



**GRADO EN INGENIERÍA EN SISTEMAS DE
TELECOMUNICACIÓN**

Curso Académico 2019/2020

Trabajo Fin de Grado

**Motor de físicas mejorado para simulador robótico
basado en tecnologías web**

Autor : Natalia Monforte Rodríguez

Tutor : Dr. José María Cañas Plaza

Agradecimientos

Con la finalización de este trabajo termina una de las mejores etapas y más importantes de mi vida. Ha sido una etapa de muchos cambios, de aprendizaje continuo y de conocer a muchas personas que han acabado siendo una parte muy importante de mi vida.

En primer lugar quería agradecer a mi tutor, José María, la posibilidad que me ha dado para poder acercarme al mundo de la robótica. También quería agradecer a todo el equipo de *Kibotics* la predisposición y el apoyo que han ofrecido en todo momento a todos los alumnos que estamos realizando el TFG. Especial agradecimiento a Rubén. Gracias a él, los primeros meses fueron mucho más fáciles.

En segundo lugar, agradecer a JCUBATAS todas las fiestas, comidas y excursiones al río que hemos hecho a lo largo de estos años. Sin vosotros, los años en la uni no habrían sido iguales. Gracias chicos.

También dar las gracias a mis padres y a mi hermana por el apoyo incondicional, por hacerme ver desde muy pequeña lo bonito que es aprender y por estar a mi lado siempre (sobre todo antes de un examen, que tiene más mérito). Y gracias también a ti, Greta. La que más paciencia ha tenido mientras le contaba todo lo que había memorizado antes de un examen.

Gracias, abuelo. Gracias por alegrarte casi más que yo con cada sobresaliente. Por enseñarme a no conformarme nunca. Por motivarme a crecer y mejorar siempre. Ojalá hubieras visto todo lo que he llegado a conseguir. Este trabajo es de los dos.

Por último, quería agradecértelo a ti, Juankar. Gracias por ser mi compañero de estudio y de vida todos estos años. Por ir de excursión a ver nodos y no parar de aprender juntos. Por animarme, apoyarme y confiar siempre.

Perseverance.

Resumen

Este Trabajo Fin de Grado está enfocado en la mejora de la plataforma de robótica educativa *Kibotics*. Esta plataforma está dirigida a niños y adolescentes de todas las edades y utiliza un simulador llamado *WebSim*, basado en *A-Frame*, para representar en tres dimensiones los escenarios de los ejercicios ofrecidos. Los ejercicios pueden solucionarse tanto en *Scratch* como en *Python* y, además, las soluciones que se desarrollan en el simulador pueden utilizarse también en los robots físicos.

En particular, este proyecto se centra en la mejora del motor de físicas de *WebSim* para dotar al simulador de un mayor realismo. Para ello, se ha diseñado e implementado un nuevo motor complementario que se encarga de materializar la fuerza autónoma de los robots de la escena y que coexiste con *CANNON*, el motor por defecto de *A-Frame*, ya que *CANNON* materializa la fricción, la gravedad y las colisiones. Además, también se han añadido varios novedosos ejercicios a la plataforma con los que se pueden explotar las mejoras que ofrece el nuevo motor de físicas implementado, incluyendo una aceleración finita realista y sensibilidad a robots con masa grande o pequeña, todo ello configurable. Estos ejercicios han servido de validación experimental.

La implementación del software del motor de físicas se ha realizado en *JavaScript* en su totalidad. Además, también se ha tratado con el lenguaje *JSON* para modificar los ficheros de configuración de los escenarios de los ejercicios.

Índice general

Lista de figuras	9
Lista de tablas	11
1. Introducción	1
1.1. Robótica	1
1.2. Tecnologías web	3
1.2.1. Tecnologías web en el lado del cliente	6
1.2.2. Tecnologías web en el lado del servidor	6
1.3. Docencia robótica	7
2. Objetivos	13
2.1. Objetivos	13
2.2. Metodología	14
2.3. Plan de trabajo	15
3. Herramientas	17
3.1. Lenguaje <i>JavaScript</i>	17
3.2. Lenguaje <i>HTML</i>	18
3.3. Lenguaje <i>JSON</i>	20
3.4. <i>Blender</i>	21
3.5. Entorno <i>A-Frame</i>	22
3.5.1. Sistema de físicas de <i>A-Frame</i>	23
3.6. Plataforma <i>Kibotics</i>	25

4. Mejora de las físicas en WebSim	29
4.1. Estudios previos: motor de físicas por defecto en <i>A-Frame</i>	29
4.1.1. Gravedad	30
4.1.2. Colisiones	30
4.1.3. Fricción	34
4.2. Motor de físicas actual para robots en <i>WebSim</i>	36
4.3. Nuevo motor de físicas complementario	38
4.3.1. Diseño	38
4.3.2. Modelo de fuerzas	41
4.3.3. Controlador PD	42
4.3.4. Timing	46
4.3.5. Implementación	49
4.4. Validación experimental	54
4.4.1. Simulación realista de robots terrestres	54
4.4.2. Simulación realista de drones	58
5. Nuevos ejercicios con físicas realistas	61
5.1. Sigue-líneas con rampa	61
5.2. Laberinto 3D para mBot	65
5.3. Laberinto para drone	68
5.3.1. Sin señalización	69
5.3.2. Con señalización	70
5.4. Fútbol competitivo	71
6. Conclusiones	75
6.1. Conclusiones y valoración de resultados	75
6.2. Líneas futuras	78
7. Bibliografía	79

Índice de figuras

1.1. Ejemplos de robots en la actualidad	3
1.2. Ejemplos de aplicaciones web	4
1.3. Ejemplo de una interacción <i>HTTP</i>	5
1.4. Aprendizaje de programación robótica con <i>Lenobotics</i>	8
1.5. Aprendizaje de programación robótica con <i>LEGO education</i>	8
1.6. Aprendizaje de programación robótica con <i>OpenRoberta</i>	9
1.7. Aprendizaje de programación robótica con <i>iRobot</i>	9
1.8. Aprendizaje de programación robótica con <i>Scratch</i>	10
1.9. Interfaz de programación en <i>Kibotics</i> de un ejercicio en <i>Scratch</i>	11
1.10. Robots soportados en la plataforma <i>Kibotics</i>	11
3.1. Interfaz de trabajo con <i>Blender</i>	22
3.2. Ejemplo de construcción 3D con <i>A-Frame</i>	24
3.3. Menú de ejercicios de <i>Kibotics</i>	26
3.4. Editor <i>Scratch</i> en <i>WebSim</i>	26
3.5. Editor <i>Python</i> en <i>WebSim</i>	27
4.1. Colisión elástica	31
4.2. Colisión inelástica	32
4.3. Escenarios de prueba de las colisiones de <i>A-Frame</i>	34
4.4. Diseño del motor de físicas complementario	39
4.5. Controlador PID	43
4.6. Diseño controlador PD en velocidad del plano horizontal	44
4.7. Diseño controlador PD en velocidad del eje vertical	44
4.8. Diseño controlador PD en velocidad angular horizontal para drone	45

4.9. Diseño controlador PD en velocidad angular horizontal para robot terrestre	45
4.10. Diseño controlador PD en posición para la altura	46
4.11. Combinación de los motores de físicas a lo largo del tiempo	49
4.12. Subida de rampa con una fuerza máxima insuficiente	55
4.13. Giro de 90° hacia la izquierda en una superficie con una fricción muy baja	56
4.14. Relación fricción - aceleración	57
4.15. Tiempo - Velocidad Controlador PD en velocidad del plano horizontal	57
4.16. Despegue con drones de diferentes masas	58
4.17. Tiempo - Posición Controlador PD en posición para la altura	59
4.18. Tiempo - Velocidad Controlador PD en velocidad del eje vertical	59
 5.1. Escenario Sigue-líneas con rampa	62
5.2. Fotograma del movimiento del ejercicio Sigue-líneas con rampa	63
5.3. Laberinto 3D para mBot	65
5.4. Fotograma del movimiento del ejercicio Laberinto 3D para mBot	66
5.5. Fotograma del movimiento del ejercicio Laberinto para drone	68
5.6. Laberinto para drone sin señalización	70
5.7. Laberinto para drone con señalización	70
5.8. Fútbol competitivo	71
5.9. Evaluador del ejercicio Fútbol competitivo	72
5.10. Fotograma del movimiento del ejercicio Fútbol competitivo	73

Índice de cuadros

3.1. Parámetros configurables del sistema de físicas de <i>A-Frame</i>	25
4.1. Resultados de las colisiones obtenidos con el escenario 1	33
4.2. Resultados de las colisiones obtenidos con el escenario 2	33
4.3. Resultados de las colisiones obtenidos con el escenario 3	33
4.4. Parámetros que caracterizan el movimiento de un robot autónomo	40
5.1. Parámetros de configuración del modelo de fuerzas y de <i>A-Frame</i> del ejercicio sigue-líneas con rampa	64
5.2. Parámetros de configuración del modelo de fuerzas y de <i>A-Frame</i> del ejercicio laberinto 3D para mBot	67
5.3. Parámetros de configuración del modelo de fuerzas y de <i>A-Frame</i> del ejercicio laberinto para drone	69
5.4. Parámetros de configuración del modelo de fuerzas y de <i>A-Frame</i> del ejercicio fútbol competitivo	73

Capítulo 1

Introducción

En este primer capítulo de la memoria se van a explicar los conceptos clave entorno a los cuales se ha desarrollado este Trabajo de Fin de Grado. Entender qué son la robótica y las tecnologías web y por qué son importantes es fundamental, ya que la combinación de ambos conceptos ofrece nuevas posibilidades para la docencia robótica.

Por otro lado, en este capítulo también se va a introducir el concepto de motor de físicas en los simuladores robóticos, ya que una importante parte del trabajo se ha basado en la generación de un nuevo motor de físicas que permite recrear, con un mayor realismo, los movimientos realizados por los robots en la plataforma web de docencia robótica *Kibotics*.

1.1. Robótica

La robótica es la disciplina que estudia la creación de máquinas automatizadas capaces de recrear comportamientos humanos o animales en función del software que lleven incorporados. Estas máquinas son las que se denominan robots. Un robot presenta dos partes bien diferenciadas: el hardware y el software. En el hardware se encuentran sensores, actuadores y ordenadores y en el software es donde reside la inteligencia.

Haciendo un breve repaso de la historia de los robots, cabe destacar que desde el 85 a.C. ya se empezaron a crear los primeros robots en la Antigua Grecia. En esa época la creación

de robots se basaba en el intento de replicar personas por medio de máquinas. De hecho, esas máquinas ni siquiera se denominaban robots. Este término fue acuñado en 1920 por Karel Čapek como homenaje a su obra teatral *Rossum's Universal Robots*, que trataba de una empresa encargada de fabricar humanos artificiales para facilitar la realización de tareas a los trabajadores de las fábricas. Así, la palabra robot procede de *robbota*, que en checo significa trabajo forzado o servidumbre¹.

Hoy en día, los robots están presentes prácticamente en cualquier ámbito de la vida de cualquier persona. Ya se han creado robots capaces de recrear casi cualquier actividad realizada por el ser humano o que nos facilita la realización de las mismas. Algunos de los robots más populares en la actualidad son los siguientes. En la Figura 1.1 se ofrece una representación gráfica de ellos.

- Robots que se encargan de la limpieza del hogar, como por ejemplo, Roomba. Este tipo de robot ya registra millones de unidades vendidas, provocando una revolución del mercado de electrodomésticos de limpieza.
- Robots de cocina, como por ejemplo, Thermomix. Estos robots son capaces de cocinar cualquier receta que el usuario seleccione. También ha registrado un gran número de unidades vendidas y se considera un avance muy valioso puesto que ayuda al ahorro de tiempo, que está muy valorado hoy en día.
- Drones, como por ejemplo, Tello. Cada vez están surgiendo más y más aplicaciones para los drones como pueden ser la monitorización de cultivos, operaciones de rescate, vigilancia de playas o ayuda en los incendios.
- Coches autónomos, como por ejemplo, Tesla. Este avance ha supuesto una revolución en el mundo automovilístico y, gracias a la llegada del 5G, se van a poder explotar aún más las posibilidades que presenta la conducción autónoma.
- Robots en medicina, como por ejemplo, el Robot DaVinci. Estos robots son capaces de ayudar a los cirujanos durante complicadas intervenciones quirúrgicas con un gran grado de precisión.

¹<https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/>

CAPÍTULO 1. INTRODUCCIÓN

- Robots en el ámbito educativo y lúdico, como por ejemplo, los Robots LEGO. Estos robots permiten acercar la robótica a los niños desde edades muy tempranas, para que aprendan desde el principio la utilidad e importancia de la robótica.



(a) Roomba



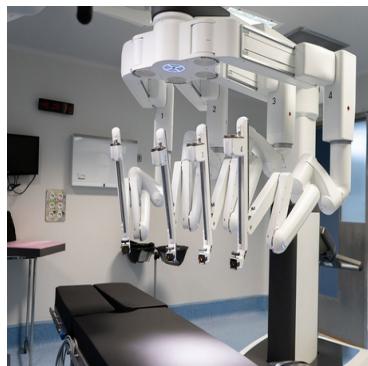
(b) Thermomix



(c) Tello



(d) Tesla



(e) Robot DaVinci



(f) Robots LEGO

Figura 1.1: Ejemplos de robots en la actualidad

1.2. Tecnologías web

Las tecnologías web están en continuo desarrollo. Actualmente, existen tecnologías web tanto en el lado del cliente como en el lado del servidor. La idea de esta separación es marcar las diferentes partes de un sistema software para poder controlarlo de una forma más eficaz. Por este motivo, el *frontend* recoge los datos o interactúa con el usuario final y el *backend* procesa esos datos o sirve los contenidos.

Hoy en día, son muy comunes las aplicaciones web que se basan en estas tecnologías. Las aplicaciones web basan su funcionamiento en un modelo cliente - servidor en el que el cliente se encarga de recoger los datos que introduce el usuario y el servidor los procesa. Gracias a ellas, se puede leer en el navegador web el correo electrónico, ver películas y series, leer el periódico, comprar productos, etc. Algunas de las aplicaciones web con más éxito en los últimos años son *Gmail*, *Spotify*, *Netflix*, *Amazon* o *AliExpress*.

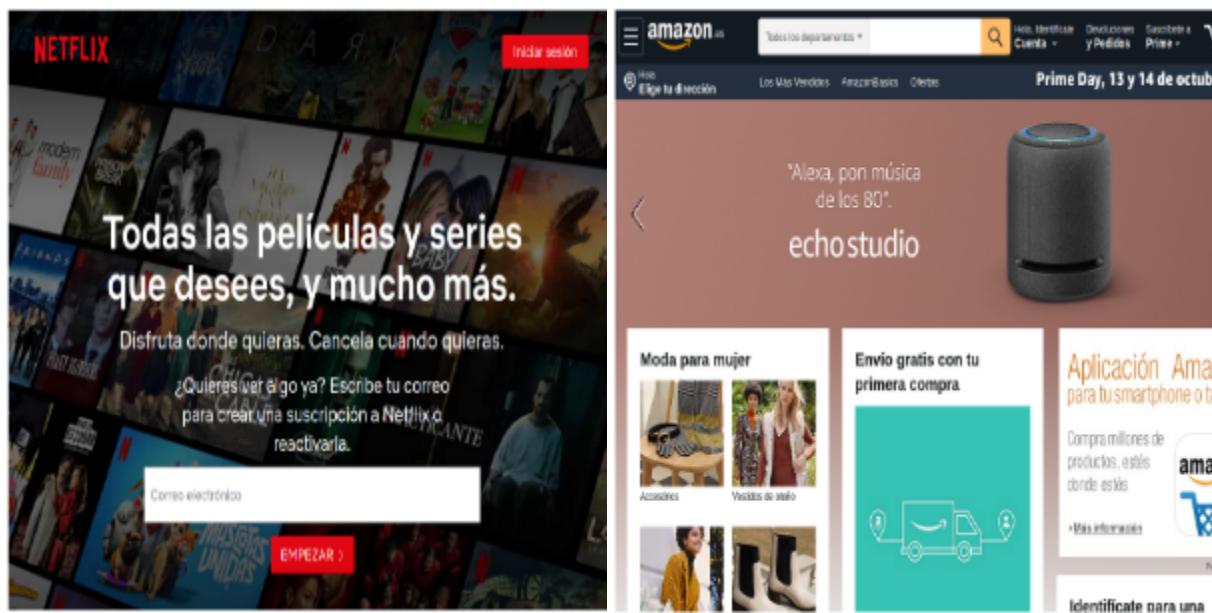


Figura 1.2: Ejemplos de aplicaciones web

Por un lado, el *frontend* engloba todas aquellas tecnologías web del lado del cliente que se encargan de recopilar los datos y mostrárselos al usuario. Principalmente, existen tres tecnologías de *frontend*: *HTML*, *CSS* y *JavaScript*. Estas tres tecnologías permiten al usuario interactuar con el servidor web, utilizando un navegador como intérprete.

Por otro lado, el *backend* se encarga del almacenamiento de información en bases de datos, gestión de servidores y servir las vistas de las páginas web seleccionadas por el desarrollador en el lado del cliente o servir los contenidos multimedia. En el *backend*, el número de tecnologías es más extenso. La programación *backend* incluye lenguajes como *PHP*, *Python*, *Ruby*, *.NET* o *Java* y las bases de datos sobre las que se trabaja habitualmente pueden ser *SQL*, *MongoDB* o *MySQL*.

La comunicación entre cliente y servidor se realiza utilizando el protocolo *HTTP* (protocolo de transferencia de hipertexto). Este protocolo funciona mediante la emisión de una serie de peticiones y respuestas entre el cliente y el servidor usando diferentes métodos. Existen varios métodos *HTTP*, los más comunes son los siguientes²:

- **GET:** solicitud de datos de un recurso concreto.
- **PUT:** reemplazo de las representaciones actuales del recurso de la petición.
- **POST:** envío de datos a un recurso concreto, normalmente provocando el cambio de estado del servidor.
- **DELETE:** eliminación de un recurso.
- **HEAD:** solicitud de datos de un recurso concreto tal y como ocurre con el método *GET*, pero la respuesta no incluye cuerpo.

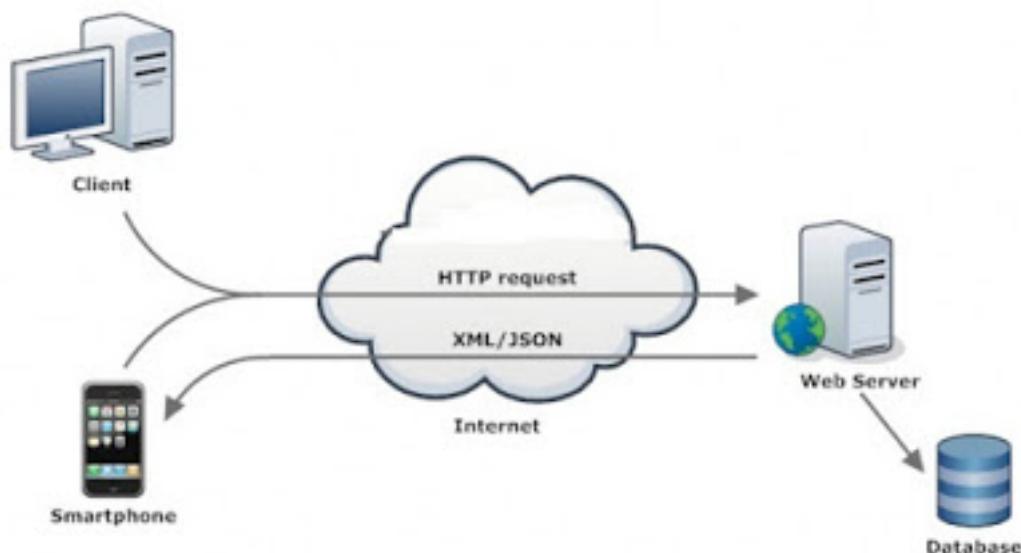


Figura 1.3: Ejemplo de una interacción *HTTP*³

²<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

³<http://dsanchezz.blogspot.com/2015/10/rest.html>

1.2.1. Tecnologías web en el lado del cliente

Las tres tecnologías web del *frontend* que permiten la interacción entre usuario y servidor web son las siguientes:

- *HTML*: es un lenguaje de marcado que permite diferenciar los contenidos y definir la estructura de un sitio web. Permite dividir una página web en diferentes secciones: títulos, texto, imágenes, pie de página, etc. Es la base de toda página web. Sigue el modelo de objetos *Document Object Model (DOM)*, que permite que el contenido de una página esté disponible para los programas en *JavaScript*. Hoy en día, los navegadores emplean la versión *HTML5*, que introduce como mejora las etiquetas de vídeo, de audio, de diálogos entre personas, de secciones y de bloques de contenidos.
- *CSS*: es un lenguaje de hojas de estilo que permite modificar la apariencia de una página web. Gracias a este lenguaje, se puede organizar mejor el código al poder separar los datos (lenguaje *HTML*) del diseño (lenguaje *CSS*). Hoy en día, la versión disponible en los navegadores es *CSS3*.
- *JavaScript*: es un lenguaje de programación interpretado que permite definir el comportamiento de una página web (por ejemplo, al hacer click en un enlace). Por ello, gracias a este lenguaje el usuario puede interaccionar con la página web.

1.2.2. Tecnologías web en el lado del servidor

En el lado del servidor, algunas de las tecnologías que permiten acceder a bases de datos, escalabilidad, gestionar los servidores y servir las páginas web mediante la utilización de plantillas son:

- *Node.js*: uno de los lenguajes frecuentemente utilizado en la programación del *backend* es el lenguaje *JavaScript*. *JavaScript* se creó para su uso en el *frontend* en un principio; sin embargo, gracias al motor *node.js*, este lenguaje puede ser interpretado en el lado del servidor sin necesidad de un navegador.
- *Django*: es un entorno web de alto nivel programado en *Python*. Es muy rápido y fácil de utilizar. Además, dispone de una interfaz para el manejo de las bases de datos *SQL* y se puede acceder a ellas en el código mediante instrucciones *Python*.

- *Spring*: entorno basado en *Java* que simplifica el desarrollo de aplicaciones. Facilita las tareas de configuración y el despliegue en el servidor.

1.3. Docencia robótica

Las tecnologías web ofrecen nuevas posibilidades para la robótica. Una de estas posibilidades es la docencia robótica. La docencia robótica ha estado creciendo en los últimos años gracias a que se trata de un campo de gran utilidad y que es una manera atractiva y divertida de introducir a los niños en la tecnología. Hoy en día existen muchas iniciativas para promover la docencia robótica. De hecho, ya se ha introducido en el currículum oficial de secundaria en muchas comunidades autónomas de España y en otros países del mundo.

En consecuencia, especialmente en los últimos años, han surgido algunas plataformas dedicadas a la docencia robótica que se encargan de impartir cursos de iniciación a la robótica desde edades muy tempranas para conseguir que los niños se sientan atraídos por esta rama desde el principio y aprendan a pensar como verdaderos programadores desde muy pequeños.

La docencia robótica comparte los fundamentos de la educación *STEM* (*Science, Technology, Engineering and Mathematics*). Esta educación promueve el aprendizaje de la ciencia, tecnología, ingeniería y matemáticas y fomenta el desarrollo de un pensamiento crítico y la creatividad.

Un ejemplo de estas plataformas que se comentan es el de *Lenobotics*⁴. *Lenobotics* es un programa que se encarga de impartir cursos de robótica en centros educativos para desarrollar las habilidades cognitivas de los niños que resultan necesarias para la programación.

⁴<https://lenobotics.com/>



Figura 1.4: Aprendizaje de programación robótica con *Lenobotics*

Por su parte, *LEGO education*⁵ también ofrece a los más pequeños la posibilidad de iniciarse en la robótica a través de sus kits. Estos kits permiten aprender unas primeras nociones de electrónica, robótica y programación.



Figura 1.5: Aprendizaje de programación robótica con *LEGO education*

⁵<https://education.lego.com/es-es>

CAPÍTULO 1. INTRODUCCIÓN

Otro ejemplo es el de *OpenRoberta*⁶, que ofrece una interfaz de programación por medio de bloques de texto para diferentes modelos de robot. También dispone de una sección con ejercicios resueltos de muestra, para que los más inexpertos puedan empezar a familiarizarse con ellos.

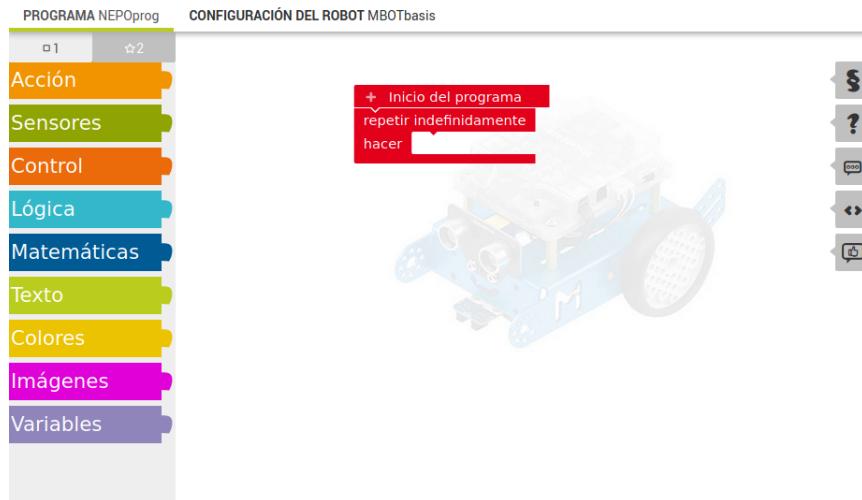


Figura 1.6: Aprendizaje de programación robótica con *OpenRoberta*

*iRobot*⁷ ofrece la posibilidad tanto de crear modelos de robot como de programar el cerebro de esos robots. Este entorno hace especial hincapié en la enseñanza *STEM* como mejor método de aprendizaje de robótica.



Figura 1.7: Aprendizaje de programación robótica con *iRobot*

⁶<https://lab.open-roberta.org/>

⁷<https://edu.irobot.com/>

*Scratch*⁸ es un lenguaje de programación visual que fue desarrollado por el Grupo *Lifelong Kindergarten del MIT Media Lab*. Hoy en día, se utiliza frecuentemente en la educación de niños y adolescentes, ya que permite el aprendizaje de la programación sin tener un amplio conocimiento del código. Ofrece al usuario la posibilidad de programar construyendo una secuencia de código a partir de diversos bloques de acciones. El programador es capaz de construir la secuencia de código con rapidez y facilidad, ya que cada bloque incluye una secuencia de texto que explica la función que desempeña. Por ello, la secuencia de código finalmente podrá ser leída e interpretada como si de un texto se tratase.

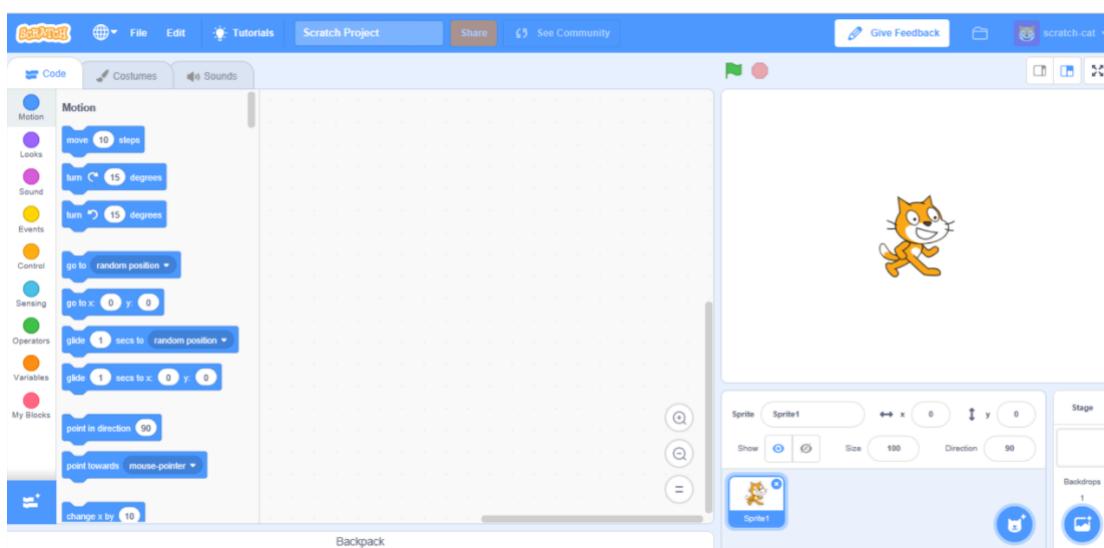


Figura 1.8: Aprendizaje de programación robótica con *Scratch*

El presente trabajo se va a centrar en la plataforma *Kibotics*⁹, la cual es un entorno web para docencia en robótica y programación que permite a niños y adolescentes aprender programando. *Kibotics* apuesta por una enseñanza principalmente práctica, ya que resulta mucho más atractiva tanto la enseñanza como el aprendizaje de este modo que únicamente con clases teóricas.

Kibotics se basa en la utilización del simulador *WebSim* que, a su vez, está basado en la tecnología *A-Frame* para representar los mundos. Este simulador permite la creación de numerosos diferentes ejercicios para los robots que tienen soporte en la plataforma: piBot, mBot, fórmula 1 y drone Tello. Estos ejercicios podrán solucionarse tanto en lenguaje *Scratch* (especialmente

⁸<https://scratch.mit.edu/>

⁹<https://kibotics.org/>

CAPÍTULO 1. INTRODUCCIÓN

indicado para aquellos alumnos que no hayan programado anteriormente) como en lenguaje *Python*¹⁰ (para aquellos niños que cuenten con nociones previas).

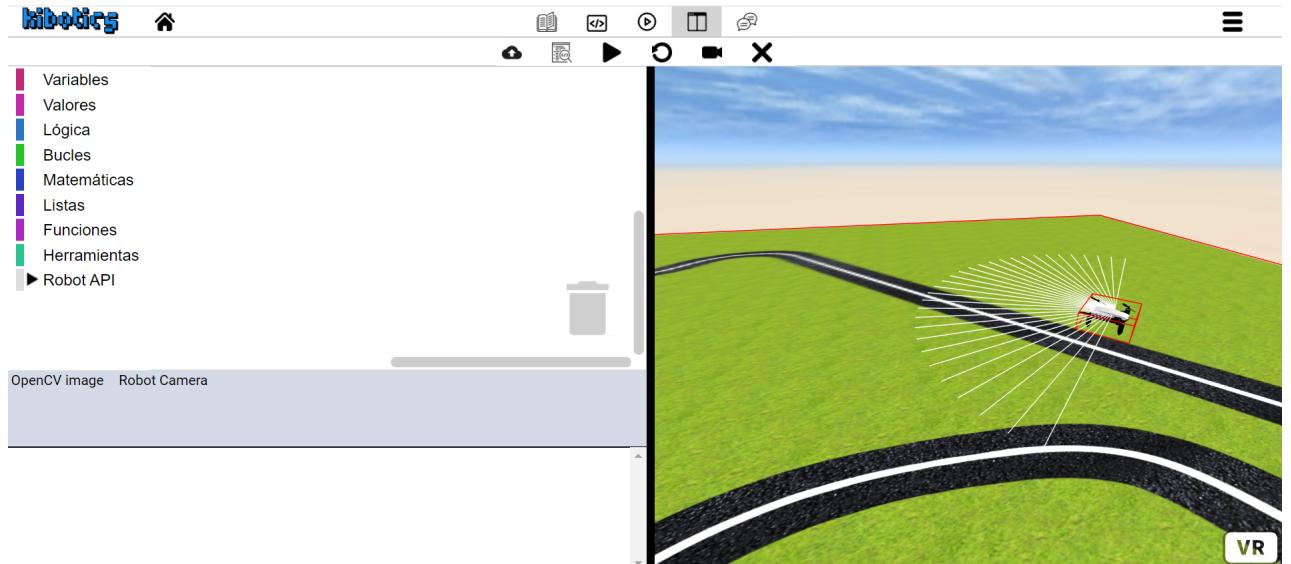


Figura 1.9: Interfaz de programación en *Kibotics* de un ejercicio en *Scratch*

Muchos robots cuentan con cámaras incorporadas en el hardware, lo que permite la creación de ejercicios que se deben solucionar mediante la utilización de la visión artificial además de los ejercicios que se puedan solucionar mediante el uso de los sensores de los robots (sensores infrarrojos, por ejemplo). La Figura 1.10 muestra los robots que soporta actualmente la plataforma.

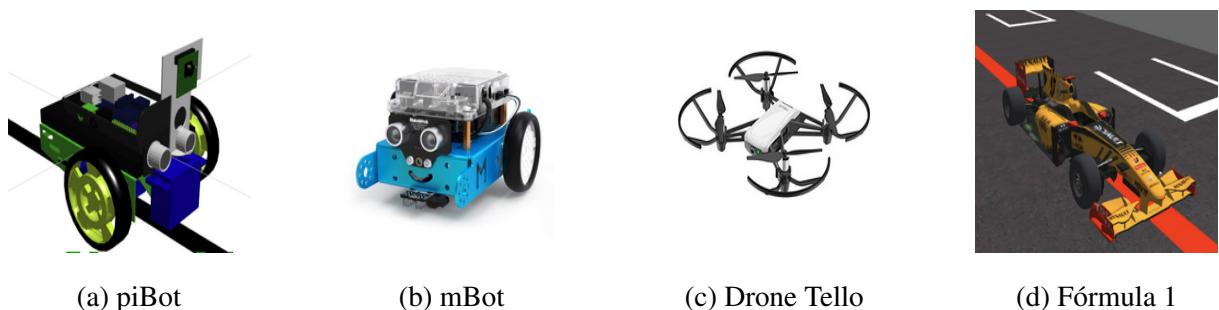


Figura 1.10: Robots soportados en la plataforma *Kibotics*

Por último, cabe destacar la importancia de los motores de físicas que incorporan los simu-

¹⁰<https://es.python.org/>

ladores robóticos usados tanto en docencia como en investigación robótica. Un motor de físicas es un software capaz de realizar simulaciones de ciertos sistemas físicos como la dinámica del cuerpo en movimiento, la fricción y la elasticidad de una colisión. Se emplean con mucha frecuencia en los videojuegos, para recrear con un mayor realismo el movimiento de los personajes.

Existen numerosos motores de físicas como *Box2D*¹¹ (simulaciones 2D), *Cocos2D*,¹² (simulaciones 2D) *Ammo.js*¹³ (simulaciones 3D) o *CANNON*¹⁴ (simulaciones 3D). El presente trabajo se va a centrar en este último, ya que es el que emplea *A-Frame* en la actualidad.

En el capítulo 3, se explicará con mayor detalle las peculiaridades de este motor de físicas en cuestión.

¹¹<https://box2d.org/>

¹²<https://www.cocos.com/>

¹³<https://github.com/kripken/ammo.js/>

¹⁴<http://schteppe.github.io/cannon.js/>

Capítulo 2

Objetivos

En este capítulo se explican los objetivos del presente trabajo, la metodología que se ha seguido para alcanzarlos y la planificación que se ha llevado durante el proceso de desarrollo.

2.1. Objetivos

En los mundos de los simuladores que se emplean en docencia robótica hay objetos estáticos, móviles y los propios robots programados por los niños. Los objetivos que persiguen este trabajo son los siguientes:

- Desarrollar un motor de físicas basado en *A-Frame* que permita replicar de modo realista el movimiento autónomo de los robots programados por los estudiantes de la plataforma *Kibotics* y que se complemente con *CANNON*, el motor por defecto que materializa la gravedad, rozamiento y los choques.
- Crear varios ejercicios en la plataforma educativa *Kibotics* que saquen partido del nuevo motor de físicas y sean vistosos, incluyendo sus escenarios y que sirvan de validación experimental.

Además, estos objetivos deben satisfacerse cumpliendo los siguientes requisitos:

- Materialización de robots con distinta masa y un movimiento autónomo realista, con una aceleración máxima limitada y capacidad de control acotada.

- Coexistencia con el motor por defecto *CANNON* que no requiera la modificación de su código fuente.

2.2. Metodología

Con el fin de asegurar el correcto desarrollo del Trabajo de Fin de Grado se estableció una reunión semanal con el tutor para compartir los progresos realizados durante la semana y en la que el tutor me pudo orientar sobre dónde dirigir los esfuerzos. Paralelamente a las reuniones semanales, también se ha contado con un canal de slack en el que se encuentran todos los contribuyentes de la plataforma *Kibotics* donde se han podido plantear todo tipo de dudas durante el proceso de aprendizaje.

También se ha elaborado un blog en el que se han ido compartiendo los resultados y el trabajo que se ha realizado cada semana. El blog se ha implementado gracias al dominio gratuito que ofrece GitHub para crear un blog¹ basado en GitHub Pages. En el README de mi Github se ha incluido un enlace pinchable para acceder a dicho blog².

El modelo de desarrollo software seleccionado ha sido el método en cascada. En primer lugar, se han ido marcando subobjetivos de implementación. Después de finalizar un subobjetivo, se pasaba a una fase de verificación de las funcionalidades y corrección de errores tras la cual se reanudaba el proceso de desarrollo. Las fases en las que se ha dividido la implementación realizada han sido:

- Implementación del movimiento lineal del drone (eje vertical).
- Implementación del movimiento lineal para robots terrestres en el plano horizontal.
- Implementación de la estabilización del drone cuando se encuentra inmóvil durante el vuelo.
- Implementación del movimiento angular para robots terrestres y drones.

¹<https://roboticslaburjc.github.io/2019-tfg-natalia-monforte/>

²<https://github.com/RoboticsLabURJC/2019-tfg-natalia-monforte>

Para integrar el código de las mejoras o aportaciones realizadas al código fuente de la plataforma *Kibotics*, cabe destacar que se ha utilizado el sistema que ofrece GitHub para integrar código mediante la creación de incidencias, de nuevas ramas y de parches. Para que los desarrolladores pudiesen añadir las nuevas funcionalidades al código fuente oficial de *Kibotics*, se creaba una nueva rama actualizada con los últimos cambios de la rama principal. Sobre esta rama se desarrollaba la solución a cada incidencia. Una vez incluidos los cambios se explicaban en un comentario o parche y se subían a la nueva rama creada del repositorio de *Kibotics*. El siguiente paso consistía en solicitar la fusión de los cambios de esta rama con la rama principal, abriendo peticiones *pull request*. Tras la solicitud de la fusión del parche los desarrolladores que cuentan con más experiencia verifican que los cambios son correctos y, si es así, integran los cambios a la rama maestra oficial, dando por resuelta la incidencia. Los comandos necesarios para realizar la integración del código a la rama creada son los siguientes:

```
git checkout -b issue-XXX  
git add -ruta-del-fichero-a-añadir  
git commit -m "Comentario para el commit"  
git push -u origin issue-XXX
```

2.3. Plan de trabajo

El proceso de elaboración del Trabajo de Fin de Grado se ha dividido en cinco fases distintas:

- **FASE 1:** aprendizaje y primera toma de contacto con las tecnologías web necesarias para la elaboración del trabajo. Especialmente *A-Frame* y *JavaScript*.
- **FASE 2:** estudio del código de *Kibotics-WebSim* y de las físicas de *A-Frame* (motor de *CANNON*).
- **FASE 3:** Elaboración de un motor de físicas complementario para el simulador *WebSim*.
- **FASE 4:** creación de los primeros mundos utilizando las funcionalidades proporcionadas por *Blender* y *A-Frame* y creación de nuevos ejercicios para incluir en la plataforma.

Capítulo 3

Herramientas

En este capítulo se explica con un mayor grado de detalle qué herramientas han sido necesarias para el desarrollo del trabajo. Principalmente, se han utilizado los lenguajes de programación *JavaScript*, *HTML*, *JSON*, *Python* y *Scratch*. Por otro lado, se han utilizado aplicaciones como *Blender* para el modelado de objetos 3D y el simulador *WebSim* para la recreación de los mundos tridimensionales.

3.1. Lenguaje *JavaScript*

JavaScript es un lenguaje de programación interpretado de alto nivel que se encuentra bajo el estándar *ECMAScript*¹. Este lenguaje es comúnmente conocido por su uso en los scripts de las páginas web. No obstante, dada su orientación a objetos y a ser un lenguaje de programación basada en prototipos y de un solo hilo, es usado en otros muchos entornos externos de la página web: *Node.js*, *Apache CouchDB* o *Adobe Acrobat*².

La sintaxis es similar a la utilizada en *Java* y *C++*. De esta manera, se facilita el aprendizaje del lenguaje ya que está basado en conceptos ya conocidos por el programador. Las estructuras del lenguaje, tales como sentencias condicionales (if y switch) y bucles (while y for), funcionan de una manera similar a como lo hacen en otros lenguajes de programación³.

¹Especificación de lenguaje de programación en el que se definen tipos dinámicos y soporte de programación orientada a objetos

²<https://developer.mozilla.org/es/docs/Web/JavaScript>

³<https://developer.mozilla.org/es/docs/Web/JavaScript>

Las siguientes características son las principales:

- Lenguaje estructurado similar a la estructura utilizada en *Java* y *C++*.
- *ECMAScript 2015* añadió la palabra clave *let*, que permite que el alcance de la variable se corresponda con el bloque en el que esta se haya definido (*block scoping*).
- Tipado débil, es decir, el tipo de datos se asocia al valor de la variable en un preciso momento.
- El lenguaje está formado por objetos.
- Lenguaje interpretado, es decir, se compila justo-a-tiempo. No es necesario disponer de un compilador adicional, cada navegador incluye un intérprete que se encarga de ejecutar el código.

Este TFG está enteramente programado en *JavaScript*, ya que se trata de una aplicación web que corre en el lado del cliente.

3.2. Lenguaje *HTML*

HTML es un lenguaje de marcado que define la estructura de una página web. *HTML* ofrece una serie de elementos que permiten etiquetar diferentes partes de una misma página web en una misma clase para otorgarles una misma apariencia. Además, *HTML* permite cambiar el estilo de las palabras (por ejemplo, a cursiva, a negrita, agrandar o reducir el tamaño de letra, etc)⁴.

Las partes principales del elemento *HTML* son las siguientes:

- **Etiqueta de apertura:** se trata del nombre del elemento y se incluye entre paréntesis angulares (<>). Indica el inicio del elemento.
- **Etiqueta de cierre:** similar a la etiqueta de apertura salvo que incluye, además, una barra de cierre (/) precediendo al nombre de la etiqueta. Indica el fin del elemento.

⁴https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics

- **Contenido:** todo aquello que se incluye entre la etiqueta de apertura y la de cierre.
- **Elemento:** es el conjunto formado por las etiquetas de apertura y cierre y el contenido del elemento.

Las etiquetas básicas de *HTML* son la siguientes:

- *<html>* ..: comienzo del documento HTML.
- *<head>* ..: cabecera de la página.
- *<body>* ..: cuerpo de la página.
- *<h1>, <h2>, etc.*: son los títulos o encabezados.
- *<a>* ..: define los enlaces.
- *<table>* ..: es una tabla.
- *<p>* ..: define un párrafo.
- ** ..: imágenes.
- ** ..: define una lista.

Además, cada elemento puede incluir uno o más atributos que permiten añadir mas información acerca de ese elemento. Por ejemplo, añadir información sobre el estilo del elemento. A continuación, se incluye un fragmento de código *HTML* a modo de ejemplo.

```
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset="utf-8">
  <title>Mi código de prueba</title>
</head>
<body>
  <p class="editor-note">Esto es<strong>una prueba</strong></p>
</body>
</html>
```

Kibotics emplea *HTML* para crear las plantillas de las diferentes páginas que sirve la aplicación web.

3.3. Lenguaje *JSON*

El acrónimo *JSON* significa JavaScript Object Notation (Notación de Objetos de JavaScript). Se trata de un formato para el intercambio de datos. Es un lenguaje sencillo para la escritura y lectura humana y, al mismo tiempo, resulta fácil para las máquinas interpretarlo y procesarlo. *JSON* está constituido por dos estructuras: una colección de pares nombre - valor y una lista ordenada de valores. Dado que estas convenciones son conocidas por otros lenguajes como *C*, *C++*, *C*, *Java*, *JavaScript*, *Perl*, *Python*, se trata de un lenguaje ideal para el intercambio de datos⁵.

⁵<https://www.json.org/json-es.html>

En *JSON*, estas estructuras se presentan de la siguiente forma:

- **Objeto:** conjunto desordenado de pares nombre - valor. Un objeto va encerrado entre llaves (). La sintaxis es la siguiente:

```
objeto {  
    nombre1: valor1,  
    nombre2: valor2,  
    ...  
}
```

- **Array:** colección de valores. Van encerrados entre corchetes []. Un valor puede ser una cadena de caracteres con comillas dobles, un número, true, false o null, un objeto o un array. Estas estructuras pueden anidarse.

La plataforma *Kibotics* emplea *JSON* para los ficheros de configuración de los diferentes escenarios utilizados en los ejercicios que ofrece la aplicación. Mediante un *parser* se recopila la información necesaria del fichero de configuración *JSON* para construir el mundo en *A-Frame*.

3.4. *Blender*

*Blender*⁶ es un programa informático multi plataforma, es decir, compatible para distintos sistemas operativos como Windows o Linux. Las funciones principales que se pueden realizar con *Blender* son el modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. También se pueden realizar actividades relacionadas con la composición de vídeo.

Durante el presente trabajo se ha utilizado esta herramienta para modificar la rotación y apariencia de los robots de los que dispone la plataforma *Kibotics*. Además, *Blender* permite exportar los modelos en formato glTF (GL Transmission Format). glTF es un formato de archivo basado en el estandar *JSON*. Permite la compresión de escenas y modelos 3D para minimizar el tiempo de ejecución de los programas en los que posteriormente se utilicen.

⁶<https://www.blender.org/>

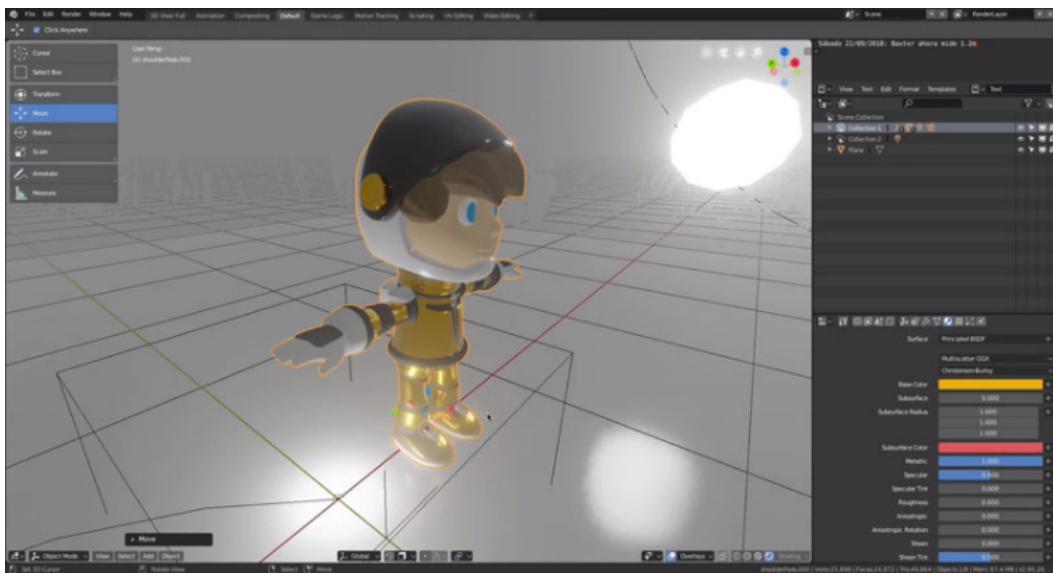


Figura 3.1: Interfaz de trabajo con *Blender*

3.5. Entorno A-Frame

A-Frame es un marco web para crear escenas de realidad virtual en el navegador. Ha sido utilizado por empresas como *Google*, *Disney*, *Samsung*, *Toyota* o *CERN*, entre otras. Además, algunas de ellas, como *Google*, *Microsoft*, *Oculus* y *Samsung* han llegado a realizar contribuciones. Sus principales características son las siguientes⁷:

- **Permite un uso sencillo de la realidad virtual:** para usar *A-Frame* basta con colocar las etiquetas `<script>` y `<a-scene>`. *A-Frame* se encarga del modelado 3D y la realidad virtual, no es necesaria la instalación de ningún paquete externo.
- **HTML declarativo:** *A-Frame* está basado en *HTML*, por ello es fácil y accesible para cualquier programador, puesto que *HTML* es un lenguaje ampliamente conocido.
- **Arquitectura de componente de entidad:** *A-Frame* sigue el patrón *ECS* (entidad-componente-sistema). Se trata de un patrón de desarrollo de juegos basado en el principio de composición sobre herencia. De esta manera, se otorga una mayor flexibilidad en la definición de entidades ya que cada objeto de la escena se corresponde con una entidad y cada entidad, a su vez, está compuesta por uno o más componentes que contienen datos y estado de la

⁷<https://aframe.io/docs/1.0.0/introduction/features>

entidad. Por tanto, una entidad puede verse modificada en tiempo de ejecución si alguno de los componentes que agrega modifica sus datos.

- **Multiplataforma:** *A-Frame* es compatible con plataformas tan variadas como *Vive*, *Rift*, *Windows Mixed Reality*, *Daydream*, *GearVR* y *Cardboard*.
- **Rendimiento:** las actualizaciones de *A-Frame* se realizan en la memoria y con poco gasto energético. *A-Frame* se encuentra optimizado para *WebVR*.
- **Inspector visual:** *A-Frame* cuenta con un inspector visual integrado. Este se despliega al presionar la combinación de teclas < *ctrl* > + < *alt* > + *i*. El inspector permite detectar el origen de problemas o desarrollar una mejor distribución de la escena con menos esfuerzo.
- **Componentes:** *A-Frame* cuenta con una gran cantidad de componentes con los que trabajar. Esta amplia variedad va desde componentes geométricos básicos o materiales hasta componentes como la teletransportación, la realidad aumentada o componentes personalizados por el usuario.

3.5.1. Sistema de físicas de *A-Frame*

Las físicas de *A-Frame* soportan dos motores de físicas: *Ammo Driver* y *CANNON*. Actualmente, el motor que está en uso por defecto es el de *CANNON*⁹. No obstante, ya ha sido añadido el soporte de *Ammo.js* al sistema de físicas, y se preveé que *CANNON* acabe quedando obsoleto con el paso del tiempo.

Para la instalación del sistema de físicas de *A-Frame* basta con incluir el siguiente script en el código *HTML* de la aplicación:

```
<script src="//cdn.rawgit.com/donmccurdy/aframe-physics-system/v4.0.1/dist/aframe-physics-system.min.js"></script>
```

El sistema de físicas de *A-Frame* cuenta con dos tipos de cuerpos: dinámicos y estáticos.

⁸<https://aframe.io/>

⁹<https://github.com/donmccurdy/aframe-physics-system>



Figura 3.2: Ejemplo de construcción 3D con *A-Frame*⁸

- **Cuerpo dinámico:** aquellos objetos de la escena que presentan libertad de movimiento. Estos objetos se ven afectados por la gravedad, la fricción y las colisiones.
- **Cuerpo estático:** aquellos objetos de la escena que no necesitan modificar su posición en la misma. Son cuerpos fijos y sin animaciones. Otros cuerpos dinámicos podrán colisionar con un cuerpo estático, pero el cuerpo estático no verá modificada su posición tras la colisión.

El sistema de físicas ofrece la posibilidad de añadir una malla de colisión a un objeto de la escena. Existen mallas de colisión de diferentes formas, por lo que se debe seleccionar aquella que se ajuste mejor al objeto en cuestión. Se puede elegir entre el ajuste automático, una caja, un cilindro, una esfera, un cuerpo convexo o una primitiva (plano, cilindro o esfera seleccionadas automáticamente en función de la primitiva *A-Frame* correspondiente).

Cada escena de *A-Frame* admite una serie de parámetros a los que se les puede modificar el valor para ajustar el mundo a las características deseadas. Si no se especifica el valor que se desea de un parámetro, este tomará el valor por defecto. Algunos de los parámetros que admite

una escena son *debug*, que cuando está a *true* muestra las mallas de colisión de los objetos de la escena o *gravity*, *friction* y *restitution*, que se corresponden con la gravedad, fricción y coeficiente de restitución, respectivamente, del mundo simulado.

Atributo	Valor por defecto
debug	true
gravity	-9.8
friction	0.01
restitution	0.3

Cuadro 3.1: Parámetros configurables del sistema de físicas de *A-Frame*

3.6. Plataforma *Kibotics*

La batería de ejercicios que incluye el entorno *Kibotics* se ejecuta usando el simulador robótico *Websim*. Se trata de un simulador diseñado para el aprendizaje de conceptos básicos de programación de robots especialmente para niños. El simulador permite que los usuarios puedan programar fácilmente los movimientos de los robots, ya que simplemente tienen que acceder a la información que recogen sus sensores y enviar las órdenes precisas a los actuadores del robot. Estas órdenes se deberán programar, en *Python* o *Scratch*, dentro del editor que incorpora la interfaz de *Websim*.

El simulador está diseñado basándose en el uso del entorno *A-Frame*. A su vez, *A-Frame* se sirve del motor de físicas de *CANNON* para materializar los movimientos de los cuerpos dinámicos en la escena.

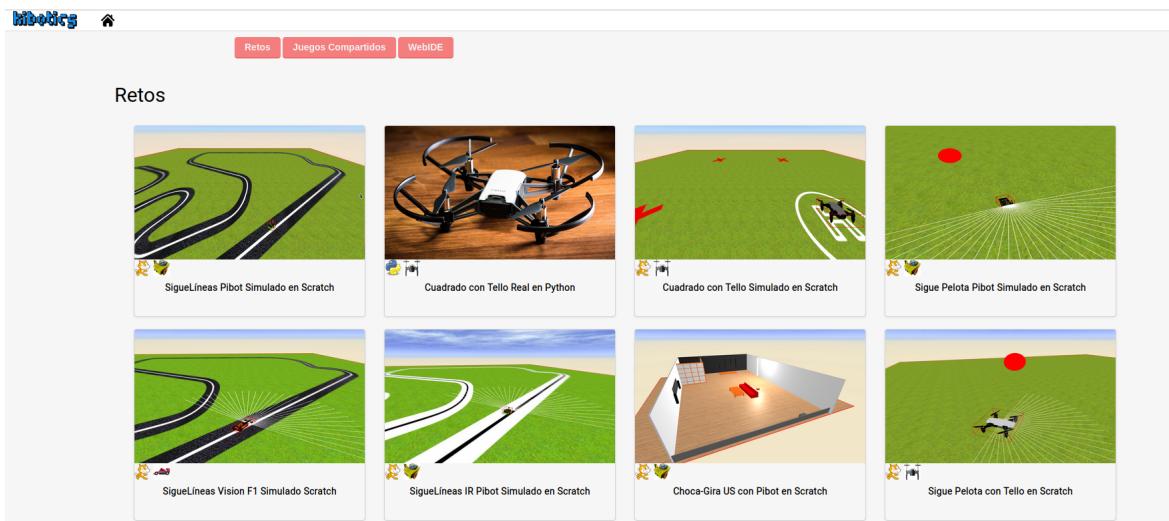


Figura 3.3: Menú de ejercicios de *Kibotics*

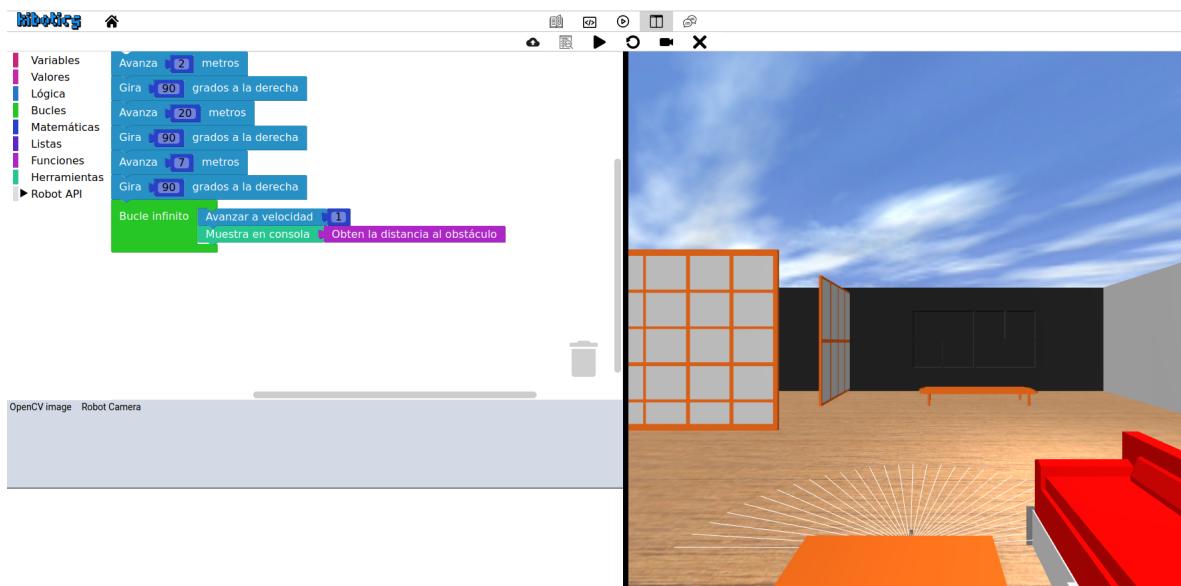


Figura 3.4: Editor *Scratch* en *WebSim*

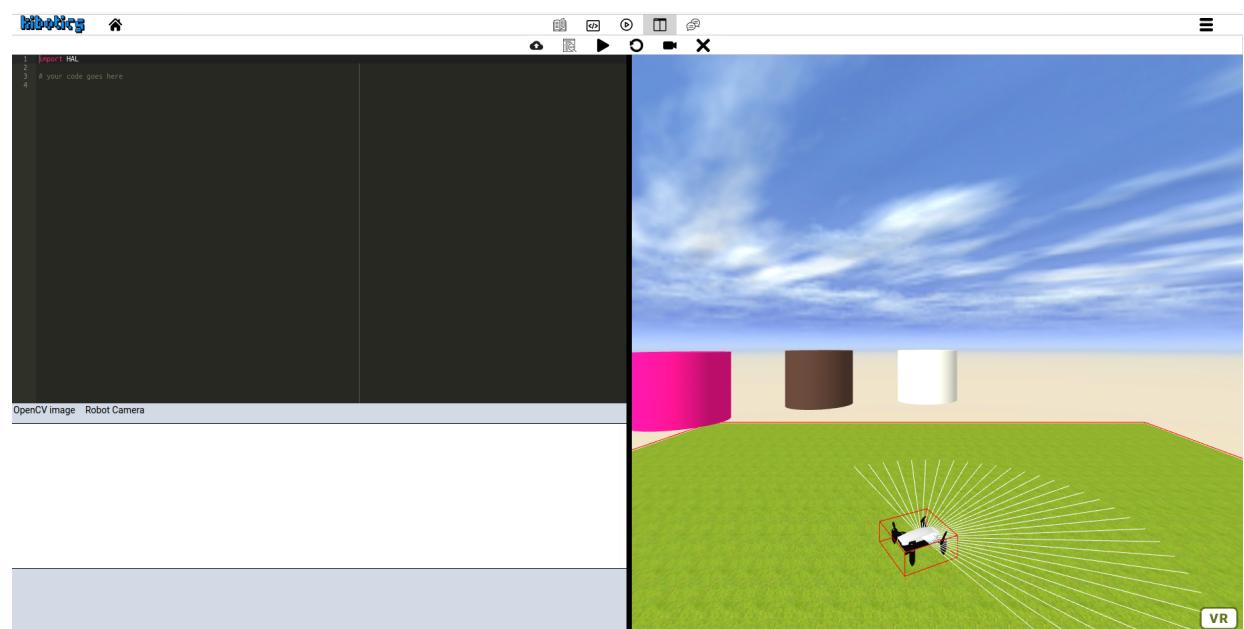


Figura 3.5: Editor *Python* en *WebSim*

Capítulo 4

Mejora de las físicas en *WebSim*

En este capítulo se va a explicar el diseño, implementación y funcionamiento del nuevo motor de físicas complementario que se ha creado y que actualmente está en producción en el entorno *Kibotics*. Como se detallará más adelante, el nuevo motor permite disponer de unas físicas más realistas en los ejercicios y simular nuevos escenarios, como rampas o pistas de hielo, ya que la gravedad y la fricción van a estar contempladas en la simulación. Además, se ganará utilidad a la hora de probar las soluciones de los ejercicios simulados en robots reales, puesto que los factores que presenta un robot físico (por ejemplo: masa, momento de incercia o velocidad máxima) o una escena real (por ejemplo: rozamiento estático, rozamiento dinámico, restitución o gravedad) van a ser parámetros que se puedan materializar en el simulador gracias a las novedades introducidas.

4.1. Estudios previos: motor de físicas por defecto en *A-Frame*

A-Frame utiliza por defecto el motor de *CANNON*¹ para materializar las físicas. *CANNON* surgió como consecuencia de la necesidad de disponer de un motor de físicas en la web y presenta importantes semejanzas respecto a otros conocidos motores: *three.js* o *ammo.js*. Su ventaja es que su código se encuentra disponible enteramente en red², escrito en lenguaje *JavaScript* y que su tamaño de archivo es considerablemente más pequeño que el de otros motores de físicas³.

¹<https://github.com/n5ro/aframe-physics-system>

²<http://schtepppe.github.io/cannon.js/docs/>

³<http://schtepppe.github.io/cannon.js/>

A continuación se detallan las posibilidades que ofrece *A-Frame* para controlar la gravedad, la restitución y la fricción de una escena simulada.

4.1.1. Gravedad

Dado que *WebSim* se basa en la tecnología de *A-Frame*, la gravedad es un parámetro configurable dentro de una escena. Los ficheros de configuración de los escenarios de los ejercicios en formato *JSON* incluyen al principio del código las siguientes líneas que permiten seleccionar el valor deseado para la gravedad.

```
"scene": {
    "gravity": -9.8
}
```

Como se ha mencionado anteriormente, previamente al cambio introducido en las físicas, los ficheros de configuración de los ejercicios soportados en la plataforma tenían definida una gravedad con valor 0. Esta configuración era necesaria para conseguir hacer volar a los drones, puesto que con una gravedad de -9.8 cualquier cuerpo sólido de la escena caía hacia abajo como consecuencia de la atracción de la gravedad, por lo que no era posible hacer volar a los robots. Con el nuevo motor de físicas, todos los ejercicios están simulados con un valor de gravedad de -9.8.

4.1.2. Colisiones

Cualquier cuerpo sólido puede colisionar con otro cuerpo incluído en la escena simulada. Las colisiones pueden tener naturaleza elástica o inelástica dependiendo del valor del coeficiente de restitución de los objetos. El coeficiente de restitución es la media de la conservación de la energía cinética cuando se produce un choque entre partículas. Cuando su valor es 1 el choque es perfectamente elástico y cuando es 0, es perfectamente inelástico.

$$\text{Coeficiente de restitución} = \frac{\text{Velocidad relativa tras la colisión}}{\text{Velocidad relativa antes de la colisión}} \quad (4.1)$$

A-Frame también ofrece la posibilidad de parametrizar el coeficiente de restitución. Este parámetro se puede configurar, al igual que la gravedad, al principio del fichero de configuración de los ejercicios incluyendo el siguiente código.

```
"scene": {  
    "physics": "restitution: 0.5"  
}
```

Colisiones elásticas

Se dice que una colisión es elástica cuando, tras el choque, se conserva toda la energía cinética. Esta se transfiere por completo desde el objeto que colisiona hasta el objeto que ha sido colisionado. En la realidad, en todo choque parte de la energía se disipa en calor, por lo que este tipo de colisiones es considerada ideal. Visualmente, el efecto que tiene es que el objeto que colisiona se queda parado y el objeto colisionado comienza a moverse a la velocidad que se movía el objeto que colisionó con él (si son de la misma masa). La Figura 4.1 muestra un ejemplo de colisión elástica.

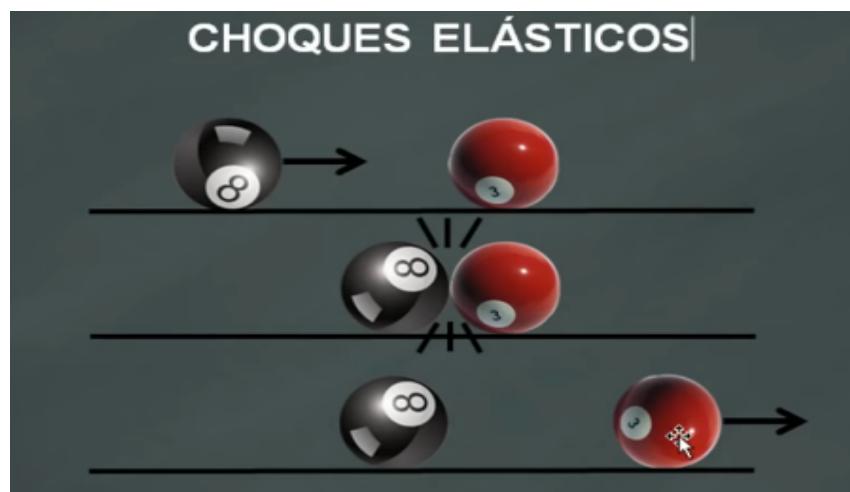


Figura 4.1: Colisión elástica⁴

⁴<https://www.youtube.com/watch?v=b9iOIr5DYj8>

Colisiones inelásticas

Por otro lado, cuando se produce una colisión inelástica el objeto que colisiona continúa teniendo parte de la energía cinética, otra parte se transfiere al objeto que ha sido colisionado y la parte restante se disipa en forma de calor. En este tipo de choques el efecto visual es que tanto el objeto que colisiona como el objeto colisionado avanzan a cierta velocidad tras el choque. La Figura 4.2 muestra un ejemplo de colisión inelástica.

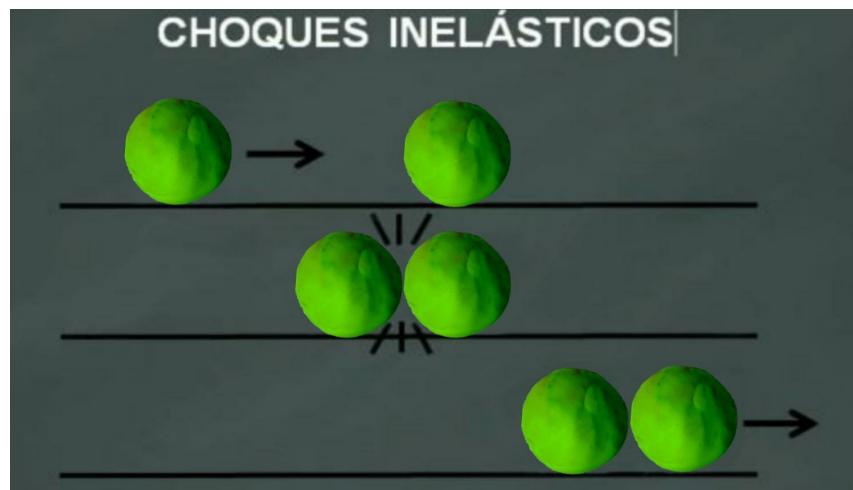


Figura 4.2: Colisión inelástica⁵

Pruebas de colisiones

Se han realizado pruebas de colisiones de *A-Frame* en tres escenarios diferentes variando el valor del coeficiente de restitución y la masa de los objetos. Los escenarios utilizados han sido los siguientes:

- **Escenario 1:** dos pelotas cayendo por sendas rampas.
- **Escenario 2:** una pelota fija en el suelo y otra cayendo por una rampa.
- **Escenario 3:** una pelota cae por una rampa y colisiona con una pared.

Los resultados obtenidos durante las pruebas se detallan en los Cuadros 4.1, 4.2 y 4.3. En este link⁶ se ofrece un vídeo con los resultados de las pruebas realizadas en el tercer escenario.

⁵Elaboración propia.

⁶<https://youtu.be/T214FNFxehs>

CAPÍTULO 4. MEJORA DE LAS FÍSICAS EN WEBSIM

	Misma masa
Restitution = 0	No hay rebote
Restitution = 0.4	Sí hay rebote
Restitution = 1	La pelota rebota al entrar en contacto con cualquier superficie de la escena

	Diferente masa
Restitution = 0	Las dos pelotas avanzan pegadas en la dirección de la de mayor masa
Restitution = 0.4	La pelota más pesada hace cambiar la dirección del movimiento de la más ligera, que se desplaza a mayor velocidad
Restitution = 1	Las dos pelotas avanzan separadas en la dirección de la de mayor masa. La pelota de menor masa coge mayor velocidad

Cuadro 4.1: Resultados de las colisiones obtenidos con el escenario 1

	Misma masa
Restitution = 0	Ambas pelotas avanzan pegadas a la misma velocidad
Restitution = 0.4	La pelota que permanecía en el suelo se mueve más rápido que la que cayó por la rampa. No avanzan pegadas
Restitution = 1	La pelota rebota al entrar en contacto con cualquier superficie de la escena

	Diferente masa
Restitution = 0	<ul style="list-style-type: none"> - Masa pelota rampa < Masa pelota suelo: ambas pelotas se quedan juntas y paradas - Masa pelota rampa > Masa pelota suelo: ambas pelotas avanzan hacia adelante juntas y a la misma velocidad
Restitution = 0.4	<ul style="list-style-type: none"> - Masa pelota rampa < Masa pelota suelo: la pelota que ha caído por la rampa rebota hacia arriba - Masa pelota rampa > Masa pelota suelo: la pelota que estaba en reposo avanza con más velocidad que la que cayó por la rampa
Restitution = 1	La pelota rebota al entrar en contacto con cualquier superficie de la escena

Cuadro 4.2: Resultados de las colisiones obtenidos con el escenario 2

	Pelota con masa grande
Restitution = 0	Permanece junto a la pared, sin rebote
Restitution = 0.4	Sí hay rebote
Restitution = 1	El rebote es muy elevado y vuelve a subir la rampa prácticamente a la misma velocidad que la bajó

	Pelota con masa pequeña
Restitution = 0	Permanece junto a la pared, sin rebote
Restitution = 0.4	Cuanto más pequeña es la masa, más grande es el rebote
Restitution = 1	Cuanto más pequeña es la masa, más grande es el rebote

Cuadro 4.3: Resultados de las colisiones obtenidos con el escenario 3

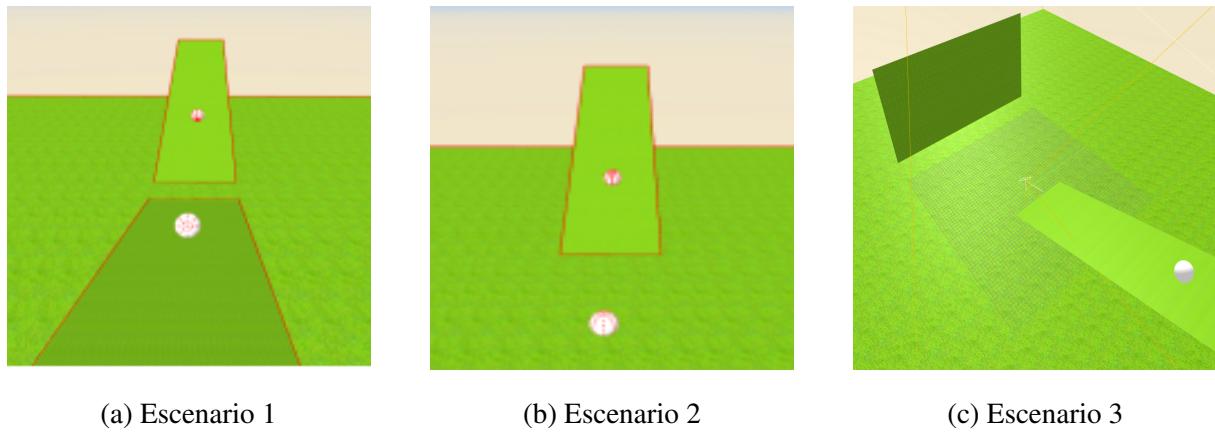


Figura 4.3: Escenarios de prueba de las colisiones de *A-Frame*

4.1.3. Fricción

Cualquier cuerpo dinámico de una escena de *A-Frame* presenta una fuerza de rozamiento que se opone al movimiento. Existen dos tipos de rozamiento: el rozamiento estático y el rozamiento dinámico. *A-Frame* incluye tres parámetros diferentes que permiten variar el rozamiento de una superficie. No obstante, no se puede seleccionar un valor concreto para la fuerza de rozamiento estática y dinámica, ya que el atributo *friction* hace variar los dos tipos de rozamiento en conjunto. Por tanto, se puede incrementar o disminuir tanto el rozamiento estático como el dinámico, pero no se pueden parametrizar con un valor concreto.

Los ficheros de configuración de los escenarios de los ejercicios en formato *JSON* incluyen al principio del código las siguientes líneas que permiten seleccionar el valor deseado para los atributos *friction* (rozamiento estático y dinámico) y *linear-damping* y *angular-damping* (rozamiento dinámico).

```
"scene": {
    "physics": "friction: 0.5"
}
attr": {
    "static-body": {
        "linearDamping":1.2,
        "angularDamping":1.2
    }
}
```

Rozamiento estático

Dos superficies rígidas en reposo no se desplazan una respecto a la otra y cuando la fuerza paralela al plano tangente es suficientemente pequeña. Cuando el coeficiente de rozamiento estático de una superficie es excesivamente pequeño, los objetos que se encuentran sobre esa superficie sí se mueven con más facilidad. Una forma de calcular el coeficiente de rozamiento estático es hacer variar la inclinación de una rampa. Cuando se alcanza un ángulo de inclinación con el cual el cuerpo comienza a descender, se dice que se ha llegado al ángulo crítico. A partir del ángulo crítico se puede obtener el coeficiente de rozamiento estático gracias a la siguiente igualdad:

$$\tan(\text{angulo crítico}) = \text{coeficiente de rozamiento estático} \quad (4.2)$$

Rozamiento dinámico

Cuando dos superficies están en contacto, el movimiento de una respecto a la otra genera fuerzas tangenciales llamadas fuerzas de fricción o rozamiento, las cuales tienen sentido opuesto al movimiento. La magnitud de esta fuerza depende del coeficiente de rozamiento dinámico. En *A-Frame*, el rozamiento dinámico se puede configurar utilizando los atributos *linear-damping* y *angular-damping* a nivel de objeto, además del atributo *friction* a nivel de escena, como se ha mencionado anteriormente.

Pruebas de la fricción

Se han realizado diferentes pruebas de fricción colocando un objeto en una rampa y variando la inclinación de la misma. Los resultados obtenidos han sido los siguientes:

- **Prueba del rozamiento estático:** se coloca un objeto sobre una rampa y se procede a la variación de la inclinación de la misma. El objeto permanece en la misma posición hasta que se alcanza un ángulo de inclinación tan elevado que el cuerpo comienza a descender (ángulo crítico). Si se configura el atributo *friction* con un valor más elevado, el ángulo crítico se alcanza más tarde, es decir, es mayor.
- **Prueba del rozamiento dinámico:** se mantiene fija la inclinación de la rampa y se varía el valor de los atributos *friction*, *linear-damping* y *angular-damping*. Con unos valores

elevados de esos tres parámetros, los robots no son capaces de subir la rampa. Sin embargo, a medida que se va reduciendo el valor de los atributos los robots comienzan a poder subir la rampa y cada vez lo hacen con mayor facilidad.

4.2. Motor de físicas actual para robots en *WebSim*

Hasta el momento, *WebSim* contaba con un motor de físicas completo. El movimiento de los robots se recreaba mediante la actualización de la posición cada 50 ms en función de la velocidad y la rotación consignada por el usuario. No entraban en juego aceleraciones ni fuerzas. Además, el hecho de actualizar la posición y la velocidad del robot constantemente (la velocidad se tomaba de manera instantánea de la última enviada por el cerebro software del robot. No existía diferencia entre la velocidad real y la velocidad deseada. Se asumía aceleración infinita del robot) sobreescrivía encima de los cambios que introducía *CANNON*, por lo que las físicas no incluían el efecto de la gravedad, la fricción ni el coeficiente de restitución.

La actualización de la posición de los robots en función de la velocidad y rotación comandadas se realizaba mediante la función *updatePosition*.

```
updatePosition(rotation, velocity, robotPos) {  
    if(simEnabled) {  
        let x = velocity.x / 10 * Math.cos(rotation.y * Math.PI / 180);  
        let z = velocity.x / 10 * Math.sin(-rotation.y * Math.PI / 180);  
        let y = (velocity.y / 10);  
        robotPos.x += x;  
        robotPos.z += z;  
        robotPos.y += y;  
    }  
    return robotPos;  
}
```

La motivación de crear un motor de físicas complementario es que las físicas implementadas no recreaban un movimiento realista y no eran suficientes para lo que se tiene disponible en un

robot real. Por ejemplo, los robots reales terrestres tienen que superar la fuerza de rozamiento para poder subir una rampa o los drones deben ser capaces de ejercer la fuerza necesaria para superar la gravedad y emprender el vuelo. Con un motor de físicas realistas, las soluciones implementadas en el simulador serán igualmente válidas para un robot físico y, además, permite la creación de ejercicios mucho más diversos, como una pista de hielo (en los que la fricción es extremadamente baja), arena (en los que la fricción es mucho más elevada) o escenarios multi-nivel con rampas (en los que los robots deberán ejercer fuerzas más grandes que en la superficie plana para poder ascender por la rampa).

Las implicaciones que tiene la materialización de la gravedad y de la fricción son las siguientes:

- **Materrialización de la gravedad:** los drones son capaces de volar en un mundo que materialice una gravedad de -9.8. Hasta el momento, los ejercicios de drones se configuraban con gravedad 0 para permitir volar al cuerpo.
- **Materialización de la fricción:** el desplazamiento de los robots no se realiza por imposición de una posición concreta, sino por la aplicación de la fuerza necesaria para alcanzar la velocidad objetivo. Al tener en cuenta la fricción, un robot deberá ejercer más fuerza sobre superficies con altas fricciones y menos fuerza en superficies con fricciones más pequeñas.

4.3. Nuevo motor de físicas complementario

4.3.1. Diseño

Este motor de físicas se utiliza para controlar el movimiento de los cuerpos dinámicos en la escena. Se le denomina complementario puesto que sólo se encarga de aplicar la fuerza autónoma al robot, dejando como tarea de *CANNON* materializar las fuerzas de la fricción y la gravedad.

$$\text{Fuerza robot} = \text{Fuerza autónoma} + \text{Fuerza gravedad} + \text{Fuerza fricción}$$

- Fuerza gravedad: materializada por *CANNON* a un ritmo marcado por el propio motor *CANNON*.
- Fuerza fricción: materializada por *CANNON* a un ritmo marcado por el propio motor *CANNON*.
- Fuerza autónoma: lo materializa nuestro motor complementario a su propio ritmo (distinto del de Cannon) y teniendo en cuenta las velocidades deseadas que marca en cada instante el código fuente del cerebro programado. El ritmo que se ha fijado ha sido de 20 ms.

Esta idea de motor complementario es radicalmente distinta de la implementación que existía hasta el momento. La función *updatePosition* actúaba como un motor completo que sobreescribía a *CANNON*. No existía combinación alguna entre ambos motores.

La arquitectura del motor de físicas complementario presenta dos niveles. El nivel superior conecta directamente con el cerebro software del robot, el cual es capaz de dar continuamente instrucciones en posición y en velocidad. No obstante, ambos tipos de instrucciones se basan en comandar una velocidad objetivo. En el caso de las instrucciones en posición el robot se moverá a la velocidad objetivo hasta que se alcance la posición consignada en la instrucción y en el caso de las instrucciones en velocidad, el movimiento no termina. Este nivel más alto se basa en un controlador PD que traduce las consignas de velocidad que le llegan del cerebro, en la fuerza necesaria a aplicar al robot para alcanzar dichas velocidades consignadas. Estas fuerzas son las

que entiende el núcleo o nivel más bajo del motor de físicas. Conociendo la masa y el momento de inercia, las fuerzas obtenidas se podrán traducir en aceleraciones. La Figura 4.4 muestra un esquema del diseño del motor.

Por otro lado, se incluye la tabla 4.4 que introduce a modo de resumen todos los parámetros que caracterizan el movimiento de los robots autónomos con el nuevo motor de físicas complementario. Tanto parámetros del modelo de fuerzas (masa, momento de inercia, fuerza máxima, torque máximo, maxima velocidad lineal, maxima velocidad angular, máxima aceleración lineal y máxima aceleración angular) como parámetros de A-Frame (coeficiente de restitución, gravedad, fricción, amortiguación lineal y amortiguación angular). Los últimos ya se han explicado y los primeros se describen a continuación.

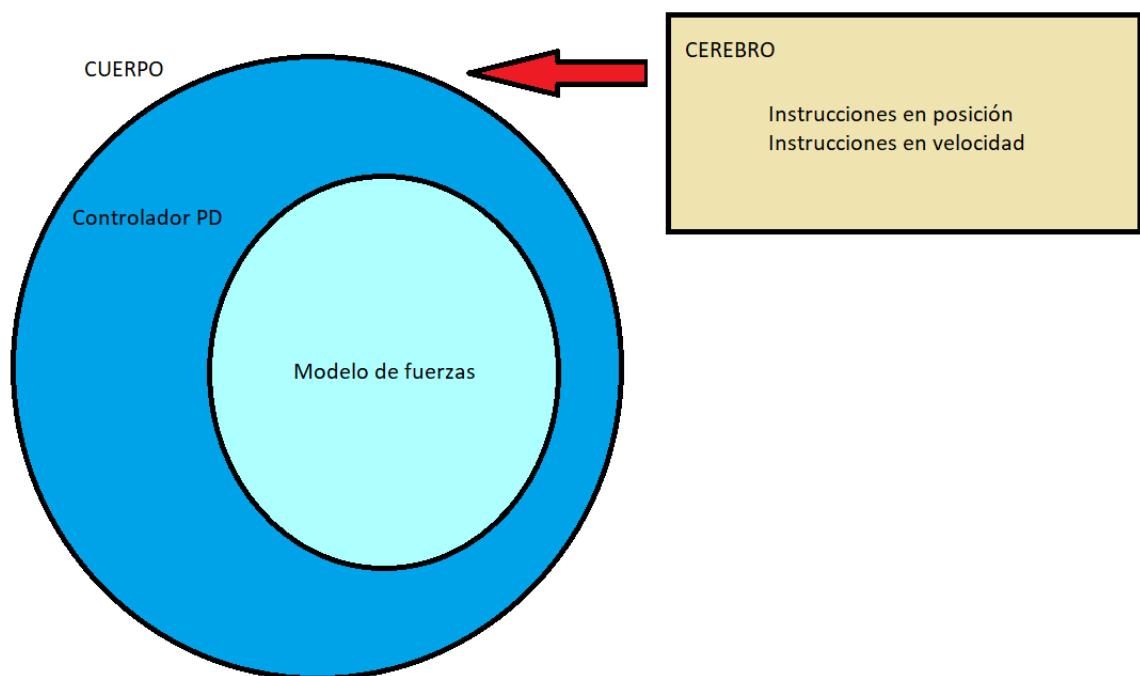


Figura 4.4: Diseño del motor de físicas complementario⁷

⁷Elaboración propia.

Parámetros del modelo de fuerzas	
mass	Masa del robot
inertia	Momento de inercia del robot
Fmax	Fuerza máxima aplicable
Tmax	Torque máximo aplicable
accelerationMax	Aceleración lineal máxima
angularAccelerationMax	Aceleración angular máxima
linealSpeedMax	Velocidad lineal máxima que puede alcanzar el robot
angularSpeedMax	Velocidad angular máxima que puede alcanzar el robot
Parámetros de A-Frame	
restitution	Conservación de la energía cinética en un choque entre partículas
gravity	Gravedad
friction	Fricción (rozamiento estático y dinámico)
linearDamping	Amortiguación lineal (rozamiento dinámico en el movimiento lineal)
angularDamping	Amortiguación angular (rozamiento dinámico en el movimiento angular)

Cuadro 4.4: Parámetros que caracterizan el movimiento de un robot autónomo

4.3.2. Modelo de fuerzas

El núcleo del nuevo motor de físicas es un modelo de fuerzas en el que, a partir de la definición de la masa y el momento de inercia del robot, se calcula la aceleración o torque a aplicar. De esta manera, un robot muy pesado deberá ejercer una fuerza mayor que un robot más ligero para alcanzar una misma velocidad.

Para que el modelo funcione es necesaria la definición de los siguientes parámetros:

- **Fuerza máxima:** es la fuerza autónoma máxima que puede aplicar un robot. Concede realismo al modelo, puesto que termina con la asunción de aceleración infinita.
- **Torque máximo:** es el momento autónomo de fuerza máximo que puede aplicar un robot, es decir, es la fuerza máxima de giro aplicable. Concede realismo al modelo, puesto que termina con la asunción de aceleración infinita.
- **Velocidad lineal máxima:** es la velocidad lineal máxima que puede alcanzar un robot.
- **Velocidad angular máxima:** es la velocidad angular máxima que puede alcanzar un robot.
- **Masa:** es necesaria para obtener el valor de la aceleración lineal.
- **Momento de inercia:** es la medida de la inercia rotacional cuando un cuerpo gira. Es el equivalente a la masa en un movimiento angular. Es necesario para obtener el valor de la aceleración angular.

```
/* Masa e inercia*/
const mass = this.robot.body.mass;
const inertia = this.robot.body.inertia.x;

/* Fuerza y torque máximos */
const fMax = 100;
const tMax = 100;
```

```
/* Aceleración lineal y angular máxima */
const accelerationMax = fMax / mass;
const angularAccelerationMax = tMax / inertia;

/* Aceleración lineal y angular máxima */
const linearSpeedMax = 10;
const angularSpeedMax= 5;
```

4.3.3. Controlador PD

El controlador PD se encarga de la traducción de las velocidades deseadas que le llegan al motor complementario en cada momento del cerebro a la fuerza autónoma a aplicar al robot.

El controlador PD es una variante del controlador PID (controlador proporcional, integral y derivativo) que no incluye la componente integral. Se trata de un controlador por realimentación que calcula la desviación o error entre una medida y el valor que se desea obtener. Cada componente tiene una utilidad diferente y depende de distintos factores:

- **Componente proporcional:** depende del error actual y su función es minimizar el error del sistema.
- **Componente derivativa:** depende de los errores pasados y permite estabilizar el sistema reduciendo la oscilación del valor de salida.
- **Componente integral:** es una predicción de los errores futuros y se utiliza cuando el componente derivativo no consigue reducir el error del sistema. Su uso es complejo ya que puede producir la desestabilización del sistema.

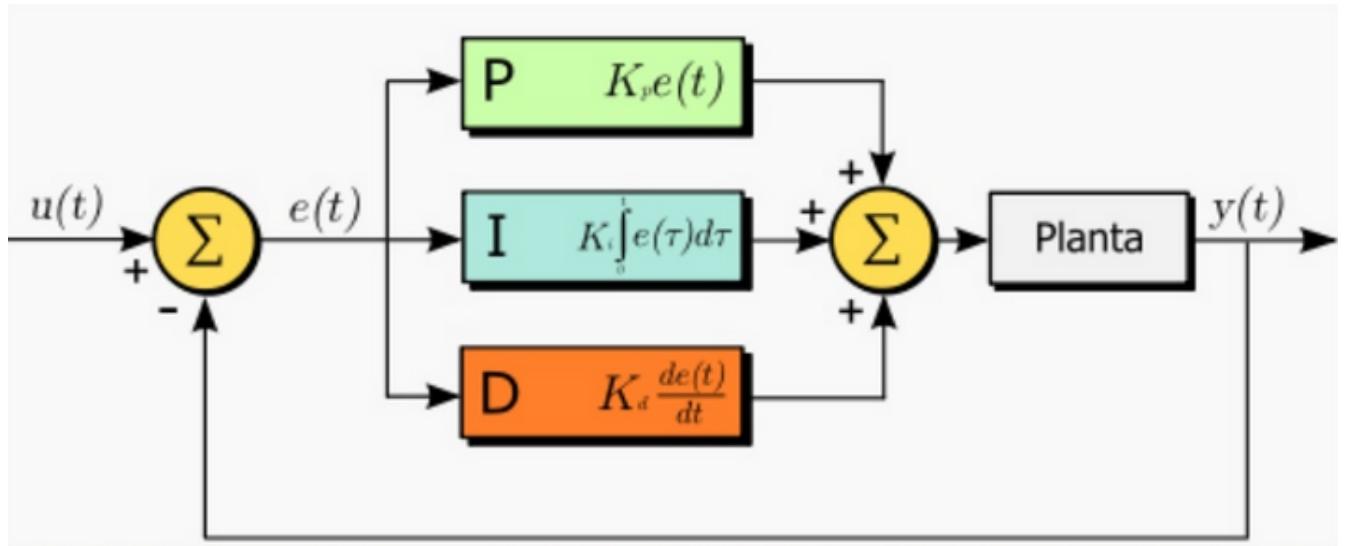


Figura 4.5: Controlador PID⁸

Mediante un sencillo algoritmo basado en la suma de estos tres componentes, el controlador es capaz de ajustar su salida a un valor de referencia. Además, el modelo incluye tres constantes que se emplean para ponderar los componentes anteriores. En este caso sólo se van a tener en cuenta los componentes proporcionales y derivativos puesto que se van a implementar controladores PD porque el error del sistema no es excesivamente elevado.

$$c(t) = K_p \cdot e(t) + K_i \cdot \int e(t) dt + K_d \cdot \frac{\partial e(t)}{\partial t}$$

Esta capa del motor incluye cuatro controladores PD diferentes que se ejecutan dependiendo del tipo de robot que esté realizando el movimiento (robot terrestre o drone) y del tipo de movimiento que se efectúe (avance lineal, giro, vuelo o suspensión en el aire). A continuación, se detallan los cuatro tipos de controladores incluidos.

- **Controlador PD en velocidad del plano horizontal:** además de la velocidad horizontal deseada, coge como entrada la velocidad resultante del plano horizontal en ese instante y genera como salida la fuerza que debe ejercer el robot para alcanzar la velocidad objetivo.

⁸<https://es.slideshare.net/quasar.0360.7912/sintonizacion-de-controladores-pid>

⁹Elaboración propia.

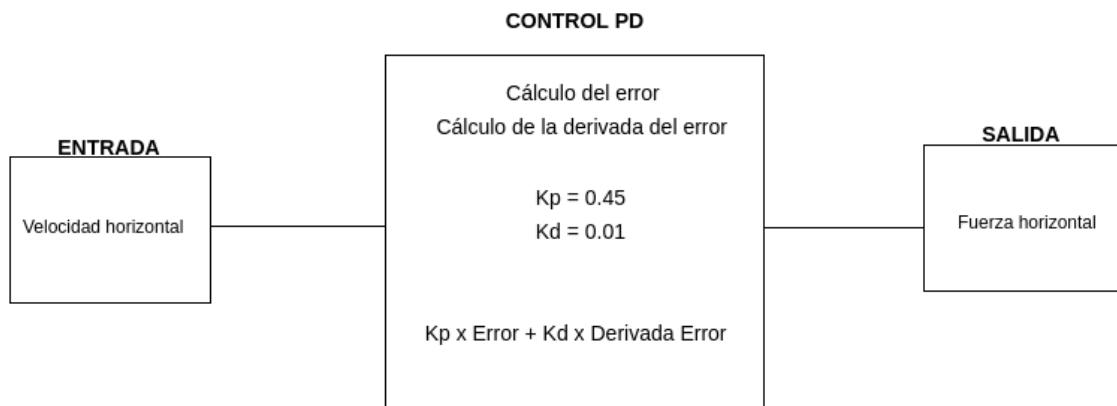


Figura 4.6: Diseño controlador PD en velocidad del plano horizontal⁹

- **Controlador PD en velocidad del eje vertical:** además de la velocidad vertical deseada, coge como entrada la componente vertical de la velocidad en ese instante y genera como salida la fuerza que debe ejercer el robot para alcanzar la velocidad objetivo.

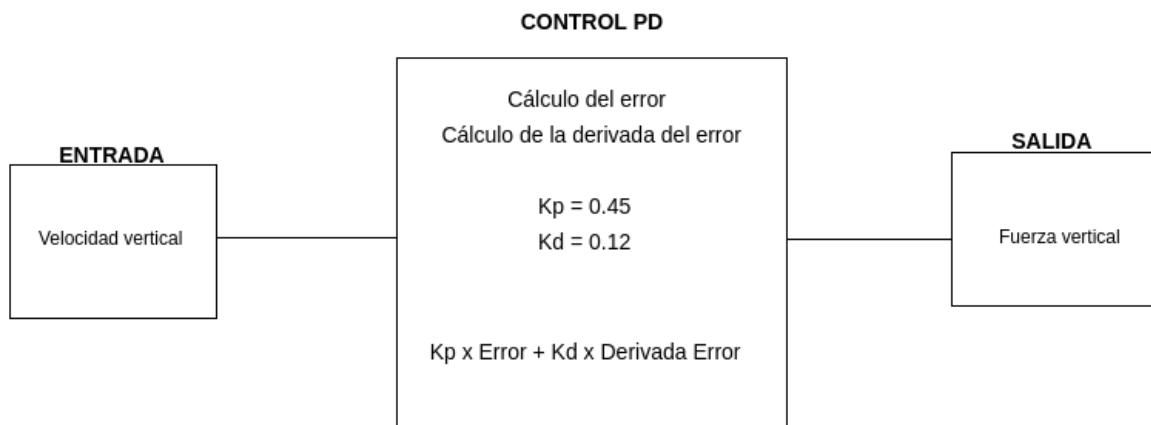


Figura 4.7: Diseño controlador PD en velocidad del eje vertical¹⁰

- **Controlador PD en velocidad angular horizontal (yaw):** además de la velocidad deseada de guiñada, coge como entrada la velocidad angular en ese instante y genera como salida el torque que debe ejercer el robot para alcanzar la velocidad de giro objetivo.

¹⁰Elaboración propia.

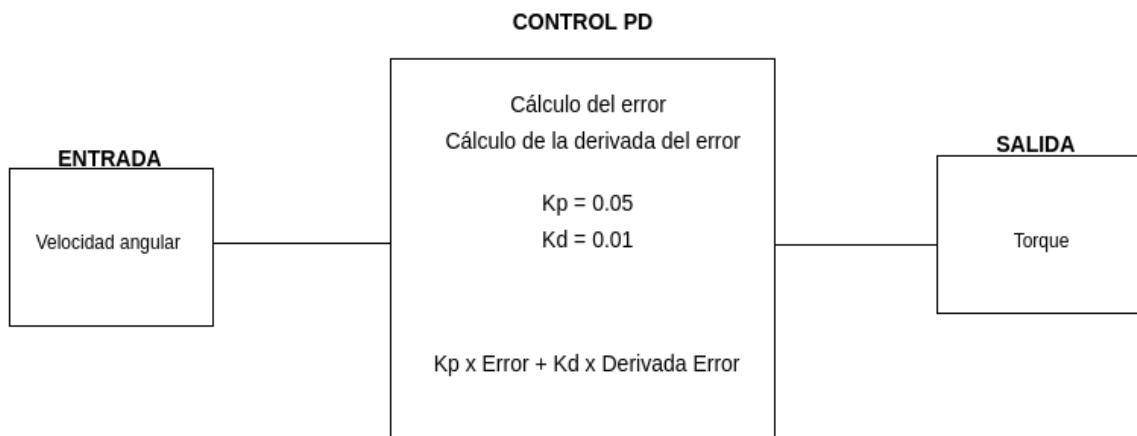


Figura 4.8: Diseño controlador PD en velocidad angular horizontal para drone¹¹

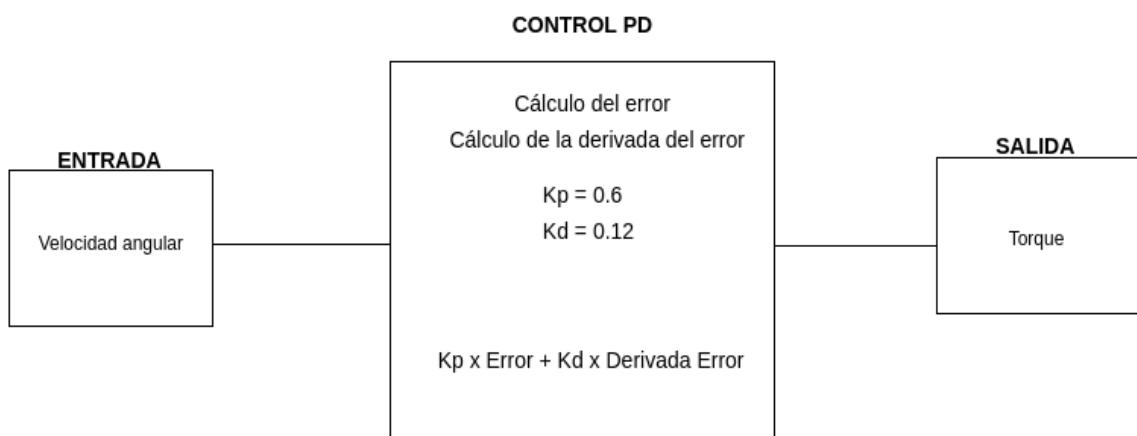


Figura 4.9: Diseño controlador PD en velocidad angular horizontal para robot terrestre¹²

- **Controlador PD en posición para la altura:** coge como entrada la componente vertical de la posición en ese instante y genera como salida la fuerza que debe ejercer el robot para mantener una posición de referencia. Se usa cuando la velocidad deseada es cero. El control en velocidad vertical cuando la velocidad consignada es cero tiembla un poco, haciendo que el movimiento del drone pierda realismo. Por esta razón, se ha optado por materializar esa velocidad vertical cero con el control en posición vertical, con el que se obtiene un mejor resultado.

¹¹Elaboración propia.

¹²Elaboración propia.

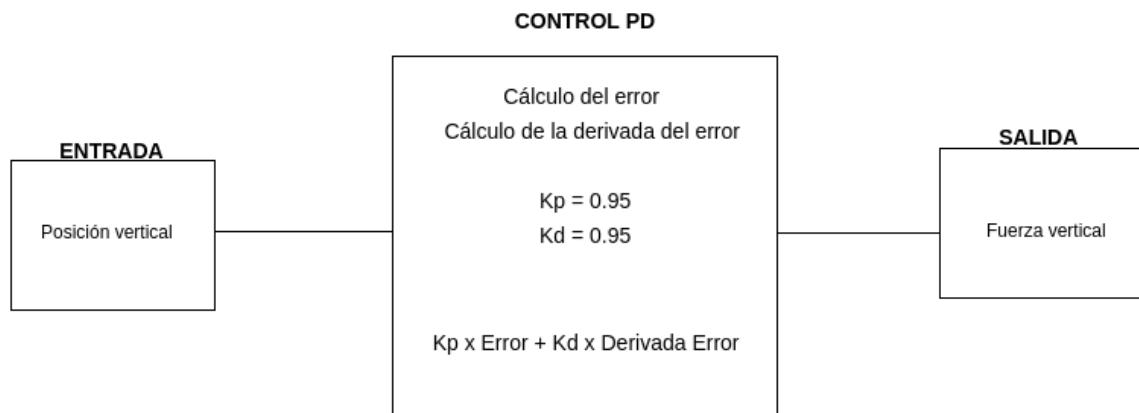


Figura 4.10: Diseño controlador PD en posición para la altura¹³

De los controladores PD implementados, son específicos para el funcionamiento del drone el controlador PD en velocidad del eje vertical (empleado en el vuelo) y el controlador PD en posición para la altura (necesario cuando el robot debe permanecer inmóvil durante el vuelo en una posición concreta). El controlador PD en velocidad del plano horizontal es empleado en el movimiento lineal tanto de robots terrestres como de drones y el controlador PD en velocidad angular horizontal se utiliza durante los giros tanto de robots terrestres como de drones. No obstante, las constantes proporcional y derivativa y el torque máximo varían en función del tipo de robot, puesto que en el caso del drone la fricción no opone resistencia durante el giro, ya que este se encuentra volando sin mantener contacto con ninguna superficie.

4.3.4. Timing

Este concepto es de especial relevancia ya que es el que hace que el motor sea complementario y no completo, es decir, es lo que permite que el motor complementario se combine satisfactoriamente con el de *CANNON* y no sobreesciba sus modificaciones.

El motor de físicas complementario se ejecuta cada 20 ms gracias a un *timeout* que lo invoca de forma periódica. Pero, ¿cuándo se ejecuta el motor de *CANNON*?

```
setTimeout(this.auxiliaryPhysics.bind(this), 20);
```

¹³Elaboración propia.

CANNON actualiza sus físicas en cada iteración del bucle de renderizado de *A-Frame*. Además, no lleva una cuenta explícita del tiempo y tampoco lo tiene en cuenta a la hora de modificar las posiciones y velocidades de los objetos de la escena. Puesto que la frecuencia de ejecución de *CANNON* es superior a la del motor complementario, ha sido necesario calcular el número de veces que se ejecuta el código *CANNON* entre dos iteraciones del motor de físicas complementario para poder realizar una correcta combinación entre ambos. El motor complementario deberá aplicar en cada iteración una aceleración X veces superior a la que corresponde, siendo X el número de veces que ha entrado *CANNON* desde la última vez que se ejecutó el motor complementario.

$$\text{Aceleración autónoma} = \text{iteracionesCANNON} \times \text{aceleración calculada}$$

El cómputo de las iteraciones de *CANNON* se ha realizado creando un nuevo componente auxiliar que hace incrementar en uno un contador por cada tick de renderizado que ejecuta *A-Frame*.

Las variables y funciones que se han añadido al código original para llevar a cabo la implementación se incluyen a continuación. Las funciones *tickCounter*, *getTickCounter* y *setTickCounter* se han utilizado para contabilizar el número de veces que se ejecuta el motor de *CANNON* entre dos iteraciones del motor complementario.

```
export var tickCounter = 0;

export function getTickCounter() {
    return tickCounter;
}

export function setTickCounter(value) {
    tickCounter = value;
}
```

Para crear el nuevo componente auxiliar se han utilizado las herramientas que ofrece *A-Frame* para el registro de nuevos componentes. La siguiente línea de código realiza el registro

del nuevo componente auxiliar que se utiliza para contabilizar las iteraciones de *CANNON*. A continuación, también se ha incluido el código del tick del nuevo componente registrado.

```
/* Registro del nuevo componente */
AFRAME.registerComponent("iterations", iterationsObj);

/* Función tick del componente "iterations" */
export var iterationsObj = {

    schema: {
        count: { type: 'number', default: 0 },
        position: { "x":0, "y":0, "z":0 }
    },
    tick: function(){
        setTickCounter(getTickCounter() + 1);
        console.log('Tick de renderizado de A-FRAME');
    }
}
```

Gracias a la correcta combinación temporal de ambos motores, se consigue que el motor de físicas complementario calcule la fuerza autónoma a aplicar a los robots dinámicos y *CANNON* materialice la gravedad y fricción de la escena. La Figura 4.11 muestra la combinación existente actualmente entre ambos motores, que permite que se hable de un motor complementario y no de un motor completo. Cada motor lleva incorporado un reloj de ejecución independiente y, en el caso de *CANNON*, es necesario disponer de un contador de ticks para monitorizar su hilo de ejecución.

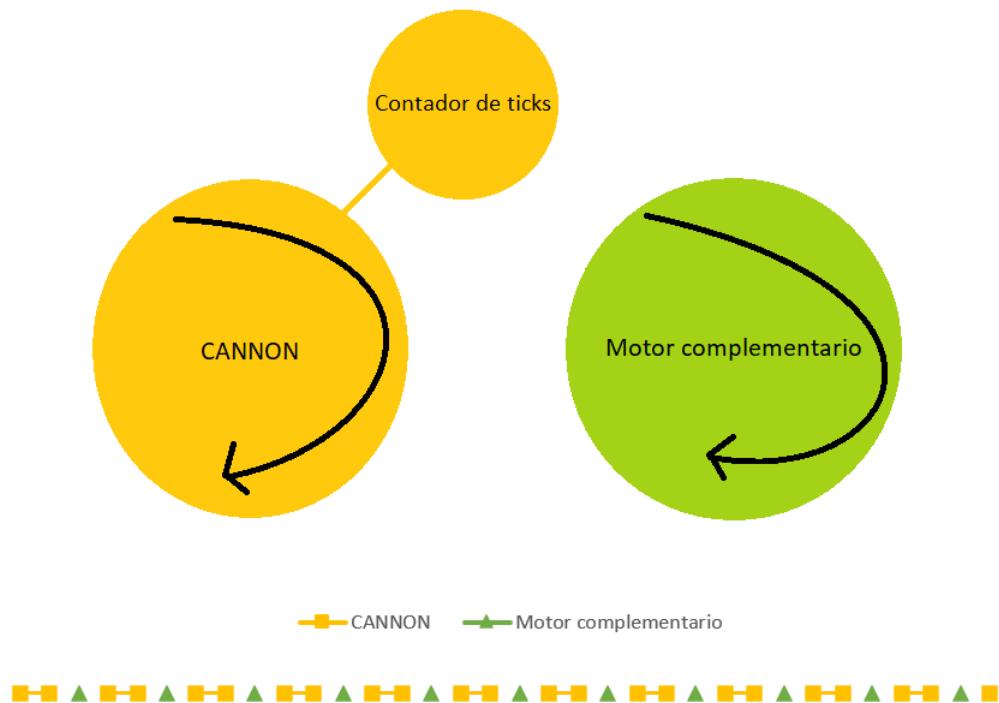


Figura 4.11: Combinación de los motores de físicas a lo largo del tiempo¹⁴

4.3.5. Implementación

A continuación se incluyen los fragmentos de código que ha sido necesario incluir en el fichero *interfacesRobot.js* del *Simcore* de *WebSim* para el correcto funcionamiento del motor complementario.

Adición de variables a la clase robot ya creada. De esta manera, el motor complementario es escalable a ejercicios multirobot. Cada robot mantiene un registro propio del valor de sus variables.

¹⁴Elaboración propia.

```
export class RobotI {  
    constructor(robotId) {  
        this.errorY = 0;  
        this.errorXZ = 0;  
        this.errorW = 0;  
        this.errorActualY = 0;  
        this.errorActualXZ = 0;  
        this.errorActualW = 0;  
        this.derivadaErrorY = 0;  
        this.derivadaErrorXZ = 0;  
        this.derivadaErrorW = 0;  
        this.forcePD = 0;  
        this.accelerationPD = 0;  
        this.commandedVelocityY = 0;  
        this.commandedVelocityXZ = 0;  
        this.commandedVelocityW = 0;  
        this.accelerationPDY = 0;  
        this.accelerationPDXZ = 0;  
        this.accelerationPDW = 0;  
        this.resultVelocity = 0;  
        this.refPos = 0;  
        this.init = true;  
        this.stop = true;  
        this.motorIterations = 0;  
    }  
}
```

Función auxiliaryPhysics. Es la función que materializa el motor complementario y se ejecuta cada 20 ms gracias al *timeout* que se ha añadido al final de la misma.

```
/*Actualización del contador de iteraciones*/
auxiliaryPhysics() {
    this.motorIterations = getTickCounter();

    /* Sólo para Drone */
    /* Si la velocidad de vuelo objetivo es 0: */
    if ((this.velocity.y <= 0.0001) || (this.velocity.y <= -0.0001)){
        if (this.init == true) {
            /* Si el movimiento aún no ha comenzado la aceleración debe ser 0 */
            this.accelerationPDY = 0;
        } else {
            /* Si el movimiento ya ha comenzado: */
            if (this.stop == true) {
                /* Si se acaba de quedar en suspensión, guardo la posición de referencia */
                this.refPos = this.robot.body.position.y;
            }
            /* Entra el controlador PD en posiciones */
            this.stop = false;
            this.accelerationPDY = this.controladorPDVerticalPos();
        }
    } else {
        this.init = false;
        this.stop = true;
        this.accelerationPDY = this.controladorPDVerticalVel();
    }

    /* Se utiliza la aceleración calculada con los controladores para obtener la
    velocidad a aplicar utilizando las fórmulas MRUA y la combinación entre motores */

    this.commandedVelocityY = this.robot.body.velocity.y +
    this.motorIterations*this.accelerationPDY;
    this.robot.body.velocity.set(this.robot.body.velocity.x,
    this.commandedVelocityY, this.robot.body.velocity.z);
}
```

```
/* Movimiento en el plano horizontal: drone y robot terreste */

/* Velocidad resultante = RaízCuadrada(Vx + Vz) */
this.resultVelocity = Math.sqrt(Math.pow(this.robot.body.velocity.x, 2) +
Math.pow(this.robot.body.velocity.z, 2));

/* Entra el controlador PD */
this.accelerationPDXZ = this.controladorPDHorizontal(this.resultVelocity);

/* Se utiliza la aceleración calculada con los controladores para obtener la velocidad
a aplicar utilizando las fórmulas MRUA y la combinación entre motores */
this.commandedVelocityXZ = this.resultVelocity +
this.motorIterations*this.accelerationPDXZ;
let rotation = this.getRotation();

/* La velocidad comandada resultante se descompone en las velocidades Vx y Vz */
this.robot.body.velocity.set(this.commandedVelocityXZ*Math.cos(rotation.y*Math.PI/180),
this.robot.body.velocity.y,this.commandedVelocityXZ*Math.sin(-rotation.y*Math.PI/180));

/* Movimiento angular: drone y robot terreste */

/* Entra el controlador PD angular */
this.accelerationPDW = this.controladorPDAngular();

/* Se utiliza la aceleración calculada con el controlador para obtener
la velocidad angular a aplicar utilizando las fórmulas MRUA y la
combinación entre motores */
this.commandedVelocityW = this.robot.body.angularVelocity.y +
this.motorIterations*this.accelerationPDW;
this.robot.body.angularVelocity.set(0, this.commandedVelocityW, 0);

setTimeout(this.auxiliaryPhysics.bind(this), 20);

}
```

Controladores PD. Se incluye como muestra del software implementado el controlador PD en velocidad del plano horizontal y el controlador PD en posición para la altura. Los controladores restantes presentan un código muy similar a estos.

```
/* Código del controlador PD en velocidad del plano horizontal */
controladorPDHorizontal(resultVelocity) {
    /* Definición constantes para el controlador, fuerza máxima y aceleración máxima */
    const mass = this.robot.body.mass;
    const kp = 0.45*mass;
    const kd = 0.01*mass;
    const fMax = 100000000;
    const accelerationMax = fMax / mass;

    /* Cálculo del error y derivada del error*/
    this.errorActualXZ = this.velocity.x - resultVelocity;
    this.derivadaErrorXZ = this.errorActualXZ - this.errorXZ;
    this.errorXZ = this.errorActualXZ;

    /* Salida del controlador */
    this.forcePD = kp*this.errorActualXZ + kd*this.derivadaErrorXZ;

    /* Obtención de la aceleración teniendo en cuenta la masa */
    this.accelerationPD = this.forcePD / mass;

    /* Límite de aceleración aplicable en cada iteración */
    if (Math.abs(this.accelerationPD) > angularAccelerationMax) {
        if (this.accelerationPD > 0) {
            this.accelerationPD = angularAccelerationMax;
        } else {
            this.accelerationPD = - angularAccelerationMax;
        }
    }
    return this.accelerationPD;
}

/* Código del controlador PD en posición para la altura*/
controladorPDVerticalPos() {
    /* Definición constantes para el controlador, fuerza máxima y
```

```

const mass = this.robot.body.mass;
const kp = 0.95*mass;
const kd = 0.95*mass;
const fMax = 100000000;
const accelerationMax = fMax / mass;

/* Cálculo del error y derivada del error*/
this.errorActualY = this.refPos - this.robot.body.position.y;
this.derivadaErrorY = this.errorActualY - this.errorY;
this.errorY = this.errorActualY;

/* Salida del controlador */
this.forcePD = kp*this.errorActualY + kd*this.derivadaErrorY;

/* Obtención de la aceleración teniendo en cuenta la masa */
this.accelerationPD = this.forcePD / mass;

/* Límite de aceleración aplicable en cada iteración */
if (Math.abs(this.accelerationPD) > angularAccelerationMax) {
    if (this.accelerationPD > 0) {
        this.accelerationPD = angularAccelerationMax;
    } else {
        this.accelerationPD = - angularAccelerationMax;
    }
}
return this.accelerationPD;
}

}

```

4.4. Validación experimental

4.4.1. Simulación realista de robots terrestres

El motor de físicas complementario permite dotar a *WebSim* de un motor de físicas más realistas puesto que únicamente actúa sobre la fuerza autónoma del robot, dejando libertad a *CANNON* para materializar tanto fricción como gravedad. Gracias a ello, se simulan mundos

en los que los requisitos se asemejan en mayor medida a las características que se encuentran en el mundo real y en los robots físicos.

En consecuencia, se van a poder desarrollar ejercicios más variados como pistas de hielo o ejercicios multinivel con rampas que interconecten los múltiples niveles. El modelo de físicas es ahora mucho más realista, pero hay que configurarlo adecuadamente para cada robot eligiendo valores apropiados para sus parámetros. A continuación, se incluyen varios ejemplos en los que se puede observar cómo, con una correcta configuración de los atributos, se puede lograr la simulación de situaciones muchos más diversas y realistas:

- **Fuerza máxima = 0.1.** No es posible ascender por una rampa dado que la fuerza máxima del mBot no es suficiente para superar la fricción. Esta situación puede solucionarse disminuyendo la fricción de la escena, reduciendo la inclinación de la rampa o aumentando la fuerza máxima del robot¹⁵.

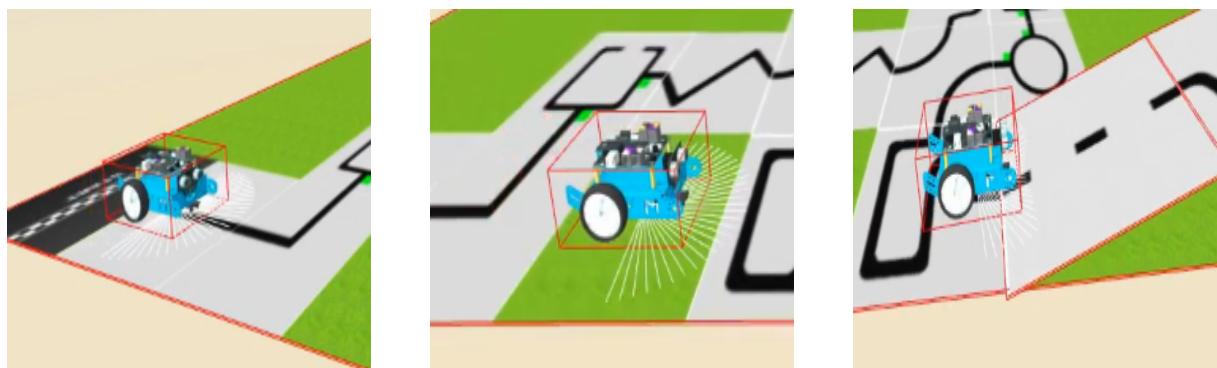


Figura 4.12: Subida de rampa con una fuerza máxima insuficiente

- **Fricción = 0.0000001.** El escenario se puede asimilar a una pista de hielo. Cuando se le consigna al robot una instrucción en posición, por ejemplo 'gira 90° a la izquierda', se observa cómo el robot no es capaz de frenar el movimiento a los 90° a pesar de que su controlador PD angular comienza a comandar fuerzas para tratar de frenarlo. Sin embargo, como la fricción es excesivamente pequeña el controlador tarda más tiempo en lograr frenar el movimiento, tal y como ocurriría en una pista de hielo¹⁶. La Figura 4.13 muestra la ejecución de esta instrucción y se observa cómo el robot finaliza el movimiento

¹⁵<https://youtu.be/LQk9GLoMIk0>

¹⁶<https://www.youtube.com/watch?v=QfPmtPeEL5k>

prácticamente a los 180° por esta razón.

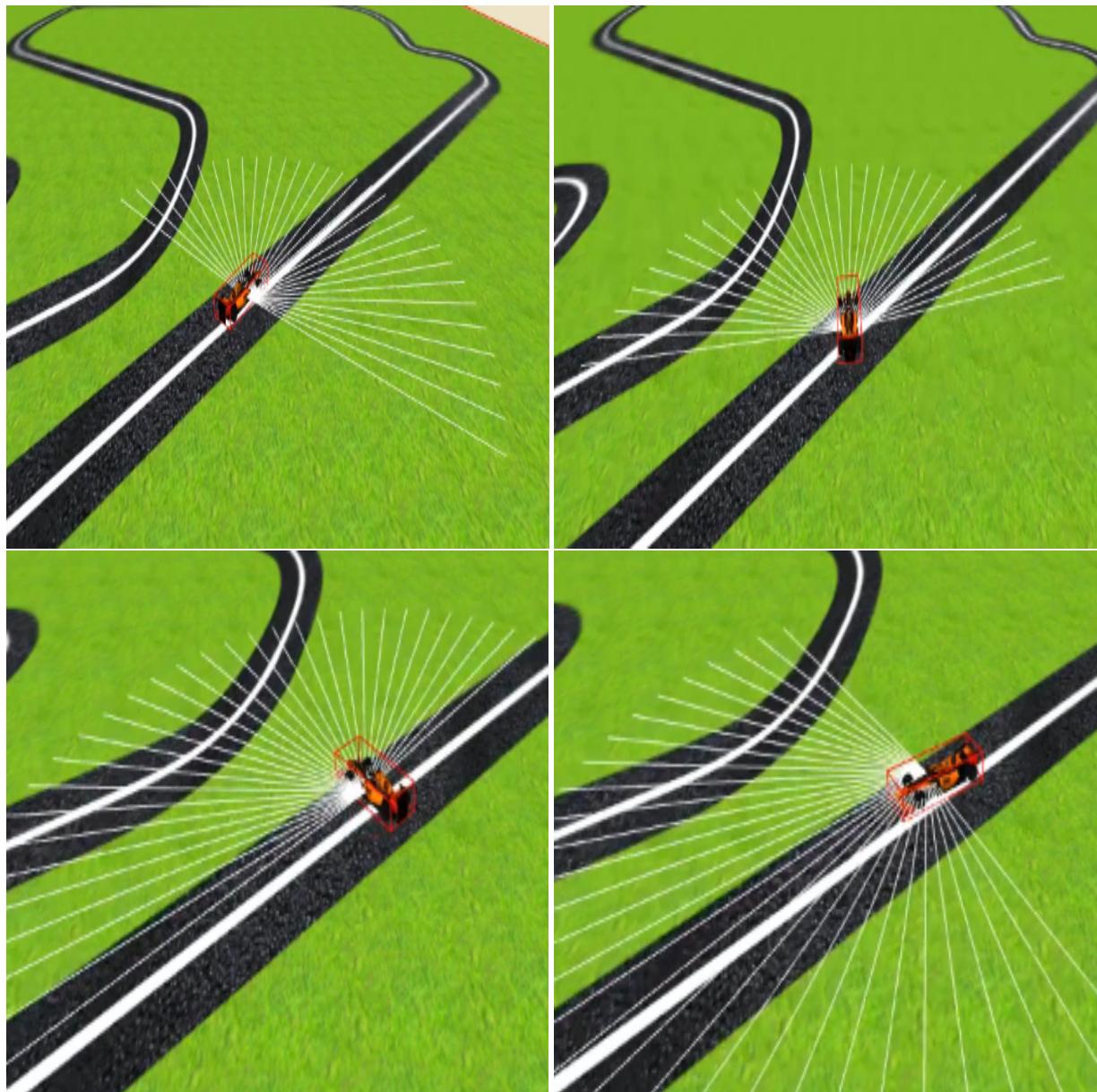


Figura 4.13: Giro de 90° hacia la izquierda en una superficie con una fricción muy baja

La Figura 4.14 muestra cómo influye en la aceleración autónoma que aplica el motor de físicas complementario el valor de fricción que se parametrice en el escenario. Cuanto mayor es la fricción, más grande es la aceleración autónoma que se debe aplicar.

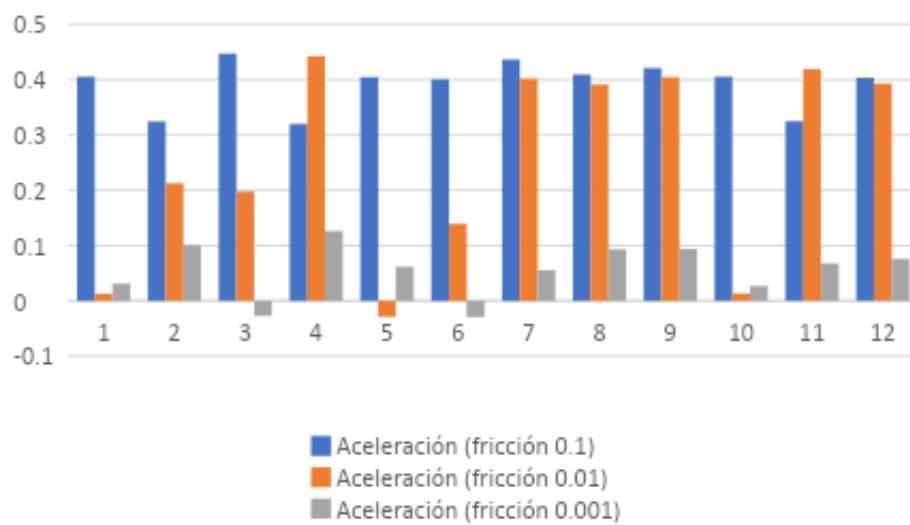


Figura 4.14: Relación fricción - aceleración

Por último, la Figura 4.15 muestra la diferencia entre la progresión de la velocidad a lo largo del tiempo con y sin el motor de físicas complementario. Se observa que con el motor de físicas complementario la velocidad tarda unos milisegundos en estabilizarse y que cada vez se va ajustando mejor al valor objetivo. Sin embargo, la curva que genera la simulación sin el motor complementario es absolutamente irreal, ya que se está asumiendo una aceleración infinita.

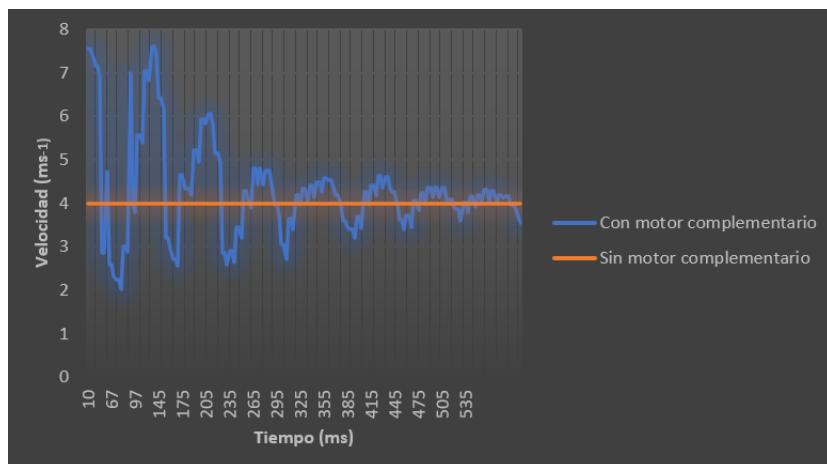
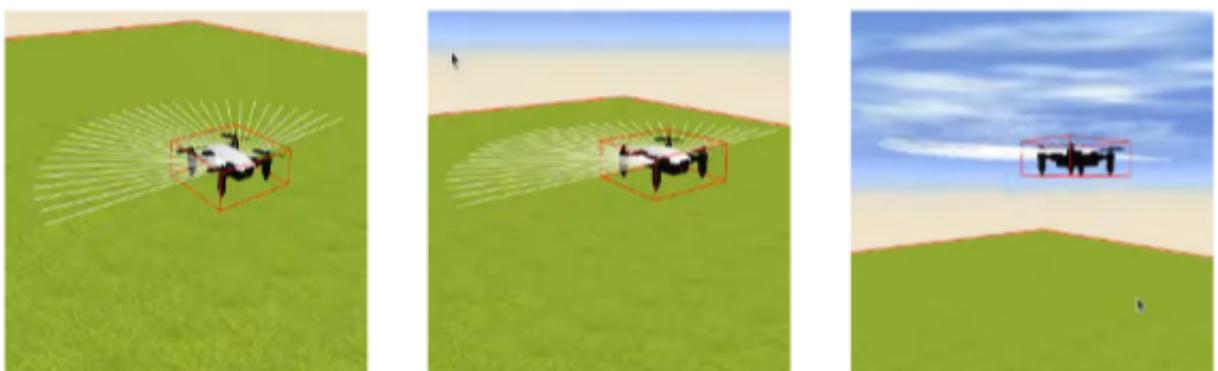


Figura 4.15: Tiempo - Velocidad Controlador PD en velocidad del plano horizontal

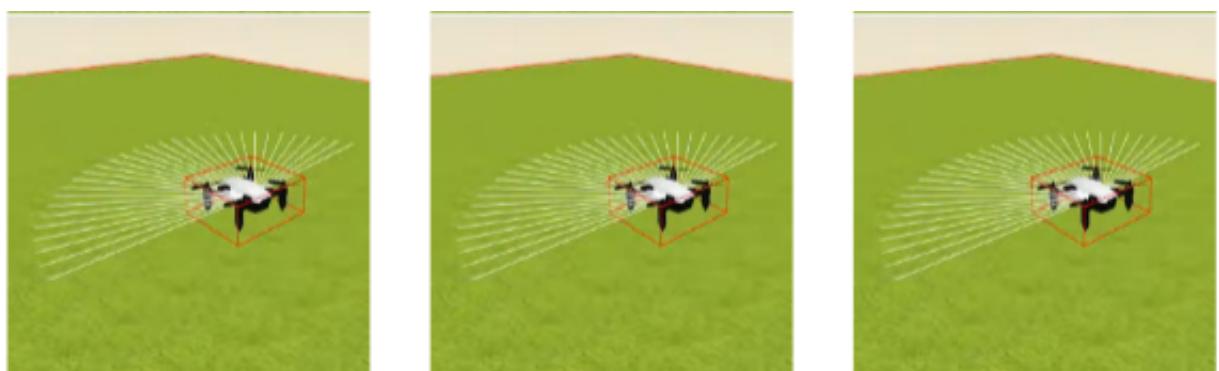
4.4.2. Simulación realista de drones

El motor de físicas complementario dota a *WebSim* de unas físicas más realistas puesto que permite a *CANNON* materializar la gravedad. Gracias a ello, se pueden simular mundos en los que, a pesar de existir una gravedad de -9.8, la fuerza autónoma que hace aplicar el motor complementario permite volar al drone.

Las físicas realistas implementadas permiten diferenciar entre el despegue de un drone ligero y el de uno mucho más pesado. Por ejemplo, para un fuerza máxima de 1 N, un drone de 1 Kg podrá despegar mientras que otro de 100 Kg no será capaz puesto que la fuerza máxima aplicada no es suficiente para superar la fuerza de la gravedad¹⁷.



(a) Despegue del drone Tello de 1 Kg



(b) Despegue del drone Tello de 100 Kg

Figura 4.16: Despegue con drones de diferentes masas

Por otro lado, el control PD en posición incrementa el realismo de las físicas también a nivel visual puesto que hace recrear un movimiento más fluido y no produce una brusca frenada co-

¹⁷<https://www.youtube.com/watch?v=S1E1zKjFkLo>

mo ocurría con la imposición de la posición con *updatePosition*. La fluidez del movimiento se consigue gracias a la fase de estabilización característica del controlador PD hasta que consigue aproximar su salida al valor de referencia¹⁸. En la Figura 4.17 se aprecia cómo el controlador tarda unos ms en estabilizar la posición y al principio la posición tiende a caer como consecuencia de la atracción de la gravedad. La Figura 4.18 muestra la progresión de la velocidad a lo largo del tiempo cuando el drone despegue.

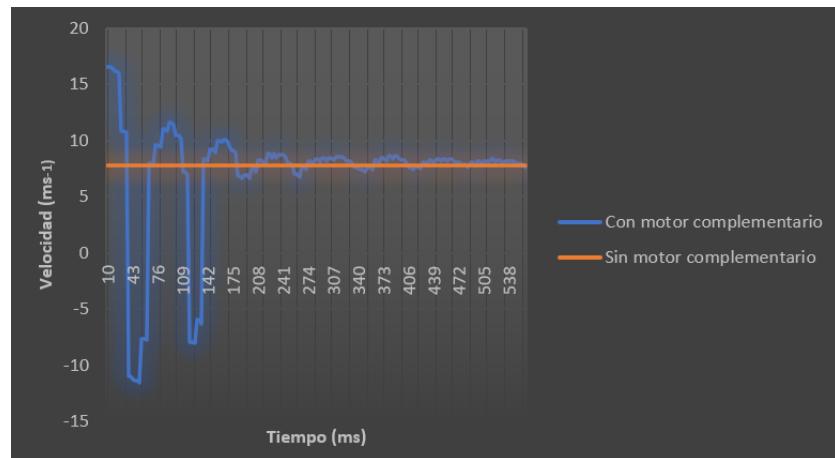


Figura 4.17: Tiempo - Posición Controlador PD en posición para la altura

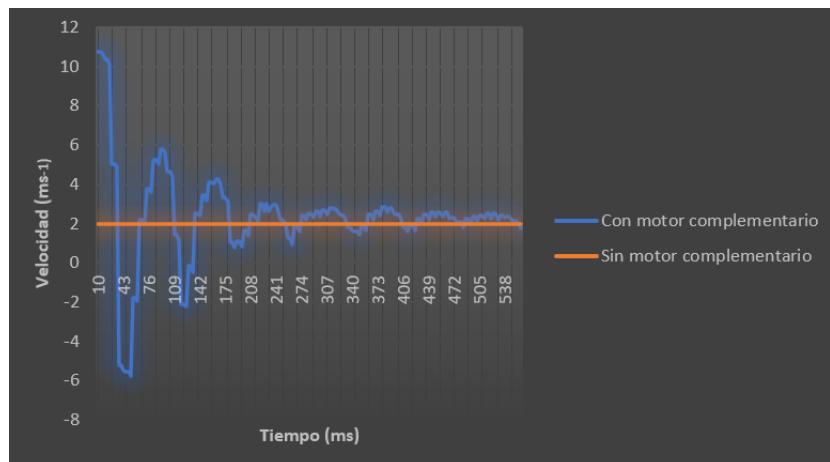


Figura 4.18: Tiempo - Velocidad Controlador PD en velocidad del eje vertical

¹⁸<https://www.youtube.com/watch?v=jSGI6KJbzTQ>

Capítulo 5

Nuevos ejercicios con físicas realistas

A modo de validación experimental, en este capítulo se presentan los nuevos ejercicios que se han creado para enriquecer el contenido educativo ofrecido en el entorno de *Kibotics*. Todos ellos permiten explotar las mejoras que ofrece el motor de físicas complementario implementado.

5.1. Sigue-líneas con rampa

Este ejercicio¹ se basa en la propuesta realizada para la competición *Robocup Junior Australia 2019*². Se trata de una versión mejorada del sigue-líneas tradicional, en el que se incluyen trayectorias más complejas y diversas, y dos niveles diferentes conectados a través de una rampa. Actualmente, este ejercicio está disponible en la plataforma *Kibotics* como un ejercicio a resolver en *Python o Scratch* y el robot que se ha seleccionado para su solución ha sido el mBot.

¹<https://www.youtube.com/watch?v=PjTr13M3o5k>

²<https://www.robocupjunior.org.au/>

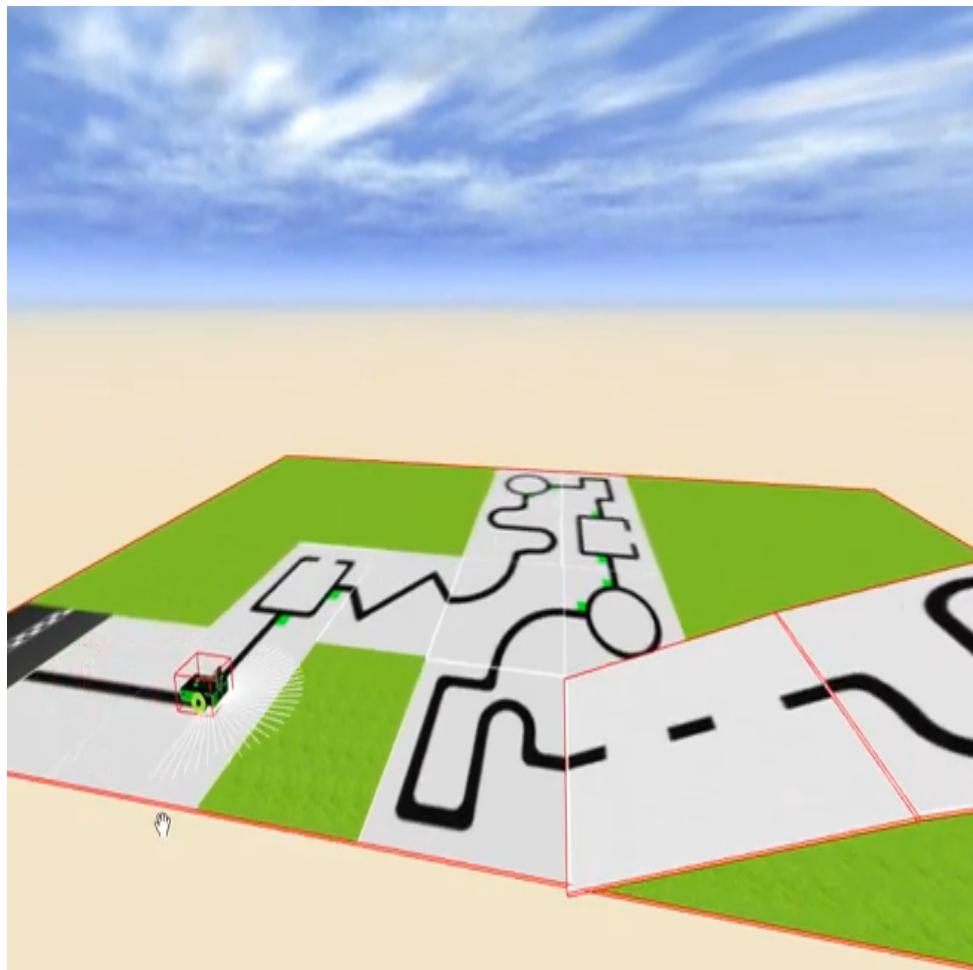


Figura 5.1: Escenario Sigue-líneas con rampa

El ejercicio aprovecha las ventajas del motor de físicas complementario en la subida de las rampas. Durante la subida, *CANNON* materializa la fricción oponiendo fuerzas de rozamiento al movimiento, mientras que el motor complementario se encarga de aplicar la fuerza autónoma necesaria para lograr que el robot ascienda por la rampa. La fuerza resultante será más grande que la necesaria para avanzar por el suelo plano sin inclinación y, además, será necesario tener en cuenta el valor de la fricción y la inclinación de la rampa para hacer más fácil o difícil la subida. Tal y como ocurriría en la realidad con un robot físico, se alcanzará un grado de inclinación o un valor de fricción con los cuales el mBot no será capaz de avanzar por la rampa puesto que el diseño del motor incluye un límite de fuerza máxima aplicable³.

³<https://www.youtube.com/watch?v=8mtK8DVkDZo>

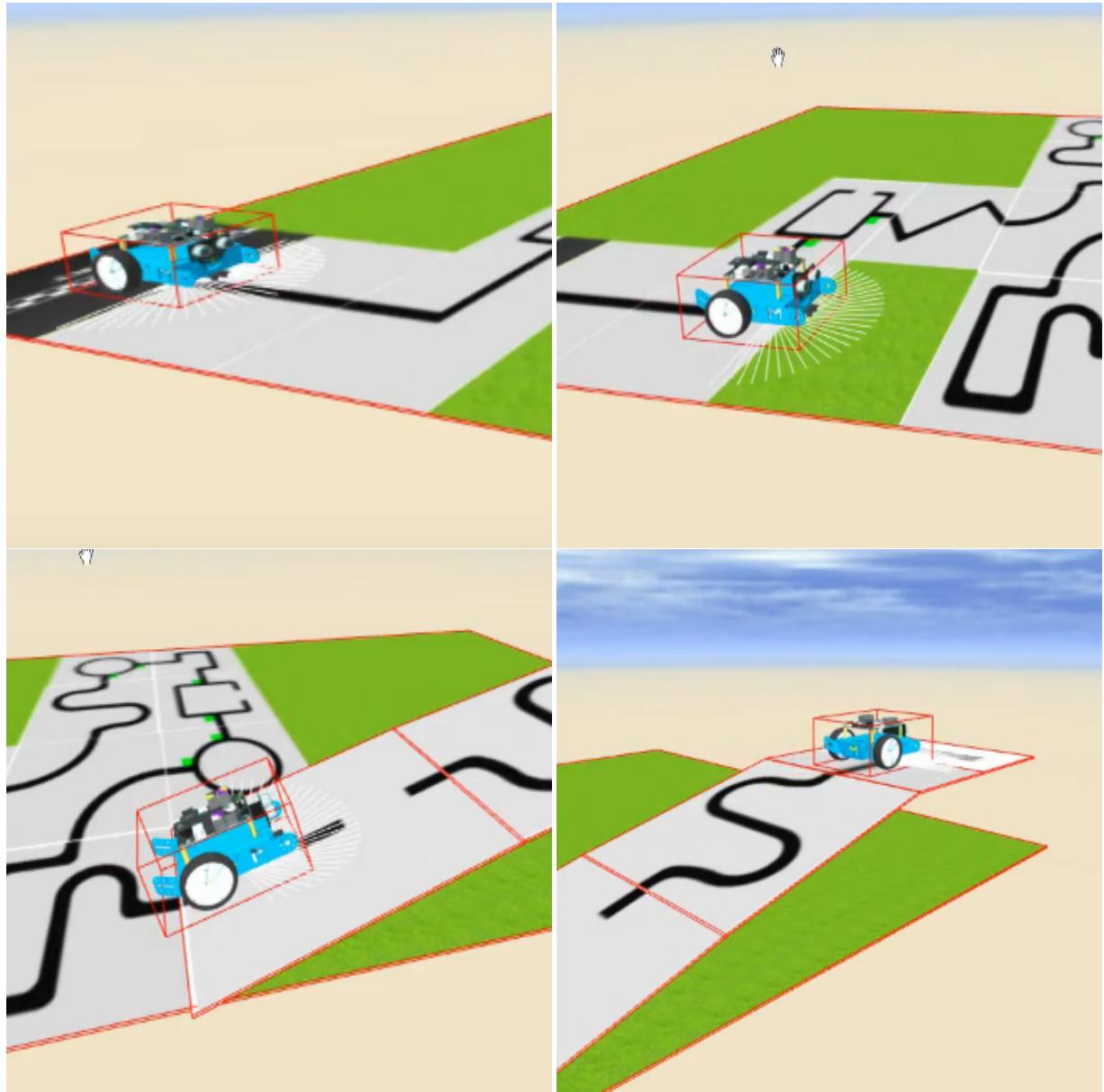


Figura 5.2: Fotograma del movimiento del ejercicio Sigue-líneas con rampa

A continuación, en la tabla 5.1, se recoge el valor de los parámetros del modelo de fuerzas y de *A-Frame* necesarios para el correcto funcionamiento del motor de físicas complementario en este ejercicio.

Parámetros del modelo de fuerzas	
mass	1
inertia	1.3
Fmax	10
Tmax	1
accelerationMax	10
angularAccelerationMax	0.77
linealSpeedMax	10
angularSpeedMax	5
Parámetros de <i>A-Frame</i>	
restitution	0.3
gravity	-9.8
friction	0.00003
linearDamping	-1.3
angularDamping	-1.3

Cuadro 5.1: Parámetros de configuración del modelo de fuerzas y de *A-Frame* del ejercicio sigue-líneas con rampa

5.2. Laberinto 3D para mBot

Este segundo ejercicio⁴ también se basa en la propuesta realizada para la competición *Robocup Junior Australia 2019*⁵. El objetivo de este ejercicio es hacer llegar al robot al punto en el que se encuentra otro robot perdido para rescatarle del laberinto.

Este ejercicio está ya disponible para el mBot en el entorno de *Kibotics* para *Python* y *Scratch*. Además, al tratarse de un laberinto multinivel, se ve beneficiado por el motor de físicas complementario en la subida de la rampa, igual que ocurría en el ejercicio anterior⁶.

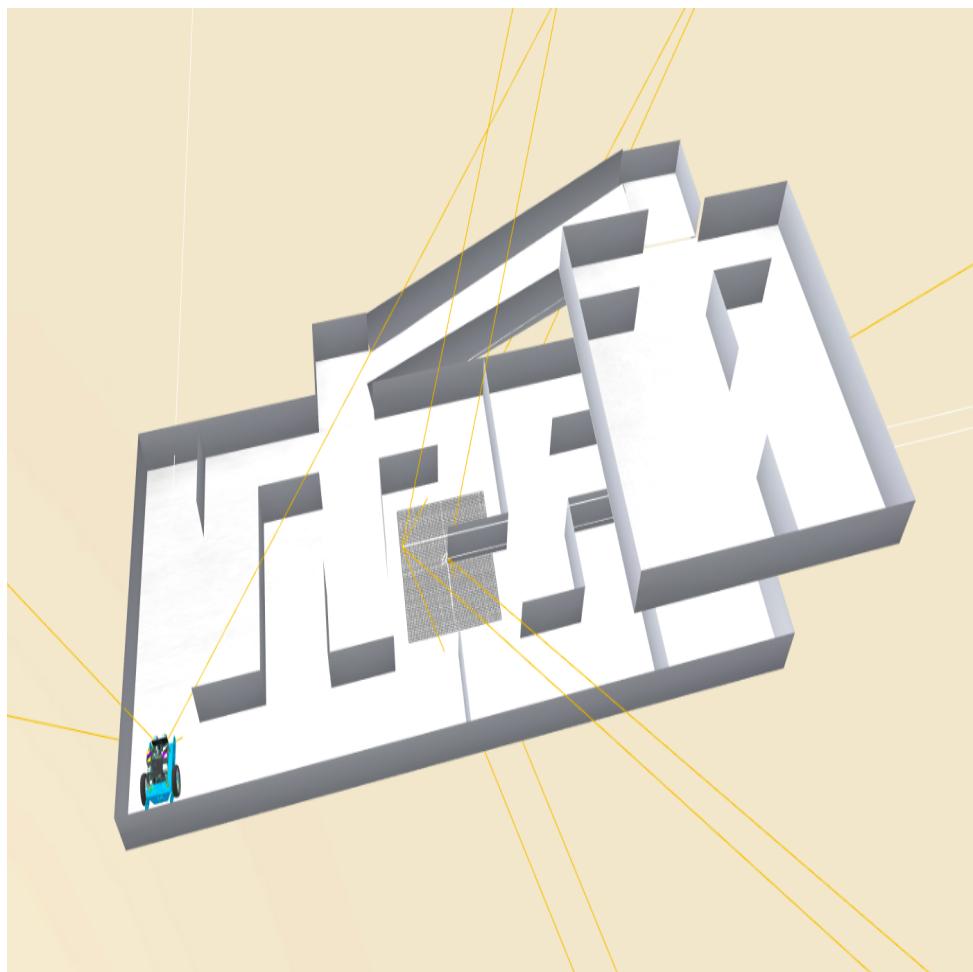


Figura 5.3: Laberinto 3D para mBot

⁴<https://www.youtube.com/watch?v=R9JXZCeSNFo>

⁵<https://www.robocupjunior.org.au/>

⁶<https://www.youtube.com/watch?v=76kgItpMpGk>

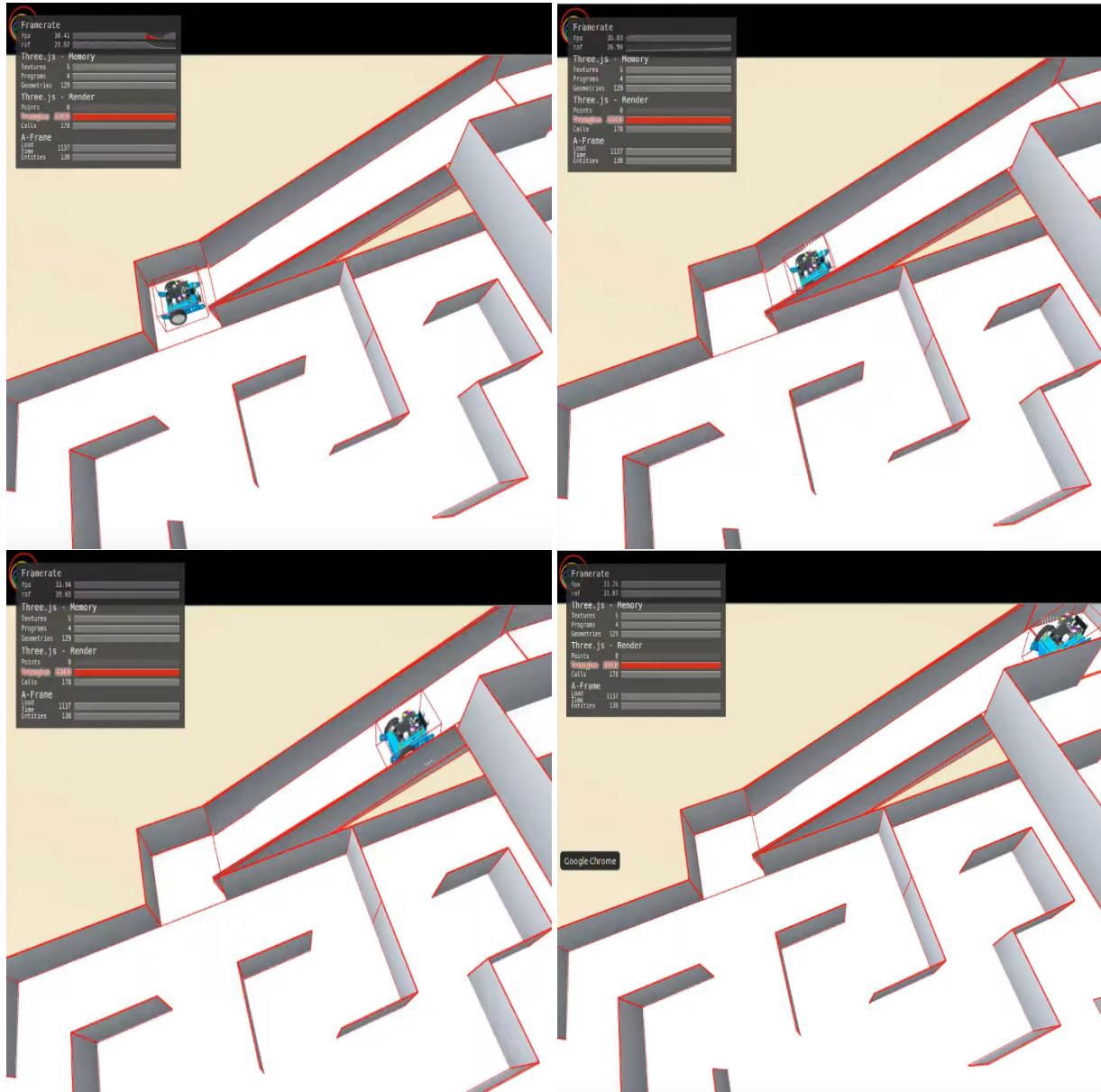


Figura 5.4: Fotograma del movimiento del ejercicio Laberinto 3D para mBot

A continuación, en la tabla 5.2, se recoge el valor de los parámetros del modelo de fuerzas y de *A-Frame* necesarios para el correcto funcionamiento del motor de físicas complementario en este ejercicio.

Parámetros del modelo de fuerzas	
mass	1
inertia	1.3
Fmax	10
Tmax	1
accelerationMax	10
angularAccelerationMax	0.77
linealSpeedMax	10
angularSpeedMax	5
Parámetros de A-Frame	
restitution	0.3
gravity	-9.8
friction	0.0005
linearDamping	-1.3
angularDamping	-1.3

Cuadro 5.2: Parámetros de configuración del modelo de fuerzas y de *A-Frame* del ejercicio laberinto 3D para mBot

5.3. Laberinto para drone

Se han incluido dos ejercicios nuevos para el drone Tello: laberinto para drone con y sin señalización. El objetivo de ambos ejercicios es que el drone encuentre la salida de un laberinto tridimensional. Este ejercicio aprovecha el motor de físicas complementario en el movimiento del drone, tanto durante el vuelo (controlador PD en velocidad) como cuando el drone permanece quieto a una cierta altura (controlador PD en posición)⁷.

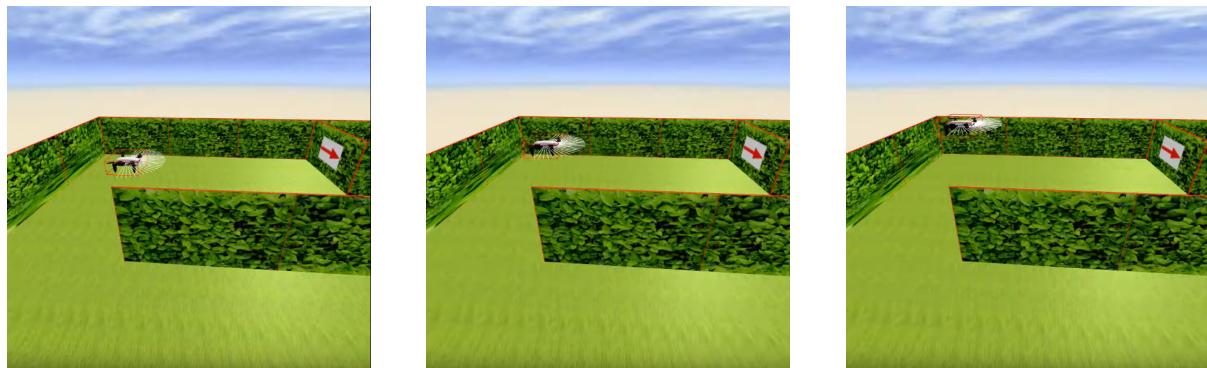


Figura 5.5: Fotograma del movimiento del ejercicio Laberinto para drone

A continuación, en la tabla 5.3, se recoge el valor de los parámetros del modelo de fuerzas y de *A-Frame* necesarios para el correcto funcionamiento del motor de físicas complementario en este ejercicio.

⁷<https://www.youtube.com/watch?v=jSGI6KJbzTQ>

Parámetros del modelo de fuerzas	
mass	1
inertia	1.3
Fmax	10
Tmax	1
accelerationMax	10
angularAccelerationMax	0.77
linealSpeedMax	10
angularSpeedMax	5
Parámetros de A-Frame	
restitution	0.3
gravity	-9.8
friction	0.0000001
linearDamping	0.01
angularDamping	0.01

Cuadro 5.3: Parámetros de configuración del modelo de fuerzas y de *A-Frame* del ejercicio laberinto para drone

5.3.1. Sin señalización

La primera versión⁸ está pensada para que el usuario logre hacer que el drone encuentre la salida mediante el uso de instrucciones en posición. Por ejemplo, 'avanza 2 metros' o 'gira a la derecha'.

⁸<https://www.youtube.com/watch?v=RLjZPhP6P3g>

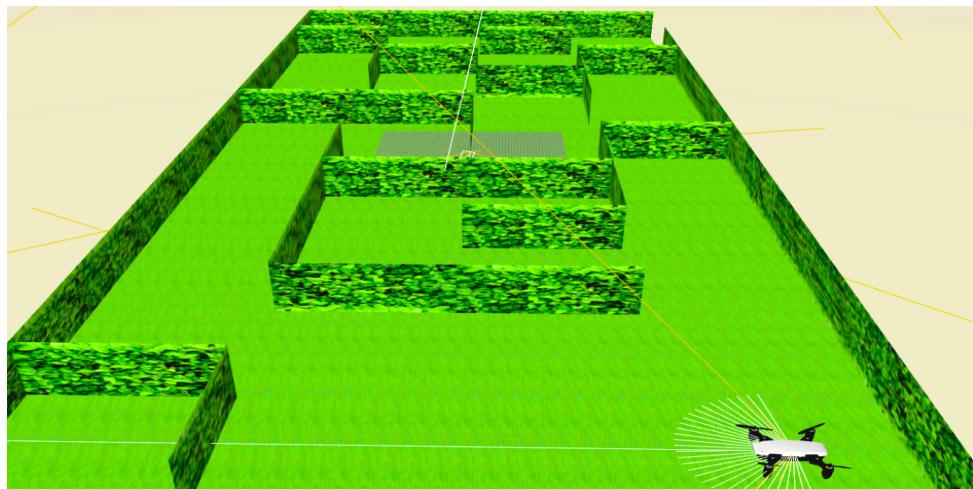


Figura 5.6: Laberinto para drone sin señalización

5.3.2. Con señalización

La segunda versión⁹ se ha implementado con el objetivo de que se utilicen los sensores y cámaras del drone para que se capten las señales que se han colocado por las paredes y que guían al drone para encontrar la salida. Gracias a la inteligencia artificial, el drone será capaz de reconocer las señales y traducirlas en un movimiento determinado.

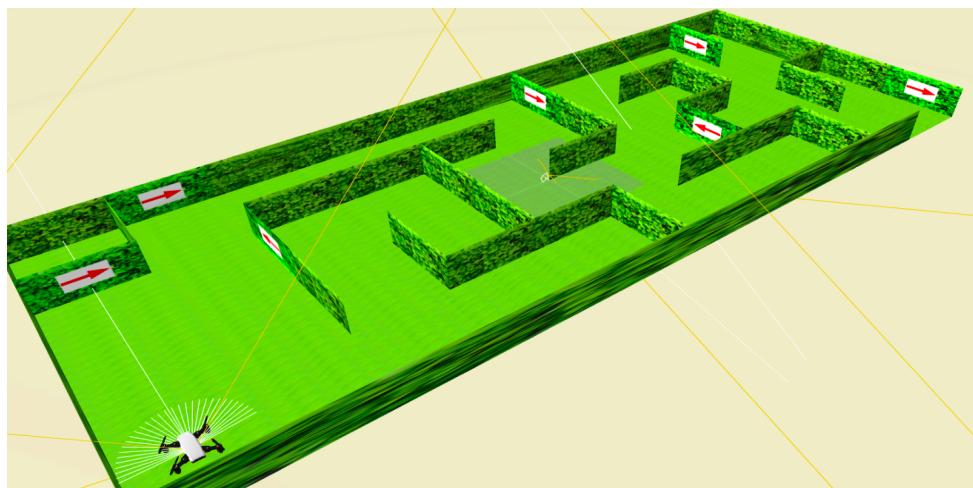


Figura 5.7: Laberinto para drone con señalización

⁹<https://www.youtube.com/watch?v=9tSqJQ7pqLc>

5.4. Fútbol competitivo

Este último ejercicio¹⁰ también se basa en una de las propuestas presentadas para la competición *Robocup Junior Australia 2019*¹¹. El objetivo de este ejercicio competitivo es meter más goles que el contrincante, es decir, simular un partido de fútbol uno contra uno. El primero que llegue a diez goles, será el ganador. Para ello, se ha implementado un evaluador que lleva la cuenta de los goles metidos por cada equipo¹²(Figura 5.9).

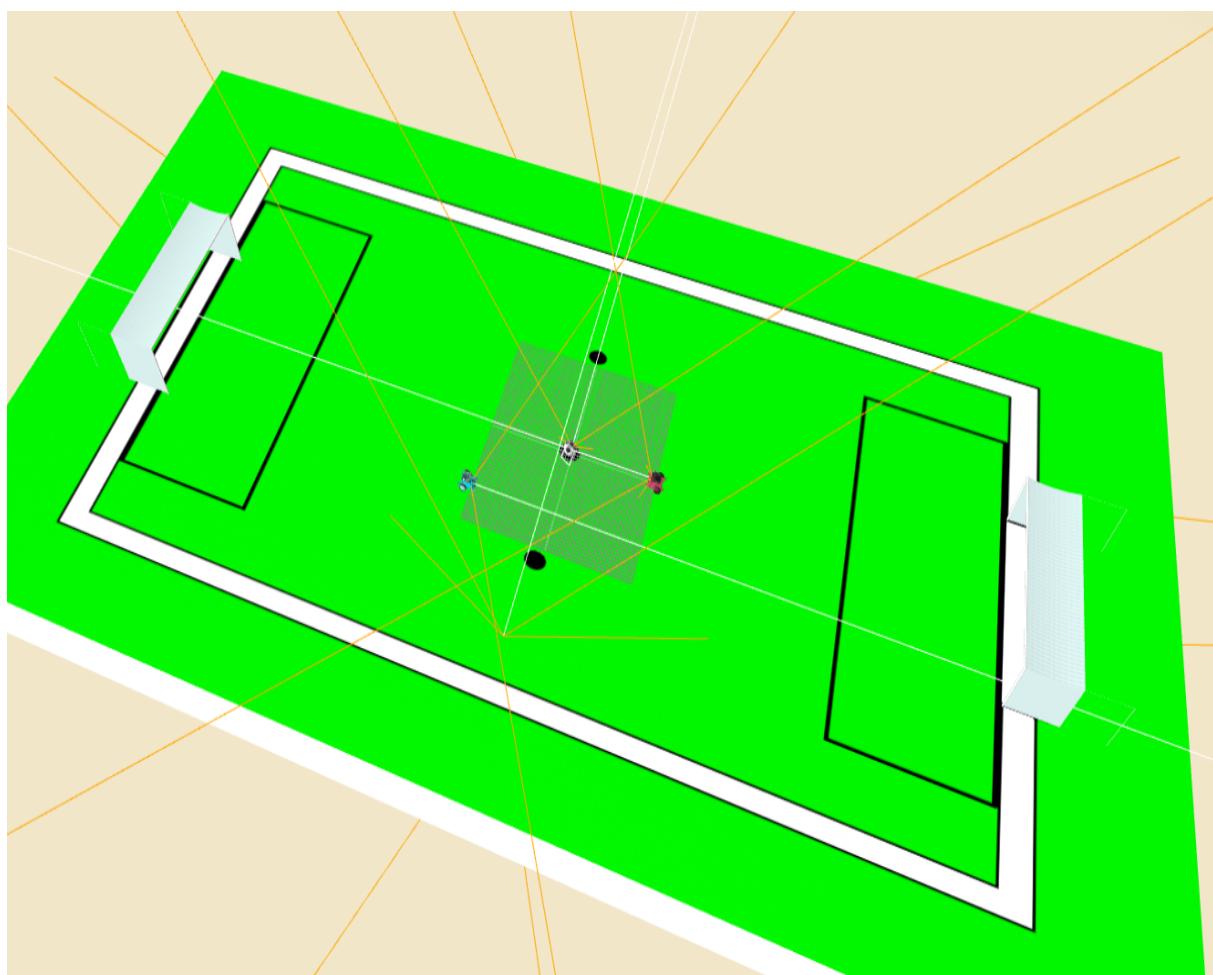


Figura 5.8: Fútbol competitivo

Este ejercicio saca provecho del estudio realizado en las físicas y del motor de físicas complementario tanto en los choques entre los robots y el balón como en el movimiento del balón. Ha sido necesaria la definición de un coeficiente de restitución adecuado para hacer realistas

¹⁰<https://www.youtube.com/watch?v=FYtAF1l4keU>

¹¹<https://www.robocupjunior.org.au/>

¹²<https://www.youtube.com/watch?v=lgwECFpTgNk>

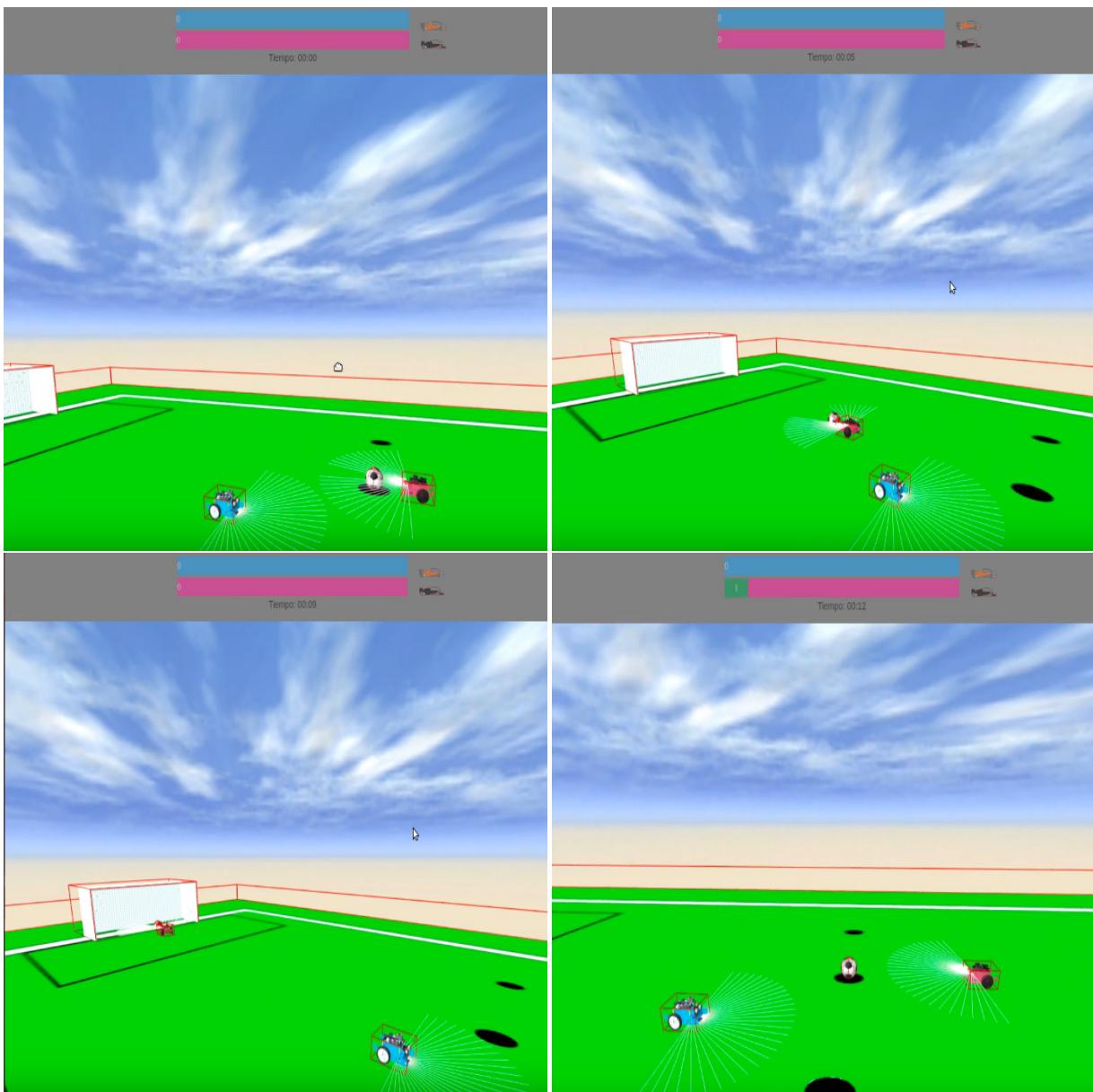


Figura 5.9: Evaluador del ejercicio Fútbol competitivo

tanto los choques como el movimiento de la pelota. En este vídeo¹³ se observa un movimiento no realista del balón, mientras que en este otro¹⁴ se ha realizado el ajuste del coeficiente de restitución y la fricción, por lo que la pelota va girando sobre sí misma mientras va avanzando, haciendo mucho más realista el movimiento.

¹³https://www.youtube.com/watch?v=7W-FB3E0B_I

¹⁴<https://www.youtube.com/watch?v=PIJRqBGPeH4>

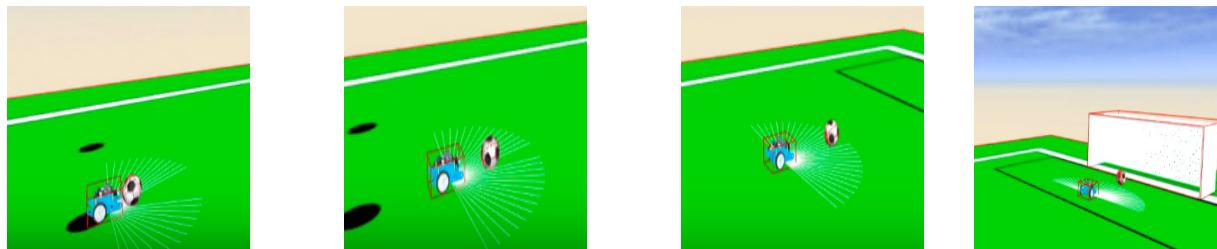


Figura 5.10: Fotograma del movimiento del ejercicio Fútbol competitivo

A continuación, en la tabla 5.4, se recoge el valor de los parámetros del modelo de fuerzas y de *A-Frame* necesarios para el correcto funcionamiento del motor de físicas complementario en este ejercicio.

Parámetros del modelo de fuerzas	
mass	1
inertia	1.3
Fmax	10
Tmax	1
accelerationMax	10
angularAccelerationMax	0.77
linealSpeedMax	10
angularSpeedMax	5
Parámetros de <i>A-Frame</i>	
restitution	0.5
gravity	-9.8
friction	0.0005
linearDamping	0.01
angularDamping	0.01

Cuadro 5.4: Parámetros de configuración del modelo de fuerzas y de *A-Frame* del ejercicio fútbol competitivo

Capítulo 6

Conclusiones

En este apartado se recogen las conclusiones a las que se ha llegado tras la realización de este Trabajo Fin de Grado y se van a valorar los resultados obtenidos. De igual forma, tras exponer las conclusiones y valoraciones, se presentarán algunas interesantes líneas futuras para mejorar y completar la implementación realizada.

6.1. Conclusiones y valoración de resultados

El objetivo principal de este trabajo era crear un motor de físicas mejorado para *WebSim* que estuviera basado en tecnologías web y que permitiese dotar al simulador robótico de unas físicas más realistas. Se ha cumplido de manera satisfactoria implementando un nuevo motor de físicas complementario que coexiste con *CANNON*. Materializa la fuerza autónoma de los robots, dejando como tarea de *CANNON* la materialización de la gravedad, la fricción y la restitución de la escena para esos mismos robots.

Anteriormente, las físicas de los robots se implementaban por medio de la función *updatePosition*, que imponía una posición y velocidad al robot sin tener en cuenta la gravedad, la fricción ni la restitución. Era un modelo cinemático donde se asumía que los robots adquirían instantáneamente la velocidad ideal que el software del robot ordenaba. En consecuencia, los robots no se movían de manera realista puesto que se estaba asumiendo una aceleración infinita. Sin embargo, el nuevo motor complementario sí permite un movimiento realista puesto que al ser complementario a *CANNON*, no elimina el efecto que este introduce con la fuerza de

gravedad, la fricción y las colisiones, abriendo un amplio abanico de nuevas posibilidades de simulación en *WebSim*.

El nuevo motor complementario consta de dos niveles. El primero incluye varios controladores PD que se encargan de traducir las velocidades deseadas que le llegan al motor complementario cada 20 ms a la fuerza autónoma a aplicar al robot para que este las alcance. El segundo nivel es un modelo de fuerzas que, a partir de la definición de la masa y el momento de inercia del robot, calcula la aceleración y par resultantes al aplicar la fuerza autónoma decidida por el primer nivel.

Los dos puntos que se pedían para cumplir este principal objetivo también se han logrado en la implementación:

- Materialización de robots con distinta masa y que recreen un movimiento autónomo realista, con una aceleración máxima limitada y capacidad de control acotada.

Este requisito se ha logrado satisfacer mediante la definición de una serie de parámetros que caracterizan el movimiento del robot en un escenario y las físicas del mismo. Se deben configurar para cada ejercicio. Algunos de estos atributos son propios de *A-Frame* y otros son parámetros del modelo de fuerzas del nuevo motor complementario. En la tabla 4.4 se incluía el detalle de cada uno de estos parámetros.

Además de una mejora visual ofreciendo un movimiento más fluido y realista, permite recrear nuevas posibilidades de simulación, como por ejemplo:

- La variación de la fricción del escenario permite recrear escenarios tan diversos como una pista de hielo o un campo de fútbol por donde rueda la pelota y avanzan los robots.
- La materialización de la gravedad hace que sea necesario ejercer una fuerza autónoma capaz de superar la fuerza de la gravedad y hacer ascender al drone. Los objetos en los mundos de drones ahora sí pueden tener gravedad, antes estaba anulada.
- La variación de la masa de los robots implica que sea necesario ajustar el resto de parámetros. Por ejemplo, un robot más pesado deberá ejercer una fuerza más grande que un robot más ligero para alcanzar la misma velocidad.

- Los escenarios podrán incluir rampas, ya que los robots terrestres podrán subirlas si se parametrizan de forma correcta los valores de fricción y fuerza autónoma máxima.
 - Los robots tienen velocidad lineal y angular límite, al contrario de lo que ocurría con el modelo anterior.
 - Se rompe con la premisa de aceleración infinita que se ha mantenido hasta el momento por el hecho de no imponer instantáneamente la velocidad deseada. Se tiene una fuerza máxima y, por lo tanto, una aceleración máxima.
 - Por el hecho de que el motor sea complementario, *CANNON* materializa en paralelo la restitución de las colisiones, permitiendo recrear movimientos más realistas como el golpeo de una pelota.
- Coexistencia con el motor por defecto *CANNON* y que no requiera la modificación de su código fuente.

El segundo punto se consiguió satisfacer gracias a la correcta combinación en el tiempo entre ambos motores que permite que cada uno de ellos materialice la parte de las físicas que le compete de manera independiente y que trabajen en la misma escala. Para ello, fue necesario conocer el ritmo de ejecución de ambos motores. En el caso del nuevo motor complementario es de 20 ms puesto que se incluyó un *timeout* para establecer ese ritmo. En el caso de *CANNON* fue necesario el registro de un nuevo componente auxiliar en *A-Frame* que se debe incluir en todas las escenas y que permite monitorizar el ritmo de ejecución del bucle de renderizado de *A-Frame*. En cada iteración del bucle de renderizado, *CANNON* actualiza las posiciones y velocidades de los objetos de la escena materializando su parte de la dinámica.

El segundo objetivo era crear varios ejercicios en la plataforma educativa Kibotics que sacasen partido del nuevo motor de físicas y fueran más atractivos para los niños. Este objetivo también se ha completado de manera satisfactoria mediante la creación de cuatro escenarios diferentes que explotan las nuevas físicas de realistas de diferentes formas.

- Creación de escenarios multinivel gracias a la materialización de la fricción, que permite la subida de rampas: sigue-líneas con rampa y laberinto 3D para mBot.

- Recreación del movimiento efectuado por una pelota rodando gracias a la materialización de la fricción: fútbol competitivo.
- Materialización del vuelo de un drone en escenarios con gravedad -9.8: laberinto para drone.
- Materialización de las colisiones que permite ajustar el coeficiente de restitución para hacer más realistas los choques entre objetos y el golpeo de un balón: fútbol competitivo.

6.2. Líneas futuras

La implementación de este nuevo motor de físicas complementario ha supuesto un gran avance en el funcionamiento de *WebSim*. No obstante, aún existen múltiples líneas abiertas que ayudarán a mejorar aún más el entorno *Kibotics*.

- Creación de ejercicios competitivos para cuatro jugadores. Por ejemplo, se propone como punto de partida la extensión del ejercicio de fútbol competitivo uno contra uno a un ejercicio competitivo de dos contra dos.
- Exploración del nuevo motor de físicas *ammo.js* y extender la implementación del motor complementario en conjunción con este otro motor. En el futuro, *A-Frame* pasará a utilizar el motor *ammo.js* como motor de físicas por defecto, dejando a un lado el de *CANNON* ya que se preveé que quede obsoleto en los próximos años.
- Adición de efectos de sonido a los ejercicios. Un aspecto que otorgaría aún más realismo al simulador sería la adición de audio en los mismos. Sería de especial interés en ejercicios como el del fútbol competitivo, en el que se podría añadir el efecto del sonido del chute al balón y el del público celebrando un gol.

Capítulo 7

Bibliografía

1. Página oficial de *CANNON*

<http://schteppe.github.io/cannon.js/>

2. Página oficial de *ammo.js*

<https://github.com/kripken/ammo.js/>

3. Página oficial de *COCOS*

<https://www.cocos.com/>

4. Página oficial de *Box2D*

<https://box2d.org/>

5. Página oficial de *Box2D*

<https://box2d.org/>

6. Página oficial de *Revista de robots*

<https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/>

7. Página oficial de *Kibotics*

<https://kibotics.org/>

8. Página oficial de *iRobot*

<https://edu.irobot.com/>

9. Página oficial de *OpenRoberta*

<https://lab.open-roberta.org/>

10. Página oficial de *LEGO education*

<https://education.lego.com/es-es>

11. Página oficial de *Lenobotics*

<https://lenobotics.com/>

12. Página oficial de documentación *HTTP*

<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

13. URL del blog del TFG

<https://roboticslaburjc.github.io/2019-tfg-natalia-monforte/>

14. URL del GitHub del TFG

<https://github.com/RoboticsLabURJC/2019-tfg-natalia-monforte>

15. Página oficial del repositorio GitHub de las físicas de *A-Frame*

<https://github.com/donmccurdy/aframe-physics-system>

CAPÍTULO 7. BIBLIOGRAFÍA

16. Página oficial de *A-Frame*

<https://aframe.io/>

17. Página oficial de *Blender*

<https://www.blender.org/>

18. Página oficial de *JSON*

<https://www.json.org/json-es.html>

19. Página oficial de *HTML*

[https://developer.mozilla.org/es/docs/Learn/Getting\\$\\$_started\\$_with\\$_the\\$_web/HTML\\$_basics](https://developer.mozilla.org/es/docs/Learn/Getting%20started%20with%20the%20web/HTML%20basics)

20. Página oficial de *JavaScript*

<https://developer.mozilla.org/es/docs/Web/JavaScript>

21. Página oficial de *RoboCup Junior*

<https://www.robocupjunior.org.au/>

22. Página oficial de *Scratch*

<https://scratch.mit.edu/>

23. Página oficial de *Python*

<https://es.python.org/>