Navegación Autónoma de Robots Móviles mediante Deep Q-Network (DQN)

Documentación Técnica del Proyecto

4 de diciembre de 2025

Resumen

Este documento describe la formulación matemática y la configuración experimental utilizada para el entrenamiento de un agente robótico TurtleBot3. Se detalla el diseño de la función de recompensa (*Reward Shaping*), la arquitectura de la red neuronal y la justificación de los hiperparámetros seleccionados para el algoritmo DQN.

1. Formulación Matemática

El problema de navegación se modela como un Proceso de Decisión de Markov (MDP) definido por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma)$.

1.1. Espacio de Estados ($\mathcal{S}$)

Para capturar la dinámica temporal del entorno (velocidad y aceleración relativa de obstáculos), el estado s_t no es una única lectura sensorial, sino una concatenación temporal (*Frame Stacking*) de k observaciones consecutivas.

Sea $x_t \in \mathbb{R}^N$ el vector de características en el tiempo t, compuesto por N_{lidar} lecturas de escáner láser y la posición polar relativa al objetivo. El estado se define como:

$$s_t = [x_t, x_{t-1}, \dots, x_{t-k+1}] \tag{1}$$

Donde $k = 3$ (History Length) y la dimensión total de la entrada es $(N_{lidar} + 2) \times k$.

1.2. Espacio de Acciones ($\mathcal{A}$)

El agente opera en un espacio de acciones discreto $\mathcal{A} = \{0, 1, 2, 3, 4\}$. Estas acciones se traducen en comandos de velocidad lineal (v) y angular (ω) para el robot diferencial:

$$(v, \omega)_t = \begin{cases} (0, 0) & \text{si } a_t = 0 \quad \text{(Stop)} \\ (v_{ref} + \delta_v, \omega_{turn}) & \text{si } a_t = 1 \quad \text{(Giro Izq.)} \\ (v_{ref} + \delta_v, -\omega_{turn}) & \text{si } a_t = 2 \quad \text{(Giro Der.)} \\ (v_{ref} - \delta_{back}, 0) & \text{si } a_t = 3 \quad \text{(Retroceso/Lento)} \\ (v_{ref} + \delta_{fast}, 0) & \text{si } a_t = 4 \quad \text{(Avance Rápido)} \end{cases} \tag{2}$$

1.3. Ingeniería de Recompensas (Reward Shaping)

Para acelerar la convergencia y garantizar movimientos suaves, se implementa una función de recompensa compuesta. La recompensa total R_t es:

$$R_t = r_{base} + r_{prog} + r_{head} + r_{obs} + r_{vel} + r_{wiggle} - r_{perp} \tag{3}$$

1.3.1. Progreso y Alineación

Sea $\vec{p}_{rob}$ la posición del robot, $\vec{p}_{goal}$ la meta y $\vec{p}_{start}$ el inicio. Definimos el vector ruta ideal $\vec{v}_{path} = \vec{p}_{goal} - \vec{p}_{start}$. La recompensa por progreso se basa en la proyección escalar del desplazamiento sobre la ruta ideal:

$$r_{prog} = k_{prog} \cdot (proj_t - proj_{t-1}) \tag{4}$$

Adicionalmente, se penaliza la desviación perpendicular $d_\perp$ (distancia del robot a la recta ideal) para mantener la trayectoria recta:

$$r_{perp} = k_{perp} \cdot d_\perp \tag{5}$$

1.3.2. Penalización de Obstáculos (Safety)

Para evitar colisiones de forma proactiva, se utiliza una función de penalización hiperbólica basada en la distancia mínima al obstáculo (d_{min}). Si $d_{min} < d_{safe}$:

$$r_{obs} = -4{,}0 \cdot \left(\frac{d_{safe} - d_{min}}{d_{min}} \right) \tag{6}$$

Esta formulación penaliza exponencialmente a medida que el robot se acerca al límite de seguridad.

1.3.3. Estabilidad de Control

Para evitar oscilaciones rápidas ("zig-zagueo"), se aplica una penalización si el agente alterna acciones opuestas consecutivamente:

$$r_{wiggle} = -0{,}5 \quad \text{si } (a_t = 1 \wedge a_{t-1} = 2) \vee (a_t = 2 \wedge a_{t-1} = 1) \tag{7}$$

1.4. Algoritmo y Pérdida

Se utiliza la actualización de Bellman con una red objetivo (*Target Network*) para estabilizar el aprendizaje. La función de pérdida $\mathcal{L}(\theta)$ a minimizar es el Error Cuadrático Medio (MSE):

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \tag{8}$$

2. Configuración Experimental e Hiperparámetros

A continuación se detallan los hiperparámetros utilizados en el entrenamiento, extraídos directamente de la implementación del agente y el nodo de entrenamiento.

Cuadro 1: Hiperparámetros del Algoritmo DQN y Justificación

Parámetro	**Valor**	**Justificación y Efecto**
Learning Rate (α)	0,001	Valor estándar para el optimizador Adam. Un valor mayor podría desestabilizar la red, mientras que uno menor haría el aprendizaje demasiado lento.
Gamma (γ)	0,95	Factor de descuento. Se eligió 0.95 en lugar del clásico 0.99 para priorizar recompensas a mediano plazo, lo cual es crucial en navegación reactiva para evitar colisiones inminentes.
Epsilon Decay	0,996	Decaimiento lento de la exploración. Dado que el entorno es continuo y complejo, se requiere mantener la exploración activa durante más episodios para evitar mínimos locales.
Batch Size	64	Tamaño del lote para el *Experience Replay*. Provee un buen equilibrio entre estabilidad del gradiente (varianza reducida) y velocidad computacional.
Memory Size	5000	Capacidad del buffer de repetición. Limitado a 5000 transiciones para asegurar que el agente aprenda de experiencias relativamente recientes, adaptándose a cambios en la estrategia de navegación.
Frame Stack (k)	3	Número de tramas apiladas. Permite a la red neuronal inferir la velocidad y dirección de los objetos dinámicos, algo imposible con una sola lectura de LiDAR.
Arquitectura MLP	128×128	Dos capas ocultas de 128 neuronas. Suficiente capacidad para aproximar la función Q no lineal sin incurrir en un costo computacional excesivo para inferencia en tiempo real.
Target Update	100 pasos	Frecuencia de actualización de la red objetivo. Desacopla la red de predicción de la de objetivo, reduciendo la divergencia durante el entrenamiento.

2.1. Notas sobre la Implementación

El entrenamiento se realiza utilizando `sklearn.neural_network.MLPRegressor` con el solver 'adam'. Se ha implementado un mecanismo de seguridad donde el robot reduce su velocidad lineal proporcionalmente a la cercanía de obstáculos, independientemente de la acción seleccionada por la red, para facilitar el aprendizaje en etapas tempranas.