Navegación autónoma en simulación de multitudes.

Modelación y simulación computacional basada en agentes: Proyecto Final

Sergio Miguel Fernández Martínez

Junio 2020

Índice general

Capítulo 1

Introducción

En ambientes de multitudes robots autónomos pueden moverse de forma predecible y de manera defensiva para evitar colisiones con humanos, [1]. En trabajos anteriores se planeaban rutas para los robots y se comportaban de acuerdo al modelo de fuerzas sociales. estos enfoques han representado una desventaja debido a la complejidad computacional y el pobre desempeño ante situaciones nuevas.

Para obtener mejores resultados este trabajo propone la aplicación de algoritmos de deep reinforcement learning para la tarea de navegación autónoma.

1.1. Objetivo

El finalidad del trabajo se divide en dos tareas. La primera tarea será diseñar un ambiente de multitudes simulado basado en el modelo de fuerzas sociales. El segundo objetivo será lograr entrenar un robot para que navegue y llegue a una posición objetivo en en el ambiente diseñado, dicha tarea deberá ser aprendida usando el algoritmo conocido como deep deterministic policy gradient.

Capítulo 2

Modelo de Fuerzas Sociales

La predicción del comportamiento de multitudes es esencial para garantizar la seguridad en casos de emergencia, [1]. Estas predicciones para ser lo suficientemente descriptivas deben de tomar en cuenta la movilidad, la densidad de peatones y la dinámica de la multitudes. El comportamiento de multitudes pueden ser simulado a partir de varios métodos numéricos como el modelo de fuerzas sociales. Esta técnica fue propuesta por Helbing y Mòlnar en 1995, [1]. El modelo de fuerzas sociales es un modelo físico para el comportamiento psicológico que depende fuertemente de varios parámetros, [1]. Dichos parámetros se establecen de acuerdo a estudios empíricos. En este capítulo describiremos brevemente al modelo de fuerzas sociales y sus elementos.

2.1. Modelación de multitudes

Una gran variedad de efectos colectivos y fenómenos de autoorganización son observados en estudios empíricos de la dinámica de peatones. Estos efectos surgen de la toma de decisiones de cada individuo, son influidos por las interacciones con otros individuos y los deseos personales, [1]. Este proceso de decisión consiste de tres niveles, [1]. En el primero, se establece una estrategia y se eligen movimientos de acuerdo a un objetivo principal. El segundo nivel es táctico, describe las decisiones tomadas a corto plazo, que están basadas en las interacciones con el entorno. El tercer nivel define el actual comportamiento al caminar del peatón, tomando decisiones inmediatas como evitar colisionar.

2.1.1. Enfoques de simulación

Antes de describir por completo el modelo de fuerzas sociales será necesario hablar de las distintas categorías de los modelos que estudian el comportamiento de multitudes. Los métodos de estudio de multitudes son clasificados de acuerdo a sus características y su propósito. Principalmente, encontramos tres tipos: microscópico, mesoscópico y macroscópico, [1].

- En los modelos **microscópicos** cada peatón se representa como individuo y no como conjunto, cada uno tiene comportamiento y velocidad deseada propia [1]. Se muestran las interacciones entre todos los individuos.
- En los modelos **mesoscópicos** se usa una distribución de probabilidad sobre los estados microscópicos de los peatones para describir su dinámica, [1].
- En los modelos **microscópicos** los individuos no pueden ser distinguidos. Estos modelos muestran cantidades como densidades o el momentum promedio, [1].

Adicionalmente, se puede categorizar a los modelos de acuerdo a la distinción de espacio y tiempo, es decir, si es discreto o continuo y si son visibles los individuos o grupos, en otras palabras, homogéneos o heterogéneos.

Elegir uno de los enfoques anteriores depende del motivo de estudio. A continuación mostraremos algunos ejemplos importantes con las características anteriormente mencionadas:

Enfoque	**Escala**	**Espacio y tiempo**	**Individual o grupos**
Autómata celular	Microscópico	Discreto	Ambos
Celosía de gas	Microscópico	Discreto	Homogéneo
Fuerza social	Microscópico	Ambos	Homogéneo
Dinámica de fluidos	Macroscópico	Continuo	Homogéneo
Basado en agentes	Microscópico	Ambos	Heterogéneo
Teoría de juegos	Microscópico	Discreto	Homogéneo

2.1.2. Modelo

Este modelo se caracteriza por determinar el movimiento del peatón por cuatro efectos o fuerzas. La influencia y la interpretación física de cada una de las fuerzas puede ser tratada por separado. Los efectos son tratados de forma similar que a las fuerzas que aparecen en la segunda ley de Newton para la ecuación de movimiento. Estos efectos del modelo de fuerzas sociales influyen en las decisiones del agente, dando como resultado el movimiento del agente. Partimos de la siguiente ecuación, donde se asume que m_α es constante.

$$\mathbf{F} = m_\alpha f$$

Sea α un agente cualquiera. A continuación, veremos los cuatro efectos que influyen el movimiento del peatón.

Driving effect

Esta fuerza asegura que el agente α tenga movimiento de acuerdo a la velocidad deseada del agente (v_α^0) y su dirección de destino ($\mathbf{e}_\alpha^0$), [1]. Esta fuerza es descrita de la siguiente forma:

$$\mathbf{f}_\alpha^0(\mathbf{v}_\alpha, v_\alpha^0 \mathbf{e}_\alpha) = \frac{v_\alpha^0 \mathbf{e}_\alpha^0 - \mathbf{v}_\alpha}{t_\alpha}, \tag{2.1}$$

donde t^α es un factor de escalamiento temporal.

Boundary effect

Los peatones tienden a mantener distancia con los límites del espacio donde caminan. Cuando hay mayor acercamiento a los bordes los agentes prestan más atención, [1].

Este fenómeno puede ser descrito como un efecto repulsivo. El agente α repele el borde B siguiendo la divergencia $-\nabla_{r_{\alpha B}}$, [1]. Esta repulsión se describe de la siguiente forma:

$$\mathbf{f}_{\alpha B}(\mathbf{r}_{\alpha B}) = -\nabla_{\mathbf{r}_{\alpha B}} U_{\alpha B}(\|\mathbf{r}_{\alpha B}\|) \tag{2.2}$$

donde $U_{\alpha B}$ es el potencial de repulsión que decrece exponencialmente cuando se aleja el agente del borde.

$$U_{\alpha B}(\|\mathbf{r}_{\alpha B}\|) = U_{\alpha B}^0 e^{-\|\mathbf{r}_{\alpha B}\|/R} \tag{2.3}$$

Interaction effect

Este efecto promueve que cada agente mantenga cierta distancia del resto a la hora de moverse, [1]. Nuevamente se genera un efecto de repulsión con respecto a otro agente β descrito por la siguiente expresión:

$$\mathbf{f}_{\alpha\beta}(r_{\alpha\beta}) = -\nabla_{\mathbf{r}_{\alpha\beta}} V_{\alpha\beta}[b(r_{\alpha\beta})], \tag{2.4}$$

$$V_{\alpha\beta}[b(r_{\alpha\beta})] = V^0_{\alpha\beta} e^{-b/\sigma} \tag{2.5}$$

donde

- σ, es la distancia de interacción permitida.
- b, es el semi eje menor de la elipse que considera la distancia entre los agentes α y β.

$$2b = \sqrt{(\|\mathbf{r}_{\alpha\beta}\| + \|\mathbf{r}_{\alpha\beta} - \mathbf{v}_\beta \Delta t\|)^2 - \|v_\beta \Delta t\|^2} \tag{2.6}$$

Attractive effect

Este efecto se define como la atracción del agente α hacia un objeto i en el espacio, [1]. Se describe de la siguiente manera:

$$\mathbf{f}_{\alpha i}(\|\mathbf{r}_{\alpha\beta}\|, t) = -\nabla_{\mathbf{r}_{\alpha i}} W_{\alpha i}(\|\mathbf{r}_{\alpha\beta}\|, t). \tag{2.7}$$

La magnitud de este efecto decae conforme va pasando el tiempo y el agente pierde interés en el objeto.

En este trabajo no haremos uso de este efecto.

Effective angle

Tanto en *iteraction effect* como en *atractive effect* se da con mayor influencia para objetos que son percibidos en la dirección deseada del agente α que los que se encuentran detrás, [1]. Para tomar en cuenta esta circunstancia, tomamos en cuenta el peso ω_ϕ, [1], donde 2ϕ es el ángulo de visión, este peso se define de la siguiente forma:

$$\omega_\phi(\mathbf{e}, \mathbf{f}) = \begin{cases} 1 & \text{si } \mathbf{e} \cdot \mathbf{f} \leq \|\mathbf{f}\| \cos(\phi) \\ c_\phi & \text{e.o.c.} \end{cases}. \tag{2.8}$$

En particular para el valor de *interaction effect* esto resulta de la siguiente forma:

$$\mathbf{f}_{\alpha\beta}(\mathbf{e}_\alpha, \mathbf{r}_{\alpha\beta}) = \omega_\phi(\mathbf{e}_\alpha, -\mathbf{f}_{\alpha\beta}) \mathbf{f}_{\alpha\beta}(\mathbf{r}_{\alpha\beta}). \tag{2.9}$$

Efectos al movimiento

La suma de los cuatro efectos resulta en el movimiento del agente de acuerdo a la segunda ley de Newton:

$$\sum f_\alpha = \frac{d\mathbf{v}_\alpha}{dt} \tag{2.10}$$

$$\mathbf{f}_\alpha(t) = \mathbf{f}^0_\alpha + \sum_\beta \mathbf{f}_{\alpha\beta} + \sum_B \mathbf{f}_{\alpha B} + \sum_i \mathbf{f}_{\alpha i} \tag{2.11}$$

Es necesario considerar que este movimiento se pude ver afectado por fluctuaciones aleatorias.

$$\frac{d\mathbf{w}}{dt} = \mathbf{f}_\alpha(t) + \text{fluctuaciones} \tag{2.12}$$

Como no es posible simular tiempo continuo, es necesario considerar un paso de tiempo Δ_t lo suficientemente pequeño como para cambiar la velocidad del agente. Esta velocidad se determina de la siguiente forma:

$$\mathbf{w}_\alpha(t + \Delta t) = \mathbf{f}_\alpha(t) \Delta t + \mathbf{v}_\alpha(t). \tag{2.13}$$

Cada agente es limitado a una cantidad máxima de rapidez $v_\alpha^{\text{máx}}$. Actualizamos a la velocidad del agente de acuerdo a la siguiente regla:

$$\mathbf{v}_\alpha(t+\Delta t) = \begin{cases} v_\alpha^{\text{máx}}\hat{\mathbf{w}}_\alpha(t+\Delta t) & \|\mathbf{w}_\alpha(t+\Delta t)\| \leq v_\alpha^{\text{máx}} \\ \mathbf{w}_\alpha(t+\Delta t) & \text{e. o. c,} \end{cases} \tag{2.14}$$

donde $\hat{\mathbf{w}}_\alpha$ es la velocidad normalizada de la expresión (2.13).

De acuerdo a Helbing y Molnar la posición del agente se describe por $\searrow_\alpha$ y sigue la siguiente ecuación:

$$\frac{d\mathbf{r}_\alpha}{dt} = \mathbf{v}_\alpha(t). \tag{2.15}$$

En cada paso de tiempo se actualiza la posición de acuerdo a la siguiente regla:

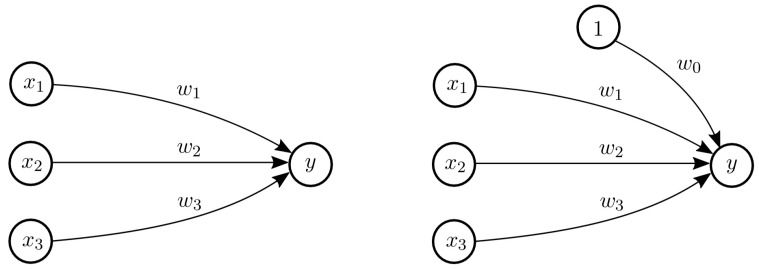
$$\mathbf{r}_\alpha(t+\Delta t) = \mathbf{v}(t)\Delta t + \mathbf{r}_\alpha(t). \tag{2.16}$$

Capítulo 3

Deep Reinforcement Learning

3.1. Redes neuronales

Una red neuronal artificial es un conjunto de neuronas, se compone por capas. En la primera capa hay unidades simples de procesamiento que reciben valores de entrada. En las siguientes capas la información fluye y se hacen relaciones con los datos. Finalmente, en las capas de salida se produce un resultado en función de los valores de entrada. En la figura podemos ver un esquema de una arquitectura simple.



(a) Without Bias (b) With bias

Figura 3.1: red simple, [2]

Para calcular el valor y, una neurona toma los valores ponderados de los valores de entrada, $y = \sum w_i x_i$, donde x_i es el valor de entrada y w_i es el peso correspondiente.

3.1.1. Redes multicapa

Para este trabajo nos concentraremos en las redes multicapa del tipo *feed forward* completamente conectadas. En general las redes neuronales feed forward constan de varias capas, a esta cantidad la denotaremos por L. En este caso la información fluye en una sola dirección y se le aplican transformaciones afines, estas transformaciones constan de escalamiento por un peso w y una traslación b y posteriormente se le aplica una función no lineal g. Finalmente, en la última capa se obtiene un valor $\hat{y}$ que es una estimación de la tarea dada a la red neuronal.

Para la capa $l \leq L$ se sigue la siguiente expresión:

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

en el caso $l = 1$, tenemos que $a^{[l-1]} = x$ que es el valor de entrada y en el caso $l = L$ tenemos que $a^{[l]} = \hat{y}$.

3.1.2. Entrenamiento

Aunque el objetivo de este trabajo no es importante mencionar los aspectos teóricos de las redes neuronales, mencionaremos brevemente en que consiste el entrenamiento.

El entrenamiento consiste en modificar o actualizar los parámetros w y b de cada capa. Esta actualización se hace de acuerdo al método de *backpropagation*, que consiste en usar la dirección del gradiente de la función $\mathcal{L}$ para modificar los valores mencionados. La función $\mathcal{L}$ evalúa el modelo de acuerdo a una tarea. Este proceso permite que $\mathcal{L}$ alcance el valor óptimo deseado y como consecuencia el modelo desempeñará su tarea de la mejor manera posible.

3.2. Reinforcement Learning

Uno de los objetivos principales de la inteligencia artificial es el de poder crear agentes *inteligentes*. Estos agentes deben de ser capaces de entender su ambiente, *aprender* y poder tomar decisiones para poder tener un comportamiento adecuado en una tarea específica. Esta tarea ha sido ampliamente estudiada desde la rama de *reinforcement learning* o aprendizaje reforzado. [3]

La meta de los modelos en reinforcement learning es el de poder desarrollar agentes que mejoren su desempeño en una tarea específica a partir de la interacción con su ambiente, que deberá enviar señales y recompensas por las acciones que efectúa el agente. El agente deberá encontrar las acciones que maximicen las recompensas obtenidas.

Empezaremos definiendo los conceptos asociados a los modelos de reinforcement learning. Vale la pena mencionar que esta área fue introducida en la segunda mitad del siglo XX y muchos de los métodos fueron enfocados para tareas en tiempo discreto y con una cantidad finita de acciones. Más adelante veremos esta área para tareas más complicadas en espacio y tiempo continuo.

3.2.1. Fundamentos de Reinforcement Learning

En este parte hablaremos de los componentes que tiene un modelo de reinforcement learning, así como conceptos importantes, métodos y algoritmos.

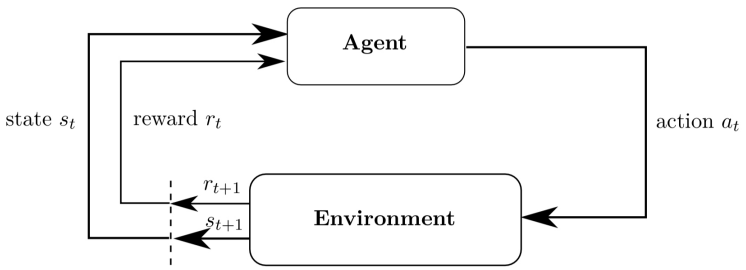


Figura 3.2: Esquema de interacción en RL, [4].

Agente

Llamamos *agente* a aquel que interactúa con el entorno o ambiente. El agente realiza ciertas acciones y recibe del ambiente observaciones y recompensas por dichas acciones. En la mayoría de los casos prácticos de

aprendizaje reforzado, el principal objetivo del agente es maximizar la cantidad de recompensas que pueda recibir.

Existen numerosos casos donde podemos hablar de agentes, por ejemplo:

- **Ajedrez**: Un jugador o un programa de computadora
- **Sistema dopaminérgico**: El cerebro de acuerdo a la información sensorial decide si la experiencia fue placentera o no.
- **Entrenamiento de un perro**: El perro.

Ambiente

Es todo aquello que el agente no puede controlar, el agente puede interactuar de manera directa o indirecta con el ambiente. El ambiente cambia conforme el agente realiza una acción o interacción y cada cambio se considera como un estado. Siguiendo los ejemplos que expusimos anteriormente, encontramos los ambientes siguientes:

- **Ajedrez**: Lo conforma el tablero y el oponente.
- **Sistema dopaminérgico**: Lo conforma todo el sistema nervioso, los órganos y todo lo que pueda percibir un individuo.
- **Entrenamiento de un perro**: Lo conforma todo lo que un perro puede encontrarse en el camino, incluyendo otros perros, el entrenador y situaciones diversas de la vida cotidiana.

Recompensa

Es un valor numérico que recibe el agente como respuesta directa a sus acciones, las recompensas motivan a que el agente tenga un comportamiento adecuado. Las recompensas positivas premian una acción deseada; mientras que las recompensas negativas castigan las acciones de las que el agente debe alejarse. En los ejemplos que hemos seguido encontramos las siguientes recompensas:

- **Ajedrez**: Se determina a partir de la conclusión del juego, es decir, se da cierta recompensa si el agente perdió, gano o empato.
- **Sistema dopaminérgico**: El sistema límbico produce dopamina cada vez que se necesita enviar una señal positiva al cerebro. Altas concentraciones de dopamina dan una sensación de placer que refuerzan ciertas actividades consideradas como buenas por el sistema.
- **Entrenamiento de un perro**: Se suelen dar premios o caricias al perro si este realiza un comportamiento deseado por el entrenador, por el contrario si el perro no tiene un comportamiento adecuado se le suele castigar.

La recompensa la solemos denotar por r o R_t, si queremos distinguir el tiempo t en que se obtuvo, es un valor en $\mathbb{R}$. Al no tener certeza del valor de R_t antes de obtenerlo, formalmente se le considera una variable aleatoria. Al espacio de recompensas lo denotamos por $\mathcal{R}$. [4]

Estado

A medida que el agente interactúa con un entorno, el proceso da como resultado una secuencia de acciones (A_t) y recompensas (R_t), lo que sucede a continuación al paso del tiempo depende de esta secuencia, cuando hay una nueva configuración del ambiente, decimos que tenemos un nuevo estado. Formalmente, denotamos al estado al tiempo t como S_t. En general al no tener certeza de los estados por los que se pasan, consideramos a las (S_t) como variables aleatorias. [4]

Muchas veces la acción que se toma depende del estado, es decir, no todas las acciones están disponibles en ciertos estados. Denotamos al espacio de acciones disponibles en el estado s como $\mathcal{A}(s)$. Al conjunto de estados lo denotamos por $\mathcal{S}$. [4]

Episodio

Es la trayectoria de estados, acciones y recompensas desde que el agente interactúa por primera vez con el ambiente hasta que finaliza su interacción y se suele ordenar de la siguiente manera:

$$S_0, A_0, R_1, S_1, \cdots, S_t, A_t, R_{t+1}, \cdots, S_{T-1}, A_{T-1}, R_T \quad 0 < t < T,$$

donde T es un tiempo terminal. [4] Gráficamente, se puede ver de la siguiente manera:

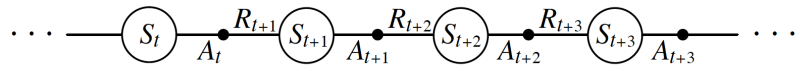


Figura 3.3: Episodio

Rendimiento Esperado

Denotaremos al rendimiento o ganancia esperada en el tiempo t como G_t y se define de la siguiente manera:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \quad 0 \leq \gamma \leq 1.$$

Donde γ es una tasa de descuento. Esta cantidad nos habla de cuanto esperó ganar desde el tiempo t, y la tasa de descuento pondera que tan importante consideró el futuro. Por ejemplo cuando $\gamma = 1$ el rendimiento solo es la suma de las futuras recompensas, en cambio cuando $\gamma = 0$ solo me interesa la recompensa inmediata. [4]

Política

Es un mapeo entre los estados del ambiente percibidos hacia acciones a tomar en esos estados. Comúnmente se le denota por $\pi(\cdot) : \mathcal{S} \to \mathcal{A}$ y define el comportamiento del agente en el tiempo, [3].

3.2.2. Procesos de Decisión Markovianos

En un proceso de decisión de Markov o *MDP* (*Markov decision processes*), los conjuntos $\mathcal{S}, \mathcal{A}, \mathcal{R}$ tienen una cantidad finita de elementos. En este caso, las variables aleatorias R_t y S_t tienen un distribución de probabilidad discreta bien definida, sólo dependen de la información anterior, [4]. En particular para $s' \in \mathcal{S}$ y $r \in \mathcal{R}$ al tiempo t, existe una probabilidad de transición desde el tiempo $t-1$, dada por la siguiente expresión

$$p(s', r|s, a) = \mathbb{P}[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a] \quad \forall s', s \in \mathcal{S}, r \in \mathcal{R} \text{ y } a \in \mathcal{A}(s)$$

Observe que

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s).$$

En los *MDP*, las probabilidades dadas por p caracterizan completamente la dinámica del ambiente. En los *MDP* también se cumple la *propiedad de Markov*, existen modelos similares donde no se cumple esta propiedad, pero no se detallará en este trabajo.

Función de valor

A partir de la definición de la política podemos definir la función de valor del estado s dada la política π como

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s]$$

En la literatura se le conoce como *State-value function*. Esta función nos permite evaluar políticas. y representa la ganancia esperada dado que se esta en un estado S. [4]

Función Q

También llamda *Action-value function* se define de la siguiente manera:

$$Q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a]$$

La idea intuitiva del propósito de la fórmula anterior es la de determinar el rendimiento esperado dado que estoy en el estado s y tomo la acción a, [4].

Ecuación de Bellman

Las ecuaciones de Bellman describen como las funciones d valor y Q se pueden descomponer en un términos de la recompensa inmediata y el valor del siguiente estado.

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1})|S_t = s] \tag{3.1}$$

$$Q(s,a) = \mathbb{E}[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})|S_t = s, A_t = a] \tag{3.2}$$

Un problema en reinforcement learning es encontrar la política óptima, es decir la que nos de la máxima ganancia. Al menos existe una política óptima denotada por π^* que es mejor que todas las demás , [4]. Todas la políticas óptimas comparten el mismo valor óptimo de la función de valor y la función Q. Los óptimos de dichas funciones las denotamos de la forma siguiente

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

A partir de las expresiones anteriores se puede determinar el siguiente criterio de optimalidad que están dadas por las *ecuaciones de optimalidad de Bellman*:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma V^*(s_{t+1})|S_t = s, A_t = a]$$

$$Q^*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')|S_t = s, A_t = a].$$

Si Q^* es conocida, la política óptima esta definida de la siguiente forma:

$$\pi^*(a|s) = \begin{cases} 1 & \text{si } a = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a) \\ 0 & \text{e.o.c} \end{cases} \tag{3.3}$$

Decimos que la política óptima es *greedy* con respecto a la función Q. Esta política define las acciones óptimas sin conocer la dinámica del ambiente (las probabilidades de transición), [4].

3.2.3. Q-learning

Este algoritmo consiste en encontrar Q^*, pertenece a los métodos *temporal- difference* (TD). Q-learning es un enfoque del tipo *model-free*, que quiere decir que no se necesita conocer la dinámica del ambiente.

Q-learning es método tabular e iterativo, que no espera conseguir el valor final de R_t para actualizar el valor de $Q(s_t, a_t)$, lo hace en cada paso de tiempo. La regla de actualización es la siguiente

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_t, a) - Q(s_t, a_t)].$$

En este trabajo, no entraremos en mucho detalle con el algoritmo pues no es el objetivo principal. Lo que es importante mencionar es que este método es del tipo *off-policy*, es decir, este método aprende una política óptima sin importar la política que sigue en el paso actual, [4].

Para garantizar la convergencia de este método se utiliza una tasa de aprendizaje $\alpha \in (0, 1)$.

Exploración vs Explotación

Un aspecto importante debe ser escoger una *política de control* que haga un balance entre exploración y explotación, es otras palabras, queremos una política que nos de la máxima recompensa, pero de vez en cuando busqué una mejor opción, que seleccioné una acción que no había intentado. Para este enfoque es muy útil las llamadas políticas ϵ-greedy:

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{si } a = argamax_{a \in \mathcal{A}} Q(s,a) \\ \epsilon/m & \text{e.o.c.} \end{cases}$$

3.3. Reinforcement Learning con Redes Neuronales

Los métodos tabulares como Q-learning resultan muy útiles para tareas con pocas acciones y estados. Sin embargo, cuando se trabaja en espacio y tiempo continuo, se pueden tener una infinidad de acciones y estados disponibles, por lo que no es factible utilizar tablas para almacenar los valores de Q^*. En la mayoría de los casos no conocemos la dinámica del ambiente, por lo cual es necesario *estimar* la función Q. En este caso es necesario usar la información de experiencias previas para poder *generalizar* y usar ese aprendizaje en nuevas experiencias. Para poder hacer esto, necesitamos que las funciones de valor sean aproximadas por una *función de aproximación.*

Un enfoque que ha sido ampliamente usado en la aproximación de funciones es el uso de redes neuronales debido a su habilidad para estimar funciones no lineales.

3.3.1. Aproximación de función de valor con Redes Neuronales

Cuando se usa una red neuronal para aproximar a la función Q es posible representar el valor de la función para espacio de estados extenso, [2]. El uso de redes multicapa representa una buena ventaja para generalizar pues desempeñan como aproximadores globales, [2].

Como se puede observar en la siguiente figura, el estado s_t y la acción a_t sirven como valores de entrada de la red y el output corresponde al valor aproximado de Q.

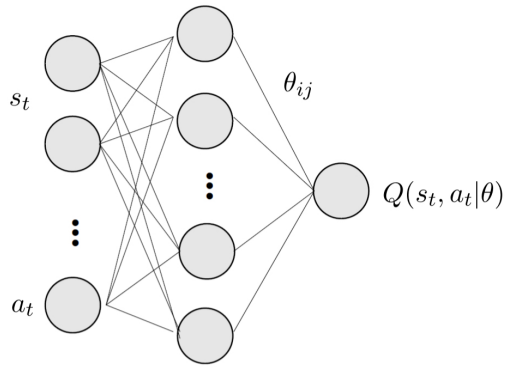


Figura 3.4: Ejemplo de red que estima la función Q, [2]

Es necesario entrenar a la red neuronal para aproximar Q, esto se hace de la misma manera que revisamos anteriormente. El objetivo del entrenamiento es encontrar los pesos θ que mejor aproximen a Q. Existe el riesgo de que estos aproximadores no sean inestables o diverjan, más adelante veremos como lidiar con este problema, [2].

Experience Replay

Una razón para que se de la inestabilidad se debe a que en el entrenamiento se actualizan los valores de la función y puede causar que la red 'olvidé' la información que vio antes, [2]. Una solución a este problema es el uso de *experience replay*, esta técnica consiste en almacenar la información $(s_t, a_t, s_{t+1}, r_{t+1})$ de tiempos anteriores en la memoria para después ser usada en el entrenamiento de la red, [2]. Esto puede acelerar el proceso de aprendizaje.

Otra causa de la inestabilidad puede deberse a la correlación entre las muestras de entrenamiento. Una vez más experience replay ofrece una solución pues toma muestras de forma aleatoria en el entrenamiento y se ha observado que esto estabiliza el entrenamiento, [2].

3.3.2. Deep Deterministic Policy Gradient

El método descrito en esta sección es un ejemplo del uso de de redes neuronales en problemas de reinforcement learning. El algoritmo llamado *deep determin istic policy gradient*(DDPG) es del tipo *model-free*, *actor-critic* y consiste en construir una red para estimar política óptima y otra para estimar el valor óptimo de Q. Este modelo propuesto por Lillicrap et al. [5], se usa para que el agente desempeñe tareas en problemas de control continuo.

La red que aproxima a Q la llamamos *critic network* o Q *network* y el valor aproximado se denota por $Q(s, a|\theta^Q)$. La red que aproxima la política la llamamos *actor network* o *deterministic policy function* y la denotamos por $\mu(s|\theta^\mu)$.

Este método ha resultado estable durante el entrenamiento ya que usa experience replay. Adicionalmente usa un mecanismo de adaptación para el aprendizaje, este mecanismo consiste en tener una copia de la red critic y la red actor, llamadas *target Q network* $Q(s, a|\theta^{Q'})$ y *target policy network* $\mu(s|\theta^{\mu'})$. Cuando se obtienen nuevos pesos se copian a las redes target y se aplica la siguiente regla de actualización:

$$\begin{aligned}\theta^{Q'} &\rightarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\rightarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'},\end{aligned}$$

donde $\tau \in (0, 1)$ es un parámetro de estabilidad del aprendizaje.

No entraremos en detalles de la parte teórica del modelo, sin embargo es importante mencionar que para el entrenamiento se usa la función de mínimos cuadrados, que esta descrita por la siguiente expresión:

$$\mathcal{L}_t(\theta_t^Q) = \mathbb{E}_{(s,a,r,s')}\left[(r + \gamma Q'(s, \mu'(s|\theta_t^{\mu'})|\theta_t^{Q'}) - Q(s, a|\theta_t^Q))^2\right],$$

esta expresión es diferenciable y por lo tanto el método admite el uso de backpropagation para la actualización de pesos.

Exploración vs Explotación en DDPG

Al igual que en el método usual de Q-learning es necesario no sólo explotar una buena política, si no que es necesario explorar una nueva y encontrar una mejor. Para poder realizar la exploración que permita mejorar la política, se usa el proceso de Ornstein-Uhlenbeck $\mathcal{O}$, que genera ruido a las acciones tomadas. Usando este ruido definimos la siguiente política:

$$\begin{aligned}\mu'(s_t) &= \mu(s_t|\theta_t^\mu) + \mathcal{O} \\ \mathcal{O}(k) &= \mathcal{O}(k-1) - \theta_{OU}\mathcal{O}(k-1) + \sigma_{OU}\mathcal{N}.\end{aligned}$$

No entraremos en los detalles de este proceso, pero vale la pena mencionar, que el ruido generado decae en cada iteración de tal forma que el aprendizaje converja.

Algoritmo 1: *Deep deterministic policy gradient* (Lillicrap et at., [5])

Inicializar aleatoriamente $Q(s, a|\theta^Q)$ y $\mu(s|\theta^\mu)$ con θ^Q y θ^μ
Inicializar Q' μ' con $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \rightarrow \theta^\mu$
Inicializar el replay buffer $\mathcal{D}$
para *cada episodio* **hacer**
    Inicializar el proceso $\mathcal{O}$ para la exploración
    Recibir una observación inicial s_1
    para *cada paso en el episodio* **hacer**
        Seleccionar $a_t = \mu(s_t|\theta^\mu) + \mathcal{O}$
        Ejecutar acción a_t, observar recompensa r_t y el nuevo estado s_{t+1}
        Almacenar (s_t, a_t, r_t, s_{t+1}) en $\mathcal{D}$
        Tomar una muestra con un tamaño N de la forma (s_i, a_i, r_i, s_{i+1}) de $\mathcal{D}$
        Establecer $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Actualizar critic minimizando $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^Q))^2$
        Actualizar actor usando *sampled policy gradient*:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\Big|_{s=s_i}$$

        Actualizar las redes target:

$$\theta^{Q'} \rightarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \rightarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'},$$

Capítulo 4

Metodología

4.1. Planteamiento del problema

En un ambiente simulado donde haya peatones caminando, un robot tendrá que alcanzar un punto fijado en el espacio, su tarea será aprender la manera de alcanzar dicho punto, evitando colisiones con otros peatones y respetando las normas sociales establecidas. El ambiente enviará a la recompensa como señal y usado el algoritmo de *DDPG* deberá de aprender la política optima para alcanzar su objetivo.

El ambiente simulado esta basado en el modelo e fuerzas sociales. Para simplificar la interacción del agente con el ambiente, se consideró un espacio sin fronteras y sin obstáculos.

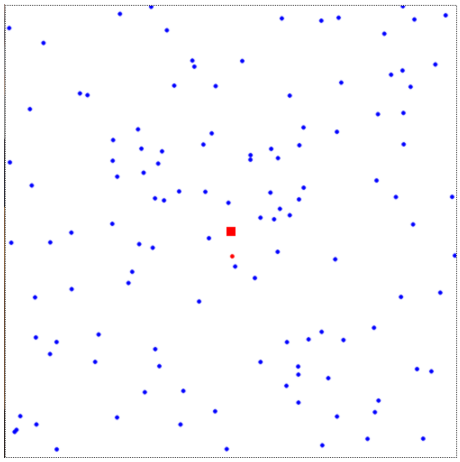


Figura 4.1: Ambiente diseñado para el robot (círculo rojo) con meta en el centro del plano (parche rojo).

Definimos al estado del robot como $s_t = (x_t, y_t, \phi_t)$ que representan las coordenadas cartesianas y la dirección en el tiempo t. Consideraremos que el agente se mantiene a una velocidad constate de $1{,}34\frac{m}{s}$, sólo será necesario que aprenda la acción $a_t = \omega_t$ que representa la velocidad angular.

En todo momento el robot sabrá su posición en el espacio con lo que podrá medir su distancia hacia su objetivo.

4.2. Función de Recompensa

La función reward esta diseñada para promover que el robot se mueva de forma suave al objetivo, mientras que en cada paso de tiempo evita colisiones y mantiene una distancia cómoda con los peatones. Lo definimos de la siguiente forma:

$$r_t = r(g)_t + r(a)_t + \left(\sum_{h=1}^{H} r(h)_t \right)$$

La ecuación consiste de tres partes. La recompensa $r(h)$ penaliza el que el agente se acerque a los peatones *humanos* del ambiente. Se escala la recompensa r_h de acuerdo a una distribución gaussiana que depende de la distancia del robot con el humano como se muestra a continuación:

$$r(h)_t = r_h \frac{1}{\sigma_h \sqrt{2\pi}} \exp(-(d(x_t, y_t, x_h, y_h) - \mu_h)^2 / 2\sigma_h^2), \tag{4.1}$$

esta expresión nos permite darle mayor peso a aquellos humanos que estén más cerca de la distancia socialmente permitida.

Adicionalmente queremos recompensar al agente por alcanzar su objetivo. Si el agente se acerca lo suficiente es recompensado y si no recibe una recompensa proporcional a su progreso con respecto al paso anterior.

$$r(g)_t = \begin{cases} r_g & \text{si } d(x_t, y_t, x_g, y_g) < D_g \\ r_p \cdot l_t & \text{e.o.c,} \end{cases} \tag{4.2}$$

$$l_t = d(x_{t-1}, y_{t-1}, x_g, y_g) - d(x_t, y_t, x_g, y_g) \tag{4.3}$$

donde l_t es el cambio de distancia con respecto al paso anterior y D_g el radio de acercamiento al objetivo para recibir r_g.

Finalmente, $r(a)_t$ penaliza al agente con la recompensa r_a para valores muy grandes o muy negativos de la velocidad angular. Esto promueve que el agente se mueva de forma suave.

$$r(a)_t = r_a |\omega_t|$$

Usaremos los valores sugeridos por [2].

Recompensa	Valor
r_h	-2.5
r_g	+5.0
r_p	+2.5
r_a	-0.03

Umbral	Valor
D_g	1.0
σ_h	1.0
μ_h	0.0

4.3. Evaluación

El entrenamiento consiste de 8 episodios con un máximo de 25 pasos de tiempo, en cada episodio se entrenará al modelo y luego se probará. La diferencia entre el entrenamiento y la prueba radica en que durante el entrenamiento el algoritmo recopila información y la alamacena en un *buffer* donde se efectuará el experience replay para entrenar las redes Q y μ, posteriormente se evaluará el modelo, en cambio, en el entrenamiento sólo se evaluará al modelo.

Para garantizar que el agente generalice cada vez mejor, en cada episodio el agente se ubicará en una posición aleatoria cercana al objetivo. La posición del agente (x_t, y_t) al iniciar el episodio se encontrará en la región uniforme $[x_g - 2{,}5, y_g - 2{,}5] \times [x_g + 2{,}5, y_g + 2{,}5]$.

Capítulo 5

Resultados

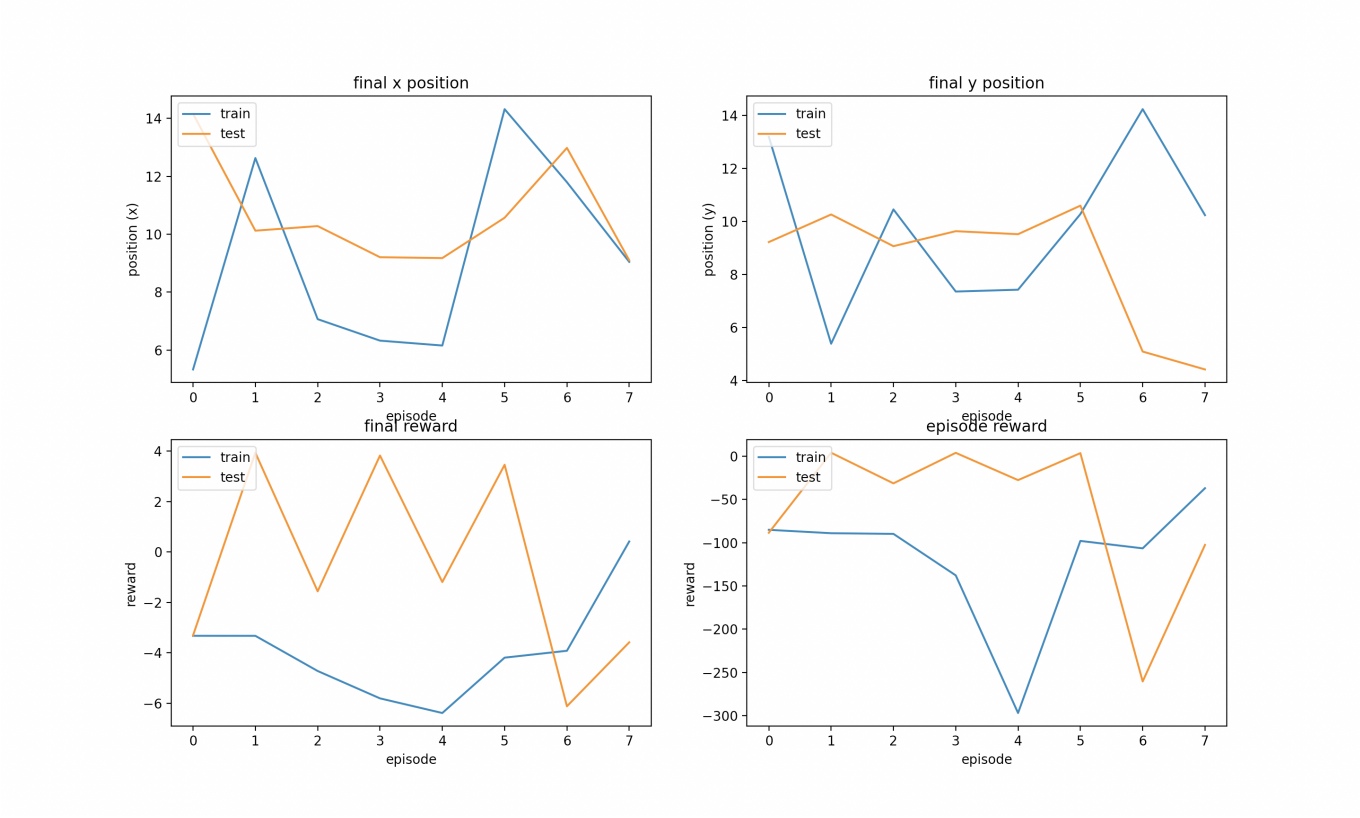


Figura 5.1: Resultados de la prueba

Debido a la poca cantidad de pasos no podemos observar una tendencia en el aprendizaje. Notamos que debido al tiempo de entrenamiento se requiere optimizar el código para hacer uso del GPU. También es posible paralelizar algunos procesos para hacer más eficiente la ejecución del código.

5.1. Conclusión

A pesar de no obtener un resultado contundente, se lograron varios resultados que pueden ayudar a mejorar este trabajo en el futuro. En primer lugar se logró diseñar un ambiente funcional para el entrenamiento de

la navegación autónoma de un robot. Se logró diseñar toda la estructura que permite entrenar a el robot y posteriormente, usar la aprendido en una simulación con visualización.

5.2. Futuro trabajo

Optimizando el código y haciendo uso del GPU y otras herramientas que permitan acelerar el proceso de entrenamiento, podría permitir establecer objetivos más ambiciosos como aumentar el tamaño de la vecindad donde aparecerá el robot, hacer uso de obstáculos y fronteras en el ambiente e incluso podría establecer el objetivo de implementar un problema de múltiples agentes en tareas navegación autónoma en un ambiente basado en el ascenso eficiente de pasajeros propuesto en [6].

Apéndice A

Simulación

Para crear la simulación se uso Mesa, que es un paquete de código abierto Apache 2.0 de Python. Este paquete permite al usuario crear modelos basado en agentes de forma sencilla. Se puede crear simulaciones muy simples usando los componentes *schedulers* y grillas espaciales. Además incluye herramientas de visualización de simulación y datos en tiempo real.

Apéndice B

Parámetros de entrenamiento

La implementación del agente para tareas de control continuo está basado en el código usado para resolver el problema del péndulo invertido mencionado en [7]. El código fue hecho en Python usando de framework Pytorch para la parte de redes neuronales.

Parámetro	**Valor**
Optimizador	Adam
Episodios	10
Pasos por episodio	80
Prueba por episodio	1
Tasa de aprendizaje de *actor*	0.001
Tasa de aprendizaje de *critic*	0.001
Número de capas ocultas	2
Neuronas por capa oculta	64
Función de activación	relu
Función de salida	lineal
γ	0.99
Batch size	32
Replay memory size	1000000
tau	0.0001
θ_{OU}	0.15
σ_{OU}	0.2

Bibliografía

[1] M.Q Capelle. Implemenation and validation of the social force model for crowd behaviour. *Delft University of Technology*, 1:1–25, 8 2018.

[2] Benjamin Coors. Navigation of mobile robots in human enviorments with deep reinforcement learning. *KTH Royal Institute of Tecnology School of computer science and communication*, 1:5–73, 6 2016.

[3] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, 2018.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[6] Gustavo Carreón Vázquez. Modelos y simulaciones multi-escala de sistemas de transporte público. 2018.

[7] Chris Yoon. Deep deterministic policy gradient explained, Mar 2019.