Documento Final

Entorno de Simulación Robótico

Anthony Figueroa

pgsimrob@fing.edu.uy

Tutor

Gonzalo Tejera

Cotutores

Gustavo Armagno, Facundo Benavides, Serrana Casella

6 de octubre de 2008

Instituto de Computación

Facultad de Ingeniería - Universidad de la República

Montevideo - Uruguay

Resumen

En los últimos años han tenido una creciente popularidad las competencias robóticas como una manera de impulsar e incentivar la investigación científica en esta área. En este escenario de desarrollo y prueba, el desarrollo de estrategias o comportamientos de control de robots es la piedra angular de estas competencias, y gran parte de la investigación se concentra en la necesidad de desarrollar comportamientos más inteligentes.

Distintas categorías de fútbol o sumo de robots son ejemplos de estas competencias, y en este contexto se debe hacer frente a problemas muy variados. Entre los problemas que se presentan están la construcción física de los robots, aspectos de comunicación y comportamiento. Para concentrar los esfuerzos en los aspectos de comportamiento, priorizando el estudio de problemas teóricos y conceptuales, es muy común la utilización de simuladores robóticos.

La utilización de simuladores juega un rol fundamental en la aplicación de técnicas de aprendizaje automático. La idea fundamental es que mediante algún mecanismo, los robots logren mejorar su comportamiento ante un determinado problema ha medida que transcurre el tiempo. Las características propias de los simuladores, como la posibilidad de repetir situaciones, o de ejecutarse por tiempo indefinido, crean el entorno ideal para la aplicación de técnicas de este tipo.

Desde hace algunos años el grupo MINA ha participado en diversas competencias robóticas, tanto en entornos reales como virtuales. De esta experiencia se planteó la necesidad de contar con un simulador con prestaciones útiles al momento de desarrollar comportamientos robóticos. Sin embargo no se encontró un simulador que reuniera satisfactoriamente una cantidad aceptable de prestaciones. Este proyecto plantea satisfacer esta necesidad mediante la creación de un simulador, cuyo propósito en un principio es la utilización del mismo como herramienta de base para desarrollar comportamientos robóticos más inteligentes.

Este documento describe las principales necesidades que se buscaron satisfacer, la investigación realizada en el área, y se explican las principales características del sistema desarrollado.

Índice

1. Introducción 7
   1.1. Generalidades 7
   1.2. Antecedentes 8
   1.3. Objetivos del proyecto 9
   1.4. Organización del documento 10

2. Estado del arte 13
   2.1. Introducción 13
   2.2. Motores Físicos 14
      2.2.1. Motores físicos analizados 18
   2.3. Casos de estudio 21
      2.3.1. GSim 21
      2.3.2. 3D Robot Soccer Simulator 23
      2.3.3. Robocup Soccer Simulator [SROBC] 23
      2.3.4. MiS20 [MIS20] 24
      2.3.5. Simulator Bob 24
      2.3.6. Webots 25
   2.4. Construcción de un prototipo 25
      2.4.1. Motivación 25
      2.4.2. Alcance 26
      2.4.3. Configuración del entorno de juego 26
      2.4.4. Configuración de los robots 26
      2.4.5. Visualizar el estado de la simulación 26
      2.4.6. Comunicación asincrónica con procesos 26
      2.4.7. Configuración del paso de tiempo 26
      2.4.8. Resultados 26

3. Descripción de la solución 31
   3.1. Aspectos Generales 31
   3.2. Aspectos de modelado 32
      3.2.1. Entidades 32
      3.2.2. Comandos 35
   3.3. Arquitectura 36
      3.3.1. Procesos y protocolos de comunicación 41
   3.4. Aspectos de Sincronización 41
      3.4.1. Modos de conexión de agentes 42
      3.4.2. Modos de ejecución de la simulación 42
   3.5. Funcionalidades 43

4. Decisiones de diseño 47
   4.1. Modelo físico 47
   4.2. Mensajes 48
   4.3. Colisiones 49
   4.4. Interfaz gráfica 51
   4.5. Extensibilidad 53
      4.5.1. Modelo físico 53
      4.5.2. Entidades 53
      4.5.3. Comandos 53
      4.5.4. Tipos de agentes y filtros 53

4.6. Performance 54
4.7. Ajuste del paso del tiempo 54
4.7.1. Ajuste automático 55

5. Experimentos y resultados 57
5.1. Verificación 57
5.2. Modelado de un caso tipo 58
5.2.1. Datos reales 59
5.2.2. Datos experimentales 59
5.2.3. Resultados 60

6. Conclusiones y trabajos futuros 63
6.1. Conclusiones 63
6.2. Trabajos a futuro 64
6.2.1. Interfaz física 64
6.2.2. Entidades 65
6.2.3. Colisiones 65
6.2.4. Comandos 65
6.2.5. Optimización 65
6.2.6. Armar entorno de simulación 66
6.2.7. Agentes Externos 66
6.2.8. Interfaz gráfica 66

Índice de figuras

1. Cuerpos rígidos y sus ejes ([Smi07]). 14
2. Empalmes ([Smi07]). 15
3. Empalme *ball and socket* ([Smi07]). 16
4. Empalme *hinge* ([Smi07]). 16
5. Empalme *slider* ([Smi07]). 17
6. Empalme *universal* ([Smi07]). 17
7. Empalme *hinge-2* ([Smi07]). 18
8. Hilos de ejecución de GSim y flujo de información entre los mismos ([ACGS03]). 22
9. Prototipo. Simulación utilizando drawstuff 28
10. Prototipo. Simulación utilizando Irrlicht 28
11. Visión MAS del sistema 32
12. Cuerpos agrupados en espacios 33
13. Cuerpos agrupados en espacios, con colisión 34
14. Colisión entre cuerpos, ampliación 35
15. Arquitectura del sistema 37
16. Diagrama de actividad de la ejecución de comandos 40
17. Mensajes 49
18. Simulación en interfaz gráfica 52
19. SumBot 61
20. SumBot, parte inferior 61
21. SimRob Commander 81

Índice de cuadros

1. Pruebas no funcionales, comparación de resultados 58
2. Modelado de un caso real. Recorrida de 0,50 metros 59
3. Modelado de un caso real. Giros 60
4. Modelado de un caso real. Mapeo de velocidades 61
5. Modelado de un caso real. Resultado recorrida de 0,50 metros 62
6. Modelado de un caso real. Resultado giros 62

1. Introducción

1.1. Generalidades

En el contexto de este proyecto se trabajó con el objetivo de construir un entorno de simulación robótico que se pudiera utilizar como herramienta de base sobre la cuál desarrolladores tengan la posibilidad de construir, probar y evolucionar sus creaciones robóticas, tanto en lo que se refiere a su construcción física como a su comportamiento, o sea, la inteligencia artificial. Para entender este objetivo surgen una serie de palabras y conceptos importantes que deben ser mirados con mayor detenimiento para comprender en profundidad las necesidades que busca satisfacer este proyecto.

Uno de los conceptos centrales y más amplios es el concepto de inteligencia artificial. Por mucho tiempo el hombre ha buscado entender sus procesos mentales, su inteligencia, analizando como es posible que un simple puñado de materia pueda percibir, entender, predecir y manipular una realidad mucho más compleja que ella misma. Esta búsqueda ingresa en el campo de la psicología cognitiva y ésta ha jugado un rol importante en la concepción de la ciencia que estudia la inteligencia artificial o IA. El campo de esta disciplina va más allá y no solo intenta comprender, sino también construir entidades inteligentes. Desde un punto de vista ideal, una entidad inteligente es aquella que comprende la mejor acción posible ante una situación dada.

La principal aplicación de esta ciencia es la creación de máquinas para la automatización de tareas que requieran un comportamiento inteligente. Algunos ejemplos se encuentran en el área de control de sistemas, planificación automática, la habilidad de responder a diagnósticos y a consultas de los consumidores, reconocimiento de escritura, reconocimiento del habla y reconocimiento de patrones. Los sistemas de IA actualmente son parte de la rutina en campos como economía, medicina, ingeniería y la milicia, y se ha usado en gran variedad de aplicaciones de software, juegos de estrategia como ajedrez de computador y otros videojuegos[WIKIIA].

Un robot se podría definir como un agente artificial, ya sea mecánico o virtual, que cuenta con algún grado de inteligencia. La robótica es la ciencia que se dedica al diseño, construcción y operación de dichos agentes.

En los últimos años han tenido un fuerte impulso las competencias entre robots para estimular a nivel académico o personal esta disciplina. En particular, el fútbol de robots ha conquistado muchos adeptos y ha tenido creciente importancia en lo que se refiere a la investigación científica en el área. Para construir un equipo de fútbol de robots competitivo se debe hacer frente a problemas muy diversos y de muy diferente índole. Las características propias de la competencia implican trabajar en un entorno altamente dinámico, donde las decisiones tomadas por los robots en el contexto de la competencia son claves. Por citar algunos ejemplos, se deben resolver problemas asociados a la propia construcción de los robots, resolver de manera adecuada los algoritmos de cooperación entre agentes, toma de decisiones, planificación de trayectorias, procesamiento de imágenes, procesamiento de información en tiempo real, entre otros.

Se han formado organizaciones en diversas partes del mundo para llevar a cabo estas competencias, organizándolas en varias categorías con diferentes características. Sin embargo, no todas las competencias de robots se llevan a cabo con robots reales, algunas de ellas son llevadas a cabo en un entorno virtual, o sea, en un simulador. Las organizaciones más reconocidas e importantes son la FIRA[1][FIRA] y la Robocup[ROBCU]. Ambas organizaciones presentan variadas categorías para distintos tipos de robots y en entornos reales y simulados.

Los entornos reales y los entornos simulados presentan muchas diferencias, pero los puntos

[1] Federation of International Robot-soccer Association

en común que presentan hacen que la utilización de simuladores sea una opción atractiva. En primer lugar la utilización de simuladores presenta la gran ventaja de mitigar considerablemente el costo económico que implica la construcción física de los robots. Además, generalmente para cambiar las características físicas del robot en un simulador simplemente se debe cambiar algunas líneas de código o parámetros de configuración, en contraste, en un modelo real generalmente hay que cambiar componentes o volver a ensamblar los mismos, lo cual puede resultar costoso. Otros problemas típicos del entorno real son el agotamiento de baterías, problemas de comunicación, problemas en los componentes físicos, entre otros. Por estas razones, la utilización de simuladores es vital para concentrar mayores esfuerzos en los problemas que surgen en la programación del comportamiento de los robots, en el cual se pueden priorizar el estudio de problemas teóricos y conceptuales, y no necesariamente problemas que emergen en el entorno real.

Los simuladores pueden ser ejecutados por tiempo indefinido, de modo que resultan especialmente útiles para desarrollar comportamientos que evolucionen con el tiempo, aplicando técnicas de aprendizaje automático. Esta es una idea que ha funcionado muy bien en el campo biológico y últimamente ha surgido un renovado interés en el campo de la robótica. El concepto de robots que automática y progresivamente optimicen su comportamiento en lo que se refiere a la resolución de un determinado problema es una idea interesante. De esta forma no se debería programar a los robots pensando en la manera o forma de resolver un problema, sino que es necesario hacerlo de manera tal que mediante algún mecanismo el robot logre comprender si sus acciones están llevando la realidad a un estado deseable, y en caso de no ser así, modificar dicho comportamiento de modo tal de mejorarlo a medida que transcurre el tiempo. Además, para la aplicación de estas técnicas, resulta conveniente tener la posibilidad de manipular el paso del tiempo, de forma tal que sea posible avanzar el paso de tiempo simulado a una velocidad mayor que el paso de tiempo real. Esto conduce a una reducción de los tiempos requeridos para optimizar el comportamiento de un robot en la etapa de aprendizaje, por lo cual resulta imprescindible contar con simuladores para lograr este objetivo.

Algunas desventajas en la utilización de simuladores deben ser tenidas en cuenta. Por lo general resulta difícil modelar de forma realista un agente simulado a partir de un modelo de referencia real, especialmente si dicho modelo de referencia es estructuralmente complejo. Además el modelo físico creado es una simplificación del modelo real, y en la creación del modelo es una decisión del diseñador del mismo definir que aspectos de la realidad son relevantes y cuales no, o sea, determinar el nivel de detalle del modelo. Por otra parte los errores introducidos por el modelo físico de la simulación, aunque éste esté basado completamente en leyes físicas, deben ser tenidos en cuenta.

1.2. Antecedentes

El Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay ha promovido diversos proyectos en el área de la robótica e inteligencia artificial en los últimos años.

En el año 2003 se realizaron tres proyectos que abarcaban aspectos diversos de la robótica. En uno de ellos se trabajó con el objetivo de construir un sistema de visión para un equipo de fútbol de robots, enfrentando el problema de procesamiento y reconocimiento de patrones en imágenes en tiempo real. En otro de los proyectos de aquel año, se trabajó con el objetivo de construir un robot a bajo costo que fuera capaz de moverse en forma autónoma, recibiendo órdenes desde un PC. Finalmente, en el proyecto restante se construyó un equipo de fútbol de robots denominado F.R.U.T.O., que se enfocó en el comportamiento del robot trabajando en un ambiente simulado. En el transcurso del desarrollo de este proyecto, se creyó conveniente la creación de un simulador de robótica, denominado GSim, con el objetivo de obtener mejores

resultados en el desarrollo del equipo.

En el año 2004 se llevó a cabo un proyecto en el cual se construyó un prototipo de robot bípedo, capaz de caminar cuasiestáticamente.

Luego, en el año 2005 se presentaron dos proyectos que tenían como objetivo seguir trabajando en el área del comportamiento de los robots y en sistemas M.A.S.[2], donde se realizó la construcción de equipos de fútbol de robots para actuar en ligas simuladas. Ambos proyectos crearon equipos para la participación en ligas simuladas, uno para la federación FIRA y otro para la Robocup.

Un poco al margen de estos proyectos, desde el año 2004, cada año el grupo MINA[3] organiza el Campeonato Uruguayo de Sumo de Robots, en el cual se les brinda un robot real a los participantes y estos programan su estrategia para intentar ganar la competencia. Este grupo también ha participado desde el año 2003 del CAFR[4], alternando entre la categoría real y la simulada.

Con estos antecedentes, este proyecto se inició en el año 2007 con el propósito de crear un entorno de simulación robótica, cuyos objetivos se describen en la próxima sección.

1.3. Objetivos del proyecto

Existía la necesidad de contar con un entorno de simulación robótica genérico, el cual pudiera ser una herramienta de utilidad para la creación de equipos de robots para distintos tipos de competencias y categorías. Como mínimo, se debía contar con la posibilidad de construir simulaciones que resultaran herramientas útiles al desarrollar robots para competencias de sumo de robots y fútbol de robots, en diversas categorías. El simuladores debería proveer una serie de funcionalidades que facilitaran la tarea de desarrollo y depuración de estrategias. Algunas de ellas se detallan a continuación:

- Creación genérica de robots: Contar con la posibilidad de modelar realidades muy diversas. Esto es fundamental, ya que para que sea posible construir los distintos tipos de robots y entornos que se desean, esto debe hacerse de manera genérica y soportar diferentes configuraciones.

- Control de robots: Exponer alguna interfaz de comunicación mediante la cual se puede controlar los robots presentes en la simulación mediante estrategias implementadas por desarrolladores, a través por ejemplo, de motores conectados a ruedas. Esto es muy importante ya que es la manera como las estrategias implementadas por los desarrolladores pueden controlar los robots existentes en la simulación.

- Manipulación del paso del tiempo: Contar con la funcionalidad de manipular el paso del tiempo de manera flexible. Esto implica que sea posible pausar la simulación y luego continuar la misma, avanzar la simulación paso a paso o de forma ininterrumpida, y también tener mecanismos para acelerar el paso del tiempo. Esta es una funcionalidad importante, tanto para depuración de estrategias como para brindar un entorno favorable para el desarrollo de estrategias que evolucionan en base a técnicas de aprendizaje automático.

- Manipulación de los robots: Contar con una serie de funcionalidades que brinden la posibilidad de manipular los robots presentes en la simulación, de modo tal que los

[2] Multi-Agent System

[3] El Grupo MINA (Network Management - Artificial Intelligence), es un grupo de investigación perteneciente al Instituto de Computación de la Facultad de Ingeniería (Universidad de la República), que realiza docencia, investigación y extensión en diversos temas relacionados con el Gerenciamiento de Redes e Inteligencia Artificial aplicada a la Robótica.

[4] Campeonato Argentino de Fútbol de Robots

mismos pudieran ser posicionados en algún lugar deseado, con el objetivo que entre otras cosas, sea relativamente sencillo implementar árbitros automáticos en competencias. Estos árbitros al detectar determinadas situaciones deben tener la posibilidad de posicionar los robots en alguna posición que este desee, por ejemplo, ordenar los robots luego de un gol en una competencia de fútbol de robots. También esta funcionalidad es de utilidad para entrenar el comportamiento de un equipo en una determinada posición, ya que se puede fácilmente ubicar cada robot en una posición deseada.

- Grabación y reproducción de estados de la simulación: El sistema debería brindar la funcionalidad para almacenar simulaciones en archivos y crear simulaciones a partir de datos previamente cargados. Esta funcionalidad puede resultar importante, por ejemplo, para probar varias veces el comportamiento de una estrategia en una situación dada.

- Extensibilidad: Contar con un sistema extensible en todos sus aspectos, en el cual desarrolladores pudieran fácilmente adaptar las funcionalidades a sus necesidades.

- Comunicación: Contar con un mecanismo de comunicación con las estrategias que controlan robots que sea independiente del lenguaje de programación utilizado en la implementación de las mismas.

- Sincronización: Se planteó el objetivo de contar con distintas formas de comunicación entre las estrategias y el simulador en lo que se refiere al sincronismo de dicha comunicación. En el caso sincrónico a cada ciclo de iteración luego de realizar la difusión con información del estado del mundo, el simulador debería esperar un determinado tiempo máximo por respuestas de las estrategias para avanzar el estado del mundo. En el caso asincrónico no debería esperarse este intervalo de tiempo, continuando con la ejecución del ciclo de iteración.

El principal objetivo de este proyecto no es solamente crear un simulador, sino crear una plataforma que posibilite a desarrolladores a crear robots más inteligentes. Para lograr esto se presentó la necesidad de construir un sistema extensible, configurable y adaptable a futuras necesidades que pudieran surgir en su utilización, cuyas funcionalidades facilitaran la tarea de modelado y depuración de comportamientos robóticos.

1.4. Organización del documento

El presente documento se ha dividido en capítulos, los mismos se describen a continuación.

Capítulo 1 - Introducción

Este capítulo es introductorio a todos los demás capítulos.

Capítulo 2 - Estado del Arte

Este capítulo tiene como objetivo presentar un resumen de la investigación realizada en la etapa de relevamiento de características de distintos simuladores disponibles en la actualidad y herramientas que eventualmente pueden ser útiles para construir este tipo de aplicaciones. Se muestran los casos más importantes y que de alguna manera ejercieron alguna influencia en la concepción de este sistema.

Capítulo 3 - Descripción de la Solución

Este capítulo tiene como objetivo describir la solución propuesta. Se explican los conceptos más importantes del sistema, su arquitectura general y se detallan las principales funcionalidades que provee el sistema.

Capítulo 4 - Decisiones de Diseño

Este capítulo tiene como objetivo describir algunas decisiones de diseño tomadas en el concepción de la solución. Se discuten alternativas y se brindan argumentos para justificar las decisiones tomadas.

Capítulo 5 - Experimentos y resultados

Este capítulo tiene como objetivo presentar los experimentos más importantes realizados y sus resultados. Esto incluye la creación de un prototipo en las etapas previas al diseño del sistema y las pruebas realizadas sobre el sistema ya construido. También se realiza el modelado de un caso real, mostrando el proceso para realizar este modelado con éxito.

Capítulo 6 - Conclusiones y trabajo a futuro

Este capítulo tiene como objetivo presentar las conclusiones luego de culminado el trabajo, y las posibles mejoras y trabajos a futuro que pueden realizarse en el sistema.

Referencias

Se presenta las referencias bibliográficas de este trabajo.

Glosario

Se definen los principales términos y conceptos mencionados en el documento.

2. Estado del arte

En este capítulo se presenta un resumen de las investigaciones realizadas en el marco del relevamiento del estado del arte de los simuladores robóticos. Aunque uno de los objetivos del sistema es poder realizar simulaciones de competencias de fútbol y sumo de robots, se analizan no solo simuladores con propósitos específicos, como una categoría específica de fútbol de robots, sino también simuladores robóticos de propósito general en los cuales se puede modelar una amplia gama de realidades. También se narran las conclusiones a las que se llegó y principales características de un prototipo que se implementó en el contexto de este proyecto, cuyo objetivo fue la mitigación de diversos riesgos técnicos.

Teniendo en cuenta que uno de los bloques constructivos fundamentales de dichos simuladores es el componente responsable de implementar el modelo físico sobre el cual se basan las simulaciones, resulta evidente que es necesario analizar alternativas y distintas librerías para la implementación de este importante componente. De este modo, se resolvió analizar distintas librerías físicas o motores físicos existentes en la actualidad, con el objetivo de analizar ventajas y desventajas de cada una, identificando la que más se ajusta a las necesidades y restricciones del sistema.

Sobre el final del capítulo se describen las distintas características de un prototipo desarrollado en la etapa de investigación de este proyecto, con el objetivo de mitigar riesgos técnicos y adquirir experiencia que pudiera ser útil en el desarrollo del sistema.

Un relato más detallado de las investigaciones realizadas se puede encontrar en el documento del Estado del Arte[AFE07]

2.1. Introducción

Uno de los puntos claves que se debió abordar en el desarrollo de este sistema es la construcción de un componente cuya función sea manejar el modelo físico de la simulación, tanto para la creación de los objetos que representan cuerpos físicos, como para su evolución a lo largo del tiempo.

El modelo físico sobre el cual se debe basar un simulador de fútbol o sumo de robots debe ser tan fiel a la realidad como sea posible. Una alternativa es utilizar un motor físico para implementar dicho modelo. Existen diversas características importantes de los motores físicos que es necesario analizar para contar con la información necesaria para realizar correctas decisiones en la elección de alguno en particular para su utilización en el marco de este proyecto.

La precisión en los cálculos matemáticos y la estabilidad de los mismos es muy importante, ya que se desea que el modelo físico sea tan fiel a la realidad como sea posible. Además, el proyecto plantea la necesidad de construir un simulador genérico y que brinde la posibilidad de simular realidades tan diversas como el sumo de robots o distintas categorías de fútbol de robots, por lo que es necesario modelar estas complejas realidades con el apoyo de un motor físico.

Por otra parte, otra característica importante es que exista la posibilidad de controlar de manera flexible el control del paso de simulación, para lograr entre otras cosas acelerar el paso del tiempo. Esta característica resulta importante en escenarios en los que se desee aplicar técnicas de aprendizaje automático para llevar a cabo la evolución de los comportamientos de los robots.

También es necesario analizar si el motor físico brinda algún tipo de funcionalidad para almacenar el estado de la simulación o incluso simulaciones enteras. Esto facilitaría la tarea de grabación de simulaciones, que podría tener como aplicación almacenar partidas enteras de fútbol o sumo de robots para luego visualizarlas con más detenimiento, o analizar el comportamiento de los robots repitiendo varias veces la misma situación.

En lo que sigue se describirán diferentes motores físicos evaluados y analizados. Luego se describirán algunos casos de estudio analizados y se detallarán las características importantes de los mismos.

2.2. Motores Físicos

La presencia de los siguientes elementos son denominadores comunes en todos los motores físicos.

Cuerpos Rígidos

Los cuerpos rígidos son la piedra angular de la simulación. Éstos se conectan mediante empalmes. Cada cuerpo tiene una primitiva de colisión asociada, que es la que determina su forma geométrica en el espacio. La forma de estas primitivas permanece incambiada a lo largo de la simulación, siendo esta forma determinante en la resolución de las colisiones entre los cuerpos.

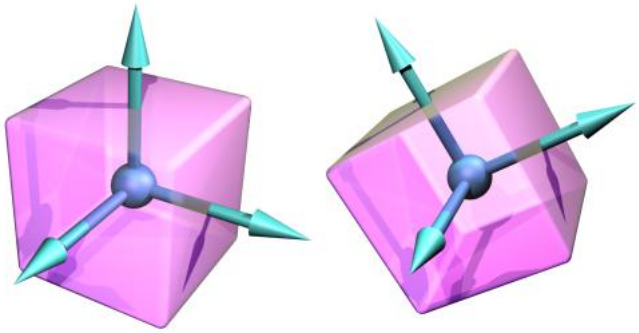

Figura 1: Cuerpos rígidos y sus ejes ([Smi07]).

Algunas primitivas de colisión aparecen repetidamente en diversos motores físicos. Algunos ejemplos de estas primitivas se detallan a continuación.

- Sphere: Representa una esfera.
- Box: Representa un prisma.
- Cylinder: Representa un cilindro común.
- Capped Cylinder: Representa un cilindro en cuyos extremos tiene medias esferas como terminación.
- Plane: Representa un plano geométrico.
- Cone: Representa un cono.
- Capsule: Representa una cápsula.

- Ray: Representa una linea infinitamente fina que empieza en determinada posición y se extiende a lo largo del eje Z.
- Triangular Mesh: Representa una colección arbitraria de triángulos. Se utiliza para representar cuerpos rígidos de cualquier forma.
- Convex Mesh: Similar al caso anterior, pero se representa utilizando una colección de vértices en lugar de una colección de triángulos. Además deben formar un cuerpo convexo.

Empalmes

Los empalmes se utilizan para conectar de alguna manera dos cuerpos rígidos en la simulación. Estos empalmes están sujetos a restricciones, que determina la manera en que están habilitados a realizar movimientos en la simulación. Los cuerpos a los cuales conecta este empalme pueden alcanzar determinadas posiciones y orientaciones uno respecto al otro, estas posiciones las determina el tipo de empalme y las restricciones sobre el mismo.

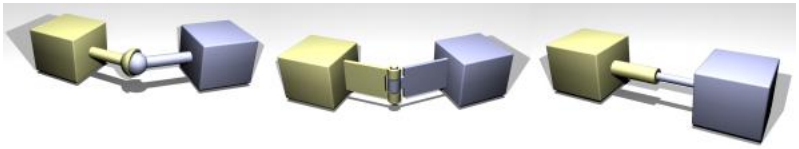

Figura 2: Empalmes ([Smi07]).

Según el tipo de restricciones sobre el movimiento de cuerpos que se desea modelar, se debe elegir el tipo de empalme que más se adapte al mismo y ajustarlo a lo que se requiera. Algunos ejemplos de empalmes que pueden ser encontrados en motores físicos se muestra a continuación.

- Ball and Socket: Se especifica un punto de anclaje entre ambos cuerpos y este empalme actúa de tal manera que mantiene la posición relativa de dicho anclaje con respecto a los cuerpos inalterada.

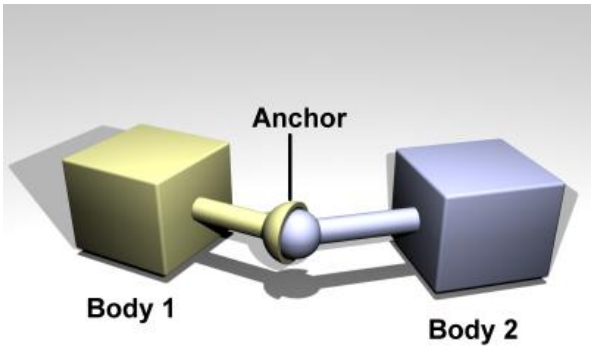

Figura 3: Empalme *ball and socket* ([Smi07]).

- Hinge: Su funcionamiento es análogo al de una bisagra. Permite a dos cuerpos rotar alrededor de un eje pero sin cambiar su distancia con respecto a dicho eje.

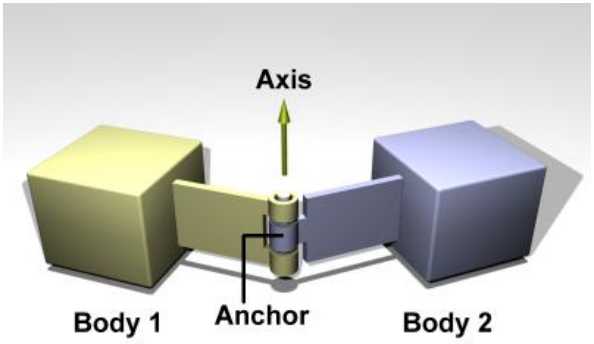

Figura 4: Empalme *hinge* ([Smi07]).

- Slider: Este empalme fija ambos cuerpos entre sí y les impone la restricción de que cada uno se pueda mover en un solo eje sin afectar al otro, esto asumiendo que dicho empalme no tiene restricciones en su movimiento. En caso de que uno de los cuerpos se mueva con respecto a los otros ejes, el movimiento del otro cuerpo será el mismo con respecto a estos otros ejes.

- Corkscrew: Es un empalme similar al anterior pero con un mayor grado de complejidad. Le permite a uno de los cuerpos rotar a medida que se mueve a lo largo del eje de

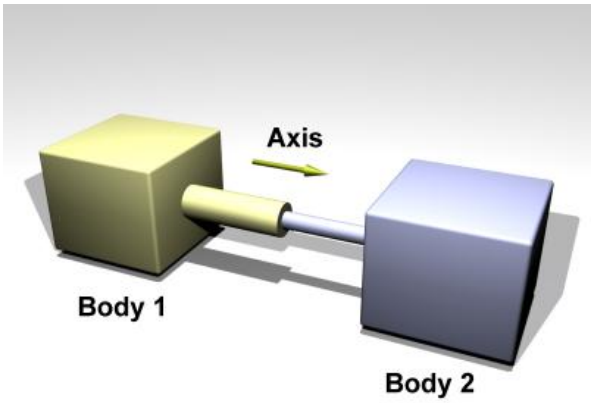

Figura 5: Empalme *slider* ([Smi07]).

movimiento del empalme, tal y como lo haría un tornillo. Se puede configurar cuanto rota un cuerpo en relación a su movimiento lineal a lo largo del eje del empalme.

- Universal: El comportamiento de este empalme es análogo al de tipo "Ball and socket" con la diferencia que presenta un grado de libertad más que dicho empalme. Dado el eje 1 en el cuerpo 1 y el eje 2 en el cuerpo 2, que es perpendicular al eje 1, este empalme mantiene estos ejes perpendiculares. En otras palabras, la rotación de ambos cuerpos con respecto a la dirección perpendicular a ambos ejes, va a ser igual. ([Smi07])

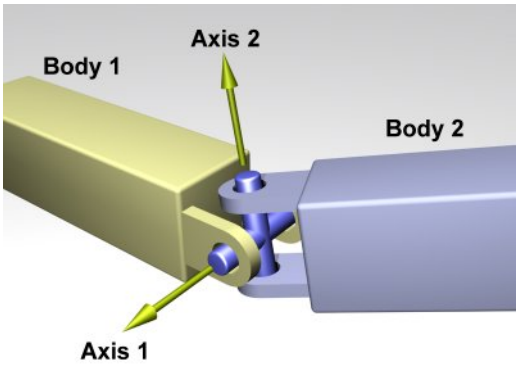

Figura 6: Empalme *universal* ([Smi07]).

- Hinge-2: Este empalme consta de dos empalmes de tipo "Hinge" conectados en serie, con diferentes ejes.

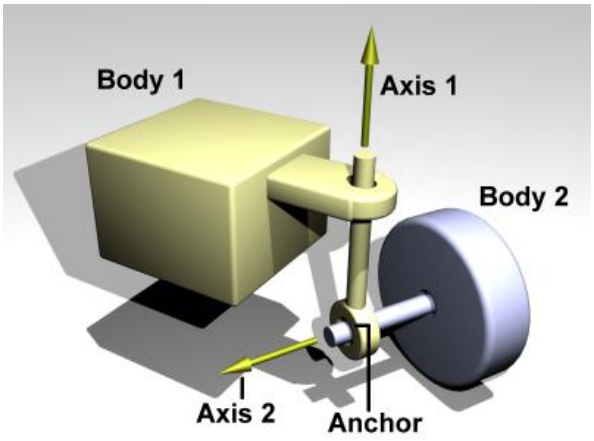

Figura 7: Empalme *hinge-2* ([Smi07]).

- Fixed: Este empalme mantiene fijas las posiciones relativas entre dos cuerpos. Si dos cuerpos están unidos mediante este empalme, mover uno de ellos resultaría en un movimiento análogo del otro cuerpo.
- Distance: Mantiene cierta distancia especificada entre dos cuerpos rígidos
- Point-In-Plane: Restringe a uno de los cuerpos asociados al empalme, a que se mueva a lo largo de un plano que contiene al otro cuerpo rígido asociado a dicho empalme. Permite a los cuerpos involucrados rotar en cualquier eje pero solo trasladarse a lo largo de dos ejes.
- Point-On-Line: Restringe a un cuerpo a que se mueva a lo largo de una línea fijada al otro cuerpo rígido con el cual está conectado. Se puede pensar como un empalme de tipo "Ball and Socket", el cual se puede trasladar a lo largo de un eje.
- Pulley: Su funcionamiento es similar al de una cuerda.

2.2.1. Motores físicos analizados

Se analizaron diversos motores físicos disponibles en la actualidad, con el objetivo de evaluar distintos factores y tener las herramientas necesarias para elegir correctamente uno de ellos. Se evaluaron diversos factores, como la facilidad de uso, la documentación, las licencias necesarias para utilizarlo, la factibilidad de modelar las situaciones requeridas, la estabilidad numérica de sus cálculos, entre otros.

El funcionamiento básico de los distintos motores físicos es similar. En primer lugar se deben crear las estructuras que van a interactuar en la simulación, o sea, los cuerpos y empalmes que conectan dichos cuerpos. Luego se avanza el estado de simulación en una cantidad dada de segundos, se detectan las colisiones, se resuelven dichas colisiones afectando el estado de los cuerpos de acuerdo a dicha colisión y se termina el ciclo de iteración. Las colisiones son resueltas de forma variada, pero de una forma o otra, se deben especificar las características de la interacción entre dos cuerpos cuando estos colisionan.

ODE

La librería ODE[ODE] es una de las librerías físicas más utilizadas, y por ende una alternativa muy importante para ser considerada al momento de elegir un motor físico. Es gratuita, multiplataforma, de código abierto y está implementada utilizando lenguaje C++, aunque su interfaz está en lenguaje C.

Presenta una variedad de primitivas de colisión adecuada para los requerimientos del sistema. Las primitivas de colisión que soporta son las siguientes: Sphere, Box, Capped Cylinder, Plane,Ray y Triangular Mesh. Ésta última puede resultar útil para modelar cuerpos de formas irregulares, pero en ODE esta primitiva acarrea diversos problemas de estabilidad y velocidad, por lo que en ocasiones puede afectar seriamente la simulación.

También presenta una gran cantidad de empalmes. Estos se nombran a continuación: Ball and Socket, Hinge, Hinge-2, Slider, Universal, Fixed, Contact, y Motor angular. El empalme de tipo contact se genera dinámicamente cuando se detecta que va ha ocurrido una colisión, y desaparece luego de un paso de la simulación. El empalme Motor angular se utiliza cuando se desean realizar controles en las velocidades angulares relativas entre dos cuerpos.

Cuando se detectan colisiones, las mismas se representan en la forma de un vector de puntos de contacto. Cada punto de contacto contiene la información acerca de sus coordenadas, su vector normal, la profundidad de penetración y que objetos están haciendo contacto. Para manejar las colisiones se debe especificar una función que se ejecuta en cada ciclo de simulación y modela el comportamiento de cada punto de contacto. En estos puntos se puede especificar, entre otras cosas, la fuerza de rozamiento entre ambos cuerpos.

En el manejo de colisiones de ODE se pueden distinguir dos conceptos importantes:

- Geoms o Objetos Geométricos: Son los objetos fundamentales del sistema de colisiones. Representan las primitivas de colisión básicas. Dos Geoms cualesquiera pueden colisionar entre sí, obteniéndose un conjunto de puntos de contacto. Se utilizan para representar cuerpos rígidos en la simulación.

- Spaces o Espacios: También es un Geom pero tiene la particularidad de agrupar otros Geoms. El objetivo de estos objetos es acelerar el manejo de colisiones de manera sensible, primero agrupando Geoms en Spaces y luego verificando colisiones de Geoms dentro de cada Space, por lo que se puede optar por no verificar las colisiones para todos los pares de Geoms existentes. Además se pueden verificar colisiones entre dos espacios determinados y utilizar jerarquías de espacio.

A cada ciclo de iteración, al avanzar el estado de la simulación se debe especificar el tamaño del paso de simulación en segundos. Al utilizar valores pequeños se obtiene mayor precisión en los cálculos y mayor estabilidad pero en contrapartida se enlentece la velocidad de la ejecución. Utilizar pasos de tiempo de tamaño variable entre una iteración y otra puede presentar problemas muy importantes en la simulación, según el autor de esta librería. Por esta razón es recomendable no modificar el paso de simulación a cada iteración, ya sea utilizando un valor fijo o modificarlo en intervalos de muchas iteraciones.

Las pruebas realizadas con esta librería condujeron a buenos resultados, ya que se logró modelar situaciones similares a las requeridas por el sistema a construir con un buen comportamiento general.

Newton Game Dynamics

Esta librería es de código cerrado, sin embargo es gratuita. Está implementada enteramente en lenguaje C, y es multiplataforma.

Según su documentación[NWI07], prioriza la exactitud física por sobre la velocidad en la simulación. Una característica importante es que resuelve el avance de la simulación de un modo determinista, lo que la hace una herramienta muy utilizada no solo en juegos sino también en simulaciones de situaciones reales, ya que la simulación puede ser reproducible.

Las primitivas de colisión soportadas son las siguientes: Box, Sphere, Cylinder, Cone, Capsule, Convex mesh y Ray. En Newton no existe el concepto de espacios, como existe en la librería anteriormente descripta

Los empalmes también son variados. Se pueden encontrar los empalmes Ball and socket, Hinge, Slider, Corkscrew, Universal y UpVector. También presenta la posibilidad de creación de nuevos tipos de empalmes, aunque esta es una tarea compleja.

El manejo de las interacciones entre los cuerpos se realiza tomando en cuenta los materiales de los mismos. Se representa el conjunto de materiales como un grafo no dirigido, en el cual los nodos son los identificadores de los distintos materiales y las aristas almacenan la información respecto a la interacción entre dos materiales. Se deben crear todos los materiales existentes en la simulación y luego para cada par de materiales posible se deben especificar las propiedades de su interacción, modelando así el comportamiento de puntos de contacto entre ambos materiales. Se debe especificar la dureza, la elasticidad y la fricción en dichos puntos de contacto. Adicionalmente se debe especificar si ambos materiales deben colisionar o no, y especificar una función de callback[5] que se ejecuta cuando esto ocurre. Aparte, existe un material por defecto que se le asigna a los cuerpos que no están asociados a ningún material.

Este enfoque para manejar las propiedades de las colisiones tiene como ventaja la facilidad en su utilización y ajuste, sin embargo se deben conocer de antemano todos los posibles materiales que van a existir en la simulación.

El manejo del paso de tiempo es muy flexible. Es posible avanzar el estado de la simulación una cantidad de tiempo variable para cada iteración. Esto es especialmente útil para avanzar la simulación al paso de tiempo real, ya que es posible, antes de realizar el paso de simulación, consultar cuánto tiempo ha pasado desde la última actualización del estado de la simulación y avanzar la misma ese lapso de tiempo.

Se realizaron pruebas utilizando esta librería modelando distintos tipos de robots. Al modelar robots móviles pequeños, de dimensiones similares a los robots de la categoría Simurosot de fútbol de robots de la FIRA, ocurrieron problemas de estabilidad numérica. No se descarta que sea posible modelar dichos robots con esta librería, pero esta no parece ser una tarea sencilla, por la inestabilidad que aparece cuando interactúan objetos pequeños.

Ageia PhysX

Esta librería [AGDEV] está programada en C++. No es multiplataforma, ya que está soportada solamente en entorno Windows.

Es extensamente utilizada en el desarrollo de juegos. Es una librería cerrada y tradicionalmente se vendió para proyectos comerciales y se utilizaba de forma libre en proyectos no comerciales, pero recientemente ha cambiado esta política y se volvió gratis para cualquier propósito. Presenta extensa y detallada documentación, con diversos documentos y ejemplos de uso. Según [AGBR07] presenta una muy interesante combinación de velocidad, precisión, robustez y estabilidad, transformándola en uno de los motores físicos más fieles a la realidad. Se pueden configurar muchas características de la simulación, como por ejemplo las unidades de masa y medida con las que se trabaja.

Las primitivas de colisión soportadas son las siguientes: Box, Sphere, Cylinder, Plane, Capsule, Convex Mesh y Heightfield. Ésta última se utiliza para crear el terreno de la si-

[5] Una función de callback es una porción de código ejecutable que es pasado como un argumento a otro código

mulación. Se puede determinar esta primitiva especificando triángulos como en la Convex Mesh pero esto menoscaba el rendimiento de la simulación, por lo que lo usual es crear esta primitiva especificando un conjunto de rectángulos.

Presenta una gran variedad de empalmes, los cuales se listan a continuación: Hinge, Ball and socket, Slider, Fixed, Distance, Point-In-Plane, Point-On-Line, Pulley y Universal. Es importante destacar que los empalmes se pueden romper, y cuando ocurre esto lo hacen de manera definitiva. Se puede especificar si un empalme es rompible o si no lo es, y en caso de que sea, se debe especificar la fuerza y el torque máximo que soporta antes de romperse.

El concepto de materiales en esta librería es similar al de la librería "Newton Game Dynamics", aunque un poco más flexible y complejo. Resultan determinantes en las características de la colisión entre dos objetos. Modelan no solo el material de la superficie de un objeto, sino también el material con el cual están hechos los objetos en su interior.

Presenta documentación exhaustiva de cada aspecto de la librería y numerosos ejemplos, sin embargo su utilización puede resultar compleja.

Otros

Se analizaron y evaluaron también los siguientes motores físicos: nvPhysics[NVPHY], Bullet Physics Library[BULL], Open Tissue[OPTI], Tokamak[TOKA] y True Axis[TAXIS]. Los mismos comparten cualidades y defectos con algunos de los motores descriptos anteriormente.

2.3. Casos de estudio

2.3.1. GSim

GSim[6][ACGS03] es un simulador desarrollado en el marco del proyecto de grado del equipo F.R.U.T.O. La principal motivación para la construcción de este simulador fue contar con un entorno en el cual se lograra realizar tareas de aprendizaje automático. Se buscaba contar con una API que se pudiera utilizar para ejecutar simulaciones automáticamente, controlando dichas simulaciones utilizando un software programado para este tarea. Este simulador es, por lo tanto, una alternativa al simulador proporcionado por la FIRA, en la cual no se puede llevar esto a cabo.

Este simulador resuelve el movimiento de los robots y sus colisiones en forma geométrica, sin basarse en un motor físico externo que respalde la simulación. Esto puede conducir a simulaciones que resulten ser físicamente poco realistas.

Sin embargo presenta numerosos puntos fuertes, como permitir establecer el estado inicial de los robots, permitir configurar diversos aspectos de la simulación incluyendo el paso de simulación y además permite ejecutar secuencialmente cada estrategia antes de avanzar el paso de simulación. A continuación se verán las principales características de GSim.

En el simulador se ejecutan cinco hilos distintos:

[6] Geometric Simulator

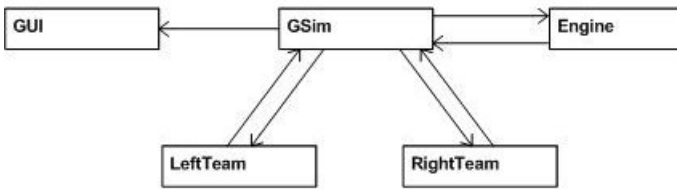

Figura 8: Hilos de ejecución de GSim y flujo de información entre los mismos ([ACGS03]).

- GSim: es el hilo de ejecución principal, el hilo padre de todos los demás. Se encarga de conectar y sincronizar el flujo de información entre todos los demás hilos
- Engine: es el hilo en el cual corre el motor de simulación
- LeftTeam: es el hilo en el cual se ejecuta la estrategia del equipo izquierdo
- RightTeam: es el hilo en el cual se ejecuta la estrategia del equipo derecho
- Gui: es el hilo en el cual se ejecuta todo lo referente a la representación en la pantalla del estado de la simulación.

La interfaz de comunicación del simulador con las estrategias está hecha de forma tal que el impacto de trasladar una estrategia desde el simulador GSim al simulador oficial de la Fira sea el mínimo posible. La comunicación del hilo GSim tanto con el hilo Engine como con los hilos de las estrategias es bidireccional y se lleva a cabo mediante el uso de colas de escritura bloqueante y lectura no bloqueante. La comunicación con la interfaz gráfica, sin embargo, es unidireccional y se utiliza la misma cola que en el caso anterior.

Para modelar el comportamiento físico de la simulación se utilizó un enfoque híbrido, modelando las colisiones desde un enfoque geométrico y modelando el resto del comportamiento con un enfoque físico, o sea, basado en leyes físicas.

Se consideró al robot como una carrocería cúbica con dos ruedas independientes a ambos lados de la carrocería, controlada cada una por un motor. Se puede representar en cualquier momento el estado de un robot mediante su ubicación, orientación, velocidad lineal y velocidad angular. Tanto la velocidad lineal como la angular se pueden traducir en términos de la velocidad en cada una de las ruedas.

El movimiento del robot, por lo tanto, está determinado por la velocidad relativa entre ambas ruedas. Además se estableció que debería existir una aceleración máxima y una desaceleración máxima para impedir que el robot cambie de velocidad instantáneamente, por lo que a cada ciclo de simulación se normalizan las velocidades de las ruedas para que las mismas cumplan con las restricciones de aceleración o desaceleración. Luego de normalizar la velocidad, se deben ajustar los valores para que la velocidad relativa entre ambas ruedas sea realmente la deseada.

La pelota se modeló como una partícula, por lo que a diferencia de los robots no tiene rotación ni velocidad angular, de esta manera el estado de la pelota queda determinado por su posición y su velocidad lineal. Además para modelar el rozamiento de la pelota con la cancha, se utilizó una constante que afecta la velocidad de la misma.

Para modelar las colisiones se utilizó un modelo puramente geométrico. Se detectan colisiones de los robots con las paredes, de la pelota con las paredes y de los robots con la pelota, pero no de robots con robots. Por ejemplo, para detectar colisiones de un robot con las paredes, se realiza una aproximación de la trayectoria del centro de masa del robot con un segmento de recta, de manera tal que los extremos del segmento coincidan con las posiciones del centro de masa de robot antes y después de avanzar el paso de simulación.

2.3.2. 3D Robot Soccer Simulator

3D Robot Soccer Simulator[RSOCF] es el simulador oficial de la categoría Simurosot de la FIRA[FIRA]. Su código es cerrado por lo que se tiene muy poca información acerca del funcionamiento interno del mismo. Está implementado utilizando el motor físico Havok, Shockwave 3D y lenguaje C. Presenta una interfaz gráfica en la cual se puede observar el campo de juego con los robots y la pelota. Además hay una variedad de opciones para controlar los partidos. Se puede ver el marcador actual y el tiempo de juego, que pueden ser modificados. Se puede pausar en cualquier momento la simulación y tomar alguna acción en el partido, como cobrar un penal, tiros libres, etc. Además existen repeticiones, en las cuales se puede ver lo que sucedió en el partido desde distintos ángulos y con diversas funcionalidades para controlar el paso del tiempo en la repetición.

Una de las limitaciones de este simulador es el hecho de que las estrategias solo puedan estar implementadas en lenguaje Lingo o en C++. Para cargar las estrategias, se debe ingresar para cada equipo el nombre del archivo con la implementación de cada estrategia. En caso de estrategias implementadas en Lingo, estas deben estar en un archivo de texto y se ingresa el nombre de este archivo. En caso de estrategias implementadas en C++ las mismas deben estar compiladas en una DLL. Las estrategias son llamadas 60 veces por segundo por el simulador, de manera bloqueante y de forma secuencial.

2.3.3. Robocup Soccer Simulator [SROBC]

Este simulador es utilizado en la liga simulada 2D de la RoboCup[ROBCU][SROBC]. Su arquitectura fue pensada basándose en un paradigma multi-agente.

Existen dos tipos de agentes, los jugadores y los entrenadores. Los jugadores básicamente ejecutan dos tipos de acciones: primarias y concurrentes. Las acciones primarias solo pueden realizarse una vez en cada ciclo de simulación y la concurrentes pueden realizarse al mismo tiempo varias de ellas concurrentemente con una acción primaria. El simulador sólo ejecuta una acción primaria aunque el jugador solicite la ejecución de más de una de ellas.

Los jugadores reciben información de su entorno mediante tres tipos de sensores, los mismos son: el sensor físico, el sensor visual y el sensor auditivo. El sensor físico brinda información como la energía actual del robot o su velocidad actual. El sensor auditivo provee al robot información referente a los mensajes enviados por otros jugadores y por el árbitro, estos mensajes están limitados tanto en calidad como en cantidad según parámetros configurables en el simulador. El sensor visual brinda información acerca de objetos cercanos al robot. Si el objeto se encuentra en determinado cono de visión, se puede obtener información acerca de la distancia y la dirección del mismo, pero si se encuentra más lejos pero dentro de otro radio de proximidad un poco mayor, se puede detectar al objeto pero sin información sobre distancias o direcciones.

El enfoque de este simulador es bastante diferente al que se busca para este proyecto, ya que no está basado en leyes físicas, sino que el avance de la simulación se da a partir de un enfoque geométrico influenciado por numerosos parámetros y reglas. Sin embargo, su arquitectura y mecanismos de comunicación presenta conceptos de interesante valor. El paradigma multi-agente en el cual está basado su arquitectura implica un grado de independencia entre módulos que actúan en la simulación que vale la pena destacar.

2.3.4. MiS20 [MIS20]

El simulador MiS20[7] [MIS20] surgió como un intento por poseer una alternativa al simulador de la categoría MiroSot Middle League de Fira para perfeccionar la estrategia utilizada por el equipo Mi20[8]. Esta necesidad surge a partir de las carencias del simulador oficial como la necesidad de constante intervención humana en los partidos y una simulación con poca precisión. MiS20 presenta numerosas ventajas deseables en cualquier simulador. Permite establecer las posiciones iniciales de los jugadores, calibrar diversos parámetros de la simulación, se puede controlar el paso de simulación, se puede entrenar diversas situaciones de juego y permite grabar y reproducir partidos. En analogía a GSim, tampoco posee un motor físico externo que lo respalde y resuelve colisiones de una manera similar.

2.3.5. Simulator Bob

Simulator Bob es un entorno de simulación 3D diseñado para simular una amplia variedad de robots móviles. Utiliza ODE[Smi07] para realizar el modelado físico y OpenGL[OPENGL] para la visualización gráfica. Su interfaz gráfica fue realizada utilizando MFC[9], y por lo tanto se puede utilizar exclusivamente en sistemas Win32. Es una aplicación gratuita y de código abierto.

El funcionamiento de este entorno de simulación se basa en la interacción de componentes, los mismos se pueden clasificar en:

- Mecánicos: son cuerpos rígidos conectados por empalmes.
- Actuadores: permite aplicar fuerzas y torques a los cuerpos.
- Sensores: realiza mediciones siguiendo ciertos criterios.
- Controladores: módulos de software que pueden acceder a datos de sensores, actuadores y otros controladores.

Para controlar el paso de tiempo de la simulación solamente se presenta tres opciones: empezar, pausar o detener. En la primera opción simplemente se continúa la simulación desde el estado inicial o el último estado antes de una detención. Tanto la segunda como la tercera opción se pueden utilizar para interrumpir momentáneamente la simulación, pero su diferencia radica en que en la tercera opción todas las velocidades de los cuerpos se llevan a cero.

Se dispone de una funcionalidad de grabación de simulaciones, por la cual es posible grabar en un fichero AVI un vídeo de la simulación desde el punto de vista de la cámara activa en el momento.

Además se puede especificar el nivel de detalle y distintas opciones gráficas, de modo de ajustar la performance a las necesidades. Se pueden especificar distintos modos de representación gráfica, configuración de la iluminación, entre otras propiedades.

[7] Mission Impossible Simulator
[8] Mission Impossible Twenty
[9] Microsoft Foundation Classes

2.3.6. Webots

Webots es un avanzado simulador de robótica. Trae herramientas para definir modelos propios, definir las características de la simulación física y escribir controladores para los robots. Tiene versiones para Windows, MacOS y Linux, pero tiene la desventaja de ser una aplicación de código cerrado y no gratuita. Utiliza ODE[Smi07] para realizar el modelado físico y OpenGL[OPENGL] para la visualización gráfica.

Se pueden modelar robots y simulaciones tan complejas como soporte la librería física, ODE en este caso. El tiempo de simulación es muy flexible y puede ser ajustado de acuerdo a las necesidades. Es posible avanzar a la simulación paso a paso, a velocidad normal o a velocidad avanzada, en este último caso sacrificando precisión.

La información acerca del modelo que se desea simular es obtenida desde un archivo. Este archivo contiene toda la información acerca de los objetos que componen la simulación, sus propiedades físicas, actuadores, sensores, controladores de robots, propiedades de visualización y varias cosas más. Este archivo puede ser creado o editado desde la interfaz de usuario, a través de una interfaz amigable y fácil de utilizar, creando de esta forma los objetos deseados. Existen una gran cantidad de actuadores y sensores disponibles que se pueden utilizar para que los robots tengan el comportamiento y releven la información deseada.

Es posible grabar vídeos de las simulaciones en ficheros AVI o WMV, pero no es posible grabar la simulación propiamente dicha, para luego reproducirla utilizando esta misma aplicación.

El comportamiento de los robots es manejado por controladores. Webots incluye interfaces en C, C++ y Java, a través de las cuales se puede programar directamente un controlador de un robot. Para cada robot que se desee controlar, es necesario indicarle a la aplicación el ejecutable que contiene el programa de control, ya sea de extensión .exe en el caso de C y C++, o .class en caso de haber sido desarrollado en java.

La interfaz de usuario posee la funcionalidad de permitir el control mediante el mouse de los objetos de la simulación de manera muy sencilla, por lo que resulta útil para realizar simulaciones interactivas.

Esta aplicación además provee la funcionalidad de transferencia de controladores de robots hacia diversos tipos de robots reales comerciales existentes en la actualidad.

2.4. Construcción de un prototipo

2.4.1. Motivación

La principal motivación para la creación de un prototipo fue la necesidad de reducir riesgos técnicos en diversas áreas. Por un lado, se deseaba tener elementos para lograr realizar una correcta elección de las librerías a utilizarse en este proyecto. Estas librerías se utilizarían para implementar el motor físico y el motor gráfico del sistema final, por lo que fue de vital importancia elegir las mismas de forma correcta. Además se buscaba obtener experiencia en el manejo de dichas librerías, especialmente en el manejo del motor físico del simulador, el cual presenta cierto grado de complejidad. Se deseaba elegir el motor físico de tal manera que con el mismo se lograra cumplir con todos los requerimientos del sistema. Por otro lado se deseaba estudiar la integración entre el componente físico y el componente gráfico del simulador, con el objetivo de independizar ambos componentes.

Finalmente, otro aspecto que motivó fuertemente la creación de un prototipo, fue el deseo de crear un sistema simplificado que cumpliera con buena parte de los requerimientos más críticos del sistema final, el cual se pudiera evolucionar para llegar al sistema deseado.

2.4.2. Alcance

En este prototipo se planteó la creación de una simulación de robots simplificada que presentara las características que se describirán a continuación.

2.4.3. Configuración del entorno de juego

El modelo físico del entorno de juego debe ser configurable. El mismo debe ser obtenido por el simulador a través de un archivo hecho por el usuario. Para esto se definió una sintaxis en la cual se define el terreno de juego. Como una simplificación, el entorno de juego debe estar compuesto simplemente por un conjunto de cajas, a las cuales se le puede especificar diversos parámetros.

2.4.4. Configuración de los robots

El modelo físico de los robots debe ser configurable. El mismo, al igual que el entorno de juego debe ser obtenido desde un archivo. Para esto se realizaron diversas simplificaciones. En primer lugar, el chasis del robot está compuesto por una caja a la cual se le especifican sus características. Además, solo se requiere un tipo de empalme, el cual conecta al chasis con una rueda. El robot puede tener cualquier cantidad de ruedas, a las cuales se les puede especificar sus características. Cada una de dichas ruedas con su respectivo empalme que las conecta con el chasis. A este empalme también se le puede especificar sus características, como sus ejes.

2.4.5. Visualizar el estado de la simulación

El estado de la simulación debe poder ser visualizado a través de una interfaz gráfica. La misma debe ser una librería externa al motor físico, por lo que se debe estudiar su integración. No es deseable que exista una dependencia entre esta librería y el motor físico elegido para esta implementación en particular.

2.4.6. Comunicación asincrónica con procesos

Para controlar los robots se debe implementar un protocolo de comunicación simple con un proceso que los controle. El mismo debe obtener el estado del mundo comunicándose con el simulador y enviarle información acerca de la velocidad que se desea aplicar a cada rueda.

2.4.7. Configuración del paso de tiempo

El prototipo debe proveer la funcionalidad de configurar el paso del tiempo. Esto significa que se debe poder avanzar la simulación a velocidad acelerada o también hacerlo paso a paso. Se deben estudiar los límites que deberán existir sobre este requerimiento y los efectos no deseados que pueden llegar a ocurrir al no respetar dichos límites.

2.4.8. Resultados

Motor físico

Entre los diversos motores físicos que podrían llegar a cumplir los requerimientos de este proyecto, se decidió utilizar la librería Newton Game Dynamics[NGD]. Se tomó esta decisión con base en las buenas características de esta librería ya estudiadas en [AFE07]. Fundamentalmente, la flexibilidad en el manejo del paso del tiempo y la facilidad de uso la transformaba en una interesante alternativa.

En un principio se notó un buen comportamiento del modelo físico del simulador llevado a cabo utilizando esta librería. Sin embargo al realizar pruebas utilizando robots pequeños, análogos a los robots de la categoría MiroSot Middle League de la FIRA[FIRA], se observó que esta librería presentaba serios problemas de estabilidad numérica. Es probable que ajustar los parámetros de la simulación de tal manera que se logre un correcta simulación sea posible, pero la escasa experiencia en la utilización de esta librería no permite asegurar con certeza dicha hipótesis. De esta manera, se descartó la utilización de dicho motor físico.

La siguiente alternativa que se evaluó fue la utilización de la librería ODE. Esta librería no presenta tanta flexibilidad en el manejo del paso del tiempo como la anterior alternativa. Según su manual[Smi07] no es recomendable cambiar el tamaño del paso de simulación a cada iteración, ya que se puede caer en pérdidas de precisión importantes. Sin embargo presenta diversas cualidades que implican que esta sea una alternativa a tener en cuenta. Se logró modelar una correcta simulación utilizando esta librería. Se logró que utilizando los archivos de configuración, se creara un modelo físico correspondiente a la configuración expresada a través de los mismos. Por otro lado, no se observaron inestabilidades numéricas serias como se observaron con la anterior alternativa. Se probaron diversas configuraciones de robots y diversas cantidades de los mismos y se obtuvieron resultados satisfactorios. Además, se logró cumplir los requerimientos de configuración del avance del paso del tiempo, logrando avanzar el mismo de paso a paso, o a velocidad acelerada.

Una consideración muy importante a tener en cuenta es el hecho de que ODE debe utilizar lapsos de tiempo de largo fijo para avanzar la simulación. Esto significa que no se puede cambiar este período en cada paso de la iteración, sino que debe ser siempre el mismo. Utilizar largos variables, lleva según su autor, a efectos no deseados en las simulaciones, como problemas de precisión matemática en los cálculos.

Un punto a tener en cuenta es que la velocidad con la que se ejecuta la simulación puede variar notablemente dependiendo de las prestaciones de la computadora sobre la cual se ejecuta. Este prototipo no presenta mecanismos que contemplen estas variaciones y logre ajustar el comportamiento del simulador a las mismas.

El requerimiento de manipulación del paso de tiempo, fue cumplido satisfactoriamente utilizando esta librería.

Motor gráfico

La librería física utilizada incluye una librería gráfica muy fácil de utilizar denominada "drawstuff", pero acoplada fuertemente al componente físico específico. Fue muy útil su utilización en las pruebas realizadas sobre el motor físico, sin embargo un interesante desafío era integrar la librería ODE con una librería gráfica externa, manteniendo la independencia entre estos módulos. Se utilizó la librería Irrlicht[IRRL], la cual resultó de fácil utilización e integración a la simulación física, ya que presenta diversos ejemplos y buena documentación en su sitio oficial. Especialmente presenta documentos de como integrarla con diversas librerías físicas, incluyendo ODE.

Es importante destacar que antes de iniciar la simulación es posible elegir el método de visualización deseado, que puede ser drawstuff o Irrlicht. En las figuras 9 y 10 se pueden ver imágenes de ambas alternativas en una simulación utilizando 3 robots.

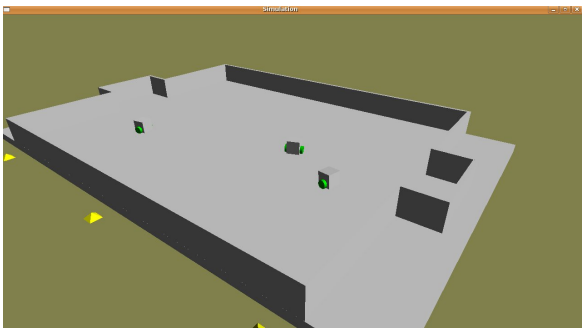

Figura 9: Prototipo. Simulación utilizando drawstuff

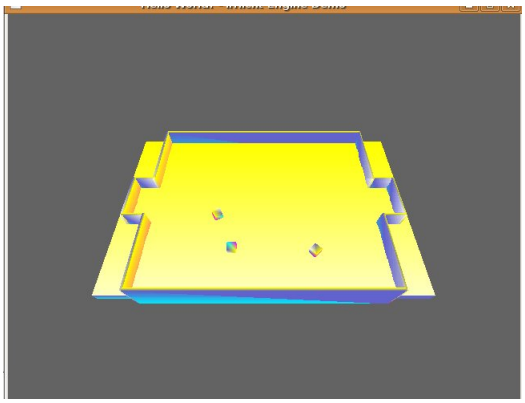

Figura 10: Prototipo. Simulación utilizando Irrlicht

Comunicación

Se implementó una estrategia muy simple para controlar los robots. Su funcionamiento consiste simplemente en enviar velocidades aleatorias dentro de un rango dado. Para realizar la comunicación de dicha estrategia con el simulador, se implementó un algoritmo sincrónico, en el cual el simulador iniciaba la ejecución de un hilo, y en este hilo enviaba la información del estado de los robots y aguardaba la respuesta de forma sincrónica. Esta respuesta contenía las velocidades de las ruedas de los robots generadas en el proceso que implementaba una estrategia simple. Se obtuvieron resultados satisfactorios en este prototipo de comunicación.

Luego de esto, se implementó un nuevo protocolo de comunicación, que tiene como característica principal el asincronismo. El simulador crea dos hilos nuevos que se utilizan para la comunicación con la estrategia. En uno de los hilos se hace difusión del estado del mundo a través de multicast, de forma tal que se puedan registrar procesos para escuchar dicha difusión de forma dinámica. En el otro hilo se reciben paquetes, que en este caso son los enviados por el proceso que controla los robots. Dicho proceso inicia un hilo que recibe información del mundo a través de multicast y actualiza su información acerca del estado del mundo. Por otra parte, envía cada cierto lapso de tiempo un paquete que contiene las velocidades deseadas de los robots, que es recibido por el simulador.

Se cumplieron los objetivos de comunicación planteados, ya que la misma funcionó correctamente y no se detectaron desviaciones del comportamiento normal del simulador por el hecho de utilizar recursos computacionales en tareas de comunicación.

Conclusiones

Se cumplieron satisfactoriamente con los requerimientos propuestos en el contexto de este prototipo. Se generó experiencia en el manejo de librerías claves, lo que fue de gran importancia en el desarrollo del sistema final. Por otra parte, se considera que los requerimientos propuestos para el desarrollo de este prototipo fueron acertados, ya que contribuyeron a la mitigación de diversos riesgos técnicos y implicaron investigación en puntos claves. Además, se generó código que fue reutilizado, principalmente en lo que respecta a la utilización del motor físico elegido.

3. Descripción de la solución

En este capítulo se describirán los aspectos más relevantes de la solución propuesta para el sistema planteado. Se introducirán algunos conceptos importantes para entender el funcionamiento del sistema, explicando las principales características de los mismos. Además, se describirá la arquitectura de un modo general, analizando los bloques constructivos sobre los cuales se basa la solución.

3.1. Aspectos Generales

Un requerimiento importante del sistema es el hecho de que diversas características de la simulación puedan ser manipuladas. También debe ser posible de alguna manera especificar y crear los objetos que se desean modelar, como por ejemplo el entorno de juego, los robots y otros elementos que se quieran modelar. Además debe existir algún mecanismo mediante el cual dichos robots puedan ser controlados por estrategias o programas implementados por el usuario del sistema. Para lograr esto se consideró la posibilidad de aplicar los conceptos de Sistemas Multi-Agentes o MAS para modelar esta realidad.

Un sistema Multi-Agente se puede definir como un conjunto de agentes que interactúan para lograr un objetivo. Un agente de software es un sistema de computación capaz de realizar acciones de forma autónoma en el ambiente en que está situado. Éstos son por lo general vistos como entidades inteligentes, equivalentes en términos computacionales a un proceso del sistema operativo, que existen dentro de cierto contexto o ambiente, y que se pueden comunicar a través de un mecanismo de comunicación inter-proceso, usualmente un sistema de red, utilizando protocolos de comunicación. En cierto modo, un sistema multiagente es un sistema distribuido en el cual los nodos o elementos son sistemas de inteligencia artificial, o bien un sistema distribuido donde la conducta combinada de dichos elementos produce un resultado en conjunto inteligente [WMAS].

El sistema construido en el contexto de este proyecto podría verse a través del enfoque de los sistemas multiagentes. Las simulaciones son llevadas a cabo a partir de la interacción del núcleo de simulación y los agentes externos al mismo. El núcleo de simulación es el agente que mantiene el modelo físico de la realidad, lleva a cabo la evolución del modelo y efectúa acciones sobre éste. Los agentes externos mantienen un diálogo con el núcleo de simulación, enviando y recibiendo mensajes.

En el núcleo de simulación, la recepción de mensajes desencadena la ejecución de comandos, y estos comandos son los responsables de efectuar acciones sobre el estado de la simulación. De esta forma los agentes externos pueden ejercer control sobre la simulación. Los mensajes que son enviados desde el núcleo de simulación hacia los agentes externos son de carácter informativo, contienen respuestas a peticiones de los agentes, información del estado del mundo, etc.

El comportamiento de los robots se debe implementar como un agente externo, que a través de mensajes puede lograr la creación de dichos robots en el modelo físico del núcleo de simulación, así como también el control de los mismos.

Los agentes externos conectados al núcleo de simulación eventualmente podrían tener objetivos muy variados, podrían ser procesos que controlan robots, interfaces gráficas que reciben información de la simulación y luego la despliegan de alguna manera, aplicaciones que solo registran información, etc. A partir del momento que el agente que actúa como núcleo de simulación se inicia, en cualquier momento otros agentes pueden conectarse a él, con la posibilidad de llevar a cabo distintas acciones que se especifican a través de los mensajes intercambiados.

Un esquema de esta realidad se puede ver a continuación.

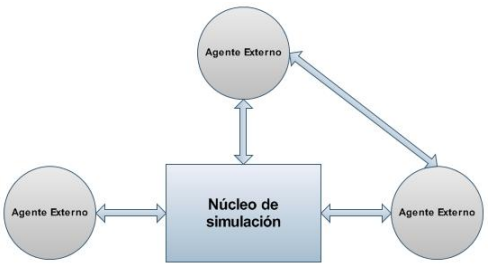

Figura 11: Visión MAS del sistema

3.2. Aspectos de modelado

Naturalmente el deseo del usuario al utilizar un simulador es lograr modelar situaciones que se comporten de manera tan similar como sea posible a un modelo de referencia real. Existen un gran número de factores a tener en cuenta al momento de realizar modelos simulados, pero entender correctamente algunos conceptos centrales en el sistema es importante, como los son los conceptos de Entidades y Comandos. Dichos conceptos se describen a continuación, explicando sus principales características y la forma como interactúan para lograr el objetivo de modelar realidades.

3.2.1. Entidades

Los objetos son cuerpos y empalmes con determinada disposición en el mundo simulado que interactuan entre sí. Inicialmente se debe definir que objetos van a interactuar en la simulación. Para agrupar conjuntos de objetos relacionados, se presenta el concepto de entidades.

Concretamente una entidad está compuesta por cuerpos y empalmes que los conectan. En términos más abstractos se puede decir que una entidad agrupa comandos y restricciones. Agrupa comandos porque cada tipo de entidad tiene una serie de comandos de entidad relacionados a él. Estos comandos son responsables de llevar a cabo la ejecución de operaciones sobre la entidad, como moverla o controlar motores en caso de que la entidad lo amerite. Agrupa restricciones porque ciertos tipos de entidades pueden estar conectados a ciertos tipos o cantidades de objetos.

En el contexto de este proyecto se crearon tres tipos de entidades: robots, campos de juego y pelota. Una entidad de tipo robot encapsula comandos para mover la entidad en el espacio, controlar los motores, entre otros. Además, no presenta restricciones en cuanto a los objetos que la componen, o sea, que puede contener la cantidad y tipos de cuerpos y empalmes que se desee. Una entidad de tipo campo de juego no presenta comandos asociados a ella, ya que no se consideró necesario. Por otra parte, presenta restricciones en cuanto a los objetos que la componen; esta entidad está asociada a cualquier cantidad de cuerpos, pero no admite empalmes en su composición. Una entidad de tipo pelota encapsula el comando que la mueve en la simulación. También presenta restricciones en cuanto a su composición, ya que solamente puede estar compuesta por un cuerpo, y ese cuerpo debe ser de tipo esfera.

Espacios

Las entidades también juegan un papel fundamental en la optimización de colisiones. Cada ciclo de iteración el sistema debe verificar la colisión entre los distintos cuerpos que componen

el modelo físico. Una estrategia para hacer esto es verificar si ha ocurrido una colisión para todos los pares de cuerpos posibles en la simulación. Esto puede ser muy costoso en términos computacionales si la cantidad de colisiones esperadas o la cantidad de cuerpos es alta. Una manera más eficiente de detectar colisiones es agrupando cuerpos en espacios. Utilizando estos espacios se detectan en primer lugar que espacios pueden estar potencialmente colisionando, y en caso que esto ocurra, verificar las colisiones entre cada par de cuerpos posible, tal que pertenezca uno a cada espacio.

Para resolver las colisiones, por lo tanto, agrupar los cuerpos en espacios es una técnica que utilizan los motores físicos para optimizar la resolución de este problema. También se podría asumir que cuerpos en un mismo espacio son entidades conexas y no es deseable detectar colisiones internas allí. Esta es una de las funciones más importantes de las entidades, agrupar cuerpos en espacios asumiendo que no es necesario detectar colisiones internas a los espacios. En la actual implemenentación del sistema, no se verifican colisiones entre cuerpos pertenecientes a una misma entidad.

Para visualizar este concepto se puede observar la figura 12, en la que se encuentran representados una serie de cuerpos. Los objetos pertenecientes a un mismo espacio están representados con un mismo color y agrupados dentro de una circunferencia. En este ejemplo, cada circunferencia representa un espacio. Si dos circunferencias se intersectan en algún punto, esto significa que los espacios que ellas representan están posiblemente colisionando. Cabe destacar que esta agrupación a través de una circunferencia se realiza a modo de ejemplo y con el objetivo de facilitar la visualización. Los cuerpos que pertenecen a un mismo espacio no necesariamente se encuentran cercanos o agrupados espacialmente.

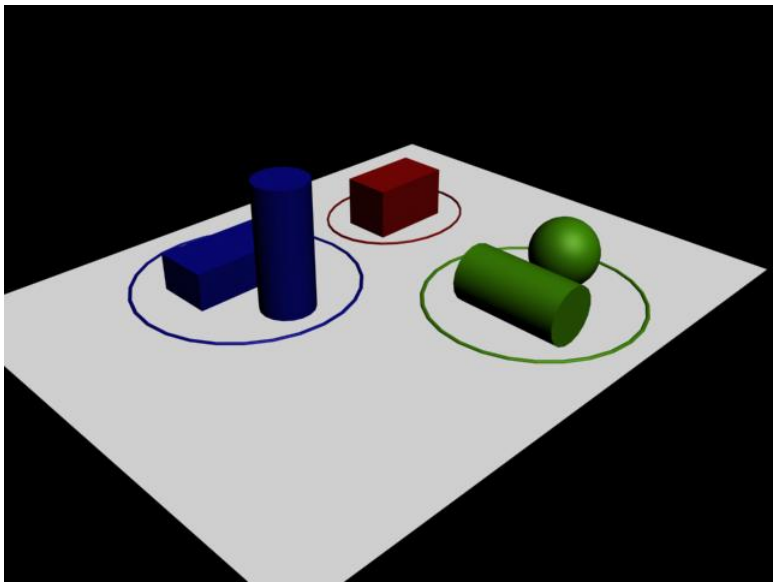

Figura 12: Cuerpos agrupados en espacios

Existen tres espacios en esta situación, dos de los cuales agrupa dos cuerpos cada uno y el restante está formado por un cuerpo. Para detectar colisiones en un escenario como este, en primer lugar se verifican colisiones entre espacios mediante una heurística resuelta por el

motor físico. Dicha heurística es aplicada a cada par de espacios en la situación. En este caso no se detecta ninguna colisión entre espacios.

Realizando un cambio de posiciones de los cuerpos de color verde, se obtiene la situación que se muestra en la figura 13.

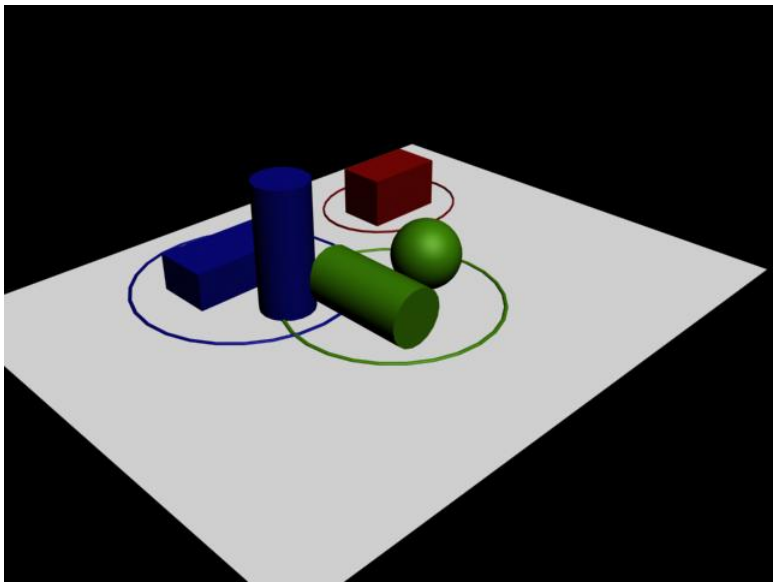

Figura 13: Cuerpos agrupados en espacios, con colisión

En este segundo escenario existe una colisión entre cuerpos de distintos espacios. Para detectar esta colisión se debe detectar primero que espacios están potencialmente colisionando. Luego de aplicar una heurística, verificando colisiones para cada par de espacios posible, se llega a la conclusión que solamente existe una posible colisión entre espacios. Dicha colisión se da entre el espacio que contiene cuerpos de color verde y el espacio que contiene cuerpos de color azul. Una vez realizada esta detección, se procede a verificar colisiones para cada par de cuerpos, tal que uno de ellos pertenece a un espacio y el otro pertenece al otro espacio implicados en la colisión. De esta forma, es posible llegar a la conclusión que solamente existen colisiones entre el cilindro de color verde y el cilindro de color azul, como se puede ver con más detalle en la figura 14.

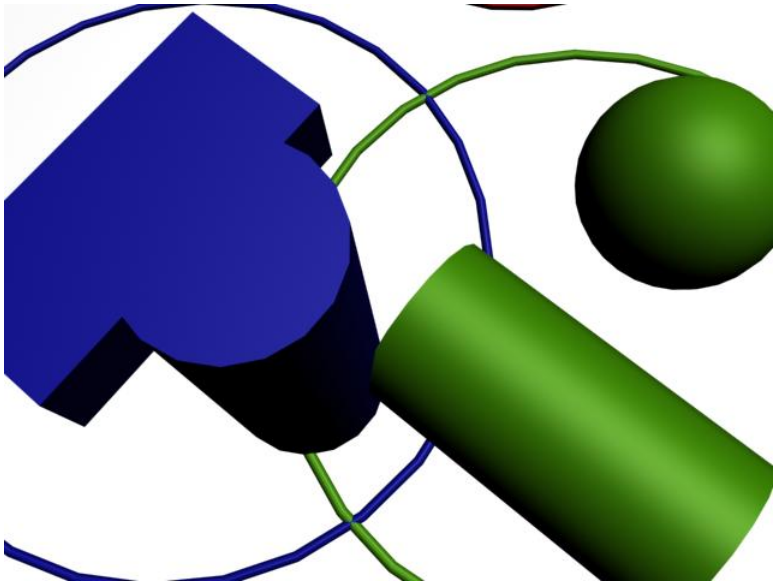

Figura 14: Colisión entre cuerpos, ampliación

3.2.2. Comandos

Los comandos encapsulan gran parte de la lógica del sistema. Todas las acciones efectuadas por los agentes externos que se conectan al sistema son llevadas a cabo por comandos. Una vez que el sistema detecta la petición para ejecutar comandos con determinados parámetros, se identifican las instancias de los comandos correspondientes y se procede a la ejecución de los mismos. En este momento la ejecución del ciclo de iteración se encuentra interrumpida a la espera de la ejecución de dichos comandos, que se ejecutan secuencialmente respetando el orden de llegada de las solicitudes, y la ejecución del ciclo de iteración continúa una vez que se haya terminado la ejecución de estos comandos. De esta manera se realizan acciones sobre la simulación.

Existen dos tipos de comandos:

- Comandos de entidad: tienen como objetivo realizar alguna acción sobre alguna entidad específica de la simulación. Al momento de su ejecución, el comando además de tener a su disposición todas las interfaces de los distintos controladores del sistema, puede acceder a operaciones de la interfaz que provee la entidad a la cual está destinado. Ejemplos de comandos de entidad son los responsables de mover entidades, o controlar motores de robots.

- Comandos de control: tienen como objetivo realizar alguna acción general sobre la simulación. Al contrario del caso anterior, no tienen una entidad asociada a él. Es importante destacar que la creación de una entidad es un comando de control, ya que al momento de su ejecución no hay una entidad asociada a él en el sistema, sino que va a ser creada posteriormente.

3.3. Arquitectura

Luego de comprender algunos conceptos generales del sistema, es posible analizar con mayor detenimiento la estructura interna del sistema desarrollado. Su arquitectura está compuesta por diversos módulos que interactúan para lograr el objetivo de llevar a cabo simulaciones robóticas. A partir de la descomposición en módulos realizada se obtuvo una arquitectura con fuerte foco en la extensibilidad del sistema. Cada módulo fue diseñado tomando en cuenta que cambios internos en el funcionamiento de dicho módulo no afecten sus interfaces de comunicación, de forma tal que estos cambios sean transparentes para el resto del sistema.

Los módulos principales que componen al núcleo de simulación son:

- Comunicación: para todas las tareas de comunicación con agentes externos, sincronización, entre otras.
- Motor de simulación: responsable de la coordinación de todas las variables de simulación y administración general de la simulación.
- Comandos: responsable de mantener los comandos de control que tienen como objetivo efectuar acciones que alteren el estado de la simulación.
- Interfaz física genérica: responsable de la representación física del modelo simulado. Expone una serie de interfaces genéricas para la creación y mantenimiento de distintos tipos de estructuras físicas. Encapsula un módulo denominado ODE, cuya implementación está orientada a la biblioteca física homónima, implementando todas las interfaces propuestas por la interfaz física genérica.
- Entidades: responsable de mantener todas las entidades existentes en el sistema y realizar las tareas necesarias siempre que se desee crear alguna entidad o realizar acciones sobre entidades existentes. Encapsula un módulo denominado "Comandos de entidad", que se encarga de mantener los comandos asociados a una entidad en particular. Estos comandos tienen acceso total a una entidad específica en el sistema, además de las distintas interfaces a las que tienen acceso los comandos de control.

Un diagrama de la arquitectura planteada se muestra a continuación, mostrando el flujo de información entre los distintos módulos.

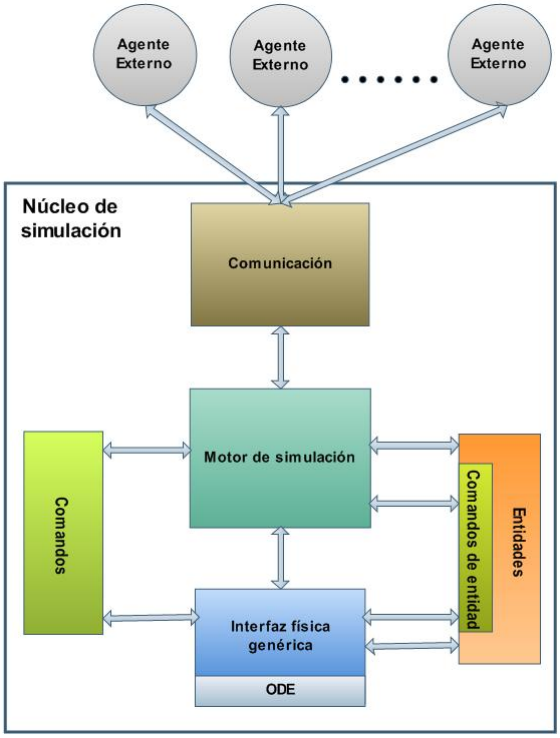

Figura 15: Arquitectura del sistema

A continuación se describirán cada uno de los módulos que interactúan en el sistema.

Agentes externos

Los agentes externos son procesos o hilos, remotos o locales, externos al núcleo de simulación, que mantienen algún tipo de diálogo con el mismo. La comunicación se realiza a través de mensajes con un formato predefinido, cuya especificación puede ser consultada en [AFM08].

Estos agentes pueden ejercer un control total sobre las distintas funcionalidades que brinda el sistema. Esto incluye la ejecución de todo tipo de comandos de control, como la creación de entidades, los cambios en el paso de simulación, entre otros. También es importante destacar que los agentes pueden registrarse en el sistema en dos modos diferentes: sincrónico o asincrónico. Los detalles acerca del comportamento del sistema frente a cada uno de estos modos de conexión se puede encontrar en la sección 3.4.

Un agente externo puede eventualmente tener diversos y variados propósitos. Se podría por ejemplo implementar un agente que actuara como una interfaz gráfica para desplegar utilizando algún método de visualización los datos recibidos acerca del estado de la simulación, de modo que se pueda observar los distintos objetos existentes y sus interacciones. También se podría implementar simplemente un observador de la simulación que almacenara la información que recibe en algún formato conveniente.

Otra alternativa sería implementar un agente cuya función principal fuera controlar uno o más robots. En una posible simulación con el objetivo de modelar un partido de fútbol de robots, el control de cada uno de estos robots se haría a través de un agente externo que lo

controle. Este agente externo al recibir información acerca del estado del mundo, le enviaría al núcleo de simulación un mensaje que se traduce en un comando de entidad para controlar cada una de las ruedas de la entidad de tipo robot. De esta forma se podrían controlar cada uno de los robots y llevar a cabo simulaciones de partidos de fútbol de robots. El sistema es lo suficientemente flexible, de forma tal que es posible que un solo agente externo pueda controlar un equipo entero de fútbol de robots, o como alternativa se podría implementar una agente para controlar cada robot en el campo de juego.

El agente al registrarse debe especificar su tipo. La diferencia en el comportamiento de agentes de distintos tipos registrados en el sistema radica en el tipo de información que desean recibir. A cada ciclo de iteración los agentes reciben mensajes conteniendo información acerca del estado del mundo. Estos mensajes, denominados mensajes de difusión, pueden contener toda la información acerca de los diferentes objetos y entidades que componen la simulación, número de iteración y el tiempo transcurrido desde el inicio de la simulación. Cada tipo de agente tiene un filtro asociado. Estos filtros son utilizados para seleccionar que información recibirá una agente externo en cada mensaje de difusión. De esta forma, dependiendo del tipo de agente implicado, se puede personalizar la información que un agente recibe a cada difusión.

En la actualidad existen dos tipos de agentes en el sistema, uno destinado a agentes que sean interfaces gráficas y otro destinado a agentes que controlen entidades de tipo robots.

Módulo Interfaz física genérica

Este módulo se encarga de manejar el modelo físico de la simulación. Es su responsabilidad proporcionar una interfaz para la creación, actualización y obtención de datos de los distintos objetos físicos representables en el sistema, y de las distintas características de la realidad modelada comunes a todos los objetos .

El conjunto de objetos que representan el modelo físico genérico mantenido por este componente, están estrechamente vinculados a un conjunto de objetos mantenidos por el componente que interactúa con el motor físico elegido para la implementación del sistema. Este motor físico podría ser sustituido por otro, desde que el componente que interactúe con él cumpla con la interfaz genérica que se propone. En este caso el motor físico utilizado es ODE y se encapsula el comportamiento referente al control del modelo realizado sobre dicha librería en un componente que implementa la interfaz propuesta, en este caso el Módulo ODE.

Es importante notar que el sistema propuesto soporta los objetos físicos necesarios para crear robots y estructuras utilizables para la creación de entidades físicas que se correspondan a las necesarias para satisfacer los requerimientos presentes en el alcance del sistema, sin embargo los tipos de objetos soportados se pueden extender fácilmente, dando lugar a la creación de estructuras físicas más complejas.

El módulo expone una interfaz mediante la cual se pueden realizar diversas acciones de control sobre el modelo físico. Adicionalmente expone una interfaz por cada objeto básico que compone el modelo simulado. Cada tipo de objeto tiene una interfaz con sus características propias. Los tipos de objetos pueden ser cuerpos o empalmes.

Los cuerpos soportados por la actual implementación del sistema son esferas, prismas, cilindros y un tipo especial de cilindros creados con el propósito específico de modelar ruedas. Se soportan solamente dos tipos de empalmes en la actual implementación, uno con el propósito de fijar las posiciones relativas de dos cuerpos, análogo al empalme Fixed descripto en la sección 2.2, y otro empalme denominado Wheel Joint cuyo propósito es conectar ruedas a cuerpos de robots, su funcionamiento es similar al empalme Hinge-2 descripto en la sección 2.2.

El motor físico subyacente podría ser fácilmente modificado o incluso con cambios menores podrían coexistir diversos motores físicos. Un posible uso de esta característica podría ser la ejecución de varias simulaciones en paralelo, con distintos motores físicos, para una eventual comparación de resultados.

Otra característica importante de este módulo es el manejo de materiales para modelar las interacciones entre cuerpos. Es posible crear materiales y controlar el comportamiento de la interacción entre distintos materiales. También a cada cuerpo creado se le debe asociar un material.

Módulo ODE

El motor físico utilizado para la implementación del sistema fue ODE. En este módulo se implementan todas las interfaces propuestas por la interfaz física genérica pero interactuando específicamente con esta librería física. Cada instancia de objeto existente en el sistema, cuerpos y empalmes, tiene un representante implementado en este módulo, creando y administrando los mismos mediante llamadas a la librería ODE.

Módulo Motor de simulación

En este componente se concentra la inteligencia necesaria para la coordinación y sincronización de los distintos módulos del sistema y agentes externos al mismo. Su ejecución es cíclica, y se termina solamente cuando se detiene el proceso en el cual se está ejecutando. Este ciclo básicamente ejecuta una serie de tareas que son responsables por realizar las acciones necesarias para garantizar que los agentes externos logren obtener el control sobre la simulación, todos los objetos que interactúan en ella, y todos sus parámetros modificables.

Básicamente, en un ciclo de iteración se realizan las siguientes tareas:

1. Verificar si hay mensajes nuevos recibidos por el módulo de comunicación
2. Decodificar dichos mensajes, identificar el comando correspondiente a cada mensaje y decodificar los parámetros de ejecución
3. Ejecutar dichos comandos
4. Avanzar el estado del mundo
5. Enviarle información acerca del nuevo estado del mundo a los agentes externos conectados

Módulo Comandos

Este modulo se encarga de administrar y controlar la ejecución de los diversos comandos de control existentes en el sistema. Cada vez que un agente externo envía un mensaje que se traduce en la ejecución de un comando, en este módulo se procede a la identificación del comando por su clave y se le delega el control de la ejecución del sistema en caso que corresponda. El comando por lo tanto tiene a su disposición todas las interfaces propuestas por el módulo "Motor de Simulación" y por el módulo "Interfaz física genérica".

En la figura 16 se muestra en un diagrama de actividad, el escenario típico del sistema en la ejecución de un comando. También se muestra la alternativa de ejecutar comandos de entidad en lugar de comandos de control.

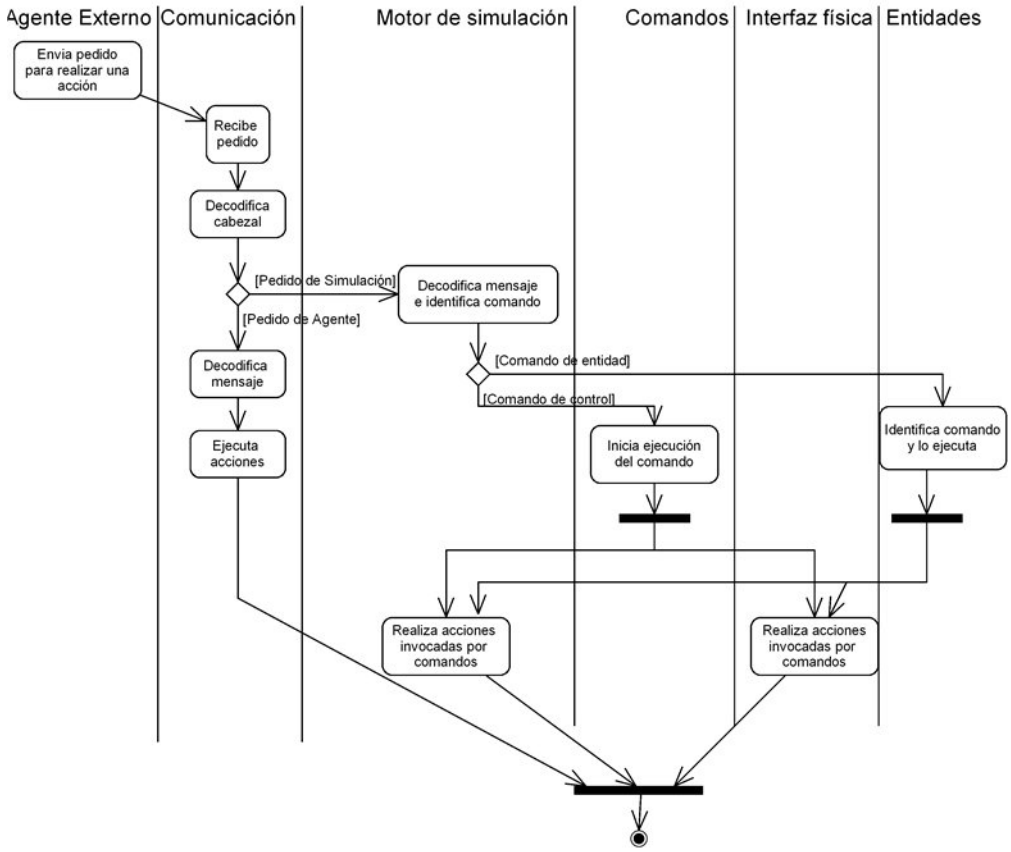

Figura 16: Diagrama de actividad de la ejecución de comandos

Módulo Entidades

Este módulo maneja y mantiene las distintas entidades existentes en la simulación. Provee una interfaz para la creación de las mismas y para efectuar acciones o consultas sobre entidades ya existentes. Para realizar estas acciones o consultas, las entidades se identifican a través de una clave única. Ejemplos de este tipo de acciones son cambiar la posición de una entidad, obtener información acerca de una entidad, entre otras.

Módulo Comandos de entidad

Cada tipo de entidad tiene un conjunto de comandos asociados a él. Estos comandos tienen total acceso a la entidad específica asociada a ellos, así como también a las distintas interfaces de los módulos "Motor de simulación" y "Interfaz física genérica". Este módulo encapsula estos comandos que resultan importantes por ejemplo para el control de robots a través de agentes externos.

Módulo Comunicación

Este módulo se ejecuta en un hilo diferente al que se ejecutan los demás módulos del núcleo de simulación. Maneja todo lo referente a la comunicación del sistema con los agentes que se conectan a él. Básicamente escucha en un puerto por posibles mensajes que le puedan llegar, y una vez que esto sucede, el mensaje es decodificado y si es necesario se ejecuta la operación correspondiente o se envía el mensaje al módulo "Motor de Simulación" en caso que le corresponda a este módulo procesarla.

Es importante en las tareas de sincronización de iteraciones con agentes, y para esto mantiene un estrecho vínculo con el módulo "Motor de Simulación", ya que debe notificarlo con la llegada de mensajes de sincronización.

También maneja la información que se le envía a los agentes externos, filtrando dicha información en función de que tipo de agente se le debe enviar información. Además, en cada ciclo de iteración verifica que agente debe recibir información y cuál no.

3.3.1. Procesos y protocolos de comunicación

El núcleo de simulación es un proceso con dos hilos. En uno se ejecuta el módulo de comunicación y en el otro se ejecutan los demás módulos que lo componen. La comunicación entre dichos hilos se realiza a través del protocolo UDP, con el envío de mensajes que son decodificados en el destino. Los agentes externos pueden eventualmente ejecutarse en distintos procesos y la comunicación de dichos agentes con el módulo "Comunicación" también es llevada a cabo mediante el protocolo UDP.

La comunicación entre el núcleo de simulación, a través del módulo "Comunicación", con los agentes se hace a través de mensajes con determinada estructura. Mayores explicaciones acerca de los mensajes intercambiados se puede encontrar en la sección 4.2.

3.4. Aspectos de Sincronización

En el contexto de competencias de robots, cada cierto intervalo de tiempo los agentes externos deberían controlar los motores de estos robots, especificándoles velocidades de acuerdo a su estrategia. Una posible manera de hacerlo es enviar mensajes que ejecutan esta acción sin ningún control de que éste cumpla su cometido en la iteración esperada, o sea, en cada ciclo de iteración. En este caso, luego de realizar una difusión, no se interrumpe la ejecución del ciclo mientras se esperan mensajes de los agentes. Este es un caso similar a lo que sucede en la vida real. Otra posible manera de hacerlo es a cada iteración, luego de una difusión, esperar por mensajes de los agentes que controlan robots de modo tal que se garantice que las velocidades indicadas a los motores se apliquen en la iteración esperada por dichos agentes. Este caso es similar al comportamiento del simulador de la FIRA, descripto en la sección 2.3.2, que ejecuta las estrategias de los equipos de robots de manera secuencial y bloqueante.

El componente del núcleo de simulación responsable por mantener el diálogo con los agentes externos es el módulo "Comunicación". En esta comunicación no se establece una conexión, por lo que no se garantiza que todos los mensajes enviados lleguen a su destino.

Los mensajes enviados por los agentes externos al núcleo de simulación pueden ser para registrarse en el sistema, realizar consultas, o efectuar acciones sobre la simulación, como crear entidades, controlar robots, mover cuerpos, etc. Los mensajes enviados desde el núcleo de simulación al agente pueden tener como objetivo el envío de respuestas con confirmaciones de acciones realizadas, respuestas a pedidos de información o también para enviar información acerca del estado actual del mundo.

3.4.1. Modos de conexión de agentes

Existen dos modos de conexión de agentes para la recepción de mensajes de difusión: sincrónico y asincrónico. Cuando un agente externo se registra en el sistema, este debe especificar si desea recibir mensajes de difusión y en caso que lo desee, el intervalo de tiempo mínimo entre distintos envíos de mensajes de difusión. Este intervalo es opcional, y en caso que no sea especificado se realiza el envío de mensajes de difusión siempre que exista nueva información disponible.

En la conexión asincrónica, a cada ciclo de ejecución o iteración, se realiza el envío de un mensaje de difusión al agente externo en caso que corresponda y no se espera una respuesta de manera sincrónica.

En la conexión sincrónica se espera como respuesta un mensaje de sincronización desde el agente externo. Los mensajes de difusión contienen el número de iteración al cual corresponden, y los mensajes de sincronización con una iteración especifican este parámetro.

3.4.2. Modos de ejecución de la simulación

También el sistema puede estar funcionando en cada uno de estos modos. La combinación entre el modo de ejecución del sistema y los modos de conexión de los agentes externos al sistema son los que efectivamente determinan el comportamiento del sistema en lo que se refiere a la comunicación con dichos agentes externos. En primer lugar se describirá el comportamiento del sistema en el modo de ejecución asincrónica y se describirá las variantes del mismo de acuerdo a las distintas conexiones de agentes externos. Luego se describirá el comportamiento del sistema en el modo de ejecución sincrónica de manera similar.

Modo de ejecución asincrónica Si el sistema se está ejecutando en el modo de ejecución asincrónica, a cada ciclo de iteración, luego de avanzar el estado de la simulación, se realiza una difusión del estado del mundo enviándole mensajes a todos los agentes externos conectados al sistema que corresponda. Esto significa que recibirán información de difusión aquellos agentes que cumplen las siguientes condiciones:

- Al momento de registrarse al sistema han indicado que desean recibir mensajes de difusión
- Al momento de registrarse al sistema no han especificado un intervalo de tiempo mínimo entre mensajes de difusión o si lo especificaron, dicho intervalo ya ha transcurrido desde la última difusión.

El sistema eventualmente puede tener agentes conectados en modo sincrónico y otros en modo asincrónico. El comportamiento del sistema es análogo al caso en que todos los agentes se encontraran conectados en modo asincrónico. Se realiza la difusión en caso que corresponda y se continúa la ejecución, avanzando el estado de la simulación sin esperar ningún tipo de mensajes de los agentes para hacerlo.

En caso que los agentes conectados de modo sincrónico envíen mensajes de sincronización con alguna iteración, estos simplemente son ignorados por el sistema que funciona en modo asincrónico. Ya que los mensajes de difusión contienen información acerca del modo de funcionamiento del simulador, estos agentes podrían fácilmente detectar que el sistema no está en el modo que ellos esperan y dejar de enviar mensajes de sincronización.

Modo de ejecución sincrónica Si el sistema se está ejecutando en el modo de ejecución sincrónica, a cada ciclo de iteración realiza una difusión del estado del mundo enviándole mensajes a todos los agentes externos conectados al sistema a los que corresponda realizar

dicho envío. Luego, si tiene agentes conectados en modo sincrónico, espera de ellos un mensaje de sincronización con la iteración cuya difusión fue enviada antes de avanzar el estado de la simulación. Sin embargo, espera por un tiempo máximo para ésto. En caso que en un intervalo de tiempo dado no reciba el mensaje de sincronización esperado de algún agente conectado en modo sincrónico, avanza la simulación. En caso que este mensaje llegue en otro momento luego de esto, este mensaje es simplemente descartado. También es importante destacar que solo se esperan mensajes de sincronización de aquellos agentes externos conectados de modo sincrónico a los cuales se le envió mensajes de difusión.

Pueden ocurrir varios escenarios que se pueden presentar durante la ejecución sincrónica del simulador.

Un posible escenario es que todos los agentes conectados al sistema se encuentren conectados en modo asincrónico. En este caso no tiene sentido esperar mensajes de sincronización de ningún agente, y el comportamiento del sistema es análogo al modo de ejecución asincrónica, o sea, se realiza la difusión y no se espera por mensajes de sincronización antes de avanzar la simulación.

Otro escenario es que todos los agentes conectados al sistema lo hagan en modo sincrónico. En este escenario luego de realizar la difusión se espera mensajes de sincronización de todos aquellos agentes a los cuales se les realizó el envío de mensajes de difusión. No se avanza el estado de la simulación mientras no se cumplan una de las siguientes dos condiciones:

- Se recibe un mensaje de sincronización con la iteración desde todos aquellos agentes a quienes se les envió de mensajes de difusión.
- Expira el tiempo estipulado de espera de mensajes de sincronización.

Un ejemplo en el que este escenario puede resultar útil, sería en el desarrollo de estrategias de control de un equipo de fútbol de robots para competir en la liga simulada de la FIRA, cuyo simulador utiliza comunicación sincrónica con las estrategias.

En un último escenario algunos agentes conectados al sistema están en modo sincrónico y otros en modo asincrónico. En este caso el comportamiento es similar al caso anterior, con la única diferencia que solamente se esperará por aquellos agentes conectados en modo sincrónico a los que se les envió mensajes de difusión antes de avanzar el estado de la simulación. Se ignoran los agentes en modo asincrónico al momento de verificar si se han recibido todos los mensajes de sincronización esperados.

Este escenario podría ser útil, por ejemplo en el caso que uno o varios de los agentes conectados al sistema tengan tiempos de respuesta mucho mayores que los demás. Estableciendo una conexión asincrónica con los agentes que tardan más en responder, garantiza que la simulación no se encuentre a la espera de dichos agentes antes de avanzar a cada iteración.

3.5. Funcionalidades

A continuación se describirán las funcionalidades más importantes presentes actualmente en el sistema. Todas ellas son ejecutadas mediante el envío de mensajes al núcleo de simulación. Estos mensajes se traducen a comandos, que son ejecutados y efectúan las acciones. El catálogo completo de funcionalidades y explicaciones más detalladas de su utilización y detalles se puede encontrar en [AFM08].

Registro de agente

Esta funcionalidad permite a los agentes registrarse en el sistema. Entre otras cosas debe indicar el tipo de agente, una clave única, modo de conexión, su puerto y dirección de red.

Información de agente

Esta funcionalidad permite a un agente obtener información acerca de otro agente conectado al sistema. La información que se obtiene son su puerto y dirección de red. El objetivo de esta funcionalidad es que distintos agentes externos puedan conectarse y comunicarse entre sí.

Creación de entidad

Esta funcionalidad es la más compleja del sistema. Su objetivo es la creación de entidades, de las cuales se debe especificar su tipo, estructura y propiedades. En la actual implementación del sistema existen solamente 3 tipos de entidades: robot, pelota y campo de juego.

Registro de materiales e interacciones

Esta funcionalidad permite registrar materiales y las interacciones entre dichos materiales. Una interacción es un conjunto de propiedades físicas que son tomadas en cuenta cuando dos materiales entran en contacto.

Especificar posición de entidad

Esta funcionalidad permite cambiar la posición de una entidad en cualquier momento del transcurso de la simulación. Se debe especificar las coordenadas en cada uno de los tres ejes cartesianos y la entidad que se desea posicionar allí.

Controlar motores de entidad

Esta funcionalidad permite controlar los motores asociados a ruedas de un robot. De esta manera se puede controlar el movimiento de los robots, especificando las velocidades de las ruedas .

Especificar modo de ajuste automático

Esta funcionalidad permite pasar un modo de avance de la simulación en el cual se ajustan automáticamente los parámetros de tamaño de paso de simulación y cantidad de pasos por iteración, de forma tal que el tiempo simulado transcurra a una velocidad similar al tiempo real. Este ajuste depende de la complejidad de la simulación realizada y del hardware sobre el cual se ejecuta el sistema.

Especificar modo de paso de tiempo

Esta funcionalidad permite que el sistema intercambie el modo de avance de la simulación. Existen dos posibles modos de avance: el modo paso a paso y el modo normal. En el modo paso a paso es necesario ejecutar un comando para que la simulación realice un paso en la simulación. Estos distintos modos permiten tener control sobre el avance de la simulación, de forma tal que se puede pausar la simulación en un momento dado.

Especificar bandera de paso de tiempo

Esta funcionalidad permite al simulador avanzar un paso en la simulación en caso que el avance de la simulación se esté realizando en modo paso a paso.

Especificar modo de comunicación

Esta funcionalidad permite cambiar el modo de ejecución del sistema a modo sincrónico o asincrónico.

Especificar tiempo de espera en comunicación sincrónica

Esta funcionalidad permite especificar el tiempo de espera de mensajes de sincronización de agentes externos conectados en modo sincrónico al sistema ejecutándose en modo sincrónico.

Sincronización con iteración

Esta funcionalidad permite que un agente externo se sincronice con una iteración que debe especificar.

Especificar tamaño del paso de simulación

Esta funcionalidad permite especificar el tamaño del paso de simulación, o sea, el período de tiempo en segundos que se va a avanzar la simulación cuando corresponda.

Especificación de modo rápido

Esta funcionalidad permite especificar si cada vez que el sistema va a avanzar la simulación lo va a hacer en modo normal o en modo rápido. El modo rápido es especialmente útil en realidades con mucha complejidad física, o sea, muchos cuerpos y empalmes que interactúan.

Especificación de pasos de tiempo por iteración

Esta funcionalidad permite especificar cuántos pasos se realizará en la simulación cada vez que sea necesario avanzar la misma. Esto significa que se pueden realizar varios pasos a cada ciclo de iteración en la ejecución del sistema.

Guardar simulación a archivo

Esta funcionalidad permite grabar el estado de la simulación en un archivo. Se almacenan la información completa acerca del estado del modelo físico de la simulación, todas las entidades que lo componen, y información de los diversos parámetros y variables de estado del sistema. No se almacena información acerca de los agentes externos que se encuentran conectados al sistema.

Cargar simulación desde archivo

Esta funcionalidad permite cargar una simulación y el estado del sistema desde un archivo.

4. Decisiones de diseño

En este capítulo se describirán algunas decisiones importantes de diseño efectuadas en el transcurso de este proyecto. Estas decisiones contemplan diversos aspectos importantes del sistema, y tomarlas de manera adecuada fue determinante para cumplir de manera correcta los objetivos del proyecto. Se analizan las alternativas que se plantearon, analizando las ventajas y desventajas de cada una, y se brindan argumentos que justifican las decisiones tomadas.

4.1. Modelo físico

Probablemente una de las decisiones más importantes fue determinar de que manera se haría el modelado físico de la realidad. Debido a la necesidad de contar con la posibilidad de modelar una amplia cantidad de posibles situaciones y realidades, emergió la necesidad de contar con un modelo físico basado en leyes físicas que respaldara la simulación.

Sin embargo, aunque pueda parecer que implementar un motor físico con estas características es sencillo, es una tarea extraordinariamente compleja. Por lo tanto, se planteó la necesidad de utilizar alguna de las librerías físicas o motores físicos existentes en la actualidad, en los cuales se pueden implementar modelos físicos realistas[HH07]. La complejidad en la utilización de estas librerías resultaba mucho menor que la necesaria para implementar un modelo físico realista de otra manera. Esto planteó un nuevo problema, tomar la decisión en cual de todas las librerías físicas se iría a utilizar entre todas las disponibles en la actualidad.

Decidir de manera adecuada qué motor físico utilizar fue importante. Se evaluaron varios de ellos y se tomó la decisión de utilizar ODE[ODE]. En primer lugar se implementó un prototipo utilizando esta librería. Lo que motivó a su realización fue la necesidad de mitigar riesgos técnicos y conocer a fondo dicha librería, de modo de tomar correctas decisiones en la determinación de la arquitectura del sistema y de detectar de forma temprana posibles elementos que indicarán una mala decisión de la librería elegida. Este prototipo fue descripto en la sección 2.4.

La librería física elegida cuenta con las características buscadas, y aunque algunos de los requerimientos del sistema podrían haber sido implementados con otro motor físico, presentaba algunas ventajas que superaban a las desventajas. En primer lugar la complejidad inherente a modelar realidades en motores físicos varía considerablemente de uno a otro. Entre los motores físicos estudiados, ODE resultó ser uno de los que más fácilmente se lograba modelar entornos físicos similares a los necesarios para cumplir con los objetivos de este proyecto. Sumado a una clara interfaz y una buena documentación, la usabilidad resultó un punto fuerte de esta librería.

Un requerimiento importante que debería contemplar la librería elegida era que se pudiesen modelar una amplia cantidad de realidades diferentes. Uno de los objetivos del proyecto era modelar distintos tipos de robots en distintos tipos de entorno y con comportamientos muy variables. En este aspecto la gran mayoría de las librerías presentan una amplia gama de posibilidades, excepto algunas aún en desarrollo, descartadas tempranamente.

Por otra parte, la estabilidad numérica de las simulaciones era un punto a tener en cuenta. Algunas librerías tienen problemas cuando las masas de los objetos difieren mucho, o cuando se desean modelar objetos pequeños que interactúan con objetos grandes. La librería elegida obtuvo resultados alentadores, y se lograron reproducir simulaciones con ciertas características que en otras librerías causaban serios problemas de estabilidad numérica. Una librería con muchas características deseables y ventajas, pero cuya utilización fue descartada debido a problemas de este tipo fue Newton Game Dynamics[NGD].

Otra importante característica que se buscaba era que la librería contara con un flexible manejo del paso del tiempo. Entre los requerimientos estaban poder poner en estado de pausa

la simulación, avanzar paso a paso y acelerar la velocidad de paso de tiempo. Básicamente todos las librerías evaluadas contaban con la posibilidad de mantener la simulación en pausa y avanzar paso a paso, pero la aceleración del paso del tiempo era una característica más compleja y se contaba con distintas alternativas para llevarla a cabo. Uno de los métodos que se podrían utilizar para lograr esto era a cada iteración avanzar la simulación una cantidad de tiempo mayor a la que debería hacerse en el paso de tiempo normal. Según [Smi07], esto siempre implica una pérdida de precisión y en caso de avanzar la simulación en un factor mucho mayor que el paso normal, implica caer en problemas de estabilidad numérica.

El avance más rápido que se puede lograr sin caer en estos problemas depende de la realidad modelada y de la librería física elegida. Pocas librerías evaluadas se mostraron lo suficientemente flexibles para cumplir este requerimiento de manera satisfactoria, entre ellas ODE. Otras maneras de ajustar el paso del tiempo para que la simulación se realice a mayor velocidad se discuten en la sección 4.7.

Además, se deseaba que el sistema fuera portable, o que por lo menos pudiera ser compilado y ejecutado en diversos sistemas sin sufrir mayores modificaciones. Para esto debería utilizar librerías portables. Entre las librerías evaluadas algunas fueron descartadas debido a este motivo. En particular, la librería Ageia PhysX [AGEIA] fue descartada debido a que aunque cumplía con todos los requerimientos deseados y sus características parecían muy prometedoras, solo contaba con una versión para plataformas Windows. En cambio, ODE es una librería portable, de código abierto, que puede ser transportada para otros sistemas.

4.2. Mensajes

Para lograr comunicar agentes que eventualmente estuvieran implementados en diferentes lenguajes se tomó la decisión de utilizar sockets en la comunicación. El protocolo elegido fue UDP, ya que no se consideró necesario el establecimiento de conexiones entre los agentes y el núcleo de simulación. Como el objetivo del sistema es comportarse de manera tal que el comportamiento se asemejara a la realidad, se consideró que la velocidad de llegada de los mensajes era un factor crucial, y que la pérdida de algunos paquetes no comprometería la posibilidad de alcanzar los objetivos deseados.

A medida que aumentan los agentes conectados al sistema, es de esperar que el tráfico de mensajes aumente considerablemente. Por esto minimizar la cantidad de mensajes intercambiados sin menoscabar la posibilidad de realizar las acciones deseadas era una cuestión importante. El protocolo de comunicación entre los agentes brinda la posibilidad de efectuar diversas acciones especificándolas en un solo mensaje. El procesamiento de los mensajes por parte del núcleo de simulación constituye un cuello de botella, por lo que optimizar este elemento es fundamental para mejorar el rendimiento del sistema. Las acciones sobre el núcleo de simulación son codificadas en comandos. Cada mensaje puede contener diversos comandos que se ejecutan en orden.

En un mensaje UDP recibido por el núcleo de simulación pueden estar codificados diversos mensajes de tipo “Request Message”, que a su vez pueden ser de tipo “Agent Message” o “Simulation Message”. El primero de estos tipos corresponde a mensajes que contienen un pedido para realizar alguna operación relacionada con agentes, como el registro de los mismos o el pedido de información de otro agente. El segundo encapsula mensajes que tienen como objetivo realizar alguna acción sobre la simulación. Estos mensajes son los denominados “Request” o pedidos y pueden ser de dos tipos: “Entity Request” y “Control Request”.

Cada pedido especifica el comando que debe ser ejecutado y encapsula los datos necesarios para llevar a cabo dicho comando. Los pedidos de tipo entidad o ”Entity Request” se traducen a comandos asociados a una entidad especifica en la simulación, de forma tal de realizar operaciones específicas sobre esa entidad, como por ejemplo controlar motores, mover entidades, etc. Los pedidos de control o “Control Request” no son tan específicos y

se traducen a comandos que realizan acciones generales sobre la simulación, no asociados a ninguna entidad que ya exista. Ejemplos de estos pedidos son cambios en las propiedades físicas generales de la simulación, control del paso de tiempo, guardar la simulación, etc. Cabe destacar que la creación de una entidad también es un pedido de este tipo ya que la misma no existe antes de la ejecución de este comando. A continuación se muestra un diagrama que ilustra esta realidad.

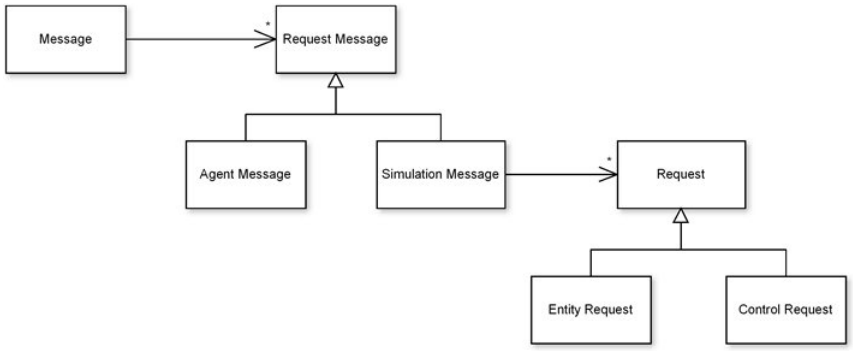

Figura 17: Mensajes

Los agentes conectados al sistema pueden registrarse en dos modos: sincrónico y asincrónico. Por otra parte, el sistema también puede estar ejecutándose en alguno de estos dos modos. A cada ciclo de iteración, el sistema realiza una difusión con información acerca del estado de la simulación, enviando esta información a los agentes que corresponda. A continuación, su manera de actuar varía según cual sea su modo de ejecución.

El modo sincrónico implica que a cada ciclo de iteración, el simulador espere una notificación de cada agente conectado a el en modo sincrónico para avanzar la simulación. En caso que uno o más de estas notificaciones no lleguen, se espera un lapso de tiempo configurable y se avanza. Las notificaciones se realizan a través del envío de un mensaje que lleva a la ejecución de un comando responsable de sincronizar a un agente con una iteración. En caso que en determinada iteración, todos los agentes conectados en modo sincrónico estén sincronizados con la misma, se procede a avanzar hacia la siguiente iteración. Si no existen agentes conectados en modo sincrónico, el sistema actúa de manera análoga a la ejecución asincrónica.

Si el simulador está funcionando en modo asincrónico, no se realiza ninguna sincronización con los agentes conectados al mismo, sin importar si algunos agentes se registraron en modo sincrónico. Se avanza la simulación tan pronto como sea posible, o sea, luego que se hayan terminado todas las tareas correspondientes a esa iteración.

4.3. Colisiones

El motor físico utilizado modela las colisiones creando un conjunto de puntos de contacto cuando detecta alguna. A estos puntos de contactos se le debe especificar una serie de parámetros que aplican restricciones al movimiento de los cuerpos, obteniendo de esta manera la resolución de la colisión. En el contexto del sistema, con el objetivo de contribuir a la usabilidad del sistema y aportarle carga semántica, a cada cuerpo físico de la simulación se le asocia un material. A cada par de materiales se les asocia una interacción. Cada interacción tiene asociado una serie de parámetros que son los que en definitiva regulan las colisiones.

Al momento de crear entidades se debe especificar que cuerpos la componen. A cada cuerpo se le puede especificar un material. En caso que no se le especifique un material a un cuerpo, se le especificará uno por defecto. Además existe un comando que permite crear materiales y especificar interacciones. Este debería ser ejecutado anteriormente a la creación de las entidades que utilizan los materiales que este comando registra. Cuando se detecta que dos cuerpos están colisionando, se busca los materiales que los componen y se busca en el sistema una interacción que regule el comportamiento de la colisión entre dichos materiales. En caso que se encuentre la interacción, se especifican los parámetros correspondientes para resolver la colisión. En caso contrario se asigna una interacción por defecto a la colisión y se procede de manera análoga.

Los parámetros que componen las interacciones se detallan a continuación.

- Bounciness: Cuando dos cuerpos están conectados por empalmes, el empalme implica que dichos cuerpos tengan cierta posición relativa entre sí. Durante la simulación puede ocurrir que dichos cuerpos se muevan de manera tal que esta posición relativa no sea la adecuada. Cuando esto ocurre, a cada paso de simulación se aplica una fuerza correctiva con el objetivo de llevar a los cuerpos a una posición correcta. Esta fuerza es controlada por este parámetro, que puede tener un valor entre 0 y 1. Especifica que proporción del error se corregirá en la siguiente iteración. Su valor recomendado es entre 0.1 y 0.8. Cuando dos objetos colisionan, regula la fuerza que se ejerce a los cuerpos para que estos no se atraviesen.

- Hardness: Cuando dos cuerpos colisionan, esto implica que se deben cumplir algunas restricciones en cuanto al movimiento de ambos. Este parámetro denota que tan rígidas son estas restricciones. Su valor varía de 0 a 1. Utilizar valores menores implica restricciones más rígidas, contribuyendo para la estabilidad numérica y la velocidad de la simulación.

- Friction: Coeficiente de fricción aplicado al punto de contacto entre dos cuerpos, afectando a los mismos de la manera determinada por las leyes físicas. Su rango de posibles valores va desde 0 hasta el infinito.

- Slip 1 y Slip 2: Estos parámetros implican que dos superficies en contacto se deslicen a una velocidad relativa proporcional a la fuerza aplicada tangencialmente a la superficie. Son particularmente útiles para modelar interacciones entre ruedas y la superficie. Si se especifica la fuerza de rozamiento entre dichos cuerpos como infinita, la rueda girara perfectamente sobre si misma avanzando sobre el piso, sin deslizar hacia ninguna dirección. Sin embargo si se le aplica una fuerza perpendicular a su dirección podría desearse que la rueda se deslizara sobre la superficie en esa dirección. Sin embargo, como se especificó una fuerza de rozamiento infinita, esto no debería pasar. Para contemplar este tipo de casos se utilizan estos parámetros, uno en cada dirección del movimiento relativo entre los cuerpos.

Estos parámetros son utilizados por el motor físico de la simulación para resolver como afectan las colisiones al comportamiento de los cuerpos.

La decisión de crear el concepto de materiales y manejas las colisiones de los cuerpos a través de sus interacciones fue tomada con el objetivo de facilitar la tarea de ajustar el comportamiento de los puntos de contacto entre los cuerpos. De esta manera se logra agrupar cuerpos con características similares asociándolos a un mismo material y ajustando las interacciones con todos los demás materiales se logra que el comportamiento se ajuste a lo pretendido. Este enfoque presenta un fuerte carga semántica, ya que resulta más intuitivo el manejo de los materiales e interacciones entre ellos que especificar las interacciones entre cada par de cuerpos posible en el sistema.

Es importante destacar que existe un material por defecto, que se le asigna a todo cuerpo que no tenga un material asociado. También existe una interacción por defecto que es asignada a todos los puntos de contacto en colisiones en las que intervienen materiales cuya interacción no fue registrada.

4.4. Interfaz gráfica

Se pueden plantear dos objetivos básicos para la interfaz gráfica del sistema:

1. Visualizar la simulación física propiamente dicha, viendo los distintos objetos que interactúan en la misma.
2. Poder controlar la simulación, efectuando acciones que permitan controlar diversos factores de la misma, como sus propiedades físicas, mover entidades o manipular el paso de tiempo.

Ambos objetivos planteaban problemas totalmente diferentes y se consideró apropiado resolver ambos problemas con enfoques completamente diferentes.

Para el primer problema se podrían considerar dos alternativas:

- Crear un agente que recibiera información del núcleo de simulación, decodificarla y dibujar la pantalla basado en esta información.
- Implementar una interfaz gráfica acoplada al núcleo del simulador, que directamente dibujara la pantalla con la información del motor físico.

La primera alternativa tiene como ventaja que el núcleo de simulación no debe realizar tareas que no sean las referentes a la implementación el modelo físico de la simulación y el procesamiento de mensajes que actúan sobre él. No se gastarían recursos computacionales dibujando los objetos presentes en el modelo físico de la simulación en la interfaz gráfica, ya que se podría enviar toda la información y un agente externo, eventualmente remoto, podría dibujar los objetos en su interfaz gráfica. Sin embargo existen una serie de desventajas muy notorias en este caso. En primer lugar se introduce un tiempo de retardo importante entre el envío de la información y su posterior dibujado. La decodificación de los mensajes recibidos puede ser una pesada carga en caso que existan muchos objetos que dibujar, por lo que el agente externo que dibuja la pantalla debería tener mucha capacidad de procesamiento en caso que desee representar los objetos de manera relativamente actualizada. En el transcurso de este proyecto se hicieron algunas pruebas de este caso sin resultados positivos. Los tiempos dedicados a la decodificación de los mensajes de difusión y su posterior representación en la interfaz gráfica conducían a tiempos de retardo considerables.

La segunda alternativa le agrega tareas al núcleo de simulación, ya que el mismo a cada iteración debe dibujar la pantalla con la información actualizada. Sin embargo, este enfoque presenta diversas ventajas. En primer lugar no existe tiempo de retardo, siempre se dibuja la pantalla con la última información disponible, o sea, el estado real del modelo físico. Por otra parte, la decodificación de la información se hace dentro del núcleo de simulación, y no se avanza la simulación mientras esto no se termine, por lo tanto no debe considerarse el problema del Productor-Consumidor como en el caso anterior. En este proyecto se tomó este enfoque con muy buenos resultados. Sin embargo existe una pequeña variante, el sistema se puede ejecutar en modo con interfaz gráfica o sin ella. De este modo, en caso de que sea conveniente, el sistema se ejecuta sin realizar todo el procesamiento referente al dibujado de los objetos, optimizando su rendimiento. Esto puede resultar útil en escenarios en los cuales no sea necesario observar en la interfaz gráfica el transcurso de la simulación, como podría ser entrenamientos de estrategias mediante alguna técnica de aprendizaje automático.

Se evaluaron diversos motores gráficos para su utilización para dibujar los objetos de la simulación. Sin embargo, por razones de rendimiento se optó por la implementación directa de las primitivas de dibujado utilizando OpenGl. De esta manera se optimizó al máximo la utilización de recursos, ya que no era necesario contar con gráficos complejos para realizar la representación gráfica de los objetos físicos de la simulación.

Una captura de pantalla de dicha interfaz gráfica durante la ejecución de una simulación se muestra en la figura 18. En la misma se pueden observar tres entidades de tipo robot, una entidad de tipo pelota y una entidad de tipo campo de juego.

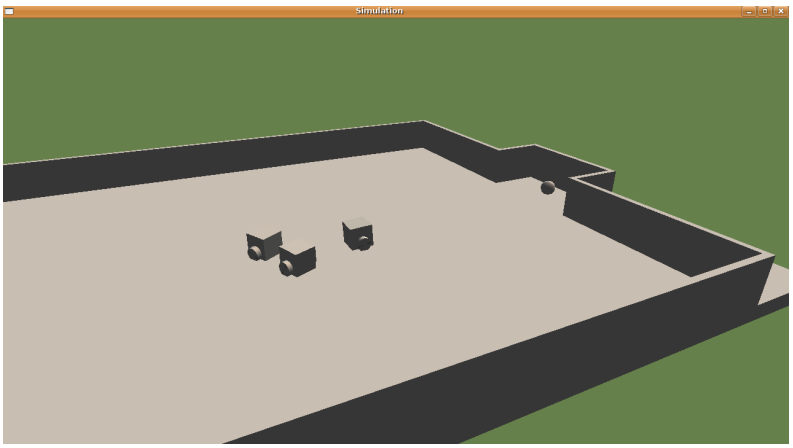

Figura 18: Simulación en interfaz gráfica

El segundo objetivo que debería ser cumplido por la interfaz gráfica sería poder realizar acciones sobre la simulación. Nuevamente para esto existían dos enfoques:

- Implementar en el núcleo de simulación un componente que actuara de manera interactiva y recibiera datos de entrada, por ejemplo del teclado.
- Crear un agente que le enviara mensajes al núcleo de simulación para llevar a cabo estos pedidos.

En el primer enfoque el tiempo de retardo es muy pequeño, ya que se censaría las entradas en la búsqueda de comandos a cada iteración. Sin embargo esto le agregaría tareas al núcleo de simulación, lo que iría a afectar su rendimiento. Además, el hecho que comandos ingresados a través de la interfaz de usuario tengan cierto tiempo de retardo, no es algo preocupante. Este enfoque fue descartado.

El segundo enfoque tiene como ventaja de favorecer la modularización del sistema. Un agente externo interactúa con el usuario y en caso de que el usuario desee ejecutar un comando, dicho agente le manda el mensaje correspondiente al núcleo de simulación. Se creyó que este enfoque era el más conveniente y dio lugar a la implementación de una aplicación en lenguaje Java, la cual cuenta con una interfaz gráfica para el envío de comandos. Esta aplicación se describe con mayor detalle en el Anexo 2. Esta aplicación cuenta con diversas funcionalidades que permiten ejecutar comandos con el objetivo de realizar acciones sobre la simulación.

4.5. Extensibilidad

La arquitectura del sistema está fuertemente enfocada en el requerimiento de la extensibilidad. El sistema es altamente extensible, tanto en las funcionalidades que presenta como en las distintas estructuras que utiliza. En el marco de este documento se presentan algunas maneras de extender el sistema. Para un explicación más detallada de las formas de extender el sistema se puede consultar el Manual Técnico[AFM08].

4.5.1. Modelo físico

El módulo "Interfaz Física Genérica" presenta una serie de clases abstractas o interfaces para su utilización. Estas interfaces son implementadas por clases que interactúan con un motor físico concreto. Se puede fácilmente cambiar el motor físico utilizado sustituyendo dichas clases concretas por otras que interactúen con otro motor físico. Básicamente, existen interfaces que representan cada uno de los tipos básicos de objetos soportados y para el controlador general del modelo físico. Los objetos básicos soportados son los diferentes tipos de cuerpos y empalmes que se pueden modelar actualmente en el sistema. Dichos objetos pueden extenderse de modo de soportar otros tipos de cuerpos y empalmes. Se debe crear una interfaz de dicho objeto que se acople a la interfaz genérica presentada e implementar dicha interfaz utilizando una clase concreta que interactúe con el motor físico. Se podría de esta manera crear nuevos tipos de cuerpos, como cuerpos formados por la unión de triángulos, o nuevos empalmes, como el "Hinge" o "Ball and Socket". Estos empalmes y cuerpos de ejemplo son soportados por el motor físico, pero no están representados en la interfaz física genérica propuesta. De esta manera se podría mapear de forma completa todos los cuerpos y empalmes soportados por el motor físico.

4.5.2. Entidades

El sistema provee una interfaz de la cual deben heredar todas las entidades del sistema. Estas deben cumplir con cierta estructura básica, pero existe gran flexibilidad en cuanto a las funcionalidades que puede encapsular una entidad. En particular, sería interesante contar con una entidad sin ningún tipo de restricciones estructurales con algunos comandos básicos, de modo de que pueda ser utilizado para cualquier tipo de entidad.

4.5.3. Comandos

La existencia de comandos le aporta una gran flexibilidad al sistema, y gran parte de las extensiones y nuevas funcionalidades que se podrían considerar, pueden ser resueltas simplemente implementando un nuevo comando. Al agregar una nueva funcionalidad se tienen dos opciones: resolverla utilizando un comando de control o resolverla utilizando un comando asociado a alguna entidad. Simplemente se debe crear uno de estos comandos, implementando la interfaz de comandos y asociarlo al controlador de comandos correspondiente. Existe un controlador de comandos de control y otro controlador de comandos de entidades. Para ejecutar correctamente un comando, debe decodificarse un pedido encapsulado en un mensaje, y decodificar los distintos parámetros o tipos de datos utilizados para la invocación del comando.

4.5.4. Tipos de agentes y filtros

Cada tipo de agente que se conecta al sistema, tiene asociado un filtro, que en el momento de enviarle información acerca del estado del mundo determina que información se trasmite y cual no. Esto es especialmente útil para ocultar determinada información, agregarle ruido

a los datos, cambiar las unidades de las magnitudes, etc. Cambiar el tipo de filtro asociado a un tipo de entidad es realmente sencillo, así como también crear tipos de agentes nuevos con el filtro que se desee. Además, es necesario realizar modificaciones en el decodificador de este módulo para que contemple los nuevos tipos de agente.

4.6. Performance

Debido a que una realidad simulada debe asemejarse a un sistema en tiempo real, la performance es un factor determinante al evaluar el sistema. Para esto se tomaron una serie de decisiones para incrementar el rendimiento del sistema.

En primer lugar, se resolvió implementar el simulador utilizando el lenguaje C++. Entre otras ventajas, este es uno de los lenguajes orientados a objetos con mejor rendimiento en términos de tiempo computacionales[MCLAN, LAN]. También todas las librerías utilizadas, incluyendo el motor físico, están desarrolladas en C o C++.

Adicionalmente se resolvió que el protocolo de comunicación sería UDP, o sea, envío de datagramas sin conexión. La alternativa descartada fue utilizar el protocolo TCP, basado en conexiones. Se consideró que la prioridad es que los mensajes se entreguen lo más rápido posible a sus destinatarios, y no que todos los mensajes sean entregados o que se garantizara el correcto orden de los mismos.

Considerando el hecho que buena parte del tiempo de ejecución del sistema se utiliza para decodificar y ejecutar comandos, la mayoría de los esfuerzos fueron centrados en la optimización de aquellos comandos que son ejecutados más a menudo. De esta forma se buscó que aquellos comandos que se ejecutan a cada iteración o varias veces por iteración, tengan un comportamiento muy eficiente. Algunos ejemplos de este tipo de comandos son el comando responsable de controlar motores de robots, o de sincronización con la iteración.

4.7. Ajuste del paso del tiempo

El sistema presenta una serie de mecanismos para ajustar el paso del tiempo del simulador al comportamiento deseado, tanto si se quiere que este se asemeje al tiempo real como si lo que se desea es avanzar la velocidad de simulación a mayor velocidad. Entre los factores que influyen en este funcionamiento están la complejidad de modelo físico simulado, tráfico de mensajes entre agentes externos y el sistema, y el hardware sobre el cual se está ejecutando. Dada una simulación ya ajustada, si alguno de estos dos factores sufre variaciones importantes, es conveniente analizar si el comportamiento de la simulación continúa siendo el deseado.

A continuación se explicarán algunas variables que pueden ser ajustadas y como afecta cada una de ellas al sistema. Todas ellas pueden ser modificadas en cualquier momento de la simulación.

- Tamaño del paso de simulación: A cada ciclo de iteración se avanza el estado del mundo en una cierta cantidad de segundos. Esta variable refleja la cantidad de segundos que se avanza. El rango recomendado se encuentra entre los 0.010 y 0.070 segundos. Cuanto mayor es el valor de esta variable, más rápido se avanza, pero en contrapartida se pierde precisión numérica. Presenta mayor influencia cuando el cuello de botella en lo que se refiere al avance de la simulación se encuentra en la complejidad del modelo físico.

- Modo de avance: Cada vez que se va a avanzar la simulación, se puede hacerlo en dos posibles modos: normal o rápido. Este modo influye en la resolución numérica de los cálculos necesarios para avanzar el modelo físico. En el modo rápido se gana velocidad de cálculo pero se pierde precisión. Es particularmente útil en simulaciones con grandes

cantidades de entidades que interactúan entre sí. A medida que aumenta la cantidad de objetos en la simulación, más notoria se hace la diferencia entre los dos modos. En contrapartida, al igual que el caso anterior, se pierde precisión numérica.

- Cantidad de pasos por iteración: Cuando la cantidad de agentes conectados al sistema es muy grande, el tráfico de mensajes puede eventualmente ser un cuello de botella, ya que el sistema podría pasar la mayor parte de su tiempo decodificando mensajes y ejecutando comandos. Esta variable especifica cuantos pasos se avanza la simulación física a cada paso de simulación del simulador. Esto implica que el intercambio de mensajes no se hará a cada paso en el modelo físico, sino que se darán tantos pasos como se especifique antes de proceder a la decodificación de mensajes nuevos, ejecución de comandos y broadcast de información. Es probablemente la manera más práctica y directa de avanzar la simulación a mayor velocidad que la velocidad normal. Aumentar este parámetro no implica pérdida de precisión numérica, solamente implica que desde el punto de vista de los agentes conectados al sistema, el tiempo simulado entre dos recepciones de información de difusión va a ser mayor, y sus comandos no se ejecutarán a cada ciclo de iteración del modelo físico. Por lo tanto, desde el punto de vista de dichos agentes, el paso de simulación es de (Tamaño del paso de simulación)*(Cantidad de pasos por iteración) segundos.

4.7.1. Ajuste automático

Existe una funcionalidad que permite que el avance a cada iteración se realice de forma tal que el tiempo simulado y el tiempo real transcurran aproximadamente a la misma velocidad. Para esto el sistema dispone de un modo de avance en el cual se realiza el ajuste automático del tamaño del paso de simulación y la cantidad de pasos por iteración. Este ajuste o corrección se realiza tomando como referencia el tiempo promedio por iteración a lo largo de un número determinado de iteraciones, ajustando los parámetros tomando como base a dicho dato.

Se aplica una heurística que le asigna valores a los parámetros ajustados, cumpliendo con la restricción que el tamaño del paso de simulación sea menor a un determinado valor, para no comprometer la precisión y estabilidad matemática del modelo.

Sea T el tiempo medio por iteración desde la última correción, t un valor decimal que representa el tamaño del paso de simulación y c un valor entero que representa la cantidad de pasos por iteración. Dado k un valor máximo para c, se debe encontrar valores para c y t de forma tal que $c \times t = T$, con $c \leq k$ y $t \geq 1$. Se debe elegir el menor valor de t que cumpla las restricciones, de modo tal que la cantidad de pasos por iteración sea la menor posible.

El tiempo promedio por iteración se puede ver afectado por diversos factores, como por ejemplo un aumento en el número de agentes conectados al núcleo de simulación, un aumento en la complejidad del modelo físico a partir de la creación de nuevas entidades. En el caso que el tiempo medio por iteración aumente considerablemente, también aumentará la cantidad de pasos por iteración, ya que cuando$T \to \infty$ se cumple que $c \to \infty$.

A cada iteración, el núcleo de simulación ejecuta todos los comandos que han llegado desde la última iteración. En caso que la cantidad de pasos por iteración aumente, se avanzará a cada iteración un mayor número de veces la simulación sobre el modelo físico. Esto provoca que se pierda contacto entre los agentes externos y la simulación, en lo que se refiere a la posibilidad de intervenir en la misma a través de la ejecución de comandos.

5. Experimentos y resultados

En este capítulo se relatan experimentos realizados y sus resultados. Se relatan algunos experimentos realizados en el período de verificación del sistema, más específicamente las pruebas no funcionales realizadas. Finalmente se realiza el modelado de un robot real, explicando el proceso seguido para construir la simulación correctamente.

5.1. Verificación

En la etapa de verificación del sistema se realizaron una serie de pruebas para detectar errores e identificar posibles cuellos de botella en lo que se refiere al rendimiento del sistema. Se realizaron diversas pruebas tanto funcionales como no funcionales.

El equipo utilizado para realizar las pruebas contaba con un procesador AMD Sempron 2200 y 256 MB de memoria RAM, con sistema operativo Linux Ubuntu 6.06.

Las pruebas funcionales consistieron en la ejecución de comandos que correspondían a las distintas funcionalidades del sistema, analizando el correcto comportamiento del sistema ante la ejecución de cada uno de ellos. Se ejecutaron estos comandos en distintas combinaciones que se consideraron relevantes.

Las pruebas no funcionales consistieron en someter al sistema a una gran carga de trabajo. Esto se hizo aumentando progresivamente la cantidad de agentes conectados al sistema y la complejidad del modelo físico. Se generó un flujo de mensajes que aumentaba progresivamente, y se midió como se deterioraba el rendimiento del sistema a lo largo de este proceso. La métrica que se utilizó para medir la respuesta del sistema fue el tiempo medio necesario para realizar todas las tareas necesarias en una iteración completa. Esto implica recibir mensajes, decodificarlos, ejecutar los comandos que esos mensajes encapsulaban y avanzar el estado del mundo.

Se obtuvieron datos para diversos escenarios y del análisis de estos datos emerge el hecho que el aumento de agentes externos conectados al sistema afecta al rendimiento del sistema en una medida mucho mayor de lo que lo hace el aumento de la complejidad de la simulación. También se puede obervar que el peso relativo de ejecutar al sistema con interfaz gráfica en la cual observar la simulación es cada vez más bajo a medida que aumenta la cantidad de agentes.

Se detallarán los distintos casos de prueba no funcionales realizados y los resultados obtenidos. Una descripción más detallada de las pruebas realizadas se puede encontrar en el Documento de Verificación[AFV08].

Caso 1

En este caso de prueba se conectaron al simulador 5 agentes cada uno de los cuales crea una entidad de tipo robot y alguno de ellos crea una entidad de tipo pelota y otra de tipo campo de juego. Todos los agentes ejecutan comandos para controlar motores de uno de los robots y envían esta información en cada iteración.

Caso 2

En este caso de prueba es análogo al anterior pero se ejecuta por un largo periodo de tiempo.

Caso 3

En este caso de prueba se conectó al simulador 1 agente que crea 10 entidades de tipo robot, una entidad de tipo pelota y otra de tipo campo de juego. Este agente ejecuta comandos

para controlar motores de todos los robots y envía esta información en cada iteración.

Caso 4

En este caso de prueba se conectaron al simulador 5 agentes cada uno de los cuales crea dos entidades de tipo robot y alguno de ellos crea una entidad de tipo pelota y otra de tipo campo de juego. Todos los agentes ejecutan comandos para controlar motores de dos de los robots y envían esta información en cada iteración.

Caso 5

En este caso de prueba se conectan al simulador 10 agentes cada uno de los cuales crea una entidad de tipo robot y alguno de ellos crea una entidad de tipo pelota y otra de tipo campo de juego. Todos los agentes ejecutan comandos para controlar motores de uno de los robots y envían esta información en cada iteración.

Caso 6

En este caso de prueba se conectaron al simulador 20 agentes cada uno de los cuales crea una entidad de tipo robot y alguno de ellos crea una entidad de tipo pelota y otra de tipo campo de juego. Todos los agentes ejecutan comandos para controlar motores de uno de los robots y envían esta información en cada iteración.

Resultados

En la ejecución de estos casos de prueba se consideró el tiempo medio por iteración en las primeras 1000 iteraciones. La única excepción es el segundo caso de prueba que considera el tiempo medio por iteración entre las iteraciones 3000 y 4000.

El cuadro 1 muestra los resultados para cada caso de prueba en dos escenarios: la ejecución del sistema con interfaz gráfica y sin ella. También se muestra el porcentaje del tiempo de procesamiento que sería necesario para representar la simulación en la interfaz gráfica.

Caso	Tiempo medio por iteración con GUI	Tiempo medio por iteración sin GUI	% GUI
Caso 1	22.3 ms	13.1 ms	41.2 %
Caso 2	22.8 ms	13.2 ms	42.1 %
Caso 3	26.9 ms	15.1 ms	43.9 %
Caso 4	37.9 ms	26.0 ms	31.4 %
Caso 5	57.0 ms	48.2 ms	15.4 %
Caso 6	135.5 ms	124.1 ms	8.4 %

Cuadro 1: Pruebas no funcionales, comparación de resultados

5.2. Modelado de un caso tipo

En esta sección se realizará el modelado de una simulación a partir de un modelo de referencia real, y se ajustará el comportamiento de la simulación para que se corresponda de manera lo más cercana posible al caso real.

5.2.1. Datos reales

Para este experimento se modelarán un robot SumBot, construido por el grupo MINA de la Facultad de Ingeniería de la Universidad de la República. Se verificará que el comportamiento del mismo sea similar a lo observado en la realidad. Se modelará también un entorno en el cual se creará un campo de juego de proporciones análogas al utilizado en la categoría Simurosot de fútbol de robots de la FIRA.

5.2.2. Datos experimentales

El Sumbot es un robot con dos ruedas a ambos extremos de su chasis, simétricas con respecto a éste. Básicamente se pueden observar 3 tipos de objetos:

- Chasis: Prisma con las siguientes características:
  - Altura: 0.13 metros
  - Largo: 0.11 metros
  - Ancho: 0.11 metros
  - Masa 1.1 kg
- Ruedas: Son cilindros con las siguientes características:
  - Radio. 0.024 metros
  - Espesor: 0.015 metros
  - Masa: 0.045 kg
- Patines: Debajo de los robots hay patines adheridos al piso del mismo, estos se modelan utilizando esferas unidas al chasis.
  - Radio: 0.004 metros
  - Masa: 0.01 kg

Estos robots se controlan especificando velocidades a cada una de sus ruedas. El rango de velocidades que se utiliza son los números enteros de -5 a 5 inclusive.

Se tomaron diversas mediciones experimentales de los robots reales. Estas mediciones y sus valores se detallan a continuación. En todas las mediciones y comparaciones se considera un comportamiento simétrico de los robots, es decir, el par de velocidades 3,3 debería ser análogo al par -3,-3.

Línea recta

Se tomaron mediciones del tiempo necesario para que los robots recorrieran 0,50 metros en línea recta, con velocidades iguales en ambas ruedas, recorriendo las distintas velocidades posibles. Los valores tomados se muestran a continuación.

Velocidades (izquierda, derecha)	Tiempo en segundos
1,1	11.81
2,2	9.5
3,3	7.52
4,4	6.23
5,5	4.11

Cuadro 2: Modelado de un caso real. Recorrida de 0,50 metros

Giros

Se tomaron mediciones del tiempo transcurrido entre el inicio y el final de un giro de 360º utilizando distintas combinaciones de velocidades en ambas ruedas, de modo tal que exista alguna diferencia entre la velocidad de la rueda izquierda y derecha. Además en los casos que el robot no giró sobre su propio eje, se midió el diámetro de giro. En un par de casos se consideró que el diámetro de giro era despreciable en el ajuste que se deseaba realizar, ya que era muy pequeño y acarrearía errores en su medición.

Velocidades (izquierda, derecha)	Tiempo en segundos	Diam. de giro en mts.
5,4	19.16	0.79
5,3	13.54	0.42
5,2	11.21	0.34
5,1	10.4	0.26
5,0	6.5	Despreciable
5,-5	3.33	Sobre el eje
4,-4	4.88	Sobre el eje
3,-3	6.1	Sobre el eje
2,-2	7.6	Sobre el eje
1,-1	10.01	Sobre el eje
3,1	9.34	Despreciable

Cuadro 3: Modelado de un caso real. Giros

Procedimiento seguido

Para realizar el modelado del robot se procedió de la siguiente forma:

1. Crear los distintos objetos correspondientes al chasis, ruedas y patines, con los correspondientes empalmes.
2. Ajustar todos los parámetros e interacciones para observar un comportamiento razonable.
3. Encontrar velocidades en la simulación cuyo comportamiento sea similar al comportamiento con las velocidades reales. Para esto se simuló el experimento en el que se recorrió 0,50 metros en línea recta y se buscaron velocidades simuladas que hicieran que el robot recorriera en la simulación la misma distancia en tiempos similares.
4. Verificar el comportamiento de dichas velocidades en el caso de los giros y si es necesario volver a ajustar parámetros. Volver al paso anterior.

De esta manera se procedió hasta obtener resultados satisfactorios. No existe una única manera de ajustar un modelo simulado y este procedimiento pretende ser simplemente un ejemplo de como ajustar un modelo de simulación que se basa en un modelo de referencia real.

5.2.3. Resultados

Se realizó el modelado con éxito. El mensaje de creación de los objetos en la simulación se puede encontrar en el anexo correspondiente.

En la figura 19 se pueden ver dos imágenes. La primera es una captura de pantalla de la simulación realizada en la cual se puede apreciar el robot modelado y el campo de juego en el cual se mueve este robot. La segunda es una foto del robot real en una posición similar.

Figura 19: SumBot

A continuación se muestran una captura de pantalla de la simulación en la cual el robot se encuentra al revez, y una foto del robot real en una situación similar.

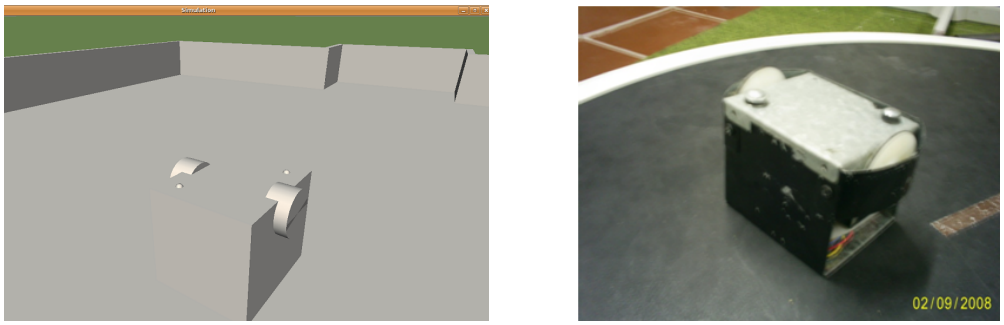

Figura 20: SumBot, parte inferior

Mapeo de velocidades

El mapeo de velocidades obtenido finalmente se en el cuadro 4. En la columna izquierda se puede observar las velocidades que se utilizan en el modelo de referencia real, y en la columna derecha las velocidades obtenidas luego de la realización del ajuste.

Velocidad real	Velocidad en la simulación
1	2.2
2	2.6
3	3.25
4	4
5	6

Cuadro 4: Modelado de un caso real. Mapeo de velocidades

Línea recta

El ajuste de las velocidades se realizó de modo tal de minimizar el error en el tiempo necesario para recorrer 0.50 metros. Se obtuvieron resultados que se muestran en la tabla a continuación. La columna " % Error" indica el porcentaje de error con respecto a las mediciones análogas utilizando las velocidades reales correspondientes a las velocidades simuladas dadas.

Velocidad en la simulación (izquierda, derecha)	Tiempo en segundos	% Error
2.2, 2.2	11.37	3.7
2.6, 2.6	9.63	1.4
3.25, 3.25	7.74	2.9
4, 4	6.33	1.6
6, 6	4.29	4.4

Cuadro 5: Modelado de un caso real. Resultado recorrida de 0,50 metros

Giros

En el caso de los experimentos de giros se obtuvieron los resultados que se muestran en la tabla a continuación. Se muestra el porcentaje de error en los tiempos de giro y el porcentaje de error del diámetro de giro con respecto a la medición en el modelo de referencia.

Vels. en la sim. (izq,der)	Tiempo en segs	% Err. tiempos	Diam. de giro en mts.	%Err. dists.
6, 4	18.72	2.3	0.74	6.3
6, 3.25	14.2	3.88	0.41	2.4
6, 2.6	11.4	1.7	0.31	8.8
6, 2.2	10.11	2.9	0.25	4.2
6, 0	6.1	6.1	Despreciable	-
6, -6	3.3	0.9	Sobre el eje	-
4, -4	4.68	4.1	Sobre el eje	-
3.25, -3.25	5.61	8	Sobre el eje	-
2.6, -2.6	7	7.9	Sobre el eje	-
2.2, -2.2	9.04	9.7	Sobre el eje	-
3.25, 2.2	8.66	7.3	Despreciable	-

Cuadro 6: Modelado de un caso real. Resultado giros

6. Conclusiones y trabajos futuros

En este capítulo se detallan las conclusiones alcanzadas luego de la finalización del proyecto. También se analizan diversas mejoras y extensiones que eventualmente se le podrían realizar al sistema.

6.1. Conclusiones

El principal objetivo de este proyecto no fue solamente crear un simulador en el cual se pudieran realizar simulaciones físicas de una determinada realidad. Tampoco fue crear una aplicación en la cual se lograra simular competencias de sumo o fútbol de robots. El verdadero objetivo fue crear una herramienta con la cual los usuarios del sistema lograran optimizar el comportamiento de sus robots, tanto a nivel individual como colectivo.

En este contexto, al tomar decisiones de diseño se hizo hincapié en la extensibilidad que debería tener el sistema. Se construyó el mismo con el alcance necesario para realizar simulaciones orientadas al desarrollo de robots para competencias de sumo o fútbol. Sin embargo, la extensibilidad lograda permite que el usuario del sistema lo pueda adaptar a sus necesidades y lograr cubrir un amplio rango de situaciones.

La necesidad de modelar simulaciones de sumo o fútbol de robots es satisfecha a través de la creación de entidades que interactúan en el modelo físico de la simulación. Estas entidades pueden modelar robots, el campo de juego o la pelota de una competencia de fútbol de robots. Se cuenta con la posibilidad de modelar estas entidades con una amplia variedad de características. Por ejemplo, es posible modelar robots que están compuestos por objetos rígidos simples y ruedas con movimientos independientes, cuyas cantidades solamente están limitadas por el poder de procesamiento del hardware sobre el cual se está ejecutando el sistema.

El sistema brinda algunas herramientas muy útiles en tareas de optimización y depuración, así como también como para la aplicación de técnicas de aprendizaje automático. Entre otras cosas, se brinda la posibilidad de manejar de manera flexible el paso de tiempo, contando con la posibilidad de detener el mismo, avanzar paso a paso, o avanzar a mayor velocidad que el tiempo real. Adicionalmente, la funcionalidad de ajuste automático del paso de tiempo permite que la simulación transcurra a una velocidad tal que el tiempo real y el tiempo simulado coincidan, logrando un mayor nivel de independencia respecto del aumento de carga o caída de performance del sistema.

Otras funcionalidades importantes construídas en el sistema refieren a la posibilidad de almacenar en un archivo una simulación, cargar una simulación desde un archivo, especificar la posición de entidades, aplicarle velocidades a los motores de un robot y cambiar propiedades físicas del entorno.

Se construyó el sistema con un enfoque multiagente, en el cual la simulación es controlada por agentes externos a través de mensajes. Esto logra una total independencia de la plataforma sobre la cual se están ejecutando los agentes externos y el lenguaje de programación utilizado para su desarrollo. Mediante los mensajes enviados por dichos agentes, estos pueden acceder a todas las funcionalidades que brinda el sistema. En el contexto de competencias robóticas donde se desea utilizar el sistema, es importante destacar que se cuenta por parte de los agentes externos con la posibilidad de controlar el comportamiento de los robots, especificando la velocidad de sus motores y también obtener información acerca del estado de la simulación a través de la recepción de mensajes de difusión.

En resumen, se cumplieron con los requerimientos establecidos inicialmente, optando por diseñar un simulador genérico con fuerte foco en la extensibilidad. Es una herramienta que puede ser de gran utilidad al momento de desarrollar comportamientos robóticos y evolucionar dichos comportamientos a través de técnicas de aprendizaje automático. También

puede ser utilizado para probar nuevos modelos físicos antes de construirlos realmente. Entre los puntos débiles se destaca el hecho de que la variedad de empalmes y cuerpos que se pueden modelar es bastante limitada. Si se desea modelar realidades más complejas es necesario extender el sistema para que soporte mayor variedad de cuerpos y empalmes.

Mirando en perspectiva, el proyectó planteó diversos problemas interesantes, a los que alguien que desee construir un sistema similar deberá enfrentarse.

En primer lugar, interactuar con motores físicos no es una tarea sencilla, y ocultar parte de su complejidad al usuario del sistema era importante para lograr una buena usabilidad. Proponer una interfaz física genérica y flexible, la cual soportara un posible cambio de motor físico, fue algo que demandó mucha investigación. Para definir ésta interfaz se buscó un denominador común entre los distintos motores físicos que se relevó.

Otro problema interesante que se planteó fue la necesidad de contar con distintas formas de comunicación entre los robots y los procesos que los controlan, buscando adaptabilidad a distintos contextos.

Sin embargo, el mayor problema que se encontró en el desarrollo de este proyecto fue lograr que todos los componentes del sistema funcionen correctamente en un contexto donde se busca procesamiento en tiempo real. En este contexto, un tiempo de procesamiento elevado por parte del sistema influye negativamente en la calidad de sus prestaciones. Las tareas más demandantes en términos computacionales que se realizan son el avance y mantenimiento del modelo físico, el procesamiento gráfico para mostrar en pantalla el estado de la simulación y la decodificación de mensajes recibidos desde agentes externos. Estas tareas deben realizarse a cada iteración del sistema, y cumplirlas de manera correcta y muy eficiente fue sin dudas el mayor desafío de este proyecto.

6.2. Trabajos a futuro

A continuación se describen las líneas de trabajo a futuro más importantes, explicando los distintos aspectos del sistema que pueden ser mejorados.

6.2.1. Interfaz física

El sistema propone una interfaz física genérica, la cual está implementada utilizando el motor físico ODE. Esta interfaz genérica expone ciertas funcionalidades del motor físico utilizado, pero muchas otras funcionalidades no están contempladas.

Varios de los tipos de cuerpos que presenta ODE no están contemplados, solamente los tipos de cuerpos fundamentales pueden ser utilizados a través de la interfaz física del sistema. Una importante mejora sería proveer el catálogo completo de cuerpos del motor físico a través de la interfaz física, así como también exponer cuerpos más complejos que a su vez se implementaran en ODE a través de la creación de diversos cuerpos.

La gran mayoría de los empalmes presentes en el motor físico tampoco están contemplados y no pueden ser utilizados a través de la interfaz física. Solamente dos empalmes están presentes en la interfaz física genérica: Fixed y WheelJoint. Este último internamente se representa con un empalme de tipo Hinge-2. Proveer todos los empalmes de ODE a través de la interfaz física, e incluso otros más complejos (creando internamente un empalme universal con determinadas características) sería una importante mejora.

El hecho de utilizar una interfaz física genérica y tener una arquitectura extensible, hacen que sea posible cambiar el motor físico que implementa dicha interfaz, logrando por ejemplo, la posibilidad de disponer de diversas implementaciones de la interfaz física, cada una utilizando un motor físico genérico. La arquitectura permitiría también cambiar de manera dinámica el motor físico utilizado, eventualmente sin que esto pueda ser notado para un

observador de la simulación. También cambiar el motor físico por otro que eventualmente pueda tener mayores prestaciones sería una importante mejora en el sistema.

6.2.2. Entidades

Actualmente el sistema soporta la creación de tres tipos de entidades: robots, campos de juego y pelota. Se resolvió crear estas tres entidades para proveer un contexto semántico que facilitara la tarea de creación de simulaciones de sumo y fútbol de robots.

Sin embargo, podría ser necesario crear entidades más complejas para manejar una variedad mayor de situaciones. Sería especialmente útil contar con una entidad genérica sin ningún tipo de restricciones en cuanto a los cuerpos y empalmes que la componen. Si a esto se suma contar con todos los cuerpos y empalmes que brinda el motor físico, se tendría una entidad que podría modelar físicamente cualquier construcción que se pudiera realizar con el motor físico utilizado.

Si se quisiera contar con elementos como sensores en los robots, las entidades serían el contexto ideal para resolver esta situación. Tomando como ejemplo un sensor de proximidad, el mismo se podría implementar accediendo a la información de los distintos cuerpos del modelo físico y calculando las distancias entre éstos y la entidad que tiene el sensor. Esta información se podría trasmitir en los mensajes de difusión, junto a la demás información de la entidad en cuestión.

También sería una importante mejora disponer de alguna herramienta para construir de manera más sencilla la especificación del modelo físico de una entidad. Esta herramienta podría tener una interfaz gráfica en la cual se pudiera visualizar el modelo físico actual, realizando modificaciones de manera interactiva, y generando automáticamente la especificación del modelo creado. Esta especificación sería la que luego se enviaría encapsulada en el mensaje de creación de la entidad.

6.2.3. Colisiones

Con el objetivo de optimizar el rendimiento del sistema en cuanto al manejo de colisiones, internamente se detectan solamente las colisiones entre cuerpos que pertenecen a distintas entidades. No se detectan por ejemplo, colisiones entre dos cuerpos que componen un mismo robot. Una posible mejora posibilitaría manejar este tipo de colisiones también, de manera tal que se detectaran y manejaran colisiones de distintos cuerpos que pertenecen a una misma entidad.

6.2.4. Comandos

Gran parte de la lógica del sistema y del manejo de sus funcionalidades está encapsulada en comandos. Los comandos presentan la manera más sencilla y eficaz de extender las funcionalidades del sistema.

Se podrían crear comandos de control para proveer nuevas funcionalidades de control de la simulación o también crear comandos de entidad para aumentar la interacción de las entidades con los agentes externos al sistema.

6.2.5. Optimización

Debido al gran procesamiento de información que se realiza, la optimización de todos los algoritmos utilizados en este procesamiento, impactaría positivamente en la performance del sistema. Aún existe un buen margen para realizar optimizaciones algorítmicas en todos los componentes. En particular, cuando la carga de mensajes recibidas por el núcleo de simulación es muy grande, el tiempo necesario para su decodificación es una cuello de botella. Por

lo tanto, mejoras algorítmicas en la decodificación de estos mensajes acarrearía considerables mejoras en el rendimiento del sistema en situaciones de mucha carga.

6.2.6. Armar entorno de simulación

Sería una funcionalidad útil contar con la opción de cargar datos iniciales de una simulación desde un archivo siempre que se ejecuta el sistema. Este archivo podría ser un parámetro de ejecución opcional.

6.2.7. Agentes Externos

Dado que cualquier proceso puede comunicarse con el sistema (a través del protocolo UDP), la variedad de propósitos con los cuales un agente se puede conectar al sistema es muy grande. En particular, dado el contexto de este proyecto, algunos ejemplos de agentes externos que pueden desarrollarse son: herramientas de logging de información, interfaces gráficas, árbitros automáticos para competencias de robots, distintas estrategias de control de robots, entre otros.

6.2.8. Interfaz gráfica

Sería una gran mejora en cuanto a usabilidad si se brindara al usuario la posibilidad de realizar modificaciones sobre el modelo físico a través de la interfaz gráfica, por ejemplo, manejándo los eventos del ratón. Estas modificaciones podrían ser cambiar la posición de entidades utilizando el ratón, cambiar la velocidad de las ruedas de robots utilizando el teclado, etc.

Referencias

[ABRA05] Gustavo Armagno, Facundo Benavides, and Claudia Rostagnol, "Descripción de la Arquitectura del Sistema" Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República ,Uruguay, 2005.

[ABRR05] Gustavo Armagno, Facundo Benavides y Claudia Rostagnol, Robosoccer Engine, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.

[ACGS03] Alvaro Castroman, Manual de usuario GSim, Setiembre 2003.

[AFE07] Anthony Figueroa, Estado del Arte, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.

[AFM08] Anthony Figueroa, Manual Técnico, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2008.

[AFPR07] Anthony Figueroa, Implementación de un Prototipo, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.

[AFR07] Anthony Figueroa, Requerimientos y Planificación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2007.

[AFV08] Anthony Figueroa, Verificación, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2008.

[AGBR07] Thiago Souto, Maior Cordeiro de Farias, Daliton da Silva, Guilherme de Sousa Moura, Márcio Augusto Silva Bueno, Veronica Teichrieb and Judith Kelner, Tutorial Ageia PhysX, 2007.

[AGDEV] Sitio web http://devsupport.ageia.com, Mayo 2007.

[AGEIA] Sitio web http://www.ageia.com, Mayo 2007.

[AM06] Axel Seugling and Martin Rolin, Evaluation of Physics Engine and Implementation of a Physics Module in a 3d-Authoring Tool, Master Thesis, Department of Computing Science, Umea University, Sweden, Marzo 2006.

[BULL] Sitio web http://www.continuousphysics.com, Mayo 2007.

[CAN05] Canales, R.; Casella, S.; Rodríguez, P. "Proyecto Forrest Liga de Simulación 2D RoboCup - Descripción de la Arquitectura", Instituto de Computación, Facultad de Ingeniería, Uruguay, Junio 2005.

[CCS05] Raúl Canales, Serrana Casella y Pablo Rodriguez, Fútbol de Robots: Liga de Simulación 2D Robocup, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, Junio 2005.

[CDCO] Sitio Web http://www.codecolony.de/opengl.htm, Marzo 2008

[CHEN03] Chen, Dorer, Foroughi, Heintz, Huang, Kapetanakis, Kostiadis, Kummeneje, Murray, Noda, Obst, Riley, Steffens, Wang, Yin. ,User Manual - RoboCup Soccer Server, versión 7.07 and laters - Febrero 2003.

[DEVM] Sitio Web http://devmaster.net, Mayo 2007.

[DF03] Hans Dollen y Wim Fikkert, MiS20 The Robotic Soccer Simulator, Junio 2003.

[DMPHI] Sitio Web http://www.lec.ufrgs.br/~dmbasso/phi/doku.php, Mayo 2007.

[DVGAM] Sitio Web http://www.idevgames.com, Diciembre 2007.

[EUCSP] Sitio Web http://www.euclideanspace.com/, Diciembre 2007

[FIRA] Sitio Web FIRA. http://www.fira.net, Junio 2007 .

[GPHYS] Sitio Web http://www.gphysics.com/physics-engines, Diciembre 2007

[HAVOK] Sitio Web http://www.havok.com, Mayo 2007.

[HH07] Henrik Hansson, Craft Physics Interface, Master Thesis, University of Linkoping, Sweden, Enero 2007.

[IRRL] Sitio web http://irrlicht.sourceforge.net, Mayo 2007.

[Java] Sitio Web http://www.java.com, Octubre 2007.

[KRU] Kruchten, P. "The "4+1" View Model of Software Architecture". Rational Software Corp. IEEE Software12(6), November 1995.

[LAN] Sitio Web http://osnews.com/story/5602/Nine_Language_Performance_Roundup:_Benchmarking_Math_&_File_I_O, Marzo 2008.

[MCLAN] Sitio Web http://furryland.org/~mikec/bench/, Marzo 2008.

[MIS20] Sitio web http://hmi.ewi.utwente.nl/MiS20/, Mayo 2007.

[NGD] Sitio Web Newton Dynamics Engine, http://www.newtongamedynamics.com, Mayo 2007.

[NHC] Sitio Web http://www.runehunter.phpnet.us/NewtonHelpCode.html , Mayo 2007.

[NVPHY] Sitio Web http://www.thephysicsengine.com, Mayo 2007.

[NVWIK] Sitio Web http://en.wikipedia.org/wiki/NV_Physics_SDK, Mayo 2007.

[NWI07] Sitio Web Newton Wiki, http://walaber.com/newton_wiki/index.php, Mayo 2007.

[ODE] Sitio Web http://www.ode.org, Mayo 2007.

[OGRE] Sitio Web http://www.ogre3d.org, Mayo 2007.

[OGRW] Sitio Web http://www.ogre3d.org/wiki/index.php/Main_Page, Mayo 2007.

[OMWB07] Olivier Michel, Webots: Professional Mobile Robot Simulation, 2007.

[OPENGL] Sitio Web http://www.opengl.org, Octubre 2007

[OPNW] Sitio Web http://laral.istc.cnr.it/wiki/index.php/OPAL_Newton, Diciembre 2007

[OPTI] Sitio Web http://www.opentissue.org, Mayo 2007.

[PHY2D] Sitio Web http://physics2d.googlepages.com/home, Diciembre 2007.

[PSBOB06] Ing. Patrik Stellmann, Simulator Bob, 3D Simulation Environment for Mobile Robots, Febrero 2006.

[ROBCU] Sitio Web http://www.robocup.org, Mayo 2007.

[RSOCF] Robot Soccer v1.5a http://www.fira.net/soccer/simurosot/R_Soccer_v15a_030204.exe, Mayo 2007.

[RUP] IBM Rational Unified Process. Disponible en: http://www.rational.com/rup, Octubre 2007.

[SIMBOB] Sitio Web http://simbob.sourceforge.net/, Octubre 2007

[Smi07] Rusell Smith, Open dynamics engine v0.8 user guide, Mayo 2007.

[SROBC] Sitio Web http://sserver.sourceforge.net/, Mayo 2007

[TAXIS] Sitio Web http://trueaxis.com, Mayo 2007.

[TOKA] Sitio Web http://www.tokamakphysics.com, Mayo 2007.

[UML] Unified Modeling Language. Disponible en: http://www.omg.org/uml, Octubre 2007.

[WBTS] Sitio Web http://www.cyberbotics.com/, Octubre 2007.

[WIKIIA] Sitio Web http://es.wikipedia.org/wiki/Inteligencia_Artificial, Febrero 2008.

[WMAS] Sistemas Multi-Agentes, Sitio Web http://es.wikipedia.org/wiki/Sistema_multi-agente, Febrero 2008.

Glosario

Aprendizaje automático. Rama de la Inteligencia Artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender.

FIRA. Federation of International Robot-soccer Association.

F.R.U.T.O. Proyecto en el cuál se desarrolló un equipo de fútbol de robots que se deempeñó en la liga simulada de la FIRA.

Fútbol de Robots. Competencia robótica en la cual dos equipos de robots compiten con el objetivo de marcar una mayor cantidad de goles en el arco rival.

GSim. Geometric Simulator. Simulador desarrollado en el marco del proyecto F.R.U.T.O.

Inteligencia Artificial. Rama de la ciencia informática dedicada al desarrollo de agentes racionales artificiales.

MINA. El Grupo MINA (Network Management - Artificial Intelligence), es un grupo de investigación perteneciente al Instituto de Computación de la Facultad de Ingeniería.

Motor físico. Librería basada en leyes físicas que mantiene el modelo físico de una simulación.

Multicast. Es el envío simultáneo de información a un determinado grupo de destinatarios.

ODE. Open Dynamics Engine. Motor físico utilizado en este proyecto.

OpenGL. Open Graphics Library.

Robocup. Robot Soccer World Cup.

Robot. Agente artificial, ya sea mecánico o virtual, que cuenta con algún grado de inteligencia.

Sumbot. Robot construido por el Grupo MINA. Participa en el Campeonato Uruguayo de Sumo de Robots.

Sumo de Robots. Competencia robótica en la cual cada robot tiene el objetivo de empujar su oponente hacia afuera de un área determinada del campo de juego.

TCP. Transmission Control Protocol.

Tiempo real. Tiempo que transcurre en la realidad.

Tiempo simulado. Tiempo virtual que transcurre en el contexto de una simulación.

UDP. User Data Protocol.

Anexo 1 - Modelado del SumBot

El mensaje enviado al núcleo de simulación para crear el modelo simulado del robot SumBot y un campo de juego análogo al de la categoría Simurosot de FIRA se muestra a continuacion.

```
<Message>
  <Simulation_Message>
    <Control_Request Command=Register_Materials_Interactions">
      <Material>
        <Material_Id>0</Material_Id>
        <Material_Name>Wood</Material_Name>
      </Material>
      <Material>
        <Material_Id>1</Material_Id>
        <Material_Name>Metal</Material_Name>
      </Material>
      <Material>
        <Material_Id>2</Material_Id>
        <Material_Name>Patas</Material_Name>
      </Material>
      <Interaction>
        <Material_1>0</Material_1>
        <Material_2>0</Material_2>
        <Hardness>0.15</Hardness>
        <Bounciness>0.6</Bounciness>
        <Slip1>0.1</Slip1>
        <Slip2>0.1</Slip2>
        <Friction>1</Friction>
      </Interaction>
      <Interaction>
        <Material_1>0</Material_1>
        <Material_2>1</Material_2>
        <Hardness>0.1</Hardness>
        <Bounciness>0.1</Bounciness>
        <Slip1>1.3</Slip1>
        <Slip2>2</Slip2>
        <Friction>99</Friction>
      </Interaction>
      <Interaction>
        <Material_1>1</Material_1>
        <Material_2>1</Material_2>
        <Hardness>0.15</Hardness>
        <Bounciness>0.6</Bounciness>
        <Slip1>0.1</Slip1>
        <Slip2>0.1</Slip2>
        <Friction>1</Friction>
      </Interaction>
      <Interaction>
        <Material_1>0</Material_1>
        <Material_2>2</Material_2>
        <Hardness>0</Hardness>
```

```
      <Bounciness>1</Bounciness>
      <Slip1>0</Slip1>
      <Slip2>0</Slip2>
      <Friction>0</Friction>
    </Interaction>
  </Control_Request>
  <Control_Request Command="Create_Entity" Type="Robot">
    <Response>true</Response>
    <Data>
      <Entity_id>1</Entity_id>
      <Structure>
        <Body Type="Box">
          <Body_id>0</Body_id>
          <Material>0</Material>
          <Mass>0.9</Mass>
          <Position_X>0</Position_X>
          <Position_Y>0</Position_Y>
          <Position_Z>0.25</Position_Z>
          <Rotation_X>0</Rotation_X>
          <Rotation_Y>0</Rotation_Y>
          <Rotation_Z>0</Rotation_Z>
          <Size_X>0.13</Size_X>
          <Size_Y>0.11</Size_Y>
          <Size_Z>0.11</Size_Z>
        </Body>
        <Body Type="WheelCylinder">
          <Body_id>1</Body_id>
          <Material>1</Material>
          <Mass>0.045</Mass>
          <Position_X>0</Position_X>
          <Position_Y>0.055</Position_Y>
          <Position_Z>0.212</Position_Z>
          <Rotation_X>1.570796</Rotation_X>
          <Rotation_Y>0</Rotation_Y>
          <Rotation_Z>0</Rotation_Z>
          <Radius>0.024</Radius>
          <Length>0.025</Length>
        </Body>
        <Body Type="WheelCylinder">
          <Body_id>2</Body_id>
          <Material>1</Material>
          <Mass>0.045</Mass>
          <Position_X>0</Position_X>
          <Position_Y>-0.055</Position_Y>
          <Position_Z>0.212</Position_Z>
          <Rotation_X>1.570796</Rotation_X>
          <Rotation_Y>0</Rotation_Y>
          <Rotation_Z>0</Rotation_Z>
          <Radius>0.024</Radius>
          <Length>0.025</Length>
```

```
</Body>
<Body Type="Sphere">
  <Body_id>8</Body_id>
  <Material>2</Material>
  <Mass>0.01</Mass>
  <Position_X>-0.045</Position_X>
  <Position_Y>-0.035</Position_Y>
  <Position_Z>0.196</Position_Z>
  <Rotation_X>0</Rotation_X>
  <Rotation_Y>0</Rotation_Y>
  <Rotation_Z>0</Rotation_Z>
  <Radius>0.004</Radius>
</Body>
<Body Type="Sphere">
  <Body_id>9</Body_id>
  <Material>2</Material>
  <Mass>0.01</Mass>
  <Position_X>0.045</Position_X>
  <Position_Y>0.035</Position_Y>
  <Position_Z>0.196</Position_Z>
  <Rotation_X>0</Rotation_X>
  <Rotation_Y>0</Rotation_Y>
  <Rotation_Z>0</Rotation_Z>
  <Radius>0.004</Radius>
</Body>
<Joint Type="WheelJoint">
  <Joint_id>0</Joint_id>
  <Body_1>0</Body_1>
  <Body_2>1</Body_2>
  <Axis_1_X>1</Axis_1_X>
  <Axis_1_Y>0</Axis_1_Y>
  <Axis_1_Z>0</Axis_1_Z>
  <Axis_2_X>0</Axis_2_X>
  <Axis_2_Y>1</Axis_2_Y>
  <Axis_2_Z>0</Axis_2_Z>
  <Hardness>0.001</Hardness>
  <Bounciness>0.9</Bounciness>
  <Maximum_Torque>10</Maximum_Torque>
</Joint>
<Joint Type="WheelJoint">
  <Joint_id>1</Joint_id>
  <Body_1>0</Body_1>
  <Body_2>2</Body_2>
  <Axis_1_X>1</Axis_1_X>
  <Axis_1_Y>0</Axis_1_Y>
  <Axis_1_Z>0</Axis_1_Z>
  <Axis_2_X>0</Axis_2_X>
  <Axis_2_Y>1</Axis_2_Y>
  <Axis_2_Z>0</Axis_2_Z>
  <Hardness>0.001</Hardness>
```

```
            <Bounciness>0.9</Bounciness>
            <Maximum_Torque>10</Maximum_Torque>
          </Joint>
          <Joint Type="FixedJoint">
            <Joint_id>2</Joint_id>
            <Body_1>0</Body_1>
            <Body_2>8</Body_2>
          </Joint>
          <Joint Type="FixedJoint">
            <Joint_id>3</Joint_id>
            <Body_1>0</Body_1>
            <Body_2>9</Body_2>
          </Joint>
        </Structure>
        <Motors>
          <Motor_id>0</Motor_id>
          <Motor_id>1</Motor_id>
        </Motors>
        <Reference_Body>0</Reference_Body>
      </Data>
    </Control_Request>
    <Control_Request Command="Create_Entity" Type="Field">
      <Data>
        <Entity_id>7</Entity_id>
        <Structure>
          <Body Type="Box">
            <Body_id>0</Body_id>
            <Material>0</Material>
            <Mass>-1</Mass>
            <Position_X>0</Position_X>
            <Position_Y>0</Position_Y>
            <Position_Z>0</Position_Z>
            <Rotation_X>0</Rotation_X>
            <Rotation_Y>0</Rotation_Y>
            <Rotation_Z>0</Rotation_Z>
            <Size_X>2.8</Size_X>
            <Size_Y>1.82</Size_Y>
            <Size_Z>0.1</Size_Z>
          </Body>
          <Body Type="Box">
            <Body_id>1</Body_id>
            <Material>0</Material>
            <Mass>-1</Mass>
            <Position_X>0</Position_X>
            <Position_Y>-0.9</Position_Y>
            <Position_Z>0</Position_Z>
            <Rotation_X>0</Rotation_X>
            <Rotation_Y>0</Rotation_Y>
            <Rotation_Z>0</Rotation_Z>
            <Size_X>2.2</Size_X>
```

```
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>2</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>0</Position_X>
    <Position_Y>0.9</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>0</Rotation_Z>
    <Size_X>2.2</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>3</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>1.1</Position_X>
    <Position_Y>-0.56</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>1.5708</Rotation_Z>
    <Size_X>0.7</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>4</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>1.1</Position_X>
    <Position_Y>0.56</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>1.5708</Rotation_Z>
    <Size_X>0.7</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>5</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>-1.1</Position_X>
```

```
    <Position_Y>-0.56</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>1.5708</Rotation_Z>
    <Size_X>0.7</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>6</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>-1.1</Position_X>
    <Position_Y>0.56</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>1.5708</Rotation_Z>
    <Size_X>0.7</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>7</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>1.2</Position_X>
    <Position_Y>0.22</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>0</Rotation_Z>
    <Size_X>0.21</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
  </Body>
  <Body Type="Box">
    <Body_id>8</Body_id>
    <Material>0</Material>
    <Mass>-1</Mass>
    <Position_X>-1.2</Position_X>
    <Position_Y>0.22</Position_Y>
    <Position_Z>0</Position_Z>
    <Rotation_X>0</Rotation_X>
    <Rotation_Y>0</Rotation_Y>
    <Rotation_Z>0</Rotation_Z>
    <Size_X>0.21</Size_X>
    <Size_Y>0.02</Size_Y>
    <Size_Z>0.4</Size_Z>
```

```
</Body>
<Body Type="Box">
  <Body_id>9</Body_id>
  <Material>0</Material>
  <Mass>-1</Mass>
  <Position_X>1.3</Position_X>
  <Position_Y>0</Position_Y>
  <Position_Z>0</Position_Z>
  <Rotation_X>0</Rotation_X>
  <Rotation_Y>0</Rotation_Y>
  <Rotation_Z>1.5708</Rotation_Z>
  <Size_X>0.46</Size_X>
  <Size_Y>0.02</Size_Y>
  <Size_Z>0.4</Size_Z>
</Body>
<Body Type="Box">
  <Body_id>10</Body_id>
  <Material>0</Material>
  <Mass>-1</Mass>
  <Position_X>-1.3</Position_X>
  <Position_Y>0</Position_Y>
  <Position_Z>0</Position_Z>
  <Rotation_X>0</Rotation_X>
  <Rotation_Y>0</Rotation_Y>
  <Rotation_Z>1.5708</Rotation_Z>
  <Size_X>0.46</Size_X>
  <Size_Y>0.02</Size_Y>
  <Size_Z>0.4</Size_Z>
</Body>
<Body Type="Box">
  <Body_id>11</Body_id>
  <Material>0</Material>
  <Mass>-1</Mass>
  <Position_X>1.2</Position_X>
  <Position_Y>-0.22</Position_Y>
  <Position_Z>0</Position_Z>
  <Rotation_X>0</Rotation_X>
  <Rotation_Y>0</Rotation_Y>
  <Rotation_Z>0</Rotation_Z>
  <Size_X>0.21</Size_X>
  <Size_Y>0.02</Size_Y>
  <Size_Z>0.4</Size_Z>
</Body>
<Body Type="Box">
  <Body_id>12</Body_id>
  <Material>0</Material>
  <Mass>-1</Mass>
  <Position_X>-1.2</Position_X>
  <Position_Y>-0.22</Position_Y>
  <Position_Z>0</Position_Z>
```

```
                <Rotation_X>0</Rotation_X>
                <Rotation_Y>0</Rotation_Y>
                <Rotation_Z>0</Rotation_Z>
                <Size_X>0.21</Size_X>
                <Size_Y>0.02</Size_Y>
                <Size_Z>0.4</Size_Z>
              </Body>
            </Structure>
          </Data>
        </Control_Request>
    </Simulation_Message>
</Message>
```

Anexo 2 - Interfaz para el envío de comandos

Con el objetivo de de hacer más sencilla la tarea de controlar las distintas funcionalidades del sistema a través del envío de comandos, se desarrolló utilizando el lenguaje Java[Java] una aplicación denominada "SimRob Commander", cuya interfaz gráfica permite de manera sencilla enviar mensajes al sistema.

Una captura de pantalla durante la ejecución de esta aplicación se puede ver en la figura 21.

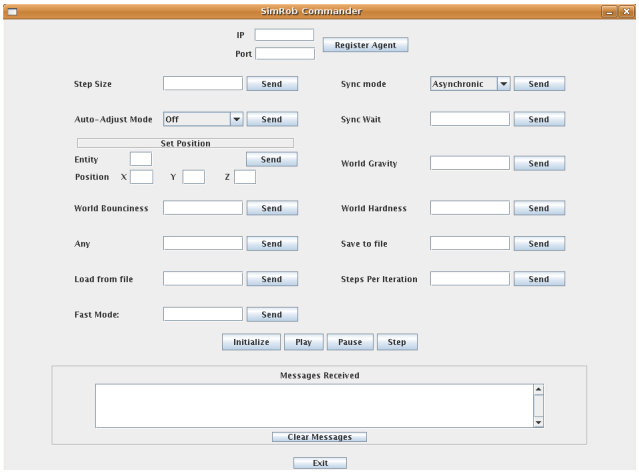

Figura 21: SimRob Commander

Se presentan diversas opciones en la interfaz gráfica, cuyo objetivo es el envío de un mensaje que una vez recibido y decodificado en el núcleo de simulación dispara la ejecución de un comando.

En la parte superior de la pantalla se cuenta con la opción de registrar el agente, en este caso la aplicación "SimRob Commander" en el sistema. Para esto se debe indicar la dirección de red y el puerto en el cual se está ejecutando el sistema. El registro permite comenzar a recibir mensajes de respuesta luego de la ejecución de cada uno de los comandos correspondientes. Estos mensajes una vez que son recibidos, son desplegados en la caja de texto que se encuentra cerca de la parte inferior de la interfaz.

No es necesario el registro del agente para el envío de comandos, sin embargo es necesario ingresar la dirección de red y el puerto en el cual se está ejecutando el sistema para que los mensajes lleguen a su destino correctamente.

Existen diversas funcionalidades en las cuales se debe ingresar algún valor correcto y luego presionar el botón contiguo. Al hacer esto, la aplicación crea un mensaje encapsulando esta información y lo envía. Ejemplos de este tipo de funcionalidades son la especificación del tamaño del paso de simulación, la solicitud para almacenar la simulación en un archivo, entre otras.

Otras funcionalidades que no requieren el ingreso de ningún valor solamente se ejecutan presionando el botón correspondiente. Ejemplos de estas funcionalidades son las utilizadas para controlar el paso del tiempo cuando el sistema se encuentra en modo paso a paso.

Mayor información se puede encontrar en el Manual Técnico del Sistema[AFM08].