Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Modelado en Pygame del péndulo invertido y pautas para un control mediante técnicas de Reinforcement Learning.

Autor: Sergio Gómez Urbano
Tutor: Alejandro del Real Torres

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Modelado en Pygame del péndulo invertido y pautas para un control mediante técnicas de Reinforcement Learning.

Autor:
Sergio Gómez Urbano

Tutor:
Alejandro del Real Torres
Profesor asociado

Dep. de Ingeniería de Sistemas y Aotumática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Modelado en Pygame del péndulo invertido y pautas para un control mediante técnicas de Reinforcement Learning.

Autor: Sergio Gómez Urbano

Tutor: Alejandro del Real Torres

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, me gustaría mostrar mi agradecimiento a Alejandro del Real Torres, por darme la posibilidad de pertenecer a este proyecto y por permitir que fuese llevado a cabo bajo su supervisión. Por otro lado, gran parte de la realización de este proyecto se debe también a Santiago Moreira, que desde el inicio invirtió parte de su tiempo disponible en que los alumnos partícipes de este proyecto tuviésemos las pautas para conseguir llegar a algo útil y para aprender a utilizar todas las herramientas necesarias en referencia a los temas tratados en este trabajo.

Asímismo, agradecer a mi familia por el apoyo que me han dado en todo momento para que no desistiera en este trabajo ni en ninguno de los objetivos marcados durante el desarrollo del grado, haciendo posible su consecución en todos los aspectos posibles. Por último, agradecer a Inma Gálvez su continuo cariño y amor incondicional en los buenos y malos momentos, así como su afán por ser de ayuda en cada paso que he dado en los últimos años.

Resumen

El Reinforcement Learning, es un campo del aprendizaje automático que cuenta con unas características distintivas, las cuales le dotan de utilidades específicas respecto al resto de áreas de este ámbito. A grandes rasgos, el Reinforcement Learning consiste en un agente software, cuyo acometido será maximizar una determinada recompensa o premio, para ello realizará unas determinadas acciones que en un correcto funcionamiento lograrán optimizar el resultado perseguido por el programador.

De lo citado anteriormente, se puede extraer que el Reinforcement Learning no es un método para encontrar una solución óptima, sino más bien un aprendizaje progresivo o aproximación. Dicho esto, es fácilmente deducible que el método en cuestión será una opción a tener en cuenta en entornos donde la perfección es difícil de obtener, incluso de definir. El resultado de lo expuesto, es la elección de este método para buscar una solución en el difícil acometido de controlar el péndulo invertido. Consideraremos éste como un sistema 2D, donde el agente ya descrito se encargará de dar unos impulsos al carro hacia delante o atrás, elecciones que será capaz de tomar tras su aprendizaje.

De la ambición por mostrar los resultados del trabajo realizado en tiempo real, surge la tarea de elegir un entorno de programación donde sea fácilmente implementable el método de aprendizaje por refuerzo, así como posible la representación de gráficos en directo para escenificar el péndulo invertido. El que se tuvo en cuenta desde el primer momento, y finalmente el elegido, fue Python. Su elección se entiende a raíz de que existe una librería muy conocida para este lenguaje, que permite la implementación y actualización en tiempo real de dinámicas y gráficos en consonancia con ellas. Esta librería es Pygame, originalmente concebida para realización de videojuegos, pero cuya versatilidad le permite ser utilizada en todo tipo de aplicaciones que conlleven actualización de gráficos basados en ecuaciones.

Así, se tienen las herramientas con las que se perseguirá el objetivo de mantener erguido el poste de péndulo invertido. Si bien este es el objetivo final de un proyecto muy ambicioso, en el presente trabajo de fin de grado se estrecha el cerco hasta llegar únicamente a la consecución de un modelo funcional del péndulo invertido, así como a dar las pautas de un futuro control.

Abstract

Reinforcement Learning is a region of machine learning which has many distinctive features. These endow it of specific utilities in regard to the rest of areas belonging to this ambit. Roughly, Reinforcement Learning consists of a software agent whose aim is to maximize a certain reward or prize, chasing this target, it will do certain actions that in a correct operation will optimize the result pursued by the programmer.

From the aforementioned, it is posible to deduce that Reinforcement Learning is not a method to find a unique and optimal solution, rather, it's a progressive learning or aproximation. Having said that, is easy to deduct that the method in question will be an option to consider in environments where the perfection is hard to obtain even to define. As a result of the above, Reinforcement Learning has been chosen for the difficult challenge of controlling the inverted pendulum. It will be considered as a 2D system, where the agent already described will be responsible of provide impulses forward or backward to the cart. Decisions that it will be able to take after it learning.

From the ambition to show the results of the performance of the agent in real time, it comes up the task of choosing a programming environment. This one must be able to easily represent live graphics to stage the inverted pendulum, it will also be necessary to consider the ease of implementation of the learning method. The one that was kept on mind from the first moment, and finally the one chosen was Python. That choice is understood because of the existence of a library well known in this language, Pygame. This library is able to implement and update dynamics and graphs in real time in line with equations inserted by the programmer.

Accordingly, these are the tools which will be used to pursuit the objective of keeping erect the pole of the inverted pendulum. While this is the final objective of a very ambicious project, in the current final degree project, the scope is limited to get a functional model of the cart-pole and set the guidelines for a future control.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

e.o.c.	En cualquier otro caso
e	número e
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
sgn	Función signo
rect	Función rectángulo
∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
:	Tal que
<	Menor o igual
>	Mayor o igual

1 OBJETIVOS

1.1 Contexto del proyecto y objetivos

El presente Trabajo de Fin de Grado conforma el cierre de expediente de unos estudios en Ingeniería Electrónica, Robótica y Mecatrónica. Es por ello la concordancia que se encuentra entre la materia del grado con la temática del documento actual. Este Trabajo de Fin de Grado consistirá en una parte dentro de un proyecto mayor, que a su vez es englobado dentro de un objetivo aún más exigente, familiarizar a los alumnos de esta rama con la inteligencia artificial.

De forma más concreta, este proyecto consistirá en realizar el modelo físico programado en un entorno interactivo (Pygame), que valdrá como soporte para un futuro control por Reinforcement Learning del mismo. Así, sabiendo la premisa a la que se desea llegar, se explicará detalladamente lo que se ha conseguido, además de dar las pautas e incluir ecuaciones orientativas para el futuro control perseguido. El hecho de que la librería Pygame tenga una intención inicial de ser útil para la creación de videojuegos, hace que esté inherentemente concebido de forma que la inclusión de ecuaciones dinámicas está totalmente implementada y es bien recibida. Aunque durante el transcurso del grado en Ingeniería Electrónica, Robótica y Mecatrónica no se está obligado en ninguno de sus cursos a la utilización del reconocido lenguaje Python, éste era uno de los candidatos a utilizar siempre que se realizaba un trabajo que ofreciese algún tipo de libertad. De ahí que no se parte de cero en cuanto a la utilización del lenguaje en sí, sin embargo, el tutor siempre ha dotado a los alumnos componentes de éste proyecto del material necesario para completar los conocimientos anteriores, así como para forjar nuevas técnicas que han sido de utilidad a la hora de irse familiarizando con el entorno de Python 3 y las distintas formas de llevar a cabo su programación.

Así, se fue formando a los distintos alumnos partícipes de este proyecto de una forma útil y concisa, siendo dotados de todo tipo de material referente tanto a Python, como al campo del Reinforcement Learning. Para este último ámbito, hemos dispuesto de una extensa bibliografía que ha debido ser condensada hasta el punto de recolectar los conocimientos necesarios. En el caso del presente proyecto, para introducir lo que sería un futuro control del modelo presentado mediante la ya citada técnica de inteligencia artificial. Con el objetivo firme de cumplir estas metas, se han dado las guías durante el tiempo que ha ocupado la realización del proyecto, de este modo, como ejercicios de aprendizaje, han sido propuestas varias actividades por parte del tutor encargado de guiar el Trabajo de Fin de Grado. Para comenzar, se dedicó un tiempo al aprendizaje teórico de Python, y más específicamente Pygame, así como teoría sobre el Aprendizaje por Refuerzo. El primer ejercicio donde se funden ambas herramientas, fue cuando se propuso resolver el famoso problema del "Multi-armed bandit" aplicando la teoría aprendida durante la fase anterior, aunque no fue tarea fácil debido a la falta de contacto con las utilidades que eran de provecho para este propósito durante nuestra carrera universitaria, se logró resolver, sirviendo esto de motivación y aprendizaje para continuar emprendiendo el camino hacia el objetivo real del proyecto. Con esto, ya existía la capacidad de comprender la idea que fundamenta el Reinforcement Learning, sin embargo, faltaba por involucrar otra parte muy importante para este proyecto, la representación gráfica dadas unas ecuaciones dinámicas. Con este propósito se propuso otra práctica, consistente en tomar contacto con Python inspeccionando y tratando de hacer visual un modelo dado de péndulo simple, un modelo más sencillo pero que igualmente iba a ser de utilidad para asentar conocimientos útiles para el objetivo final. Finalmente, se proporcionó un modelo de este péndulo simple perfeccionado donde se permitía la interacción del usuario con las ecuaciones físicas, mediante una fuerza aplicada sobre el mismo péndulo. Tras estudiar, analizar y comprender las interacciones posibles en esta biblioteca de Python, se puede concluir que se disponía de todas las herramientas necesarias para ejecutar la totalidad del proyecto perseguido, conseguir un modelo interactivo del péndulo invertido, y planear como serán implementadas las ecuaciones del Aprendizaje por refuerzo sobre él.

Para acabar con esta sección introductoria, se debe comentar que el péndulo invertido en sí consiste en un péndulo que dispone de su centro de masa por encima de su pivote. Esto quiere decir que es inestable y sin ayuda adicional se caerá. La única forma de que se suspenda de manera estable en la posición en que la masa queda encima del poste es mediante un método de control. Así, el péndulo invertido o cart-pole es un problema clásico en dinámica y se utiliza como prueba para multitud de estrategias de control, en este caso, el Reinforcement Learning. El equilibrado de este péndulo inestable se realizaría mediante el movimiento del punto de apoyo, mediante un carro conectado a unas guías, a un motor o similares.

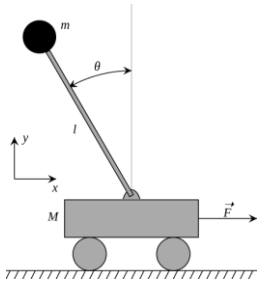

Figura 1-1, modelo de un péndulo invertido.

1.2 Herramientas utilizadas

1.2.1 Hardware

La elaboración del proyecto fue llevada a cabo en su totalidad en un ordenador portátil propio, ya que esta fase trata de únicamente de un trabajo de investigación y programación. Los softwares a utilizar, así como el material bibliográfico necesario ha sido proporcionado por el tutor, además de material libre encontrado en la red. Las características del PC utilizado son prácticamente irrelevantes dado el bajo requerimiento necesario para la ejecución, pero quedan descritos de la siguiente forma:

- Procesador: Intel Core i7-8750H
- Memoria RAM: 16GB
- Disco duro: 1TB HDD 256GB SSD

Como ya se ha dicho anteriormente y se puede comprobar, no han sido necesarias mayores prestaciones por parte del hardware utilizado.

1.2.2 Software

Para la realización de este TFG han sido de utilidad distintos tipos de software, tanto para su mera realización como para la conservación de los distintos archivos de código que han sido generados.

- '**Anaconda**': Este software de código abierto consiste en una distribución abierta de los lenguajes Python y R para computación científica, es decir, ciencia de datos, aplicaciones de aprendizaje automático, procesamiento de datos a gran escala, análisis predictivo etc. El objetivo específico de este software consiste en simplificar la administración de paquetes y su despliegue. En este caso concreto se ha usado también 'Anaconda Navigator', que es una interfaz gráfica que permite arrancar de forma intuitiva y fácil distintas aplicaciones de utilidad en la programación Python. A continuación, se especificarán las aplicaciones que han sido utilizadas para la consecución del trabajo, pero por defecto, las utilidades disponibles en 'Anaconda Navigator' son las siguientes:

Figura 1-2, Anaconda Navigator.

  - JupyterLab
  - Jupyter Notebook
  - QtConsole
  - Spyder
  - Glueviz
  - Orange
  - Rstudio
  - Visual Studio Code

- **'Jupyter Notebook'**: Jupyter notebook es un entorno interactivo donde se pueden desglosar las instrucciones de Python, haciendo uso de las características propias de un lenguaje interpretado. Esto le hace ser de gran utilidad para iniciarse en este tipo de programación, ya que da la posibilidad de comprobar la funcionalidad de cada instrucción aisladamente, y si es el caso, también de crear scripts de longitud indefinida.

Figura 1-3, Jupyter Notebook.

- **'Spyder'**: Spyder es un entorno de desarrollo de código abierto dedicado a la programación científica en el lenguaje Python. Este entorno da una gran cantidad de opciones a la hora de desarrollar y depurar el código Python, es por ello que ha sido el escogido para llevar a cabo la programación del modelo interactivo completo. De hecho, además de lo ya mencionado da la posibilidad por si solo de mostrarnos por pantalla el resultado en tiempo real del modelo programado, en este caso mediante la librería Pygame.

Figura 1-4, Software Spyder.

- **'Pygame'**: Pygame es un paquete de módulos del lenguaje Python. Aunque su premisa es la de crear videojuegos, este da cavidad a su uso para muchas otras finalidades. La facilidad para renderizar utilizando estos paquetes e implementar gráficas que obedecen a unas dinámicas dadas, lo hace una herramienta perfecta para confeccionar y comprobar la correcta puesta en marcha de un modelo como es el perseguido en este proyecto. Aunque no es el caso, su uso también ha proliferado en la creación de interfaces para usuario y programas multimedia.

Ilustración 2-5, Pygame.

2 INTRODUCCIÓN Y BASES TEÓRICAS

Al tratarse este trabajo de una fracción englobada dentro de un proyecto mayor, que compone la realización del modelo físico interactivo y las bases teóricas para continuar posteriormente cerrando el lazo de control, el grueso del mismo será lo expuesto en esta sección. En ella se relacionará el modelo realizado con las ecuaciones propias del método de control "Reinforcement Learning". Además, se profundizará acerca de las tres bases principales de este proyecto, Pygame, el modelo físico a implementar y el propio método de Aprendizaje por Refuerzo.

Inicialmente se realizará una aproximación a cada uno de los temas de forma genérica mientras que a posteriori se acabará concretando para el caso concreto. De este modo se expondrá como ha sido realizado el código, así como todas las herramientas referentes a programación utilizadas, continuando con las indicaciones pertinentes para implementar un futuro control. Todo lo expuesto aquí será contrastado posteriormente en una sección de demostración de resultados.

2.1 Sistema físico de péndulo invertido

2.1.1 Introducción y objetivos perseguidos

De la premisa de implementar controles mediante Reinforcement Learning a distintos sistemas físicos donde pueda ser útil y servir de aprendizaje, surge la idea de realizar este tipo de control en un péndulo invertido, sistema que ya se ha mencionado anteriormente en la introducción y en el que se profundizará y concretará a continuación.

De forma simplista, el funcionamiento del péndulo invertido se puede asimilar al caso en que se intenta sostener con la palma de la mano un cuerpo alargado de forma que se intente mantener erguido constantemente. Para ello se debe ajustar constantemente el movimiento de la mano para lograr mantener el palo sobre la vertical. En el caso del Cart-Pole o péndulo invertido sucede algo similar, ya que la estabilidad es muy débil y los movimientos del carro que sujeta el poste deben ser constantes y precisos para mantener esta característica. De este modo, el sistema escogido se perfila como uno de los bancos de prueba preferentes a la hora de realizar cualquier tipo de control, ya sea lineal o no lineal. Del mismo modo cabe destacar como característica de este modelo su versatilidad, es decir, el basto campo de aplicación del que dispone, que se extiende desde la robótica a la aeronaútica, donde se usa por ejemplo en algunos modelos de lanzaderas.

Aunque en ocasiones se restringe la libertad de rotación del poste por necesidades y/o comodidad para el control testeado en ese momento, en el caso del modelo presentado se ofrece la libertad original de que debe disponer este modelo, que es la de girar 360º sobre si mismo. Este movimiento rotatorio lo realizará sobre un carro, componente donde se realizará la actuación y que realiza una trayectoria lineal. Una característica trascendente y que vale la pena mencionar de este sistema es que se trata de un sistema subactuado, es decir, dispone de más grados de libertad que señales de control, información que se puede deducir de la descripción dada anteriormente. Este péndulo invertido es naturalmente inestable, si bien se puede mantener erguido en un entorno inalterado e ideal iniciando el ángulo del poste a 90º con la horizontal, la cuestión sustancial sería controlar este modelo en un $t>0$ y lograr mantenerlo erguido.

2.1.2 Procedimiento para obtener ecuaciones del modelo

Mediante un análisis dinámico, teniendo en cuenta las zonas donde el sistema tiene libertad para hacer un movimiento, y donde es posible aplicar una actuación, se pueden obtener las ecuaciones que posteriormente servirán para implementar este sistema en el código dedicado a la simulación del funcionamiento. Cabe destacar que, aunque a continuación se detallará la forma de obtener un modelo completo para el péndulo invertido, a la hora de amoldarlo al entorno Pygame se han llevado a cabo una serie de adaptaciones para hacer posible la

compatibilidad del sistema con el tiempo de muestreo y de renderización del entorno.

Inicialmente se definen los parámetros a utilizar y la notación que se usará para dar nombre a los diferentes componentes y las medidas útiles en el sistema.

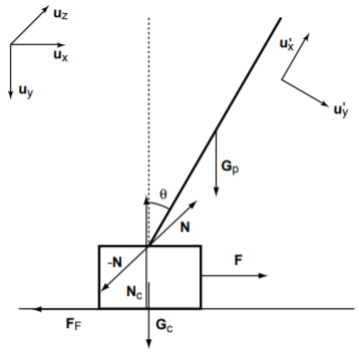

Figura 2-1, Dinámica del Cart-pole.

En esta imagen quedaría definido un análisis de las entidades dinámicas existentes en la posición dada de un sistema de péndulo invertido. Aparte de ellas se van a definir en la tabla a continuación los parámetros a tener en cuenta en lo que serán las ecuaciones definitivas:

Símbolo	Descripción
θ	Ángulo del poste con la vertical
$\dot{\theta}$	Velocidad angular del poste
$\ddot{\theta}$	Aceleración angular del poste
X	Posición del carro
$\dot{x}$	Velocidad lineal del carro
$\ddot{x}$	Aceleración lineal del carro
G	Fuerza de la gravedad
F	Fuerza aplicada en el carro
Mc	Masa del carro
Mp	Masa del poste
L	Longitud del poste (2l realmente)
μ_p	Fricción entre poste y articulación

μ_c	Fricción entre carro y guía

Tabla 2-1, variables del modelo físico

Aunque no es un asunto que concierna directamente a la realización de este trabajo, se va a comenzar por desmentir unas ecuaciones ampliamente extendidas, ya que se encuentran en gran cantidad de artículos y publicaciones y contienen algunos errores que pueden llevar a futuros errores en simulación. De hecho, eran las elegidas para ser implementadas en un primer momento, hasta que se hizo notable el fallo que contienen.

$$\theta^{\cdot\cdot} = \frac{gsin\theta + cos\theta\left[\frac{-F - Mpl\theta^{\cdot 2}sin\theta + \mu_c sgn(x^{\cdot})}{Mc + Mp}\right] - \frac{\mu_p \theta^{\cdot}}{Mpl}}{l\left[\frac{4}{3} - \frac{Mpcos^2\theta}{Mc + Mp}\right]}$$

$$x^{\cdot\cdot} = \frac{F + Mpl[\theta^{\cdot 2}sin\theta - \theta^{\cdot\cdot}\cos\theta] - \mu_c sgn(x^{\cdot})}{Mc + Mp}$$

El error en estas extendidas ecuaciones del péndulo invertido reside en que la fuerza de fricción entre el carro y las guías o la superficie donde se mueve es $\mu_c sgn(x^{\cdot})$. En un análisis dimensional, esta fricción es adimensional en lugar de tener las unidades de una fuerza. Otro error a considerar es el hecho de considerar la gravedad como una magnitud negativa. De hecho estas ecuaciones serían consistentes con un valor positivo de g, de otro modo, el término gsinθ podría producir una aceleración angular negativa para un ángulo pequeño.

Ahora sí, se procede a encontrar las ecuaciones correctas para un sistema de péndulo invertido. Para ello se ha de aplicar la segunda ley de Newton, para mayor claridad del uso de estas ecuaciones se puede recurrir al diagrama incluído en la página anterior.

$$F + F_f + G_c - N + N_c = M_c a_c$$

De acuerdo con la ley de acción-reacción, el poste actúa sobre el carro con una fuerza –N. En esta ecuación, F_f es la fricción entre el carro y la guía y a_c es la aceleración del carro. Se tiene por tanto lo siguiente (ver figura):

- F=Fu_x
- F_f=-$F_f u_x$
- $G_c = M_c g u_y$
- N=$N_x u_x - N_y u_y$
- $N_c = -N_c u_y$
- $a_c = x^{\cdot\cdot} u_x$

Siendo ux y uy los vectores unitarios de los ejes de referencia. Descomponiendo la ecuación anterior en los ejes x e y queda:

$$F - F_f - N_x = M_c x^{\cdot\cdot}$$

$$M_c g + Ny - Nc = 0$$

Para las fricciones, según el modelo de Coulomb y suponiendo que el movimiento del carro queda limitado a arriba o abajo, la fuerza de fricción es:

$$F_f = \mu_c |N_c| sgn(x^{\cdot}) = \mu_c N_c sgn(N_c x^{\cdot})$$

Aplicando la segunda ley de Newton al movimiento lineal del poste se tiene:

$$N + G_p = m_p a_p$$

Donde :

$$G_p = m_p g u_y$$

La aceleración a_p del centro de masas del poste es debida al efecto compuesto por la aceleración del carro y la velocidad angular del poste $w = \theta^{\cdot} u_z$ y velocidad angular $\varepsilon = \theta^{\cdot\cdot} u_z$:

$$a_p = a_c + \varepsilon \, x \, r_p + w \, x \, (w \, x \, r_p),$$

Donde r_p es el vector que representa la posición del centro de masas del poste con respecto a la articulación alrededor de la cual gira. Por tanto, quedaría:

$$a_p = x^{\cdot\cdot} u_x + l\theta^{\cdot\cdot} u_z \times (sin\theta u_x - cos\theta u_y) + l\theta^{\cdot 2} u_z \times [u_z \times (sin\theta u_x - cos\theta u_y)]$$

Tras simplificar esta ecuación con los productos vectoriales de los vectores unitarios, se debe introducir en la ecuación donde aplicamos la 2ª ley de Newton al movimiento lineal del carro y descomponiendo en las direcciones x e y, resulta:

$$N_x = m_p(x^{\cdot\cdot} + l\theta^{\cdot\cdot}\cos\theta - l\theta^{\cdot 2} sin\theta)$$
$$m_p g - N_y = m_p(l\theta^{\cdot\cdot}\sin\theta + l\theta^{\cdot 2} cos\theta)$$

Aplicando la 2ª ley de Newton al movimiento rotacional del poste alrededor de su articulación se tiene:

$$M = I\varepsilon + r_p \times a_c$$

Dónde $M = r_p \times G_p - \mu_p \theta^{\cdot} u_z$, es la suma de los esfuerzos de torsión no inerciales actuando sobre el poste relativos a la articulación, $I = 4/3 m_p l^2$ es el momento de inercia del poste relativo a la articulación, y $-r_p \times a_c$ puede ser interpretado como el esfuerzo de torsión generado por la fuerza inercial causada por la aceleración del carro. De este modo:

$$m_p g l sin\theta - \mu_p \theta^{\cdot} = \frac{4 m_p l^2 \theta^{\cdot\cdot}}{3} + m_p x^{\cdot\cdot} l\cos\theta$$

De las ecuaciones anteriores ya se podía aislar $x^{\cdot\cdot}$, quedando:

$$x^{\cdot\cdot} = \frac{F + m_p l(\theta^{\cdot 2} sin\theta - \theta^{\cdot\cdot}\cos\theta) - F_f}{m_c + m_p}$$

Por último, introduciendo e valor de $x^{\cdot\cdot}$ en la ecuación anterior se tendría :

$$\theta^{\cdot\cdot} = \frac{gsin\theta + cos\theta\left[\frac{-F - m_p l\theta^{\cdot 2} sin\theta + F_f}{m_c + m_p}\right] - \frac{\mu_p \theta^{\cdot}}{m_p l}}{l\left[\frac{4}{3} - \frac{m_p cos^2\theta}{m_c + m_p}\right]}$$

Llegados a esta ecuación se puede observar como ahora si hay exactitud dimensional en los términos componentes de ella. A continuación, y por último se expone como serían las ecuaciones en caso de supresión de la fricción:

$$\theta^{\cdot\cdot} = \frac{gsin\theta + cos\theta\left[\frac{-F - m_p l\theta^{\cdot 2} sin\theta}{m_c + m_p}\right]}{l\left[\frac{4}{3} - \frac{m_p cos^2\theta}{m_c + m_p}\right]}$$

$$x^{\cdot\cdot} = \frac{F + m_p l(\theta^{\cdot 2} sin\theta - \theta^{\cdot\cdot}\cos\theta)}{m_c + m_p}$$

2.1.3 Ecuaciones utilizadas en el modelo

En el caso concreto del modelo utilizado se han implementado las últimas ecuaciones citadas. Sin embargo, para que en la simulación se observase un movimiento del péndulo invertido coherente con la realidad, se han introducido algunos factores de corrección a lo largo de ambas ecuaciones, calculados a raíz del estudio del movimiento observado con las ecuaciones originales, y teniendo en cuenta el método de integración utilizado, así como el método de renderizado con el que Pygame da forma a este modelo. Estos factores de corrección, así como las ecuaciones completas utilizadas se mostrarán posteriormente en la puesta en escena del código completo de la simulación.

Si bien se han utilizado las ecuaciones donde se obvia la fricción se da la posibilidad de incluir un rozamiento teórico en las ecuaciones mediante el cambio de valor de una variable, que también se puede anular para distintos experimentos.

2.1.4 Método de integración para resolución de ecuaciones

El método utilizado para la integración de las ecuaciones expuestas en el apartado 2.1.2, ha sido el método de integración numérica de Euler, llamado así en honor a Leonhard Euler. Si bien no es un método altamente sofisticado, ha resultado de gran utilidad en la realización del proyecto y de fácil integración en el entorno de Pygame.

El método de Euler consiste en un método de primer orden, donde además se suponen verificadas las hipótesis del Teorema de Picard, así como que existe solución única para el problema. El teorema de Picard es el que establece que existe unicidad en el problema de valor inicial de Cauchy, es decir, que se tiene una única solución. La simplicidad del teorema de Euler trae como consecuencia un error local proporcional al cuadrado del tamaño del paso elegido, y un error global proporcional al paso elegido.

Si bien usualmente se utiliza para construir otros métodos más complejos, ha sido escogido como el método definitivo a utilizar a la hora de resolver las ecuaciones en el caso del presente proyecto.

La metodología que lleva a cabo este sistema de integración es encontrar la recta tangente a la curva, de la que se conoce el punto inicial. A raíz de ir encontrando rectas tangentes a puntos que se sitúan en cada paso de tiempo, se construye la función que constituye la integral de la original. Así, se obtendrá una curva poligonal que, en el caso de haber escogido un paso de tiempo pequeño, no diferirá en gran medida de la curva original.

Inicialmente, se divide el trazado completo en intervalos de ancho h:

$$h = \frac{x_f - x_o}{n}$$

Una vez tenemos estos intervalos, el valor inicial nos da la posición del primer punto $P_0 = (x_0, y_0)$, por donde pasará la curva ya mencionada. Ya teniendo el punto inicial se evaluaría la derivada de F(x):

$$F'(x) = \frac{dy}{dx}|_{P0} = f(x_0, y_0)$$

Conocemos pues, el punto inicial así como la pendiente de la recta a describir. Iterando con este proceso, se pueden obtener infinitos puntos de forma similar, y así es como se ha realizado la integración para la simulación.

2.2 Pygame

Pygame es un conjunto de módulos del lenguaje Python de código abierto y gratuito creado originalmente con la finalidad de la realización de videojuegos. Esta librería permite la creación de una pantalla donde una serie de objetos pueden seguir el movimiento de unas físicas programadas, además de permitir la implementación de mecánicas de colisión que ayuden en el proceso de crear estos gráficos, que no tienen que ser necesariamente con finalidad lúdica.

El hecho de que sea código abierto se refiere a que el código fuente está disponible para cualquier usuario que quiera acceder a él, ya sea para la mejor interpretación del objeto utilizado como para su modificación y consumo propio. Por tanto, los softwares y demás utilidades que reciben la denominación de código abierto se suelen convertir en colaboraciones públicas donde cada usuario aporta su grano de arena para mejorar, actualizar, y depurar éstas. Se remarca esta característica de Pygame porque es una parte importante de que esta librería haya sido la utilizada a la hora de implementar el modelo de péndulo invertido. De hecho, una parte de las dudas y dificultades que han surgido durante la realización del sistema han podido ser resueltas gracias a la vasta comunidad que existe detrás de este conjunto de módulos. Otra motivación encontrada a la hora de escoger esta librería para la implementación del modelo es la sencillez de Pygame a la hora de aplicar sus funcionalidades. De hecho, esta librería es ampliamente utilizada a modo de aprendizaje, adoptando así la filosofía general de Python, que presume de compaginar una potencia de programación considerable con sencillez. Además de lo ya mencionado, también se ha considerado por su portabilidad, ya que es soportado por Windows, Linux, Mac OS X, BeOS, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, y QNX, lo que asegura que un gran número de personas van a poder utilizar el programa que se realice.

A partir de aquí se profundizará en los aspectos de Pygame que han sido de utilidad en la elaboración de este proyecto, dando así una idea de la importancia que han tenido estos módulos.

2.2.1 Instalación e importación

Pygame no viene incluido al descargar Python, pero, así como el intérprete Python, Pygame es gratuito y se pueden encontrar con mucha facilidad los archivos de instalación de este en la web. Una vez abierto el archivo de instalación y seguidos los pasos indicados, se puede comprobar que se tiene escribiendo en consola:

```
>>> import pygame
```

Si no aparece nada tras pulsar ENTER, entonces se puede decir que Pygame ha sido instalado con éxito.

Tras esto, al abrir el programa en edición, se debe de importar el módulo Pygame, cosa que se hace escribiendo lo siguiente en el editor de texto:

```
1. import pygame, sys
2. from pygame.locals import *
```

La primera línea se encarga de importar los módulos "sys" y "pygame", ya que con comas se pueden implementar varios módulos a la vez en una línea. La segunda línea importa el módulo "pygame.locals", aquí se encuentran gran cantidad de constantes que se usarán con frecuencia al usar pygame, como pueden ser QUIT o K_ESCAPE.

Por último, en este apartado, se cita la necesidad de llamar a pygame.init ( ) en todo programa realizado con Pygame, ya que es necesario antes de llamar a otras funciones de esta librería. Así se habría concluido con los pasos necesarios de inicialización de Pygame.

2.2.2 Despliegue del monitor de Pygame

Una de las premisas principales de una herramienta concebida para la creación de videojuegos, es que disponga de una interfaz gráfica con el usuario, o pantalla donde se mostrará lo programado. Específicamente, en Pygame, se llega a esto mediante la función:

```
>> pygame.display.set_mode (wide, high)
```

Este monitor creado es un módulo dentro del propio módulo de Pygame, y tendrá "wide" píxeles de ancho por "high" de alto, el color de cada píxel puede ser configurado de forma independiente respecto a los demás. Los parámetros de entrada para la función mencionada anteriormente son una tupla de los píxeles de ancho y largo de que dispondrá el monitor, sin embargo, se pueden añadir otros datos de entrada auxiliares que escapan del alcance de este proyecto. Finalmente, la función devolverá un objeto del tipo pygame.Surface, para abreviar, un objeto de este tipo no es más que un nombre que se da a un tipo de dato sobre el que, en Python, se pueden realizar distintas acciones específicas.

Para la inclusión de colores en su interfaz, Pygame utiliza el modelo archiconocido de síntesis aditiva RGB. Esto quiere decir que, mediante una combinación de los colores rojo, verde y azul, se llega al resto de colores buscados. En cuanto a la resolución, tenemos 8 bytes por canal, es decir, 256 tonos distintos de cada color, lo que supone un total de más de 16 millones de colores posibles. De forma orientativa se deja una tabla donde se puede observar como definir algunos colores bastante conocidos:

Color	RGB Values
Black	(0, 0, 0)
Blue	(0, 0, 255)
Gray	(128, 128, 128)
Green	(0, 128, 0)
Lime	(0, 255, 0)
Purple	(128, 0, 128)
Red	(255, 0, 0)
Teal	(0, 128, 128)
White	(255, 255, 255)
Yellow	(255, 255, 0)

Tabla 2-2, colores con RGB 8 bytes

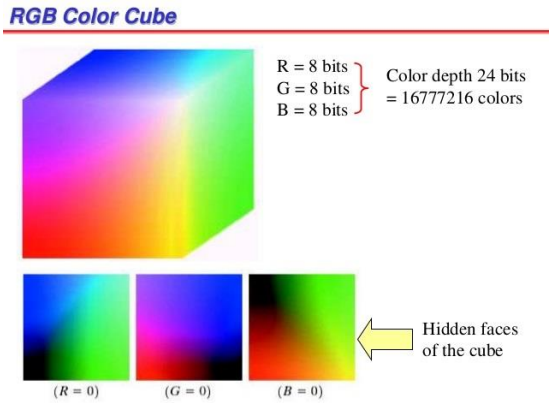

Figura 2-2, cubo RGB 8 bits

2.2.3 Renderizado y opciones de representación

Con el fin de crear formas geométricas más complejas, en Pygame, se pueden encontrar muchas formas de representar figuras típicas como rectángulos, líneas, círculos etc. De este modo, mediante la combinación de estas formas geométricas y la actualización de su posición según unas físicas dadas en el programa, se pueden crear videojuegos 2D, así como distintos modelos de sistemas físicos o utilidades de gran variedad de campos.

El tipo de dato pygame. Rect (), representa áreas rectangulares de un cierto tamaño y localización. Para crear un nuevo rectángulo, los parámetros que se dan son las coordenadas X, Y del vértice superior izquierdo, así como la anchura y altura del rectángulo en cuestión. El tipo de dato rect tiene gran cantidad de atributos que describen al rectángulo al que se refieren, algo positivo de crear este tipo de datos es que se reconfigurará al cambiar cualquiera de sus atributos para dejar todos los demás con sentido lógico y concordancia. La función pygame.get_rect () proporcionará la posición y tamaño de una superficie dada.

El método fill, se utiliza en Pygame para rellenar por completo una superficie de un color que se le pasa como parámetro, en el caso de tener por ejemplo una superficie llamada Péndulo y querer rellenarla totalmente de blanco, se tendría que escribir:

```
>>Pendulo.fill (255, 255, 255)
```

Este método se lleva a cabo normalmente sobre las superficies, ya que cambiar el objeto superficie en memoria conlleva mucho menos trabajo que modificar la imagen de la pantalla.

"Drawing Functions" en Pygame

Estas herramientas proporcionadas por Pygame, son las que darán realmente la posibilidad de crear las formas geométricas mencionadas anteriormente. Aunque se ha mencionado ya lo que es una superficie y como se puede tratar como un rectángulo, este es por así decirlo el fondo del trabajo realizado realmente, y las formas geométricas deseadas se dibujarán sobre la superficie con las Drawing Functions proporcionadas por Pygame.

- Función pygame.draw.polygon ( ): Esta función permite dibujar cualquier polígono, los parámetros que se le han de pasar son en orden los siguientes:
  - Superficie sobre la que se encuentra el polígono.
  - Color del polígono.
  - Ubicación de cada uno de los vértices, donde el último se unirá automáticamente al primero dado.
  - Opcionalmente, un parámetro que represente el grosor de las líneas, sino este polígono estará relleno.

Figura 2-3, ejemplos de polígonos con pygame.draw.polygon ( )

- Pygame.draw.line ( ): Como su nombre indica, esta función es la encargada de dibujar una línea de un punto a otro dados, los parámetros son los siguientes:
  - Superficie sobre la que se dibuja.
  - Color de la línea.
  - Coordenadas de un extremo de la línea.
  - Coordenadas del otro extremo de la línea.
  - Grosor.

- Pygame.draw.circle ( ): Se encarga de plasmar una circunferencia en pantalla, con los parámetros siguientes:
  - Superficie en la que se encuentra el círculo.
  - Color del círculo.
  - Coordenadas X, Y, del centro del círculo.
  - Radio.
  - Al igual que el polígono, aquí se da el grosor a no ser que queramos que esté relleno, en cuyo caso se pasaría un 0

- Pygame.display.update ( ): En pygame, nada se “dibuja” realmente hasta que es llamada esta función. Esto se debe a un motivo explicado también anteriormente, es más lento editar realmente en la pantalla que cambiar el contenido en memoria.

2.2.4 “Game Loop” o bucle de funcionamiento

~~**Bucle de funcionamiento y eventos**~~

El ampliamente conocido en Pygame como “Game Loop”, es el bucle dentro del cual el funcionamiento del sistema o juego implementado está constantemente siendo ejecutado, hasta cientos de veces por segundo. Este bucle de funcionamiento, está constantemente comprobando si existen nuevos eventos, actualizaciones en la representación de la superficie o del estado del sistema. Los eventos en sí son objetos de Pygame, y estos suceden cuando el usuario teclea, mueve el ratón o clicka, por ejemplo.

El bucle es un clásico “while True:”, lo que quiere decir que es un bucle cuya condición siempre estará activa y por tanto se repetirá infinitamente bajo cualquier circunstancia.

Para la detección de eventos, se utiliza la función pygame.event.get ( ), esta detectará todos los eventos que hayan ocurrido desde la última vez que se le llamó, que se almacenarán en una lista de eventos y podrán ser identificados por un atributo que se le asocia a cada uno de ellos llamado “type”. Si este atributo contiene la variable “QUIT”, significará que el usuario ha cerrado la ventana y quiere terminar el programa. Pygame genera el evento “QUIT” cuando el usuario pulsa el botón con forma de X para cerrar la ventana, o bien si por ejemplo se apagase el PC.

Como utilidad muy importante para el presente trabajo, Pygame crea la posibilidad de que un evento se cree mediante una pulsación de teclado. De este modo el usuario puede interactuar con el programa en ejecución durante el bucle de funcionamiento y conseguir una reacción o alteración de éste. La forma de identificar que un evento haya sido activado mediante teclado, es el event. type denominado KEYDOWN. Nuevamente, para saber si se tiene un evento de tipo KEYDOWN se utilizaría la función pygame.event.get ( ). Aunque en este caso no ha sido de utilidad, también existe un tipo de evento llamado KEYUP, útil para detectar cuando se deja de pulsar una tecla.

Si el tipo de evento encontrado resulta ser KEYDOWN, existe otro atributo donde se podrá detectar que tecla ha sido exactamente la pulsada, este recibe el nombre de event.key, y puede tomar los valores: K_LEFT, K_RIGHT, K_UP, K_DOWN, además de cualquier letra del teclado alfanumérico escribiendo ord(‘Tecla deseada’).

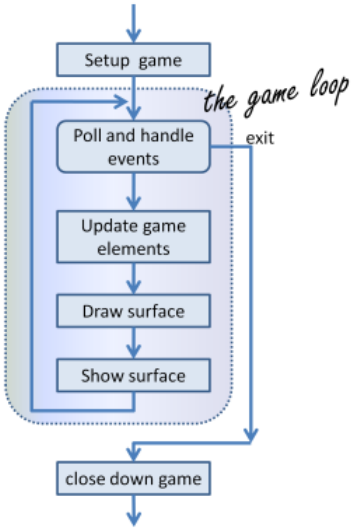

Figura 2-4, game loop

2.3 Reinforcement Learning

El Reinforcement Learning constituye un área del aprendizaje automático donde un agente software debe aprender la acción más productiva a base de los "conocimientos" obtenidos de experiencias anteriores. Consiste en saber qué hacer, unir situaciones con acciones de la forma más productiva. El ente que aprende no sabe qué acciones realizar, pero sí que podrá observar cual es la acción que obtiene una mayor recompensa a partir de probar éstas. Las dos características más importantes del Reinforcement Learning son la prueba y error y orientar los aprendizajes a situaciones posteriores.

Un agente de aprendizaje debe ser capaz de captar la situación de su entorno, así como de tomar decisiones que afecten a éste. De este modo, se conseguirá que este entorno llegue a la situación que se tiene como objetivo. El Reinforcement Learning es un método de aprendizaje donde se mezcla la exploración y la explotación, ya que debe saber si una acción ya conocida conseguirá una recompensa positiva, pero también, si no se conoce una acción adecuada, probar alguna otra que quizás podría ser de utilidad. Otra característica clave del Reinforcement Learning es que éste considera explícitamente el problema completo de un agente orientado a un objetivo, interactuando éste con un entorno incierto.

Una buena forma de comprender la mecánica de este campo de aprendizaje automático es considerar algunos de los ejemplos posibles, y aplicaciones que han guiado el desarrollo de éste.

- Un maestro de ajedrez realiza un movimiento. El sistema debe responder previendo posibles réplicas, e inmediatamente obtendrá juicios intuitivos de la deseabilidad de movimientos y posiciones particulares.
- Un robot móvil decide si debe adentrarse en otra habitación en busca de más basura que recoger, o si en cambio, debería volver a la estación de carga, lo hará en función de la carga de batería, además de lo fácil que ha sido en otras situaciones llegar a la estación.

Como se puede comprobar, las acciones del agente pueden afectar al estado del entorno (situación del tablero de ajedrez), afectando así a las opciones y oportunidades del agente en situaciones posteriores.

2.3.1 Introducción y objetivos

Como ya se ha dicho anteriormente, este proyecto se engloba dentro de otro mayor, donde el objetivo final es familiarizar a los alumnos con el Reinforcement Learning, más específicamente en aplicaciones de control, y concretando para el presente trabajo, para la estabilización del péndulo invertido. Sin embargo, al ser este un paso previo al objetivo final, se debe hablar del aprendizaje por refuerzo de forma general, además de dar las pautas para un futuro control.

Más allá del entorno y el agente software, se deben identificar cuatro subelementos dentro de un sistema de Reinforcement Learning. Estos son: política de actuación, recompensa, función de valor y opcionalmente un modelo del entorno.

La política de actuación define el comportamiento del agente en un momento dado. En otras palabras, y aunque lleve mucho trabajo y especificaciones de por medio, es la actuación que llevará a cabo el agente dadas unas circunstancias en el entorno. En algunos casos la política de actuación se corresponderá con una función o una tabla de búsqueda, mientras que en otros casos tendrá detrás grandes procesos de computación y una larga búsqueda de la acción correcta. En general, la política de actuación debe ser estocástica, especificando las probabilidades de cada acción.

Una recompensa define el objetivo de un problema de Reinforcement Learning. En cada paso de tiempo, el entorno manda al agente un número denominado recompensa. El propósito de este agente es maximizar las recompensas recibidas a largo plazo. La recompensa, por así decirlo, define como de buenos o malos son los eventos para el agente y el objetivo que se persigue. Las recompensas son, por tanto, el motivo principal que puede llevar a un cambio de la política de actuación.

Mientras que las recompensas indican como de bueno puede ser un evento en el momento exacto en que se produce, la función de valor se identifica con la bondad a largo plazo, es decir, la cantidad de recompensas que el agente puede esperar lograr en un futuro. Un estado del entorno, puede que siempre conlleve una recompensa inmediata de poco interés, sin embargo, su función de valor puede indicarnos que a largo plazo este estado conlleva a un número de recompensas elevado y por tanto beneficiar al agente, o bien el caso contrario. No se debe interpretar que la función de valor es un elemento de más valor que las recompensas, ya que, el único objetivo de una función de valor es obtener recompensas mayores, y sin estas ni siquiera podría existir ya que se basa en ellas. Desafortunadamente, es mucho más difícil estimar las funciones de valor que las recompensas, las recompensas son dadas directamente por el entorno, pero las funciones de valor pueden ser reestimadas de la secuencia de observaciones que lleve a cabo el agente durante su tiempo de funcionamiento.

Por último, un sistema de Reinforcement Learning puede tener un modelo, es decir, no basarse en su totalidad de la prueba y error, sino que se enfocan a optimizar un modelo previamente creado por el usuario mimetizando en la medida de lo posible el original.

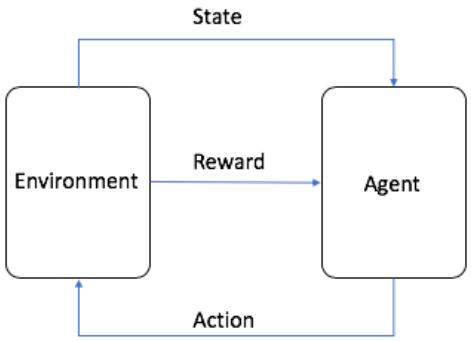

Figura 2-5, esquema de un sistema Reinforcement Learning

Una vez se conocen todos los elementos y la mecánica de funcionamiento, la mejor forma de familiarizarse con este método es ejemplificar y comenzar a proveerlo de ecuaciones que puedan maximizar recompensas de un entorno dado.

2.3.2 Explotación y exploración.

Este concepto consiste en un básico para comenzar a comprender la mecánica del Reinforcement Learning. De forma general, se supone que te enfrentas repetidamente a una elección entre K opciones (en el caso del proyecto solo serían la fuerza a la izquierda y a la derecha). Después de cada elección se recibe una recompensa numérica, elegida de una distribución aleatoria. El objetivo debe ser maximizar esta recompensa esperada total en un periodo de tiempo, por ejemplo 10 acciones o "time-steps". Explicado así, resulta muy análogo a como sería la implementación en el modelo de péndulo invertido, ya que la recompensa numérica obtenida sería simplemente en función de lo cerca que esté el péndulo de encontrarse totalmente vertical.

Se suponen K opciones posibles, que devuelven unas determinadas recompensas en cada paso de tiempo, después de repetir la acción de probar una de ellas varias veces, la acción más lógica sería intentar maximizar las ganancias concentrando las acciones en los hechos que mejor recompensa han ofrecido. En cualquier ejemplo propuesto, cada una de las k acciones tiene una recompensa esperada o media, a esto se le llama la función de valor de una acción. Se denota la acción tomada en el momento t como At, y la recompensa obtenida por ésta, Rt. La función de valor de esta acción arbitraria "a", se denota por q*a y es la recompensa esperada para a

$$q*(a) = E[R_t|At = a]$$

Si el value o función de valor de cada acción fuese conocido, sería trivial, ya que siempre se escogería la acción con más value. Se asume por tanto que no se conoce con certeza, pero sí una estimación, que es la que se ha obtenido a raíz de la experiencia Qt(a). Se le llama así porque es una estimación de la función de valor real q*(a).

Al tener estimaciones, siempre habrá una acción cuya estimación sea la más alta de las probables acciones, a esto se llama las acciones "greedy", lo que quiere decir avaricioso. En cambio, elegir una acción diferente a la greedy, se considera una exploración. Así tenemos que la "explotación" sería escoger siempre la acción greedy, este sería el método correcto para que en el siguiente paso se maximice la recompensa, pero no tiene por qué ser así en el siguiente paso, ya que, si se explora una acción y resulta tener mejor función de valor, dará una recompensa mayor a largo plazo. Aunque se esté utilizando un caso más ejemplificativo, se puede observar como estas K acciones se pueden identificar con las fuerzas a la izquierda y derecha en el péndulo invertido, y las recompensas como ya se dijo anteriormente, el ángulo en que nos encontremos (cercanía a estar totalmente erguido).

Incluso en el caso de que se conociese totalmente la función de valor de una acción greedy, las otras de las que no se conoce con certeza su value, podrían ser más fructíferas a largo plazo, luego si se dispone de una gran cantidad de pasos de tiempo, puede ser más productivo explorar que optar siempre por la opción más beneficiosa a corto plazo.

Funciones de acción-valor

Se empieza por estimar los "values" empíricos de las acciones, para poder usar estas estimaciones. Se recuerda que el verdadero valor de una acción es la recompensa media que se obtiene cuando esta acción se elige. Una forma natural de estimar es promediando las recompensas realmente recibidas:

$$Q_t(a) = \left(\frac{Suma\ de\ las\ recompensas\ cuando\ se\ ha\ escogido\ acción\ a}{veces\ que\ se\ escoge\ a}\right)$$

La forma más simple de elegir acción es la greedy, coger la que más recompensa instantánea de, y en caso de haber más de una, aleatoriamente;

$$A_t = argmax\ Q_t(a)$$

Una opción para explorar, es actuar de forma greedy la mayoría de casos, pero con una probabilidad ε, se elegirá una de las otras opciones, de forma equiprobable entre todas. Se llama a estos casos ε-greedy, una ventaja de estos métodos es que, al final, al subir el número de steps o pasos de tiempo, cada acción será muestreada infinitas veces, lo que asegurará que todas las Qt(a) converjan a q*(a).

Implementación incremental

Todos los métodos de elección de acción, estiman el valor de las acciones como medias de las recompensas obtenidas. Sin embargo, si se siguiese haciendo de esta manera, la memoria y capacidad computacional requerida iría aumentando conforme más recompensas se vayan obteniendo. Por un lado, se necesitaría capacidad en memoria para guardar la recompensa, y, además, se necesitaría más computación para sumarla al numerador. Esto se puede evitar si se usa una fórmula incremental para actualizar las medias, dadas Qn y Rnm se podría tener Qn+q de la siguiente forma:

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$$

Así, solo se requeriría memoria para Qn y Rn.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n]$$

Donde alpha es el parámetro de la magnitud del paso de tiempo, que debe ser entre 0 y 1 y constante. Con esto se tiene que la Qn+1 se puede calcular como una media ponderada de las recompensas pasadas y la estimación inicial Q1.

Valores iniciales optimistas

Este método consiste en establecer unas recompensas iniciales con un valor muy alto respecto a la recompensa que se puede obtener realmente (se es optimista), el premio que se obtendrá siempre será menor que la estimación de otra de las opciones que se tienen. Por tanto, esto motiva al agente software a explorar, incluso eligiendo siempre la opción más greedy o codiciosa.

Selección de acción por límite de confianza superior

La exploración siempre es necesaria porque no se tiene exactitud de las estimaciones. El método greedy elegirá la opción que parece mejor en el presente, pero se podría estar escapando una opción mejor, de ahí que se

explore. El método ε-greedy, hace que se explore cada cierto tiempo, pero lo hace de modo que todas las opciones tienen las mismas opciones de ser elegidos para la exploración. Sin embargo, sería más lógico que se filtrasen las acciones con más opciones de ser óptimas. Una forma efectiva de hacer esto sería lo siguiente:

$$At = argmax[Q_t(a) + c\sqrt{\frac{\ln t}{Nt(a)}}]$$

Nt(a) se corresponde con el número de veces que se ha seleccionado la opción A. c > 0, controla el grado de exploración. Si Nt(a)=0 se considera que a es una acción que maximiza las ganancias. La idea de este método para seleccionar la acción adecuada es que el término dentro de la raíz cuadrada es una medida de la incertidumbre o varianza en la estimación del valor de a.

Figura 2-6, Ejemplo de varias acciones con distintas recompensas.

2.3.3 Procesos de decisión finitos de Markov

A continuación, se introducirá el problema de las finitas decisiones de Markov o "finite MDPs". Este problema involucra evaluación de recompensas, como en el apartado anterior, pero también un aspecto asociativo, elegir distintas acciones en distintas situaciones. MDPs son una formalización clásica de la toma de decisiones secuencial. En este tipo de problemas, la repercusión de las acciones no cae solo sobre la siguiente acción, sino también sobre las siguientes. Mientras que, en el caso anterior, se estimaba la función de valor q*(a), en los MDPs se estima el value q*(s,a), es decir, la función de valor asociado a cada acción en cada estado. Esta dependencia con el estado es esencialmente para darle credibilidad a las acciones tomadas en el largo plazo.

Los MDPs son formas matemáticamente idealizadas del problema de Reinforcement Learning para los cuales se pueden hacer suposiciones teóricas de forma precisa.

Interfaz Agente-Entorno.

Los MDPs están llamados a ser el siguiente paso del problema de aprender a partir de la interacción para obtener una meta. El agente interactúa con un medio continuamente, uno eligiendo opciones y el otro cambiando debido a ésta. El agente y el medio interactúan en cada instante de tiempo t, donde el agente recibe alguna representación del estado del medio, y en función a esto elige una acción. En un MDP finito, el conjunto de estados, acciones y recompensas, tienen un número finito de elementos. En cada paso de tiempo el agente se encontrará en un estado y realiza una acción. En consecuencia, al paso de tiempo siguiente se obtendrá una recompensa numérica Rt+1, y se encontrará en un estado distinto St+1.

Normalmente se usa una función p de cuatro argumentos, que es capaz de dar el siguiente estado y la recompensa obtenidas a partir del estado y la acción del instante t. Aunque en ocasiones será de importancia conocer otras expresiones. No obstante, tanto los estados como las acciones pueden referirse a cualquier cosa, de alto o bajo nivel, se definen las acciones como decisiones que se quieren aprender y los estados cualquier cosa que se puede

conocer que puede ser útil. En el caso del Reinforcement Learning, el límite entre el medio y el agente no es como en un robot del mundo físico, por ejemplo, ya que en él sería fácil definir el medio como todo lo que exceda físicamente a este. Sin embargo, en el aprendizaje por refuerzo, los motores, sensores etc. Serían considerados parte del medio. Las recompensas a pesar de ser computadas dentro del cuerpo físico se consideran externas al agente.

La regla general que se toma es que cualquier cosa que no pueda ser cambiada arbitrariamente por el agente se considera fuera de él. Siempre se considerará que la forma en que las recompensas son computadas es algo externo al agente. De hecho, en muchas ocasiones el agente puede conocer todo acerca del medio y aun así esto será un problema difícil de Reinforcement Learning.

La aplicación de los MDPs al presente problema no es fácil, ya que habría que muestrear el ángulo que toma el péndulo hasta puntos en que la precisión sería inferior a la deseada, sin embargo, se podría hacer de esta forma teniendo claras cuales son las dos acciones que se tienen y como habría que utilizarlas en función del entorno (el ángulo). En Reinforcement Learning, el propósito del agente está formalizada en términos de una señal especial llamada recompensa. Pasando del medio al agente, en cada instante de tiempo la recompensa es un simple número real. Se puede pensar que la meta es la maximización del valor esperado de las recompensas acumuladas. Esta forma de hacerlo es de lo más distintivo de este método de Machine Learning. Aunque puede parecer limitante, en la práctica se ha demostrado que es flexible y ampliamente aplicable. Cabe remarcar que no se darán recompensas positivas por las "submetas" que se consigan, sino por la meta final, es decir, si un agente trata de jugar al ajedrez, no recibirá recompensas si consigue cosas como tomar el centro de la mesa, sino por acciones que lleven realmente a ganar partidas, o en el caso del péndulo invertido, que conduzcan a que el péndulo consiga erguirse por completo.

2.3.4 Q-Learning

Q-Learning es una forma de aprendizaje por refuerzo que se concibe sin modelo del sistema. También puede ser visto como un método de programación dinámica asíncrona. Proporciona al agente la capacidad de aprender a actuar de forma óptima en un MDP, aprendiendo a raíz de la experiencia. Un agente intenta una acción en un estado particular, y evalúa las consecuencias en términos de la recompensa inmediata recibida, así es como va estimando la función de valor de la acción que está escogiendo. Intentando todas las acciones en todos los estados repetidamente, aprende cual es la mejor de forma promedia, juzgando en función de las recompensas a largo plazo. Siendo Q-Learning un método relativamente primitivo (1989) puede actuar en sistemas bastante sofisticados.

Se considera un agente computacional en un entorno discreto, finito, eligiendo una de unas finitas acciones posibles en cada paso de tiempo. El entorno descrito constituye un proceso de Markov con el agente como controlador. En el paso n, el agente está preparado para registrar el estado "Xn" del entorno, y puede elegir la acción "An" en consonancia. Recibirá una recompensa probabilística Rn, cuyo valor medio depende únicamente del estado y la acción, y el estado del entorno cambiará a un nuevo estado "Yn" de acuerdo a la ley:

$$Prob[y_n = y|x_n, a_n] = P_{x_n y}[a_n]$$

La tarea a la que se enfrenta el agente es la de determinar una política de actuación óptima, que maximice la recompensa descontada total esperada. Con recompensa descontada, se quiere decir que las recompensas recibidas "s" pasos después, no valen tanto como la recibida en el momento actual, así se define el factor de descuento, $\gamma^s (0 < \gamma < 1)$. Por tanto, se tiene que si el factor de descuento vale 0, solo se considerarán las recompensas actuales, y conforme aumenta se da más valor a las futuras. Bajo una política de actuación π, el valor del estado x es:

$$V^{\pi}(x) = R_x(\pi(x)) + \gamma \sum_{y} P_{xy}[\pi(x)]V^{\pi}(y)$$

El agente espera recibir $R_x(\pi(x))$ inmediatamente por llevar a cabo la acción π recomendada, y entonces se mueve a un estado que es "valioso" $V^{\pi}(y)$ para ello, con probabilidad $P_{xy}[\pi(x)]$. La tarea a la que se enfrenta un agente Q es la de determinar un π^* sin conocimiento inicial de estos valores. Para una política π, se define el valor de Q como:

$$Q^{\pi}(x,a) = R_x(a) + \gamma \sum_{\gamma} P_{xy}[\pi(x)]\, V^{\pi}(y)$$

En otras palabras, el valor de Q es la recompensa descontada esperada por ejecutar una acción a en el estado x y siguiendo la política π después de eso. El objetivo en Q-Learning es estimar el valor de Q para una política óptima. Si a* es una acción donde se espera el máximo de recompensa, una política óptima será entonces $\pi^*(x)=a^*$. En Q-Learning la experiencia del agente consiste en una secuencia de distintos estados o episodios. En el episodio n, el agente:

- Observa el estado actual Xn.
- Elige la actuación y lleva a cabo la acción An.
- Observa el siguiente estado Yn.
- Recibe una recompensa inmediata Rn.
- Ajusta su Qn-1 usando un factor $\propto$n de acuerdo con:

$$Q_n(x,a) = \begin{cases} (1-\propto_n) Q_{n-1}(x,a) + \propto_n [r_n + \gamma V_{n-1}(y_n)] & si\ x = x_n\ y\ a = a_n \\ Q_{n-1}(x,a) & en\ cualquier\ otro\ caso \end{cases}$$

Donde:

$$V_{n-1}(y_n) \equiv \max\{Q_{n-1}(y,b)\}$$

Por supuesto, en los primeros estados de aprendizaje, los valores de Q podrían no ser precisos. Los valores iniciales de Q para todas las acciones y estados se asumen conocidos.

2.3.5 Suposiciones y fundamentos para un control del péndulo invertido con Q-Learning

Como ya se ha dicho anteriormente, el núcleo de Q-Learning reside en estimar el valor para cada pareja de posibles estados y acciones obteniendo recompensas. La figura mostrada a continuación es el agente que actualmente se encuentra en estado S0 y tiene dos opciones, acción 1 que llevará al estado S1 y la acción 2 que llevará al estado S2. Ir al estado S1 proporcionará una recompensa de +5 y el estado S2, -5. S1 es un buen estado mientras que S2 no lo será en este caso, si el agente lleva a cabo la acción A1, la Q de S0 será positiva, mientras que será negativa si se avanza hasta S2.

Con formato: Fuente:

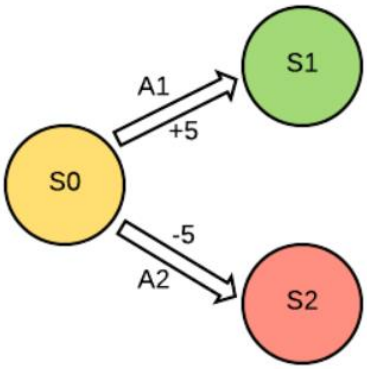

Figura 2-7, ejemplo de toma de decisiones en Q-Learning.

La actualización de Q se hará de acuerdo a la siguiente ecuación:

$$Q(s_t, a_t) = Q(s_t, a_t) + \propto \cdot \left(r_t + \gamma \cdot maxQ(s_{t+1}, a) - Q(s_t, a_t)\right)$$

La nueva Q para una tupla de acción y estado dependerá del antiguo valor de Q, así como de la recompensa inmediata que se recibirá y el valor máximo Q obtenible en el siguiente estado. Finalmente se tendrá una tabla de Q que mapea los estados y acciones a sus respectivas Q, entonces dado cualquier estado, la mejor acción puede ser elegida simplemente viendo la Q con mayor valor en la tabla.

Para implementar Q-Learning en el Cartpole, solo hay que tener cuidado al discretizar el espacio de estados, ya que el entorno en el que se encuentra es continuo y se necesita discretizar los estados para llevar a cabo la tabla de Q mencionada. Se debe por tanto discretizar cuatro variables:

- x (posición del carro): Tendrá un valor del inicio al fin de la pantalla de representación en Pygame.
- v (velocidad del carro): Este valor es el que puede tener una variación mayor, ya que las velocidades a las que puede llegar el carro son grandes, tanto hacia atrás como hacia delante.
- Theta (ángulo del poste): Se le debe dar un intervalo de ángulos alrededor de la posición erguida, a más holgado sea este intervalo, más difícil será la tarea de control.
- w (Velocidad angular del poste): Al igual que la velocidad del carro es un intervalo de gran variación debido a la dispersión que existe entre las posibles velocidades que este puede llevar.

En la ecuación también se puede observar que aparecen alpha y épsilon, los cuales son factores de aprendizaje y de exploración. Épsilon se corresponde con el factor de exploración mencionado anteriormente en este punto, lo que quiere decir que con una probabilidad épsilon se tomará una decisión aleatoria distinta a la que se sabe que proporciona una mayor recompensa en el siguiente paso.

2.3.6 "Policy Gradient".

Un "Policy Gradient" trata de entrenar al agente sin conocer necesariamente el valor de cada par estado-acción en un entorno, tomando pequeños pasos de tiempo y actualizando la política de actuación basada en la recompensa obtenida en este paso. El agente puede recibir recompensa inmediatamente por una acción o puede recibirla en un tiempo posterior, como al final del episodio. Se diseñará la política intentando que el agente aprenda usando la ecuación de π que se mostrará posteriormente, donde θ es el vector de parámetros, s es un estado particular, y a una acción:

$$\pi_\theta(s, a)$$

Se aplicará la técnica llamada "Monte-Carlo Policy Gradient", que quiere decir que se tendrá un agente ejecutándose a través de un episodio entero y después se actualizará la política basándose en las recompensas obtenidas. Se actualiza la política tomando una muestra de la función de valor de una determinada acción, lo que se ha llamado anteriormente Q.

$$Q^{\pi_\theta} = (s_t, a_t)$$

La función de valor de la acción está definida como la recompensa esperada tomando la acción a, en el estado s, siguiendo la política π. Se sabe que por cada paso que de la simulación se obtiene una recompensa de 1. Se puede usar esto para calcular el "policy gradient" en cada paso de tiempo, donde r es la recompensa para un par estado-acción particular. En lugar de usar la recompensa instantánea, r, se usará en su lugar una recompensa a largo plazo v_t, donde v_t es la suma descontada de todas las futuras recompensas para lo que dure el episodio. De este modo, a más duración de ejecución del episodio, mayor recompensa para un par estado-acción particular en el presente. v_t es, por tanto:

$$v_t = \sum_{k=0}^{N} \gamma^k r_{t+k}$$

Donde gamma es el factor de descuento. Después de cada episodio se aplica el "Monte-Carlo Policy Gradient" para mejorar la política de actuación de acuerdo a la ecuación:

$$\Delta\theta_t = \alpha \Delta_\theta log \pi_\theta(s_t, a_t) v_t$$

3 RESULTADOS

Como resultado definitivo del presente proyecto, se tiene un modelo funcional y realista del péndulo invertido, además de las pautas para establecer un futuro control con técnicas de Reinforcement Learning en base a este modelo. El sistema construido contará con las ecuaciones descritas en el apartado correspondiente de la sección anterior de esta memoria, y con el método de integración de Euler para conseguir su velocidad y posición tanto angular como lineal del carro.

Por otro lado, para servir de realimentación en un futuro control, cada vez que una tecla sea pulsada, izquierda o derecha, se imprimen por pantalla los valores cinemáticos del momento de pulsación, de este modo cuando se tenga un control se le podrán pasar los valores de esas variables e identificar en cierto modo la recompensa obtenida en cada paso de tiempo. En el modelo actual, las fuerzas de actuación sobre el carro son de un solo valor, es decir cada vez que se pulse la tecla existirá una fuerza actuante de un valor determinado en una dirección determinada, sin embargo, a la hora de implementar el futuro control se podría flexibilizar esto para dar algo más de versatilidad al agente software, aunque haría este control inherentemente más complicado. Siguiendo con el modelo creado, se puede observar que se ha implementado un algoritmo para incluir una fuerza de rozamiento de un valor constante en dirección contraria a la marcha del carro o en el sentido contrario de giro del péndulo, sin embargo es un valor teórico ya que hasta que no se cuantifiquen~~cen~~ los términos de fricción y se den valores realistas de las magnitudes cinemáticas y dinámicas del modelo este carecerá realmente de sentido a la hora de ser cuantitativo, más bien se incluye este rozamiento a efectos cualitativos y de observar como actuaría una fricción sobre el sistema.

La superficie creada para la representación del modelo consiste en una superficie de Pygame de dimensiones 800x800 píxeles, mientras que podría parecer que el modelo es demasiado grande para la superficie dada, es decir, que se da poco margen de maniobra, esto se hace a efectos de ser realista ya que en todo sistema de péndulo invertido real que se puede encontrar no existe un espacio muy grande donde se pueda maniobrar, dando más dificultad al sistema de control que se escoja. Las masas de carro y poste, así como la fuerza de gravedad son los necesarios para que el sistema disponga de lógica y tenga unas dimensiones parecidas a uno real. La elección del paso de tiempo escogido s~~í~~ que conlleva algo más de trabajo y ha sido algo más delicado debido a que es vital para la correcta simulación del modelo, primeramente, se escogió teniendo en cuenta el entorno de renderización que se tiene en Pygame, y luego, de forma empírica se fue ajustando a las necesidades del modelo para acercarse lo más posible a un comportamiento real.

Otro objetivo que se puede considerar cumplido es el de conseguir una representación que, aunque simplista, cumple con las necesidades relativas a mostrar fielmente el comportamiento de un péndulo invertido real. De este modo se pueden observar con certeza como afectan todo tipo de variaciones introducidas a la hora de la programación lo cual es el objetivo real cuando se crea un modelo de este tipo. Para la creación de esta representación se utilizaron las herramientas básicas de “draw” proporcionadas por Pygame, que aunque no son realmente sofisticadas pueden cumplir con su propósito a la hora de realizar un modelo de estas características, simplemente dando unos valores de offset a las distintas representaciones realizadas para que aparezcan en el centro de la pantalla y asociándolas a las variables pertinen~~e~~tes para que se muevan en consonancia a los elementos a que están relacionadas pueden cumplir con el propósito perseguido.

La parte del proyecto que da más pie a libertad de realización es la de la implementación de las físicas del péndulo y su actualización, que en el caso concreto de este sistema se ha realizado con la función yield que crea un generador y va actualizando en cada paso de tiempo todas las variables pertinentes y útiles para el funcionamiento del modelo.

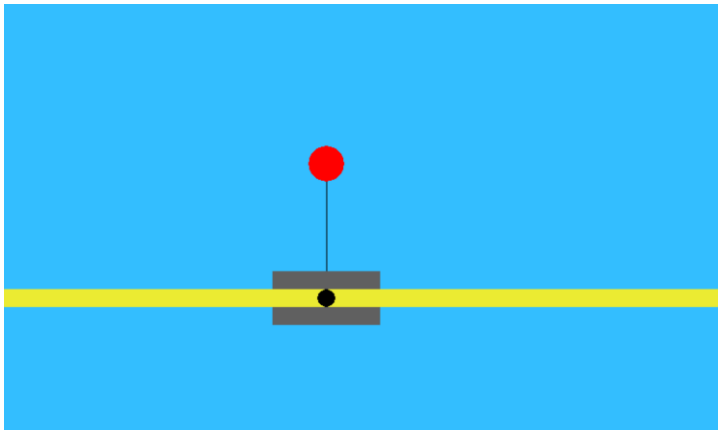

Figura 3-1, modelo en estado de reposo

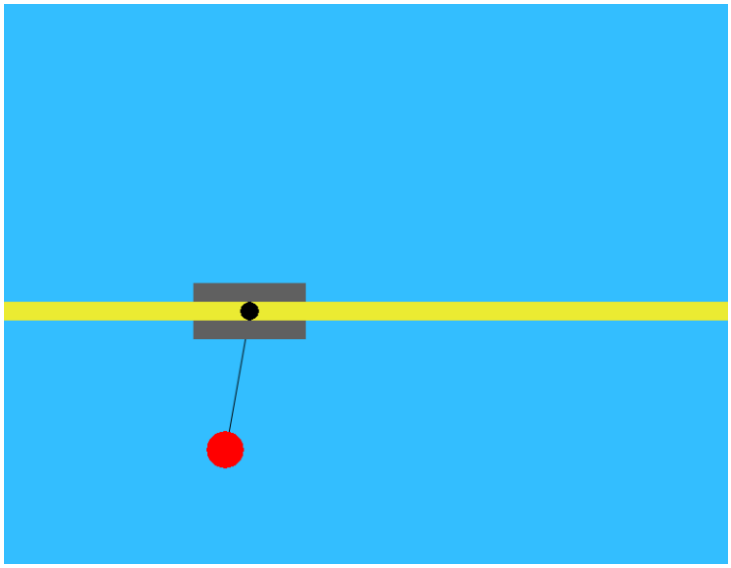

Figura 3-2, modelo tras algunas actuaciones y pasos de tiempo

Como última funcionalidad a considerar lograda, se tiene la interacción usuario-sistema mediante el teclado. Este se ha logrado mediante los eventos de Pygame, donde primero se debe de buscar la captación de algún

evento, y a raíz de ello y tras identificar que suceso ha causado éste, realizar las acciones pertinentes. En el caso concreto de este modelo las posibles teclas a pulsar son:

- Tecla izquierda: Imprime por pantalla los valores de los parámetros cinemáticos en el momento de la pulsación y aplica una fuerza sobre el carro hacia la izquierda.
- Tecla derecha: Igual que la tecla izquierda pero la fuerza aplicada sería hacia la derecha.
- ENTER: Resetea el modelo a una posición inicial que se indica en el fragmento de código asociado a este evento.
- ESC: Detiene la simulación y cierra la ventana de Pygame.

```
Right arrow pressed 36

Angulo con vertical: 194.74985315940927

 Posicion horizontal: 263.23153269370925

 Velocidad lineal: 0.633130767440863

 Velocidad angular: -6.080604613166819
```

Figura 3-3, Consola cuando tecla es pulsada

El valor que aparece a la derecha de la tecla elegida es el número de teclas que se han pulsado durante la simulación, y el resto los parámetros cinemáticos junto con su valor en el momento de la pulsación.

Además, como prueba de funcionamiento se adjuntarán junto con esta memoria algunos vídeos representativos del comportamiento del modelo cuando se interactúa con él con y sin rozamiento.

Vídeo 1: Comportamiento del modelo con rozamiento:

En este vídeo se puede observar cómo tanto el carro como el péndulo se frenan cuando se deja de actuar sobre ellos, acabando ambos estáticos tras algunos pasos de tiempo.

Vídeo 2: Comportamiento del modelo sin rozamiento:

En este caso, de un comportamiento sin rozamiento, el péndulo nunca pararía de oscilar alrededor del carro, y si se le da una fuerza a este derivaría hasta el infinito en mayor o menor velocidad en función de la cantidad de veces que se actúe sobre él (magnitud de la fuerza).

4 CONCLUSIONES

4.1 Alcance del proyecto

Cómo ya se ha mencionado anteriormente, el presente proyecto no es completo por sí mismo, sino que es parte de un objetivo mayor, en el que se pretende lograr controlar este modelo mediante unas técnicas específicas dentro del Reinforcement Learning. Así se podrían destacar como resultados específicos logrados dentro de este subproyecto los siguientes:

- Modelo fiel a ecuaciones físicas deducidas.
- Interacción con el usuario mediante el teclado.
- Introducción de Pygame como método para implementar sistemas dinámicos.
- Introducción y bases teóricas del modelo, Pygame y Reinforcement Learning.
- Mención de varias metodologías dentro del Reinforcement Learning que podrían formar parte de un futuro control.

De este modo se daría por concluido el trabajo actual, que como se deduce de lo anterior no consta únicamente del código implementado en el siguiente apartado, sino que también trata de sentar las bases teóricas de una serie de herramientas que serán de utilidad en un futuro proyecto completo.

4.2 Objetivos de control para un futuro

El proyecto consta actualmente de un control manual para controlar este modelo de péndulo invertido, que se acciona mediante una interacción con el teclado. El objetivo que se plantea para que se pueda considerar un trabajo completo, es el de que se pueda conmutar entre este control manual mediante teclado, y un control automático mediante alguna de las formas de Reinforcement Learning explicadas en el marco teórico.

Una vez se disponga de este control, las fuerzas que se dan al carro hacia izquierda o derecha serán similares, pero vendrán dictaminadas por un agente software. El agente, o los agentes obtenidos, serán evaluados y comparados con otros métodos de control. Esta evaluación de la calidad del control en el péndulo invertido se hace mediante la observación de la cantidad de episodios que se es capaz de mantener erguido el poste, y cuanto se tarda en llegar a esta estabilidad, por tanto, si es el caso, habría que unificar criterios en cuanto a los parámetros dinámicos y la longitud del paso de tiempo para que la comparación con otros métodos de control sea justa y equitativa.

5 CÓDIGO

En esta última sección, se incluye el código completo del programa, así como se explicarán las utilidades que faltan por concretar, ya que muchas fueron explicadas en el apartado dedicado a Pygame, donde se dan detalles de todas las funciones utilizadas referentes a esta librería. Sin embargo, se deben explicar también otras herramientas utilizadas, como pueden ser la programación orientada a objetos y otros útiles específicos de Python que merecen ser mencionados. El código se incluirá en distintas particiones, que serán minuciosamente explicadas en caso de que su funcionalidad no haya sido expuesta ya anteriormente en esta memoria.

```
3   import sys,os
4   from math import sin,cos,pi,atan2,sqrt
5   import math
6   import pygame
7   import numpy.random as rnd
8   from pygame.locals import *
9   import pygame.surfarray
10  itheta=pi/2
11  COLOR = {'black' : (0,0,0),
12           'red'   : (255,0,0),
13           'micolor' : (235,235,50),
14           'blue'  : (51,190,255),
15           'grey'  : (96,96,96),
16           'orange': (255,204,153),
17           'red'   : (255,0,0)}
18
19  SCREEN_WIDTH  = 800
20  SCREEN_HEIGHT = 800
21  SCREEN_DIM    = (SCREEN_WIDTH,SCREEN_HEIGHT)
22  SCREEN_CENTER = (SCREEN_WIDTH//2,SCREEN_HEIGHT//2)
23  DEFAULT_INTEGRATION_METHOD = 'euler_ac'
24  gravity=9.8      #Se toma la gravedad como una cte. de valor -9.8m/s^2
25  masscart=1.0     #Masa del carro
26  masspole=0.001;      #Masa del poste
27  length=0.5      #Mitad de la longitud del poste
28  total_mass=masspole+masscart   #Masa total
29  polemass_length=masspole*length    #Masa  del poste por la mitad de su longitud
30  tau=0.01          #Longitud temporal de cada estado
31  #PARAMETROS DE AJUSTE MODELO/ENTORNO PARA COMPORTAMIENTO REALISTA:
32  ajt=0.2;
33  ajx=40
34  ajf=7
35  ajftheta=0.7
36  frocstandardang=1.4;
37  frocstandardlin=0.2
```

Figura 5-1, Inclusión de librerías e iniciación de variables

En este primer fragmento de código, se empieza por incluir las librerías necesarias, mediante el comando "import". Estas serán tanto algunas muy conocidas que ayudan a implementar funciones matemáticas en Python, como numpy o math, además de Pygame, que es la librería muy comentada anteriormente y que hace posible la realización del modelo físico en tiempo real y la renderización de gráficos entre otras muchas cosas.

Tras esto, se procede a asignar valores a multitud de variables que serán de utilidad a posteriori en la realización del proyecto, como son las dimensiones físicas del modelo, los ajustes del modelo al entorno de pygame para un comportamiento realista, o las dimensiones de la pantalla entre otras cosas.

```
def gen_ipendulum_physics_euler_ac(x,x_dot,theta,theta_dot,force,ac):
    print ("value passed when key is pressed: ",ac)
    while True:
        frocang=0; #Variable para añadir una fuerza de rozamiento
        froclin=0
        costheta=math.cos(theta)     #Coseno de theta
        sintheta=math.sin(theta)
        if(theta_dot>0.08):
            frocang=-frocstandardang;
        elif(theta_dot<-0.08):
            frocang=frocstandardang;
        else:
            frocang=0;
        if(x_dot>0.001):
            froclin=-frocstandardlin;
        elif(x_dot<-0.001):
            froclin=frocstandardlin;
        temp1 = ( force+polemass_length * theta_dot * theta_dot * sintheta) / total_mass
        temp2= ((ajftheta*force)+polemass_length * theta_dot * theta_dot * sintheta) / total_mass
        temp1=temp1*ajf
        temp2=temp2*ajf
        thetaacc = ((gravity * sintheta - costheta* temp2) / (length * (4.0/3.0 - masspole * costheta * costheta / total_mass)))+frocang
        xacc  = (temp1 - polemass_length * thetaacc * costheta / total_mass) +froclin   #Aceleración
        x_dot=x_dot+tau*xacc     #Velocidad
        x=x+tau*x_dot*ajx        #Posición
        theta_dot=(theta_dot+tau*thetaacc)   #Velocidad angular
        theta=theta+tau*theta_dot*ajt
        force=0
        yield(x,x_dot,theta,theta_dot )   #Lo que devuelve función cada paso
PHYSICS_GENERATORS = {
    'euler_ac':   gen_ipendulum_physics_euler_ac,
}
```

Figura 5-2, Ecuaciones físicas

A continuación, se procede a incluir la ecuación de movimiento del péndulo invertido. Esto se hace mediante una función def que recibe como parámetros de entrada la posición del carro y su velocidad, así como la posición y velocidad angular del péndulo, además de la fuerza y la tecla pulsada por el usuario. Esta función se encuentra envuelta en su totalidad por un bucle while True, es decir, se estará ejecutando constantemente. Como se puede observar la ecuación se corresponde con las expuestas en el apartado 2 de este proyecto, donde se deducían para un modelo de péndulo invertido, sin embargo, se encuentran fraccionadas por temas de comodidad en la escritura de código siendo temp1 y temp2 partes de las ecuaciones mencionadas. El método de Euler también expuesto anteriormente se estaría aplicando a la hora de obtener las velocidades y posiciones de carro y péndulo, donde a la aceleración se le multiplica por un paso de tiempo tomado anteriormente y se le suma el valor de velocidad anterior.

En ~~consecuencia~~consecuencia, con el modelo original, se ha incluido un método de introducir una fuerza de rozamiento tanto para el péndulo como para el carro, donde si bien aún no se ha dimensionalizado, se puede ver con sentido físico una fuerza de rozamiento constante si se consideran unos coeficientes de fricción inalterables a lo largo del experimento.

Por último, la última línea dentro de la función de las ecuaciones físicas convierte a esta función en un generador, ya que yield, a diferencia de return conserva la iteración del bucle para la próxima vez que sea invocado, haciéndolo inmensamente útil para una aplicación como esta.

```python
class Pendulum(pygame.sprite.Sprite):
    """renders a fixed pivot pendulum and updates motion according to differential equation"""
    def __init__(self,pivot_vect,bob_radius,bob_mass,init_angle, integration_method = DEFAULT_INTEGRATION_METHOD):
        pygame.sprite.Sprite.__init__(self) #llamada a inicializar el Sprite
        phys_gen_init = PHYSICS_GENERATORS[integration_method]
        self.phys_gen = phys_gen_init(theta = init_angle,theta_dot = 0,x=0,x_dot =0,force=0, ac=0)
        self.bob_radius = bob_radius
        self.angle      = init_angle
        self.force=0
        self.vang=0
        self.vlin=0
        self.posx=0
        self.cont=0
        #Longitud completa del péndulo
        self.image  = pygame.Surface((800,800)).convert()
        self.rect   = self.image.get_rect()
        #Esto elige donde está el extrmo superior izquierdo de un rectángulo
        #pivot_vect está asociado al centro de la pantalla, le resta el balanceo para que el péndulo quepa bien
        self.bob_rect = None
        self.bobix=0
        self.bobiy=0
        self._render() #Renderizado del péndulo
```

Figura 5-3, Clase Pendulum

Una vez se llega a esta partición de código se comienza a utilizar otra herramienta que ha sido de gran utilidad a la hora de la realización del modelo, que es la programación orientada a objetos. Primeramente, se define una clase llamada Pendulum. Los parámetros que incluye aparecen a continuación del constructor, que es "__init__", que simplemente es el encargado de inicializar los atributos del objeto creado. Entre estos atributos se encuentran la ecuación física ~~utilizadacon~~utilizada con sus valores iniciales, todos los parámetros cinemáticos del modelo, así como se le da valor a distintas variables que serán útiles en la graficación posterior del modelo.

```python
    def _render(self):
        self.image.fill(COLOR['blue'])  #Rellena el fondo de color azul
        #Dibujando carro
        pygame.draw.aaline(self.image,COLOR['black'],(self.posx+60+300,330),(self.bobix+60+300,self.bobiy+330),True)
        pygame.draw.rect(self.image,COLOR['grey'],(self.posx+300,300,120,60),0)
        pygame.draw.rect(self.image,COLOR['micolor'],(0,320,800,20),0)                          #Decorar
        pygame.draw.circle(self.image,COLOR['black'],(round(self.posx+360),330),10,0)           #Decorar
        pygame.draw.circle(self.image,COLOR['red'],(round(self.bobix+300+60),self.bobiy+330),20,0) #Bola
        #La línea anterior dibuja una línea negra del centro de la pantalla a la posición de la bola
    def update(self):

        self.angle = self.phys_gen.__next__()[2]
        self.vang=self.phys_gen.__next__()[3]
        self.posx=self.phys_gen.__next__()[0]
        self.vlin=self.phys_gen.__next__()[1]
        self.cont=self.cont+1

        elangle  = self.angle
        iposx=self.posx
        velo=self.vlin
        vela=self.vang
        posicionx=iposx+round((length)*300*sin(elangle))
        posiciony=-round((length*300)*cos(elangle))
        self.bobix=posicionx
        self.bobiy=posiciony
        self._render()
    def reset (self,key_pos):
        self.angle =itheta
        self.vang=0
        self.posx=0
        self.vlin=0
        self.force=0
        self.phys_gen = gen_ipendulum_physics_euler_ac(self.posx,self.vlin,self.angle,self.vang,self.force,key_pos)
        key_pos = 0
```

Figura 5-4, Funciones render, update, reset

En esta otra parte de código, se llevan a cabo tres funciones parte de la clase creada anteriormente Pendulum.

En primer lugar, se crea la función render, en ella se graficarán como ya se explicó anteriormente en esta memoria los componentes del modelo. Primero se rellena el fondo de pantalla de color azul, y a posteriori se crea una línea que une la posición en que se encuentra el carro (posx) con la posición del péndulo en sí (bobix,bobiy), el resto de magnitudes que se encuentran siendo sumadas a estas coordenadas son parte del offset que se da en pantalla para que el sistema aparezca en el centro de ésta al inicio de la simulación. Finalmente, se dibujan tanto el carro como el péndulo en las posiciones de inicio de ambos.

La función update es la encargada de actualizar los valores cinemáticos del modelo, en cada paso de tiempo ejecutará las ecuaciones y obtendrá un nuevo valor de posición aceleración y velocidad. Además de esto se hacen ciertos ajustes para que al monitorizar estos valores por pantalla aparezcan correctamente.

Finalmente, la función reset se encarga de que cuando se pulse la tecla conveniente el péndulo y el carro vuelvan a la posición inicial.

```
127     def hit_right (self,key_pos):
128         self.angle = self.angle
129         self.vlin=self.vlin
130         self.vang=self.vang
131         self.posx=self.posx
132         self.force=25
133         self.phys_gen = gen_ipendulum_physics_euler_ac(self.posx,self.vlin,self.angle,self.vang,self.force,key_pos)
134     def hit_left (self,key_pos):
135         self.angle = self.angle
136         self.vlin=self.vlin
137         self.vang=self.vang
138         self.posx=self.posx
139         self.force=-25
140         self.phys_gen = gen_ipendulum_physics_euler_ac(self.posx,self.vlin,self.angle,self.vang,self.force,key_pos)
```

Figura 5-5, Pulsación de teclas

Una vez se llega a este punto es tiempo de implementar las funciones que actuarán en caso de tocar unas las teclas izquierda o derecha. Mientras que las variables cinemáticas se seguirán actualizando de igual manera, en el caso de ser pulsadas se incluirá una fuerza a la ecuación que desequilibra ésta en favor de uno de los lados.

```
141 def main():
142     """this function is called when the program starts.
143     it initializes everything it needs, then runs in
144     a loop until the function returns.
145     """
146     #Se inicializa todo
147     pygame.init()
148     screen = pygame.display.set_mode(SCREEN_DIM)
149     pygame.display.set_caption('Pendulum Simulation')
150     background = pygame.Surface(screen.get_size())
151     background = background.convert()
152     background.fill(COLOR['blue'])
153     clock = pygame.time.Clock()
154     p1 = Pendulum(pivot_vect=SCREEN_CENTER,bob_radius=30,bob_mass=1,init_angle=0)
155     free_group = pygame.sprite.RenderPlain((p1,))
156     held_group = pygame.sprite.RenderPlain()
157     screen.blit(background, (0, 0))
158     pygame.display.flip()
159     NUM_HITS  = 0
```

Figura 5-6, Principio de programa principal

Main es el programa principal desde el cual se les dará uso a todas las funciones y demás utilidades mencionadas anteriormente. Inicialmente, se deben llevar a cabo todas las formalidades para crear un entorno válido de Pygame, inicializar el propio Pygame, crear una pantalla de las dimensiones especificadas anteriormente y dar una ~~fomra~~forma inicial a ésta (en el presente caso, rellenarla totalmente del color azul del fondo). A posteriori se inicializa el reloj de Pygame y se crea un objeto de la clase Pendulum explicada anteriormente que recibirá el nombre p1 y tendrá como parámetros los que se pueden observar en imagen. También se puede ver la variable NUM_HITS que especifica la cantidad de teclas pulsadas desde inicio de la simulación, será útil para tener en un futuro un "feedback" de un control por Reinforcement Learning.

```
160     while True:
161         clock.tick(60)
162         anglecheck=-p1.angle*180/pi
163         pasito=anglecheck//360
164         repa=anglecheck
165         if(anglecheck<0):
166             repa=360+anglecheck
167         #Eventos activados por teclado y exposición de variables en el momento
168         for event in pygame.event.get():
169             if event.type == QUIT:
170                 return
171             elif event.type == KEYDOWN and event.key == K_ESCAPE:
172                 return
173             elif event.type == KEYDOWN and event.key == K_RIGHT:
174                 NUM_HITS = NUM_HITS + 1
175                 print ("\nRight arrow pressed",NUM_HITS)
176                 print("\nAngulo con vertical:",)
177                 print("\n Posicion horizontal:",p1.posx)
178                 print("\n Velocidad lineal:",p1.vlin)
179                 print("\n Velocidad angular:",p1.vang)
180                 key_pos = +1.0
181                 for b in free_group:
182                     b.hit_right(key_pos)
183             elif event.type == KEYDOWN and event.key == K_LEFT:
184                 NUM_HITS = NUM_HITS + 1
185                 print ("\nLeft arrow pressed!", NUM_HITS)
186                 print ("\nRight arrow pressed",NUM_HITS)
187                 print("\nAngulo con vertical:",repa)
188                 print("\n Posicion horizontal:",p1.posx)
189                 print("\n Velocidad lineal:",p1.vlin)
190                 print("\n Velocidad angular:",p1.vang)
191                 key_pos = -1.0
192                 for b in free_group:
193                     b.hit_left(key_pos)
```

Figura 5-7, Inicio bucle principal y captación de eventos.

En esta otra fracción de código, se ve como comienza el bucle del programa principal, que realiza algunos ajustes adicionales en el ángulo que se mostrará por pantalla para que siempre esté entre 0 y 360º. Además, se comienzan a captar los eventos, en este caso pulsación de teclas izquierda y derecha, donde se puede ver también que al pulsar cada una de ellas se mostrarán todas las variables cinemáticas por pantalla. No se hará más hincapié en el tema de captación de eventos porque fue ampliamente explicado en el apartado de la memoria donde se habla de las utilidades de Pygame.

```
194                 elif event.type == KEYDOWN and event.key == K_RETURN:
195                     NUM_HITS = 0
196                     print ("RESET!", NUM_HITS)
197                     print(repa)
198                     key_pos = 0.0
199                     for b in free_group:
200                         b.reset(key_pos)
201             free_group.update()
202 
203             screen.blit(background,(0,0))
204             free_group.draw(screen)
205             held_group.draw(screen)
206             pygame.display.flip()
207 
208     if __name__ == "__main__":
209         main()
210 
```

Figura 5-8, Tecla Return e inicio de Main

Finalmente, se especifica una tecla para realizar un reset, como se esperaba tras ver la función reset incluida entre las inherentes de la clase Pendulum. Además de encontrarse aquí las funciones para dibujar pantalla, actualizar la captación de eventos e inicio del propio programa principal.

REFERENCIAS

[1] García, José*†, Ramírez, Luis, Siordia, Xóchitl, Martínez, Trinidad. (2016). *Las leyes de Newton en el modelado y control del péndulo invertido sobre un carro.*

[2] Razvan V. Florian. (2007). *Correct equations for the dynamics of the cart-pole system.*

[3] J. M. Sanz-Serna. (20087).*El método de Euler de integración numérica*.

[4] Al Sweigart. (2015). *Invent Your Own Computer Games with Python*. 3rd Edition.

[5] Christopher J.C.H. Watkins. (1992).*Technical Note Q-Learning.*

[6] A.I. Wiki. A Beginner's Guide to Deep Reinforcement Learning. Available at: Available at: https://skymind.ai/wiki/deep-reinforcement-learning.

[7] Richard S. Sutton y Andrew G. Barto. (2018). *Reinforcement Learning. An Introduction*. 2ª Edición.