Información Tecnológica
Vol. 28(5), 55-64 (2017)
doi: 10.4067/S0718-07642017000500008

Simulación Cinemática de un Robot Seguidor de Línea para el Desarrollo del Videojuego de Programación Rusty Roads en el Framework Unity

Oscar F. Gómez y Urbano E. Gómez*
Facultad de Ingeniería de Sistemas e Informática, Universidad Pontificia Bolivariana (UPB), Autopista a Piedecuesta Km 7, Edificio L, Oficina 301, Bucaramanga, Colombia (e-mail: oscar.gomezs@upb.edu.co; urbano.gomez@upb.edu.co)

* Autor a quien se debe ser enviada la correspondencia

Recibido Mar. 2, 2017; Aceptado Abr. 24, 2017; Versión final Jun. 29, 2017, Publicado Oct. 2017

Resumen

Se presentan los resultados del proyecto de investigación *Rusty Roads*, un videojuego que brinda a sus jugadores un marco de trabajo para el diseño y programación de robots seguidores de línea y presenta una interfaz tridimensional animada para simular el comportamiento en un entorno competitivo. Durante el desarrollo del proyecto se establecieron los factores clave de diseño físico que pueden afectar el desempeño de un seguidor de línea, se hizo una caracterización de un robot existente para obtener parámetros realistas en la simulación y se establecieron ecuaciones discretas para simular el movimiento en el *framework* de desarrollo *Unity*, verificando su precisión con ecuaciones continuas exactas establecidas. Este artículo presenta el proceso de obtención y prueba de dichas ecuaciones, que fueron la base de la capa de simulación del videojuego y permitieron el desarrollo posterior del componente de visualización 3D.

Palabras clave: videojuegos; simulación; robótica; seguidores de línea; mecánica; robot

Kinematic Simulation of a Line Follower Robot for the Creation of the Programming Videogame Rusty Roads in the Unity Framework

Abstract

This paper presents the results of a research project for developing Rusty Roads, a videogame that simulates line follower robots in tridimensional, competitive environments. During the software development process, it was necessary to define the key physical factors that could affect the performance of the robot. A real robot was considered to define real parameters for the simulation, and to define discrete equations for simulating the robot in the Unity framework, comparing their accuracy with that of continuous, exact equations specifically developed for this study. This paper presents the process for obtaining and testing those equations, which were the foundations for the successful development of the simulation layer of the videogame and allowed the subsequent creation of its 3D visualization component.

Keywords: video games; simulation; robotics; line followers; mechanics; robot

INTRODUCCIÓN

Un seguidor de línea es un robot que tiene la capacidad de desplazarse sobre una trayectoria dada mediante el seguimiento completamente autónomo de una ruta indicada visualmente (en algunas variantes, las rutas se indican con información no visual sino, por ejemplo, magnética), generalmente en forma de una línea negra sobre un fondo blanco o viceversa (Restrepo *et al.*, 2009). El funcionamiento de un seguidor de línea se soporta en la implementación de un controlador que incorpora un algoritmo definido por el diseñador del robot. Dicho controlador –cuya complejidad puede variar entre un circuito analógico sencillo hasta un programa ejecutándose en un microcontrolador, una tarjeta Arduino o un computador– debe basarse en las lecturas de uno o más sensores para determinar si la trayectoria del robot es adecuada o no y, en consecuencia, ajustar la velocidad de rotación de los motores que están acoplados a las ruedas del robot para mantenerlo sobre la línea que se está siguiendo (Montiel *et al.*, 2015).

Como ejemplo, la Figura 1 ilustra el instante en que uno de los sensores (verde) capta la línea, mientras que el otro (rojo) no lo hace. El controlador del robot podría usar esa información para concluir que el robot no debería ir en línea recta porque se saldría de la línea por la derecha, y corregiría el rumbo haciendo que el motor derecho gire más rápido que el izquierdo.

Fig. 1: Funcionamiento de un seguidor de línea de dos sensores.

Los seguidores de línea suelen utilizarse en ámbitos académicos como una forma interesante de enseñar nociones de robótica, sistemas de control o inteligencia artificial. Así, por ejemplo, la realización de un seguidor de línea puede incorporar el aprendizaje de programación de microcontroladores, control de velocidad por modulación de ancho de pulso (PWM), aplicación de lógica difusa, algoritmos genéticos o control PID, entre otras temáticas (Restrepo *et al.,* 2009; Diaz, 2004; Sitoula, 2014).

El desarrollo de un robot requiere del adecuado manejo de diversos factores mecánicos, ambientales y económicos que pueden influir seriamente en su desempeño. El material del robot, la calidad de los motores y sensores, el nivel de carga de las baterías e incluso la luz medioambiental pueden marcar la diferencia entre un robot operativo y uno no funcional. Dependiendo de las habilidades que quieran desarrollarse puede ser deseable el tener que manejar estos factores; las habilidades se generan a partir de experimentos como los propuestos en la plataforma de prácticas en robótica en donde los aprendices deben implementar programas para mover el robot evitando varios obstáculos durante el recorrido (Payá *et al*, 2007). Sin embargo, si lo que se desea es enfocarse en mejorar habilidades de algoritmia y control, lo ideal sería removerlos por completo.

Los autores desarrollaron un videojuego consistente en un simulador de seguidores de línea en el que los robots de los participantes compiten entre sí en un entorno tridimensional animado. El software remueve una gran cantidad de factores físicos difíciles de controlar y se enfoca en que los jugadores diseñen los algoritmos de sus robots para que optimicen el seguimiento lo más posible. El juego busca brindar un marco de trabajo común para el desarrollo de habilidades de programación y la creación de algoritmos de seguimiento de alta calidad que puedan compararse de manera objetiva y precisa, y que incluso pueda servir como herramienta de apoyo durante la creación de un robot físico.

El videojuego fue desarrollado en Unity ya que brinda numerosas ventajas sobre desarrollar el videojuego sin él, una de ellas, de enorme relevancia para el tema de este artículo, es el manejo automático del tiempo de la simulación física por parte del framework ya que se encarga automáticamente de que el código de simulación física se ejecute en pasos iguales de tiempo, independientemente de la cantidad de cuadros por segundo a los que se esté ejecutando el juego, desacoplando efectivamente la simulación del componente gráfico del juego. Para usar el videojuego, el jugador inicialmente debe crear el controlador de su robot en el lenguaje de programación LUA –un lenguaje de *scripting* ampliamente utilizado en la industria de videojuegos (Llopis, 2012)-, así como realizar un diseño básico de la estructura física del robot consistente en indicar la distancia entre sus ruedas y la posición de sus sensores. Una vez listo el robot, éste puede cargarse en el juego, que simula su comportamiento en una pista (que también puede ser diseñada por los jugadores). La visualización de la simulación puede ser básica –mostranudo la pista tal cual es: una línea

negra sobre fondo blanco como se observa a la izquierda de la Figura 2– o inmersiva, como se observa a la izquierda –representando a los robots en una carrera de alta velocidad de hasta 3 participantes simultáneos en un escenario tridimensional.

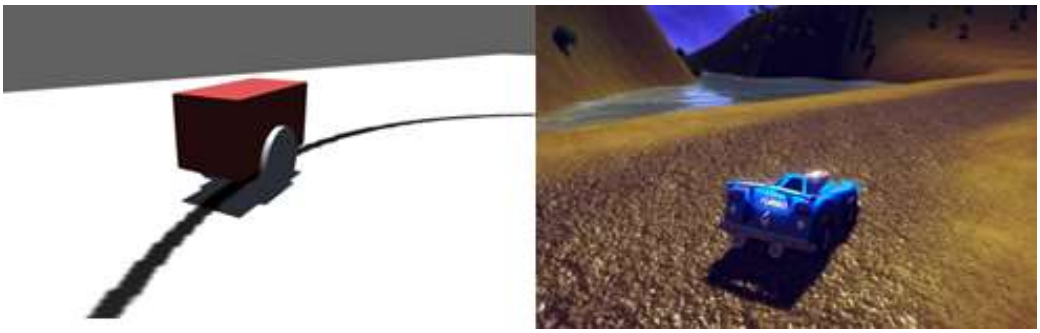

Fig. 2: Simulación con visualización básica.

Una de las primeras fases del desarrollo del videojuego consistió en el establecimiento de la manera como se simularía el comportamiento de los robots. Este artículo presenta una revisión completa de las consideraciones que se tuvieron en cuenta y los experimentos que se llevaron a cabo durante dicha fase, así como de las conclusiones de diseño a las cuales se llegó al momento de implementar el videojuego.

DESARROLLO DE LA PROPUESTA DE SIMULACIÓN

En la simulación del videojuego se minimizaron los factores físicos que afectaran el movimiento del robot. Así, se estableció que las ruedas de los robots siempre rodarían sin deslizar y que el peso del robot se ignoraría, de modo que no se tendrían en cuenta los efectos de la inercia ni la inercia rotacional en su movimiento. Como consecuencia adicional de ignorar el peso, se decidió modelar el movimiento del robot cinemática y no dinámicamente, lo cual implica que el movimiento se definiría a partir de la aplicación directa de aceleraciones (lineales o angulares), sin analizar ni tener en cuenta las causas del movimiento, como fuerzas o momentos. Como se verá más adelante, estas restricciones permitieron desarrollar simulaciones lo suficientemente realistas para el juego.

Otras restricciones tuvieron que ver con el diseño físico del robot. En la realidad, un robot puede ser construido con una cantidad ilimitada de variantes. Por ejemplo, puede tener cualquier forma física, usar cualquier cantidad de ruedas de diversas características o disponer de cualquier cantidad de sensores ubicados de manera arbitraria, entre otras. Esas variaciones podrían afectar significativamente el desempeño de un robot. Debido a ello, podría parecer importante que los jugadores tuviesen que realizar diseños de los robots que deban tener esos factores en cuenta. Sin embargo, la opinión de los autores es que sólo ciertos factores clave aportan significativamente al desarrollo de algoritmos de seguimiento interesantes, y sólo dichos factores deberían ser tenidos en cuenta en el modelado, mientras que los demás deberían simplemente restringirse.

Como ejemplo, la Fig. 3 muestra una fotografía de un seguidor de línea Dynabot II. En la imagen derecha se observa que el robot dispone de dos ruedas alineadas en un mismo eje, dos sensores infrarrojos y una rueda de apoyo omnidireccional, todo acoplado a una base de acrílico. Factores como la posición de la rueda de apoyo –cuya única función es actuar como punto de apoyo para mantener al robot estable y permitir su libre desplazamiento y rotación– o la forma de la base de acrílico son poco relevantes al momento de diseñar y simular: se puede asumir que los jugadores no ubicarían la rueda de apoyo en donde no cumpla su función y que nunca darían a la base una forma que dificulte la rotación o movimiento.

Fig. 3: Robot Dynabot II.

También se analizó el hecho de que un robot real podría tener cualquier cantidad de ruedas de radios distintos y ubicaciones arbitrarias. Sin embargo, de nuevo, es muy probable que los jugadores eviten ubicar las ruedas de maneras absurdas y darles radios distintos, que sólo dificultarían la labor de seguimiento. Debido a ello, y teniendo en cuenta que dos ruedas son suficientes para controlar el movimiento del robot por completo a la vez que se minimiza el peso de éste, y que no es común encontrar robots que usen más ruedas, se definió que los robots simulados tendrían dos ruedas de igual tamaño acopladas a un mismo eje, igual que el Dynabot II. La longitud del eje, esto es, la distancia entre las ruedas, sí sería definible por el jugador al diseñar su robot, ya que afecta directamente la manera como el robot gira y no tiene un valor óptimo por sí solo, de manera que los jugadores pueden variarla para lograr comportamientos distintos y dar un mejor control a su algoritmo.

Con los sensores hubo una cantidad considerable de flexibilidad: los jugadores pueden diseñar robots que hagan uso de cualquier cantidad de sensores (que, por supuesto, puede limitarse como parte de las reglas de una competición o torneo que se lleve a cabo usando el videojuego), ubicados en cualquier posición con respecto a las ruedas. La decisión de permitir que los jugadores ubiquen los sensores libremente se debió al hecho de que el posicionamiento y la cantidad de sensores de un robot son importantes para maximizar el efecto de un buen algoritmo, de modo que un diseño flexible en ese sentido seguramente permitirá la creación de resultados interesantes, e incluso el estudio de algoritmos avanzados de seguimiento. En cuanto al tipo de sensores, el juego sólo permite el uso de sensores que captan el color del suelo que esté justo debajo de ellos.

Finalmente, y con el fin de que los robots simulados tuviesen un comportamiento cercano a la realidad, varias características técnicas del robot (velocidad de giro máxima y aceleración angular de los motores, radio de las ruedas, etc.) se definieron a partir de una completa caracterización que se hizo al Dynabot II por medio de mediciones físicas directas y cálculos realizados a partir de procesamiento cuadro a cuadro de videos del robot funcionando (Miranda & Gamboa, 2016). Una vez definido todo lo anterior, se procedió a desarrollar el modelo de simulación.

Modelo de ecuaciones físicas

Una forma de realizar la simulación es mediante el establecimiento de un conjunto de ecuaciones que puedan evaluarse para obtener el estado exacto del robot (posición y orientación) en cualquier instante de tiempo. Desafortunadamente, como se analizará más adelante, establecer un modelo completo de este tipo no siempre es viable. A pesar de esto, y con el fin de contar con una referencia exacta que pudiese usarse para evaluar otras formas de realizar la simulación, se realizó un modelo exacto con la restricción de que las aceleraciones del sistema serían nulas, esto es, que las ruedas girarían a velocidad angular constante. Bajo la restricción mencionada, el robot se moverá en trayectorias circulares, como se muestra en la Fig. 4:

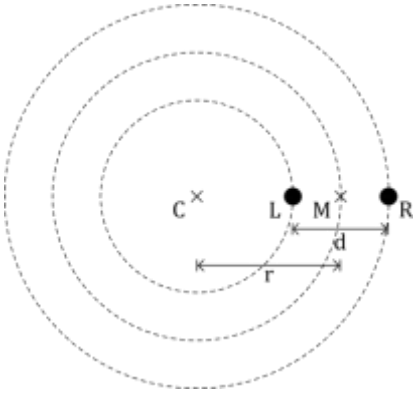

Fig. 4: Trayectoria de un robot con ruedas girando a velocidad constante, con $\omega_r > \omega_l > 0$.

En la ilustración, el frente del robot empieza apuntando hacia arriba. Los puntos L y R representan la posición de las ruedas izquierda y derecha del robot, que están separadas por una distancia d, y M es el punto medio entre éstas. La trayectoria del robot tendrá un centro de rotación en un punto C, ubicado a una distancia r del punto M; dicha distancia depende de las velocidades angulares ω_l y ω_r de las dos ruedas, y puede variar entre 0 (cuando el robot giraría sobre su propio eje, lo cual sucede cuando ambas ruedas giran a la misma velocidad pero en sentido contrario, $\omega_l = -\omega_r$) e infinito (cuando el robot se mueve en línea recta, lo cual sucede cuando ambas ruedas giran a la misma velocidad, $\omega_l = \omega_r$). El sentido de giro del robot será antihorario como el mostrado si $\omega_r > \omega_l$, y horario (hacia la derecha) si $\omega_l > \omega_r$.

Con las observaciones anteriores en mente, se procedió a analizar el comportamiento de la orientación del robot, esto es, hacia dónde apunta el robot, sin tener en cuenta la ubicación del mismo. Este análisis se realizó estableciendo un marco de referencia con origen ubicado en la posición de la rueda izquierda. Con respecto a dicho marco de referencia, la rueda izquierda no se desplaza y la rueda derecha describe una ruta circular alrededor del origen, como se muestra en la
Fig. 5.

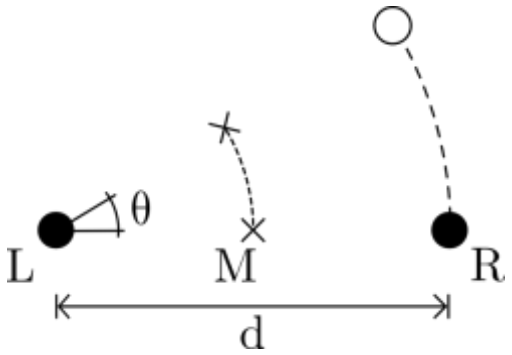

Fig. 5: Movimiento del sistema respecto a un marco de referencia en la rueda izquierda (L).

Obsérvese que el tiempo que tarda el robot en dar un giro completo es el mismo que tarda la rueda derecha en recorrer la circunferencia que describe su trayectoria en este marco de referencia. Dicha circunferencia corresponde al perímetro de un círculo de radio d, esto es, $2\pi d$.

Ahora bien, dado que la rueda gira a velocidad angular ω_r, su velocidad lineal en el marco de referencia absoluto será $v_r = \omega_r r$, donde r es el radio de la rueda, mientras que en el marco de referencia relativo a L será igual a $v_r - v_l$, donde, por analogía, $v_l = \omega_l r$. Lo anterior implica que la rueda da toda la vuelta en una cantidad de tiempo t.

$$t = \frac{2\pi d}{v_r - v_l}. \tag{1}$$

Por su parte, el robot está girando a velocidad angular ω. Ello significa que da una vuelta completa de 2π radianes en un tiempo.

$$t = \frac{2\pi}{\omega}. \tag{2}$$

Igualando y despejando para ω:

$$\omega = \frac{v_r - v_l}{d}. \tag{3}$$

Dado que la velocidad angular es la variación del ángulo de orientación por unidad de tiempo, es posible definir la ecuación 4 para la orientación de un robot con orientación inicial θ_0 en cualquier instante de tiempo t:

$$\theta(t) = \theta_0 + \frac{v_r - v_l}{d} t. \tag{4}$$

Analizando a continuación el movimiento del robot en el marco de referencia absoluto, es viable analizar M –el punto medio entre las ruedas del robot–, que gira alrededor de un centro de rotación a velocidad angular constante. Es posible, por tanto, modelar la trayectoria del punto usando las ecuaciones paramétricas que describen un movimiento circular (Knisley, 2001):

$$\begin{aligned} x(t) &= r\cos\big(\theta(t)\big) + x_c, \\ y(t) &= r\sin\big(\theta(t)\big) + y_c, \end{aligned} \tag{5}$$

donde $\theta(t)$ es el ángulo de giro, que en este caso estará dado por la ecuación 4, r es la distancia del M al centro de rotación, y (x_c, y_c) es la ubicación del centro de rotación.

El valor de r se puede calcular a partir de la relación entre velocidad lineal y angular, $v = \omega r$, si se tiene en cuenta que, en el marco de referencia absoluto, la velocidad lineal de M es igual a $(v_r + v_l)/2$ y su velocidad angular es igual a la velocidad angular del sistema. Así,

$$r = \frac{v}{\omega} = \frac{d(v_r + v_l)}{2(v_r - v_l)}. \tag{6}$$

Finalmente, las coordenadas del centro de rotación se obtienen a partir de la posición inicial del robot (x_0, y_0), su orientación inicial θ_0 y la distancia r:

$$\begin{aligned} x_c &= x_0 - r\cos(\theta_0), \\ y_c &= x_0 - r\sin(\theta_0). \end{aligned} \tag{7}$$

Reemplazando y factorizando:

$$x(t) = x_0 + \frac{d(v_r + v_l)}{2(v_r - v_l)}\left(\cos\left(\theta_0 + \frac{v_r - v_l}{d}t\right) - \cos(\theta_0)\right), \tag{8}$$

$$y(t) = y_0 + \frac{d(v_r + v_l)}{2(v_r - v_l)}\left(\sin\left(\theta_0 + \frac{v_r - v_l}{d}t\right) - \sin(\theta_0)\right). \tag{9}$$

Nótese que es necesario dar un tratamiento especial al caso en que $v_r - v_l = 0$, el cual ocurre cuando las dos ruedas giran a la misma velocidad angular y, por lo tanto, $v_r = v_l = v$. En este caso, el robot se mueve en línea recta en dirección perpendicular a la orientación del eje de las ruedas, y el centro de rotación está a una distancia infinita. Para esta situación, las ecuaciones de movimiento pueden derivarse directamente a partir de la velocidad del robot y su orientación, ambas constantes, teniendo en cuenta que la velocidad es la variación de la posición por unidad de tiempo:

$$x(t) = x_0 + vt\cos\left(\theta_0 + \frac{\pi}{2}\right), \tag{10}$$

$$y(t) = y_0 + vt\sin\left(\theta_0 + \frac{\pi}{2}\right). \tag{11}$$

Las ecuaciones (4), (8) y (9) coinciden con las obtenidas por Lucas (2000) por medio del planteamiento de las ecuaciones diferenciales que rigen un sistema como el analizado aquí para su posterior integración simbólica. Adicionalmente, Lucas observa que en el caso especial $v_r = v_l$, las ecuaciones producen indeterminaciones del tipo $0/0$, por lo que es viable utilizar la regla de l'Hôpital para encontrar sus respectivos límites, correspondientes a las ecuaciones (10) y (11).

La limitante del modelo matemático descrito en esta sección, que sólo es aplicable cuando las ruedas del robot giran a velocidad angular constante, hace que el modelo sea poco útil para simular un robot autónomo, dado que éste variará continuamente la velocidad de giro de sus ruedas mientras intenta cumplir su objetivo de seguir la línea del recorrido propuesto. Añadir el componente de aceleración a las ecuaciones mediante un análisis cinemático como el desarrollado no es viable dado lo irregular de la trayectoria que sigue un robot cuando sus ruedas varían su velocidad. De hecho, como Lucas comenta en (Lucas, 2000), si se añade el componente de aceleración a las ecuaciones diferenciales que describen la posición del robot, el resultado son integrales que forzosamente deben resolverse mediante métodos numéricos.

Simulación Numérica

Una segunda aproximación para modelar el comportamiento del robot consistió en desarrollar una simulación numérica. En este tipo de simulación, se crea un modelo compuesto por las ecuaciones diferenciales básicas del sistema que se quiere simular, y éstas posteriormente se evalúan en pasos discretos que representan pequeños cambios de tiempo (esto es, se convierten en diferencias finitas). Es de este modo, simulando paso a paso, que funcionan los motores de física que se utilizan para desarrollar videojuegos o animaciones (Millington, 2007). De hecho, la creación de la simulación numérica de este proyecto equivale a la creación de un motor de física de características particulares.

Para empezar a desarrollar el modelo de simulación, se plantearon las ecuaciones básicas de movimiento y rotación del robot. En cualquier instante de tiempo dado, el robot tendrá una orientación θ y se estará moviendo a una velocidad $\boldsymbol{v}$ como se muestra en la Fig. 6:

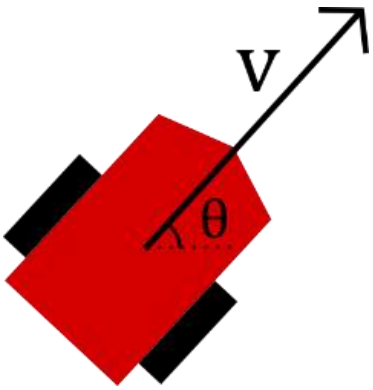

Fig. 6: Velocidad y orientación de un robot en movimiento.

Adicionalmente, en el mismo instante de tiempo, el robot estará rotando a una velocidad angular ω. Tanto v como ω pueden calcularse a partir de las velocidades angulares de las dos ruedas y la distancia entre éstas siguiendo el mismo análisis realizado en la sección 0: $v = (v_l + v_r)/2$ y $\omega = (v_l - v_r)/d$. Ahora bien, la velocidad angular ω equivale a la variación del ángulo de orientación por unidad de tiempo, esto es,

$$\omega = \frac{d\theta}{dt}, \tag{12}$$

lo cual puede aproximarse por medio de una ecuación de diferencias:

$$\omega = \frac{\Delta\theta}{\Delta \mathrm{t}}. \tag{13}$$

En la ecuación de diferencias anterior, Δt representa una cantidad pequeña de tiempo y $\Delta\theta$ representa cuánto varió θ en dicho tiempo. Dado que una variación equivale a la diferencia entre un valor nuevo y uno anterior,

$$\omega = \frac{\theta_{nuevo} - \theta_{anterior}}{\Delta t}, \tag{14}$$

de manera que el nuevo ángulo θ_1 puede calcularse así:

$$\theta_{nuevo} = \theta_{anterior} + \omega\Delta t. \tag{15}$$

Lo anterior significa que es posible calcular cuál será el nuevo ángulo de orientación del robot en un instante de tiempo dado a partir del ángulo actual, la velocidad angular y la duración de dicho instante de tiempo, lo cual implica que es posible obtener el comportamiento de dicha orientación a lo largo del tiempo si se inicia con una orientación inicial y se aplica la ecuación anterior sucesivamente, una y otra vez. El análisis puede aplicarse de manera análoga a la posición y la velocidad para obtener el conjunto de ecuaciones 16:

$$\begin{aligned} x_{nuevo} &= x_{anterior} + v\,cos(\theta)\,\Delta t \\ y_{nuevo} &= y_{anterior} + v\,sin(\theta)\,\Delta t \end{aligned} \tag{16}$$

En esencia, ejecutar las ecuaciones repetitivamente equivale a realizar una integración numérica sobre las ecuaciones: se está encontrando un valor de posición o rotación a partir de valores instantáneos de velocidad lineal o angular. Más en particular, el proceso equivale a la aproximación de la integral por medio del método de Euler. La exactitud del resultado de la integración, es decir los valores de ubicación y orientación del robot, dependerá directamente del tamaño del intervalo dt: entre menor sea el valor del intervalo, mayor será la precisión de la integración. Ello significa que, por ejemplo, es mucho más exacto ejecutar 10 veces el código anterior con un intervalo $dt = 0.1$ que ejecutarlo una sola vez con $dt = 1$ (Fig. 7). El problema radica en que un intervalo menor de tiempo implica una mayor cantidad de ejecuciones de código para simular una misma cantidad de tiempo: simular 10 segundos de tiempo usando un intervalo de 0.1 segundos requiere que el código de simulación se ejecute 100 veces, mientras que hacerlo con un intervalo de 1 segundo requiere únicamente 10 ejecuciones.

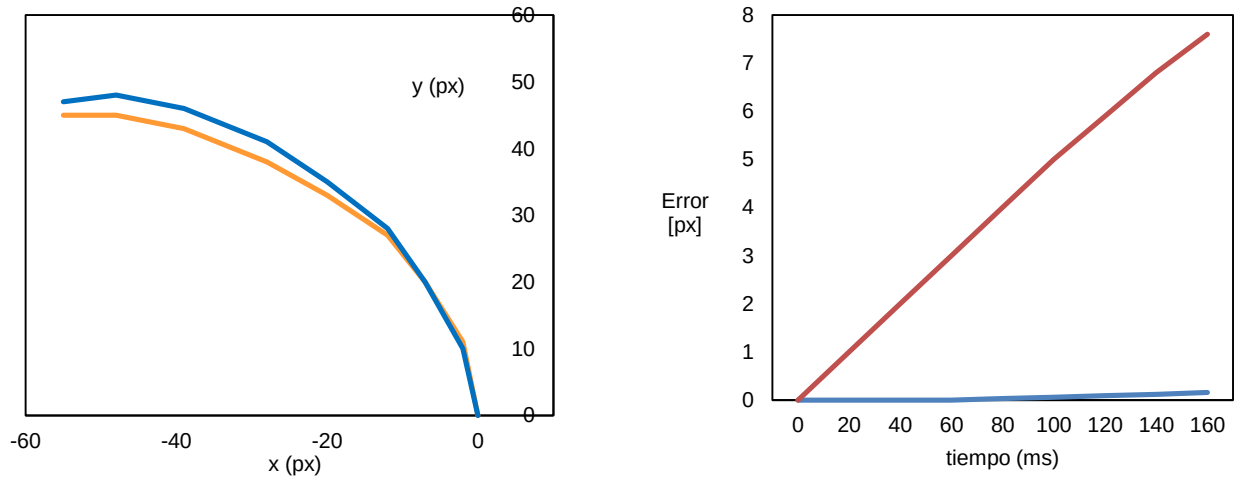

Fig. 7: Movimiento simulado mediante integración con el método de Euler, con $dt = 20\,ms$ (rojo) y $dt = 0.4\,ms$ (negro). Izquierda: posición del robot en dos dimensiones. Derecha: error absoluto de la posición en comparación con el modelo matemático exacto.

Ahora bien, existen otros métodos de integración numérica que brindan niveles de exactitud considerablemente mayores que el del método de Euler al evaluarse con iguales intervalos de tiempo, que tendrían la ventaja de producir simulaciones con buenos niveles de exactitud con menos tiempo de cómputo. Tal es el caso de la integración por medio de la regla de Simpson, que establece (Süli & Mayers, 2003) lo siguiente:

$$\int_a^b f(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right). \quad (17)$$

La regla de Simpson es un caso particular del método de Runge Kutta de cuarto orden (RK4) (Sülly *et al*., 2003), que, al igual que el método de Euler, permite aproximar soluciones de ecuaciones diferenciales de la forma $\dot{y} = f(x, y)$, siendo $\dot{y} \equiv \partial y/\partial x$. En efecto, RK4 se reduce a la regla de Simpson si la ecuación que se quiere resolver es de la forma $\dot{y} = f(t)$, esto es, f es independiente de y, tal y como sucede en este caso, dado que la aceleración es independiente de la velocidad y la velocidad es independiente de la posición.

Para adaptar la regla a la simulación, debe tenerse en cuenta que a representa el momento de tiempo en el que se encuentre la simulación, de manera que $f(a)$ es el valor que tenga el dato a integrar (por ejemplo, la velocidad del robot) en ese momento en la simulación, y que $a - b$ representa el ancho del intervalo de integración, que en este caso es igual a dt, la duración del paso. Dado que para avanzar la simulación un paso sólo importa la duración del paso y no los valores inicial y final del tiempo, es posible establecer $a = 0$ y $b = dt$:

$$\int_0^{dt} f(x)dx = \frac{dt}{6}\left(f(0) + 4f\left(\frac{dt}{2}\right) + f(dt)\right). \quad (18)$$

Simulación numérica con aceleración

Los modelos descritos hasta ahora no incluyen un componente de aceleración. Añadirlo es trivial en la integración con el método de Euler: basta con actualizar las velocidades angulares de las dos ruedas con base en la aceleración angular de cada una. Actualizar el modelo con integración por regla de Simpson es más complejo. En efecto, para integrar los valores de posición (x y y), se requiere calcular tres valores de ángulo θ y velocidad v, uno para cada momento de tiempo 0, $dt/2$ y dt. A su vez, para calcular cada uno de esos valores de θ por medio de la regla de Simpson, se requiere de tres valores de ω, y algo similar sucede con el cálculo de los valores de v y ω. La creación de funciones que calculen todos esos valores intermedios facilita considerablemente esta aplicación "en cadena" de la regla de Simpson.

CONCLUSIONES

Las consideraciones tomadas respecto al modelo físico de los robots permitieron disminuir la complejidad de la simulación sin reducir las capacidades del videojuego de diseñar y probar robots seguidores de línea y

de este modo usarse como herramienta para el aprendizaje. Sin embargo, existen potenciales mejoras que podrían desarrollarse para incrementar el realismo físico de la simulación y permitir que el comportamiento simulado se acerque aún más al comportamiento de un robot real que ejecute un algoritmo equivalente, tales como la incorporación a la simulación de los efectos de la inercia y la inercia rotacional de los robots.
La utilización de una simulación física abre posibilidades interesantes para la creación de escenarios con elementos que serían difíciles o imposibles de crear en la realidad, como puertas que se abren a distancia, paredes que pueden romperse o puntos de teletransportación que reubiquen a los robots de manera instantánea sin afectar su cantidad de movimiento.

El videojuego desarrollado está compuesto por dos capas de software principales: la simulación (*back-end*) y la visualización (*front-end*). Ambas se desarrollaron de forma que estuviesen desacopladas, de modo que es posible, por ejemplo, desactivar la visualización y ejecutar simulaciones enteras de manera inmediata, lo cual es útil para estudiar los resultados de un algoritmo de inmediato, mientras que la conjunción de ambas capas brinda la experiencia de usuario completa y brinda motivación para el desarrollo de habilidades de programación.

Varias de las decisiones tomadas por los autores al momento de diseñar el simulador no sólo afectan la calidad de la simulación, sino que también definieron (o removieron) mecánicas del videojuego final. Aunque todas las decisiones de diseño se tomaron intentando prever las dinámicas que surgirían una vez las personas utilizaran el juego, la única forma cierta de comprobar si dichas decisiones fueron acertadas o no es mediante la futura realización de pruebas (*playtesting*) (Hunicke *et al.*, 2004), que son fundamentales si se quieren realizar juegos de calidad (Fullerton, 2008) (Schell, 2014).

La metodología descrita en este artículo permitió la creación de la capa de simulación, pero otras consideraciones tuvieron que tenerse en cuenta para desarrollar la visualización, algunas de estas también fueron físicas. El ejemplo más claro se encuentra en el manejo del eje vertical de la visualización: en la simulación, los robots siempre se mueven sobre un plano paralelo al suelo, pero en la visualización pueden moverse en tres dimensiones; el movimiento sobre el eje vertical ocurre cuando el terreno tiene ascensos o descensos. Lo mismo sucede con las rotaciones: en la simulación, el robot sólo rota alrededor del eje vertical, pero en la visualización puede rotar alrededor de los otros dos ejes para adaptarse a la inclinación del terreno.

Fig. 8: Ejemplo de inclinación y altura. La representación *wireframe* (líneas negras) permite visualizar las inclinaciones del terreno.

REFERENCIAS

Diaz, J.C. On the Straight and Narrow. The University of Tulsa. (En la web: https://goo.gl/a6Zjwq), 12 de octubre de 2004. Acceso: 25 de noviembre de 2016 (2004)

Fullerton, T. Game Design Workshop: A Playcentric Approach to Creating Innovative Games, 2ª Ed., Burlington, Morgan Kaufmann, Estados Unidos (2008)

Hunicke, R., M. Leblanc, y R. Zubek. MDA: A formal approach to game design and game research. Proceedings of the Challenges in Games AI Workshop, 19th National Conference of Artificial Intelligence, Menlo Park, California, Estados Unidos (2004)

Knisley, J. Multivariable Calculus Online. East Tennessee State University. (En la web: https://goo.gl/SsyTth), 21 de marzo de 2001. Acceso: 25 de noviembre de 2016 (2001)

Llopis, N. 14th Annual Front Line Awards, Game Developer Magazine, 19 (1), 43-52 (2012)

Lucas, G. W. A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators. (En la web: https://goo.gl/dzaF1T), septiembre 12 de 2013. Acceso: 11 de octubre de 2016 (2013)

Millington, I. Game Physics Engine Development, Morgan Kaufmann Publishers, San Francisco, California, Estados Unidos (2007)

Miranda, J.D., y C.A. Gamboa. Caracterización del Sistema Robótico Seguidor de Línea Avanzado con plataforma Arduino: Dynabot II, sin publicar (2016)

Montiel, H., Jacinto, E., y Martínez, F. Generación de Ruta Óptima para Robots Móviles a partir de Segmentación de Imágenes, Información Tecnológica, 26(2), 145-152 (2015)

Payá, L., Reinoso, O., Gil, A., Jiménez, L. Plataforma Distribuida para la realización de prácticas de Robótica Móvil a través de Internet, Información Tecnológica, 18 (4), 27-38 (2007)

Restrepo, C., J. Molina, y C. Torres. Algoritmo genético para la ubicación óptima de sensores en un robot seguidor de línea. Scientia *et* Technica, Universidad Tecnológica de Pereira, 41(1), 81-93 (2009)

Schell, J. The Art of Game Design: A Book of Lenses, CRC Press, Boca Ratón, Florida (2014)

Sitoula, A. PID Tutorials for Line Following. Let's Make Robots. (En la web: https://goo.gl/WomS6n), abril 16 de 2014. Acceso: 11 de octubre de 2016 (2014)

Süli, E., y D. Mayers. An Introduction to Numerical Analysis, Cambridge University Press, Nueva York, Estados Unidos (2003)