figura 98 muestra un sencillo procedimiento para la extracción de una referencia que puede ser implementado de manera directa en Python haciendo uso de las librerías *numpy* y *scipy* [151].

En cuanto a la parte de aprendizaje automático, el conjunto de datos con las imágenes procesadas conteniendo las referencias se introduce en un perceptrón multicapa, el cual es entrenado con el mencionado algoritmo de propagación hacia atrás estándar. Esta implementación resulta también sencilla usando la librería *PyBrain* [152].

Los resultados de un entrenamiento típico se muestran en la Figura 97, la cual representa el error de clasificación en función del número de iteraciones de entrenamiento. En este ejemplo, la red neuronal constaba de 324 entradas (correspondientes a imágenes de 18x18 píxeles), 7 unidades ocultas y 4 salidas.

El conjunto de datos de imagen consistió en 30 imágenes de cada una de las cuatro clases, con un total de 120 imágenes. Esto se dividió en dos grupos de datos, uno de entrenamiento con 90 imágenes y otro de prueba con 30 imágenes (ver la leyenda del gráfico de la Figura 97 para los errores de entrenamiento y de prueba). La red resultante clasificó correctamente el 97% de las imágenes.

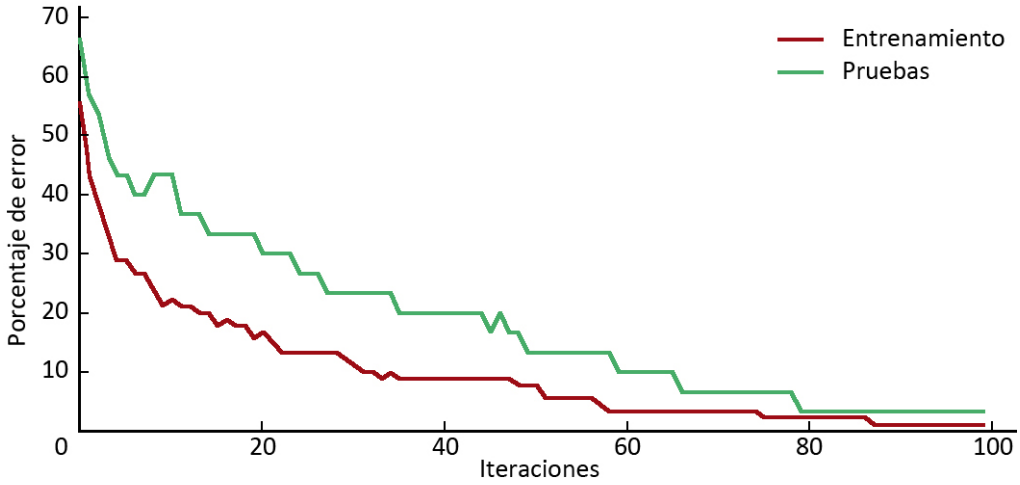

Figura 97. Evolución del error de clasificación de referencias durante el entrenamiento de la red neuronal

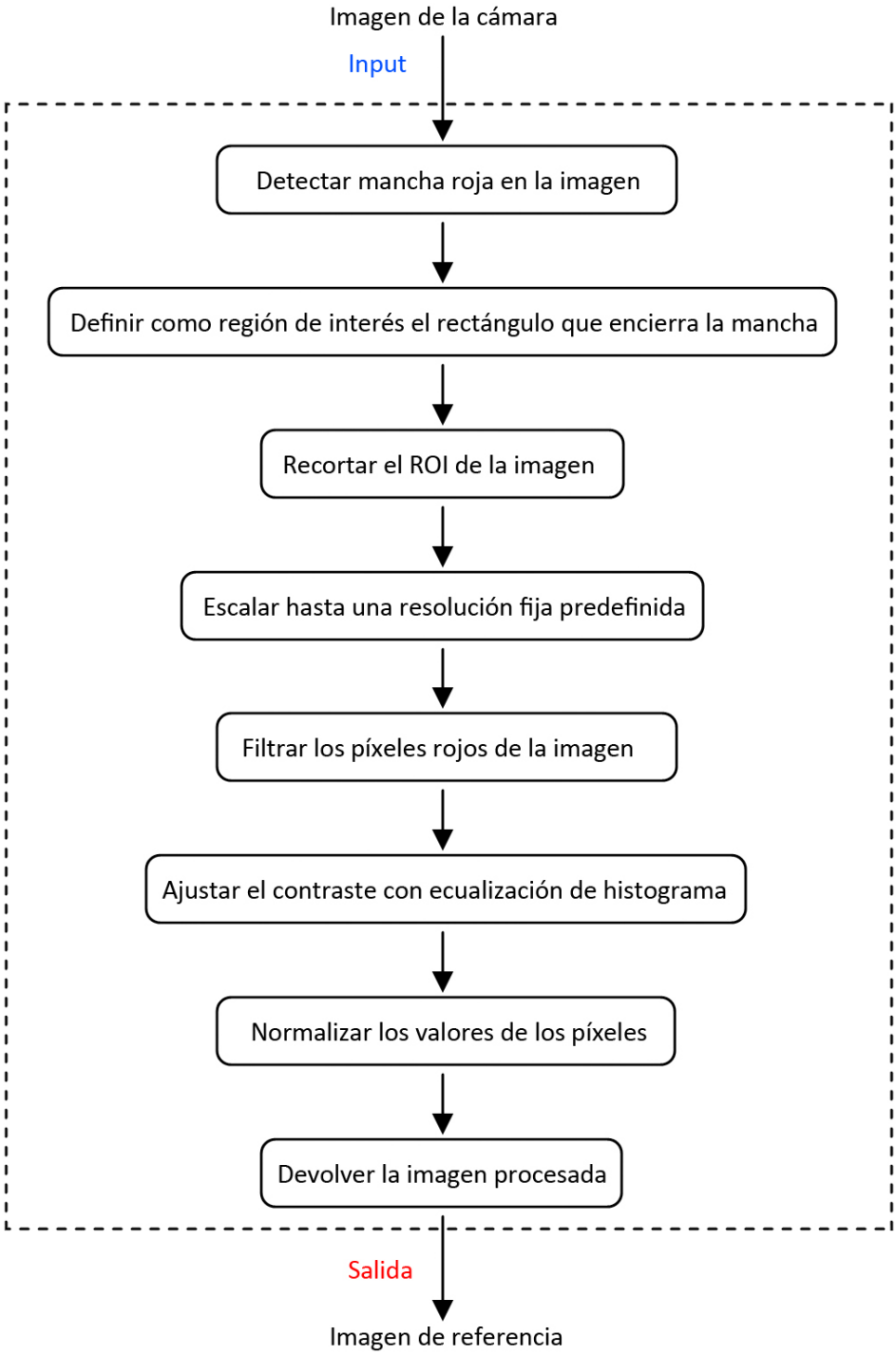

Figura 98. Algoritmo para la extracción de una referencia en la imagen captada por una cámara

b) Caso de estudio sobre Búsqueda Avanzada:

Algoritmos Genéticos

Controlar un robot bípedo con numerosos grados de libertad es una tarea difícil de realizar en entornos complejos, debido a que los robots humanoides requieren de comportamientos rápidos, estables y adaptativos. El objetivo de este ejemplo es el desarrollo de una solución para la generación automática de un modo de andar a gatas, inspirado en trabajos de investigación sobre Algoritmos Genéticos para la síntesis de un comportamiento para caminar [153].

Algunos movimientos que realizamos los humanos (y los robots humanoides) son inherentemente periódicos, repitiendo varias veces el mismo conjunto de pasos (por ejemplo, caminar, gatear, etc.). Estos movimientos se pueden expresar de forma matemática mediante una función periódica que puede descomponerse en una suma de osciladores simples, tal como se muestra en la Ecuación 4.

$$f(t) = C + \sum_{n=1}^{N} A_n \sin\left(n\frac{2\pi}{T} + \varphi_n\right)$$

Ecuación 4. Función periódica obtenida como resultado de una suma de osciladores simples

Donde N es el número de frecuencias, C es el desplazamiento, A_n = 1..N son amplitudes, T es el período y Φ_n = 1..N son fases. Aplicando estos osciladores a cada articulación es posible generar un gateo, el cual puede ser implementado y probado con un robot humanoide como el NAO.

La Figura 95 muestra la estructura del humanoide NAO con los ejes de referencia de sus articulaciones. La idea principal detrás de la definición de este gateo es colocar un oscilador en cada articulación que interviene en el movimiento con el fin de definir su trayectoria. Siguiendo este criterio, los osciladores se colocan en las siguientes articulaciones del lado derecho (R) e izquierdo (L): *ShoulderPitch*, *ShoulderRoll*, *ElbowRoll*, *HipPitch*, *HipRoll* y *KneePitch*. Por tanto, se utilizan 12 osciladores de frecuencia única. Dado que cada oscilador de frecuencia única tiene 4 parámetros para definir, se necesitan 48 parámetros para determinar completamente el modo de gatear. En casos como éste, es común asumir una simetría según un plano sagital, el cual determina los mismos movimientos de las articulaciones correspondientes a los lados izquierdo y derecho con un desplazamiento de fase de medio período. Además,

el período de todos los osciladores debe ser el mismo para mantener todas las articulaciones sincronizados por un único reloj de frecuencia. Estas consideraciones reducen el número de parámetros a 19. Si los parámetros se definen para las articulaciones del lado izquierdo, aquéllos correspondientes a las articulaciones del lado derecho pueden ser fácilmente obtenidos: para articulaciones definidas como *roll* ambos lados (derecho e izquierdo) realizan las mismas trayectorias a lo largo del tiempo; sólo es necesario cambiar el signo del desplazamiento. Para articulaciones definidas como *pitch*, el lado derecho se puede conseguir mediante la adición de una fase π al oscilador correspondiente. El conjunto de parámetros desconocidos forman el genoma que será utilizado por el algoritmo genético para generar el gateo.

La Figura 99 representa el robot NAO en posición para gatear, mientras que los resultados de un proceso evolutivo típico se muestran en la Figura 100. Éste es un experimento muy sencillo con una pequeña población de sólo 10 individuos y sólo 3 generaciones. El resto de los parámetros fueron elegidos como en los mencionados trabajos de investigación sobre Algoritmos Genéticos, a saber:

- Método de la ruleta para la selección.
- Método de rango real para la mutación, con una probabilidad definida por pm = 0,5.
- Método uniforme para cruce.
- Fracción de la población creada por cruce definido por pc = 0.8.

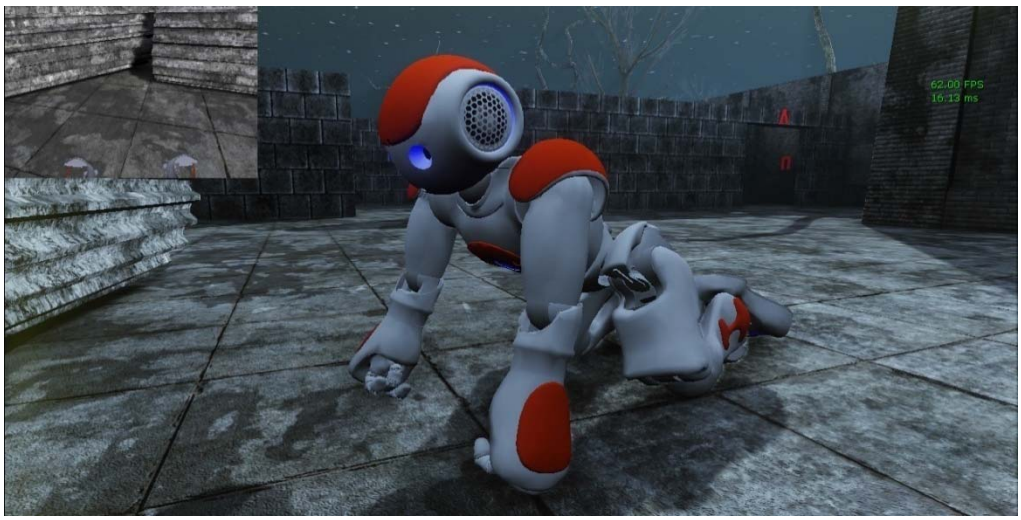

Figura 99. Humanoide virtual NAO en posición de gateo implementando un algoritmo genético

La función de aptitud se define por la distancia recorrida por el robot mientras gatea durante una cantidad de tiempo determinada e invariable (5 segundos en este caso). El gráfico de la Figura 100 representa la evolución de la función de aptitud con el número de generaciones. En sólo tres generaciones, la puntuación media casi se ha duplicado. La aptitud máxima alcanzada por un individuo ha alcanzado 0,54 metros recorridos durante una prueba de 5 segundos.

Es importante tener en cuenta que estos resultados fueron obtenidos por estudiantes de Sistemas Inteligentes durante una sesión de laboratorio. Por lo tanto, no puede afirmarse que supongan una mejora sobre el estado del arte en el control de gateo de robots, pero el objetivo es demostrar que los estudiantes son capaces de llegar a experiencias significativas, utilizando el marco de software y simulación presentado.

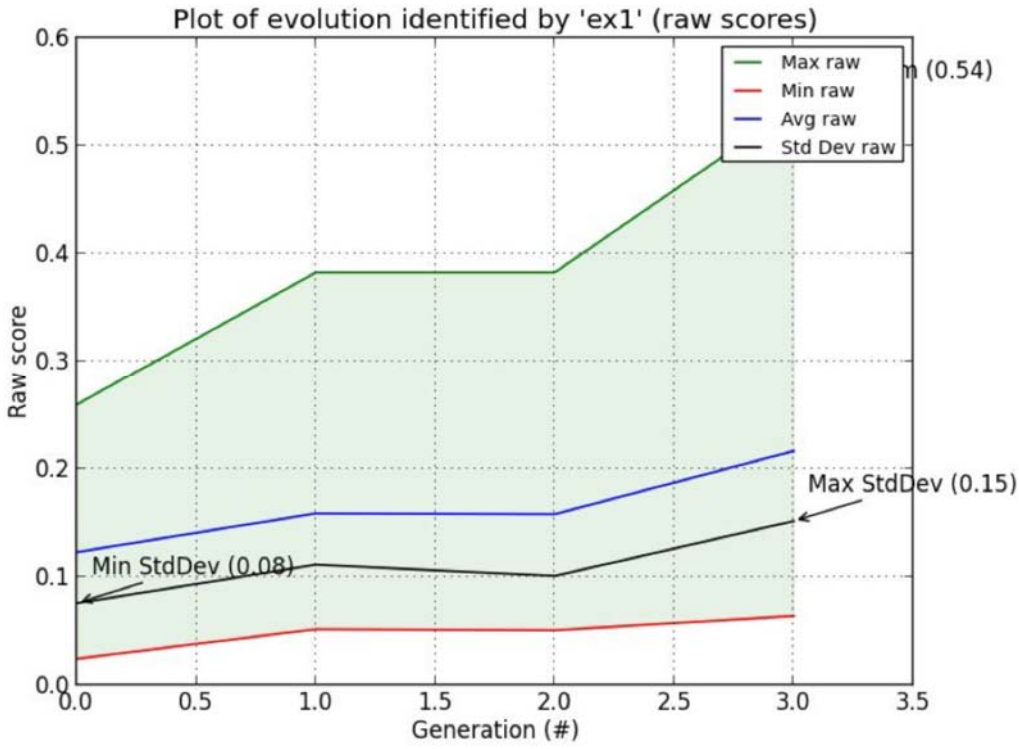

Figura 100. Evolución de la función aptitud frente a generaciones durante el proceso evolutivo

6.4. Conclusiones

Sistemas Inteligentes es un área que está ganando relevancia en la actual revisión de

los planes de estudios de Informática. En este capítulo se ha presentado un simulador para robótica capaz de integrarse perfectamente con una plataforma de desarrollo de software basada en Python, la cual aborda los principales temas de Sistemas Inteligentes, proporcionando software listo para ser utilizado por los alumnos en el aula y en el laboratorio para la implementación directa de problemas en esos temas.

Los entornos virtuales descritos en este capítulo proporcionan una simulación realista, ofreciendo una funcionalidad excepcional y una estética atractiva. La intención es motivar a los estudiantes a pasar más tiempo con el simulador para implementar algoritmos de última generación mediante el uso de los recursos y librerías existentes. Como se ha comprobado en la elaboración de estos entornos, USARSim resulta excelente para simulaciones donde la visión juega un papel fundamental, ofreciendo una amplia y completa gama recursos visuales muy difíciles de encontrar en los simuladores de robótica actuales. Su motor de juego resulta en este sentido totalmente imprescindible, acercando todavía más los ensayos virtuales a los reales.

El decreciente coste de potentes tarjetas gráficas combinado con este software de simulación permite que los entornos más exigentes puedan diseñarse para encarnar una gran cantidad de algoritmos que hoy en día se enseñan de una manera abstracta. Esto evita a su vez el coste económico y la carga extra de trabajo que supone mantener una flota de robots reales en condiciones operativas. Además, todo el software propuesto en este capítulo tiene la ventaja de ser totalmente gratuito para uso no comercial, fácilmente instalable y apoyado por una extensa documentación online, lo que permite a cualquiera acceder a ensayos con robots por el coste de una conexión a Internet.

Como aspecto mejorable de esta plataforma software cabe destacar la multiplicidad de aplicaciones informáticas que la componen, cuya instalación y configuración inicial puede confundir o desanimar a algún potencial usuario.

Capítulo 7

SIMULACIÓN DE COMPETICIONES ROBÓTICAS

En un marco global donde los concursos de proyectos de robótica son cada vez más comunes, es conocido [154] que las competiciones pueden resultar una herramienta importante para promover la madurez intelectual, tal como lo define el modelo de Perry [155]. Una competición implica un problema perfectamente definido, pero a su vez abierto a muchas posibles soluciones. En las competiciones se anima a los estudiantes a trabajar colaborativamente en equipo y las metas a alcanzar proporcionan un aspecto contextual óptimo para la aplicación de sus conocimientos.

7.1. Sobre la enseñanza de la robótica en la UJI

La experiencia del personal docente de la Universidad Jaume I (UJI) de Castellón impartiendo asignaturas relacionadas con la robótica demuestra que la introducción de atractivas plataformas robot en las clases y las competiciones robóticas son medios efectivos para captar el interés de los estudiantes por los contenidos de informática [23].

Desde 1993 hasta la actualidad los profesores del Departamento de Ingeniería y Ciencia de los Computadores de la UJI han observado una reveladora evolución en el interés de los alumnos por una asignatura optativa de robótica, la cual está estrechamente relacionada con el uso de robots en clase y las competiciones robóticas [1].

Entre 1993 y 1998 tal asignatura estaba orientada a la programación y la ingeniería del software, estando el trabajo de laboratorio limitado al uso de simuladores de brazos manipuladores. Con la llegada de asequibles kits de robots móviles, en 1999 se hizo viable el uso de robots reales en el aula. Así, se introdujeron prácticas de laboratorio con pequeños robots móviles y se promovió una competición de sumo

entre los estudiantes (ver Figura 101). No es necesario decir que la competición aumentó enormemente el interés de los estudiantes en su trabajo de laboratorio. Dado que fue el primer evento de este tipo en la universidad, levantó un gran interés no solo entre los estudiantes de informática, sino en toda la comunidad educativa. Este interés también fue difundido en los medios de comunicación, con varias referencias en la prensa local.

Figura 101. A la izquierda un estudiante programando un pequeño robot móvil en el laboratorio. A la derecha competición de robots luchadores de sumo en la UJI, curso académico 1999-2000

Como se muestra en la Figura 102, el porcentaje de estudiantes que eligieron la asignatura de robótica casi se duplicó a partir del año 2000, y continuó aumentando hasta alcanzar un inédito 26% en el año 2002. Estas cifras representan un aumento de inscripción aproximadamente del 100% sobre el valor medio correspondiente a las ediciones previas a la utilización de pequeños robots móviles en el aula.

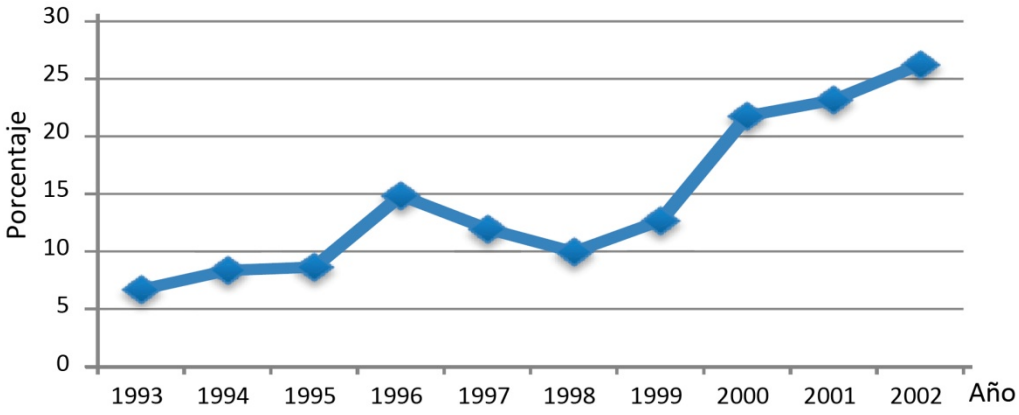

Figura 102. Porcentaje de estudiantes de informática de la UJI matriculados en la asignatura optativa de robótica entre los años 1993 y 2002. Planes de estudio de 1991

Después de algunos cursos con un moderadamente creciente interés de los estudiantes por cursar la asignatura de robótica, en 2006 se introdujo una nueva plataforma robótica en el aula que supuso un nuevo impulso en la motivación de los alumnos por la robótica. Se trataba de un pequeño robot humanoide altamente autónomo con el que los estudiantes debían aprender a programar comportamientos sencillos, poniéndolos posteriormente a prueba en una competición de sumo entre compañeros de clase. Además, el ganador de esta competición local se clasificaría para el campeonato nacional contra otras universidades españolas (ver Figura 103).

Figura 103. A la izquierda estudiante programando un robot humanoide en el laboratorio. A la derecha combate de robots humanoides en la competición CEABOT'08 entre los equipos de la UJI y la UHU

Esta vez, la noticia se difundió no sólo en los periódicos, sino también en la televisión y la radio. Como resultado, la tasa de matrícula creció significativamente en 2007 y los años posteriores, logrando una cifra sin precedentes de 46% en 2009, y manteniéndose superior al 40% en los años sucesivos, lo que representa aproximadamente tres veces la proporción de las ediciones anteriores. Otro factor importante de este aumento podría ser que el equipo de la UJI ganó el concurso nacional durante tres ediciones consecutivas.

Las cifras anteriores todavía resultan más sorprendentes y significativas si tenemos en cuenta que desde hace diez años existe una tendencia global decreciente en el interés de los estudiantes por cursar estudios en informática y STEM en general (ver Figura 104), tal como ya se ha comentado en el capítulo 1 de esta tesis.

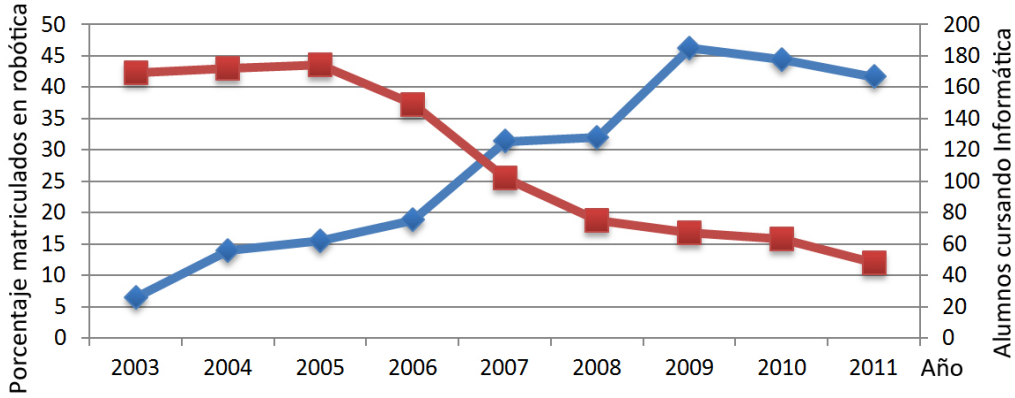

Figura 104. Porcentaje de estudiantes que cursan la asignatura optativa de robótica (rombos azules) y número absoluto de estudiantes matriculados en la carrera Informática (cuadros rojos). Planes de estudio de 2001

No obstante existe una contrapartida a los buenos resultados obtenidos con las sesiones prácticas y las competiciones con robots en cuanto al interés y motivación de los estudiantes. El profesorado de la asignatura de robótica ha experimentado una sobrecarga de trabajo en los últimos cursos académicos por la gran cantidad de horas necesarias para la configuración y el mantenimiento de una flota de pequeños robots en condiciones operativas. Por este motivo, el departamento de Ingeniería y Ciencia de los Computadores de la UJI aboga por el uso de simuladores y competiciones de robots virtuales en la enseñanza [1]. Muchos beneficios de la competición con robots también pueden ser obtenidos mediante entornos virtuales, como lo demuestra la capacidad educativa y el atractivo de los videojuegos. Hoy en día, son posibles entornos virtuales con gran realismo en un ordenador estándar y existen multitud de herramientas software de libre acceso para la creación de un laboratorio de robótica o competición virtual. La simulación fidedigna de la física hace que la programación de robots virtuales suponga un reto tan difícil como la programación de robots reales, mientras se reduce el trabajo de mantenimiento al mínimo. Además, como ya se ha comentado en capítulos anteriores, los mundos virtuales permiten la introducción de mejoras de todo tipo tales como elementos fantásticos que enriquecen la experiencia de simulación, por lo que se espera que los estudiantes puedan disfrutar del curso, haciendo la robótica atractiva para ellos.

Otro aspecto importante que ha contribuido a esta apuesta por el regreso a la simulación es facilitar la educación a distancia en robótica. No hay que olvidar que la formación a distancia es una vía que ha demostrado ser muy útil para extender el

interés por la ciencia y el conocimiento en general y puede servir, a su vez, para contrarrestar la decreciente tendencia actual en estudios de este tipo. Los simuladores pueden ser descargados de Internet por los estudiantes e incluso permiten ser programados online sin necesidad de tal descarga mediante herramientas web para robótica como las *robot web tools* [49].

En síntesis, se espera que el uso de simulaciones realistas y las competiciones virtuales de robots proporcionen el mismo atractivo y posibilidades de aprendizaje que las plataformas robóticas hardware, minimizando la cantidad de trabajo técnico necesario para la preparación del curso y facilitando la formación a distancia.

7.2. Marco de competiciones robóticas simuladas

Existen multitud de posibles competiciones que se pueden plantear como ejercicio estimulante para que los alumnos pongan a prueba sus conocimientos de robótica, inventando algoritmos y estrategias que permitan derrotar a sus oponentes. Son especialmente famosas las competiciones virtuales de fútbol como RoboCup Soccer Simulation League [156] o las de búsqueda y rescate de víctimas en entornos de desastres como RoboCup Rescue [157]. En los últimos años están tomando cada vez mayor relevancia los ensayos en los que se mide la capacidad de un robot para desempeñar tareas cotidianas tanto dentro de una vivienda como en el exterior. Esta idea se ha trasladado a las competiciones de robots simulados en la forma de RoboCup @Home [156]. También podemos encontrar novedosas competiciones robóticas como RoboCup @Work[67], donde se premia el diseño innovador de plataformas robot y manipuladores para realizar trabajos en cooperación con trabajadores humanos en un ambiente industrial. Se trata, en cualquier caso, de eventos que ayudan a fomentar y difundir el interés por la ciencia en general y por la robótica en particular, proporcionando a los participantes un apasionante clima de trabajo en equipo.

Aunque el abanico las posibilidades es amplio, dada la tradición heredada de las competiciones entre robots reales en la Universidad Jaume I de Castellón, la primera idea que surgió al plantear este tipo de actividad fue la de una lucha de sumo entre robots. No obstante, hay que tener en cuenta que cuando se plantea una competición virtual desaparecen muchas de las restricciones que limitan qué se puede llevar a cabo en el entorno de la universidad y qué no. Así, una lucha de sumo con robots reales es

[67] http://www.robocupatwork.org/

relativamente fácil de implementar porque solo requiere de un pequeño espacio plano de color claro limitado por una línea circular de color oscuro y un par de robots. En cambio, un partido de fútbol con robots humanoides al estilo de RoboCup Soccer Standard Platform[68], (ver Figura 105) plantea un reto mucho mayor, ya que para un evento así serían necesarios dos equipos de 5 robots cada uno, dos porterías estandarizadas y un terreno de juego de 7,4 metros de largo por 5,4 metros de ancho entre otras muchas cosas. Esto no solo supondría un coste económico elevado en robots hardware, sino que también sería necesario disponer de un espacio extenso y diáfano donde poder ubicar el campeonato.

Figura 105. Competición de fútbol entre robots reales RoboCup Soccer Standard Platform League[69]

Las competiciones virtuales son ajenas a casi todas las exigencias económicas, temporales y espaciales que dificultan o imposibilitan llevar a cabo muchas competiciones con robots reales. Esto permite expandirse a nuevas experiencias que planteen toda clase de nuevos retos a los estudiantes. Con esta filosofía he desarrollado diferentes entornos de simulación, los cuales permiten albergar competiciones de diversa índole en las que los alumnos de robótica pueden implementar algoritmos variados con distintas plataformas robóticas (ver Tabla 12).

Para poder comparar de manera sencilla las capacidades de los tres simuladores bajo estudio, he implementado tres veces el entorno con más probabilidades de ser

[68] http://www.informatik.uni-bremen.de/spl/bin/view/Website/Downloads
[69] http://www.oregonlive.com/today/index.ssf/2013/06/netherlands_hosts_robocup_thou.html

explotado por los alumnos de la Universidad Jaume I. Así, he desarrollado entornos para lucha de sumo tanto para Webots como Gazebo y USARSim.

Entorno	Competición virtual simulada
Edificio TI de la Universidad Jaume I desarrollado con USARSim	Campeonato de lucha de sumo entre robots Pioneer 3 AT
Ring de Sumo desarrollado con Webots	Campeonato de lucha de sumo entre robots Pioneer 3 AT
Ring de Sumo desarrollado con Gazebo	Campeonato de lucha de sumo entre robots Pioneer 3 AT
Ring de Boxeo desarrollado con USARSim	Campeonato de lucha de boxeo entre robots humanoides NAO
Entorno doméstico desarrollado con USARSim	Campeonato de fútbol entre robots humanoides NAO

Tabla 12. Relación de entornos para la simulación de competiciones robóticas entre estudiantes

7.3. Lucha de sumo. Entornos virtuales para competiciones robóticas

En este apartado describo los entornos para competiciones de lucha de sumo desarrollados con Webots, Gazebo y USARSim.

7.3.1. Lucha de sumo con Webots 7

Para que tenga lugar una lucha de sumo simulada entre robots autónomos móviles es necesario un entorno con un ring adecuado y dos robots con características similares que puedan distinguirse visualmente con facilidad. Webots 7 permite obtener entornos de simulación y modelos de robots personalizados de manera sencilla utilizando un entorno gráfico simple e intuitivo. No obstante, los procesos para la obtención de un entorno y un robot son muy diferentes entre sí, motivo por el que describo el entorno del ring de sumo separadamente del robot Pioneer 3 AT.

`Ring de sumo`

Un entorno virtual destinado a albergar una competición de sumo puede entenderse relativamente sencillo, pues a priori no es necesario más que un círculo bordeado por una gruesa línea de color oscuro. Sin embargo, para ganar en realismo mejorando la sensación de juego y el grado de inmersión de los estudiantes, he ubicado la competición en un pequeño terreno de juego. De este modo, se evita la sensación de que el ring se encuentra flotando en el espacio exterior.

Con las luchas de sumo de robots reales a veces resulta complicado saber si uno de los robots ha atravesado la línea exterior del ring y por tanto ha perdido el combate. Aprovechando las ventajas inherentes a los mundos virtuales e intentando evitar tal situación de incertidumbre, he diseñado un ring de sumo de manera que resulte inmediato saber cuándo ha terminado el combate y, por tanto, hay un ganador. La lucha tiene lugar sobre un cilindro de 4 metros de diámetro y una altura de 50 centímetros colocado sobre el suelo de parquet de lo que podría entenderse como una pista de deportes (ver Figura 106). De este modo, cuando un robot es empujado fuera del círculo de lucha, perdiendo el combate, cae del ring al suelo resultando muy vistoso y evidente que el combate ha terminado. Esto no es recomendable ni operativo hacerlo con robots reales por motivos obvios. El área para el combate la he dimensionado de manera que dos robots del tipo Pioneer 3 AT o similar puedan moverse y maniobrar con facilidad.

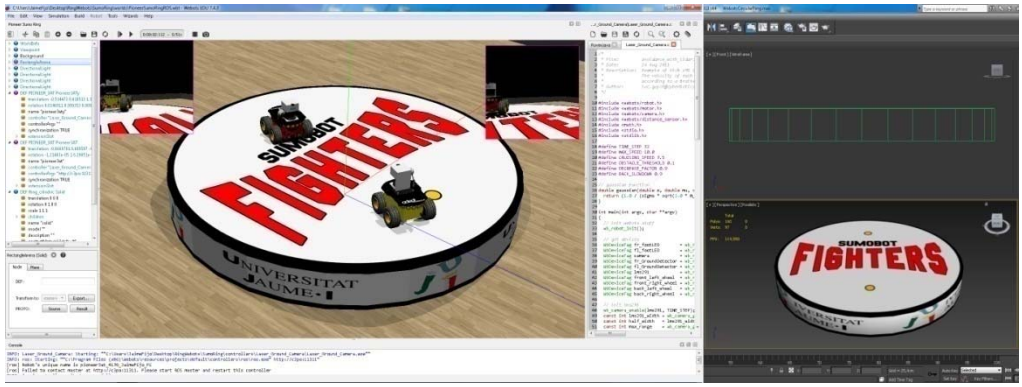

Figura 106. A la izquierda una captura de pantalla de una simulación de un combate de sumo entre dos robots Pioneer 3 AT con Webots. A la derecha el ring de sumo visto en 3D Studio Max

El ring de sumo y su entorno lo he modelado con 3D Studio Max de manera que el número de polígonos resulte reducido (solo unos 160 triángulos para el ring). Los objetos obtenidos con 3D Studio Max los he exportado a formato WRML97 (*.WRL) debido a que Webots solo puede importar objetos 3D que se encuentren en ese formato. Esto supone una limitación importante a la hora de incorporar objetos 3D a los entornos virtuales creados con este simulador. WRML97 no es un formato muy común cuando se habla de objetos sueltos, sino que suele usarse para guardar escenas completas. Además, esto significa que desde Webots no es posible importar animaciones, tales como brazos robot trabajando en un ambiente industrial o gente paseando, lo que reduce de manera notoria las posibilidades de este simulador para

recrear escenas con realismo. La textura del área de lucha la he creado con Adobe Photoshop buscando un aspecto funcional y realista. Se trata de un círculo blanco con dos marcas en amarillo que indican los puntos de salida de los robots contrincantes. El círculo está bordeado por una gruesa línea de color negro que delimita el ring visualmente y puede ser detectada por los sensores de los robots. Para ganar en realismo y publicitar la competición he añadido en el centro del ring un llamativo rótulo donde se lee "sumobot fighters" y el logo de la UJI en la superficie lateral. La textura de parquet de la pista la he obtenido de Internet y le he aplicado un mapeado para que se repita sobre el plano que simula el suelo. En cuanto a la importación de texturas, Webots resulta muy versátil, permitiendo importar imágenes en los formatos más usuales.

En cuanto a las envolventes de colisión del ring y del suelo de la pista, he de decir que ha resultado muy sencillo obtenerlas con Webots, pero en absoluto gracias a las facilidades que ofrece este simulador, sino porque solo se trata de un cilindro y un plano. La manera definir y ubicar envolventes de colisión en Webots resulta, en general tediosa, ya que acostumbra a requerir repetidos cálculos matemáticos y no resulta fácil determinar si se están dimensionando y situando correctamente. El procedimiento consiste básicamente en definir una forma geométrica con sus dimensiones y su ubicación mediante coordenadas cartesianas con la ayuda única de una perspectiva en 3D, lo que termina convirtiéndose en un proceso largo para objetos con muchas envolventes de colisión.

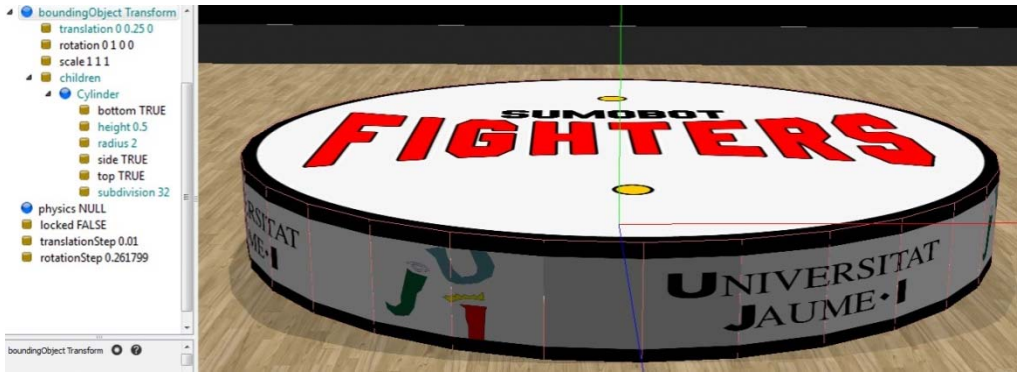

Figura 107. Contorno de colisión del ring de sumo representado por un cilindro con aristas en rosa

La Figura 107 muestra la envolvente de colisión del ring de sumo como un cilindro de 0,5m de altura y 2m de radio representado por sus aristas en color rosa. A la izquierda

de la imagen se ve cómo se define tal envolvente (*boundingObject*) por medio de su ubicación (traslación y rotación), su escala y su forma cilíndrica.

Modelo de robot Pioneer 3AT con Webots 7

Para llevar a cabo una competición de robots luchadores de sumo es necesario determinar previamente qué plataforma robótica se adapta bien a las necesidades del combate. Las singularidades de una lucha de sumo recomiendan el uso de robots dotados con las siguientes características:

- Movilidad mediante ruedas o cadenas para poder arrastrar al oponente fuera del ring.
- Sensor laser para obtener la distancia al oponente o los límites del ring a distancias medias o largas.
- Cámara de video para identificar tanto al oponente como los límites del ring.
- Red de sensores sonar para advertir la presencia del adversario o del suelo a distancias cortas.

Existen multitud de plataformas robóticas que con pocas o ninguna modificación cumplirían los citados requisitos, pero como en este caso se trata de preparar una competición simulada, lo más operativo es utilizar un modelo de robot que sea conocido por los alumnos y, además, esté incluido en el simulador a utilizar.

Figura 108. A la izquierda se muestra un modelo de robot Pioneer 3AT incluido en Webots. A la derecha está el modelo modificado con cámara y sensores para detectar presencia de suelo

El robot Pioneer 3AT de ActivMEDIA Robotics [57] reúne casi todas las características necesarias para una lucha de sumo y, además, es una de las plataformas robóticas

más comunes en los simuladores. Tanto Webots como Gazebo o USARSim incluyen un modelo físico de Pioneer 3AT.

El modelo de robot Pioneer 3AT que incorpora Webots 7 consta de un equipamiento básico de 16 sensores de distancia sonar. A este robot es posible añadirle otras funciones mediante el siguiente hardware adicional incluido también en Webots 7:

- Sensor de distancia laser Sick LMS291
- Sensor de distancia laser Hokuyo modelos URG-04LX, URG-04LX-UG01 y UTM-30LX
- Estéreo cámara Kinect
- Pinza Pioneer3 Gripper
- Sensores o actuadores de uso general u otros creados por el usuario

Con el objetivo de garantizar la máxima cobertura sensorial y facilitar la detección del oponente y los bordes exteriores del ring, al modelo básico de Pioneer 3AT de Webots le he añadido el sensor laser Sick LMS291, dos sensores de distancia sonar orientados de manera oblicua al suelo y una cámara personalizada (ver Figura 108).

He creado dos sensores sonar para Webots cuya única función es dar información de la presencia o ausencia de suelo frente al robot. Estos sensores sonar los he ubicado en la parte frontal del robot inclinados 45 grados hacia el suelo, permitiendo al robot conocer si existe continuidad en el terreno sobre el que se mueve. A estos sensores les he dado forma de huella para que el usuario deduzca rápidamente cuál es su función. Además, para que el usuario reciba realimentación visual en tiempo real de las lecturas de estos detectores de suelo, he programado en C++ un funcionamiento que ilumina en rojo la forma de huella del sensor cuando no hay continuidad en el suelo o la deja apagada en gris si existe tal continuidad.

La Figura 109 muestra a la izquierda los sensores de suelo, estando uno de color rojo avisando de falta de continuidad en el terreno. A la derecha de la imagen se puede ver una vista lateral del robot con la orientación de estos sensores respecto a la vertical.

La cámara que he creado para el robot Pioneer 3AT es en su base una modificación de aquélla que incorpora Webots para uso general. Hay que considerar que el sensor cámara incluido en Webots 7 no tiene correspondencia visual, no siendo posible

determinar a simple vista si un robot puede captar imágenes de video. Para que el usuario pueda identificar fácilmente tanto la presencia de la cámara como su posición y orientación le he dado aspecto de ojo. Esta apariencia busca a su vez la inclusión de elementos fantásticos para aumentar la motivación de los usuarios por utilizar el simulador.

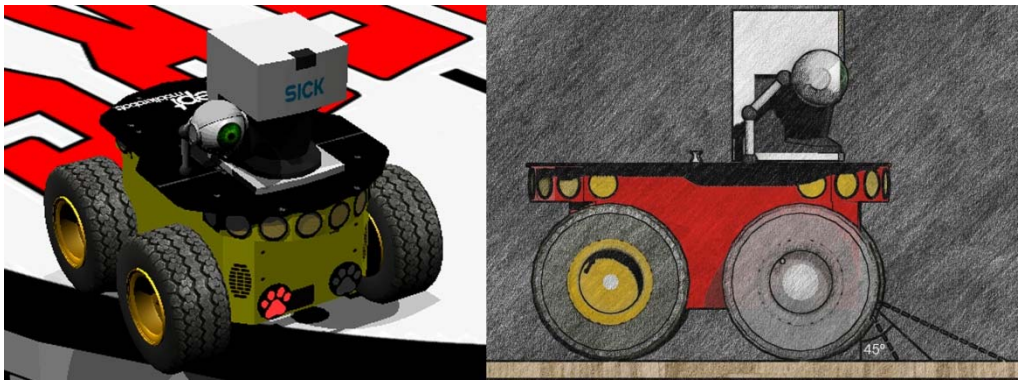

Figura 109. Vistas de los sensores sonar para la detección de presencia de suelo en un Pioneer 3AT

El modelado 3D de los sensores de suelo y de la cámara los he realizado con 3D Studio Max y los he tratado de manera similar a la ya comentada para el ring de sumo. Tanto la cámara como los sensores los he ajustado de manera que su número de polígonos no resulte elevado, manteniendo un buen aspecto cuando se los observa de cerca. En este sentido, hay que destacar que Webots 7 no permite el uso de objetos con diferentes niveles de detalle (LODs), lo que ha dificultado el diseño de la cámara, quedando ésta con 4114 polígonos. En cuanto a las colisiones de estos sensores, he estimado que no revisten importancia ninguna para la competición de sumo y, por simplicidad, no los he dotado de envolventes de colisión.

Para finalizar, es importante señalar que la naturaleza de un combate de sumo requiere que haya dos robots de iguales características, pero visualmente diferentes. Para ello he creado con Adobe Photoshop una nueva textura semejante a la del robot original, pero de color amarillo. Después he definido una nueva clase en Webots 7 llamada P3ATy (Pioneer 3AT yellow) que es una modificación de la clase P3AT incluida en el simulador, pero con esta nueva textura en su código.

7.3.2. Lucha de sumo con Gazebo 2.2

El extendido uso de ROS en la Universidad Jaume I y su sencilla interacción con Gazebo 2.2 han hecho más que recomendable el desarrollo de un entorno de competición en este simulador para los estudiantes de robótica. Dadas las plataformas robóticas incluidas en Gazebo 2.2 y las intenciones de esta investigación, he considerado interesante realizar un ring de sumo de similares características al desarrollado para Webots 7. De este modo la comparación de los resultados obtenidos con diferentes simuladores resulta más evidente. Por contra, realizar un segundo ring para lucha de sumo no amplía el abanico de posibilidades a la hora de organizar diversas competiciones durante el curso, pero ésta es una cuestión que he resuelto con USARSim, como explico más adelante en este capítulo.

Ring de sumo

El entorno para la competición simulada de sumo consiste en un ring cilíndrico, un suelo de parquet, cuatro postes con focos y cuatro vallas publicitarias que delimitan el recinto (ver Figura 110).

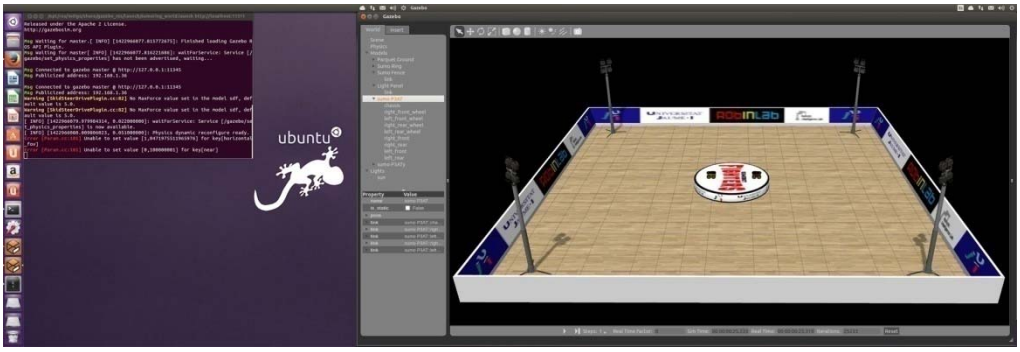

Figura 110. Entorno Linux con Gazebo 2.2 mostrando el ring para una competición de robots Pioneer 3AT luchadores de sumo

El Ring de sumo es el mismo que he usado en el entorno para Webots 7 sin ninguna modificación en diseño, pero con diferencias importantes en otros aspectos. Así, para exportar el ring en condiciones de ser utilizado en Gazebo 2.2 solo es necesario guardar el modelo 3D en formato collada (**.dae*) por un lado y las texturas en formato jpeg (**.jpg*) por otro. Esta manera de proceder resulta bastante práctica al tratarse de formatos muy habituales cuando se trabaja con software de CAD. Sin embargo, una vez guardados los archivos correspondientes al objeto 3D y a las

texturas en sus convenientes carpetas de Gazebo, todavía deben realizarse algunas operaciones imprescindibles para un correcto funcionamiento. Por una parte, hay que editar el archivo `*.dae` de forma que su código apunte a la ruta correcta para encontrar todas sus texturas, tal como se ejemplifica seguidamente mediante un segmento del archivo `Circular_Ring.dae` correspondiente a la malla 3D del ring.

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.1">
...
<library_images>
        <image id="Map #1595-image" name="Map #1595">
                <init_from>sumoring.jpg</init_from>
        </image>
 </library_images>
```

Por otra parte, es recomendable modificar el archivo de materiales de Gazebo 2.2 `gazebo.material` definiendo un nuevo material por cada textura que se añada a las simulaciones. Si no se lleva a cabo este último paso Gazebo 2.2 da la impresión de funcionar perfectamente, pero no se proyectan sombras sobre las texturas no codificadas en el mencionado archivo. Este error es tan frecuente que puede observarse una falta de sombras en la mayoría de videos de simulaciones con Gazebo presentes en YouTube. Aunque la presencia de sombras en las simulaciones es considerada innecesaria en ocasiones, la realidad es que sin ellas la experiencia del usuario se degrada considerablemente, perdiéndose realismo y la referencia de distancia al suelo. Así, cuando un objeto no proyecta sombra alguna sobre el terreno es imposible determinar su ubicación exacta en la escena, resultando muy difícil saber incluso si está en contacto con el suelo o flotando en el aire.

La Figura 111, extraída de un documento de Han-Wei Shen[70], ejemplifica los problemas de ubicación que surgen por ausencia de sombras mediante dos aves en sendos espacios virtuales, de manera que en el de la izquierda no se proyectan sombras, mientras que en el de la derecha sí lo hacen. Nótese que en la imagen de la izquierda queda indeterminada la posición relativa del ave sobre la superficie azul. En la imagen de la derecha una simple sombra ha resuelto tal incertidumbre.

[70] http://web.cse.ohio-state.edu/~whmin/courses/cse5542-2013-spring/19-shadow.pdf

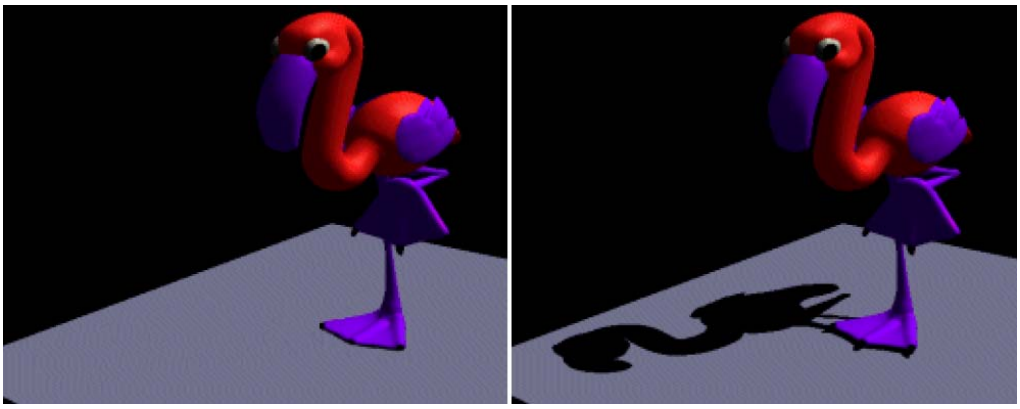

Figura 111. Ave en dos espacios virtuales poniendo en evidencia la necesidad de sombras para determinar la ubicación de los objetos presentes en la escena

A continuación incluyo un segmento de código del archivo *gazebo.material* donde se define un material llamado "Parquet" para que la textura de parquet del suelo pueda recibir sombras del resto de elementos como el ring, los postes de focos o los propios robots. Es importante señalar que el mencionado archivo de materiales se encuentra en una carpeta de Gazebo 2.2 protegida contra escritura ocasional (.../usr/share/gazebo-2.2/media/materials/scripts), resultando necesario ejecutar comandos previos como *gksudo nautilus* para añadir nuevos materiales.

```
material Gazebo/Parquet
{
 technique
 {
  pass
  {
   ambient 1.0 1.0 1.0 1.0
   diffuse 1.0 1.0 1.0 1.0
   specular 0.2 0.2 0.2 1.0 12.5
   texture_unit
   {
    texture parquet.jpg
    filtering trilinear
   }
  }
 }
}
```

La pista de parquet que he utilizado con Gazebo 2.2 es la misma que diseñé para el ring de sumo de Webots 7 con idénticas salvedades a las ya explicadas en cuanto a formatos de archivo y sombras.

En cuanto a los postes con focos o las vallas publicitarias que delimitan el área de la competición, hay que decir que su diseño y forma de ser tratados para añadirlos a las simulaciones no difiere del ring o el suelo de parquet. Las columnas con focos las he incluido principalmente para añadir realismo a la escena, evitando que el entorno quede demasiado vacío e irreal. Las vallas que bordean el entorno del ring, en cambio, tienen otras funciones: en primer lugar, evitar que los robots salgan del área de la competición, perdiéndose en el infinito; en segundo lugar, ocultar la visión del final de la pista en el horizonte uniéndose con la nada; en tercer lugar, promocionar la competición de sumo conjuntamente con los rótulos del ring.

Las colisiones en Gazebo 2.2 es posible definirlas, bien como en Webots 7 mediante un conjunto de figuras geométricas simples con ubicación respecto al centro de coordenadas del objeto en cuestión o bien utilizando la propia malla 3D del objeto como contorno de colisiones. El primer método supone cálculos de física sencillos y menos carga de procesamiento de datos, por contra requiere invertir tiempo definiendo formas geométricas simples que se adapten al contorno del objeto. El segundo método permite definir las envolventes de colisión de manera que coincidan exactamente con la geometría del objeto sin consumo de tiempo, pero por contra supone mayor carga de cálculos de física. Dado el escaso número de objetos presentes en la escena de la lucha de sumo y el carácter estático de los elementos citados hasta ahora, he optado por utilizar este último método referenciando la malla 3D de cada objeto en el apartado de colisiones de sus respectivos archivos de configuración `model.sdf`. A continuación incluyo un segmento de código del archivo `model.sdf` del ring de sumo donde se muestra esta manera de proceder.

```
---------------------------------------------------------------------------------------------------------------------
<collision name="collision">
  <geometry>
    <mesh>
      <uri>model://sumo_ring/meshes/Circular_Ring.dae</uri>
    </mesh>
  </geometry>
...
</collision>
---------------------------------------------------------------------------------------------------------------------
```

En lo referente a la creación de entornos con Gazebo 2.2 creo importante señalar que los elementos que componen la escena pueden ubicarse en ella bien utilizando el editor gráfico de niveles de este simulador o bien editando el correspondiente archivo definitorio del entorno **.world*. El primer método resulta complicado debido principalmente a la enorme escasez de herramientas del editor de niveles de Gazebo 2.2. El segundo procedimiento es más operativo, pero supone un gran coste en tiempo para el que desee crear un entorno con Gazebo debido a la gran cantidad de pruebas y errores necesarios para afinar la ubicación de cada elemento.

Modelo de robot Pioneer 3AT con Gazebo 2.2

Gazebo 2.2 al igual de Webots 7 incorpora varias plataformas robóticas listas para usar, incluyendo un modelo de robot Pioneer 3AT. El modelo de Pioneer 3AT de Gazebo, en cambio, resulta mucho menos exacto que el de Webots en todos los sentidos, recomendando el desarrollo de un nuevo modelo si se quieren obtener simulaciones realistas y evitar poca aceptación del modelo por parte de los usuarios.

En el aspecto visual, el modelo de robot Pioneer 3AT incluido en Gazebo 2.2 se muestra muy poco detallado y con graves errores en la manera de procesar la luz que recibe de su entorno. El nivel de detalle de un modelo 3D puede medirse por la cantidad de triángulos que lo forman y por la complejidad de las texturas que lo decoran. En este sentido, el modelo de Pioneer 3AT de Gazebo 2.2 está compuesto por solo 1587 triángulos y no incorpora textura alguna, pues en su lugar se han utilizado colores uniformes. Para que un modelo 3D tenga buen aspecto en simulación no es suficiente con que su nivel de detalle sea alto, sino que además el objeto en cuestión debe procesar correctamente la luz que recibe del entorno, de manera que las luces y las sombras se distribuyan sobre el modelo de un modo coherente con las condiciones de iluminación simuladas. Para que esto último suceda se pueden utilizar diferentes técnicas con el programa de diseño 3D utilizado en la creación el objeto en cuestión, según la información que utilice el simulador para procesar la luz. Gazebo 2.2, y la mayoría de simuladores no basados en motor de juego, es muy básico en este sentido y para distribuir la luz sobre los objetos solo utiliza la información de sus grupos de suavizado (*smoothing groups*), es decir la relación que tienen entre sí, en cuanto a distribución de luz, las diferentes superficies (triángulos habitualmente) que componen un objeto. El efecto de una mala distribución de *smoothing groups* se puede observar en el cubo de la izquierda de la Figura 112, donde la luz se suaviza entre las distintas caras del objeto dificultando distinguir su forma, llegando a

desaparecer algunas aristas y sobreexponiendo superficies que deberían estar menos iluminadas teniendo en cuenta la dirección en la que la luz incide sobre el cubo (nótese la dirección de la sombra que el objeto proyecta sobre el plano verde).

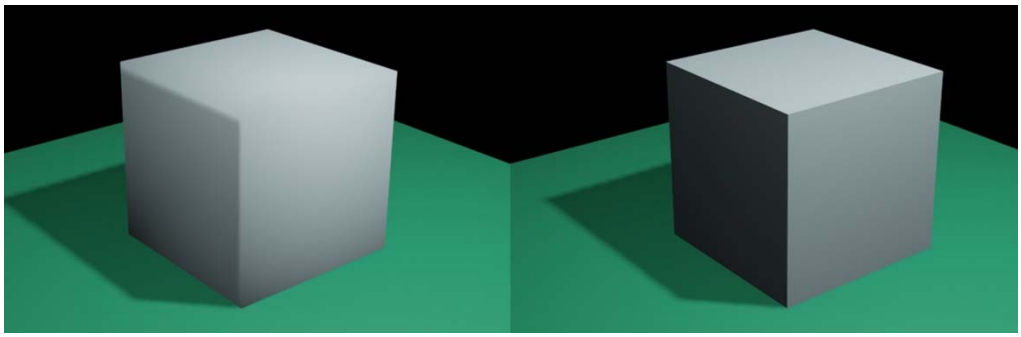

Figura 112. Dos cubos renderizados en 3D Studio Max con diferencias en la definición de sus smoothing groups. El cubo de la izquierda presenta una distribución incorrecta de smoothing groups, mientras que el de la derecha tiene una distribución más adecuada

Cuando se observa el modelo de Pioneer 3AT de Gazebo 2.2 llama la atención que no hay ninguna superficie bien iluminada, por lo contrario, parece estar todo el robot ensombrecido de manera que hay muchas superficies, incluso partes del robot enteras, que no se distinguen unas de otras por estar todas ellas en negro. Da la impresión que el robot se encuentra en el interior de una habitación oscura.

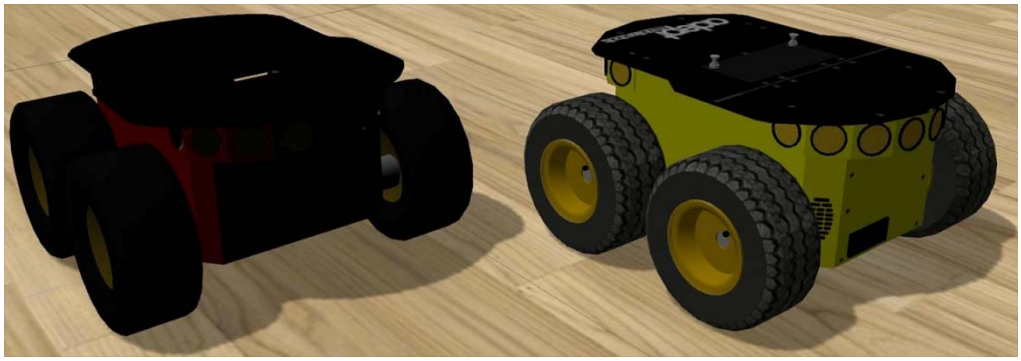

Figura 113. Comparación entre el modelo de robot Pioneer 3AT incluido en Gazebo 2.2 y el creado por mí para una competición de sumo entre estudiantes

La Figura 113 muestra a la izquierda el modelo de robot Pioneer 3AT incluido en Gazebo 2.2. Aquí se pueden observar todos los aspectos comentados acerca de su nivel de detalle e iluminación, comparándolo con el modelo que he creado para sustituirlo a la derecha de la imagen. Así, queda patente que el modelo de la izquierda

es escaso en detalles, no tiene texturas y tampoco procesa la luz correctamente. El modelo de la derecha lo he diseñado con un número de polígonos razonable (10334 triángulos), con varias texturas que añaden mucho detalle y cuidando los *smoothing groups*.

Es importante señalar que un aspecto correcto y realista puede resultar muchas veces vital para entender lo que sucede durante una simulación. Utilizando el modelo de Pioneer 3AT de Gazebo observé que la ausencia de detalles como tornillos o válvula de hinchado para las ruedas y textura que simule un neumático provoca que sea imposible para el usuario determinar si las ruedas del robot están siquiera girando. Esto lleva a numerosas incertidumbres en cuanto al estado del robot, pues podría estar moviéndose deslizando sobre sus ruedas paradas, podría estar parado con las ruedas girando sin tracción, etc.

En lo referente al aspecto funcional del modelo de robot Pioneer 3AT incluido en Gazebo 2.2, destaca negativamente la tosca envolvente de colisión del chasis en forma de caja. Por lo demás, se trata de un buen modelo que da la impresión de simular correctamente los movimientos del robot y su interacción con el entorno. Así pues, para poder simular correctamente un combate de sumo entre dos robots Pioneer 3AT con Gazebo 2.2, he tenido que crear un nuevo modelo 3D para ese robot con un aspecto adecuado y unas colisiones ajustadas a su geometría.

Archivos para una lucha de sumo con Gazebo 2.2

El trabajo con Gazebo 2.2, aunque no es excesivamente complicado, sí resulta algo singular en comparación con Webots o USARSim. Además, el desarrollo de un entorno virtual con Gazebo requiere generar una cantidad de archivos relativamente elevada, aunque se trate de escenas con pocos objetos presentes.

Dada la complejidad del sistema de archivos que me ha sido necesario crear para llevar a Gazebo un entorno de lucha de sumo, expongo a continuación tales ficheros organizados en dos tablas. La Tabla 13 describe todos los archivos que he creado o modificado con su descripción y la ruta donde los he guardado para su correcto funcionamiento, exceptuando los archivos correspondientes a los robots Pioneer 3AT que se enumeran en la Tabla 14.

Nombre de archivo	Descripción	Ruta
model.config	Descripción de la valla publicitaria	..\.gazebo\models\sumo_fence
model.sdf	Definición completa de la valla publicitaria	...\.gazebo\models\sumo_fence
publifence.dae	Objeto 3D de la valla publicitaria	..\.gazebo\models\sumo_fence\meshes
robinlab.jpg	Textura para la valla publicitaria	..\.gazebo\models\sumo_fence\materials\textures
model.config	Descripción de la columna con focos	..\.gazebo\models\sumo_lightpost
model.sdf	Definición completa de la columna con focos	..\.gazebo\models\sumo_lightpost
lightpanel.dae	Objeto 3D de la columna con focos	..\.gazebo\models\sumo_lightpost\meshes
lightpanel.jpg	Textura para la columna con focos	..\.gazebo\models\sumo_lightpost\materials\textures
model.config	Descripción de la pista de parquet	..\.gazebo\models\sumo_parquet
model.sdf	Definición completa de la pista de parquet	..\.gazebo\models\sumo_parquet
parquetground.dae	Objeto 3D del suelo de parquet	..\.gazebo\models\sumo_parquet\meshes
parquet.jog	Textura para el suelo de parquet	..\.gazebo\models\sumo_parquet\materials\textures
model.config	Descripción del ring de sumo	..\.gazebo\models\sumo_ring
model.sdf	Definición completa del ring de sumo	..\.gazebo\models\sumo_ring
Circular_Ring.dae	Objeto 3D del suelo de parquet	..\.gazebo\models\sumo_ring\meshes
sumoring.jpg	Textura para el suelo de parquet	..\.gazebo\models\sumo_ring\materials\textures
sumoring.world	Definición completa del entorno de lucha de sumo	..\.gazebo\worlds
sumoring_world.launch	Archivo para ejecutar el entorno del ring de sumo desde ROS	..\opt\ros\indigo\share\gazebo_ros\launch
gazebo.material	Definición de materiales de Gazebo 2.2	..\usr\share\gazebo2.2\media\materials\scripts

Tabla 13. Archivos que componen un entorno simple para competiciones simuladas de lucha de sumo con Gazebo 2.2

Nombre de archivo	Descripción	Ruta
model.config	Descripción del robot Pioneer 3AT de color rojo	..\.gazebo\models\sumo_p3at
model.sdf	Definición completa del robot Pioneer 3AT rojo	..\.gazebo\models\sumo_p3at
chassis.dae	Objeto 3D del chasis del robot Pioneer 3AT	..\.gazebo\models\sumo_p3at\meshes
wheel.dae	Objeto 3D de una rueda derecha del robot	..\.gazebo\models\sumo_p3at\meshes
wheelref.dae	Objeto 3D de una rueda izquierda del robot	..\.gazebo\models\sumo_p3at\meshes
p3at_full.jpg	Textura para el robot Pioneer 3AT rojo	..\.gazebo\models\sumo_p3at\materials\textures
model.config	Descripción del robot Pioneer 3AT amarillo	..\.gazebo\models\sumo_p3aty
model.sdf	Definición completa del robot amarillo	..\.gazebo\models\sumo_p3aty
chassis.dae	Objeto 3D del chasis del robot Pioneer 3AT	..\.gazebo\models\sumo_p3aty\meshes
wheel.dae	Objeto 3D de una rueda derecha del robot	..\.gazebo\models\sumo_p3aty\meshes
wheelref.dae	Objeto 3D de una rueda izquierda del robot	..\.gazebo\models\sumo_p3aty\meshes
p3at_fulyl.jpg	Textura para el robot Pioneer 3AT amarillo	..\.gazebo\models\sumo_p3aty\materials\textures

Tabla 14. Archivos necesarios para definir un nuevo modelo de robot Pioneer 3AT en Gazebo 2.2 en dos variantes de color

7.3.3. Lucha de sumo con USARSim v.1.2 para UDK

Con el fin de poder comparar resultados con facilidad, he creído conveniente desarrollar también con USARSim v.1.2 para UDK un ring de sumo para competiciones virtuales entre estudiantes de robótica. No obstante, teniendo en cuenta la enorme cantidad de trabajo realizado con el edificio TI de la UJI y las mencionadas ventajas de mezclar elementos fantásticos con otros reales, esta vez he procedido de manera diferente. He aprovechado el modelo 3D y las texturas del edificio TI desarrollados para USARSim v.3.3 (para UT2004) y he creado un nuevo entorno con USARSim v.1.2 para UDK, añadiendo un ring de sumo. De este modo, el entorno del edificio TI vuelve a estar disponible para simulaciones de alta fidelidad, permitiendo una fácil comparación entre ambas versiones de USARSim.

El edificio TI es bien conocido por los estudiantes de informática de la UJI, ya que en su interior se haya una multitud de seminarios y laboratorios donde realizan algunas

de sus sesiones prácticas durante la carrera. Esta familiaridad de los estudiantes con el edificio TI espero que incremente su grado de aceptación y su interés por utilizar el simulador.

En resumen, el detallado modelo 3D del edificio TI introduce un ambiente conocido y realista que combinado con elementos fantásticos o irreales (el ring de sumo) facilita atractivas simulaciones hiperrealistas que pueden mejorar la experiencia de los estudiantes, promoviendo su interés por la robótica.

La Figura 114 muestra una vista global del edificio TI con el ring de sumo y su entorno cercano visto desde la fachada posterior. El ring de sumo lo he ubicado en un área que en la realidad está destinada a la carga y descarga de vehículos (ver Figura 115). Cabe destacar que este completo entorno puede utilizarse para implementar simulaciones muy diversas, no solo competiciones de lucha de sumo.

Figura 114. Vista global del edificio TI de la UJI para simulaciones con USARSim v.1.2 para UDK

Figura 115. Detalle de un ring de sumo para simular competiciones robóticas entre estudiantes

Edificio TI y ring de sumo con USARSim v.1.2 para UDK

Como en el capítulo 6 de esta tesis ya he hablado del desarrollo de entornos con USARSim para UDK y ya he descrito el modelo de edificio TI en el capítulo 5, en este apartado expongo únicamente aquello que aporta algo nuevo o interesante.

En cuanto al modelo de edificio TI adaptado a UDK, son destacables las visibles mejoras en iluminación y materiales, la gran superioridad de la nueva bóveda celestial y la facilidad con la que tratan los elementos animados. El resultado es un entorno mucho más realista que el desarrollado con USARSim v.3.3 para UT2004.

La Figura 116 muestra el interior de los edificios TI desarrollados para ambas versiones de USARSim. Aquí pueden apreciarse las mejoras en iluminación y realismo de materiales de Unreal Engine 3 de UDK (imagen de la derecha) sobre Unreal Engine 2 de UT2004 (imagen de la izquierda).

Figura 116. Pasillo de la primera planta del edificio TI de la UJI enfrente del laboratorio de robótica inteligente. La imagen de la izquierda corresponde a USARSim v.3.3 y la de la derecha a USARSim v.1.2 para UDK

Respecto al ring de sumo incluido en este entorno, pueden citarse las siguientes ventajas sobre los desarrollados para Webots o Gazebo:

- Realistas sistemas de partículas como fuego, humo, etc.
- Fuentes de viento simuladas.
- Objetos con física de trapo.
- Fluidos realistas.
- Fuentes de sonido ambiental.
- Materiales realistas.

El ring de sumo de este entorno supone la implementación de un recurso habitual en los video juegos que consiste en mezclar los elementos fantásticos que forman parte de una trama o historia con elementos reales de modo que el conjunto resulte plausible y sea aceptado como real. De esta manera, incluyendo el ring de sumo en un entorno conocido por los estudiantes, he intentado conseguir un efecto de realidad paralela que permita mayor aceptación, facilitando la inmersión y mejorando la experiencia del usuario.

En este entorno, como en el laberinto, he incluido recursos destinados a evitar los efectos adversos que sobre la inmersión tienen el silencio total o la quietud absoluta. Se trata de fuentes de sonido ambiental como pájaros, ranas (en el agua que rodea el ring) o fuego y elementos que se mueven como llamas, humo, banderas, árboles agitados por el viento, pájaros volando u ondas en el agua. Tanto los árboles como las banderas tienen en común que su modelo 3D está dotado de un esqueleto que les confiere la capacidad de moverse según las articulaciones definidas entre los huesos. Los árboles y plantas que he incluido son recursos de UDK disponibles en sus bibliotecas para agilizar la creación de entornos. En cuanto a las banderas, éstas son de diseño propio tanto el modelo 3D como las texturas y, además, tienen como función adicional publicitar el torneo de sumo. Una vez diseñadas en 3D Studio Max e importadas desde UDK, las banderas las he dotado de un material que simula tela y de una física de trapo que permite que ondeen con el viento (ver Figura 117).

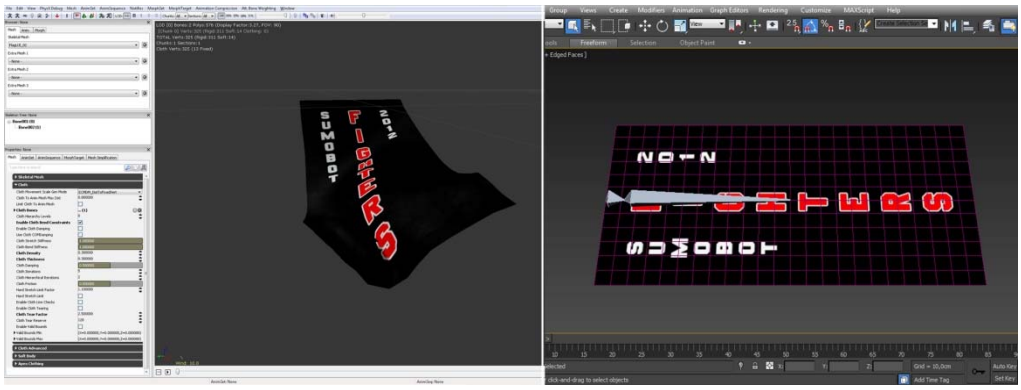

Figura 117. Bandera de la competición de sumo. A la izquierda se puede observar la bandera en el editor de UDK con física su de trapo, mientras que a la derecha se puede ver en gris los dos huesos que definen su estructura de esqueleto

Con el objetivo de aprovechar la ausencia de limitaciones físicas inherentes a cualquier mundo virtual, he rodeado de agua el ring de lucha. De esta manera, el robot

que pierde el combate cae al agua resultando más evidente la imposibilidad de seguir luchando. El agua la he simulado con la suma de tres elementos: en primer lugar, una lámina que simula fluido y reacciona como tal generando olas al recibir un impacto; en segundo lugar un material que confiere aspecto de agua a esa lámina de fluido; en tercer lugar, un volumen de post procesado de imagen que provoca una borrosidad que simula la visión de objetos a través de agua algo turbia cuando se ubica la cámara dentro del agua. La conjunción de estos elementos confiere al agua simulada un aspecto y un comportamiento realista. La intención original del agua iba un poco más allá en el sentido que al caer el robot al agua debía escucharse un sonoro "chof" y ,acto seguido, el simulador debía dar realimentación al usuario dando por terminado el combate. Esta última parte se ha implementado con éxito en uno de los entornos que se detallan en el capítulo 8, demostrándose que con USARSim para UDK es posible crear entornos para competiciones robóticas que detecten cuando éstas han alcanzado su fin, informando de ello a los usuarios.

Modelo de robot Pioneer 3AT con USARSim v.1.2 para UDK

USARSim v.1.2 para UDK incluye un modelo de robot Pioneer 3AT listo para usar y con un alto nivel de fidelidad tanto gráfica como funcional. Esto ha permitido utilizar este modelo de robot para los combates de sumo, siendo solamente necesario crear un nuevo modelo de Pioneer 3AT en diferente color (amarillo) para disponer de dos contrincantes visualmente distinguibles.

En cuanto a la fidelidad gráfica del modelo de Pioneer 3AT de USARSim, cabe destacar que se trata de un modelo compuesto por 5752 triángulos y varias texturas que simulan un nivel de detalle superior, confiriéndole un aspecto correcto con una carga gráfica reducida (ver Figura 118).

En lo referente a la fidelidad funcional, este modelo posee unas envolventes de colisión bien ajustadas al contorno del robot y unos parámetros físicos que le otorgan un comportamiento realista en simulación. Además, este modelo de Pioneer 3AT, al igual que el robot real, permite ser equipado con multitud de sensores ya incluidos en USARSim, permitiendo al usuario crear cualquier otro sensor o actuador de manera relativamente sencilla utilizando Unreal Script tal como ya se detalló en el capítulo 5 de esta tesis.

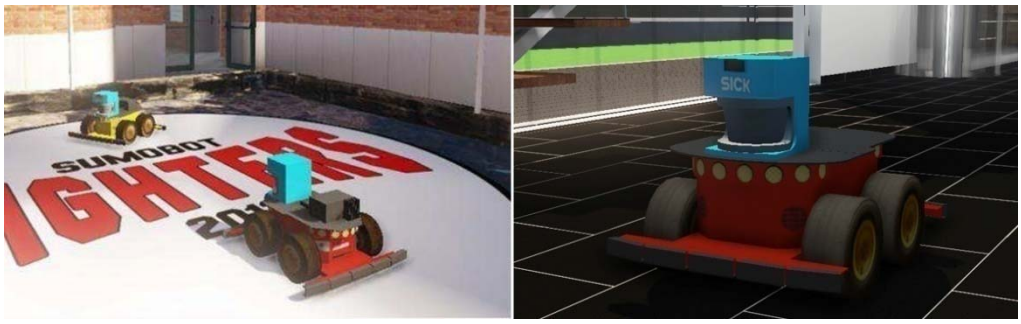

Figura 118. Modelo de robot Pioneer 3AT incluido en USARSim v.1.2. para UDK. A la izquierda dos Pioneer 3AT en dos colores en una lucha de sumo. A la derecha vista de detalle del robot

En resumen, el modelo de robot Pioneer 3AT y los sensores con los que puede equiparse con USARSim para UDK resultan perfectamente utilizables simular competiciones de sumo entre estudiantes.

7.4. Lucha de boxeo con USARSim v.1.2 para UDK

Las competiciones de robots luchadores de sumo, como he mencionado, tienen mucha tradición en la UJI y por ese motivo he estimado conveniente crear tres entornos para comparar diferentes simuladores. No obstante, no hay motivo para poner límites donde no los hay ni debería haberlos, es decir, ¿Por qué no aprovechar las infinitas posibilidades inherentes a la simulación para llevar a cabo nuevas competiciones en robótica educativa? Con esta idea decidí crear un nuevo reto para los estudiantes, preparando un sencillo entorno virtual para albergar una competición de boxeo (ver Figura 119).

Figura 119. Ring para competiciones simuladas de boxeo entre robots humanoides con USARSim

Una competición de robots luchadores de boxeo reúne muchas cualidades que considero interesantes en este tipo de simulaciones. En primer lugar, una competición de boxeo entre robots supone algo novedoso y con un alto atractivo que puede ayudar a captar la atención y el interés de los estudiantes. En segundo lugar, los combates de boxeo son conocidos por la mayoría de las personas y sus reglas son sencillas de entender. En este caso, las reglas se reducen a que el primer robot que cae al suelo pierde el combate. En tercer lugar, el robot necesario para esta competición es del tipo humanoide, lo cual supone una novedad para los estudiantes y una oportunidad para implementar nuevos algoritmos. En cuarto lugar, el entorno virtual necesario para este tipo de eventos no requiere muchos elementos y es, por tanto, sencillo de construir.

El entorno que he diseñado para simular competiciones de boxeo entre humanoides consiste básicamente en un ring de lucha situado en el oscuro interior de un gran cubo negro donde solo se distinguen las luces de unos focos (ver Figura 120).

Figura 120. Vista global del ring de boxeo durante un entrenamiento (imagen de la izquierda) y detalle de un humanoide interactuando con el saco (imagen de la derecha)

El ring lo he creado con una forma octogonal similar a los utilizados en los combates de MMA (Mixed Martial Arts). Este diseño permite aislar un poco más el ring del resto del entorno, de manera que el usuario no percibe la existencia de un vacío más allá del área de lucha. Además, los soportes verticales de las vallas que delimitan el octógono ofrecen un espacio perfecto para la colocación de rótulos que publiciten la competición, tal como sucede con los combates reales de MMA. Tanto los objetos 3D como las texturas que componen el ring los he creado siguiendo un procedimiento similar al ya explicado en otros entornos de USARSim para UDK. Del ring solo es

destacable la enorme funcionalidad que ofrece UDK en cuanto a la creación de materiales, permitiendo obtener unas vallas realistas combinando texturas de malla metálica con mapas de relieve y opacidad.

La Figura 121 muestra en su parte derecha el aspecto de la valla del ring en simulación, mientras que a la izquierda de la imagen se ve una captura del editor de materiales de UDK donde se aprecia una esfera con el material de la valla, obtenido éste como composición de diferentes texturas y funciones matemáticas. Estas funciones matemáticas permiten operar con los valores que contienen los canales de color u opacidad usando un lenguaje de bloques funcionales muy completo.

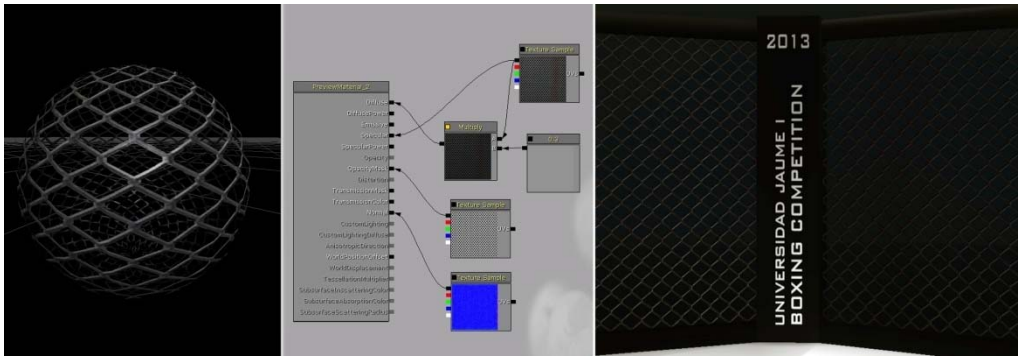

Figura 121. Valla con material realista. A la izquierda, una captura de pantalla del editor de materiales de UDK. A la derecha, la el aspecto de la valla en simulación

Las texturas que conforman el material de la valla se muestran con mayor detalle en la Figura 122, donde pueden observarse de izquierda a derecha la textura de color, el mapa de opacidad y el mapa de relieve.

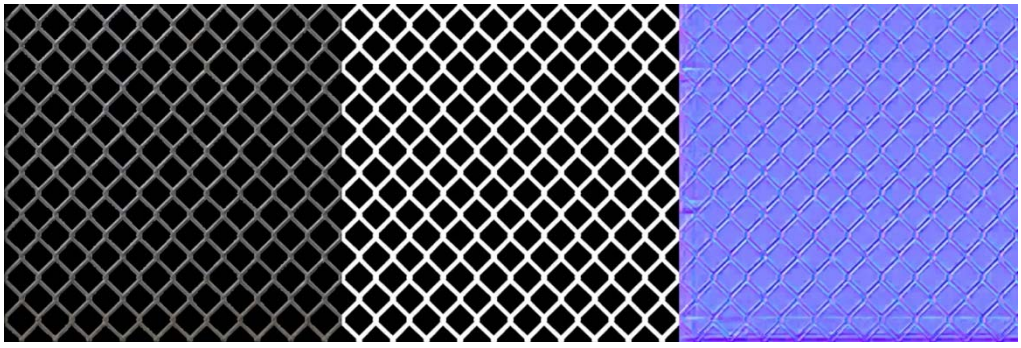

Figura 122. Texturas que componen el material de la valla del ring. De izquierda a derecha se observa la textura de color, el mapa de opacidad y el mapa de relieve

El espacio que rodea el ring está prácticamente vacío. Se compone de un enorme cubo hueco con paredes y suelo de color negro mate para que no refleje la luz ni atraiga la atención del usuario. Toda la escena simula estar iluminada por cuatro columnas de focos que son los únicos elementos existentes entre el ring y los límites del entorno. Para obtener este efecto de iluminación he dotado la superficie de vidrio de los focos de un material que simula emitir luz, generando un aurea blanquecina a su alrededor. Para reforzar este efecto he añadido delante de cada foco una superficie cónica con un material blanquecino con opacidad decreciente que simula el efecto visual de un haz de luz. Para implementar la iluminación en sí, he situado un punto de luz cercano a cada grupo de focos, de manera que la luz incide en el área de lucha desde cuatro puntos diferentes, permitiendo ver el combate desde cualquier ángulo.

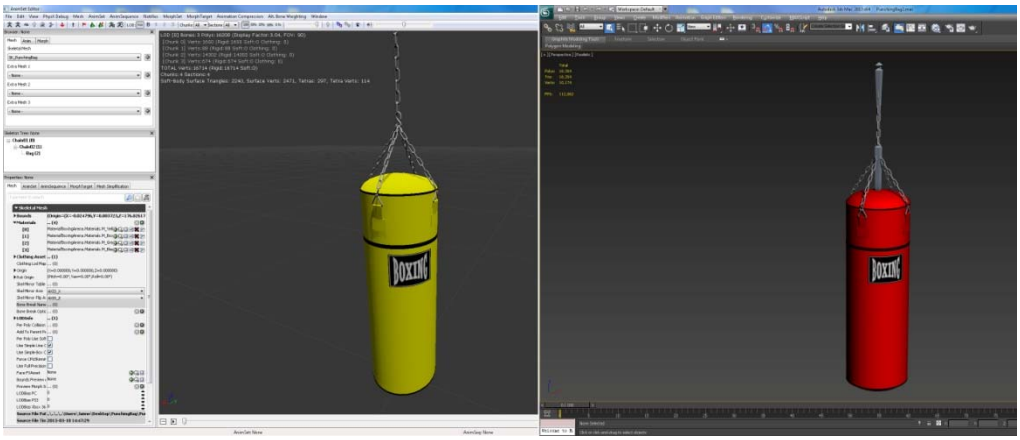

Figura 123. Saco de boxeo visto como SkeletalMesh en el editor de UDK (izquierda de la imagen) y como malla dotada de huesos (en gris) en 3D Studio Max (derecha de la imagen)

Con el objetivo de ofrecer a los estudiantes una oportunidad para practicar en solitario sus algoritmos de lucha, he añadido un realista saco de boxeo en el ring que puede ser retirado para la competición. Este saco reacciona a los golpes de manera similar a como lo haría uno real gracias a su diseño y a sus parámetros físicos. En cuanto al diseño, el saco lo he creado con 3D Studio Max y está compuesto por dos elementos fundamentales que son las cadenas de sujeción y el saco en sí, quedando articulado mediante una estructura de huesos. El conjunto del saco con su estructura de huesos, convenientemente configurados, los he importado desde UKD en calidad de "SkeletalMesh" (maya articulada) lo que permite definir una física basada en articulaciones y huesos (ver Figura 123). Una vez definidos adecuadamente los parámetros físicos del saco, éste es capaz de reaccionar a los golpes de manera

realista, resultando un elemento de interacción novedoso y de gran utilidad para que los estudiantes practiquen con sus robots.

En cuanto al robot humanoide elegido para esta competición, me he decidido por el NAO de Aldebarán Robotics. Este robot tiene unas capacidades que superan las necesidades de la competición y ya lo tengo modelado e introducido en USARSim, quedando disponible para su uso en varias versiones y colores.

7.5. Competiciones domésticas con USARSim

En 2013 el profesorado del Departamento de Ingeniería de los Computadores de la UJI encargado impartir la asignatura de robótica me trasladó la necesidad de implementar simulaciones en un entorno doméstico realista para robots humanoides. La idea provenía del auge que en los últimos años han experimentado los ensayos destinados probar algoritmos mediante los que un robot humanoide debe realizar determinadas tareas domésticas [106]. Mi intención original respecto a una simulación de tales características consistió en realizar una vivienda con todo su interior y un exterior plausible, de manera que un robot humanoide como el NAO pudiera moverse con comodidad por su interior interactuando con los objetos a su alcance. La elección del simulador fue inmediata, elegí USARSim v.1.2 para UDK dado que con ningún otro de los simuladores bajo estudio podía imaginar el desarrollo de un entorno semejante. Tanto Webots como Gazebo fallan estrepitosamente cuando se intenta implementar una atmósfera realista, es decir, con un horizonte y cielo plausibles y, además, ambos simuladores ofrecen muchas dificultades cuando el número de objetos que componen la escena es elevado. Además, este simulador (USARSim para UDK) ya lo había empleado con éxito en la construcción del edificio TI y su entorno inmediato. La elección del humanoide NAO se debió una vez más a que era (y sigue siendo a día de hoy) el más utilizado en el mencionado departamento y, además, ya disponía de un modelo de alta fidelidad de este robot que desarrollé para el entorno del laberinto expuesto en el capítulo 6.

La idea del entorno doméstico pronto evolucionó impulsada por proyectos y propuestas relacionadas con la competición de HUMABOT[71] 2014 [106]. El personal del departamento de Ingeniería y Ciencia de los Computadores de la UJI tenía varias propuestas interesantes para la implementación de una competición robótica durante

[71] http://www.irs.uji.es/humabot/

el mencionado evento. Por una parte, podría organizarse una competición en la que los robots humanoides realizaran tareas domésticas en una cocina al estilo RoboCup @Home [156]. Por otra parte, se planteaba también la posibilidad de implementar algoritmos que permitieran a un humanoide encontrar la salida de un laberinto de que simulara las habitaciones de una casa. Otra posibilidad era preparar una prueba en la que un humanoide debía recoger objetos diseminados por el suelo y colocarlos dentro de una caja como si de un niño recogiendo juguetes se tratara. En resumen, estas ideas sobre posibles competiciones robóticas con robots reales en HUMABOT 2014 aconsejaron preparar un entorno más completo que permitiera simular diversos comportamientos en un ambiente realista. Así pues, diseñé un extenso mundo virtual con capacidad para llevar a cabo distintas competiciones entre estudiantes (ver Figura 124).

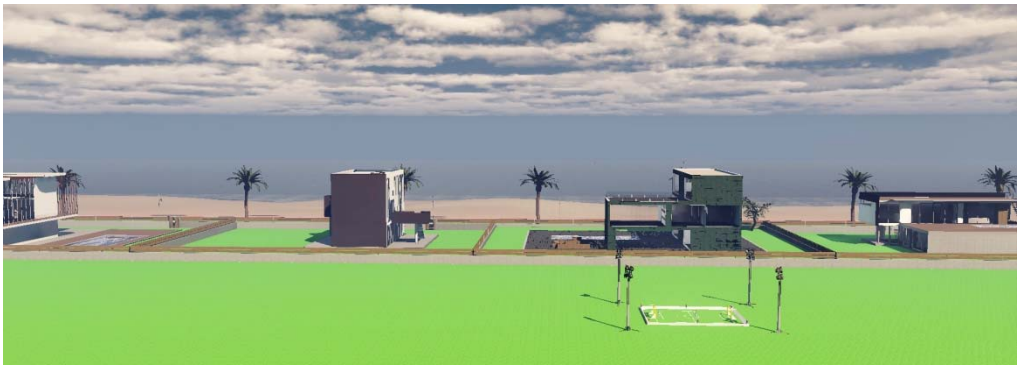

Figura 124. Vista general de un entorno realista para simulación de diferentes competiciones robóticas

A continuación detallo por separado las distintas áreas que componen este entorno multipropósito. Dado que ya he explicado el proceso de obtención de otros escenarios y robots con USARSim v.1.2 para UDK, en los siguientes puntos me centro únicamente en los aspectos que considero importantes de cara a la competición entre estudiantes o novedosos (por no haberlos incluido en los entornos expuestos hasta ahora) e interesantes para implementarlos en futuras simulaciones.

La casa

La idea de una competición de tareas domésticas se basa en implementar algoritmos de diversa índole mediante los cuales un robot humanoide como el NAO pueda encargarse de:

- Buscar y reconocer productos alimenticios en una estantería para deducir qué productos faltan o hay que comprar.
- Poner en marcha o apagar un horno.
- Recoger objetos que se encuentran en el suelo y colocarlos en el lugar adecuado.
- Navegar por la vivienda.

Con estas especificaciones presentes he construido una vivienda virtual de tres plantas con amplios espacios y pasillos para permitir la navegación de los robots. Los techos los he hecho un tanto altos y he dispuesto grandes cristaleras y ventanas para facilitar la visión del interior del edificio. Un ascensor de paredes de vidrio permite a los robots moverse por todas las plantas de la vivienda con facilidad sin tener que utilizar las escaleras. También he creado un extenso y detallado terreno que genera la ilusión que la vivienda se encuentra en una urbanización frente a una playa.

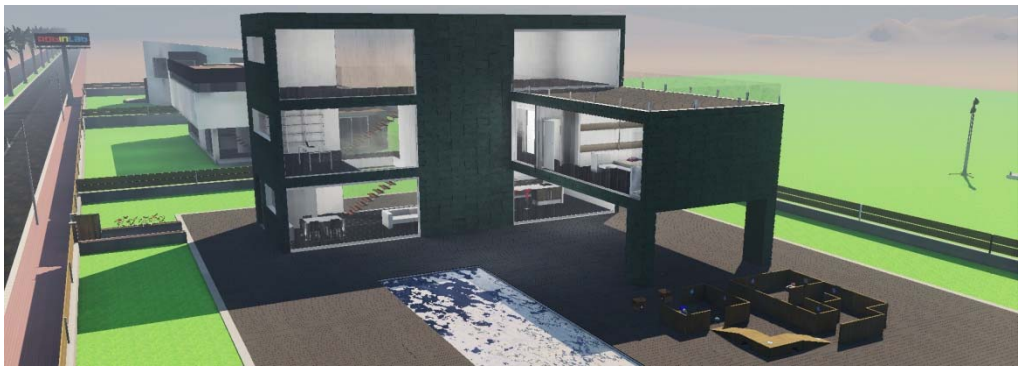

Figura 125. Edificio para simulaciones de entornos domésticos con USARSim v.1.2 para UDK

La Figura 125 en su parte central muestra el exterior de la vivienda principal de este entorno de simulación. En la imagen puede observarse la gran cantidad de superficies de vidrio que he usado (ventanas, puertas, barandillas, etc.) para facilitar la observación de lo que ocurre en el interior sin tener que quitar paredes o techos, manteniendo el aspecto realista. En la parte inferior derecha de la imagen se

encuentra un laberinto que cuenta con una abertura de entrada donde he situado diversos objetos y cajas con diferentes propiedades físicas que explicaré más adelante en este capítulo.

La Figura 126 muestra una vista general de la planta baja de esta vivienda donde se aprecia el salón, la cocina, las escaleras al primer piso y el ascensor en forma de tubo de cristal cilíndrico.

Figura 126. Vista general de la planta baja de un entorno de simulación de tareas domésticas

La vivienda destinada a simular competiciones de tareas domésticas la he dotado de objetos cotidianos y de un horno que puede ser encendido y apagado por un humanoide como el NAO mediante un pulsador situado a 50cm del suelo. Este horno dispone de una pantalla LCD que ofrece al usuario realimentación en tiempo real de su estado operativo, mostrando las letras "ON" en verde o las letras "OFF" en rojo.

La Figura 127 muestra un robot NAO dentro de la cocina simulada en este entorno cerca del horno, donde además se encuentran unos estantes en los que el robot deberá reconocer ciertos productos alimenticios y determinar cuál es el que falta y hay que comprar.

El control del horno se realiza mediante un lenguaje que dispone UDK para la programación de inteligencia artificial llamado *Kismet*. Se trata de un lenguaje gráfico basado en la interconexión de bloques funcionales que operan a modo de puertas lógicas. UDK dispone de una amplísima y completa variedad de bloques funcionales que según el estado de sus variables de entrada (eventos, variables booleanas, valores numéricos, etc.) provocan cambios en sus variables de salida.

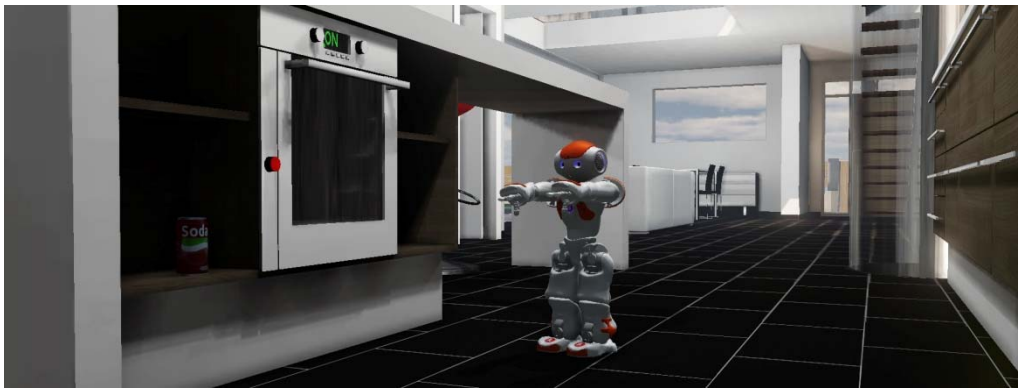

Figura 127. Cocina de un ambiente doméstico para simulación de tareas cotidianas. En la izquierda de la imagen se observa un horno con un LCD que indica su estado y un pulsador que permite a un humanoide NAO apagarlo o encenderlo

En la Figura 128 se muestra el conjunto de bloques funcionales que gobiernan el horno de la cocina y el mensaje mostrado en su pantalla LCD tanto para lectura del usuario como del robot simulado.

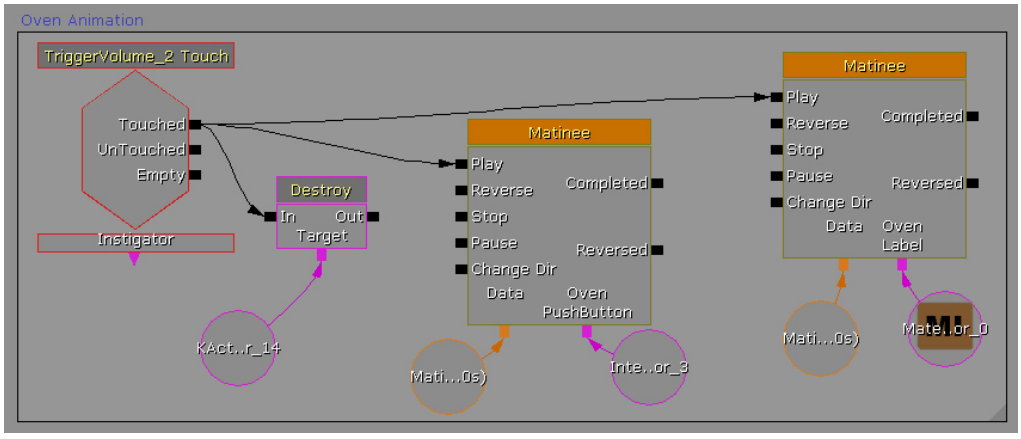

Figura 128. Lenguaje de programación Kismet de inteligencia artificial de UDK basado en bloques funcionales. La figura muestra los bloques funcionales que controlan el estado del horno

Como se verá en el siguiente capítulo el lenguaje *Kismet* permite ofrecer al usuario información de todo tipo durante la simulación, tanto sobreimpresa en la pantalla como de forma auditiva. Así, por ejemplo, el usuario puede conocer qué objetivos le quedan por alcanzar, cuánto tiempo de simulación ha transcurrido, valoración de los logros obtenidos mediante puntuación, etc.

El jardín

Una de las pruebas del ambiente doméstico consiste en implementar algoritmos que permitan reconocer objetos y situarlos en su ubicación correcta. Esta parte de la simulación se lleva a cabo mediante unas piezas de madera con diferentes formas geométricas y colores inspiradas en los juegos infantiles de construcción. Estas piezas las he diseñado con colores que destacan sobre su entorno para facilitar su detección y con un tamaño tal que permite que sean manipuladas por un NAO sin tener que usar los dedos de sus manos. Las piezas deben ser recogidas por el robot y colocadas en el interior de dos cajas marcadas con un código QR. Con el fin de conseguir mayor realismo y ofrecer realimentación al usuario de cuándo se produce el contacto de estas piezas con el suelo, con el robot o con las cajas, he generado materiales físicos tanto para las piezas como para las cajas. Así, mediante el editor de materiales físicos de UDK he creado un material que simula madera en densidad, rozamiento e incluso en sonido al ser golpeado. De igual manera, he definido un material que simula cartón para las citadas cajas.

La Figura 129 muestra los objetos necesarios para esta prueba esparcidos por el área de entrada al laberinto. Para hacer más realista esta prueba he introducido una de las cajas como *SkeletalMesh* y he ajustado su física de manera que sus solapas están articuladas y reaccionan tanto ante los golpes como ante las inercias.

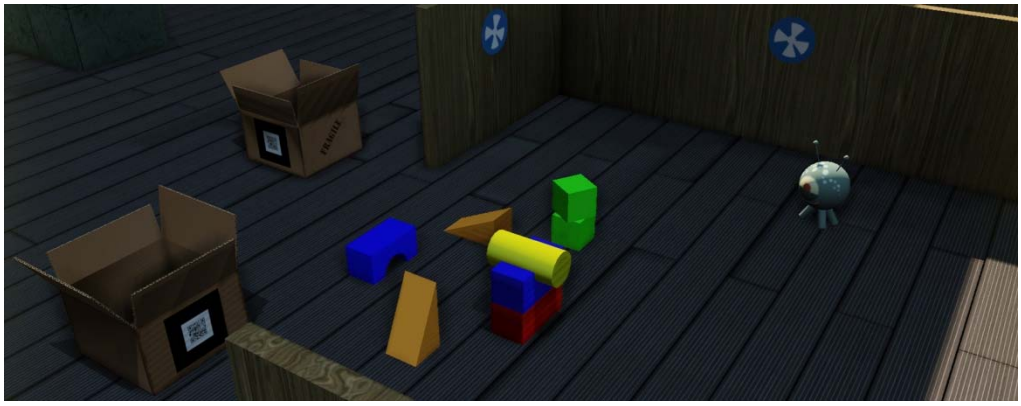

Figura 129. Objetos con diversas formas geométricas y comportamientos físicos que el robot ha de colocar dentro de las cajas de cartón marcadas con un código QR

Uno de los objetos que el robot debe colocar dentro de las citadas cajas es una especie de alienígena de peluche. Este elemento lo he incluido para que los estudiantes puedan experimentar algoritmos de agarre con objetos constituidos por varios

cuerpos sólidos unidos por articulaciones. El proceso de creación de este objeto es similar al del saco de boxeo, introduciendo un esqueleto dentro de la malla 3D y definiendo su correspondiente física mediante el *physics asset editor* de UDK tal como muestra la Figura 130.

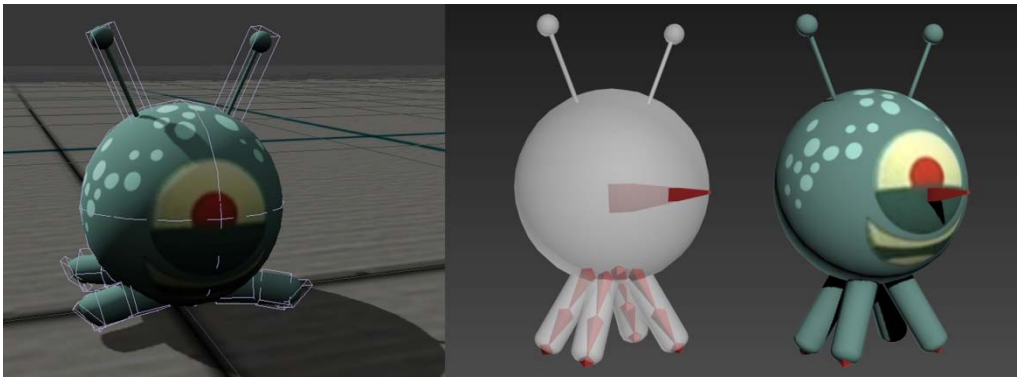

Figura 130. Alienígena de juguete compuesto por varios cuerpos sólidos articulados. De izquierda a derecha se observa el objeto durante una prueba de física, el esqueleto que define sus articualciones en rojo y la malla 3D con su textura

7.6. Resolución de un laberinto con USARSim

Como ya he comentado, una de las posibles competiciones robóticas para HUMABOT 2014 era encontrar la salida de un laberinto. En el capítulo 6 describía un entorno con un laberinto destinado a la implementación de algoritmos de clasificación de imágenes durante las sesiones prácticas sobre sistemas inteligentes. No obstante, la competición HUMABOT Challenge se lleva cabo con robots reales, por tanto el entorno virtual destinado a entrenar a los estudiantes para tal competición debe ser fácilmente reproducible en el mundo real. De este modo, decidí diseñar un laberinto optimizado para su posible construcción a escala real en HUMABOT. Dado que ya disponía de un extenso y realista entorno doméstico creado para simular posibles competiciones de HUMABOT, ubiqué el laberinto en el jardín de la casa principal.

El laberinto está constituido fundamentalmente por dos tipos de paneles de madera de dimensiones 600 x 600 x 50mm y 900 x 600 x 50mm (ancho x alto x espesor) para facilitar que sea reproducido en competiciones con robots reales. Para guiar al robot indicándole el camino a seguir, he colocado una serie de imágenes que el humanoide NAO ya reconoce en su software de fábrica y, por tanto, no requiere de

entrenamiento previo. Inspirado en un exitoso entorno de simulación [158] he añadido a la salida del laberinto un tramo de recorrido con pendiente que pone a prueba el algoritmo de caminar y el control (ver Figura 131).

Figura 131. Vista general de un laberinto en base a paneles de madera fácilmente reproducible en competiciones con robots reales

En este laberinto he añadido un elemento poco usual en los simuladores de robótica y que considero importante por su potencial utilidad. Se trata de una animación que muestra al usuario de manera gráfica qué comportamiento debe implementar para superar la prueba del laberinto. De este modo, el usuario puede recibir información de forma gráfica a parte de las instrucciones que se puedan dar mediante mensajes de voz o sobreimpresos en pantalla. En este caso, cuando el robot penetra en la zona marcada con el rótulo "GET A DEMO" la inteligencia artificial que he dispuesto en forma de *Kismet* ejecuta una animación de un robot NAO que recorre todo el laberinto hasta llegar a su salida.

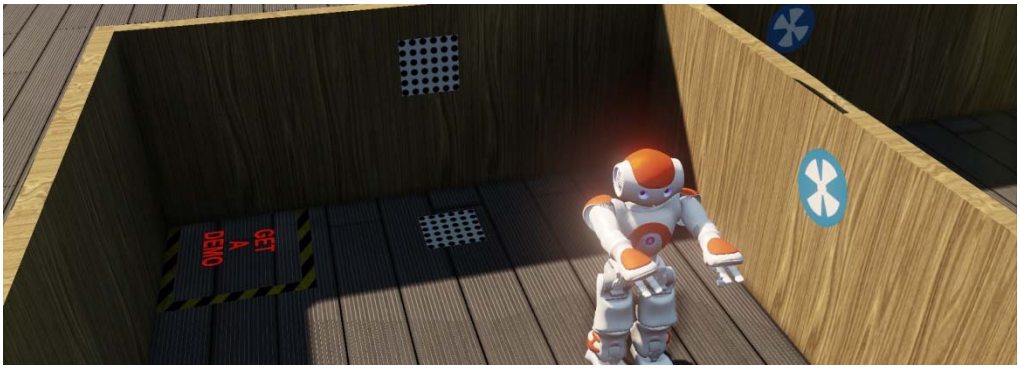

Figura 132. Vista de detalle de los elementos importantes de un entorno de simulación laberíntico

La Figura 132 muestra un par de elementos importantes que he incluido en este laberinto simulado. En primer lugar, cabe destacar las imágenes en forma de matriz de círculos negros sobre fondo blanco para la calibración de las cámaras del robot NAO. En segundo lugar, se aprecian sobre los paneles verticales unos círculos azules con formas blancas en forma de sector angular en su interior, las cuales corresponden al mencionado conjunto de imágenes ya reconocidas por el robot NAO.

7.7. Liga de fútbol con USARSim v.1.2 para UDK

Las competiciones de robots jugadores de fútbol, como ya he comentado al principio de este capítulo, son muy populares y atraen gran interés tanto dentro como fuera de la comunidad científica. Con el fin de aprovechar este atractivo, decidí incluir también un campo de fútbol para robots humanoides en el extenso terreno que rodea el citado entorno doméstico. Para que la experiencia resultara contrastable con otras competiciones más famosas y prestigiosas, seguí fielmente la normativa de RoboCup RoboCup Standard Platform League[72] 2012 para el diseño del campo y todos sus componentes (ver Figura 133).

Figura 133. Campo de fútbol para simulación de una competición de fútbol según las reglas de RoboCup Standard Platform League 2012

Para llevar a cabo una competición de fútbol simulada con garantías de éxito diseñé un terreno de juego según la normativa de RoboCup Standard Platform League 2012 (ver Figura 134), al que añadí unos cilindros con códigos de colores para facilitar la orientación de los robots (ver Figura 135). Todos los elementos incluidos en este entorno destinados a simular la competición de fútbol los he creado de manera similar

[72] http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2012.pdf

a otros ya expuestos anteriormente salvo una de las porterías en la que hice unas pruebas en un intento de incrementar el realismo. Para simular la deformación que sufre la red al ser golpeada por la pelota o movida por el viento, he dotado una de las redes de una estructura de huesos y una configuración física que permite un comportamiento similar al de una bandera. Aunque UDK ofrece la posibilidad de crear tal elemento, el resultado no es muy vistoso ni tampoco el esperado, pues aunque el movimiento con el viento simulado resulta realista, la red no se deforma apenas al ser golpeada por la pelota.

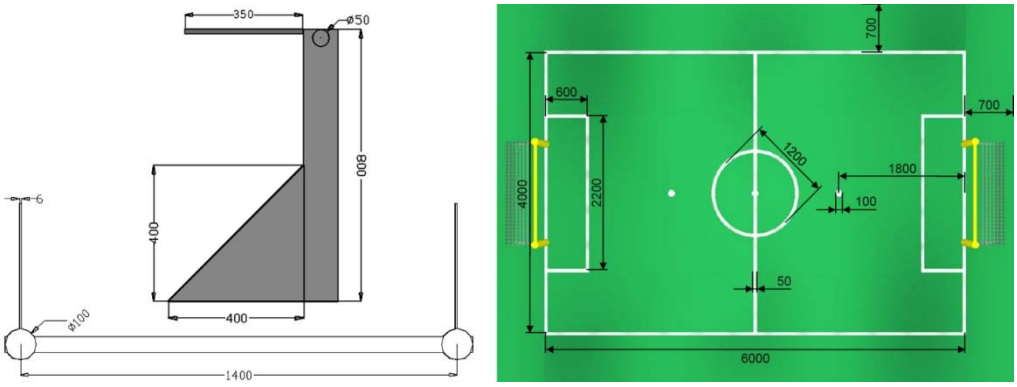

Figura 134. Dimensiones oficiales en mm de un campo de fútbol y sus porterías para la competición RoboCup Standard Platform League 2012

Figura 135. Campo para liga de fútbol entre humanoides NAO con USARSim v.1.2 para UDK

En cuanto al modelo de robot para tal competición de fútbol, he elegido el NAO debido a que desde hace años es el robot oficial de la RoboCup Standard Platform League y, además, ya dispongo de este modelo en variedad de colores para componer

varios equipos. Queda pendiente asignar un número a cada robot y una distribución de colores conforme a la normativa de RoboCup, pero esto último no lo he considerado necesario ni interesante para mi investigación sobre simuladores y lo he dejado como posible mejora para el futuro.

En este entorno para competiciones simuladas de fútbol sería interesante la adición de un par de elementos que considero importantes para mejorar la fidelidad funcional y la evaluación de los logros conseguidos por los usuarios. En primer lugar, podría haber incluido un marcador electrónico que muestre en tiempo real el número de goles marcados por cada equipo. En segundo lugar, sería igualmente interesante añadir un temporizador que indicase el tiempo restante para la finalización del partido, dando por terminado éste al finalizar la cuenta atrás. Ambas cosas pueden hacerse con relativa facilidad usando algunos bloques funcionales de *Kismet*, tal como demuestro en el capítulo 8 de esta tesis, donde detallo un entorno de simulación en el que sí he incluido elementos similares, consiguiendo un importante efecto de realimentación para el usuario.

7.8. Entorno doméstico multicompetición. Otras características relevantes

Este entorno multipropósito lo he diseñado intentando obtener la máxima fidelidad gráfica y funcional posible para determinar las capacidades reales de USARSim v.1.2 para UDK. Un entorno realista con exteriores requiere una gran bóveda celestial y un extenso terreno que no debe dar la impresión de estar vacío. Por este motivo decidido crear un escenario grande e incluir en él todo aquello que resulte interesante de cara a las simulaciones robóticas de carácter educativo. En los apartados anteriores he detallado la obtención de los elementos que componen los espacios destinados a cada una de las mencionadas competiciones, pero existen otros elementos que USARSim me ha permitido desarrollar para este entorno y estimo interesantes por ofrecer un importante extra en realismo.

UDK permite importar objetos dotados de un esqueleto (*SkeletalMesh*) y sus animaciones desde otras aplicaciones como 3D Studio Max, utilizando un formato estándar y cómodo de usar. Esto resulta de vital importancia cuando en las simulaciones se intenta introducir objetos articulados que se mueven, especialmente si tal movimiento varía en función de lo que ocurra en la escena. Como ejemplo de

objetos que ejecutan siempre la misma animación pueden destacarse los brazos robot o la maquinaria de los ambientes industriales que se reproducen en competiciones robóticas simuladas como la mencionada VMAC [51]. Un ejemplo de animaciones que dependen del estado de la simulación lo podemos encontrar en los seres humanos que deben ser encontrados y rescatados en los entornos objeto de desastres reproducidos en competiciones robóticas como RoboCup Rescue Simulation League [50]. Estos humanos deben ejecutar diferentes animaciones como correr, caminar agachados, estar tumbados en el suelo moviendo los brazos, levantarse, saltar, etc. UDK dispone de un entorno gráfico para gestionar animaciones de objetos definidos como *SkeletalMesh*, el cual permite crear conjuntos de animaciones que son controladas por bloques funcionales de *Kismet* durante las simulaciones.

En este entorno he incluido algo de fauna que se mueve por la escena de manera cíclica pero suficientemente convincente para crear una ilusión de vida inteligente en el usuario, incrementando así la fidelidad funcional. Se trata de un cangrejo que he animado en 3D Studio Max y de una gaviota que viene incluida en una de las librerías de recursos de UDK (ver Figura 136). Para otorgar mayor fidelidad a la playa incluida en este entorno he creado un oleaje realista, tres materiales que simulan agua de mar con diferente proporción de espuma y he añadido sonidos propios de una playa con gaviotas.

Figura 136. Animales que ejecutan animaciones simulando la existencia de vida en el entorno

Si bien estas animaciones podrían considerarse innecesarias por no tener relación directa con ninguna de las competiciones virtuales para las que he creado este entorno, estimo que su inclusión incrementa el grado de realismo y la capacidad del entorno para ofrecer una experiencia inmersiva y atractiva. Además, la creación de tales animaciones me ha permitido ahondar en el conocimiento de las capacidades de

USARSim para UDK en cuanto a animación. A este respecto me gustaría destacar la funcionalidad extra que la combinación de movimientos animados con otros simulados puede aportar a las simulaciones en general.

Con el incentivo de una presentación para un proyecto de investigación del Departamento de Ingeniería y Ciencia de los Computadores de la UJI, llevé a cabo tal combinación de movimientos animados con simulados obteniendo muy buenos resultados. En dicha presentación un robot aéreo debía patrullar un edificio público en busca de objetos sospechosos de manera que si encontraba uno este robot avisaría por WIFI a un robot terrestre cuya función sería recoger el objeto sospechoso y llevarlo a un lugar seguro. Realicé un vídeo[73] obtenido de UDK donde el movimiento de los robots correspondía a una animación, pero el agarre del objeto sospechoso (una mochila olvidada en un hall) era simulado (ver Figura 137). El resultado fue totalmente convincente y realista y lo considero de gran aplicación a las simulaciones robóticas.

Figura 137. Combinación de movimientos animados y simulados en una secuencia de detección y agarre. El robot terrestre cuenta con un brazo con el que toma una maleta olvidada en el suelo

[73] http://www.youtube.com/watch?v=dJJmZ6cKISA

7.9. Conclusiones

Las competiciones robóticas son un exigente campo de pruebas para los simuladores dada la diversidad de algoritmos, sensores y actuadores que intervienen en la definición y ejecución de los movimientos de los robots implicados. Equipos de robots interactúan entre ellos compartiendo información y estrategias para alcanzar un objetivo común, el cual a su vez puede ir cambiando con el tiempo. Puntuaciones, tiempo transcurrido o restante, situaciones o comportamientos antirreglamentarios, etc. deben ser tenidos en cuenta por la inteligencia artificial del simulador y ser ofrecidos en tiempo real a los usuarios para que éstos conozcan el estado de la competición en todo momento. Todas estas exigencias deben ser tenidas en cuenta a la hora de seleccionar un simulador para crear entornos destinados a acoger competiciones robóticas, ya que cada simulador ofrece unas capacidades diferentes y requiere una manera de trabajar distinta.

El objetivo principal de esta tesis es la obtención de simulaciones didácticas y atractivas en robótica. Para ello los simuladores han de tener capacidad para incluir en sus entornos la mayor cantidad posible de aquellos elementos que facilitan tales simulaciones. A continuación expongo los puntos fuertes y débiles que han aflorado en los simuladores bajo estudio durante la creación de entornos de competición, centrándome en aquellos atributos importantes para la obtención de experiencias didácticas y atractivas (ver Tabla 15).

El simulador Webots 7 ofrece modelos de robots realistas y configurables de manera relativamente sencilla mediante un entorno de edición gráfico. Webots permite, además, introducir elementos personalizados (llamados *protos*) dotados de inteligencia que otorgan gran funcionalidad a sus entornos mediante la programación en C de su comportamiento. Haciendo uso de estos *protos* he creado (entre otras cosas) los sensores de suelo de los Pioneer 3AT luchadores de sumo y se podría, por ejemplo, realizar un marcador completo para una liga de fútbol. Por otra parte, esta manera de programar la inteligencia del entorno (no necesariamente la de los robots), aunque probablemente acertada para los usuarios estudiantes o investigadores, resta mucha fluidez al desarrollador de escenarios. En cuanto a realismo gráfico, cabe destacar que la mediana calidad de la iluminación, la pobreza de materiales, la inexistencia de varios niveles de detalle (LODs) y, en general, la dificultad para crear entornos grandes con presencia de numerosos objetos convierten Webots 7 en poco recomendable si se buscan experiencias inmersivas.

USARSim v.1.2 para UDK ofrece un entorno de desarrollo gráfico muy completo que facilita enormemente la creación de entornos de simulación. Al igual que Webots y Gazebo, USARSim dispone de un gran número de modelos de robots, sensores y actuadores listos para usar, pero la cantidad de recursos que USARSim ofrece para el desarrollo de entornos es infinitamente superior. Esta importante característica se ve sustancialmente incrementada por la excelente capacidad de USARSim para importar todo tipo de recursos desde otras aplicaciones de CAD. En cuanto a la fidelidad gráfica y funcional de los modelos, USARSim permite disfrutar de modelos altamente realistas manteniendo la carga de computación en niveles aceptables en todo momento. Esto es debido a la conjunción de su excelente editor de materiales con la capacidad de manejar diferentes niveles de detalle. Hay que indicar que cada nivel de detalle consta de sus propios materiales y sus envolventes de colisión, por lo que no solo cambia el número de polígonos, sino también el tamaño de sus texturas y la complejidad de sus contornos de colisión. Una de las características de USARSim más diferenciadoras en cuanto a la fidelidad de las simulaciones es la posibilidad de añadir sonidos de contacto a todos los objetos. Estos sonidos incrementan notoriamente el realismo y la realimentación para el usuario, el cual no necesita fijar su atención en los objetos para saber cuándo han entrado en contacto unos con otros, han sido tocados por un robot o han caído al suelo. El completo editor gráfico de inteligencia artificial *Kismet* permite dotar de cualquier funcionalidad que se desee tanto al escenario como a los objetos que lo componen. No obstante, cuando se trata de definir nuevos robots, sensores o actuadores, debe realizarse mediante un lenguaje de scripts propio de UDK (*UnrealScript*) que debe aprenderse primero. En este sentido, USARSim resulta menos óptimo que Gazebo o Webots.

Gazebo 2.2 dispone de una activa comunidad de desarrollo y de unas capacidades en constante evolución. Las versiones de este simulador se suceden con rapidez en los últimos meses y sus librerías de recursos (tanto robots, sensores y actuadores como objetos de cualquier tipo) están en constante crecimiento. Esto convierte a Gazebo en uno de los simuladores de robótica de mayor proyección en la actualidad y, por tanto, en una apuesta segura si se está buscando un simulador para utilizarlo a medio o largo plazo. En la actualidad, sin embargo, Gazebo carece de herramientas suficientes para permitir un fluido desarrollo de entornos virtuales. El trabajo con Gazebo debe realizarse en buena parte sin un entorno gráfico que agilice el trabajo, resultando poco intuitivo para el usuario, el cual puede llegar a desorientarse por la generación de una cantidad importante de archivos aún en el caso de escenarios

pequeños. La capacidad de Gazebo para importar recursos de otras aplicaciones de CAD es muy limitada, así como la información o tutoriales acerca de cómo hacerlo, dificultando el trabajo del desarrollador de modelos y escenarios. La fidelidad gráfica y funcional de muchos de los modelos que Gazebo resulta muy por debajo de las capacidades de cómputo del hardware actual, siendo frecuente encontrar robots de baja resolución tanto en número de polígonos, como en envolventes de colisión o tamaño de las texturas. Muchos de estos modelos carecen totalmente de texturas, las cuales han sido reemplazadas por colores uniformes o incluso presentan errores de diseño 3D como los ya mencionados grupos de suavizado, quizá debido a que sus creadores tengan un perfil más orientado a la informática que al diseño 3D. Estos errores de diseño 3D también pueden observarse en algunos modelos de USARSim, debido probablemente al mismo motivo (ver Figura 138). En cuanto la capacidad de Gazebo para generar escenarios realistas, ésta no difiere mucho de la de Webots, pues comparten prácticamente idénticos problemas al respecto.

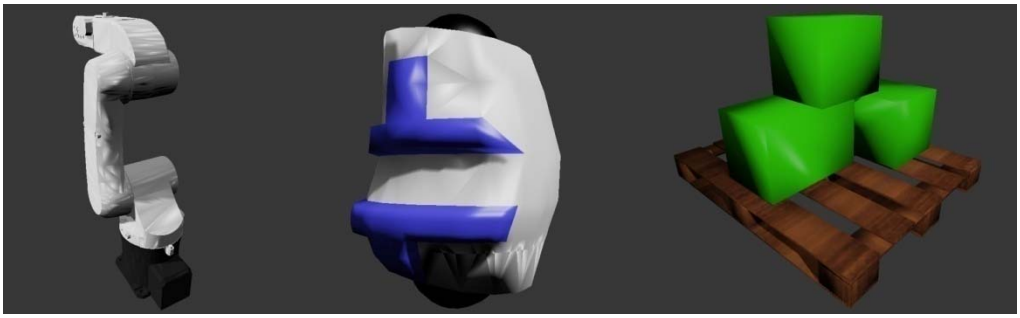

Figura 138. Algunos errores de configuración de grupos de suavizado en USARSim v1.2 para UDK

En resumen, un simulador idóneo para robótica sería básicamente aquél que permitiera el uso de los middlewares y entornos de programación más comunes y potentes, la creación de modelos de alta fidelidad tanto gráfica como funcional y un fácil desarrollo de escenarios que ofrecieran al usuario información y realimentación en tiempo real durante una atractiva experiencia inmersiva. Aunque ninguno de los simuladores robóticos analizados da la impresión de ser perfecto, vistas las capacidades que han mostrado durante la generación de los modelos y entornos descritos, estimo que USARSim v.1.2 para UDK es con diferencia el que más se aproxima al simulador idóneo.

Característica	Webots 7	Gazebo 2.2	USARSim v.1.2 para UDK
Materiales multitextura	No	No	Si
Niveles de detalle	1	1	4
Fidelidad de los modelos de robots incluidos	Alta	Baja	Alta
Fidelidad de la bóveda celestial	Baja	Baja	Alta
Fidelidad del terreno	Media	Media	Alta
Sonido ambiental	No	No	Si
Sonido 3D	No	No	Si
Sonido de contacto entre objetos	No	No	Si
Cálculo automático de envolventes de colisión	No	No	Si
Cálculo automático de niveles de detalle	No	No	Si
Sistemas de partículas	No	No	Si
Objetos con física de trapo	No	No	Si
Simulación de lámina de fluido	No	No	Si
Animación de personajes/objetos	No	No	Si
Animación de texturas	No	No	Si
Entorno destructible	No	No	Si
Facilidad de uso para el desarrollador	Media	Baja	Alta
Facilidad de instalación	Alta	Media	Media
Gratuito	No	Si	Si
Open source	No	Si	Si

Tabla 15. Tabla comparativa de las capacidades de Webots 7, Gazebo 2.2 y USARSim v.1.2 para UDK

Capítulo 8

SIMULACIONES ONLINE

El uso de Internet y sus contenidos multimedia ha traído nuevas oportunidades para el aprendizaje de robótica. Ya sea como sustituto o complemento de lecciones en el aula, hoy en día los recursos online ofrecen acceso a material de aprendizaje y aplicaciones de auto-evaluación de la adquisición de conocimientos [159]. Las simulaciones online representan, por tanto, un excelente recurso educativo que debe estudiarse y optimizarse para facilitar el acceso a los conocimientos de robótica a través de Internet.

8.1. Introducción

Los cursos online estándar sobre robótica presentan algunos inconvenientes inherentes a la formación a distancia. En primer lugar, los alumnos deben implementar sus algoritmos de programación de robots mediante el uso de simuladores, middlewares, entornos de programación, máquinas virtuales, etc. Se trata en muchas ocasiones de numerosos paquetes de software que primero hay que descargar para después instalarlos y configurarlos adecuadamente para que el conjunto funcione de forma correcta. En el capítulo 6 de esta tesis he expuesto la multiplicidad de software que en ocasiones es necesario usar para programar robots, el cual se incrementa si éstos son simulados. En segundo lugar, también juega en contra de los propósitos del curso la diversidad tanto hardware como software de los propios equipos informáticos de los alumnos. Tanto el sistema operativo como la escasa potencia de algún equipo puede dificultar o incluso impedir la realización de determinados ejercicios prácticos. En tercer lugar, en la actualidad la interacción con robots reales es muchas veces insustituible, motivo por el que hay que seguir investigando en simulaciones que resulten tan enriquecedoras como la experiencia con robots reales [42].

Los laboratorios remotos y los sistemas robóticos en línea llevan en funcionamiento casi dos décadas con un éxito considerable [46]. Con la llegada de middleware multiplataforma [47] y la adopción de nuevos y potentes estándares World Wide Web [48], podríamos estar entrando en una nueva era dorada para los laboratorios basados en la web.

La disponibilidad de este tipo de plataformas seguramente aumentará la productividad de la comunidad investigadora, convirtiéndose a su vez en un recurso educativo de un valor incalculable para estudiantes y público interesado. Sofisticadas plataformas robóticas inteligentes de todo el mundo podrían ser accesibles con el único coste de una conexión a Internet para el usuario.

Hoy en día, existe una gran cantidad de sistemas inteligentes habilitados para ser controlados a distancia a través de Internet, ofreciendo al usuario remoto la posibilidad de monitorizar el estado de sus sensores y salidas. Un ejemplo impresionante es el PR2 Remote Lab [160], el cual permite que una amplia comunidad de investigadores de todo el mundo tenga acceso a una moderna y cara plataforma robótica.

Sin embargo, la mayoría de tales sistemas están construidos ad hoc con sus soluciones a medida para gestión y desarrollo, observándose la falta de un marco estandarizado para laboratorios remotos. Esto unido al frecuente dilema entre ofrecer capacidades o mantener la seguridad impide una extensión generalizada del acceso a dichos sistemas. En cuanto al mantenimiento de la seguridad se puede citar que, por lo general, la interfaz de los laboratorios remotos sólo permite el control de algunos elementos del robot. En algunos casos, se proporcionan capacidades de scripting para la ejecución de un conjunto limitado de comandos [161].

Prácticamente desde su concepción entre finales de los 80 y principios de los 90, Internet se utilizó para permitir que sus usuarios supervisaran o interactuaran con robots y sistemas autónomos remotos [46].

Con el principal logro de estar en línea durante más de diez años, el Telerobot de la Universidad de Western Australia (UWA) se ha convertido en uno de los laboratorios remotos más populares, proliferando desde su aparición sistemas similares en todo el mundo [162] [163] [164] [165] [166] [167] [168].

El PR2 Remote Lab [167] [160] representa un hito en sistemas de robots online. Los intentos anteriores se centraron en experimentos sencillos y el aprendizaje en línea, y no se basan en marcos de middleware robóticos compartidos. Este laboratorio utiliza unas herramientas software llamadas *Robot Web Tools* [169], las cuales son una colección de módulos y herramientas de código abierto para la creación de aplicaciones orientadas a robots basados en web, lo que permite a las aplicaciones web interactuar con una gran diversidad de robots que ejecutan ROS.

Aunque la mayoría de los laboratorios remotos existentes han demostrado ser de gran valor para la difusión del uso y el conocimiento de los robots online, parece que existe un importante vacío que aún no se ha llenado: la necesidad de proporcionar una manera simple, transparente y segura de ejecutar programas reales para los usuarios de un laboratorio remoto [49].

8.2. Programación online de robots remotos

En los últimos años ha habido una proliferación de sitios web educativos centrados en la programación interactiva en línea. Algunos de ellos son *MOOCs* (siglas en inglés para cursos masivos online abiertos) como los que ofrecen las empresas Coursera[74] o Udacity[75]. Otros vienen sin asociaciones universitarias o procesos de certificación, como Codecademy[76] o Khan Academy[77].

Los entornos de programación online trabajan directamente en el navegador del usuario sin necesidad de descargar e instalar un compilador. El abanico de lenguajes de programación disponibles ofrece una amplia gama que incluye Javascript, Ruby o Python [170], así como lenguajes más simples dirigidos a jóvenes estudiantes como Logo [171] o lenguajes gráficos como Scratch [172].

R^{3}PO (ROS-based Remote Robot Programming and Operation) es una iniciativa para llevar los existentes laboratorios robóticos remotos a una nueva dimensión, añadiendo la flexibilidad y potencia de escribir código ROS en un navegador de Internet y ejecutarlo en el robot remoto con un solo clic. El código se ejecuta en el servidor del robot a plena velocidad, es decir, sin ningún retardo de comunicación, y

[74] http://www.coursera.org/
[75] http://www.udacity.com/
[76] http://www.codecademy.com/
[77] http://www.khanacademy.org/cs

la salida del proceso es devuelta al programador. Toda la información sobre tal ejecución, almacenada en ROS topics, queda disponible para su descarga desde el servidor para su posterior análisis. Construido sobre Robot Web Tools, R³PO está preparado para funcionar con cualquier robot o simulador basado en ROS.

La Figura 139 muestra un diagrama de bloques del sistema R³PO. El lado del servidor R³PO se basa en las mencionadas Robot Web Tools, para su comunicación con los clientes. Además, R³PO también se comunica directamente con ROS para empezar nuevos procesos de manera dinámica tales como la ejecución de los programas cliente o el registro de datos mediante *rosbag* [173].

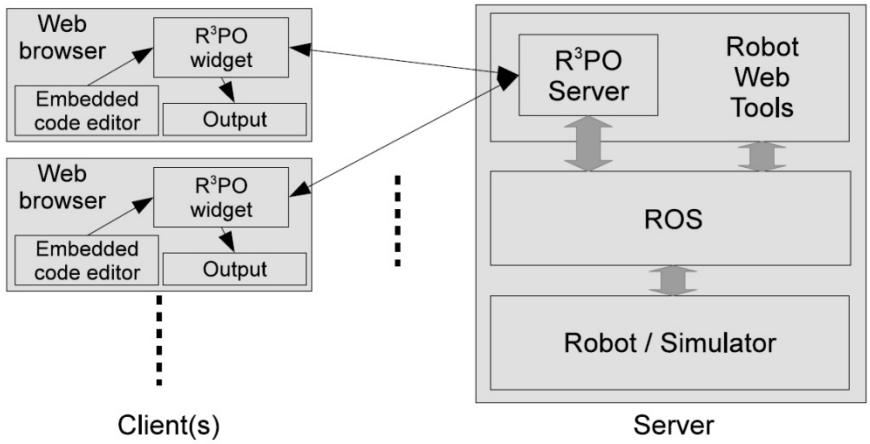

Figura 139. Vista general de R³PO. El bloque de la izquierda corresponde al lado del usuario (cliente), mientras que el de la derecha corresponde al servidor

La Figura 140 representa una vista más detallada de la interacción entre los lados de cliente y servidor. El código del programa se escribe en el lado del cliente mediante un editor integrado en el navegador. Al pulsar el botón de ejecución, el cliente R³PO genera una meta que se compone de un identificador para el programa, el código en sí mismo como una cadena y un parámetro adicional que indica si una bolsa de temas ROS debe ser guardada en memoria o no. Esta información se envía desde el cliente al servidor a través de *rosbridge* [174] y su capa de transporte *WebSocket* [174], de manera similar a la empleada por el PR2 Remote Lab.

A continuación, el código se ejecuta con la herramienta *rosrun* [173] con su salida estándar redirigida al servidor R³PO. Tras la finalización del proceso de usuario, el proceso rosbag es detenido, su salida es guardada en un archivo y devuelta de nuevo al cliente para un posible análisis de resultados.

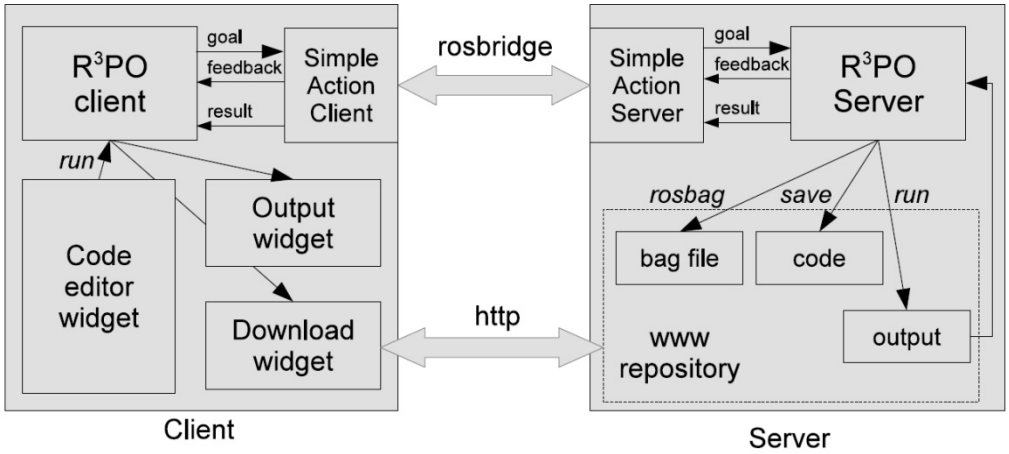

Figura 140. Esquema con el detalle de la interacción entre cliente (usuario) y servidor de R³PO

Además, puede obtenerse realimentación adicional del proceso gracias a la estrecha integración de R³PO con Robot Web Tools: los widgets existentes permiten visualizar un mapa en 2D, un modelo de robot 3D, o un flujo de MJPEG procedente de una cámara remota [169].

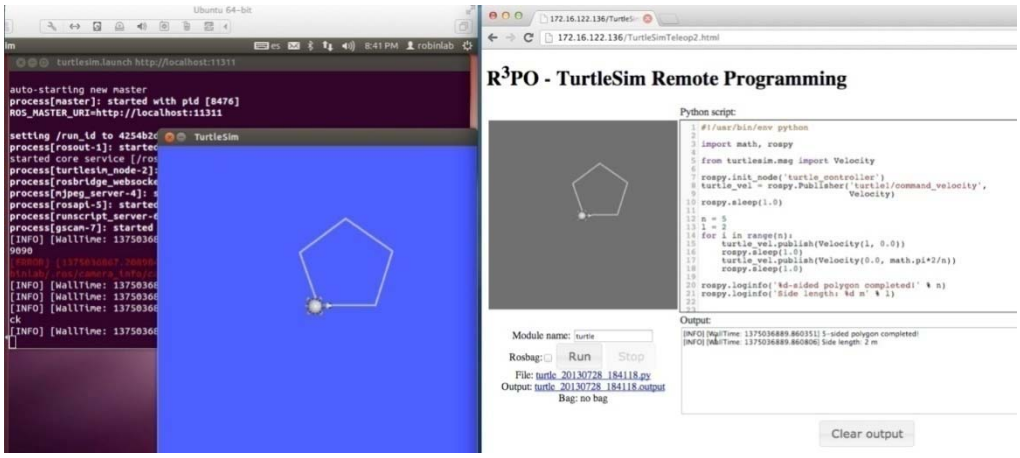

Figura 141. Programación remota con TurtleSim mediante R³PO. A la izquierda se observa el escritorio del servidor, mientras que a la derecha está el escritorio del cliente

La Figura 141 muestra un ejemplo de uso de R³PO donde se programa remotamente el simulador TurtleSim[78] [175]. El código escrito por el usuario cliente en su navegador (ventana de la derecha) ha sido enviado al servidor y ejecutado por TurtleSim (ventana izquierda). La salida se devuelve y se muestra en el navegador.

[78] http://wiki.ros.org/turtlesim

R[3]PO ha evolucionado en dirección a la docencia en robótica enriqueciéndose con materiales pedagógicos hasta convertirse en un sistema mucho más completo llamado Robot Programming Network (RPN) [49] [175]. RPN es una iniciativa para la creación de una red de laboratorios de robótica de carácter educativo con capacidades de programación remota. Se compone tanto de materiales de cursos abiertos en línea como de servidores listos para ejecutar y probar los programas escritos por los estudiantes a distancia (ver Figura 142). Los estudiantes tienen acceso a servidores en línea, donde se identifican y ejecutan su código de programación. Los servidores ejecutan diversos de entornos de simulación, permitiendo también el acceso a robots reales al finalizar con éxito los módulos del curso. El uso de middleware robótico estándar (ROS) y la naturaleza distribuida de RPN permite que el sistema pueda extenderse a un gran número de entornos de aprendizaje, plataformas robot y laboratorios remotos con el objetivo de construir una gran red social para la enseñanza online de robótica.

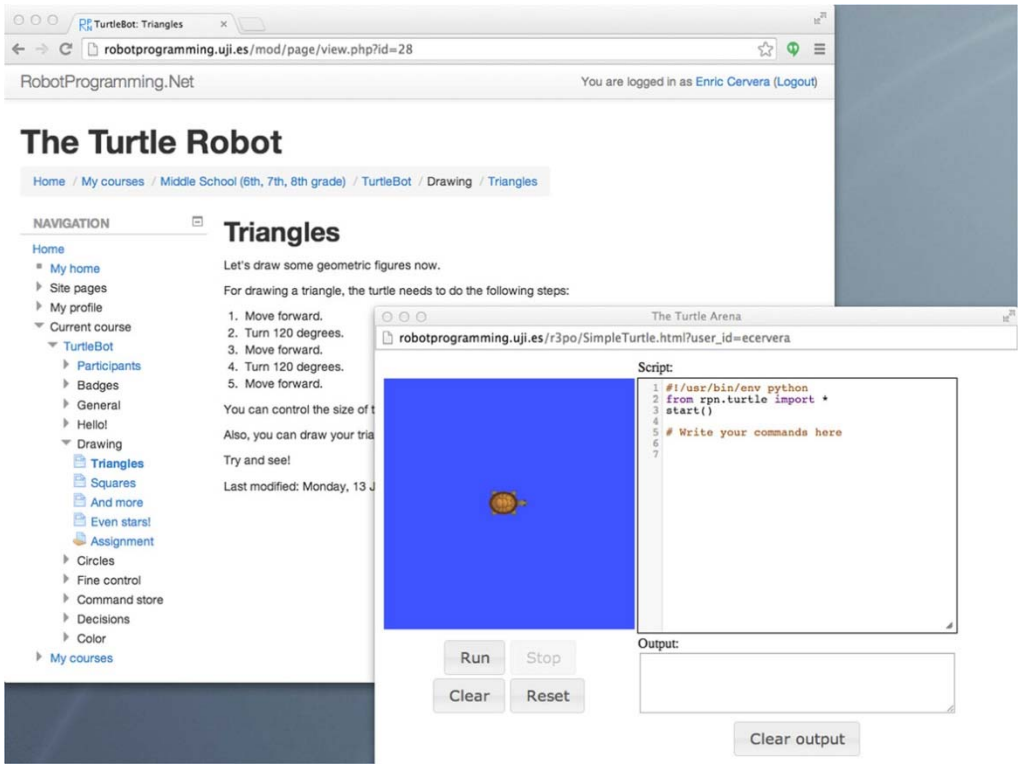

Figura 142. Vista general de RPN visto desde el lado del usuario. En la imagen se aprecia el simulador TurtleSim integrado en una plataforma para la enseñanza online

La Figura 143 muestra una página con los mejores registros de los alumnos en un ejercicio con el simulador TurtleSim y la plataforma RPN. Esta idea puede ser exportada a cualquier simulación donde pueda valorarse el resultado obtenido mediante una puntuación con el fin de proporcionar autoevaluación e incrementar la motivación por medio de la competición entre estudiantes.

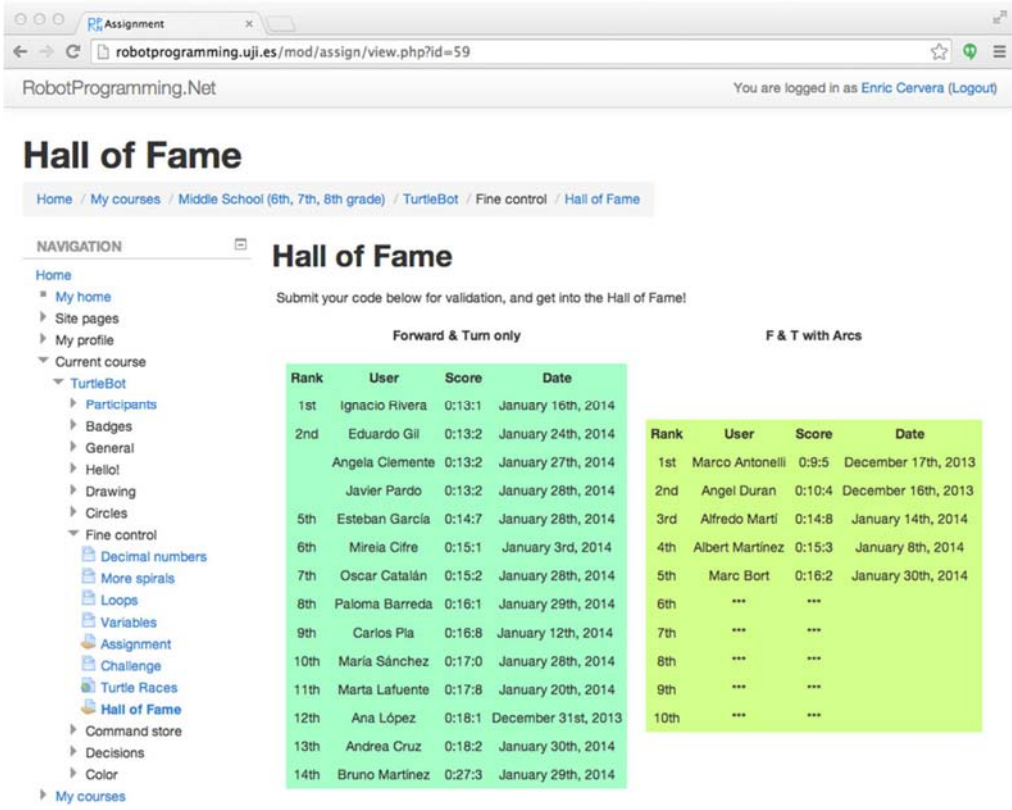

Figura 143. Lista de los mejores registros obtenidos con el simulador TurtleSim de un curso online piloto con Robot Programming Network (RPN)

La Figura 144 muestra la arquitectura general de RPN. El usuario se conecta a Internet a través de un navegador, y se concede el acceso al servidor de un sistema de gestión de aprendizaje (learning management system, LMS) tal como Moodle [176]. Este acceso debe ser autentificado y autorizado por el LMS, para ello el usuario debe iniciar sesión con una cuenta de usuario reconocido o con una identificación proporcionada por otro servicio Web (por ejemplo, Gmail o Facebook). El código de usuario se ejecuta en una máquina virtual, donde se utiliza una interfaz de programación de aplicaciones

(API) segura para interactuar con los módulos de ROS de la red a través de los tópicos y servicios ROS disponibles.

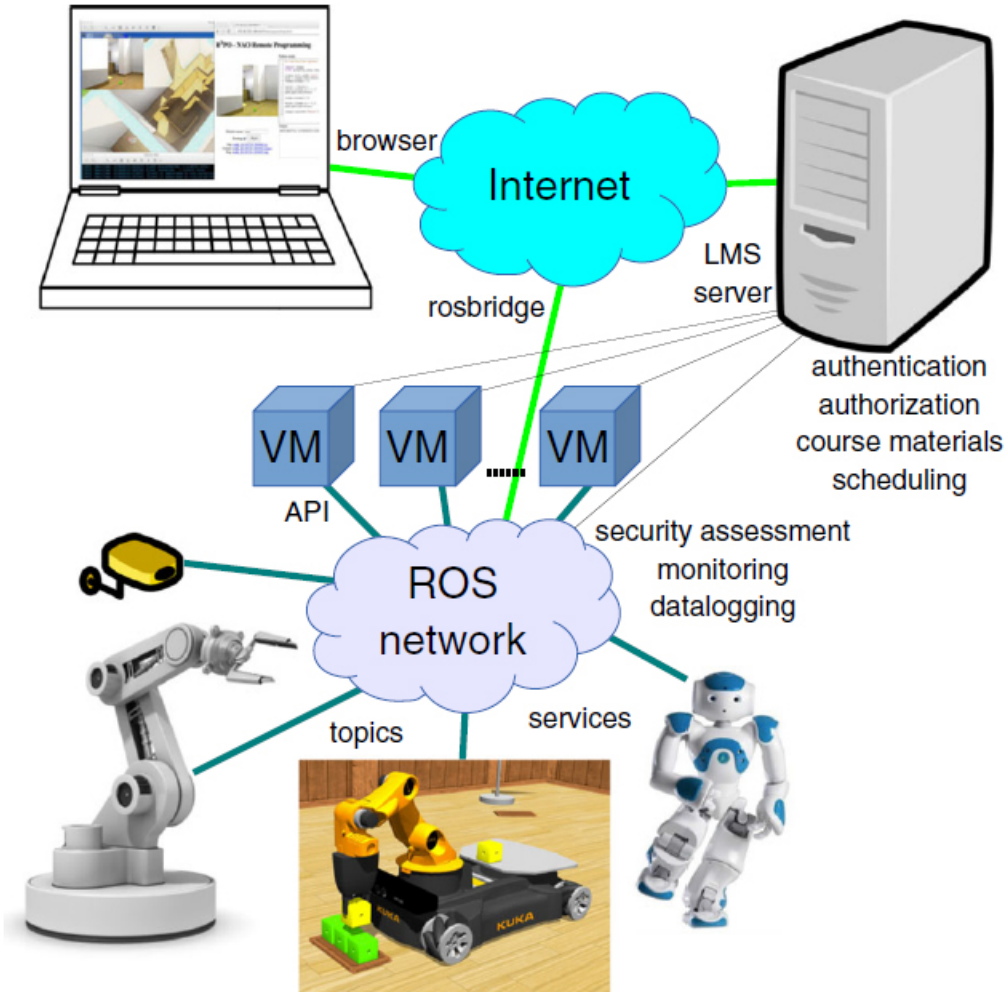

Figura 144. Estructura general de RPN. El usuario a través de su navegador web puede programar online robots reales o simulados

8.3. Entornos para simulación online

Las recientes iniciativas R^{3}PO y RPN han abierto un valioso campo de pruebas para las simulaciones online sobre laboratorios remotos a través de plataformas de aprendizaje como Moodle. Del mismo modo, tanto R^{3}PO como RPN han tenido que ser verificados tanto en funcionalidad como viabilidad, siendo necesario el uso de simuladores adaptados para tal fin.

He desarrollado dos entornos de simulación con un doble objetivo. Por una parte, verificar el buen funcionamiento de la plataforma software que compone R^{3}PO y RPN. Por otra parte, comprobar las capacidades que ofrecen los simuladores bajo estudio para su integración en un curso online de robótica, aprovechando el excelente campo de pruebas proporcionado por R^{3}PO y RPN. Además, atendiendo otras necesidades que describo más adelante, uno de los entornos lo he orientado a ensayos de Sistemas Inteligentes e Inteligencia Artificial, mientras que el otro lo he preparado para realizar simulaciones de competiciones robóticas.

8.3.1. Simulación online de Sistemas Inteligentes / IA con USARSim

El escenario que he preparado para simulaciones online de Sistemas Inteligentes e Inteligencia Artificial lo he desarrollado durante los meses de pruebas de la iniciativa R^{3}PO para programación online. Para este proyecto he escogido el simulador USARSim v.1.2. para UDK, ya que después de los estudios realizados parece el más adecuado cuando se puede emplear un sistema operativo Windows y el desarrollador de niveles tiene experiencia con software CAD. También es relevante el hecho que USARSim es un simulador gratuito para uso no comercial, lo que ofrece a los alumnos del curso online la posibilidad adicional de descargarlo y utilizarlo sin restricciones, aunque esto no es en absoluto necesario gracias al uso de R^{3}PO para la programación online de robots en laboratorios remotos.

He implementado un escenario realista para la simulación de robots humanoides inspirado en un exitoso entorno de ShanghAI Lectures (ver Figura 145). ShanghAI Lectures [158] son fundamentalmente conferencias sobre Inteligencia Artificial y Natural que se llevan a cabo a través de videoconferencia en la Universidad de Zúrich en Suiza, la Universidad de Salford en el Reino Unido, Universidad de Shanghai Jiao Tong en China, y alrededor de otras 12 universidades de todo el mundo. Los estudiantes de las universidades participantes trabajan juntos en los ejercicios

prácticos, utilizando el simulador Webots. La tarea a resolver consiste en guiar un humanoide NAO a través de un circuito de obstáculos, el cual está particionado en sectores con forma de habitación y organizado siguiendo una complejidad creciente. Se trata de un entorno especialmente interesante porque ya ha sido demostrada su adecuación para la formación a distancia sobre contenidos de IA, su estructura es sencilla y, además, permite ser utilizado para competiciones de carreras de obstáculos entre los estudiantes.

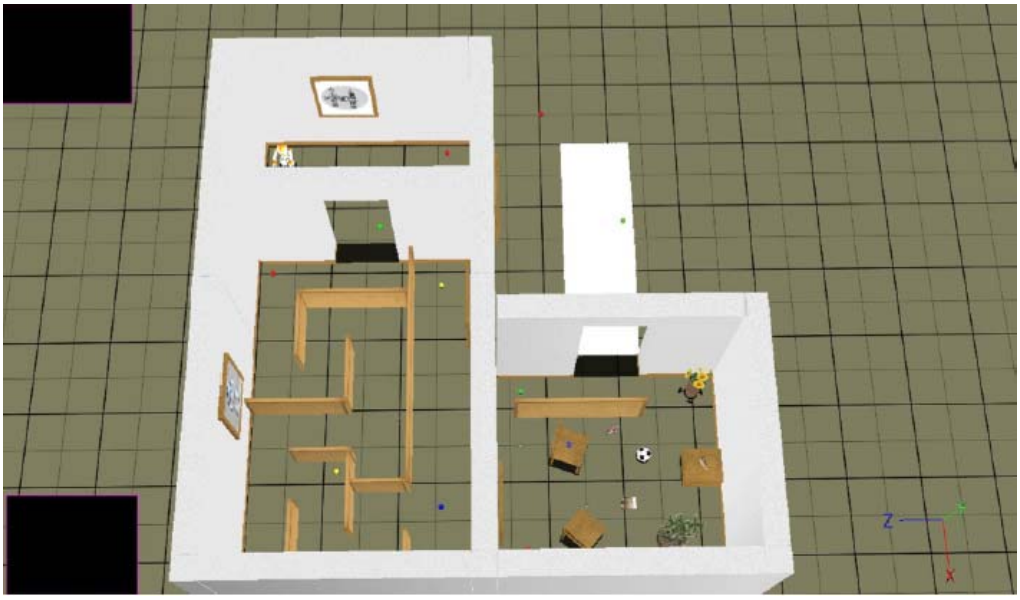

Figura 145. Vista general del entorno de ShanghAI Lectures para la simulación de carreras de obstáculos con el robot humanoide NAO

La Figura 146 muestra una vista general del entorno que he creado para simulaciones robóticas online. Se trata de un escenario dividido en cuatro sectores organizados en orden creciente de dificultad. El robot parte del sector en forma de habitación vacía situado arriba a la izquierda de la imagen, donde debe encontrar la puerta hacia el siguiente sector. Una vez en el sector del laberinto, el robot ha de alcanzar la salida del mismo sin ayuda de marcas de ningún tipo hacia el sector de los obstáculos. Para superar la tercera habitación (abajo a la derecha de la imagen) el robot debe moverse entre una variedad de objetos con diferentes formas y comportamientos físicos hasta la salida a la piscina. El cuarto sector consiste en una estrecha pasarela con pendiente en sus extremos que discurre por encima de una piscina y termina en la meta final del escenario.

Figura 146. Versión mejorada con USARSim del entorno de simulación Webots para IA de ShanghAI Lectures

Con el inicio de la simulación arranca un temporizador que indica el tiempo restante para la finalización de la misma. Este tiempo se ve incrementado automáticamente cuando el robot supera cada sector, de manera que la consecución de cada objetivo facilita el siguiente. Existe también, a modo de autoevaluación, un sistema de puntuación que bonifica cada sector superado y la rapidez empleada en términos de tiempo sobrante, para que los usuarios puedan medir sus progresos y comparar sus logros. El sistema de puntuación es de especial importancia en esta simulación online porque constituye la base para incorporar una competición entre alumnos con un registro de las mejores puntuaciones obtenidas.

Como complemento de interés para los alumnos que deseen ir un poco más allá, he diseminado un conjunto de bolas de colores fácilmente distinguibles sobre el suelo de madera del circuito. Cada vez que el robot golpea una de esas bolas, ésta desaparece emitiendo un sonido característico e incrementándose tanto el marcador "balls" como el tiempo disponible para completar el circuito. El reto consiste, pues, en llegar a la meta marcada como "GOAL" obteniendo la mayor puntuación posible, la cual es directamente proporcional al tiempo sobrante y al número de bolas golpeadas por el robot.

Este escenario presenta sustanciales mejoras tanto en fidelidad audiovisual como funcional sobre el mencionado entorno Webots de ShanghAI Lectures. En cuanto a fidelidad audiovisual, este entorno no difiere mucho de los otros que he desarrollado con USARSim ya descritos en capítulos anteriores, por lo que no voy a detallar esta parte del trabajo. En cambio, centraré la explicación en los elementos que incrementan la fidelidad funcional ya que los considero interesantes e innovadores. En este sentido, destacan en este entorno dos características que extienden sensiblemente la funcionalidad de las simulaciones:

- El usuario recibe cuantiosa información y realimentación en tiempo real tanto a nivel visual como auditivo. Esto permite conocer en todo momento la progresión y el estado en que se encuentra la simulación, facilitando la evaluación de los algoritmos empleados por los estudiantes.
- El escenario contiene objetos frágiles que pueden romperse si reciben un impacto. El entorno deja de ser indestructible y el usuario toma más control sobre la configuración del mismo.

Realimentación visual y auditiva. Evaluación de la simulación

En este entorno el usuario recibe información de todo tipo mientras avanza por los diferentes sectores que lo componen. Nada más empieza la simulación, se muestra en pantalla la lista de tareas que debe realizar el robot humanoide, de manera que la aparición de cada instrucción va acompañada de un corto sonido para llamar la atención del usuario. Del mismo modo, cuando el robot completa un objetivo pasando de un sector al siguiente, el usuario es informado de cuál es el próximo reto a conseguir. Tanto la sobreimpresión en pantalla de esta información como el sonido que la acompaña lo he implementado mediante el gestor gráfico de inteligencia artificial Kismet que incorpora UDK. La simulación termina bien cuando el robot alcanza la meta ("GOAL") o bien cuando se agota el tiempo disponible para ello. En cualquiera de los dos casos el usuario es informado del resultado mediante un mensaje en pantalla y le es arrebatado el control de la cámara, tomando ésta un plano general del escenario mientras suena una música para indicar que la simulación ha terminado. La realimentación gráfica se completa con un triple marcador siempre visible en pantalla que permite conocer en todo momento el número de bolas golpeadas por el robot, la puntuación obtenida y el tiempo disponible para que el robot supere el sector donde se halla.

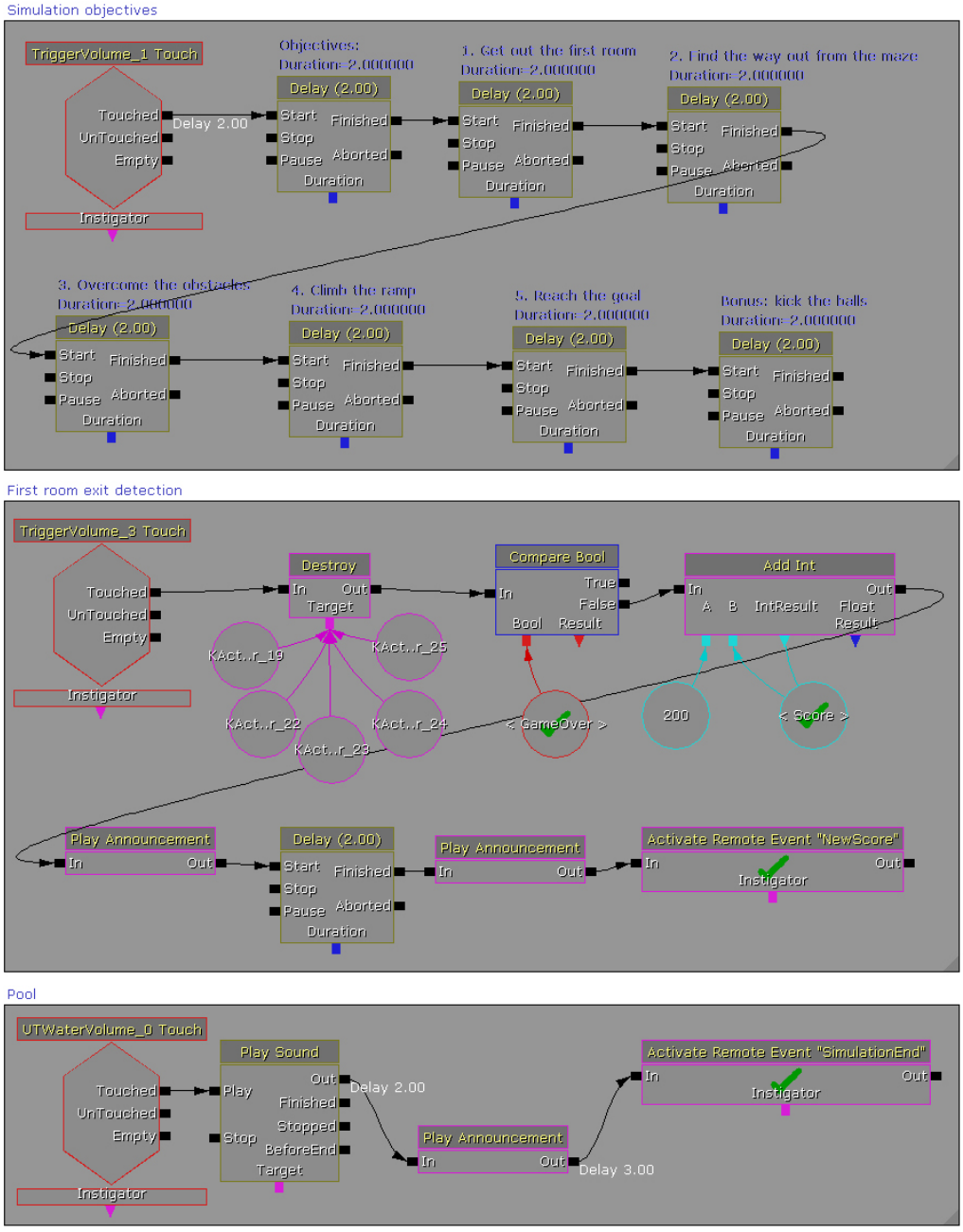

Figura 147. Gestor gráfico de IA de UDK (Kismet) mostrando 3 instrucciones compuestas por bloques funcionales. De arriba a abajo se observan las instrucciones de información de objetivos (arriba), detección de primer objetivo cumplido (centro) y caída a la piscina (abajo)

La Figura 147 muestra una captura de pantalla con tres instrucciones Kismet de las muchas que componen la inteligencia artificial de este entorno. La primera instrucción ("Simulation objectives") informa al usuario acerca de los objetivos a superar durante la simulación. Cuando un robot es lanzado dentro del escenario su entrada es captada por un detector volumétrico y se desencadena una sucesión de mensajes de texto en pantalla (con aviso auditivo) espaciados 2 segundos.

La segunda instrucción ("First room exit detection") muestra cómo se detecta la entrada del robot en el sector del laberinto desde la primera habitación, se informa al usuario del objetivo cumplido y del siguiente reto y, finalmente, se añade la puntuación correspondiente al marcador. La detección del paso del robot de un sector a otro se realiza mediante objetos invisibles que accionan un interruptor volumétrico al ser empujados por el robot. Después de su uso tales objetos son destruidos automáticamente para no interferir en la simulación.

La tercera instrucción ("Pool") ofrece la secuencia de acciones que siguen a la detección de la caída del robot a la piscina. Un sonido de "chof" y un mensaje en pantalla informan al usuario que el robot ha caído al agua y ya no es posible continuar, activándose la instrucción de final de simulación.

La programación gráfica mediante bloques funcionales integrada en el propio editor de niveles de UDK resulta de gran utilidad y sencillez de uso. Además, su carácter gráfico permite al usuario seguir con facilidad el flujo de la información.

La Figura 148 muestra un robot NAO entrando en el sector del laberinto tras abandonar la habitación de inicio. En el centro de la imagen se puede ver el mensaje "Find the way out from this cardboard maze" indicando cuál es el reto del segundo sector. Abajo a la izquierda de la imagen se encuentra el triple contador de logros con el número de bolas conseguidas, el tiempo restante para completar el circuito (en segundos) y la puntuación conseguida. Este marcador con la puntuación es de vital importancia para fomentar la competición entre los estudiantes.

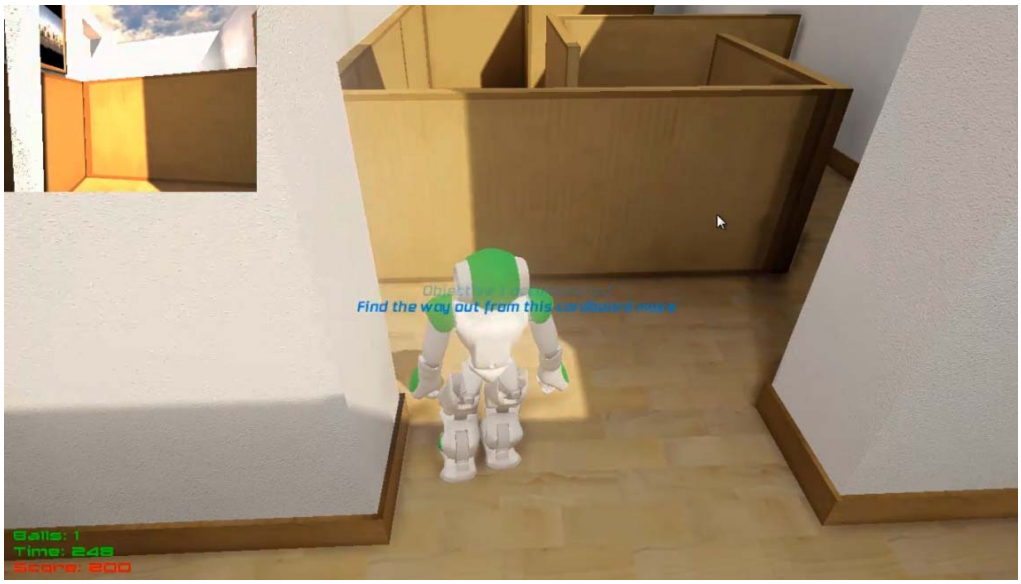

Figura 148. Robot NAO entrando en el sector del laberinto tras abandonar la habitación de inicio en un entorno para simulación y programación robótica online mediante R^{3}PO

Para reforzar la realimentación visual he añadido mensajes de voz que informan del tiempo restante para completar el circuito, de manera que el usuario recibe avisos cada minuto, cuando solo quedan 30 segundos y durante la cuenta atrás de los últimos 10 segundos. Se trata de mensajes de voz en formato *.wav* que ya incorporan las librerías de UDK, pero que pueden ser sustituidos o enriquecidos con nuevos archivos de audio personalizados para cada entorno de simulación. La Figura 149 muestra el aspecto del *Sound Cue Editor*, una de las herramientas fundamentales de UDK para la configuración de sonidos.

En el capítulo 7, en el apartado correspondiente al entorno con ring rodeado de agua para competiciones de sumo, he indicado que lo ideal sería disponer de detección de caída al agua de los robots con el fin de dar el combate por finalizado desde el propio simulador. Aprovechando la presencia de la piscina en el entorno para simulaciones online, he implementado tal detección utilizando Kismet y una serie de sensores estratégicamente colocados. Así, cuando el robot NAO atraviesa la superficie del agua el simulador hace sonar un "chof", se muestra el mensaje "Bath time?" en pantalla y pasados 3 segundos se activa la secuencia de final de simulación. Esta manera de proceder es directamente exportable al escenario del ring de sumo, donde permitiría ofrecer realimentación instantánea a los usuarios competidores dando el combate por terminado desde el propio simulador.

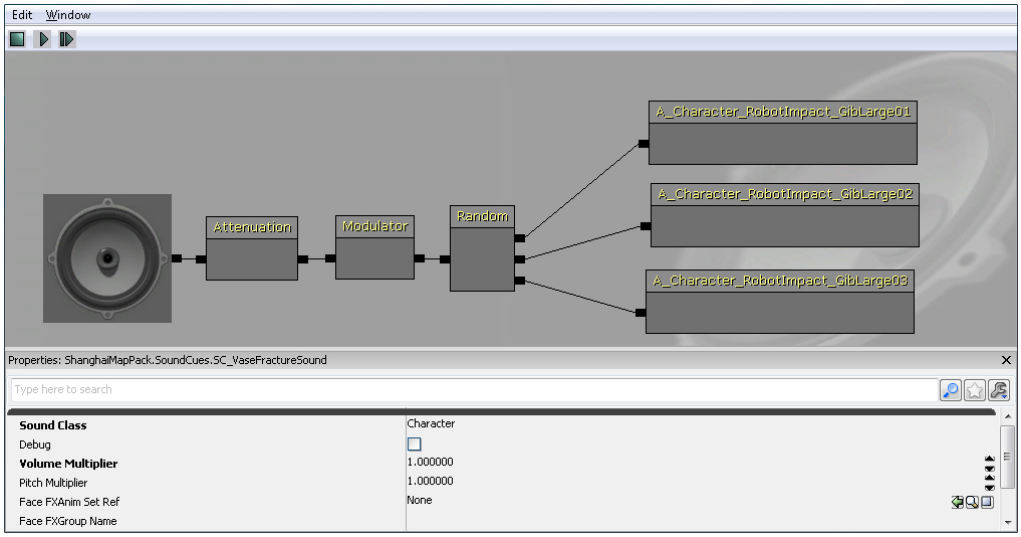

Figura 149. Captura de pantalla del Sound Cue Editor de UDK. Arriba se aprecia la estructura de bloques funcionales del editor, mientras que abajo se muestran las propiedades generales o aquéllas correspondientes al bloque seleccionado

La Figura 150 muestra la red de sensores (en azul) que permiten la detección de los diversos eventos que informan del estado de la simulación. Estos sensores solo son visibles durante la edición del escenario, permaneciendo invisibles durante las simulaciones. Los sensores situados en las puertas permiten detectar el paso del robot de un sector al siguiente, de igual manera que aquéllos colocados en la meta determinan el fin de la simulación. En esta figura destacan los mencionados sensores de caída a la piscina situados sobre la lámina de agua.

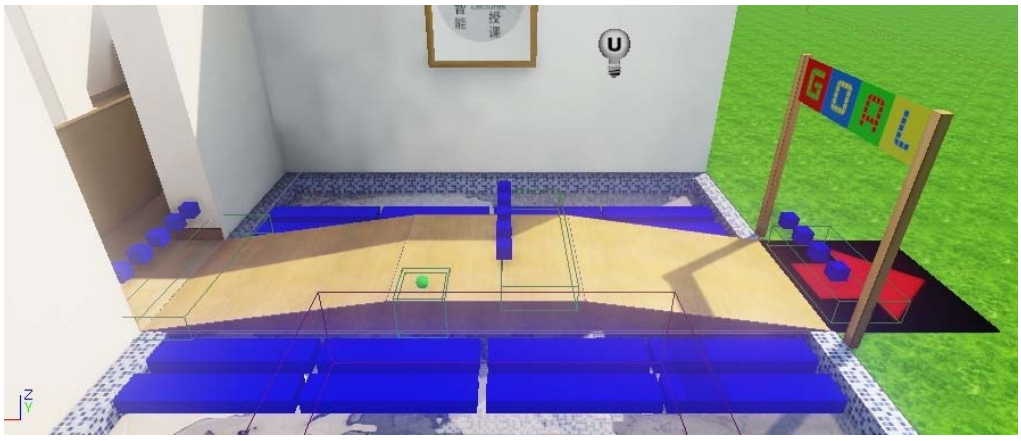

Figura 150. Perspectiva del sector de la piscina tomada con el editor de niveles de UDK. Destaca la red de sensores (en azul) empleados para detectar eventos de diversa índole

Esta realimentación audiovisual es de suma utilidad para mejorar la experiencia del usuario durante las simulaciones. No obstante, la parte auditiva solo es perceptible si el simulador está instalado en el equipo del usuario, dado que la plataforma RPN no es capaz, por el momento de ofrecer los sonidos de la simulación. Aun así, la completa información que aparece sobreimpresa en pantalla es más que suficiente para seguir la simulación y evaluar los algoritmos empleados.

Escenario destructible. Control sobre el entorno

Con el objetivo de seguir avanzando en la introducción nuevas tecnologías que permitan expandir la funcionalidad de las simulaciones, he enriquecido el circuito de obstáculos con objetos destructibles.

En un entorno virtual, cuanto mayor es la fidelidad gráfica mayores son las expectativas de fidelidad funcional que levanta en el usuario [88]. Esto significa que si, por ejemplo, se adorna una habitación con un detallado teléfono, es muy probable que el usuario llegue a pensar que es posible levantar el auricular y que éste de señal. Por lo contrario, la ausencia de funcionalidad o realismo allí donde se presupone o la presencia de incoherencias en el entorno virtual son valoradas negativamente por los usuarios, teniendo conocidos efectos negativos en temas de satisfacción e inmersión [91].

Muchas simulaciones robóticas se llevan a cabo en escenarios con presencia de objetos frágiles o compuestos por piezas que se desensamblan con relativa facilidad al recibir un impacto. Se trata con frecuencia de ambientes domésticos con elementos decorativos como jarrones y figuritas de cristal o porcelana, incluso equipos electrónicos como ordenadores portátiles o teléfonos móviles. Estos objetos tan normales y cotidianos se convierten en extraños y grotescos cuando durante una simulación se manifiestan indestructibles o incluso inamovibles. Este paradójico comportamiento de tales objetos es, sin embargo, extrañamente normal en las simulaciones robóticas y no puede generar sino cierto rechazo en el usuario. Desafortunadamente, la enorme mayoría de simuladores para robótica no contemplan la posibilidad de incluir objetos destructibles.

NVIDIA APEX PhysX Lab[79] es una herramienta software que permite, entre otras cosas, generar realistas patrones de rotura a partir de objetos 3D creados con otras aplicaciones CAD (ver Figura 151). Del mismo modo, también es posible definir las piezas en las que se descompondrá un objeto ensamblado al recibir un impacto.

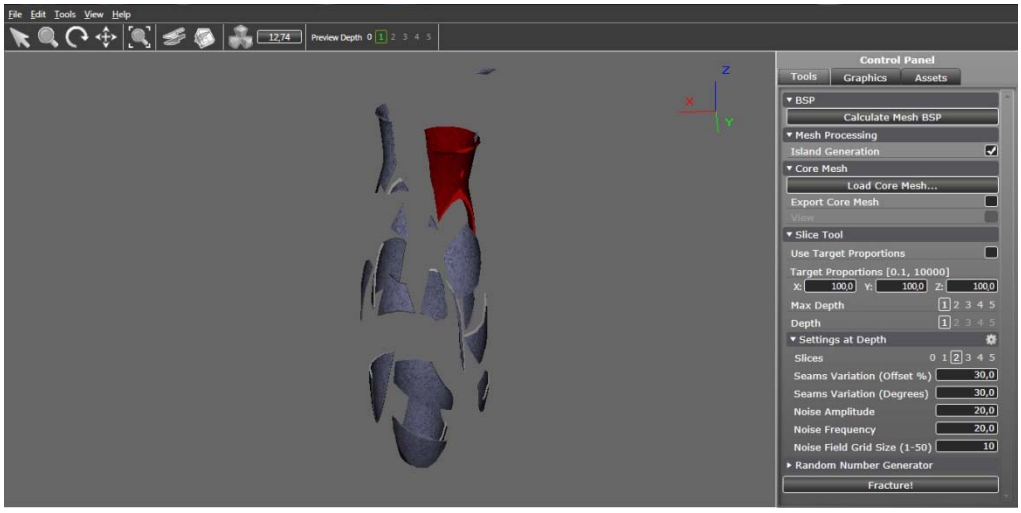

Figura 151. Captura de pantalla del editor de fracturas de NVIDIA APEX 1.1 PhysX Lab. La imagen muestra un jarrón importado de 3D Studio Max, la textura superficial y la de rotura

UDK permite importar de NVIDIA APEX PhysX Lab tanto los modelos 3D como su patrón de fractura y las texturas implicadas, generando un objeto de tipo *Apex `Destructible Asset`* que puede ser configurado para obtener roturas de gran realismo (ver Figura 152).

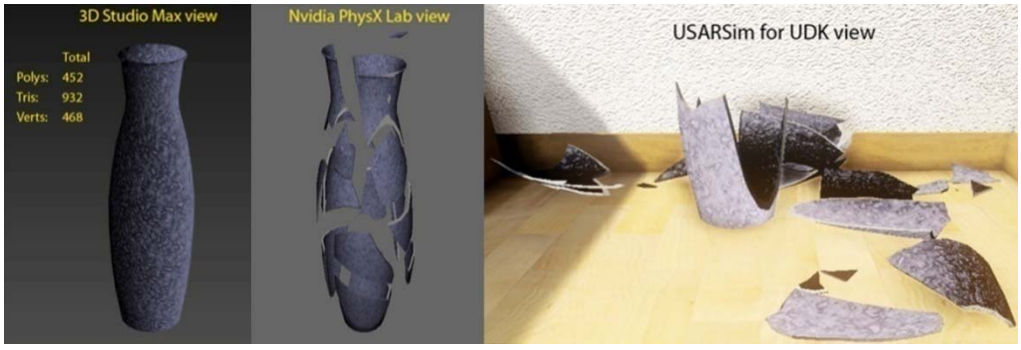

Figura 152. Vistas de un jarrón. De izquierda a derecha se muestra la evolución del jarrón en su paso por diferentes aplicaciones para poder ser destruido durante una simulación con USARSim

[79] http://physxinfo.com/wiki/PhysXLab

Los objetos de tipo Apex Destructible Asset ofrecen al desarrollador de entornos virtuales un control total sobre la malla destructible. Puede definirse la fuerza de impacto necesaria para provocar la rotura, el sonido del objeto al quebrarse y de los fragmentos al caer al suelo, efectos de polvo, humo o fuego generados por la rotura, grado de diseminación de los fragmentos, etc.

La capacidad de USARSim para aprovechar las nuevas tecnologías a través de UDK lo convierte en un simulador muy completo que supera con creces al resto de simuladores tanto en fidelidad gráfica como funcional. Sin duda, la presencia de objetos destructibles en las simulaciones robóticas lleva un paso más allá el nivel realismo, facilitando experiencias más inmersivas y mayor satisfacción en el usuario.

8.3.2. Simulación online de competiciones robóticas.

HUMABOT Challenge

HUMABOT Challenge [106] es la competición robótica oficial para el IEEE-RAS International Conference on Humanoid Robots que se celebró el pasado mes de noviembre de 2014 en Madrid.

El desafío HUMABOT Challenge se caracteriza por utilizar plataformas robóticas de tamaño medio, estando orientado para el uso de robots humanoides como el NAO de Aldebaran Robotics, el DARwIn-Op de Robotis [177] o cualquier otro robot con características similares. Los investigadores participantes se centran en el desarrollo de algoritmos para robots totalmente autónomos, es decir, aquéllos que operan sin intervención humana. La filosofía de esta competición es considerar los robots como una parte integral de la casa, ayudando a sus ocupantes humanos a tener una vida más cómoda.

En la edición 2014 de HUMABOT, los robots debían realizar una serie de tareas en la cocina (ver Figura 153). Se trataba de conseguir que un robot humanoide desempeñara tres labores domésticas diferentes que requerían la programación de algoritmos de naturaleza variada. Estos algoritmos debían resolver, entre otros, problemas de manipulación, agarre y reconocimiento de objetos bajo condiciones de iluminación indeterminadas. Para facilitar la navegación, se permitía a los equipos participantes colocar por el escenario algunas marcas como códigos QR.

Figura 153. Vista general de la cocina utilizada en la competición HUMABOT Challenge 2014 (imagen izquierda) y detalle del mueble principal con los fogones y estantes (imagen derecha)

La primera misión consistía en realizar una tarea relacionada con la seguridad doméstica que radicaba en apagar un fogón. El robot debía identificar qué fogón de la cocina estaba encendido y, seguidamente, apagarlo accionando el interruptor correspondiente (ver Figura 154). La relación entre interruptores y fogones era conocida de antemano.

Figura 154. Secuencia de instantáneas correspondientes a la primera prueba ejecutada por el equipo de la Universidad Politécnica de Cataluña. El robot NAO identifica el fogón encendido y lo apaga accionando el pulsador pertinente

En la segunda prueba el robot debía hacer una lista de la compra. Para ello el robot identificaría qué productos (de un conjunto predefinido) estaban disponibles en los

diferentes estantes de la cocina, deduciendo cuáles faltaban y por tanto habría que comprar. La Figura 155 muestra los seis productos que el humanoide tenía que buscar por la cocina. Se trata de alimentos comunes como pastillas de caldo de pescado, clavo, cereales (muesli), palomitas de maíz, té y café.

La tercera tarea doméstica consistía en preparar una comida. Para superar este último reto el robot debía coger un tomate de peluche de encima de una mesa y dejarlo dentro de una cacerola situada a unos centímetros del tomate (ver Figura 156).

Figura 155. Set de productos que el robot debe reconocer para elaborar una lista de la compra. La fila de arriba corresponde a fotografías de los objetos reales, mientras que la fila de abajo corresponde a capturas de pantalla de los objetos en el escenario simulado

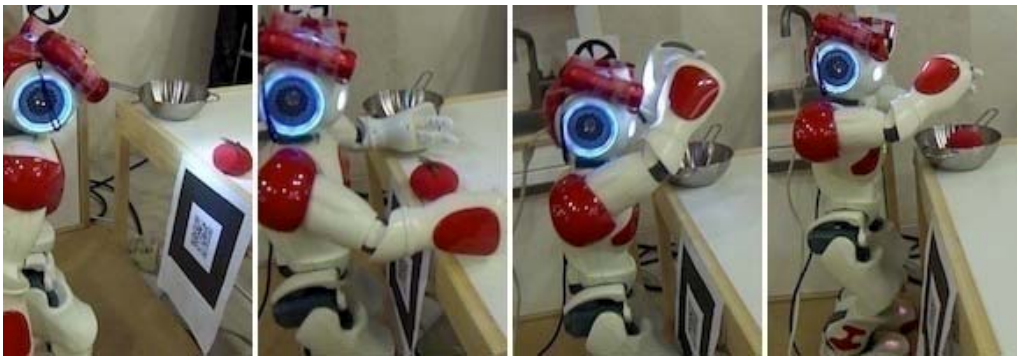

Figura 156. Secuencia de instantáneas tomadas durante unas pruebas previas a la competición. Un robot NAO coge un tomate de peluche y lo deja dentro de una cacerola

Cada prueba se calificaba sobre 20 puntos, de modo que cada equipo tenía derecho a 2 intentos de 3 minutos de duración para realizar cada tarea. Como puntuación se tomaría la mejor calificación de los dos intentos, siendo penalizadas las intervenciones humanas durante las pruebas.

Un aspecto importante para incrementar las probabilidades de éxito de HUMABOT Challenge 2014 era facilitar a los equipos interesados en participar en el reto la posibilidad de reproducir el escenario para que pudieran probar sus algoritmos antes de la competición. Así, los muebles y objetos que conformaban el escenario se eligieron de manera que resultaran fáciles de adquirir, su precio era reducido y los comercializaba una conocida multinacional del mueble.

Aunque la competición estaba orientada a robots reales, siguiendo la filosofía de facilitar el acceso a todos aquéllos investigadores interesados en participar, se dispuso la posibilidad de probar los algoritmos en un simulador. De este modo, los equipos que no pudieran permitirse adquirir una plataforma real podrían familiarizarse con el escenario de la competición por medio de dos entornos simulados con Webots disponibles para ser descargados desde la web oficial de HUMABOT Challenge[80]. En la misma web se incluyó también un enlace e instrucciones para la descarga y uso de una versión de 30 días de prueba de Webots 7, para el eventual caso que algún equipo no dispusiera de licencias de tal simulador. Del mismo modo, se facilitó acceso a una plataforma real para los participantes registrados. Tal acceso se implementó utilizando RPN para la programación online de un robot NAO ubicado en un escenario real (ver Figura 157). Estos equipos tendrían a su disposición un robot NAO el día de la competición.

Figura 157. Robot NAO en el escenario de la competición HUMABOT Challenge programado de manera online a través de la plataforma RPN (Robot Programming Network)

HUMABOT Challenge supuso una nueva oportunidad para el estudio de los simuladores de plataformas robóticas. Tal como he explicado, debía desarrollarse un entorno de simulación que permitiera reproducir fielmente el escenario de la

[80] http://www.irs.uji.es/humabot/

competición. Además, la posibilidad de utilizar dos plataformas robóticas (NAO y DARwIn-Op) requería la creación de dos versiones del entorno para simplificar su uso a los equipos de investigadores participantes. Teniendo en cuenta tales premisas, y conociendo las capacidades de los simuladores robóticos más interesantes de cara a esta investigación, decidí utilizar Webots 7 para desarrollar aquellos entornos. A pesar de que los escenarios para esta competición los desarrollé convencido de la superioridad (siempre en términos generales) de USARSim sobre Gazebo y Webots, creí conveniente el uso de este último debido a una serie de particularidades que a continuación expongo.

En primer lugar, no estaba previsto incluir el simulador en la plataforma RPN por estar todavía en pruebas en aquel momento y para agilizar la introducción de los algoritmos, de modo que los usuarios deberían descargar e instalar el simulador en sus propios equipos informáticos. Webots 7, resulta más sencillo de instalar que USARSim o Gazebo y, además, éstos dos últimos requieren que el usuario sepa desenvolverse por su árbol de carpetas para ubicar correctamente los archivos que definen un escenario y todo su contenido. Esto resultaba de vital importancia, pues los equipos participantes en la competición procedían de varios continentes y debían poder instalar y utilizar el simulador de manera autónoma.

En segundo lugar, Webots 7 incorpora un completo set de plataformas robóticas de alta fidelidad gráfica y funcional que incluye los humanoides NAO y DARwIn-Op necesarios para la competición. Esto no sucede con los simuladores Gazebo o USARSim. Gazebo 2.2 no dispone de ninguna de las dos plataformas robóticas mencionadas y USARSim v.1.2 para UDK no incorpora el robot DARwIn-Op. Esto significa que tanto con Gazebo 2.2 como con USARSim v.1.2 para UDK el usuario debe crear, modificar o adoptar modelos de estas plataformas robóticas antes de poder incluirlos en sus simulaciones.

La Figura 158 muestra las dos versiones del entorno que desarrollé para simular las condiciones de la competición HUMABOT Challenge 2014. La imagen de la izquierda corresponde a la versión para el robot DARwIn-Op, mientras que la imagen de la derecha corresponde al humanoide NAO. Exceptuando los robots, ambos escenarios son idénticos.

Figura 158. Entornos de simulación domésticos realizados con Webots 7 para HUMABOT Challenge 2014

El entorno de simulación para esta competición era aparentemente sencillo pero requirió un trabajo de clonación importante. Para garantizar la máxima similitud con el escenario real tuve que reproducir cada mueble y objeto con la mayor exactitud posible, prestando especial atención a las texturas de los productos que ha de identificar el robot durante la elaboración de la lista de la compra. La Figura 155 muestra el aspecto de tales productos en el simulador, comparados con fotos reales.

En líneas generales, este entorno no difiere demasiado de aquéllos desarrollados con Webots ya descritos en capítulos anteriores, motivo por el cual no voy a detallar su proceso de creación. No obstante, su concepción y posterior uso ha revelado algunas de las mayores ventajas de Webots sobre el resto de simuladores tanto open source como comerciales. Así, Webots se perfila como una de las mejores opciones si se pretende incluir simulaciones de plataformas robóticas en un curso a distancia. La simplicidad de instalación y la disponibilidad de una amplia gama de modelos de robots de alta fidelidad gráfica y funcional facilitan su utilización a los usuarios inexpertos. Además, el entorno gráfico de Webots incorpora una ventana con un editor de texto que cuenta con un compilador C, simplificando notablemente la creación y depuración de código de usuario.

8.4. Pruebas experimentales de R^{3}PO con USARSim

El sistema R^{3}PO se puso a prueba con diferentes configuraciones, incluyendo tanto simuladores como robots reales. Para los ensayos con simulador se eligió el recientemente descrito entorno del circuito de obstáculos desarrollado con USARSim, utilizándose R^{3}PO para programar remotamente un humanoide NAO. Tanto la elección del simulador como la del robot no fueron arbitrarias, sino que responden a

motivos de diversa naturaleza entre los que destaca superioridad audiovisual y funcional de USARSim y la compatibilidad con ROS.

USARSim es superior a Gazebo y Webots en casi todos los aspectos que conducen a simulaciones didácticas y atractivas. En contrapartida, presenta el inconveniente de requerir cierta experiencia en los usuarios que quieran descargarlo e instalarlo. No obstante, en este caso la programación del robot simulado sería online a través de R^{3}PO sin necesidad de tales descargas o instalaciones, por lo que tal inconveniente no resultaba relevante.

ROS da soporte al humanoide NAO gracias a un controlador[81] desarrollado por Armin Hornung y el Laboratorio de Robots Humanoides de la Universidad de Friburgo. Como ROS no soporta directamente USARSim, la comunicación con el robot es posible gracias a que el driver de ROS para NAO se conecta a un módulo personalizado que ejecuta el ya mencionado middleware NAOqi, el cual ha sido desarrollado para USARSim.

Para comprobar las capacidades de programación remota con R^{3}PO en este entorno, se preparó un algoritmo simple cuyo fin era que un humanoide NAO caminase hacia adelante durante un determinado período de tiempo. El código ROS Python para tal algoritmo era el siguiente:

```
---------------------------------------------------------------------------------------------------------------
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twi st
rospy.init_node ('walk')
pub = rospy.Publisher ('cmd_vel', Twist)
rospy.sleep (1.0)
twist = Twist ( )
twist.linear.x = 1.0
pub.publish (twist)
rospy.sleep (5.0)
twist.linear.x = 0.0
pub.publish (twist)
rospy.loginfo ("Done!")
---------------------------------------------------------------------------------------------------------------
```

[81] http://www.ros.org/wiki/Robots/Nao

La ejecución resultante del código se muestra en la Figura 159. A la izquierda se puede ver el simulador, ejecutando tanto USARSim como NAOqi en una máquina virtual de Windows, donde se distingue un robot NAO en el interior del ya descrito circuito de obstáculos inspirado en ShanghAI Lectures y la vista subjetiva de la cámara del robot. En la parte inferior izquierda, una máquina virtual Linux está ejecutando el sistema ROS, incluidos los controladores NAO y el lado del servidor R^{3}PO. A la derecha, se muestra el navegador en la máquina cliente, con el código fuente, la salida y los enlaces a los archivos generados. El *`mjpegcanvasjs`*[82] *`widget`* permite al usuario ver de forma remota lo que capta la cámara del robot. Este reproductor forma parte de las Robot Web Tools y puede integrarse con facilidad en R^{3}PO.

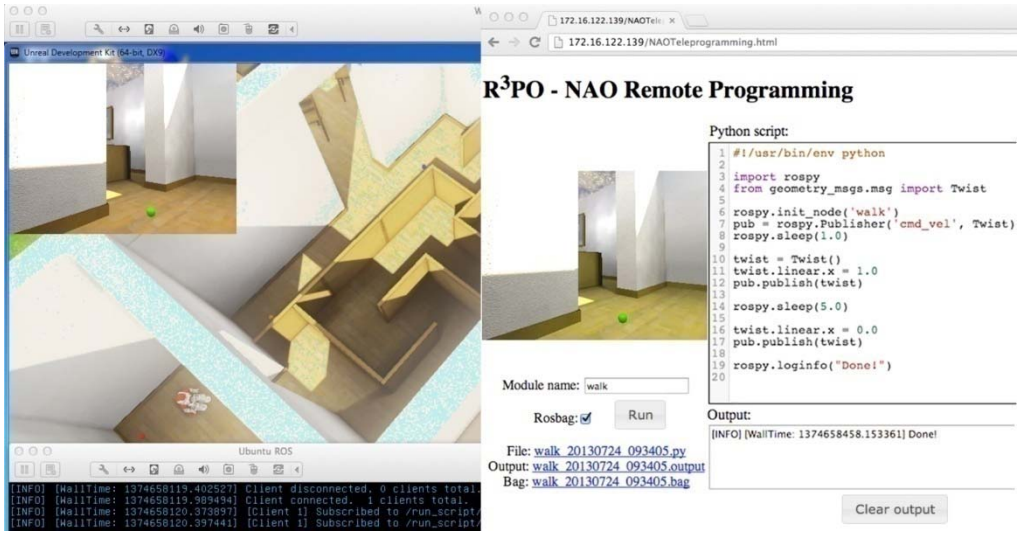

Figura 159. Programación remota del humanoide NAO mediante USARSim para UDK y R^{3}PO

[82] http://wiki.ros.org/mjpegcanvasjs/

8.5. Conclusiones

De lo expuesto en este capítulo concluyo que cuando se plantea el uso de simuladores de plataformas robóticas en el contexto de la formación a distancia es necesario conocer cómo se va a realizar la programación de algoritmos para elegir el simulador más adecuado. Por una parte, lo más usual es que los simuladores se instalen localmente en el equipo informático de cada usuario, donde también se escribe y ejecuta el código con los algoritmos que rigen el comportamiento de los robots simulados. Por otra parte, algunas iniciativas como RPN permiten la programación remota de plataformas robóticas online, ya sean éstas reales o simuladas. En tales casos, los simuladores se encuentran instalados y configurados en un ordenador servidor, el cual ofrece a los usuarios la posibilidad de escribir sus algoritmos localmente en sus propios equipos y ejecutar el código resultante de manera remota en el simulador, utilizando para ello un simple navegador web.

En el primer caso prima la sencillez de instalación, configuración y uso del simulador así como el eventual coste de las licencias software necesarias. Esto es debido a que cada usuario debe instalar y utilizar una copia local del simulador y resto de software necesario (middlewares, entornos de programación, etc.). En este contexto Webots 7 resulta sensiblemente más ventajoso que USARSim v 1.2 para UDK o Gazebo 2.2 por su sencillez de instalación y uso, por incorporar gran cantidad de plataformas robóticas de alta fidelidad, por integrar un entorno de programación y un compilador C/C++ y por ofrecer a los usuarios una versión gratuita del simulador con 30 días de prueba, tiempo normalmente suficiente para implementar una gran cantidad de algoritmos.

En el segundo caso, el hardware del usuario así como su habilidad para instalar y configurar simuladores y middlewares dejan de ser variables relevantes en el proceso de selección de un simulador. Del mismo modo, tampoco resulta un importante la capacidad económica de los usuarios para adquirir licencias software. Así pues, la programación online de plataformas robóticas remotas permite elegir simuladores atendiendo casi exclusivamente a sus prestaciones como tales, otorgando gran libertad al desarrollador de escenarios. En este contexto, considero que USARSim v1.2 para UDK y Gazebo 2.2 resultan más adecuados que Webots 7. USARSim v1.2 para UDK es probablemente el simulador robótico más versátil y con mayores capacidades que existe en este momento, permitiendo simulaciones de enorme realismo audiovisual y funcional. Gazebo 2.2 al igual que USARSim es gratuito y, aunque su

potencia y versatilidad es inferior, presenta compatibilidad total con ROS, lo que lo convierte en una herramienta de gran utilidad y muy a tener en cuenta. En resumen, en un marco de programación remota tal como RPN o cualquier otro, si no existen problemas de compatibilidad entre el simulador y el resto de software, como desarrollador de entornos de simulación considero USARSim v1.2 para UDK como el más interesante para la creación de simulaciones realistas.

Capítulo 9

CONCLUSIONES

Concluyo esta tesis resumiendo las conclusiones a las que he llegado durante los años de estudio de los simuladores para robótica educativa. También expongo en este capítulo las principales contribuciones que esta tesis ofrece a la comunidad científica, mencionando posibles líneas de investigación o futuros trabajos que podrían poner más luz sobre los temas tratados en este documento.

9.1. Resumen de conclusiones

Como conclusiones de esta investigación destaco las grandes diferencias que he encontrado entre los simuladores cuando se trata de construir mundos virtuales o introducir nuevas plataformas robóticas, decantándose la balanza claramente hacia los simuladores basados en motores de juegos. Tales simuladores, desde el punto de vista del docente desarrollador de entornos, permiten explotar el hardware actual y las tecnologías software más vanguardistas para obtener los mayores grados de realismo, facilitando la incorporación de los elementos clave que proporcionan experiencias didácticas y atractivas. Los contenidos que introducen fantasía o misterio, y en general todos aquellos elementos que las teorías del aprendizaje señalan como importantes para mejorar la motivación de los alumnos durante las simulaciones parecen estar fuera de las miras de los desarrolladores de simuladores robóticos. Tan solo la naturaleza de los simuladores basados en motores de juegos permite salvar esta situación.

En un marco donde la robótica educativa gana cada vez mayor importancia, resulta notable la falta de aprovechamiento tecnológico (tanto a nivel de hardware como de aplicaciones software) que en general muestran los simuladores de plataformas robóticas no basados en motores de juegos, los cuales dan la impresión de estar diseñados para fines más cercanos a la investigación que a la educación.

9.2. Contribuciones

Las principales contribuciones de esta tesis son:

- Guía para la comparación y selección de simuladores para robótica educativa desde un punto de vista centrado en maximizar la satisfacción de los estudiantes y su grado de aprendizaje.
- Implementación de tecnologías vanguardistas para la mejora de la fidelidad audiovisual y funcional de los simuladores robóticos: Diversos niveles de detalle, objetos y estructuras destructibles, entes animados, sistemas de partículas, objetos articulados, simulación de tejidos o ropa, sonidos que verifican el impacto entre objetos e inteligencia artificial.
- Aplicación a las simulaciones robóticas de las teorías sobre aprendizaje con entornos virtuales.
    - Creación de modelos y escenarios de alta fidelidad audiovisual y funcional.
    - Incorporación de elementos fantásticos en las simulaciones.
    - Incremento de la estimulación sensorial para maximizar el grado de realimentación e inmersión.
    - Planteamiento de retos plausibles con dificultad progresiva.
    - Evaluación de logros en tiempo real mediante sistemas de puntuación y mensajes de texto que indican los objetivos cumplidos y los retos pendientes.
    - Maximización del control del estudiante sobre el entorno virtual.

9.3. Publicaciones

El trabajo presentado en esta tesis ha generado algunas publicaciones en revistas y conferencias internacionales que expongo a continuación.

Revistas

- J. Alemany and E. Cervera, "Simulated, Real, or Virtual Robots? - Two Decades of Teaching Experiences," *International Journal of Robots, Education and Art*, vol. 2, no. 1, pp. 1-9, 2014.

- E. Cervera, P. Martinet, R. Marín, A. A. Moughlbay, A. P. del Pobil, J. Alemany, R. Esteller, G. Casañ, "The Robot Programming Network," *Journal of Intelligent & Robotic Systems*, pp. 1-19, 2015.

Conferencias

- J. Sales, R. Beltrán, P. Sanz, R. Marín, R. Wirz, G. León, J. Claver, and J. Alemany, "The UJI industrial robotics telelaboratory: Real-time vision and networking," in *Intelligent Robots and Systems, 2008 (IROS 2008), IEEE/RSJ International Conference on,* Nice, 2008, pp. 4136

- L. Nomdedeu, J. Sales, E. Cervera, J. Alemany, R. Sebastiá, J. Penders, and V. Gazi, "An experiment on squad navigation of human and robots," in *IEEE 10th International Conference on Control, Automation, Robotics and Vision, 2008*, ICARCV 2008, pp. 1212-1218.

- J. Alemany and E. Cervera, "Design of High Quality, Efficient Simulation Environments for USARSim," in *Proceedings of the IASTED International Conference on Robotics*, 2011, pp. 226-233.

- J. Alemany and E. Cervera, "Appealing Robots as a Means to Increase Enrollment Rates: a Case Study," in *Proc. 3rd Int. Conf. on Robotics in Education (RiE 2012)*, Prague, 2012, pp. 15-19.

- G. A. Casan, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet, "ROS-based online robot programming for remote education and training," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015, pp. 6101-6106.

Capítulos en libros

- L. Nomdedeu, J. Sales, E. Cervera, J. Alemany, R. Sebastiá, and K. McAllister, "Mobile robot simulators and their application to hazardous and challenging environments," in J. Penders, Y. Baudoin, E. Cervera, and R. Marín, editors, *Robotics for Risky Interventions and Surveillance of the Environment*. Publicacions de la Universitat Jaume I, 2008.

- L. Nomdedeu, J. Sales, E. Cervera, J. Alemany, R. Sebastiá, J. Penders, and V. Gazi, "An Experiment on Squad Navigation of Human and Robots," in J. Penders, Y. Baudoin, E. Cervera, and R. Marín, editors, *Robotics for Risky Interventions and Surveillance of the Environment*. Publicacions de la Universitat Jaume I, 2008.

9.4. Trabajo futuro

Con el fin de seguir avanzando en el campo de las simulaciones para robótica educativa considero interesantes futuras líneas de investigación acerca de las posibilidades del novedoso motor de juego UE4 (Unreal Engine 4), el simulador MORSE y los servidores web de simulación.

Epic Games ha lanzado el motor de juego Unreal Engine 4 y al igual que el anterior es gratuito para fines educativos. El nuevo motor da soporte las más vanguardistas técnicas para la obtención de experiencias más inmersivas y atrayentes, permitiendo el uso de dispositivos de realidad virtual. Pero quizá lo más interesante es que unrealscrip ha sido sustituido por C++ y se ha potenciado notablemente el lenguaje Kismet. Esto podría facilitar mucho el trabajo a los docentes desarrolladores simulaciones robóticas.

Aunque todavía se encuentra en una temprana fase de desarrollo, considero que MORSE es un simulador muy a tener en cuenta en futuros estudios. El hecho de estar construido sobre una base de software totalmente open source y la potencia del motor de juego de Blender son piezas clave para creación de simuladores para robótica educativa.

Las novedosas capacidades de WebGL para visualización 3D y Web Socket para intercambio bidireccional de información entre servidor y cliente incorporadas en html5 convierten los navegadores web en poderosas herramientas para la difusión de la educación en robótica. Los servidores web de simulación permiten a los usuarios ejecutar simulaciones desde cualquier lugar mediante cualquier tipo de dispositivo (ordenador, tablet o teléfono móvil) que pueda conectarse a Internet a través de un navegador.

REFERENCIAS

[1] J. Alemany and E. Cervera, "Simulated, Real, or Virtual Robots? - Two Decades of Teaching Experiences," *International Journal of Robots, Education and Art*, vol. 2, no. 1, pp. 1-9, 2014.

[2] V. Dagdidelis, M. Sartatzemi, and K. Kagani, "Teaching (with) robots in secondary schools: some new and not-so-new pedagogical problems," in *ICALT 2005. Fifth IEEE International Conference on Advanced Learning Technologies*, 2005, pp. 757-761.

[3] A. Gage and R. R. Murphy, "Principles and experiences in using legos to teach behavioral robotics," *Frontiers in Education*, vol. 2-F4E, pp. 23-28, 2003.

[4] R. Harlan, "Adding Robotics to the Undergraduate," in *Fifteenth Annual Eastern Small College Computing*, 1999, pp. 112-119.

[5] F. Klassner and S. D. Anderson, "Lego MindStorms: Not just for K-12 anymore," *IEEE Robotics & Automation Magazine*, vol. 10, no. 2, pp. 12-18, 2003.

[6] B. Lester, "Robots' Allure: Can It Remedy What Ails Computer Science?," *Science*, vol. 318, no. 5853, pp. 1086-1087, 2007.

[7] N. J. Nilsson, "Shakey The Robot," in *SRI International*, Menlo Park, 1984.

[8] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14-23, 1986.

[9] D. Kumar and L. Meeden, "A Robot Laboratory for Teaching Artificial Intelligence," *ACM SIGCSE Bulletin*, vol. 30, no. 1, pp. 341-344, 1998.

[10] E. Cervera and P. J. Sanz, "Experiences with Minirobot Platforms in Robotics and AI Laboratory," in *Proceedings of the First Workshop on Robotics Education and Training*, 2001, pp. 85-90.

[11] S. Parsons and E. Sklar, "Teaching AI Using Lego Mindstorms," in *AAAI Spring Symposium*, 2004.

[12] J. Piaget and B. Inhelder, *La psychologie de l'enfant*. París: Presses universitaires de France, 1966.

[13] S. Papert, *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books, 1980.

[14] S. Papert, *Constructionism: A New Opportunity for Science Education-A Proposal to the National Science Foundation*. Cambridge-Massachussets: MIT Media Laboratory, 1986.

[15] O. Miglino, H. H. Lund, and M. Cardaci, "Robotics as an educational tool," *Journal of Interactive Learning Research*, vol. 10, no. 1, pp. 25-47, 1999.

[16] E Wang, "Teaching freshmen design, creativity and programming with LEGOs and labview," in *Frontiers in Education Conference, vol. 3*, 2001, pp. 11-15.

[17] K. Walstrom, T. Schambach, K. Jones, and W. Crampton, "Why Are Students Not Majoring in Information Systems?," *Journal of Information Systems Education*, vol. 19, no. 1, pp. 43-54, 2008.

[18] B. Ilardi, "Graduate Enrollment in Science and Engineering Continued To Decline in 1998," *SRS Data Brief*, December 1999.

[19] L. Carter, "Why students with an apparent aptitude for computer science don't choose to major in computer science," *ACM SIGCSE Bulletin*, vol. 38, no. 1, March 2006.

[20] J. Bergin, R. Lister, B. B. Owens, and M. McNally, "The first programming course: ideas to end the enrollment decline," in *SIGCSE Conference on*

Innovation and Technology in Computer Science Education, Nueva York, 2006, pp. 301-302.

[21] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*, 1997, pp. 340-347.

[22] A. Jardón, P. Zafra, S. Martínez, and A. Giménez, "CEABOT: Nationalwide Little humanoid robots competition; rules, experiences and new challenges," , Venice, Italy, 2008, pp. Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots.

[23] J. Alemany and E. Cervera, "Appealing Robots as a Means to Increase Enrollment Rates: a Case Study," in *Robotics in Education*, Praga, 2012, pp. 15-19.

[24] J. D. Fletcher and S. Tobias, "Using computer games and simulations for instruction: A research review," in *Proceedings of the Society for Advanced Learning Technology Meeting*, 2006.

[25] P. Petridis, I. Dunwell, S. de Freitas, and D. Panzoli, "An Engine Selection Methodology for High Fidelity Serious Games," in *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2010 Second International Conference on*, Braga, 2010, pp. 27-34.

[26] R. Blunt, "Does game-based learning work? Results from three recent studies," in *Proceedings of the Interservice/Industry Training, Simulation, & Education Conference*, Orlando, Florida, 2007, pp. 945-955.

[27] M. Prensky, *Digital game-based learning*. Nueva York: McGraw-Hill, 2001.

[28] M. E. Bredemeier and C. E. Greenblatt, "The educational effectiveness of games: A synthesis of findings," *Simulation & Gaming*, vol. 12, no. 3, pp. 307-332, 1981.

[29] K. A. Wilson et al., "Relationships Between Game Attributes and Learning Outcomes: Review and Research Proposals," *Simulation & Gaming*, vol. 40, pp. 217-266, April 2009.

[30] F. Percival and H. I. Ellington, "The place of case studies in the simulation/gaming field," *Perspectives on Academic Gaming and Simulation*, no. 5, pp. 21-30, 1980.

[31] C. Crawford, *The art of computer game design*. Berkeley, California: Osborne/McGraw-Hill, 1984.

[32] J. M. Randel, B. A. Morris, C. D. Wetzel, and B. V. Whitehill, "The effectiveness of games foreducational purposes: A review of recent research," *Simulation & gaming*, vol. 23, no. 3, pp. 261-276, 1992.

[33] R. T. Hays, "The effectiveness of instructional games: A literature review and discussion," Naval Air Warfare Center Training Systems Division, Orlando, FL, Technical Report 2005-004, 2005.

[34] R. Garris and R. Ahlers, "A game-based training model: Development, application, and evaluation," in *Interservice/Industry Training, Simulation & Education Conference*, Orlando, FL, 2001.

[35] R. Garris, R. Ahlers, and J. E. Driskell, "Games, motivation, and learning: A research and practice model," *Simulation & Gaming*, vol. 33, no. 4, pp. 441-467, 2002.

[36] T. W. Malone and M. R. Lepper, "Making learning fun: A taxonomy of intrinsic motivations for learning," *Aptitude, learning, and instruction*, vol. 3, pp. 223-253, 1987.

[37] S. de Freitas, F. Liarokapis, G. Rebolledo-Mendez, G. Magoulas, and A. Poulovassilis, "Developing an evaluation methodology for immersive learning experiences in a virtual world," in *Proceedings of 1st IEEE conference on Games and Virtual Worlds for Serious Applications*, 2009, pp. 43-50.

[38] T. Nakamoto et al., "Cooking up an interactive olfactory game display," *IEEE Computer Graphics and Applications*, vol. 28, no. 1, pp. 75-78, 2008.

[39] J. E. Driskell and D. J. Dwyer, "Microcomputer Videogame Based Training," *Educational Technology*, vol. 24, no. 2, pp. 11-17, Febrero 1984.

[40] R. Lewis, S. Stoney, and M. Wild, "Motivation and interface design: maximising learning opportunities," *Journal of Computer Assisted Learning*, vol. 14, no. 1, pp. 40-50, 1998.

[41] D. I. Cordoba and M. R. Lepper, "Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice," *Journal of Educational Psychology*, vol. 88, no. 4, pp. 715-730, Diciembre 1996.

[42] J. Alemany and E. Cervera, "Design of High Quality, Efficient Simulation Environments for USARSim," in *Proceedings of the IASTED International Conference on Robotics*, 2011, pp. 226-233.

[43] T. Mautone, V. Spiker, and D. Karp, "Using Serious Game Technology to Improve Aircrew Training," in *The Interservice/Industry Training, Simulation & Education Conference*, 2008.

[44] A. L. Alexander, T. Brunyé, J. Sidman, and S. A. Weil, "From gaming to training: A review of studies on fidelity, immersion, presence, and buy-in and their effects on transfer in pc-based simulations and games," in *Interservice/industry training, simulation, and education conference (I/ITSEC)*, 2005.

[45] C. Perez-Vidal, L. Gracia, N. Garcia, E. Cervera, and J. M. Sabater, "The on-line teaching of robot visual servoing," *International Journal of Electrical Engineering Education*, vol. 48, no. 2, pp. 202-216, April 2011.

[46] J. Trevelyan, "Lessons learned from 10 years experience with remote laboratories," in *International Conference on Engineering Education and Research (iCEER)* , 2004, pp. 687-697.

[47] M. Quigley et al., "ROS: an open-source Robot Operating System," *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[48] S. Osentoski et al., "Robots as web services: Reproducible experimentation and application development using rosjs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 6078-6083.

[49] E. Cervera, P. Martinet, R. Marín, A. A. Moughlbay, A. P. del Pobil, J. Alemany, R. Esteller, and G. Casañ, "The Robot Programming Network," *Journal of Intelligent & Robotic Systems*, pp. 1-19, 2015.

[50] C. Skinner and S. Ramchurn, "The robocup rescue simulation platform," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 1647-1648.

[51] S. Balakirsky, R. Madhavan, and C. Scrapper, "NIST/IEEE Virtual Manufacturing Automation Competition: from earliest beginnings to future direction," in *Workshop on Performance Metrics for Intelligent Systems*, New York, 2008, pp. 214-219.

[52] B. P. Gerkey et al., "Most valuable player: a robot device server for distributed control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2001, pp. 1226-1231.

[53] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, an Open-Source Multi- Robot Simulator," in *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149-2154.

[54] L. Hughes and N. Bredeche, "Simbad: An Autonomous Robot Simulation Package for Education and Research," in *From Animals to Animats 9*, vol. 4095, Springer LNCS, 2006, pp. 831-842.

[55] O. Michel, "Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39-42, 2004.

[56] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *Proceedings of the 2003 International Conference on Advanced Robotics (ICAR)*, Coimbra, 2003, pp. 317-323.

[57] Pioneer Mobile Robots, Operation Manual, 2nd Edition, ActivMEDIA, 1998.

[58] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A Survey of Commercial & Open Source Unmanned Vehicle Simulators," in *IEEE International Conference on Robotics and Automation*, Roma, Abril 2007, pp. 852-857.

[59] A. Harris and J. M. Conrad, "Survey of Popular Robotics Simulators, Frameworks and Toolkits," in *Southeastcon, 2011 Proceedings of IEEE*, Nashville, 2011, pp. 243-249.

[60] R. Hess, *The essential Blender: guide to 3D creation with the open source suite Blender*.: No Starch Press, 2007.

[61] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 4397 - 4404.

[62] G. Echeverria et al., "Simulating Complex Robotic Scenarios with MORSE," in *3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, Tsukuba, Japón, 2012, pp. 197-208.

[63] G. Echeverría, "The MORSE Robotics simulation platform," in *Blender Conference*, Amsterdam, 2010.

[64] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, pp. 189-208, 2008.

[65] R. T. Vaughan and A. Howard, "On device abstractions fo rportable , reusable robot code," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Las Vegas, 2003, pp. 2421-2427.

[66] R. Smith, "Open Dynamics Engine v0.5 User Guide," Technical report 2006.

[67] J. Craighead, J. Burke, and R. Murphy, "Using the Unity Game Engine to Develop SARGE: A Case Study," in *Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008)*, 2008.

[68] J. Craighead, R. Gutierrez, J. Burke, and R. Murphy, "Validating the Search and Rescue Game Environment as a robot simulator by performing a simulated anomaly detection task," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, 2008, pp. 2289-2295.

[69] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34 79, 2008.

[70] R. Diankov, K. Sato, H. Yaguchi, K. Okada, and M. Inaba, "Manipulation Planning for the JSK Kitchen Assistant Robot Using OpenRAVE," in *The 29th Annual Conference on Robotics Society of Japan*, AC2Q2–2, 2011.

[71] T. Laue and T. Röfer, "Simrobot - development and applications," in *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2008.

[72] T. Laue, K. Spiess, and T. Röfer, "SimRobot - A General Physical Robot Simulator and its Application in RoboCup," in *Proccedings of RoboCup Symposium. Universität Bremen (2005)*.

[73] M. Lewis, K. Sycara, and I. Nourbakhsh, "Developing a Testbed for Studying Human-Robot Interaction in Urban Search and Rescue," in *Procedings of the 10th International Conference on Human Computer Interaction (HCII'03)*, 2003.

[74] J. Wang, M. Lewis, and J. Gennari, "A Game Engine Based Simulation of the NIST Urban Search & Rescue Arenas," in *Proceedings of the 2003 Winter Simulation Conference*, 2003.

[75] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1400-1405.

[76] D. Derakhshani, *Introducing Autodesk Maya 2013*.: John Wiley & Sons, 2012.

[77] C. Maraffi, *Maya character creation: modeling and animation controls*.: New Riders, 2003.

[78] M. T. Peterson and S. Illustrator-Burke, *3D Studio Max 3 Fundamentals*.: New Riders Publishing, 1999.

[79] D. Derakhshani and R. L. Derakhshani, *Autodesk 3ds Max 2013 Essentials*.: John Wiley & Sons, 2012.

[80] J. Jackson, "Microsoft robotics studio: A technical introduction," *IEEE Robotics & Automation Magazine*, vol. 14, no. 4, pp. 82-87, 2007.

[81] K. Johns and T. Taylor, *Professional Microsoft Robotics Developer Studio*. Hoboken, New Jersey: Wrox Press, 2008.

[82] S. Cousins, "Exponential Growth of ROS," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 19-20, 2011.

[83] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot plattform," *International Journal Of Advanced Robotic Systems*, vol. 3, no. 1, pp. 43-48, 2006.

[84] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W. K. Yoon, "RT-middleware: distributed component middleware for RT (robot technology)," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005, pp. 3933-3938.

[85] J. L. B. Claraco, *Development of scientific applications with the mobile robot programming toolkit*. Málaga, Spain: The MRPT reference book, 2008,

Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain (2008).

[86] H. Bruyninckx, "OROCOS: design and implementation of a robot control software framework," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002, pp. 1-9.

[87] H. Bruyninckx, "Open robot control software: the OROCOS project," *In Robotics and Automation. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3, pp. 2523-2528, 2001.

[88] P. Petridis et al., "Game Engines Selection Framework for High-Fidelity Serious Applications," *International Journal of Interactive Worlds*, 2012.

[89] D. H. Eberly, *3D game engine design: a practical approach to real-time computer graphics*.: CRC Press, 2006.

[90] G. D. Park, R. W. Allen, T. J. Rosenthal, and D. Fiorentino, "Training effectiveness: How does driving simulator fidelity influence driver performance?," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 49, no. 25, pp. 2201-2205, Septiembre 2005.

[91] B. G. Witmer and M. J. Singer, "Measuring presence in virtual environments: A presence questionnaire," *Presence: Teleoperators and virtual environments*, vol. 7, no. 3, pp. 225-240, 1998.

[92] S. Tracy and P. Reindell, *CryENGINE 3 Game Development: Beginner's Guide*.: Packt Publishing Ltd, 2012.

[93] R. Graham, H. McCabe, and S. Sheridan, "Pathfinding in computer games," *ITB Journal*, no. 8, pp. 57-81, 2003.

[94] S. Newton, R. Lowe, R. Kember, R. Wang, and S. Davey, "The Situation Engine: A hyper-immersive platform for construction workspace simulation and learning," in *Proceedings of the 13th International Conference on Construction Applications of Virtual Reality, CONVR 2013*, 2013, pp. 262-271.

[95] S. Marks, J. Windsor, and B. Wünsche, "Evaluation of game engines for simulated surgical training," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, 2007, pp. 273-280.

[96] W. Goldstone, *Unity 3. x game development essentials*.: Packt Publishing Ltd, 2011.

[97] S. Coorg and S. Teller, "Real-time occlusion culling for models with large occluders," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, 1997, pp. 83-ff.

[98] E. Borodzicz and K. Van Haperen, "Individual and group learning in crisis simulations," *Journal of contingencies and crisis management*, vol. 10, no. 3, pp. 139-147, 2002.

[99] M. P. J. Habgood, S. E. Ainsworth, and S. Benford, "Endogenous fantasy and learning in digital games," *Simulation & Gaming*, vol. 36, no. 4, pp. 483-498, 2005.

[100] W. F. Engel, J. Hoxley, R. Kornmann, N. Suni, and J. Zink, *Programming vertex, geometry, and pixel shaders*.: Cengage Delmar Learning, 2008.

[101] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-time rendering*.: CRC Press, 2008.

[102] P. V. Sander and J. L. Mitchell, "Out-of-core rendering of large meshes with progressive buffers," in *Proceedings of the conference on SIGGRAPH*, 2006, pp. 1-18.

[103] S. Kelby, *The Adobe Photoshop CS3 book for digital photographers*.: New Riders, 2007.

[104] R. T. Apteker, J. A. Fisher, V. S. Kisimov, and H. Neishlos, "Video acceptability and frame rate," *IEEE multimedia*, vol. 2, no. 3, pp. 32-40, 1995.

[105] K. T. Claypool and M. Claypool, "On frame rate and player performance in first person shooter games," *Multimedia systems*, vol. 13, no. 1, pp. 3-17, 2007.

[106] E. Cervera, J. C. García, and P. J. Sanz, "Toward the Robot Butler: The HUMABOT Challenge [Competitions]," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 8-17, 2015.

[107] L. Nomdedeu et al., "An experiment on squad navigation of human and robots," in *IEEE 10th International Conference on Control, Automation, Robotics and Vision, 2008*, ICARCV 2008, pp. 1212-1218.

[108] J. Saez-Pons, L. Alboul, J. Penders, and L. Nomdedeu, "Multi-robot team formation control in the GUARDIANS project," *Industrial Robot: An International Journal*, vol. 37, no. 4, pp. 372-383, 2010.

[109] E. A. Skinner and M. J. Belmont, "Motivation in the classroom: Reciprocal effects of teacher behavior and student engagement across the school year," *Journal of Educational Psychology*, vol. 85, no. 4, pp. 571-581, 1993.

[110] D. A. Kolb, *Experiential learning: experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[111] D.C. Thatcher, "Promoting learning through games and simulations," *Simulation & Gaming*, vol. 21, no. 3, pp. 262-273, 1990.

[112] K. Kraiger, J.K. Ford, and E. Salas, "Application of cognitive, skill-based, and affective theories of learning outcomes to new methods of training evaluation," *Journal of applied psychology*, vol. 78, no. 2, pp. 311-328, 1993.

[113] M. Asgari and D. Kaufman, "Relationships among computer games, fantasy and learning," in *Proccedings of the 2nd International Conference on Imagination and Education*, 2004.

[114] L. E. Parker and M. R. Lepper, "Effects of fantasy contexts on children's learning and motivation: making learning more fun," *Journal of Personality and Social Psychology*, vol. 62, no. 4, pp. 626-633, 1992.

[115] T. W. Malone, "What makes things fun to learn? Heuristics for designing instructional computer games," in *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, 1980, pp. 162-169.

[116] R. W. White, "Motivation reconsidered: The concept of competence," *Psychological Review*, vol. 66, no. 5, pp. 297-333, 1956.

[117] S. Chen and D. Michael, "Proof of learning: Assessment in Serious Games," *Gamasutra*, 2005.

[118] S. W. Crown, "Improving visualization skills of engineering graphics students using simple JavaScript web based games," *Journal of Engineering Education*, vol. 90, no. 3, pp. 347-355, 2001.

[119] J. Pange, "Teaching probabilities and statistics to preschool children," *Information Technology in Childhood Education Annual* , pp. 163-172, 2003.

[120] K. E. Ricci, E. Salas, and J. A. Cannon-Bowers, "Do computer-based games facilitate knowledge acquisition and retention," *Military Psychology*, vol. 8, no. 4, pp. 295-307, 1996.

[121] A. Noble, D. Best, C. Sidell, and J. Strang, "Is an arcade-style computer game an effective medium for providing drug education to schoolchildren?," *Education for Health*, vol. 13, no. 3, pp. 404-406, 2000.

[122] T. W. Malone, "Toward a theory of intrinsically motivating instruction," *Cognitive science*, vol. 5, no. 4, pp. 333-369, 1981.

[123] J. Belanich, D. E. Sibley, and K. L. Orvis, "Instructional characteristics and motivational features of a PC-based game," U.S. Army Research Institute for the Behavioral and Social Sciences, Alexandria, VA, ARI-RR-1822, 2004.

[124] R. Rosas et al., "Beyond Nintendo: design and assessment of educational video games for first and second grade students," *Computers & Education*, vol. 40, no. 1, pp. 71-94, 2003.

[125] M. D. Woodman, "Cognitive training transfer using a personal computer-based game: A close quarters battle case study," University of Central Florida, Florida, Orlando, Doctoral dissertation 2006.

[126] M. Westrom and A. Shaban, "Intrinsic motivation in microcomputer games," *Journal of Research on Computing in Education*, vol. 24, no. 4, pp. 433-445, 1992.

[127] D. R. Baum, S. L. Riedel, R. T. Hays, and A. Mirabella, "Training effectiveness as a function of training device fidelity," HONEYWELL SYSTEMS AND RESEARCH CENTER MINNEAPOLIS MN, 82SRC37, 1982.

[128] L. Davidovitch, A. Parush, and A. Shtub, "The Impact of Functional Fidelity in Simulator-based Learning of Project Management," *International Journal of Engineering Education*, vol. 25, no. 2, pp. 333-340, 2009.

[129] G. Robertson, M. Czerwinski, and M. Van Dantzich, "Immersion in Desktop Virtual Reality," in *Proceedings of the 10th annual ACM symposium on User interface software and technology*, 1997, pp. 11-19.

[130] M. Csikszentmihalyi and R. Kubey, "Television and the Rest of Life: A Systematic Comparison of Subjective Experience," *Public Opinion Quarterly*, vol. 45, no. 3, pp. 317-328, 1981.

[131] M. J. Kilgard, "A Practical and Robust Bump-mapping Technique for Today's GPUs," in *In Game Developers Conference*, 2000, pp. 117-126.

[132] D. S. Ebert, *Texturing & modeling: a procedural approach*.: Morgan Kaufmann, 2003.

[133] G. Guennebaud, L. Barthe, and M. Paulin, "Real-time Soft Shadow Mapping by Backprojection," in *Rendering Techniques*, 2006, pp. 227-234.

[134] F. C. Crow, "Shadow algorithms for computer graphics," *ACM SIGGRAPH Computer Graphics*, vol. 11, no. 2, pp. 242-248, Julio 1977.

[135] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen, "Realistic real-time rendering of landscapes using billboard clouds," *Computer Graphics Forum*, vol. 24, no. 3, pp. 507-516, Septiembre 2005.

[136] Z. Fan, X. Jin, J. Feng, and H. Sun, "Mesh morphing using polycube-based cross-parameterization," *Computer Animation and Virtual Worlds*, vol. 16, no. 3-4, pp. 499-508, 2005.

[137] B. J. van Basten and A. Egges, "Evaluating distance metrics for animation blending," in *ACM, Proceedings of the 4th International Conference on Foundations of Digital Games*, 199-206, p. 2009.

[138] J. A. Allen, R. T. Hays, and L. C. Buffardi, "Maintenance training simulator fidelity and individual differences in transfer of training," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 28, no. 5, pp. 497-509, 1986.

[139] S. A. Weil, T. Hussain, T. Brunyé, J. Sidman, and L. Spahr, "The use of massive multi-player gaming technology for military training: A preliminary evaluation," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting (Vol. 49, No. 12, pp. 1186-1190)*, 2005.

[140] S. A. Weil et al., "Assessing the potential of massive multi-player games to be tools for military training," in *Proceedings of the interservice/industry training, simulation, and education conference (I/ITSEC)*, 2005.

[141] T. Schlömer, B. Poppinga, N. Henze, and S. Boll, "Gesture recognition with a Wii controller," in *Proceedings of the 2nd international conference on Tangible and embedded interaction. ACM*, 2008.

[142] Z. Zhang, "Microsoft kinect sensor and its effect," *MultiMedia, IEEE*, vol. 19, no. 2, pp. 4-10, 2012.

[143] B. G. Witmer and M. J. Singer, "Measuring immersion in virtual environments," U. S. Army Research Institute for the Behavioral and Social Sciences, Alexandria, VA, ARI Technical Report 1014, 1994.

[144] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*.: Prentice Hall, 2010.

[145] David Gouaillier et al., "Mechatronic design of NAO humanoid," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 769-774.

[146] F. Pérez, B. E. Granger, and J. D. Hunter, "Python: an ecosystem for scientific computing," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 13-21, 2011.

[147] Sander van Noort and Arnoud Visser, "Validation of the dynamics of an humanoid robot in USARSim," in *ACM Workshop on Performance Metrics for Intelligent Systems*, 2012, pp. 190-197.

[148] J. P. Bordes, M. Maher, and M. Sechrest, "Nvidia apex: High definition physics with clothing and vegetation," in *Game Developers Conference*, 2009.

[149] M. Garland, "Multiresolution modeling: Survey & future opportunities," in *State of the Art Reports of EUROGRAPHICS '99*, 1999, pp. 111–131.

[150] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533-536, 1986.

[151] Stefan: Colbert, Chris Van Der Walt and Gael Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22-30, 2011.

[152] Tom Schaul et al., "PyBrain," *The Journal of Machine Learning Research*, vol. 11, pp. 743-746, 2010.

[153] Hugo Picado, Marcos Gestal, Nuno Lau, Luis P. Reis, and Ana M. Tome, "Automatic generation of biped walk behavior using genetic algorithms," in

Bio-Inspired Systems: Computational and Ambient Intelligence.: Springer, 2009, pp. 805-812.

[154] R. R. Murphy, "“Competing” for a robotics education," *IEEE Robotics & Automation Magazine*, vol. 8, no. 2, pp. 44-55, 2001.

[155] M. J. Pavelich and W. S. Moore, "Measuring maturing rates of engineering students using the Perry model," in *Frontiers in Education Conference*, 1993, pp. 451-455.

[156] S. van Noort and A. Visser, "Extending virtual robots towards robocup soccer simulation and@ home," in *RoboCup 2012: Robot Soccer World Cup XVI*, 2013, pp. 332-343.

[157] H. Kitano et al., "Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research," in *Systems, Man, and Cybernetics. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, vol. 6, 1999, pp. 739-743.

[158] Nathan Labhart, Beatrice Hasler, Andy Zbinden, and Andreas Schmeil, "The ShanghAI Lectures: A case study in global education," *Journal of Universal Computer Science*, vol. 18, no. 18, pp. 2542-2555, 2012.

[159] S. Djenic, R. Krneta, and J. Mitic, "Blended learning of programming in the internet age," *IEEE Transactions on Education*, vol. 54, no. 2, pp. 247-254, Mayo 2011.

[160] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O.C. Jenkins, "PR2 Remote Lab: An environment for remote development and experimentation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3200-3205.

[161] R. Marín, P.J. Sanz, and A.P. Del Pobil, "The UJI online robot: An education and training experience," *Autonomous Robots*, vol. 15, no. 3, pp. 283-297, 2003.

[162] V. Djalic et al., "Remote laboratory for robotics and automation as a tool for remote access to learning content," in *15th International Conference on Interactive Collaborative Learning (ICL)*, 2012, pp. 1-3.

[163] L. Furler, A.S. Malik, F. Meriaudeau, and V. Nagrath, "An Auto-Operated Telepresence System for the Nao Humanoid Robot," in *International Conference on Communication Systems and Network Technologies (CSNT)*, 2013, pp. 262-267.

[164] C.A. Jara, F.A. Candelas, S.T. Puente, and F. Torres, "Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory," *Computers & Education*, vol. 57, no. 4, pp. 2451-2461, 2011.

[165] M. Kulich et al., "SyRoTek—distance teaching of mobile robotics," *IEEE Transactions on Education*, vol. 56, no. 1, pp. 18-23, 2013.

[166] P. Orduña, L. Rodríguez-Gil, D. López-de-Ipiña, and J. García-Zubia, "Sharing the remote laboratories among different institutions: a practical case," in *9th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2012, pp. 1-4.

[167] S. Osentoski et al., "Remote robotic laboratories for learning from demonstration," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 449-461, 2012.

[168] I. Santana, M. Ferre, E. Izaguirre, R. Aracil, and L. Hernández, "Remote laboratories for education and research purposes in automatic control systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 547-556, 2013.

[169] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris, "Robot Web Tools [ROS Topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 20 - 23, Diciembre 2012.

[170] D. Pritchard and T. Vasiga, "CS circles: an in-browser python course for beginners," in *44th ACM technical symposium on Computer science education*, 2013, pp. 591-596.

[171] M. O. C. Lee and A. Thompson, "Guided instruction in LOGO programming and the development of cognitive monitoring strategies among college students," *Journal of Educational Computing Research*, vol. 16, no. 2, pp. 125-144, 1997.

[172] M. Resnick et al., "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60-67, 2009.

[173] A. Martínez and E. Fernández, *Learning ROS for robotics programming*.: Packt Publishing Ltd, 2013.

[174] C. Crick, G. Jay, S. Osentoski, and O.C. Jenkins, "ROS and rosbridge: Roboticists out of the loop," in *Seventh annual ACM/IEEE international conference on Human-Robot Interaction*, 2012, pp. 493-494.

[175] G. A. Casan, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet, "ROS-based online robot programming for remote education and training," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015, pp. 6101-6106.

[176] M. Dougiamas and P. Taylor, "Moodle: Using learning communities to create an open source course management system," *World conference on educational multimedia, hypermedia and telecommunications*, vol. 2003, no. 1, pp. 171-178, 2003.

[177] I. Ha, Y. Tamura, H. Asama, J. Han, and D. W. Hong, "Development of open humanoid platform DARwIn-OP," in *Proceedings of SICE Annual Conference (SICE), IEEE*, 2011, pp. 2178-2181.