Robosoccer Engine

Gustavo Armagno
Facundo Benavides
Claudia Rostagnol

Tutores

Gonzalo Tejera, Ernesto Copello

garmagno@fing.edu.uy, fbenavid@fing.edu.uy, crostagnol@gmail.com
http://www.fing.edu.uy/inco/grupos/mina/pGrado/FIRA2005.html

Instituto de Computación
Facultad de Ingeniería Universidad de la República
Montevideo - Uruguay

Índice

1. Introducción **4**

2. Motivación **4**
2.1. Alternativas analizadas . . . 4
2.1.1. GSim . . . 4
2.1.2. Phibra . . . 5
2.1.3. MiS20 . . . 6
2.1.4. Otros . . . 7

3. Objetivo **7**

4. Modelo del mundo **8**
4.1. ODE . . . 8
4.2. Odejava . . . 8
4.3. Modelado de los cuerpos . . . 9
4.3.1. Terreno . . . 9
4.3.2. Perímetro . . . 9
4.3.3. Pelota . . . 9
4.3.4. Robots . . . 9
4.4. Parámetros . . . 15

5. Aspectos de diseño **16**
5.1. Descripción de la solución . . . 16
5.1.1. Interfaz . . . 17
5.1.2. Cuerpos . . . 18
5.1.3. Actualizaciones . . . 19
5.1.4. Colisiones . . . 19
5.2. Interacción con el entorno . . . 19
5.2.1. Etapas de la simulación . . . 20

6. Conclusiones **21**
6.1. Trabajo futuro . . . 21

Referencias **23**

Índice de figuras

1. Simulador GSim. 5
2. Prototipo de simulador Phibra. 6
3. Simulador MiS20. 7
4. Construcción, disposición y parámetros de los cuerpos que admite el prototipo. . . 10
5. Robot de la categoría MiroSot de la FIRA. 11
6. Desplazamiento de los ejes de las ruedas. 12
7. Rotación frontal y lateral del chásis. 14
8. Diagrama de clases. 17
9. Diagrama de secuencia. 20

Índice de cuadros

1. Rotación de los cuerpos [TPG+02]. a) Rotación entorno al eje *x*. d) Rotación entorno al eje *y*. c) Rotación entorno al eje *z*. d) Matriz de rotación de un cuerpo. 13
2. Algoritmo de asignación de velocidades a cada rueda del robot. 15
3. Parámetros que permiten configurar el prototipo de simulador. 16

1. Introducción

El presente documento describe el desarrollo de un motor de simulación basado en leyes físicas para la categoría SimuroSot de la FIRA, aunque posiblemente extensible a otras categorías de características similares. Está orientado a programadores que estén familiarizados con la construcción de equipos de fútbol de robots y que posean un conocimiento básico sobre el funcionamiento de librerías para simulación física, como ODE (`www.ode.org`). El propósito es ofrecer una guía que sintetice el trabajo realizado para facilitar su comprensión en caso de ser retomado en desarrollos futuros.

2. Motivación

Una de las carencias del simulador oficial para la categoría SimuroSot de la FIRA es la ausencia de una interfaz que permita manejarlo de manera programática. La necesidad de contar con esta interfaz se vuelve particularmente trascendente si se desea aplicar aprendizaje automático *offline* en la resolución de algún aspecto de la estrategia.

Debido a que la FIRA publica únicamente los binarios del simulador -lo cual descarta la posibilidad de extender sus funcionalidades-, si el objetivo es realizar experimentos sin la intervención humana, la opción es construir un simulador alternativo.

Este simulador debe contar con dos características básicas:

- Poseer un motor de simulación lo suficientemente parecido al del simulador oficial como para que el impacto de trasladar una estrategia -cuyo comportamiento ha sido ajustado en el simulador alternativo- sea mínimo. O por lo menos que tenga bajo impacto en el desempeño del equipo.
- Avenirse al mecanismo de ejecución del simulador oficial, que implica ejecutar secuencialmente cada estrategia antes de avanzar el paso de simulación.

Además, sería deseable que contara con otras características que favorezcan la eficacia y eficiencia del proceso de aprendizaje, como:

- Permitir establecer el estado inicial de los cuerpos (posición de robots y pelota).
- Permitir controlar el tamaño del paso de simulación.

En el marco del proyecto FIbRA, para el cual se estableció como requerimiento no funcional el uso de Java como lenguaje de desarrollo, también se evalúa como característica excluyente la posibilidad de que el simulador interactúe con la Máquina Virtual de Java.

2.1. Alternativas analizadas

A continuación se listan los simuladores analizados, se resumen sus principales características y las desventajas que influyeron en la decisión de no tomarlos *per se* como sustitutivo del simulador oficial.

2.1.1. GSim

GSim, analizado en [ABR05c], es un simulador para fútbol de robots desarrollado por el equipo FRUTo [Cas03]. GSim es el acrónimo de Geometric Simulator, y su nombre radica en que el simulador utiliza un enfoque geométrico para detectar y resolver colisiones entre los cuerpos.

El simulador se caracteriza por:

- permitir establecer el estado inicial de los cuerpos;

- permitir controlar el tamaño del paso de simulación;
- permitir calibrar algunos aspectos de la simulación, abriendo la posibilidad de ajustarlo al simulador oficial;
- ejecutar secuencialmente cada estrategia antes de avanzar el paso de simulación;
- proporcionar una interfaz del programador extensible;
- estar desarrollado íntegramente en Java.

La principal debilidad del simulador es que no resuelve las colisiones entre robots. Sin embargo, con algunas modificaciones es posible reemplazar su motor de simulación, lo cual lo convierte en un candidato para realizarle un mantenimiento perfectivo.

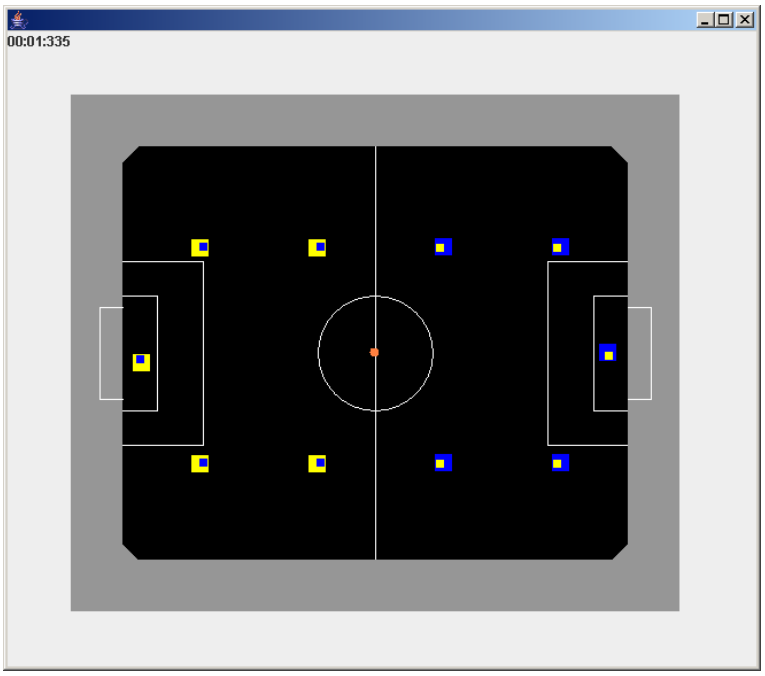

Figura 1: Simulador GSim.

2.1.2. Phibra

Phibra es un prototipo de simulador construido para evaluar una herramienta para desarrollar simuladores denominada Phi [ABR05a].

El prototipo se caracteriza por:

- poseer el soporte de un modelo físico basado en la librería ODE;
- permitir establecer el estado inicial de los cuerpos;
- permitir controlar el tamaño del paso de simulación;
- permitir calibrar algunos aspectos de la simulación, abriendo la posibilidad de ajustarlo al simulador oficial;

- posibilitar la comunicación entre el simulador y otras aplicaciones (en particular, las estrategias);
- estar codificado en un lenguaje de *scripting* denominado Lua, que facilita el mantenimiento del universo creado.

A través del prototipo fue posible encontrar una seria limitación de la herramienta Phi que la descarta como plataforma de construcción del simulador alternativo: no posee mecanismos que permitan detener la simulación. Esta limitante impide la ejecución secuencial de cada estrategia antes de avanzar el paso de simulación, característica básica del simulador oficial.

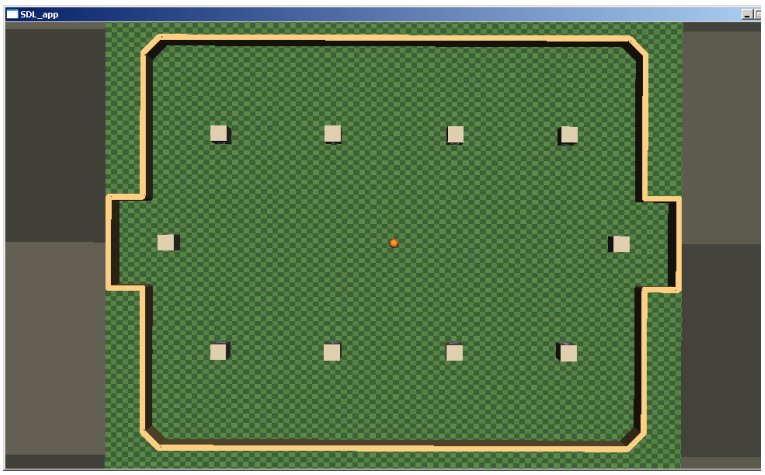

Figura 2: Prototipo de simulador Phibra.

2.1.3. MiS20

MiS20 es un simulador basado en las reglas de la categoría MiroSot *middle league* de la FIRA [ABR05b]. Fue desarrollado como parte de una tesis de grado para entrenar y probar la estrategia de un equipo real, denominado Mi20 [vdL05]. Entre sus objetivos, se establece que "los robots y la pelota deben simular su contra-parte real de una forma aproximadamente correcta" [DF03].

De los simuladores analizados, MiS20 es el más completo en cuanto a la interacción con el usuario, permitiendo controlar manualmente la ejecución y el estado del juego, establecer la posición y rotación de los cuerpos, y grabar y reproducir partidos.

El simulador se caracteriza por:

- permitir establecer el estado inicial de los cuerpos;
- permitir controlar el paso de simulación;
- permitir calibrar algunos aspectos de la simulación, abriendo la posibilidad de ajustarlo al simulador oficial;
- ejecutar secuencialmente cada estrategia antes de avanzar el paso de simulación;
- proporcionar una interfaz del programador extensible;
- estar desarrollado íntegramente en Java.

Como contrapartida, posee la misma debilidad que el simulador GSim: no detecta colisiones entre robots. Sin embargo, a diferencia del GSim, la separación entre el motor físico y el resto de la aplicación es difusa, obstaculizando un posible mantenimiento perfectivo que permita mejorarlo.

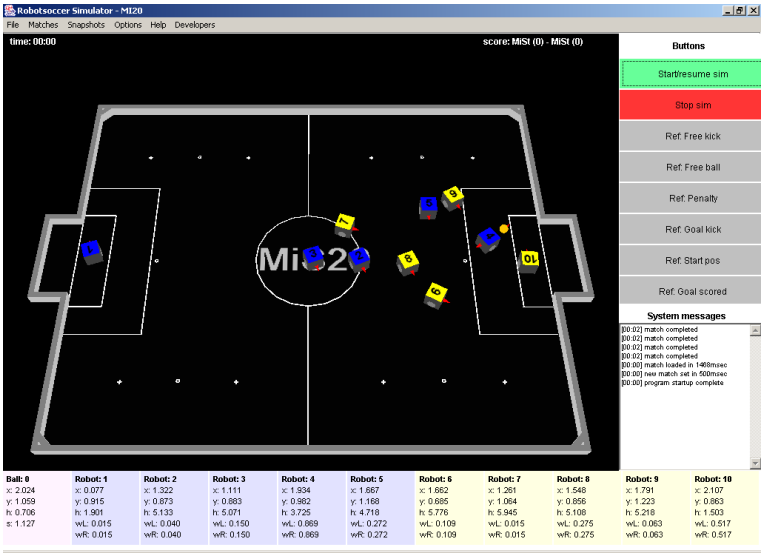

Figura 3: Simulador MiS20.

2.1.4. Otros

Physics 2D es un módulo que introduce un motor de física bidimensional en MASON, un simulador para entornos multi-agente [LBP06]. Forma parte de una tesis de maestría y su objetivo principal es "proveer una plataforma pequeña, fácil de usar y eficaz para realizar simulaciones físicamente realistas" [Tho03].

La principal debilidad del motor de simulación es que no implementa articulaciones que restrinjan la trayectoria de los robots [ea98]. Sin embargo, esta debilidad no representa una limitante ya que Physics 2D permite extender su funcionalidad para introducir nuevas restricciones en la cinemática de los cuerpos. Se señaló esta opción como una tarea a ser explorada en el futuro.

Además, se investigaron otros dos simuladores que presentaron fortalezas y debilidades similares a los analizados anteriormente [BT03, TCMC05].

3. Objetivo

Tomando como base la investigación realizada sobre dos estrategias utilizadas para modelar fútbol de robots (geométrica [Bea03, KLKZ03] y física [ea98, Smi04]), particularmente para ligas de la FIRA, y después de analizar el funcionamiento y las prestaciones que brindan los simuladores estudiados [ABR05c, ABR05a, Tho03, DF03, BT03, TCMC05] se define el alcance de esta línea de trabajo.

El objetivo consiste en prototipar un motor de simulación que se base en leyes físicas para, en primer lugar, demostrar la viabilidad de la construcción con las herramientas elegidas; y en segundo lugar, que sirva como punto de partida para futuros desarrollos que tengan como objetivo la construcción de un simulador basado en el simulador oficial.

4. Modelo del mundo

El mundo simulado está compuesto por el terreno, dieciséis paredes que conforman el perímetro de la cancha, diez robots cúbicos de dos ruedas y una pelota. La interacción entre estos cuerpos simula las leyes físicas a través del modelo físico proporcionado por la librería ODE (disponible en `www.ode.org`), implementada en C. Se optó por la utilización de la librería Odejava (disponible en `www.odejava.org`) que proporciona una interfaz en Java para utilizar las funcionalidades de ODE.

4.1. ODE

La librería ODE permite simular la dinámica entre cuerpos rígidos [Smi04]. Desde el punto de vista de la simulación, un cuerpo tiene propiedades que pueden cambiar con el tiempo -posición, velocidad lineal, orientación, velocidad angular- y propiedades que generalmente se mantienen constantes con el tiempo -masa, posición del centro de masa respecto al punto de referencia, matriz de inercia [1]-. La geometría de un cuerpo no es una propiedad dinámica y, por lo tanto, no puede cambiar.

Configuración ODE permite configurar los siguientes aspectos globales del universo simulado, el paso de simulación, la detección y la resolución de colisiones:

- vector de aceleración de caída libre ("gravedad");
- ERP (*error reduction parameter*), que permite de manera global reducir dislocaciones en los puntos de articulación;
- CFM (*constraint force mixing*), que permite de manera global suavizar las colisiones entre los cuerpos;
- tamaño del paso de simulación;
- cantidad de contactos entre cuerpos por paso de simulación.

Colisiones En cada paso de simulación, ODE detecta las colisiones producto de la interacción entre los cuerpos. Este proceso culmina con la generación de un conjunto de puntos de contacto. Cada punto de contacto aplica a dos cuerpos que colisionan. Para la resolución de cada colisión, ODE permite establecer información adicional sobre el punto de contacto (fricción, elasticidad, suavidad, etc.) determinando el comportamiento inmediato de ambos cuerpos [2].

La resolución de colisiones y la variación de los aspectos globales, de las propiedades de los cuerpos y de sus puntos de articulación, permiten definir el comportamiento de la simulación.

Unidades Las unidades de medida están basadas en el sistema MKS y los ángulos se miden en radianes.

4.2. Odejava

La librería Odejava es un nexo que permite acceder a las funcionalidades de ODE a través de Java mediante la invocación de métodos nativos usando JNI. Exporta una API de bajo nivel que encapsula las operaciones de ODE en un único objeto, y otra de alto nivel que pretende otorgar una visión orientada a objetos de los conceptos definidos por ODE [3].

[1] Describe cómo se distribuye la masa de un cuerpo entorno a su centro de masa.

[2] Desde un punto de vista estrictamente técnico, ODE crea una articulación especial por cada punto de contacto, denominada articulación de contacto. La información adicional se agrega sobre las articulaciones de contacto y no sobre los puntos de contacto.

[3] Esta opción, más adecuada para trabajar en Java, no termina de establecer un límite claro entre las responsabilidades de cada objeto, violando en general el principio de encapsulamiento.

Las principales fortalezas de Odejava son resolver la comunicación con ODE y facilitar la visualización del universo construido. Las principales debilidades son su dependencia en el componente nativo, que provoca un desfase entre las distribuciones de Odejava y ODE [4].

4.3. Modelado de los cuerpos

La figura 4 diagrama la construcción de los cuerpos, muestra su disposición y especifica los parámetros espaciales que admite el prototipo.

4.3.1. Terreno

El terreno es un plano sostén del resto de los cuerpos del modelo, representado por la ecuación 1.

$$ax + by + cz = d \tag{1}$$

Para el prototipo fueron fijados los valores $\begin{cases} a = 0 \\ b = 0 \\ c = 1 \\ d = 0 \end{cases}$, determinando el plano $z = 0$.

4.3.2. Perímetro

El perímetro de la cancha está compuesto por dieciséis prismas rectangulares. La forma del perímetro (fleftx, frightx, fbot, ftop, gleft, gright, gbot, gtop y corner_side) y las dimensiones de la pared perimetral (*height* y *thickness*) son parámetros configurables (ver 5.1.1).

4.3.3. Pelota

La pelota se modela con una esfera. A diferencia del modelo del robot, la pelota no pierde velocidad al desplazarse sobre el terreno o al rasar la pared perimetral o el chasis. Esto se debe a que en la resolución de colisiones, se establece que el rozamiento de la pelota con estos cuerpos sea nulo. El radio de la pelota y su masa son parámetros configurables.

4.3.4. Robots

El modelo geométrico de los robots es compuesto. Lo componen un chasis y dos ruedas paralelas ubicadas a ambos lados del mismo y unidas por un mismo eje como muestra la figura 4. Las dimensiones de estos cuerpos quedan determinadas por los valores que se utilizan como parámetros al momento de creación de cada robot (ver 5.1.1).

Chasis

El chasis se modela con un cubo.

[4] La última distribución de Odejava data de diciembre de 2004, mientras que la última de ODE data de junio de 2006 (verificado Agosto de 2006).

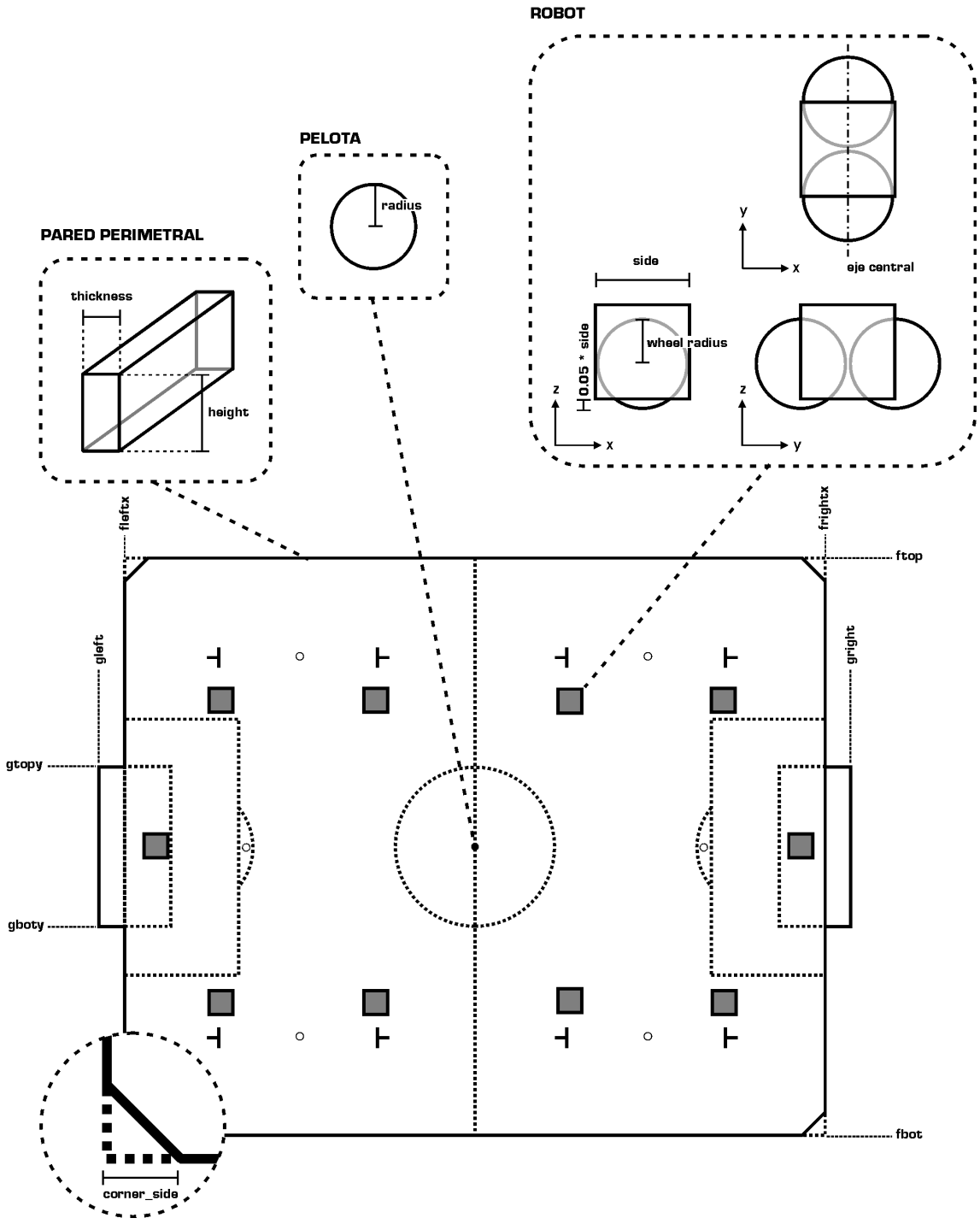

Figura 4: Construcción, disposición y parámetros de los cuerpos que admite el prototipo.

Ruedas

Las ruedas se modelan con esferas. Aunque en el modelo real los robots de la FIRA tienen ruedas cilíndricas, como se aprecia en la figura 5, la librería Odejava no proporciona construcciones de cilindros.

Este modelo de ruedas exige un manejo de colisiones diferente entre estas geometrías y los otros cuerpos que componen el modelo de mundo y que participan de la simulación. Para el caso, las únicas colisiones que se mantienen activas, y por lo tanto alteran la trayectoria de los robots, son las detectadas entre el terreno y las ruedas. Las restantes colisiones, que se pueden detectar tanto entre ruedas de un robot y otros cuerpos (pelota, paredes o chasis de otros robots) como entre ruedas de diferentes robots, permanecen inactivas sin alterar las trayectorias de los cuerpos involucrados.

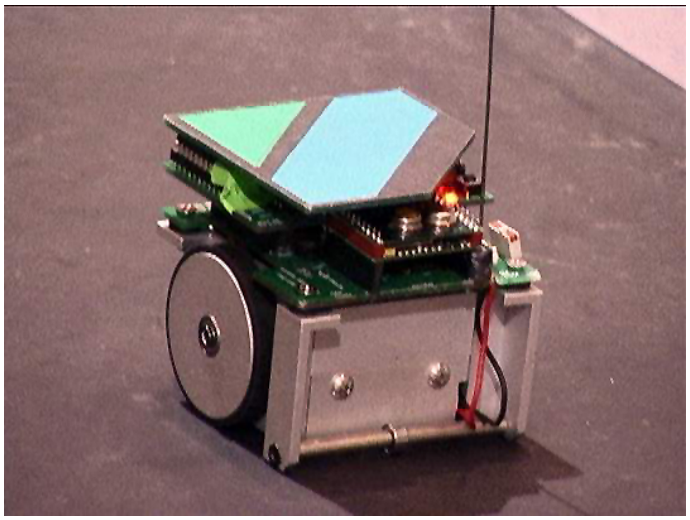

Figura 5: Robot de la categoría MiroSot de la FIRA.

Ejes

Los ejes sobre los que giran libremente las ruedas, se encuentran ubicados a una altura tal que el chasis permanezca elevado 5 % del lado del robot por encima del terreno como se muestra en la figura 4. A su vez, se puede configurar el desplazamiento de los ejes respecto al centro de la cara del robot en la dirección horizontal a lo largo del vector de orientación del robot (ver figura 6). Este parámetro es de singular importancia, ya que afecta los algoritmos de navegación puesto que las trayectorias hacia adelante y hacia atrás deben considerarse por separado. En el modelo real el eje de las ruedas se encuentra desplazado hacia la parte trasera del robot. No se conoce con exactitud esa medida.

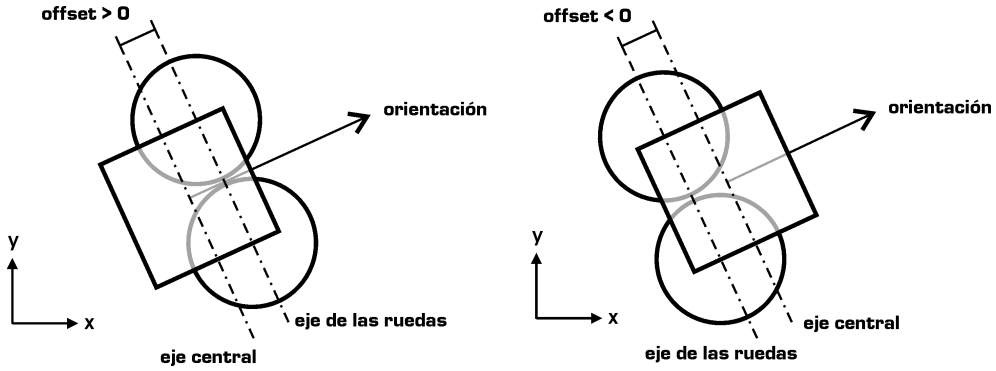

Figura 6: Desplazamiento de los ejes de las ruedas.

Cinemática

El modelo geométrico utilizado permite simular en forma correcta la cinemática que presentan los robots reales. Concretamente, este modelo permite reproducir la restricción de movimiento no holonómico [5].

El modelo cinemático se expresa en la ecuación 2 [ea98].

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} w; \quad \begin{cases} v: & \text{velocidad lineal} \\ w: & \text{velocidad angular} \\ \theta: & \text{rotación} \end{cases} \tag{2}$$

Dinámica [6]

Además de permitir modelar objetos y realizar un manejo acorde a leyes físicas, una parte central de este prototipo es permitir la interacción entre el usuario y el mundo simulado. Particularmente, es importante que el usuario pueda gobernar los movimientos de los robots afectando la velocidad de sus ruedas. En este sentido el modelo de robots elegido también incide sobre la forma en que se resuelve la simulación de la dinámica de estos cuerpos compuestos. En el caso de articulaciones rotacionales de un grado de libertad (ejes), los actuadores producen pares de fuerzas en los extremos de la articulación. El resultado es que a partir de relaciones dinámicas es posible controlar los actuadores de los ejes y, por lo tanto, el movimiento de las ruedas, logrando que los robots describan una trayectoria determinada.

Al igual que en cinemática, en dinámica, son posibles dos problemas, el directo y el inverso. El problema directo consiste en calcular el movimiento resultante a partir de las fuerzas que afectan los actuadores de las articulaciones. Por el contrario, el problema inverso consiste en calcular las fuerzas de los actuadores de las articulaciones a partir de las posiciones, rotaciones, velocidades y aceleraciones requeridos para el robot [TPG+02]. Este último es el que debe resolver el usuario para controlar las trayectorias de los robots.

Dinámica Directa

El modelo geométrico de los robots permite optar, en principio, por dos opciones para resolver este problema.

[5] Nota de redacción: no se encontró una traducción satisfactoria para *holonomic* (proveniente del Latín) [MMSV02]. La palabra "holonómico/a" no es admitida por el diccionario de la Real Academia Española.

[6] Mientras la Cinemática (del griego *kinema*, que significa movimiento) se ocupa de describir los movimientos, la Dinámica (del griego *dynamis*, que significa fuerza) analiza las causas de los mismos. [RHK96]

Una opción, que tal vez se asemeja más al modelo real, consiste en afectar la rotación de los ejes tal como si estuvieran conectados a un motor y que a partir de esa rotación giren las ruedas y por efecto de la fricción se muevan los robots. Este enfoque fue considerado sin obtener resultados positivos en los experimentos de simulación realizados. Sin embargo, no se descarta que estos resultados se deban a la inexperiencia en el manejo de la librería *ODE*, y que con más tiempo de trabajo e investigación puedan lograrse mejores.

La otra opción, en la que se basa este prototipo, es provocar el movimiento de los robots a partir de la aplicación de fuerzas horizontales perpendiculares a los ejes de las ruedas. El efecto de la aplicación de estas fuerzas, si se permite que las ruedas giren libremente sobre los ejes, es similar al que teóricamente se obtendría con la primera opción.

Un problema detectado durante la realización de las pruebas es que los robots sólo mantienen las trayectorias cuando tienen las dos ruedas apoyadas en el terreno. Si, por efecto de algún choque contra otro cuerpo o de la inercia en una trayectoria muy cerrada a velocidades altas, alguna de las ruedas pierde contacto con el terreno, la fuerza horizontal que se aplica al eje de la rueda sigue actuando, provocando que la trayectoria simulada difiera de la real. Por el contrario, en el caso real (o si se utilizara la primera opción mencionada al comienzo) la pérdida de contacto entre la rueda y el terreno anularía la fuerza de rozamiento, anulando por consiguiente el efecto de esa rueda sobre la trayectoria del robot.

Para disminuir el efecto de este problema se tiene en cuenta la rotación del chasis. Concretamente, la matriz de rotación del chasis que determina su orientación con respecto al sistema de coordenadas (x, y, z) y se expresa como composición de las rotaciones respecto a cada uno de los ejes (ver cuadro 1).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \quad \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a) b) c)

$$\begin{bmatrix} \cos\beta\cos\gamma & -\sin\alpha\cos\beta & \sin\beta \\ \cos\alpha\sin\gamma + \sin\beta\cos\gamma\sin\gamma & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta \\ \sin\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & \cos\alpha\sin\beta\sin\gamma + \sin\alpha\cos\gamma & \cos\alpha\cos\beta \end{bmatrix}$$

d)

Cuadro 1: Rotación de los cuerpos [TPG+02]. a) Rotación entorno al eje x. d) Rotación entorno al eje y. c) Rotación entorno al eje z. d) Matriz de rotación de un cuerpo.

La solución se basa en condicionar la actualización de las fuerzas horizontales que actúan sobre los ejes a la rotación del chasis (ver figura 7). Así, siempre que el valor absoluto de alguna de las componentes α o β de la rotación del chasis sea mayor que un umbral pequeño, los valores de las fuerzas no son actualizados. Cabe aclarar que *no actualizar* significa que en la iteración actual de la simulación actúen las fuerzas resultantes de la interacción de los objetos en la iteración anterior. De esta forma, se acota sustancialmente este efecto no deseado y se logra continuidad y suavidad en los movimientos de los robots.

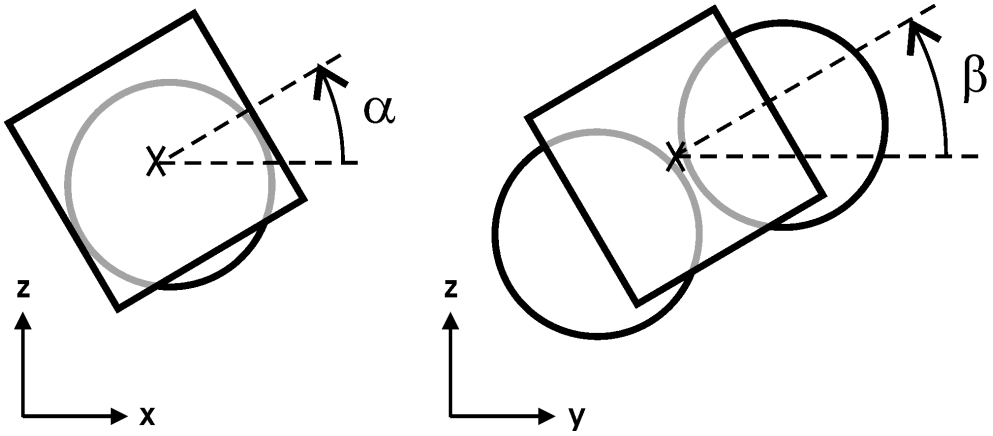

Figura 7: Rotación frontal y lateral del chásis.

Otro aspecto del problema dinámico directo consiste en relacionar el dominio de las velocidades que están permitidos, [-125, 125] *unidades de velocidad* [7], con los vectores de velocidad lineal (v_x, v_y, v_z) de cada rueda. Es decir, a partir de los valores escalares, que representan el módulo de los vectores velocidad, determinar las componentes de los mismos respecto a cada uno de los ejes de coordenadas.

El algoritmo del cuadro 2 muestra en forma esquemática como se resuelven los dos aspectos mencionados del problema dinámico directo.

[7] El simulador oficial establece ese rango de valores reales como dominio de velocidades que se pueden utilizar para mover los robots. Se utiliza el término *unidades de velocidad* ya que no hay documentación que indique en que unidades están expresados estos valores [ABR05b].

Cuadro 2: Algoritmo de asignación de velocidades a cada rueda del robot.

```
setVelocities(vl, vr) {
    umbral = 0.05;
    mRot = chassisBody.getRotation();

    // A partir de los elementos de la matriz de rotación se calcula la rotación respecto
al eje x e y.
    m00 = mRot.m00;
    m01 = mRot.m01;
    m02 = mRot.m02;
    m12 = mRot.m12;

    // Rotación respecto al eje y.
    beta = Math.asin(-m02);

    // Rotación respecto al eje x.
    alfa = Math.asin(m12 / Math.cos(beta));

    if (beta <= umbral && alfa <= umbral) {
        cosPhi = m00 / cosBeta;
        senPhi = m01 / cosBeta;

        // Según el cuadrante en el que esté el robot.
        vlX = vl * cosPhi;
        vlY = -vl * senPhi;
        vrX = vr * cosPhi;
        vrY = -vr * senPhi;

        // La velocidad en el eje z siempre es 0.
        leftWheel.setLinearVel(vlX, vlY, 0.0f);
        rightWheel.setLinearVel(vrX, vrY, 0.0f);
    }
}
```

4.4. Parámetros

En el cuadro 3 se resumen los parámetros que flexibilizan la construcción del universo y modifican el comportamiento de la simulación:

Cuadro 3: Parámetros que permiten configurar el prototipo de simulador.

Parámetro	Descripción
Gravity	Coordenada g_z del vector de aceleración de caída libre (g_x, g_y, g_z). El valor debe ser negativo para que los cuerpos sean atraidos verticalmente hacia el terreno.
ERP (*error reduction parameter*)	Permite reducir dislocaciones en los puntos de articulación. ERP varía entre 0 y 1. Si ERP=0, no se aplican fuerzas correctivas y las articulaciones de los cuerpos eventualmente podrían dislocarse a medida que avanza la simulación. Si ERP=1, entonces el motor de simulación intentará corregir posibles dislocaciones durante el próximo paso de simulación. El manual de ODE recomienda valores entre 0.1 y 0.8 [Smi04].
Step size	Tamaño del paso de simulación. El dominio de este parámetro es el conjunto de reales positivos.
Step interactions	Cantidad de contactos entre cuerpos por paso de simulación. El dominio de este parámetro es el conjunto de naturales.
FTOP, FBOT, FLEFTX, FRIGHTX, GTOPY, GBOTY, GLEFT, GRIGHT, CORNER_SIDE, WALL_THICKNESS, WALL_HEIGHT	Dimensiones de la cancha (ver figura 4).
Ball state	Posición y rotación de la pelota.
Ball radius	Radio de la pelota.
Ball mass	Masa de la pelota.
Robot state	Posición y rotación del robot.
Robot side	Lado del robot.
Robot mass	Masa del robot.
Wheel offset	Desplazamiento del eje de las ruedas respecto al centro del robot.
Wheel radius	Radio de las ruedas del robot.

5. Aspectos de diseño

Esta sección describe los aspectos de diseño más sobresalientes así como las alternativas evaluadas para construir la solución.

5.1. Descripción de la solución

El prototipo construido es un motor de simulación aplicado al dominio del fútbol de robots, particularmente para ligas de la FIRA.

Para poder simular la interacción entre los objetos del dominio es necesario contar con una herramienta que permita modelar estos objetos de tal forma que su comportamiento se vea afectado por las mismas leyes físicas que afectan a los robots reales. También debe permitir resolver aquellos eventos que los afectan (aceleraciones y colisiones) condicionados estos a su vez a las diferentes fuerzas que actúan en el modelo de mundo (gravedad, fricción, inercia).

En la figura 8 se muestra el modelo del mundo utilizado y su interacción con los componentes nativos de Odejava que son utilizados principalmente para controlar la simulación actualizando el estado de los cuerpos y resolver colisiones.

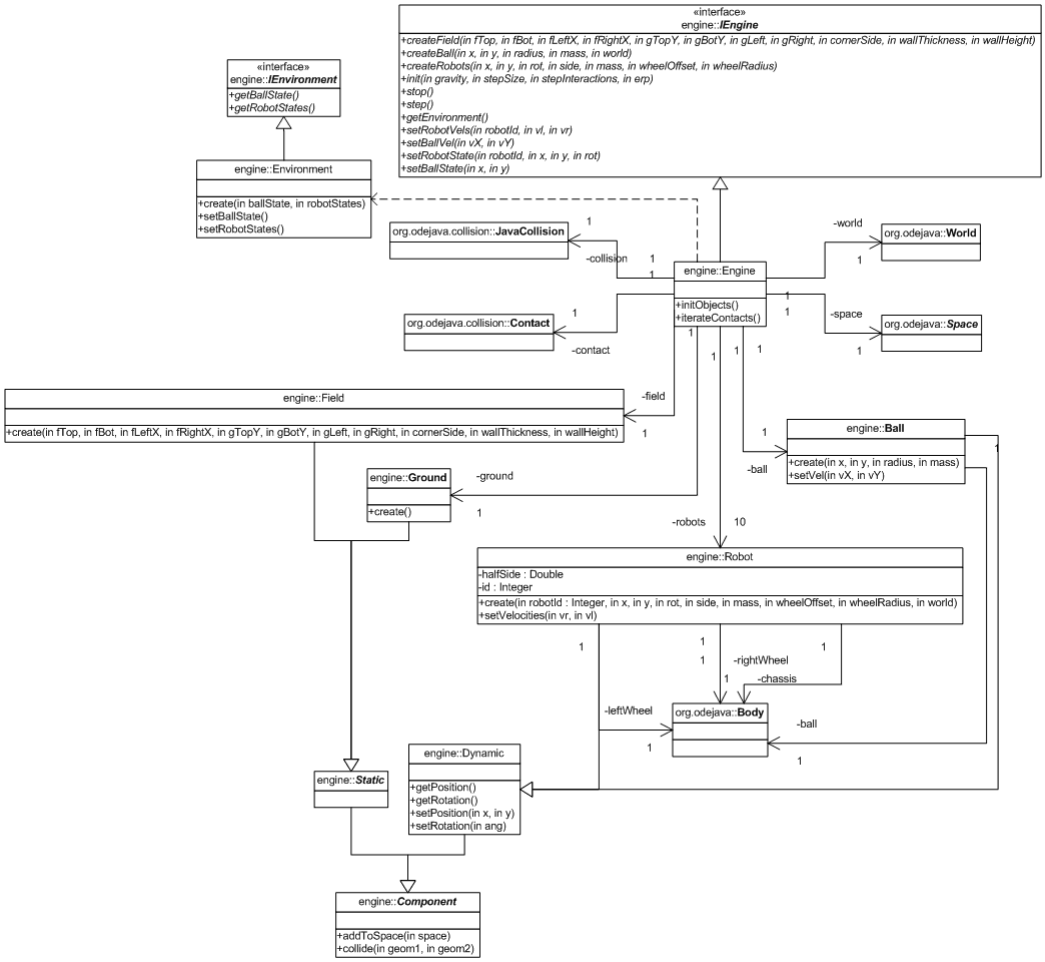

Figura 8: Diagrama de clases.

5.1.1. Interfaz

La interfaz del motor físico de simulación está representada en la interfaz *IEngine* que figura en el diagrama de clases. Las operaciones publicadas permiten la creación de un modelo del mundo similar al del simulador oficial, configuración de parámetros que afectan la simulación y un manejo básico de la misma. Por básico se entiende que es posible iniciar, terminar, avanzar paso a paso, obtener información sobre el estado de los cuerpos simulados e incidir en su comportamiento.

- *createField* : Permite crear el escenario donde se juega un partido de fútbol (piso y perímetro de una cancha) según las medidas que son recibidas por parámetro.
- *createBall* : Permite crear un cuerpo simple esférico que representa la pelota. La posición inicial, radio y masa son recibidos por parámetro.

- *createRobots* : Permite crear un conjunto de cuerpos compuestos que representan los robots. La posición y rotación de los robots es recibida a través de un arreglo de estados (triplas). El número de robots a crear queda determinado por el tamaño del arreglo. Además, el lado de las caras, la masa, el radio de las ruedas y un valor de desplazamiento que indica en qué posición relativa al centro será ubicado el eje de las ruedas, son recibidos en los restantes parámetros.

- *init* : Permite inicializar el modelo del mundo sobre el que se efectúa la simulación. La aceleración de caída libre, el tamaño del paso de tiempo, la cantidad de interacciones (puntos de colisión) por paso y un factor global de corrección de errores de simulación son recibidos por parámetro.

- *stop* : Finaliza una simulación liberando todos los recursos utilizados.

- *step* : Avanza el mundo un paso hacia adelante en el tiempo, tomando como referencia el tamaño del paso establecido en la inicialización.

- *getEnvironment* : Permite obtener información (posición y rotación) sobre los cuerpos móviles simulados (pelota y robots).

- *setBallState* : Permite modificar el estado del cuerpo *pelota* a través de la actualización de su estado (*x, y*).

- *setBallVel* : Permite afectar externamente el comportamiento del cuerpo *pelota* a través de la modificación del vector velocidad (v_x, v_y).

- *setRobotState* : Permite modificar el estado de cualquier cuerpo *robot* a través de la actualización de su estado (*x, y, rot*).

- *setRobotVels* : Permite afectar externamente el comportamiento de cualquier cuerpo *robot* a través de la modificación de la velocidad de sus ruedas (v_{left}, v_{right}).

5.1.2. Cuerpos

La librería de base utilizada para la simulación de la física (ver 4.2) maneja el concepto de geometrías y cuerpos para clasificar los diferentes objetos simulados según sean móviles (cuerpos) o fijos (geometrías). Así, a la primera clasificación pertenecen los robots y la pelota; y a la segunda el piso y el perímetro de la cancha.

Todos los objetos de la simulación están representados por la clase abstracta *Component*.

Fijos

Los objetos fijos están representados por la clase abstracta *Static*. Especializaciones de esta son el piso (*Ground*) y las paredes de la cancha (*Field*).

Estas geometrías permanecen fijas durante toda la simulación delimitando el escenario donde pueden moverse los cuerpos móviles.

- *Ground* : Representa el piso. El modelo geométrico es un plano (ver 4.3).

- *Field* : Representa las dieciséis paredes que componen una cancha de fútbol para ligas de la FIRA. El modelo geométrico de cada pared es un prisma (ver 4.3). Los parámetros permiten definir la ubicación relativa de las paredes por lo que el juego de valores utilizado como tal debe necesariamente permitir la construcción de un polígono y que éste se ajuste a las proporciones especificadas por la FIRA [ABR05b].

Móviles

Los objetos móviles están representados por la clase abstracta *Dynamic*. Especializaciones de ésta son la pelota (*Ball*) y los robots (*Robot*).

Estos cuerpos pueden permanecer en reposo o en movimiento dependiendo de la interacción externa que pueda afectar su comportamiento y de la interna con los demás objetos del modelo (fijos y móviles)

- *Ball* : Representa la pelota. El modelo geométrico es una esfera (ver 4.3). La posición inicial, el radio y la masa son establecidos al momento de su creación. Permite que se actualice su posición en cualquier momento de la simulación manteniendo en esos casos la velocidad e inercia del momento inmediato anterior. Permite que se afecte su trayectoria alterando el vector velocidad de la esfera que la modela.

- *Robot* : Representa los robots. El modelo geométrico está compuesto por tres geometrías que representan el chasis y las dos ruedas laterales (ver 4.3). La posición inicial, el lado de las caras, la masa, el diámetro de las ruedas y el desplazamiento relativo al centro del eje de las ruedas son establecidos al momento de su creación. Permite que se actualice su posición en cualquier momento de la simulación manteniendo en esos casos la velocidad e inercia del momento inmediato anterior. Permite que se afecte su trayectoria alternando la velocidad de las ruedas.

5.1.3. Actualizaciones

Las actualizaciones de los estados de los objetos del mundo, básicamente el de los cuerpos móviles, es manejado a través de la clase *World* de ODE.

Esta clase es la responsable de hacer que los cuerpos se muevan e interactúen según leyes físicas de forma de aproximar su comportamiento al que tendrían en el mundo real. Por este motivo debe conocer todos y cada uno de los objetos que conforman la simulación.

5.1.4. Colisiones

Las colisiones entre objetos del modelo son resueltas por dos clases de ODE, *Collision* y *Contact*.

La clase *Collision* tiene la responsabilidad de detectar cómo resolver las colisiones entre objetos. En primera instancia genera una lista de puntos de contacto donde quedan representados todos aquellos puntos en los que hubo una colisión entre *dos* objetos. Estos objetos son representados por la clase *Contact* que brinda la posibilidad de incidir externamente en la resolución de la colisión, permitiendo modificar parámetros de la misma (p.ej.: coeficiente de rozamiento) [Smi04]. Finalmente, una vez que se personalizaron los parámetros de los puntos de contacto detectados, la clase *Collision* resuelve la colisión determinando los nuevos estados (posición y rotación) y velocidades de los objetos.

5.2. Interacción con el entorno

La interacción con el sistema se define a través de la interfaz pública descrita en 5.1.1. En el diagrama de secuencia de la figura 9 se muestra la interacción con el sistema mostrando la invocación de las operaciones de la interfaz en orden cronológico.

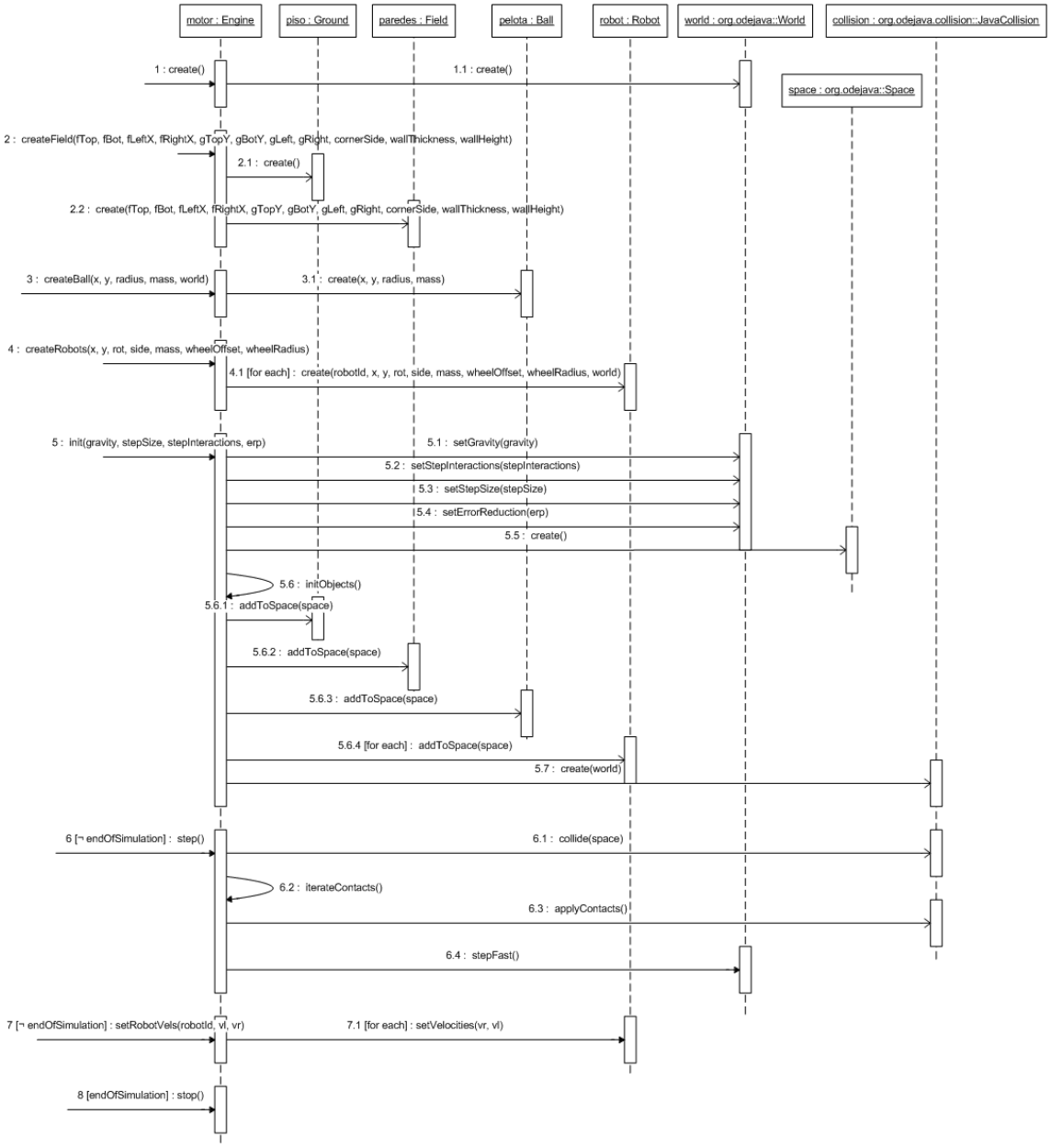

Figura 9: Diagrama de secuencia.

5.2.1. Etapas de la simulación

La simulación puede dividirse básicamente en tres etapas:

1. Inicialización: Inicializa el modelo del mundo. Incluye los pasos del 1 al 5 del diagrama de la figura 9. De los cinco pasos los únicos que pueden realizarse en diferente orden son los 3 y 4 que corresponden a la creación de la pelota y los robots respectivamente. El paso donde se configuran los parámetros que afectan la simulación (*init*) sólo debe realizarse después de la creación de todos los objetos que participan en la simulación.

2. Simulación: Ciclo de simulación. Incluye los pasos 6 y 7 del diagrama de la figura 9. En el paso 6 se avanza el mundo un paso de tiempo y en el paso 7 se actualizan las velocidades de las ruedas de los robots. Además de estos pasos, podrían utilizarse las operaciones que permiten alterar explícitamente el estado de los cuerpos móviles. Esto sería útil para propósitos de pruebas sobre las trayectorias.

3. Finalización: Finaliza la simulación liberando los recursos utilizados en el modelo de mundo. Incluye el paso 8 del diagrama de la figura 9.

6. Conclusiones

La investigación permite concluir que el uso de simulación para realizar pruebas, entrenamiento o aprendizaje de algoritmos de navegación y comportamiento es amplio y ha motivado el desarrollo de varias iniciativas (ver 2.1). Entre ellas, algunas se diferencian en el paradigma de programación, otras en el manejo de las colisiones (librerías o métodos utilizados), otras en las prestaciones que brindan al usuario final, pero todas coinciden en enfatizar la importancia de lograr que la simulación sea una imagen lo más fiel posible de la realidad.

En tal sentido, los trabajos que basaron el manejo de colisiones en un modelo geométrico presentan una debilidad importante en común: no resuelven correctamente las colisiones entre robots. Por otro lado, los que se basaron en la utilización de librerías físicas para resolver los problemas cinemático y dinámico de los cuerpos, presentan resultados que parecen aceptables a simple vista.

Otra conclusión que surge a partir de la investigación es que los desarrollos exitosos implicaron mucho tiempo de trabajo e investigación, correspondiendo, en la totalidad de los casos estudiados, a proyectos de grado o fin de carrera. Ergo, desarrollar un simulador completo, que brinde al usuario diferentes formas de interacción con el mundo simulado, que permita realizar grabaciones y reproducciones de partidos y que pueda ser configurado para que, en forma programática, ejecute partidos o jugadas repetidas veces, es un arduo trabajo que no debe ser subestimado.

El resultado que describe esta documentación es un prototipo que fue desarrollado y documentado en algo más de tres meses. A pesar de ello, constituye en sí mismo una prueba de la viabilidad del modelo elegido para la simulación así como el punta pie inicial para futuros desarrollos que continúen esta linea de trabajo.

No se realizaron pruebas suficientemente exhaustivas, ni en cantidad ni en calidad, por lo que no se puede concluir demasiado sobre la fidelidad de la simulación respecto a la realidad. A pesar de ello, la simulación presenta la continuidad esperada respecto a los movimientos, las colisiones entre los objetos del mundo son detectadas y resueltas en su totalidad, y el manejo de los recursos es correcto, lo que hace que la simulación sea sustentable en el tiempo, sin que se hayan detectado bajas en el desempeño de la misma.

6.1. Trabajo futuro

Para poder utilizar este motor de simulación en la categoría SimuroSot es necesario ajustar los parámetros del modelo hasta lograr que los movimientos de los cuerpos sea similar al descrito en el simulador oficial. Esta linea de trabajo podría entenderse como refinado y testeo de la simulación.

Respecto al modelo geométrico de los robots, teniendo en cuenta el problema comentado en la sección 4.3.4, sería conveniente profundizar el conocimiento de la librería ODE para realizar otra implementación del primer modelo propuesto con el que no se obtuvieron resultados favorables, pero que no se descarta como una mejor aproximación.

Otra linea de trabajo es el desarrollo de una herramienta que potencie las posibilidades de testeo y desarrollo de estrategias de navegación o de comportamiento de los robots. Para ello es necesario construir un *front end* que ofrezca una interfaz amigable al usuario y que cumpla con otros requisitos como: interacción con estrategias, grabación y reproducción de partidos, ejecución

programática de partidos o escenarios preestablecidos, avance paso a paso, aceleración del paso de tiempo, entre otras.

Finalmente, de estas características, las dos últimas pueden tener gran impacto en la obtención de mejores resultados tanto a nivel de algoritmos de navegación como estrategias de juego, ya que permitirían una mejor compresión y evaluación de la información de cada cuadro de simulación (paso a paso) y viabilizar la utilización de técnicas de aprendizaje automático (aceleración del paso de tiempo). Pueden abordarse en si mismas como otra línea de trabajo debido a la naturaleza compleja de los problemas que requieren resolver, manejo de la inercia en el primer caso y manejo más robusto de las colisiones en el último.

Referencias

[ABR05a] Gustavo Armagno, Facundo Benavides, and Claudia Rostagnol, *Evaluación de un prototipo de simulador simil fira, desarrollado con la herramienta phi*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. 5, 7

[ABR05b] ———, *Reglas para simurosot*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. 6, 14, 18

[ABR05c] ———, *Simulador gsim*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. 4, 7

[Bea03] Randal W. Beard, *Ball prediction for robot soccer*, Tech. report, Department of Electrical & Computer Engineering, Brigham Young University, Utah, USA, Jan 2003. 7

[BT03] Brett Browning and Erick Tryzelaar, *Übersim: A multi-robot simulator for robot soccer*, International Conference on Autonomous Agents archive, Proceedings of the second international joint conference on Autonomous agents and multiagent systems, Melbourne, Australia (2003), School of Computer Science, Carnegie Mellon University, Pittsburgh, USA. 7

[Cas03] Alvaro Castroman, *Manual del simulador gsim*, Tech. report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2003. 4

[DF03] Hans Dollen and Wim Fikkert, *Mis20 : The robotic soccer simulator*, Master's thesis, TKI Department, University of Twente, Enschede, The Netherlands, Jun 2003. 6, 7

[ea98] Jean-Paul Laumond et al, *Robot motion planning and control*, LAAS-CNRS, 1998, ISBN 3-540-76219-1. 7, 12

[KLKZ03] G. Klancar, M. Lepetic, R. Karba, and B. Zupancic, *Robot soccer collision modelling and validation in multi-agent simulator*, Mathematical and Computer Modelling of Dynamical Systems (2003), Faculty of Electrical Engineering, University of Ljubljana, Ljubljana, Slovenia. 7

[LBP06] Sean Luke, Gabriel Catalin Balan, and Liviu Panait, *Gmu evolutionary computation labs multi-agent simulator (mason)*, Feb 2006. 7

[MMSV02] Javier Mínguez*, Luis Montano*, and José Santos-Victor**, *Reactive navigation for non-holonomic robots using the ego-kinematic space*, International Conference on Robotics & Automation, Proceedings of the 2002 IEEE, Washington, USA (2002), 3074–3080, *Dept. of Computer Science and Systems Engineering, Instituto de Sistemas e Robótica, University of Zaragoza, Spain. **Instituto Superior Técnico, Portugal. 12

[RHK96] Robert Resnick, David Halliday, and Kenneth S. Krane, *Física*, CECSA, 1996, ISBN 968-26-1230-6. 12

[Smi04] Russell Smith, *Open dynamics engine v0.5 user guide*, May 2004. 7, 8, 16, 19

[TCMC05] W. Turek, D. Czyrnek, R. Marcjan, and K. Cetnarowicz, *Roboss - an universal tool for robots modelling and simulation*, Computer Methods and Systems, Knowledge Engineering - Methods and Applications, Crakow (2005), Institute of Computer Science, AGH University of Science and Technology, Krakow. 7

[Tho03] Christian Thompson, *Building a 2d physics engine for mason*, Master's thesis, Evolutionary Computation Laboratory, George Mason University, USA, 2003. 7

[TPG+02] Fernando Torres, Jorge Pomares, Pablo Gil, Santiago T. Puente, and Rafael Aracil, *Automática robótica. robots y sistemas sensoriales*, Prentice Hall, 2002, ISBN 84-205-3574-5. 3, 12, 13

[vdL05] Joost van der Linden, *Dynamic avoidance control of soccer playing mini-robots*, Master's thesis, Distributed and Embedded Systems Group (DIES), Department of Computer Science, University of Twente, The Netherlands, Dec 2005. 6