UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

EVASIÓN DE COLISIÓN PARA CONDUCCIÓN AUTÓNOMA EN UN AMBIENTE URBANO USANDO APRENDIZAJE REFORZADO PROFUNDO Y APRENDIZAJE POR REPRESENTACIÓN

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOSÉ MANUEL VILLAGRÁN ESCOBAR

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
CÉSAR AZURDIA MEZA
JUAN ZAGAL MONTEALEGRE

Esta tesis ha sido parcialmente financiada por los Proyectos
FONDECYT 1201170 y FONDEQUIP EQM1700041

SANTIAGO DE CHILE
2022

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS
DE LA INGENIERÍA, MENCION ELÉCTRICA y
MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: JOSÉ MANUEL VILLAGRÁN ESCOBAR
FECHA: 2022
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

EVASIÓN DE COLISIÓN PARA CONDUCCIÓN AUTÓNOMA EN UN AMBIENTE URBANO USANDO APRENDIZAJE REFORZADO PROFUNDO Y APRENDIZAJE POR REPRESENTACIÓN

Esta tesis aborda el problema de evasión de colisiones en vehículos autónomos para un escenario urbano usando un paradigma de aprendizaje reforzado. Si bien en la literatura se ha tratado este problema desde un foco de aprendizaje reforzado nunca se ha hecho desde una perspectiva que combine aprendizaje por imitación, reforzado y por representación. Una de las principales contribuciones de este trabajo es la propuesta de un algoritmo de aprendizaje reforzado junto a demostraciones expertas que hacen uso de representaciones de estados para el aprendizaje de una política de conducción autónoma. El algoritmo en cuestión es denominado TD3CoL y combina las ideas del algoritmo CoL más los beneficios del algoritmo TD3. Para validar la idea planteada se utiliza el simulador de vehículos autónomos CARLA, en donde se instancian vehículos con distintas posiciones y orientaciones que sirven como obstáculos para el agente en su llegada a un objetivo. Mediante una evaluación exhaustiva del controlador neuronal obtenido se comprueba la robustez de la política obtenida en 7 configuraciones diferentes del ambiente. Demostrándose que el algoritmo TD3CoL supera en desempeño a lo realizado por CoL y DDPG para las métricas radio de colisión y recompensa promedio en la situación planteada.

Agradecimientos

Agradezco al profesor Javier Ruiz del Solar por ser mi profesor y guía de este trabajo, facilitando su desarrollo a través de criticas constructivas y consejos sobre la misma.

Agradezco también a César Azurdia y Juan Cristóbal Zagal por tener la voluntad de formar parte de mi comisión evaluadora de tesis, y por darme preciados comentarios de la misma para enriquecer y mejorar el contenido de esta.

También agradezco a mi familia por acompañarme durante toda esta época de estudiante y darme apoyo incondicional en todo lo que necesitase.

Finalmente, agradezco a los proyectos FONDECYT 1201170 y FONDEQUIP EQM1700041 por el financiamiento parcial que otorgaron a este trabajo.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

1. Introducción

1.1. Motivación

Hace algunos años Klaus Schwab acuño el término "Cuarta revolución industrial" [1], término que engloba toda industria que se enmarca en el uso de tecnologías digitales en un marco de acción, mayoritariamente, autónomo. Los principales objetivos de esta revolución consisten en la interconexión de más personas a la web, automatización inteligente para el manejo y optimización de recursos, entre otras. Este mismo personaje hacía especial mención a las industrias de robótica, internet de las cosas, neurociencia y vehículos autónomos. Justamente, esta última ha sido una de las más lucrativas industrias al día de hoy por la implicancia que significaría la solución al problema de autonomía completa por parte de vehículos urbanos en términos de eficiencia energética, calidad de vida de las personas y disminución en accidentes por errores humanos.

En consecuencia a lo anterior, conducción autónoma, o más conocido como *Autonomous Driving* (AD), se ha convertido en uno de los campos de mayor atracción, no solo para la investigación científica, sino que para la industria en general. Sin embargo, su atracción no quita lo desafiante y demandante que puede llegar a ser la solución a diferentes problemas que sufre el manejo autónomo. Y si se proyecta la masiva entrada de estos vehículos al mercado, estos deben poder manejar interacciones con los demás agentes de ruta, tales como peatones, bicicletas, vehículos, etc. Como también ser capaces de manejar situaciones de riesgo e imprevistas.

1.1.1. Conducción Autónoma

Conducción autónoma o *Autonomous Driving* es la actividad de manejo de un vehículo motorizado sin la necesidad de asistencia humana. Tradicionalmente, y en el contexto de vehículos autónomos, significa "autogobernación", sin embargo, según la SAE (*Society of Automotive Engineers*) se asemeja más a la "autosuficiencia" debido a que aún está presente el humano dentro de la construcción del software y posteriores tomas de decisiones por parte del mismo [2], por lo que en sí mismo, aún no serían sistemas autónomos con respecto al significado estricto de la palabra. Esta misma sociedad definió un conjunto de niveles de automatización de vehículos autónomos. Estos son:

- Nivel 0 - *No Driving Automation*: El conductor humano es el único en conducir el vehículo. No hay intervención de ningún sistema autónomo.
- Nivel 1 - *Driver Assistance*: Asistencia compartida entre el conductor humano y el sistema autónomo. Este último puede realizar algunas tareas asignadas por el humano, tal como tareas de control lateral o longitudinal del vehículo.
- Nivel 2 - *Partial Driving Automation*: El sistema autónomo ejecuta las tareas de control lateral y longitudinal, sin embargo, el conductor humano debe estar preparado para intervenir. Un ejemplo de esto sería la unión de los sistemas de control crucero y los sistemas de mantenimiento de carril.

- Nivel 3 - *Conditional Driving Automation*: La intervención humana ya no es requerida de forma inmediata, sino que el sistema autónomo la pide con antelación si es necesario.
- Nivel 4 - *High Driving Automation*: No es necesaria ninguna intervención humana porque el sistema automáticamente se encargará de efectuar maniobras que alcancen una condición de mínimo riesgo de manera autónoma. Aunque el sistema autónomo tendrá limitantes en donde puede ser activado.
- Nivel 5 - *Full Driving Automation*: El sistema autónomo tendrá capacidad absoluta para el control del vehículo para la mayoría de situaciones que se requieran. Además, de no necesitar de intervención humana en lo absoluto.

En la actualidad los vehículos en su mayoría están categorizados en el nivel 0, que es básicamente la forma tradicional en la cual se conducen los vehículos. Aunque una gran mayoría de vehículos cuentan también con asistencia de manejo. Por lo que el nivel 1 no es algo totalmente desconocido. Un ejemplo de estos sistemas que cumplen con esta categoría son los sistemas de control de crucero adaptativo (*Adaptive Cruise Control*) los cuales controlan la velocidad del vehículo dada alguna situación particular. Y los asistentes de mantenimiento de línea (*Line-Keep Assistant*) que se encargan de mantener el vehículo entre líneas de un mismo carril. Los vehículos que cumplen con la categoría de nivel 2 no son masivos, pero han ganado bastante popularidad gracias al sistema llamado *Autopilot* incorporados en los vehículos Tesla. Este tipo de vehículos poseen sistemas que les permiten realizar control de crucero adaptativo y mantenimiento de línea de forma simultánea. Por su parte, los vehículos de nivel 3 son aún más desconocidos y difícil de obtener. Uno de los primeros vehículos categorizados con este nivel 3 y comercializable fue el vehículo de Honda con su sistema llamado *Traffic Jam Pilot*. Los vehículos de nivel 4 se dicen ser comercializables, pero por lo general no se perciben como vehículos que te den la seguridad de olvidarte por completo de la conducción de este. Por lo que, no se comercializan ni siquiera a una escala minoritaria de producción. Siendo Tesla una de las marcas de vehículos autónomos que está más cerca de lograr este nivel de autonomía. Obviamente, el nivel 5 es el objetivo que se debiese conseguir para decirse que, efectivamente, se solucionó el problema de completa autonomía. Con ello, incluso se podrían repensar los espacios dentro del vehículo, quitando los elementos esenciales para la conducción de hoy en día como lo son el freno, acelerador y manubrio.

Independiente del avance en la autonomía que se posea a día de hoy, si se tiene claro cuáles serías las ventajas y que problemas estos solucionarían. Entre las ventajas que estos poseen estarían una mayor comodidad y mejor experiencia de manejo, debido a que el conductor no estaría obligado a estar detrás del manubrio o, es más, no habría manubrio, pudiendo el conductor preocuparse de otras actividades más urgentes [3][4][5]. Por otra parte, se percibiría como una tecnología que permitiría el uso de vehículos con menos conocimiento técnico, es decir, la cantidad de esfuerzo para usar este tipo de vehículos sería menor que en los vehículos tradicionales [3][6]. También, se ha dicho que estos vehículos ayudarán a disminuir la cantidad de emisiones de gases invernaderos que el sistema de transporte produce hoy en día, en donde, el principal argumento a favor de aquello corresponde a la mejor optimización en los tiempos de viaje y distancia recorrida [9]. Aunque hay otros estudios que concluyen que no hay indicios de que aquello va a ser así con la tecnología actual, sino que la tecnología que se vaya introduciendo a estos sistemas permitirá, o no, la disminución en la emisión de gases invernaderos [7].

1.1.2. Vehículos Autónomos

La diferencia entre vehículos autónomos y los vehículos tradicionales no es menor y no es para menos, ya que se necesita todo un sistema que permita la toma de decisiones por parte de estos sistemas. A continuación, se enumeran las tareas más importantes que se deben cumplir dentro de un *pipeline* de conducción autónoma:

- **Sensar**: A partir de los sensores que tiene el vehículo se obtiene la mayor cantidad de información del ambiente.
- **Percibir y localizar**: Dada la información por parte de los sensores se realizan varias sub-tareas para entender el ambiente que rodea al vehículo autónomo y ubicar a este en el mapa. Por ejemplo, dada información de las cámaras frontales, se podría efectuar detección de líneas, objetos, segmentación semántica. Además, y en apoyo a otros sensores, se puede hacer SLAM o mapas de alta definición (HD Maps).
- **Representar escena**: Dado este entendimiento anterior, se puede juntar la información de varios sensores y realizar una predicción de los comportamientos de los diferentes elementos en la ruta. También se pueden obtener mapas de objetos que modelen y grafiquen el entorno del vehículo.
- **Planificar y Decidir**: Dada la información de un plan de ruta, se debe generar comandos de control para que el agente navegue por ciertas rutas definidas.
- **Controlar**: Dado los comandos de control de la etapa anterior, se deben ejecutar tales comandos mediantes controladores, tales como PID, MPC o de algún otro tipo, que permitan que el vehículo siga, de la forma más precisa posible, el movimiento especificado.

Cada una de estas tareas asegura que el vehículo pueda conducirse de manera segura desde un punto A hasta un punto B en un ambiente específico. Este *pipeline* es tradicional dentro del mundo de los vehículos autónomos y se ha usado desde el principio. Aunque la llegada del *Deep Learning* ha cambiado este *pipeline* y ahora estos sistemas de aprendizaje automático apuntan a aprender la conducción autonoma en una sola etapa; interactuando directamente con la entrada de los sensores y obteniendo los parámetros de control de una sola vez. Aunque esto no es necesariamente así, ya que *Deep Learning* aún es una tecnología reciente y para asegurar que este tipo de vehículos sean comercializables se sigue utilizando el *pipeline* de más arriba, pero remplazado las soluciones de algunas de esas etapas de técnicas más clásicas con estas nuevas tecnologías de procesamiento de datos.

1.1.3. Seguridad y evasión de colisiones

Dentro del mundo de los vehículos autónomos existen una variedad no menor de tópicos de investigación como por ejemplo detecciones de objetos, ciberseguridad, ética en la toma de decisiones, desarrollo de hardware, etc. Aunque uno de los tópicos más investigados es seguridad y evasión de colisiones. Esto debido a las implicancias y riesgos que tiene para la implementación de estos sistemas, pudiendo ser la mayor preocupación por parte de futuros usuarios. Para ejemplificar este punto, se puede citar el paper [11] en donde se recopila información sobre los sentimientos y percepciones de personas que viajan en bus y vehículo autónomo en Finlandia. Un hecho interesante que se dio en el estudio fue que las personas, en su mayoría, mencionan a seguridad y confianza como las características más importantes que

debe poseer un sistema vehicular autónomo. Lo interesante de aquello, es que las preguntas realizadas a los participantes no tenían como intención que hablasen de seguridad y confianza en el sistema. Concluyendo que estas dos características nombradas son las más importantes para los usuarios y que no usarían este tipo de transporte si se presentase, al menos, un error.

1.2. Hipótesis

En un contexto de aprendizaje reforzado para el aprendizaje de una política de conducción autónoma para la evasión de colisiones, una política de *Deep Reinforcement Learning* (DRL) que hace uso de aprendizaje por representación y demostraciones expertas, mejora el rendimiento del agente y permite asegurar un manejo seguro de situaciones riesgosas optimizando ciertas variables de rendimiento del agente, tales como radio de colisión y tasa de éxito. Por otra parte, el uso de aprendizaje por representación permite disminuir considerablemente los tiempos de entrenamiento, permitiendo una convergencia más rápida y así una mejor optimización, y mejores tiempos de cómputo.

1.3. Objetivos generales

El objetivo de este trabajo consiste en plantear, implementar y evaluar soluciones basadas en aprendizaje reforzado que hagan uso de demostraciones expertas y aprendizaje por representación para el problema de evasión de colisión para vehículos autónomos en un ambiente urbano.

1.4. Objetivos específicos

Con el propósito de cumplir la hipótesis enunciada y el objetivo general planteado de este trabajo, se deberán considerar los siguientes objetivos específicos:

- Implementación de un algoritmo *end-to-end* de DRL para la evasión de exo-agentes en ruta. Además de ser fácilmente escalable a situaciones que impliquen una mayor cantidad de exo-agentes.
- Evaluar la robustez del controlador del agente en distintos escenarios desafiantes y altamente complejos para la evasión de colisión.
- Demostrar la utilidad y conveniencia del uso de aprendizaje por representación para una convergencia más rápida en el aprendizaje de una política de conducción.
- Demostrar la utilidad de aquellos métodos que hacen uso de demostraciones expertas para una mayor convergencia y estabilidad en el aprendizaje del controlador neuronal. Obteniendo resultados significativamente mejor que aquellos que no emplean demostraciones expertas.

1.5. Estructura de la Tesis

La presente tesis se estructura de la siguiente forma:

- En el capitulo 2 se presenta el marco teórico que introduce los conocimientos necesarios para la comprensión de los siguientes capítulos. Específicamente, se presenta una base

conceptual para el entendimiento de los algoritmos de aprendizaje reforzado y aprendizaje reforzado profundo y su aplicación en el campo de los vehículos autónomos.

- En el capitulo 3 se presenta la metodología de la tesis, en donde se formula el problema a abordar y la delineación de la solución propuesta. Además de presentar los experimentos a realizar para comprobar la hipótesis y cumplir los objetivos planteados.
- En el capitulo 4 se presenta los resultados obtenidos de los distintos experimentos realizados para la validación de la hipótesis. Así como también un análisis a cada uno de estos para entender el aporte de los resultados a la conclusión redactada.
- Finalmente, en el capitulo 5 se presentan las conclusiones conseguidas en base al trabajo realizado, y las diferentes alternativas futuras que podrían extender y mejorar el trabajo presentado.

2. Marco teórico

2.1. Machine Learning

2.1.1. Introducción

Hay varias definiciones de lo que es *machine learning* (ML) que se ocupan en la literatura, una más precisa que otra, pero sin duda la más acertada y descriptiva es la enunciada por Tom Mitchell en su libro *Machine Learning* [12][13] y que dice:

«Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T, y medición de desempeño P, si su desempeño en las tareas en T, medido por P, mejora con la experiencia E.»

En palabras simples, *machine learning* (ML) no es más que el campo del conocimiento que se encarga de estudiar e implementar algoritmos que aprenden de datos para solucionar tareas que requiriesen de inteligencia humana. Por esta misma razón ha sido uno de los campos de las ciencias de la computación con más atención en los últimos años. Esto a causa de soluciones de problemas insignes en los más variados campos del conocimiento. Por ejemplo, en las ciencias biológicas existe un problema llamado *Protein Folding* [15] que se enunció hace más de 50 años, pero no pudo ser resuelto hasta recién el 2020 por unos investigadores del centro de investigación *DeepMind* los cuales propusieron una solución basada en Redes Neuronales Profundas denominada *AlphaFold* [16]. Otro ejemplo, pero ahora en *Natural Language Processing* (NLP), fue la creación del modelo *GPT3* [17], modelo de lenguaje autorregresivo que poseía 175 billones de parámetros y que fue entrenado bajo un paradigma de *Low-Shot Learning*. Es uno de los modelos más grandes creados para *Machine Learning* en la actualidad. Aunque lo más destacable de este modelo fueron los resultados que obtuvo en diferentes tareas de NLP. Por ejemplo, la posibilidad de generar artículos de noticias sintéticas que fuesen casi indistinguibles de aquellas noticias generadas por algún humano. Además de traducir documentos, superando al estado del arte en dos de las seis pruebas propuestas en el estudio.

Por otro lado, y por primera vez en la historia, se desarrolló un modelo de inteligencia artificial que derroto al campeón mundial de Go mediante algoritmos de *Reinforcement Learning* y redes neuronales. Modelo al cual denominaron *AlphaGo* [18]. También por el mismo laboratorio de investigación *DeepMind*.

Tal cual se dijo anteriormente, un modelo de *Machine learning* aprende de datos o ejemplos que se le entregan para solucionar una tarea específica. En este contexto, el aprendizaje tiene como objetivo la obtención de una función de mapeo o modelo $f(x;\theta) \approx y$. Donde $x \in \mathcal{X}$ son las entradas al modelo, $y \in \mathcal{Y}$ las salidas y θ los parámetros de la función o modelo. Por lo general, a las entradas x se les denomina características y a las salidas y se les conoce como etiquetas. Si la etiquetas son dadas, se convierte en un problema de aprendizaje supervisado, si no, se convierte en un problema no-supervisado. Más adelante se hablará más de este tipo

de aprendizaje. Siguiendo la notación de Tom Mitchell, la experiencia $E = \{(x_n, y_n)\}_{n=1}^{N}$ es un conjunto de datos representados por sus características y etiquetas para el entrenamiento del modelo. También existe otro conjunto de datos que lleva por nombre conjunto de prueba y se ocupa para evaluar la generalización de los datos. Es decir, este conjunto es exclusivo y no puede contener ningún dato que haya sido utilizado para el entrenamiento del modelo, o más específicamente, cumple con la siguiente identidad: $\{(x_m, y_m)\}_{m=1}^{M}, (x_m, y_m) \notin E$. La tarea T puede ser cualquier tarea de clasificación o regresión. Donde clasificación es cualquier tarea de predicción que involucre la predicción de valores categóricos o cualitativos. Mientras que regresión es cualquier tarea predictiva que implique la predicción de valores cuantitativos [19]. Siguiendo con la definición de *Machine Learning*, P es una métrica que evalúa el desempeño del modelo de manera cuantitativa. Para la tarea de clasificación y regresión se ocupa el riesgo empírico:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n; \theta)) \tag{2.1}$$

En donde ℓ es cualquier función de pérdida o costo. Las funciones de perdidas más ocupadas en ML son *Mean Squared Error* (MSE), *Cross Entropy* (CE), *Binary Cross Entropy* (BCE) y *Kullback-Leibler divergence* (KL). Las cuales aparecen en las ecuaciones 2.2, 2.3, 2.4 y 2.5, respectivamente.

$$MSE(y, \tilde{y}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - \tilde{y_n})^2 \tag{2.2}$$

$$CE(x, y) = \frac{1}{N} \sum_{n=1}^{N} - \sum_{c=1}^{C} y_c \log \frac{\exp(x_{n,c})}{\exp\left(\sum_{c=1}^{C} x_{n,c}\right)} \tag{2.3}$$

$$BCE(x, y) = \frac{1}{N} \sum_{n=1}^{N} -(y_n \log(x_n) + (1 - y_n) \log(1 - x_n)) \tag{2.4}$$

$$KL(x, y) = \frac{1}{N} \sum_{n=1}^{N} y_n (\log y_n - x_n) \tag{2.5}$$

MSE es, por lo general, la función de pérdida más ocupada en problemas de regresión. Mientras que las funciones de CE, BCE son utilizadas en problemas de clasificación. La primera en problemas de clasificación discreta multi-clases y la segunda en problemas de clasificación binaria. En cuanto a KL es útil en la medición de diferencias entre distribuciones discretas o continuas.

La forma de aprendizaje descrita anteriormente es la más usada actualmente, pero no es la única. Debido a la dificultad de conseguir datos con etiquetas, se han investigado otras formas de aprendizaje, tales como aprendizaje no supervisado, auto-supervisado y por refuerzo. La primera investiga maneras de aprendizaje para aprender lo mismo que aprendizaje supervisado, pero sin la necesidad de utilizar etiquetas. El ejemplo más típico de este tipo de aprendizaje es *clustering*. El segundo apunta a aprender representaciones subyacentes de los datos mediante alguna tarea supervisada, también sin la necesidad de etiquetas. Por ejemplo, aprendizaje por contraste (*Contrastive Learning*) forma parte de las técnicas más conocidas de esta forma de aprendizaje. Por último, aprendizaje por refuerzo o *Reinforcement Lear-*

ning (RL) tiene como objetivo el aprendizaje de una tarea o política de acciones mediante la interacción de un agente con su ambiente.

2.1.2. Redes Neuronales

Las Redes Neuronales son uno de los modelos más usados de ML en la actualidad. Estos modelos apuntan a emular el sistema neuronal humano, sin embargo, difieren en algunos aspectos, tal como en la existencia del algoritmo *backpropagation*. Cuyo algoritmo no existe en el cerebro humano debido a que no hay forma de enviar información de regreso a lo largo del axón. Otro aspecto en el cual difieren es en la complejidad de sus estructuras, en donde las redes neuronales artificiales consisten únicamente en pesos ponderables con funciones no-lineales, mientras que las redes neuronales biológicas tienen estructuras dendríticas complejas con dinámicas espacio-temporales [13].

Antes de explicar cualquier cosa sobre las redes neuronales se debe explicar la unidad básica de estas redes, el cual es el perceptrón. El perceptrón no es más que una función determinista de la forma $f(x;\theta) = \varphi(W^T x + b)$, donde $x \in \mathbb{R}^n$ es un vector de entrada, W son los parámetros o pesos del modelo, b es el sesgo o *bias* y φ es una función de activación. Las funciones de activación más conocidas se muestran a continuación:

- Sigmoide: $\delta(a) = \frac{1}{1+\exp^{-a}}$
- *Rectified Linear Unit*: ReLU(a) = max(a, 0)
- Softmax: $s(a_i) = \frac{\exp(a_i)}{\sum_j \exp(a_i)}$

La utilidad de las funciones de activación es, primeramente, hacer que el perceptrón sea diferenciable y por ende ajustable a datos de entrenamiento. Pero, también tiene una utilidad propia de cada una, en donde, por ejemplo, las funciones sigmoide y *softmax* son funciones que típicamente se ocupan en problemas de clasificación debido a sus capacidades de mapear el valor entrante entre 0 y 1.

Cuando se habla de redes neuronales artificiales se agrupan distintos modelos neuronales en el mismo término. El tipo de red neuronal más conocido es el *multilayer perceptron* (MLP) que tiene una estructura de grafo acíclico dirigido con pesos en las aristas del grafo. Y tal como el nombre lo anticipa, son estructuras que están compuestas por múltiples perceptrones ligados uno a los otros. Este tipo de red neuronal asume vectores de entradas de dimensiones fijas, cuyos datos son denominados datos estructurados o tabulares. Un ejemplo de este tipo de redes neuronales, se presenta en la figura 2.1 en donde se muestra un problema de clasificación de tipos de flores dada las características de cada una de ellas.

También existen otro tipo de redes neuronales que procesan datos como imágenes, texto o figuras geométricas, en donde redes neuronales como redes convoluciones, recurrentes, *transformers* y grafos neuronales vienen a tratar de solucionar problemas relativos a tales datos.

Estos modelos, al igual que todos, deben ajustarse a la distribución de los datos que tengan, por lo que este tipo de modelos deben hacer uso de técnicas de optimización para ello. Una de las técnicas más comunes utilizadas en redes neuronales para el ajuste de sus pesos es gradiente descendiente. Gradiente descendiente es una técnica de optimización iterativa de

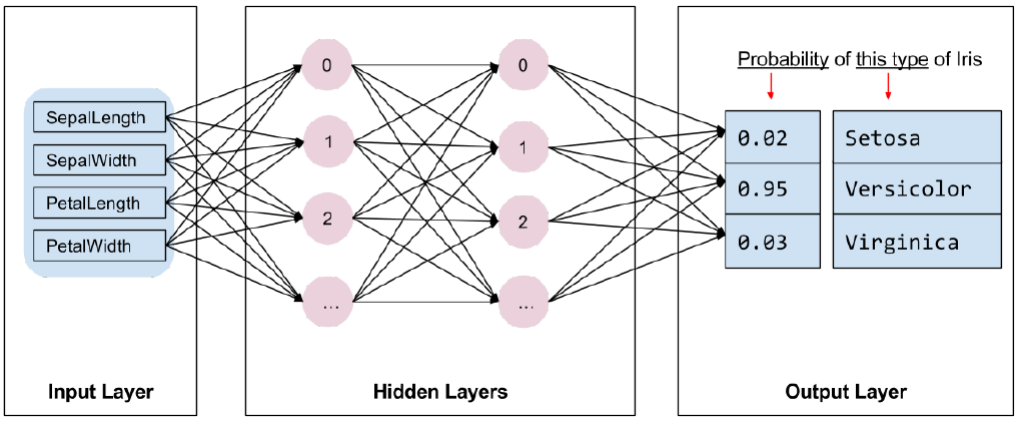


Figura 2.1: Red neuronal MLP con dos capas ocultas para la clasificación de un tipo de flores iris. Figura tomada desde [13].

primer orden para la obtención de un mínimo local. Dado los pesos actuales de la red neuronal más el gradiente de la función de pérdida con respecto a los mismos pesos, se actualizan estos mismos mediante la siguiente regla de actualización:

$$w_{ij} \leftarrow w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}} \tag{2.6}$$

Es decir, se actualizan los pesos de la red para que en cada paso de actualización se minimice aún más la función de pérdida especificada. Tomando así, la dirección de actualización que posibilite lo anterior. La rapidez para alcanzar el mínimo va a estar restringido al radio de aprendizaje α, el cual determina cuanto se actualizan los pesos con respecto a la dirección de actualización $\frac{\partial L}{\partial w_{ij}}$.

Una de las propiedades de estas redes neuronales que las hacen ser uno de los modelos más apetecidos de hoy en día es la del cumplimiento del teorema de aproximación universal [20]. Este teorema asegura que para cualquier función, existe, a lo menos, una configuración de red neuronal que la parametriza. Por lo que en teoría, este tipo de modelos podrían solucionar cualquier problema que necesitase de alguna función parametrizable.

2.1.3. Deep Learning

Gracias a las redes neuronales, se puso en boga el término *Deep Learning* (DL). Este término nació a partir de la construcción de nuevas redes neuronales MLP con múltiples capas ocultas [21] y que posibilitaron la solución de variados problemas complejos. DL es una sub-disciplina de ML que elimina la etapa de *feature engineering* que tienen los sistemas tradicionales de ML. Sin embargo, tal etapa no desaparece, sino que las redes neuronales aprenden una representación de los datos de manera inherente. A pesar de que las redes neuronales MLP son una de las primeras arquitecturas creadas, no son las únicas. También existen otros tipos de arquitecturas que procesan diferentes tipos de datos y que presentan diferentes objetivos.

2.1.3.1. Redes Neuronales Convolucionales

Las **Redes neuronales Convolucionales** (*Convolutional Neural Networks* (CNN)) tienen como objetivo el procesamiento de datos proveniente de imágenes, señales o figuras

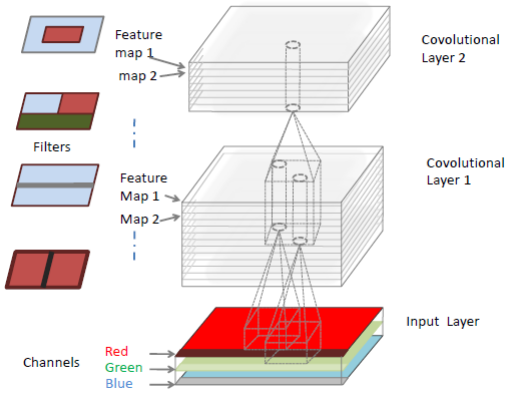


Figura 2.2: Ilustración que ejemplifica la operación convolucional por parte de una red neuronal convolutiva. Imagen tomada dese la figura 14.8 del libro [13]

geométricas. Para ello, basan su operación sobre la operación convolución, en vez, de la multiplicación matricial de las redes neuronales normales. La elección de esta operación para el procesamiento de imágenes, por parte de este tipo de redes neuronales, no fue arbitraria, ya que este tipo de operación se ha utilizado por varios años para el procesamiento de imágenes y señales. La aplicación de esta operación matemática permite la identificación de ciertas relaciones espaciales en la imagen o señal mediante el deslizamiento de un filtro. Históricamente, este filtro ha sido calculado a mano para alguna tarea específica; sin embargo, y por el modelo de aprendizaje que poseen estas redes, estos filtros se aprenden iterativamente a partir de los datos proporcionados.

Un ejemplo de como una red convolucional procesa una imagen, se muestra en la figura 2.2. En esta imagen se muestra una red neuronal convolucional de dos capas que procesa una imagen que tiene 3 canales. Los cilindros corresponden a hipercolumnas que son los vectores de característica que son resultado de la operación convolucional entre el mapa de característica $(i-1)$ y la red convolucional. La operación convolutiva mencionada se muestra en la ecuación 2.7. En ella se puede verificar como un filtro w convoluciona con una zona delimitada por $(si+u, sj+v)$ de la imagen para el canal c.

$$z_{i,j,d} = b_d + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{c=0}^{C-1} w_{u,v,c,d} x_{si+u,sj+v,c} \tag{2.7}$$

El *pipeline* clásico de una red convolucional usada para la clasificación de imágenes se muestra en la imagen 2.3. En esta figura se puede ver como se hace alusión a un tipo de operación denominado *Max Pooling*. Esta operación contempla la aplicación de tomar el máximo de una región local del mapa de características. Esto se hace para mantener invariante la posición de las características en las imágenes. Características que son útiles para la clasificación y detección de imágenes. *Max Pooling* no es la única operación usada en CNN, sino que también existe *Average Pooling* que realiza el promedio de los valores del mapa de características y *Global Average Pooling* que promedia todos los valores del mapa, para la obtención de un vector de características. Este puede ser utilizado para clasificación u algún otro motivo. En la misma imagen se muestra que el resultado de la red convolucional es

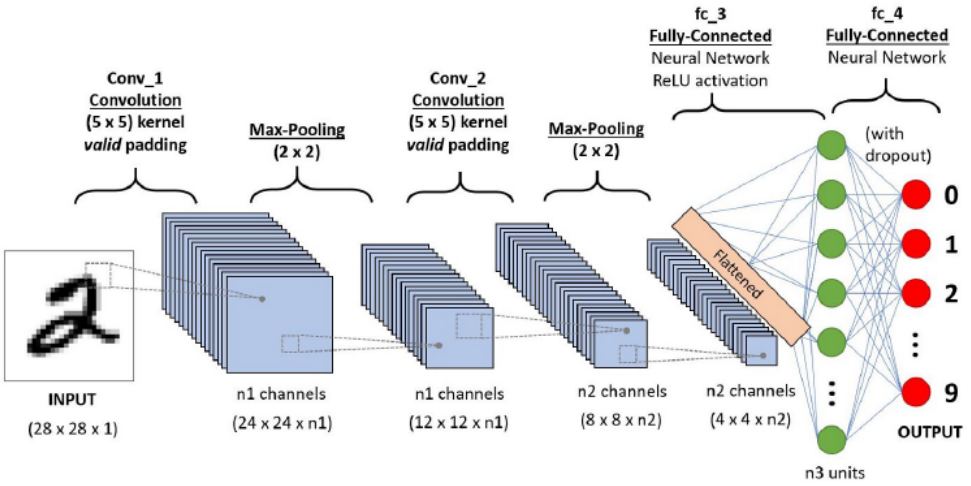


Figura 2.3: Red neuronal convolucional usada para la clasificación de una imagen numérica. Imagen tomada dese la figura 14.8 del libro [13]

entrada de una red neuronal MLP que se encarga de clasificar la imagen.

2.1.3.2. Autoencoders

Los ***AutoEncoders*** (AE) son redes neuronales que buscan aprender una representación de los datos, típicamente, para reducción dimensional. Aunque no es lo único para lo que se ocupan, ya que se han utilizado también para reconocimiento facial, detección de características, detección de anomalías y como modelos generativos. Existen diferentes tipos tales como *Sparse AutoEncoder*, *Denoising AutoEncoder* y *Variational AutoEncoder*.

Matemáticamente, los AE tienen como objetivo aprender una función parametrizable f: $z = f(x)$, en donde $x \in \mathbb{R}^n$ es un vector de entrada y la función f es denominada como *Encoder*. Generalmente, se tiene que el vector latente $z \in \mathbb{R}^m$ cumple con $m \leq n$, es decir, es un vector de menor o igual dimensión que la entrada a la red neuronal. Además, se intenta aprender una función parametrizable de reconstrucción g: $r = g(z)$ que busca reconstruir la entrada del *Encoder* y cumplir que la reconstrucción sea igual a la entrada, es decir, que se cumpla: $r = g(f(x))$ o que es lo mismo que $r = x$. La función g es denominada *Decoder*.

Para entender de mejor forma la arquitectura del AE considérese la figura 2.4, en donde se procesa una imagen para obtener una reconstrucción de la misma. En este ejemplo, tanto el *Encoder* y el *Decoder* son redes neuronales convolucionales las cuales tienen el objetivo de aprender una representación de baja dimensionalidad de la imagen de entrada, en el caso del *Encoder*, y de aprender a reconstruir la imagen numérica en base de un vector latente bajo-dimensional, por parte del *Decoder*. En la práctica, para el entrenamiento de esta red neuronal se ocupó el error de reconstrucción entre la imagen de entrada y la reconstrucción de esta misma. Esta función de costo se muestra en la ecuación 2.8 y no es más que una función MSE que cuantifica la diferencia entre ambas imágenes, para así conseguir que ambas sean lo más parecidas posible.

$$\mathcal{L}_r = MSE(x, g(f(x))) \tag{2.8}$$

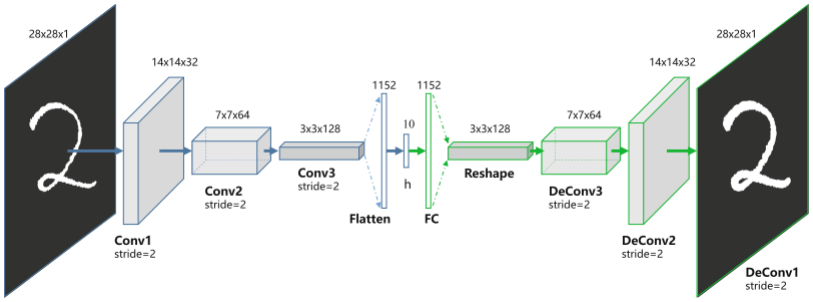


Figura 2.4: Procesamiento de una imagen por parte de una arquitectura neuronal *AutoEncoder*. Figura tomada desde [23]

2.1.3.3. Variational Autoencoder

Como se dijo anteriormente, *Variational AutoEncoder* (VAE) es un tipo de AE, sin embargo, este no es determinista, sino que estocástico. De hecho, uno de los problemas en el cual incurre los AE es la no-continuidad del espacio latente aprendido por esta arquitectura. Haciendo más difícil la clasificación de ejemplos mediante el uso de vectores latentes, debido a la dificultad en la interpolación del espacio latente. Esto también perjudica directamente a tareas de generación que podrían realizarse con esta arquitectura. Para solucionar aquello, se introdujo una etapa de muestreo gausiano entre el *Encoder* y el *Decoder*. Esta etapa cuenta con dos capas neuronales que producen dos vectores latentes distintos, μ, Σ, los cuales representan la media y desviación estándar de una distribución gaussiana multivariada, permitiendo que el espacio latente sea continuo y por ende interpolable. Además de permitir que las características generadas sean más disgregadas entre ellas. Con esto en mente, ahora el vector latente es producido mediante $z \sim \mathcal{N}(\mu, \sigma)$. Aunque si bien, se consiguieron subsanar algunos problemas que presentan los AE mediante la adición de este paso intermedio, esto introduce otro problema, que es el entrenamiento de la red neuronal. El problema es que dada la configuración con estas dos nuevas capas neuronales no se puede calcular el algoritmo de *backpropagation* analíticamente, y por ende, tampoco computacionalmente. Para solucionar aquello, se debe implementar un método de reparametrización llamado *reparametrization trick*, el cual consiste en agregar una variable aleatoria $\epsilon \sim \mathcal{N}(0, I)$, transformando así el muestreo de la variable latente a: $z = \mu + \sigma * \epsilon$. Esto permite, que el modelo pueda hacer uso del algoritmo *backpropagation* para aprender a realizar la tarea de turno. La función de costo que debe optimizar la VAE se muestra en la ecuación 2.9.

$$\mathcal{L}_{VAE} = MSE(x, x') + KL(\mathcal{N}(\mu, \sigma), \mathcal{N}(0, I)) \tag{2.9}$$

El termino de la derecha de la ecuación 2.9 se usa para mantener la distribución gaussiana cercana a una distribución normal con una media en cero y desviación estándar unitaria. Esto permite que el modelo se mantenga simple y robusto.

En la figura 2.5 se muestra una red neuronal VAE que recibe una imagen y obtiene su reconstrucción. A diferencia de la figura 2.4, se puede verificar las capas neuronales intermedias descritas anteriormente.

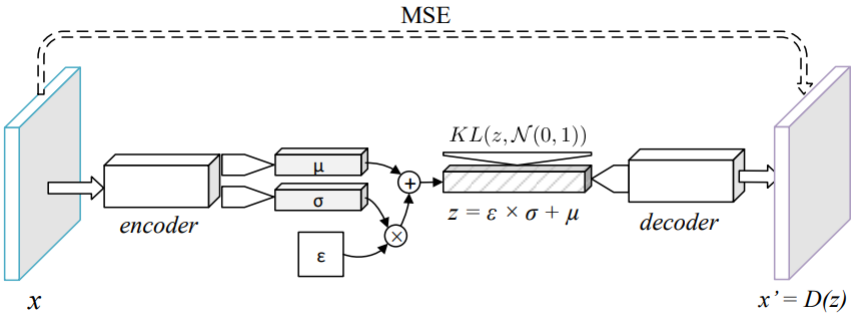


Figura 2.5: Procesamiento de una imagen por parte de una arquitectura neuronal VAE. Figura tomada desde [24]

2.2. Reinforcement Learning

La definición de *Reinforcement Learning* dada en el libro de Sutton y Barto [25] dice:

«Reinforcement Learning es aprender que hacer, como mapear situaciones a acciones, para la maximización de una señal de recompensa numérica.»

Para entender la definición anterior hace falta dar algo más de contexto de los elementos que rodean este tipo de aprendizaje. Primero, para que RL tome lugar debe existir dos elementos fundamentales: un **agente** y un **ambiente**. Este agente puede ser cualquier cosa que represente un sistema que tome acciones con base en estímulos de algún tipo que percibe desde el ambiente. Por ejemplo, podría ser un vehículo autónomo que percibe la realidad a través de una cámara frontal y conduce basándose en ello. Podría ser también, un sistema de predicción de ruteo vehicular que tome decisiones con respecto al punto inicial y final de los vehículos, congestión de la ruta, energía y puntos de carga, etc. [26]. En problemas de RL que sean multi-agentes, es decir, que contengan más de un agente en el ambiente se hace la distinción entre ego-agente y exo-agente. Ego-agente son aquellos agentes para los cuales se desea obtener una política de decisión. Mientras que exo-agente son aquellos agentes que están presentes en el ambiente, pero no necesariamente se busca optimizar su toma de decisiones, pudiendo tener políticas de decisiones completamente aleatorias. Aunque si es conveniente optimizar su política de decisiones en aquellos problemas que buscan un entrenamiento adversario con respecto al ego-agente.

Aparte de estos dos elementos relevantes, también se pueden considerar otros sub elementos que son importantes. Estos son los siguientes:

- **Política**: define como el agente se comportará en un ambiente para un tiempo dado. Es decir, la política del agente se encarga de dictar una acción a tomar por parte de este en base al estado del ambiente que percibe. Es justamente este elemento el cual debe ser aprendido para que se puedan tomar las decisiones correctas y lograr lo que se quiere con el agente.
- **Señal de recompensa**: Cuantifica el objetivo del problema. Es decir, el agente al tomar una acción y dar un paso en el ambiente, recibe una recompensa que le dice a este que tan correcto o incorrecto fue la acción que ejecuto en el estado actual.

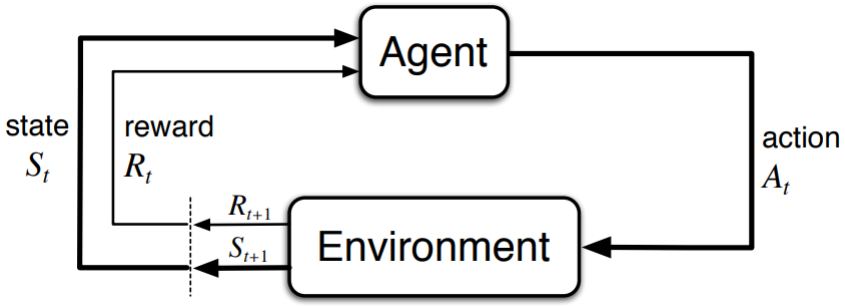


Figura 2.6: Interacción agente-ambiente en un proceso de Markov. Figura tomada desde [25]

- **Función de valor**: Cuantifica que es bueno a largo plazo, es decir, el valor en un estado específico es el total de recompensa que puede esperar obtener el agente a partir del estado actual siguiendo la política actual.

Hay un sub elemento más, pero no está presente en todos los problemas de RL, sino que solo en alguno de ellos. Este sub elemento es el modelo del ambiente. Este modelo replica el comportamiento del ambiente, permitiendo que se puedan hacer inferencias de como el ambiente se comportará. Los métodos que utilizan tal modelo se denominan **métodos basados en modelo** (*model-based methods*), mientras que los que no hacen uso del modelo del ambiente, se denominan **métodos libres de modelo** (*model-free methods*).

Históricamente, los procesos que hacen uso de agentes, ambientes y recompensas son conocidos como procesos de decisiones de Markov (*Markov Decision Process* (MDP)). Estos procesos formalizan el aprendizaje y resolución de un problema de toma de decisión secuencial. Estos se modelan mediante la tupla ($\mathcal{S}$, $\mathcal{A}$, P, $\mathcal{T}$, r):

- $\mathcal{S}$ son el conjunto de los estados en la MDP.
- $\mathcal{A}$ es el conjunto de acciones en la MDP.
- $\mathcal{T}(s'|s,a)$ es la probabilidad de transición de estados. Entrega la probabilidad de transicionar entre un s $\in \mathcal{S}$ a s' $\in \mathcal{S}$ tomando una acción a $\in \mathcal{A}$.
- $r(s,a)$ es la señal de recompensa que se obtiene por tomar la acción a $\in \mathcal{A}$ en el estado s $\in \mathcal{S}$.

En una MDP el agente y el ambiente interactúan durante un tiempo discreto. Esto quiere decir que para cada paso de tiempo t, el agente recibe una representación del estado del ambiente $s_t \in \mathcal{S}$ y en base a ello el agente toma una acción $a_t \in \mathcal{A}$. En consecuencia, el agente recibe una recompensa r_t de acuerdo a $r(s,a)$ y el ambiente evoluciona a $s_{t+1} \in \mathcal{S}$. El periodo abarcado entre $t = 0$ hasta el alcance de un estado terminal se denomina **episodio**. En la figura 2.6 se muestra gráficamente esta interacción agente-ambiente

Si se sigue la formulación anterior, se puede apreciar que cada posible valor de s_t y r_t dependen únicamente del estado y las acción precedente (s_{t-1}, a_{t-1}). Es decir, el estado contiene información sobre todos los aspectos del pasado entre la interacción agente-ambiente que causan una diferencia hacia el futuro. Esta propiedad es conocida como propiedad de

Markov [25]. El objetivo de este tipo de problemas consiste en enseñar al agente para que acumule la mayor cantidad de recompensa posible en su estadía en el ambiente. Formalmente, se busca maximizar la recompensa obtenida por parte del agente:

$$R_t = \sum_{t=t}^{T} r_t \tag{2.10}$$

En donde T es el paso final y puede ser finito o infinito, lo cual depende exclusivamente del problema a resolver. En caso que sea un problema de horizonte finito se cumple que $T < \infty$ y que lo estados finales del ambiente están definidos. Este tipo de interacción agente-ambiente llevan por nombre **interacción episódica**. Por otro lado, si se cumple que $T = \infty$, la interacción con el ambiente no tiene estados terminales, sin embargo de todas formas se debe optimizar el problema. El problema en este tipo de interacciones es que el retorno esperado en la ecuación 2.10 diverja. Para que no ocurra aquello, se introduce un factor de descuento $\gamma \in [0, 1]$:

$$R_t = \sum_{k=t}^{T} \gamma^t r_t \tag{2.11}$$

Este factor de descuento determina el valor de las presentes recompensas [25]. Si $\gamma < 1$, se elimina el problema de la divergencia que tenía la ecuación 2.10, sin embargo, si se quiere volver a esta última ecuación, solo basta con especificar $\gamma = 1$. Si $\gamma = 0$, se dice que el agente es miope y solamente le importa maximizar la recompensa inmediata. Por otro lado, si γ es cercano a uno, se dice que el agente es hipermétrope, es decir, presta más atención a comportamientos que maximicen la recompensa futura.

Una distinción necesaria a realizar en procesos MDP consiste en distinguir si el proceso que se sigue es completamente observable o parcialmente observable. Si el ambiente es completamente observable, el agente puede observar el estado completo del ambiente en cualquier momento. Sin embargo, si el ambiente es parcialmente observable, únicamente se posee un subconjunto de estados de manera conocida. Además, que estos estados pueden ser una representación incompleta y ruidosa. Este tipo de MDP's llevan por nombre procesos de decisiones de Markov parcialmente observables (*Partially Observable Markov Decision Process* (POMDP)). Este tipo de procesos son lo que comúnmente se dan en la realidad, en donde, por lo general, se ocupan sensores para capturar una representación incompleta de la realidad.

Para formalizar la obtención de una política óptima, se debe definir matemáticamente el término de función de valor (*Value Function*). La función de valor de un estado s bajo una política π, denotada por $V_\pi(s)$, se define en la ecuación 2.12.

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=t}^{T} \gamma^t r_t \middle| S_t = s \right], \forall s \in \mathcal{S} \tag{2.12}$$

Esta función es denominada también como **función de valor estado** para una política π (*state-value function*). También es necesario definir la **función de valor del par estado-acción** para una política π (*action-value function*), denotada por $Q_\pi(s, a)$, como el valor esperado empezando desde el estado s y ejecutando la acción a, para continuar bajo una política π:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=t}^{T} \gamma^t r_t \middle| S_t = s, A_t = a \right] \tag{2.13}$$

Dicho valor Q representa cuan útil es una acción para la obtención de recompensa en pasos posteriores.

Por definición, ambas funciones tienen similitudes, por lo que cada una de estas puede ser escrita por su par. En la ecuación 2.14 se muestra la función de valor estado escrita con respecto a la función de valor del par estado-acción. En la ecuación 2.15 se muestra la función de valor del par estado-acción escrita con respecto a la función de valor estado.

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[Q_\pi(s,a)] \tag{2.14}$$

$$Q_\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} V_\pi(s') \tag{2.15}$$

De las ecuaciones 2.14 y 2.15 se puede obtener la ecuación de *Bellman* para la función de valor:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} V_\pi(s')] \tag{2.16}$$

Del mismo modo, se puede obtener la ecuación de *Bellman* para la función de valor del par estado-acción:

$$Q_\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} \mathbb{E}_{a\sim\pi(a|s)}[Q_\pi(s,a)] \tag{2.17}$$

Estas ecuaciones de *Bellman* expresan una relación entre el valor del estado y el valor del estado sucesor. La función de valor $V_\pi(s)$ define un orden parcial sobre las políticas [25]. Esto significa que una política π es mejor o igual que otra política π' si se cumple que $V_\pi(s) \geq V'_\pi(s)$. La política óptima cumple con la maximización de la función de valor $V_\pi(s)$:

$$V^*(s) = \max_\pi V_\pi(s) \tag{2.18}$$

También cumple con la maximización de la función de valor par estado-acción $Q_\pi(s,a)$:

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{2.19}$$

2.2.1. Algoritmos

En esta sección se describen los algoritmos más clásicos de RL. Por lo general, los algoritmos de RL se dividen en dos categorías: Algoritmos de optimización de política (*Policy Optimization*) y algoritmos de programación dinámica (*Dynamic Programming*) (DP). Los primeros se caracterizan por el aprendizaje de una política parametrizada que no necesita de una función de valor para la toma de una acción. Mientras, que los segundos se caracterizan por la optimización de una política mediante funciones de valor. A diferencia de los primeros, los algoritmos de DP depende principalmente de funciones de valor, incluso para la toma de acciones.

2.2.1.1. Policy Iteration

Policy Iteration tiene como objetivo encontrar una política de acción que maximice la recompensa obtenida. Para lograr aquel objetivo hace uso de dos fases iterativas:

- ***Policy Evaluation***: Esta fase tiene el objetivo de evaluar una política π mediante la obtención de su función de valor estado v_π de forma iterativa. Se empieza con una aproximación inicial v_0, elegida de forma arbitraria, aproximándose en cada iteración con la siguiente regla de actualización:

$$v_{k+1} = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] \tag{2.20}$$

  De esta manera, la política v_k converge a v_π a medida que $k \to \infty$. Se debe mencionar que la función de valor estado v_k(s) para un estado s remplaza su valor para cada iteración sucesiva.

- ***Policy Improvement***: En esta fase se debe actualizar la política con respecto a la función de valor estado v_π obtenida en la fase anterior. Para ello se debe utilizar la desigualdad $v_{\pi'}(s) \geq v_\pi(s)$ para verificar que la política obtenida π' es mejor que la política anterior π. Para la actualización de la política se debe considerar la acción a producida por la política actual para un estado s: $a \leftarrow \pi(s)$ y comparar esta con la acción obtenida desde $\text{argmin}_a \sum_{s,r'} p(s', r|s, a) \left[r + \gamma V(s')\right]$. Luego, si la acción anterior a es la misma acción obtenida de la nueva política $\pi(s)$, se declara a esta última como una política estable y se termina la iteración. En caso contrario, vuelve a la fase anterior y repite el ciclo hasta converger.

Estas dos fases iterativas se alternan entre ellas continuamente para mejorar la política del agente. En el algoritmo 1 se muestra el algoritmo *Policy Iteration* escrito de manera algorítmica.

2.2.1.2. Value Iteration

Value Iteration es otro algoritmo de programación dinámica, el cual viene a tratar algunos inconvenientes de *Policy Iteration*. El principal inconveniente de *Policy Iteration* es la constante evaluación de la política para cada iteración del algoritmo. Lo cual aumenta los tiempos de cómputo del algoritmo. Para resolver este problema se puede truncar *Policy iteration* de diferentes formas, pero la más ocupada considera detener la evaluación de la política después de una de iteración. Esto da paso a la siguiente regla de actualización:

$$v_{k+1} = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] \tag{2.21}$$

Esta regla de actualización es muy parecida a la regla de actualización de *Policy Iteration*, únicamente que en este algoritmo toma el máximo sobre todas las acciones posibles para un estado $s \in \mathcal{S}$.

Al igual que en el algoritmo de *Policy Iteration* el valor óptimo de la función de valor estado v_* requiere un valor infinito de pasos. Aunque en la práctica esto es insostenible, por lo que se opta por terminar el algoritmo una vez que se haya alcanzado un valor de cambio insignificante.

Algorithm 1 Policy Iteration

1: Inicialización de la función de valor $V(s) \in \mathbb{R}$
2: Inicialización de la política $\pi(s) \in \mathcal{A}(s)$ arbitrariamente para todos los estados $s \in \mathcal{S}$
3: # Policy Evaluation
4: $\Delta \leftarrow 0$
5: **while** $\Delta < \theta$ **do**
6: **for** cada s en $\mathcal{S}$ **do**
7: $v \leftarrow V(s)$
8: $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
9: $\Delta \leftarrow max(\Delta, |v - V(s)|)$
10: **end for**
11: **end while**
12: # Policy Improvement
13: politica-estable $\leftarrow$ Verdadero
14: **for** cada s en $\mathcal{S}$ **do**
15: Accion-antigua $\leftarrow \pi(s)$
16: $\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
17: **if** Accion-antigua $\neq \pi(s)$ **then**
18: politica-estable $\leftarrow$ Falso
19: **end if**
20: **end for**
21: **if** politica-estable **then**
22: Retornar $V \approx v_*$ y $\pi \approx \pi_*$
23: **else**
24: Volver a linea 4
25: **end if**

2.2.1.3. Q-learning

Q-learning es un algoritmo de programación dinámica y libre de modelo que busca aprender la función de valor del par estado-acción Q. Este algoritmo se define mediante la siguiente ecuación:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.22}$$

La ecuación 2.22 posee la misma forma que los algoritmos de diferencia temporal, de ahí que *Q-learning* sea denominado como un algoritmo de diferencia temporal a un paso. En consecuencia, hace uso de programación dinámica para su solución. Una parte importante de la ecuación 2.22 es la que está encerrada entre paréntesis cuadrados, la cual representa el error de diferencia temporal entre el estado s a s' y que se denota por $\lambda_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$. Esta ecuación representa cuan diferente es la estimación actual de la función de valor del par estado-acción Q, con respecto, a lo que debiese darse a partir de lo aprendido sobre la recompensa que se recibirá y el siguiente estado a visitar.

La característica principal de *Q-learning* es su estatus como un algoritmo *off-policy*, es decir, actualiza el valor de la función de valor del par estado-acción Q independiente de la política que sigue el agente. Esto último se verifica en la elección de a que maximiza la

Algorithm 2 Value Iteration

Inicializar $V(s) \forall\ s \in \mathcal{S}$ de manera arbitraria, exceptuando para $V(Terminal) = 0$
$\Delta \leftarrow 0$
while $\Delta < \theta$ **do**
    for cada s en $\mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end for
end while
Retornar una política determinista $\pi \approx \pi_*$ tal que
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

Algorithm 3 Q-Learning

Inicializar $Q(s, a) \forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$ de manera arbitraria, exceptuando para $Q(Terminal, \cdot) = 0$
for cada episodio **do**
    Inicializar S
    for cada s en $\mathcal{S}$ **do**
        Elegir a desde S usando la política derivada desde Q
        Tomar la acción a, observando r y s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
        $s \leftarrow s'$
        if s es terminal **then**
            break
        end if
    end for
end for

función de valor del par estado-acción Q. Es decir, dicha acción no sigue la política actual, sino que ocupa una política *greedy*. Una política de acción *greedy* es aquella que elige la acción que maximiza la función de valor del par estado-acción Q. Considerando, obviamente, que el agente tiene un componente de exploración, ya que en caso contrario, ambas políticas serían iguales. Este algoritmo convergerá a una solución óptima, asumiendo que después de la generación de muestras y entrenamiento, la política cambia a una política *greedy*.

2.2.1.4. SARSA

SARSA al igual que *Q-learning* es un algoritmo de programación dinámica que busca aprender la función de valor del par estado-acción Q en vez de la función de valor. A diferencia del algoritmo anterior, SARSA es un algoritmo *on-policy*, es decir, hace uso de su política actual de acciones para la actualización de la función de valor del par estado-acción Q. La regla de actualización para este algoritmo es la siguiente:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{2.23}$$

Esta regla de actualización se realiza después de cada iteración del algoritmo. Si verifican la ecuación 2.23 se puede verificar que la regla de actualización hace uso de la tupla

Algorithm 4 SARSA

Inicializar $Q(s,a) \forall\ s \in \mathcal{S}, a \in \mathcal{A}(s)$ de manera arbitraria, exceptuando para $Q(Terminal, \cdot) = 0$
for cada episodio **do**
    Inicializar S
    for cada s en $\mathcal{S}$ **do**
        Elegir a desde S usando la política derivada desde Q
        Tomar la acción a, observando r y s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s$
        $a \leftarrow a'$
        if s es terminal **then**
            break
        end if
    end for
end for

$(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, tupla que le da el nombre al algoritmo.

Este algoritmo convergerá a una solución óptima, asumiendo que la política utilizada para generar la experiencia es aquella que se utilizó para la actualización de la función de valor.

2.2.1.5. Policy Gradient

Otro acercamiento al mismo problema de optimización de política sería la de optimizar de manera directa la política de acción. Este acercamiento es denominado en la literatura de RL como búsqueda de política (*Policy Search*) y consiste en la búsqueda de una política en un espacio acotado de políticas. Para lograr aquello, se puede considerar los algoritmos de optimización basada en gradiente o libre de gradiente. En esta sección, se mencionará únicamente los algoritmos que utilizan optimización basada en gradiente.

En el contexto de RL los métodos que hacen uso de gradiente pertenecen a la categoría de *Policy Gradient*. La característica más relevante de estos métodos es que no hacen uso de una función de valor para la optimización de una política de acción. Aunque la función de valor todavía puede ser usada para la optimización de la política, pero no para la elección de la acción.

Matemáticamente, los métodos que caen bajo la categoría de *Policy Gradient*, siguen un esquema de actualización que se muestra a continuación:

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)} \tag{2.24}$$

Donde $\widehat{\nabla J(\theta_t)}$ es un estimador estocástico cuyas esperanzas aproximan el gradiente de la función de costo con respecto a los argumentos θ_t. La función 2.24 es una ecuación de gradiente ascendente. La función de costo $J(\theta)$ se puede definir de la siguiente manera:

$$J(\theta) = v_{\pi_\theta}(s_0) \tag{2.25}$$

Donde s_0 es el estado inicial del ambiente. Luego, utilizando la definición de la función de valor 2.12, derivando a cada lado de la ecuación y trabajando con las ecuaciones, se puede

llegar a la siguiente igualdad:

$$\nabla J(\theta) = \nabla \mathbb{E}_{a\sim\pi(a|s;\theta)}\left[g_t\right] \tag{2.26}$$
$$= \nabla \sum_a \pi(a|s;\theta) g_t \tag{2.27}$$
$$= \sum_a g_t \nabla \pi(a|s;\theta) \tag{2.28}$$
$$= \sum_a g_t \pi(a|s;\theta) \frac{\nabla \pi(a|s;\theta)}{\pi(a|s;\theta)} \tag{2.29}$$
$$= \sum_a g_t \pi(a|s;\theta) \log \pi(a|s;\theta) \tag{2.30}$$
$$= \mathbb{E}_{\pi(a|s;\theta)}\left[g_t \nabla \log \pi(a|s;\theta)\right] \tag{2.31}$$
$$\nabla \mathbb{E}_{a\sim\pi(a|s;\theta)}\left[g_t\right] = \mathbb{E}_{\pi(a|s;\theta)}\left[g_t \nabla \log \pi(a|s;\theta)\right] \tag{2.32}$$

Donde $g_t = \sum_{k=t}^{T} \gamma^k r_k$ es la recompensa descontada de la trayectoria del agente desde el paso t. Considerando un proceso estocástico que genera una trayectoria de tuplas (s, a, r):

$$\tau = (s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$$

La ecuación 2.32 se convierte en:

$$\nabla \mathbb{E}_\tau\left[g(\tau)\right] = \mathbb{E}_\tau\left[G(\tau) \nabla \log \pi(\tau|\theta)\right] \tag{2.33}$$

Para derivar el teorema de *Policy Gradient* se debe trabajar con la derivada parcial del lado derecho de la ecuación 2.33. Para ello considérese las siguientes identidades:

$$p(\tau|\theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t,\theta) p(s_{t+1}, r_t|s_t, a_t) \tag{2.34}$$
$$\log p(\tau|\theta) = \log \mu(s_0) + \log \prod_{t=0}^{T-1} \pi(a_t|s_t,\theta) p(s_{t+1}, r_t|s_t, a_t) \tag{2.35}$$
$$= \log \mu(s_0) + \sum_{t=0}^{T-1} \log\left[\pi(a_t|s_t,\theta) p(s_{t+1}, r_t|s_t, a_t)\right] \tag{2.36}$$
$$= \log \mu(s_0) + \sum_{t=0}^{T-1} \left[\log \pi(a_t|s_t,\theta) + \log p(s_{t+1}, r_t|s_t, a_t)\right] \tag{2.37}$$

Dada la siguiente identidad encontrada y juntándola junto a la ecuación 2.33 se puede obtener el teorema de *Policy Gradient*.

$$\nabla_\theta \mathbb{E}_\tau \left[g(\tau) \right] = \mathbb{E}_\tau \left[G(\tau) \nabla_\theta \left((\log \mu(s_0) + \sum_{t=0}^{T-1} \left[\log \pi(a_t|s_t, \theta) + \log p(s_{t+1}, r_t|s_t, a_t) \right] \right) \right] \tag{2.38}$$

$$= \mathbb{E}_\tau \left[G(\tau) \left((\nabla_\theta \log \mu(s_0) + \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t, \theta) + \nabla_\theta \sum_{t=0}^{T-1} \log p(s_{t+1}, r_t|s_t, a_t) \right) \right] \tag{2.39}$$

$$= \mathbb{E}_\tau \left[G(\tau) \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t, \theta) \right] \tag{2.40}$$

Junto a este teorema se puede derivar un primer algoritmo de *Policy Gradient*: **REINFORCE**. REINFORCE es un algoritmo de *Monte-Carlo Policy Gradient* que hace uso de la ecuación 2.41 como regla de actualización:

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \log \pi(a_t|s_t, \theta_t) \tag{2.41}$$

Lo interesante de la ecuación 2.41 es que tiene una interpretación intuitiva. Cada incremento de los parámetros del modelo es proporcional al retorno G_t y al vector dado por la probabilidad de tomar la acción a_t dividido por la probabilidad de tomar esa acción. La dirección de dicho vector en el espacio de parámetros es aquella que incremente la probabilidad de tomar una acción a_t repetidamente en futuras visitas al estado s_t. Esto último está ligado directamente al hecho que se favorecen configuraciones de parámetros que favorezcan la toma de decisiones que incrementen el retorno del agente.

Uno de los problemas que sufre *Policy Gradient* es la lenta convergencia que sufre su aprendizaje. Esto debido a la alta varianza que posee. Una forma de aminorar la varianza que sufre, consiste en agregar un *baseline* b a la ecuación 2.41, derivando en la siguiente regla de actualización:

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(s_t)) \nabla_\theta \log \pi(a_t|s_t, \theta) \tag{2.42}$$

Donde la ecuación 2.42 es una generalización de 2.41, ya que el *baseline* puede ser una función uniformemente nula. A pesar de que este baseline no tiene un impacto significativo en la esperanza, si lo puede tener en la varianza. La forma más simple de estimar la función *baseline* consiste en calcular la esperanza de los retornos sobre múltiples trayectorias:

$$b = \mathbb{E} \left[G(\tau) \right] \approx \frac{1}{N} \sum_{i=0}^{N} G(\tau^i) \tag{2.43}$$

Una de las ventajas que posee este algoritmo, con respecto a lo ya vistos, es la posibilidad de uso en problemas que contemplan espacios de acciones continuos, cuya característica es imposible en *Q-learning*, debido a su característica tabular.

2.2.1.6. Actor-Critic

Actor-Critic es un algoritmo similar a REINFORCE con *baseline*, diferenciándose principalmente en la adición de un modelo que aprende la función de valor. Si bien, REINFORCE también posee una función de valor a aprender, no se considera como un crítico, ya que este se ocupa únicamente como *baseline*. Es decir, no se ocupa para *bootstrapping*.

Algorithm 5 REINFORCE con Baseline

Entrada: Parametrización de una política diferenciable $\pi(a|s,\theta)$
Entrada: Parametrización de una función de valor estado diferenciable $\tilde{v}(s,w)$
Inicializar los parámetros $\theta \in \mathbb{R}^{d'}$ de la política
Inicializar los pesos $w \in \mathbb{R}$ de la función de valor
while True **do**
    Generar un episodio $S_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_T$ siguiendo $\pi(\cdot|\cdot,\theta)$
    for t=0:T-1 **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
        $\delta \leftarrow G - \hat{v}(s_t, w)$
        $w \leftarrow w + \alpha^w \delta \nabla \tilde{v}(s_t, w)$
        $\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(a_t|s_t,\theta)$
    end for
end while

En los métodos de *Actor-Critic* se cumple que $b = V_{\pi_\theta}(s_t)$. De esta elección deriva el nombre de crítico. La regla de actualización para este tipo de algoritmos es la siguiente:

$$\theta_{t+1} = \theta_t + \alpha \left(G_t - V(s_t|W)\right) \nabla_\theta \log \pi(a_t|s_t,\theta) \tag{2.44}$$
$$= \theta_t + \alpha \left(r_{t+1} + \gamma V(s_{t+1}|W) - V(s_t|W)\right) \nabla_\theta \log \pi(a_t|s_t,\theta) \tag{2.45}$$
$$= \theta_t + \alpha \delta_t \nabla_\theta \log \pi(a_t|s_t,\theta) \tag{2.46}$$
$$\tag{2.47}$$

La ventaja de Actor-Critic con respecto a REINFORCE es la rapidez en su convergencia y la estabilidad en el aprendizaje. Esto último es debido a que posee menor varianza. Ambas características son consecuencia de la adición de *bootstrapping* a la estimación de la función de valor.

2.2.1.7. Deterministic Policy Gradient

Deterministic Policy Gradient (DPG) [28] es un algoritmo de la familia de *Policy Gradient*, con la diferencia de considerar una política determinista en vez de una política estocástica. Esto lleva a la derivación de la siguiente función de costo:

$$J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[r(s, \mu_\theta(s))\right] \tag{2.48}$$

En donde $\mu_\theta : s \to a$ es la política determinista a ajustar. Con esto en consideración se puede enunciar el teorema de DPG. A continuación se muestra el gradiente para la actualización de la política determinista mediante DPG:

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \tag{2.49}$$
$$= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{2.50}$$

Este teorema asegura la existencia del gradiente de la función de costo para el ajuste de una política determinista. Con ello es posible enunciar un primer algoritmo, el cual puede ser *On-*

Algorithm 6 Actor-Critic

Entrada: Parametrización de una política diferenciable $\pi(a|s,\theta)$
Entrada: Parametrización de una función de valor estado diferenciable $\tilde{v}(s,w)$
Inicializar los parámetros $\theta \in \mathbb{R}^{d'}$ de la política
Inicializar los pesos $w \in \mathbb{R}$ de la función de valor
while True **do**
    Inicializar s
    $I \leftarrow 1$
    while s no es terminal **do**
        $a \sim \pi(\cdot|s,\theta)$
        Tomar acción a, observar s', r
        $\delta \leftarrow r + \gamma\hat{v}(s',w) - \hat{v}(s,w)$
        $w \leftarrow w + \alpha^w \delta \nabla \hat{v}(s,w)$
        $\theta + \alpha^\theta I \delta \nabla \ln \pi(a|s,\theta)$
        $I \leftarrow \gamma I$
        $s \leftarrow s'$
    end while
end while

Policy Deterministic Actor-Critic. Este último considera las siguientes reglas de actualización:

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t) \tag{2.51}$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \tag{2.52}$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)} \tag{2.53}$$

Donde $Q^w(s_t, a_t) \approx Q^\mu(s_t, a_t)$ es el crítico que estima el verdadero valor de la función de valor del par estado-acción. Además, se debe mencionar que la estimación de la función de valor usa SARSA para su actualización.

También existe la versión *Off-policy* del algoritmo anterior. En este caso se tienen dos políticas, una política objetivo determinista μ_θ y otra política arbitraria estocástica π que se encarga de generar las distintas trayectorias. Las reglas de actualización para este algoritmo son similares a las reglas 2.51, 2.52 y 2.53, modificándose la ecuación 2.51 por la siguiente ecuación:

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \tag{2.54}$$

2.2.2. Deep Reinforcement Learning

Con la llegada del *Deep Learning* y los resultados conseguidos con esta nueva forma de aprendizaje, también se modificó la forma en la cual se concibe el campo del *Reinforcement Leaning*. Esto dio nacimiento a numerosos algoritmos que ocupan modelos de *Deep Learning* ajustados bajo un paradigma de RL, categorizándose estos como algoritmos de *Deep Reinforcement Learning* (DRL).

2.2.2.1. Deep Q-learning

Deep Q-learning es uno de los primeros algoritmos de DRL, el cual fue introducido en el paper *Playing Atari with Deep Reinforcement Learning* [27]. El algoritmo desarrollado en este paper se denomina *Deep Q-learning*, pero es más conocido como *Deep Q-networks* (DQN). Este paper ha sido uno de los trabajos más exitosos que se han documentado en DRL, permitiendo aprender de manera exitosa un controlador desde un espacio de estados de alta dimensión. Dentro de los resultados obtenidos, se pueden mencionar el aprendizaje de múltiples juegos de atari usando únicamente las imágenes del juego y una señal de recompensa que es personalizada a cada juego. Sobrepasando el nivel humano en 29 de los 49 juegos que fueron usados para la evaluación del algoritmo.

La principal diferencia entre *Q-learning* y DQN es la parametrización de la función de valor Q mediante una red neuronal. Esta característica le permite al algoritmo solucionar problemas que posean un mayor número de dimensiones en el espacio de estados. Sin embargo, la restricción sobre el espacio de acciones se mantiene, restringiendo el espacio de acciones a uno discreto y de baja dimensionalidad. Al no ser un método tabular, se debe ocupar gradiente descendiente para la actualización de los parámetros de la red neuronal. La función de costo ocupada para la actualización es la misma regla de actualización de *Q-learning*. Regla que se muestra en la ecuación 2.22.

El algoritmo 7 muestra el funcionamiento de *Deep Q-learning*. A nivel de implementación, se puede verificar dos diferencias con respecto a *Q-learning*: un buffer de memoria, o *replay memory* en inglés, y la introducción de una red objetivo. La memoria de repetición guarda la experiencia del agente como 4-tupla de $(\phi_t(s_t), a_t, r_t, \phi_{t+1}(s_{t+1}))$, donde $\phi(s)$ es una función de preprocesamiento que recibe un estado. Esta función no es imprescindible, perfectamente se podría ocupar el estado s_t. El motivo de uso de esta memoria es la eficiencia de datos que brinda y la descorrelación de las muestras al momento de muestrear el *batch* de entrenamiento. Permitiendo que la actualización de los parámetros sea suave y evite comportamientos de divergencia. Obviamente, esta forma de actualización del modelo hace que sea *off-policy* debido a que las muestras ocupadas al momento de la actualización de la red neuronal, se hayan producido a partir de un modelo con parámetros diferentes a los actuales. Justamente, esta característica es la que se comparte con su predecesor *Q-learning*.

Por otro lado, se introdujo el uso de una red objetivo, la cual se actualiza cada C pasos de la red original. Esto último permite que el objetivo y con respecto al cual se actualiza la red neuronal, se haya generado a partir de una versión más atrasada que la red neuronal original, permitiendo que las oscilaciones y divergencias sean menos probables.

Para el trabajo [27] se ocuparon redes convolucionales para procesar las imágenes del videojuego, imágenes que fueron establecidas como los estados del ambiente.

2.2.2.2. Deep Deterministic Policy Gradient

Inspirado por el éxito de DQN y el uso de redes neuronales en problemas de RL, se intentó dar una solución a problemas de RL con espacio de acciones continuos mediante el uso de redes neuronales. Para ello se usó el algoritmo de DPG el cual demostró su valía en problemas de control continuo más las propiedades de las redes neuronales, denominándose a este nuevo algoritmo como *Deep DPG* (DDPG) [53]. Aunque la adición de estas últimas no

Algorithm 7 Deep Q-learning

Inicializar un buffer de memoria $\mathcal{D}$ de capacidad N
Inicializar la red neuronal que parametriza la función de valor Q del par estado-acción
con pesos aleatorios
Inicializar la red objetivo de la función de valor $\hat{Q}$ con parámetros $\theta^{-} = \theta$
for Episodio = 1, M **do**
    inicializar una secuencia $s_1 = \{x_1\}$ y la secuencia preprocesada $\phi_1 = \phi(s_1)$
    for t = 1, T **do**
        Con una probabilidad ϵ elegir una acción aleatorio a_t
        De otra forma elegir una acción $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Ejecutar la acción a_t en el emulador y observar la recompensa r_t y la imagen x_{t+1}
        Establecer $s_{t+1} = s_t$, a_t, x_{t+1} y preprocesa $\phi_{t+1} = \phi(s_t + 1)$
        Guadar la transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en $\mathcal{D}$
        Muestrear un *minibatch* aleatorio de transiciones $(\phi_t, a_t, r_t, \phi_{t+1})$ desde $\mathcal{D}$
        Establecer $y_j = \begin{cases} r_j, & \text{para un estado terminal} \phi_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta), & \text{Para un estado no terminal} \phi_{j+1} \end{cases}$
        Ejecutar un paso de gradiente descendente con $(y_j - Q(\phi_j, a_j; \theta))^2$
        Cada C pasos reiniciar $\hat{Q} = Q$
    end for
end for

asegura la convergencia del algoritmo, debido a que no está demostrado, teóricamente, que aproximadores no-lineales que aproximen funciones de valor o par estado-acción converjan o tengan un aprendizaje estable. Para solucionar lo anterior, se tomaron ideas de DQN para la estabilización en el entrenamiento de las redes neuronales, como lo son la incorporación de redes objetivos y *replay memory*.

Si bien, la solución al problema de estabilidad se inspiró en la red objetivo implementado en [27], no es la misma, ya que se adaptó a un algoritmo del tipo *Actor-Critic* y se incorporó una actualización suave de los objetivos (*Soft update*), en vez de copiar directamente los parámetros de una red neuronal a otra. Es decir, se copian los parámetros de las redes actor y crítico en redes objetivos $\mu(s|\theta^{\mu})$ y $Q(s, a|\theta^{Q'})$, respectivamente. Luego, para cada iteración se actualizan los parámetros de las redes objetivos mediante una actualización suave: $\theta' \leftarrow \theta\tau + (1 - \tau)\theta'$. Donde $\tau \ll 1$. Obligando que las redes objetivos cambien de manera lenta y permitan un aprendizaje más estable.

Un problema típico en problemas de control continuo con espacio de estados con baja dimensionalidad es la posible diferencia de escalas entre diferentes dimensiones de una misma observación, lo cual influye de manera negativa sobre el modelo a ajustar. Para solucionar este problema, se propuso en [53] la adición de *batch normalization* a cada red neuronal. Esta técnica de normalización se encarga de normalizar cada dimensión del vector de entrada, permitiendo que el algoritmo sea robusto a problemas de diferentes dimensionalidades y escalas, sin la necesidad de re-escalar manualmente los vectores de entrada.

En el algoritmo 8 se muestra los pasos a seguir para estimar una política en una tarea específica mediante DDPG.

Algorithm 8 DDPG

Inicializar de forma aleatoria las redes critica $Q(s, a|\theta^Q)$ y actor $\mu(s|\theta^\mu)$
con parámetros θ^Q y θ^μ, respectivamente
Inicializar la redes neuronales objetivos Q' y μ' con parámetros $\theta^{Q'} \leftarrow \theta^Q$ y $\theta^{\mu'} \leftarrow \theta^\mu$
Inicializar el buffer de memoria R
for Episodio = 1, M **do**
    inicializar un proceso aleatorio $\mathcal{N}$ para la exploración de acciones
    Recibir el estado inicial s_1
    for t = 1, T **do**
        Seleccionar acción $a_t = \mu(s|\theta^\mu) + \mathcal{N}$ acorde a la política actual y el ruido de exploración
        Ejecutar la acción a_t y observar recompensa r_t y el estado siguiente s_{t+1}
        Guardar la transición (s_t, a_t, r_t, s_{t+1}) en R
        Establecer $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$
        Actualizar el critico mediante la minimización de la función de pérdida:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

        Actualizar la política del actor usando un muestreo del gradiente de la política:

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

        Actualizar las redes objetivos:

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

    end for
end for

2.2.3. Deep Reinforcement Learning para Vehículos Autónomos

La investigación de DRL ha estado orientada, mayoritariamente, a resolver problemas de simuladores de videojuegos y algunos problemas de robótica. Esto en parte a que son problemas con objetivos definidos y con un universo de posibilidades acotados. Mientras que, por su lado, problemas que involucren vehículos autónomos no han sido suficientemente estudiados debido a la complejidad que tiene el entrenamiento de un agente en el mundo real. O también, el traspaso de una política entrenada de un simulador a un vehículo real. A pesar de estos problemas, se ha logrado investigar la utilidad que tiene DRL para la solución del problema de manejo autónomo mediante la implementación de variados simuladores que permiten el estudio de este problema [54][55][56][57].

Uno de los trabajos más reconocidos en el aprendizaje de una política de conducción autónoma es el trabajo de Alex Kendall, *Learning to Drive in a Day* [58]. En esta investigación se consiguió aprender una política de conducción autónoma para un escenario de simulación y un escenario del mundo real. El escenario simulado comprende diferentes ambientes de

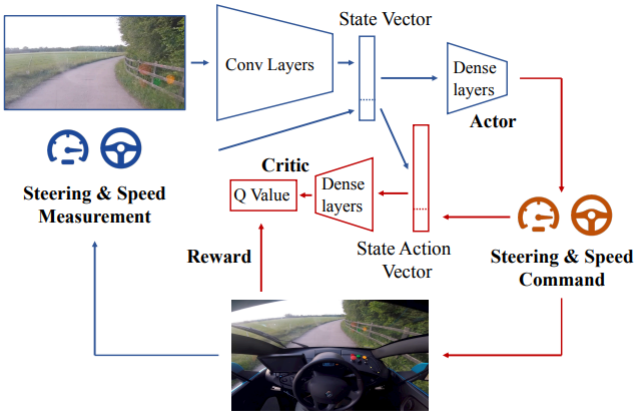


Figura 2.7: Modelo del agente ocupado en el trabajo de Kendall, *et al.* [58]. El agente esta basado en el acercamiento de algoritmos de *Actor-Critic* para el aprendizaje de comandos de control, tal como lo son el *steering* y la velocidad, en base a una imagen frontal del vehículo.

un mismo camino despejado, variando la textura, marca y topologías del camino. En donde para cada uno de estos escenarios, el vehículo debiese aprender a seguir la línea del camino (*Follow lane task*). Para solucionar tal problema, establecieron como agente una red neuronal del tipo VAE para la extracción de características desde las imágenes tomadas de una cámara frontal, y una red neuronal MLP para el control del vehículo. El modelo mencionado se muestra gráficamente en la figura 2.7. Para entrenar el agente ocuparon el algoritmo de DDPG, algoritmo que permite el entrenamiento con DRL en ambientes de control continuo, para el aprendizaje de los comandos de control del vehículo, como lo son la velocidad y dirección. La señal de recompensa que utilizaron para este problema fue la velocidad del vehículo, terminando el episodio cuando este cometiese alguna falta en contra de las reglas del tránsito. Así la función de valor de estado correspondería a la distancia recorrida por el vehículo. Con todo lo anterior implementado, pudieron comprobar que el sistema aprendía a conducir en los escenarios descritos en tan solo 10 episodios de entrenamiento. Con la idea validada en el escenario simulado, intentaron replicar el mismo experimento en un camino rural de 250 metros, llegando a obtener una política de conducción autónoma con nada más que 11 episodios de entrenamiento y 15 minutos de manejo.

Así como el trabajo de Kendall, hay otras investigaciones que apuntan a investigar la utilidad de DRL en las distintas situaciones y desafíos que plantea un ambiente de vehículos autónomos. Por ejemplo, una de las tareas más investigadas en el contexto de vehículos autónomos es la tarea de adelantamiento *Lane Change Task*, donde investigaciones como la de Chen [46] investigan el aprendizaje y comportamiento de un agente en el adelantamiento de vehículos en una carrera de velocidad. En el trabajo mencionado, implementaron DDPG y una variante de la misma con decisiones jerárquicas con gradiente inverso para entrenar una red neuronal con módulos de representaciones espacial y temporal. Demostrando que ambos algoritmos implementados lograron un 67 % y 73 % de casos éxitos en cambios de línea, respectivamente. Otra investigación sobre la misma tarea es la realizada por el paper de Wang et al. [59], en donde se implementó el algoritmo DQN con restricciones basado en reglas con un espacio de acciones de 3 movimientos: mantenerse en la pista, cambiar a la izquierda o derecha. Con ello entrenaron un agente con una red convolucional y una red neuronal MLP.

Demostraron que el acercamiento planteado logro un 80 % de cambios de línea de forma segura.

Otra tarea que es estudiada en el campo del manejo autónomo es la evasión de riesgo. Bajo este tópico hay varios trabajos que investigan distintas aristas de un aspecto similar, como lo es la seguridad en ruta. Uno de ellos es la investigación sobre la aplicación de freno de emergencia cuando se presenta alguna situación anómala. El trabajo de Fu *et al.* [49] investiga la obtención de una estrategia de frenado ante escenarios de emergencia mediante DRL. Más específicamente, hacen uso del algoritmo de DDPG para entrenar una red neuronal MLP que tiene como entradas la velocidad, posición y aceleración del vehículo líder y de seguimiento. Además de establecer una función de recompensa multi-objetivo que optimiza seguridad, efectividad y confort de los pasajeros.

Otra situación relativa a la seguridad en ruta es el estudio de métodos para el aprendizaje de una política robusta de conducción autónoma mediante generación de situaciones adversas. Una investigación que es relativa a este tópico es el trabajo de Abeysirigoonawardena *et al.* [71] el cual implementa un método de optimización bayesiana para la generación de comportamientos adversarios en contra del agente. Esto con el objetivo de aprender una política de conducción más robusta y que pueda sobrellevar cualquier situación adversaria que se le presente al agente. Con lo anterior, ellos pudieron demostrar que el método implementado lleva al aprendizaje de una política más robusta que una política entrenada en escenarios típicos de conducción. Otro trabajo ligado al mismo tema es el realizado por Pan *et al.* [60], el cual propone un método de DRL robusto a situaciones adversarias y adverso a riesgo. Para lograr aquellos implementan dos modelos de DRL, uno para el aprendizaje de la conducción del agente y otro para el aprendizaje de situaciones adversas en contra del agente. Al primero lo denominan como agente protagonista, mientras que al segundo lo denominan como agente adversario. El modelo neuronal que ocuparon es una red neuronal convolucional con una red neuronal MLP, que se divide en su última capa neuronal en k-ramas. El método de DRL que ocuparon para la actualización de los agentes es una versión modificada de DQN, en donde modificaron la ecuación de *Bellman* de la función de valor del par estado-acción Q del protagonista de la siguiente forma:

$$\hat{Q}_P(s,a) = Q_P(s,a) - \lambda_P Var_k(Q_P^k(s,a)) \tag{2.55}$$

Mientras que para el adversario considera una función de valor Q de la siguiente forma:

$$\hat{Q}_A(s,a) = Q_A(s,a) + \lambda_A Var_k(Q_A^k(s,a)) \tag{2.56}$$

Donde $Q_{(P,A)}(s,a)$ es la función de valor Q tradicional del protagonista o el adversario, según corresponda y $\lambda_{(P,A)} Var_k(Q_{(P,A)}^k(s,a))$ es el término averso a riesgo (*risk-averse*) e incentiva al protagonista/adversario a elegir acciones que minimicen/maximicen la varianza. Mientras que el operador Var es la varianza de la función Q a lo largo de las k diferentes estimaciones de la función de valor Q: $Var_k(Q(s,a)) = \frac{1}{k}\sum_{i=1}^{k}(Q^i(s,a) - \tilde{Q}(s,a))$. El modo en como interactúan ambos agentes es mediante la toma de acciones de forma alternada. En donde los primeros m pasos son del protagonista y los siguientes n pasos son del adversario. Con este planteamiento demostraron que la política de conducción obtenida consigue mejores recompensas catastróficas que las obtenidas, por ejemplo por DQN o *Ensemble DQN*.

Además de las tareas ya enunciadas, otra tarea que se investiga en el campo del manejo autónomo con foco en DRL es el estudio de métodos de *Curriculum Learning* (CL) basados en DRL para la aceleración en el aprendizaje del agente autónomo. Un estudio a destacar en esta materia es el realizado por Qiao *et al.* [51], quienes implementan un método de CL que selecciona los escenarios en los cuales va a ser entrenado el agente de manera automática. Para lograr aquello, eligen N escenarios fijos. Luego, categorizan cada uno de estos escenarios mediante la función de valor que obtiene el agente en cada uno de estos. Con esta categorización, efectúan un muestreo de escenarios mediante una distribución de boltzmann, teniéndose como escenarios más probables aquellos que posean una menor función de valor. La idea detrás de este muestreo de escenarios es la de entrenar el agente en los escenarios que le sea más difícil tener éxito y menos en aquellos que le sea más fácil al agente aprender. Para evaluar el algoritmo implementado, plantearon dos situaciones diferentes a superar. Una de ellas es el problema de atravesar una intersección, mientras que la otra es la aproximación hacia una intersección. Para la primera situación hicieron uso de acciones discretas, por lo que eligieron usar DQN para el aprendizaje, mientras que para el segundo escenario hicieron uso de acciones continuas, por lo que implementaron DDPG para su solución. En ambos casos demostraron que un paradigma de entrenamiento con CL generado de manera automática supera ampliamente en estabilidad y convergencia a un entrenamiento secuencial y aleatorio.

2.2.4. Deep Reinforcement Learning con Representation Learning

Según Yoshua Bengio en su trabajo *Representation Learning: A Review and New Perspectives* [62] define *Representation Learning* como el aprendizaje de representaciones de los datos que hacen más fácil la extracción de información útil a partir de un clasificador u otro predictor. La ventaja principal de este tipo de aprendizaje radica en la mejora del clasificador o predictor en las métricas de interés, a la vez que mejora la rapidez de convergencia. Además de las ventajas mencionadas, también se pueden extrapolar una par de ventajas más que mejoran algunos aspectos del *pipeline* de ML. Una de ellas es la interpretabilidad del modelo, permitiendo saber en qué se está focalizando el modelo para tomar las decisiones. Otra ventaja sería la selección automática de las características de importancia, lo cual permitiría no hacer uso de conocimiento experto para estas características.

Representation Learning ha sido uno de los tópicos de investigación más importante a incluir en el *pipeline* de RL debido a los beneficios que trae. Por lo mismo, se ha venido investigando de manera activa la incorporación de ambos tópicos de investigación. Uno de los trabajos más relevante e insigne en este campo es el realizado por David Ha *et al.* en *World Models* [70], el cual plantea el uso de redes neuronales generativas para el aprendizaje de información espacial y temporal de manera compacta en un ambiente de RL. A tal modelo, lo denominaron *World Model* y tiene como objetivo el aprendizaje de características relevantes del ambiente, simplificando el aprendizaje de la tarea para un agente más compacto y simple. Específicamente, ellos implementaron un modelo VAE en conjunto a una red LSTM para el aprendizaje espacial y temporal en el ambiente del videojuego *Doom*. Ambas redes neuronales representan la información visual y memoria del modelo de representación de estados, dando al controlador información enriquecida para que tome la decisión a realizar dentro del ambiente. La ventaja de usar este modelo de representación de estado es la rapidez con la cual se entrena el controlador, ya que el modelo de representación se entrena de manera no-supervisada en sus dos componentes, entrenando de manera *online* únicamente el

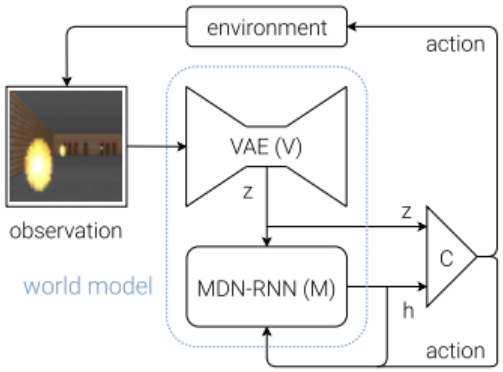


Figura 2.8: Diagrama de flujo del agente construido en [70]. La observación recibida desde el ambiente es procesada primeramente por el encoder del modelo VAE para producir un vector latente z que codifica una representación espacial del ambiente. Para luego, ser entrada de una modelo temporal que produce un vector latente oculto h que captura las dependencias temporales del ambiente. Finalmente, el controlador C toma la acción en base a los vectores z y h.

controlador. Por lo cual, se ve acelerada el aprendizaje de este último.

En el trabajo anterior no se habla explícitamente que el desacoplamiento entre el aprendizaje de la representación del ambiente y el controlador, constituye una forma de aplicación de *Representation Learning*. Es más, en problemas de control, el aprendizaje de representaciones de estados en baja dimensionalidad, que evolucionan a través del tiempo y que se ven influenciados por interacciones con el ambiente y las acciones del agente, se denominan bajo la categoría de *State Representation Learning* [63]. Al igual que *Representation Learning* es un caso particular de *Feature Learning*, enfocado principalmente en problemas de control. Este paradigma de aprendizaje ha logrado hacer a RL más rápido en tiempo de ejecución y liviano computacionalmente [63].

Dada las ventajas que posee el uso de *Representation Learning*, se ha usado en varias ocasiones en el campo de vehículos autónomos. El mejor ejemplo es el trabajo de Kendall [58], el cual utilizaba una representación espacial del ambiente en donde estaba inserto el vehículo, representación que era obtenida desde el modelo VAE. Otro trabajo relativo a vehículos autónomos que emplea este aprendizaje de representaciones de estado es el de Porav *et al.* [64], con su trabajo denominado *Imminent Collision Mitigation with Reinforcement Learning and Vision*. En él se implementa una arquitectura neuronal similar a la de *World Models* para solucionar el problema de colisiones inminentes en un ambiente de manejo autónomo urbano. Ellos pudieron demostrar que el algoritmo planteado conseguía un radio de colisión mucho menor al obtenido por un algoritmo de DQN que ocupaba información del ambiente como posición y velocidad de los exo-agentes.

2.3. Imitation Learning

Imitation Learning (IL) es un *framework* que intenta aprender algún comportamiento a partir de demostraciones expertas. Por lo mismo, también es conocido como *Learning from*

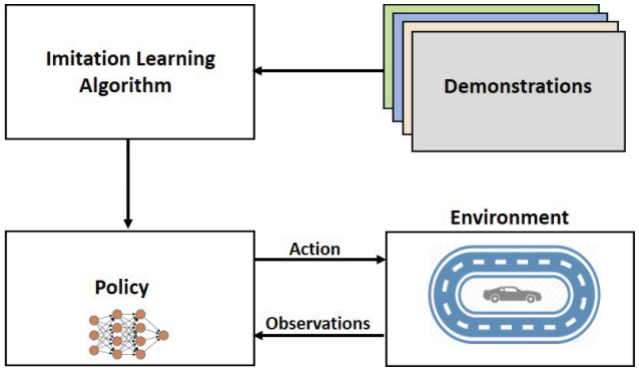


Figura 2.9: IL Pipeline. Figura tomada desde [29]

demonstration [30]. La figura 2.9 muestra el *pipeline* que define IL. Resumidamente, el algoritmo de IL toma demostraciones de un experto desde una base de datos para así obtener una política que imite la política del experto y así permita solucionar alguna tarea de interés con las mismas habilidades que la de un experto.

IL se divide en dos clases de técnicas: *Behavioural Cloning* (BC) e *Inverse Reinforcement Learning* (IRL). La primera apunta a usar un mapeo directo desde los estados a las acciones, mientras que el segundo intenta parametrizar la función de recompensa desde las demostraciones. Debido a su importancia en esta tesis, se explicará más en detalle BC. Sin embargo hay interesantes documentos que explican y resumen el aporte de esta técnica en varias áreas de la inteligencia artificial [32][33][34].

Básicamente, BC intenta obtener una política de acción π_θ para que produzca acciones similares a una política de acción experta π^*. Los parámetros θ se obtienen de la minimización de alguna función de costo tal como se muestra en la ecuación 2.57. Donde s es el estado u observación del ambiente y a^* las acciones del experto. Por lo general, la función de costo ℓ toma la forma de una función MSE o NLL.

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_i \ell(\pi_\theta(s), a^*) \tag{2.57}$$

BC puede subdividirse en métodos libre de modelo (*Model-Free* BC) y basados en modelo (*Model-Based* BC) [30]. En donde el primero aprende a resolver una tarea mediante la imitación de las acciones del experto sin la preocupación del contexto. Mientras que el segundo, aprovecha la información del ambiente para aprender las dinámicas de este de forma iterativa. Obviamente, esta complejidad extra hace que los métodos bajo esta categoría sean más exhaustivos a nivel computacional, debido a su naturaleza iterativa que los métodos libres de modelo. Pero, a su vez, son más precisos con información incompleta, lo cual no es del todo cierto para métodos libres de modelo en donde puede que el controlador produzca salidas no factibles debido al entrenamiento con un controlador imperfecto.

Las aplicaciones en las cuales se ha usado IL han sido mayormente en sistemas autónomos y robóticos, pero no han sido los únicos, de hecho se han aplicado en variados campos relativos a la computación, como por ejemplo, procesamiento de lenguaje natural [35][36]. Algunos

ejemplos de IL en robótica son los trabajos de Ratliff, Et. al. [37], el cual estudia el aprendizaje de locomoción y manipulación autónoma de un robot cuadrúpedo mediante la imitación de datos que fueron obtenidos desde el manejo por control remoto del mismo. Otro ejemplo es el trabajo de Zhang, et al. [38], quienes demostraron la efectividad IL en el aprendizaje de distintas tareas de manipulación por parte de un robot PR2. Una de las conclusiones más relevantes de este trabajo fue la efectividad que tuvo el algoritmo de IL para mapear valores de píxeles a acciones, demostrando, por ejemplo, la obtención de un 88.9 %, 77.8 % y 50 % considerando 193, 115 y 67 demostraciones de un experto, respectivamente, para la tarea de alineación de las garras de un martillo de juguete con un clavo.

2.4. Imitation Learning para Vehículos Autónomos

Junto a los campos mencionados anteriormente, uno de los más investigados con respecto a IL es el área de *Autonomous Driving*. Si bien, IL no implica necesariamente el uso de redes neuronales, el uso de éstas ha dominado el campo de IL aplicado a vehículos autónomos. Uno de los primeros trabajos que utilizo una red neuronal como modelo de conducción fue el trabajo de Pomerleau [39]. El trabajo mencionado demostró que un modelo de ML, en especial, un modelo de red neuronal, tiene la capacidad de aprender a conducir un vehículo a través de una ruta. Específicamente, lograron que el NAVLAB (vehículo equipado con diferentes sensores para pruebas de conducción autónoma) anduviese 400 metros de manera autónoma por el campus de la universidad Carnegie Mellon con una velocidad de 0.5 metros por segundo. Algo interesante a mencionar de este trabajo es la arquitectura neuronal que ocuparon para el control del vehículo. Ellos ocuparon una red neuronal MLP que tenía tantas neuronas de entrada como píxeles tenía la imagen de entrada, más la imagen proporcionada por un sistema de rango. Hoy en día, las imágenes se procesan con redes convolucionales y no con redes neuronales MLP, tal como lo hicieron ellos. Obviamente, ellos no tenían el conocimiento para saber aquello en aquel momento.

Otro trabajo insigne en la materia, está dado por el paper *End to End Learning for Self-Driving Cars* [40], paper que fue realizado por la empresa NVIDIA y que fue implementado en un sistema real de conducción autónoma que denominaron DAVE-2. Ellos implementaron un módulo CNN que aprendiese de las entradas de tres cámaras y obtuviese el comando *steering* para que el vehículo se conduciese de manera autónoma. Si bien el entrenamiento no se explicitó como IL, si tuvo una metodología de aprendizaje supervisado en donde la señal experta era el comando *steering* con lo cual se terminaba aprendiendo a partir de la acción del experto. Entre los resultados más destacados que obtuvieron se puede mencionar la obtención de un 98 % del tiempo de manejo con una modalidad de conducción autónoma en la prueba de manejo autónomo en ruta. Esto confirmo que el despliegue de un modelo de *Deep Learning* de manera *end-to-end* tiene la capacidad para aprender la tarea de manejo autónomo en producción.

Además de los trabajos anteriores, es importante mencionar la investigación de Codevilla, et al. [41] quien implemento una variación de BC con información interna del vehículo. Esto debido a que las intenciones de un agente no están completamente causadas por las observaciones, sino que también por el estado interno del conductor, como por ejemplo señaletica de tránsito o intenciones. La modelación del estado interno considero, por tanto, un conjunto de comandos que resumen la intención del agente. Esta información se incluye en el modelo neu-

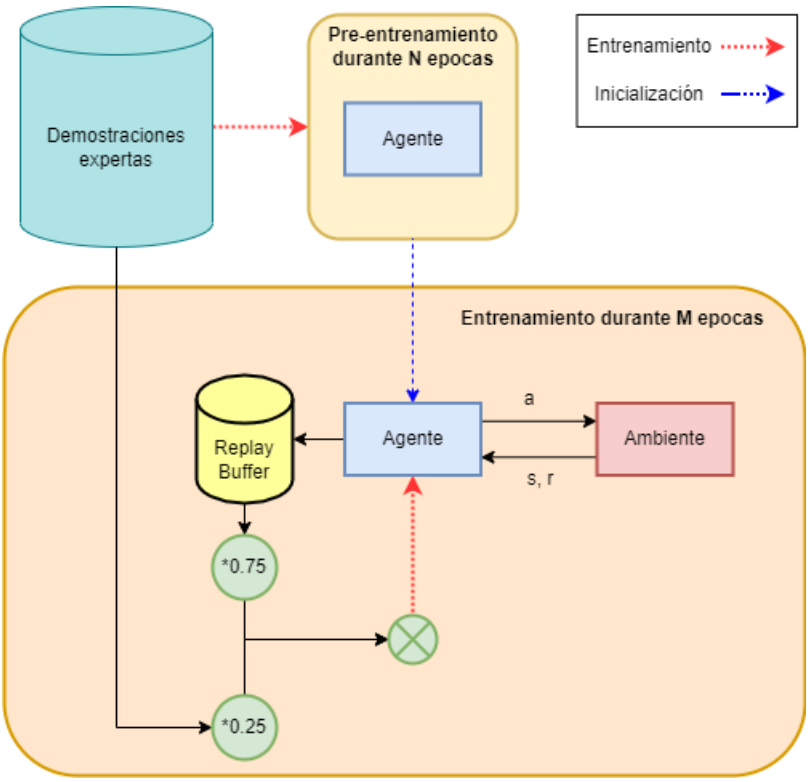


Figura 2.10: Diagrama de flujo del algoritmo CoL [52]. Figura de autoría propia.

ronal para obtener la acción a ejecutar y comparar con la acción del experto. Los resultados más interesantes de este trabajo consistieron en la validación de su método en el simulador CARLA y en un auto a escala de control remoto. Pudieron demostrar que el sistema de ellos pudo recorrer variados caminos con mínima intervención humana y sin fallar ninguna vuelta en el trayecto.

Aunque BC ha demostrado su utilidad en distintos estudios de AD, no es un método infalible, ya que sufre de algunas limitaciones. Una de esas limitaciones es *covariance shift* [42][43], que son situaciones en donde la distribución de las entradas cambia, mientras que la distribución condicional de las salidas dada la entrada, permanece invariable. Estas situaciones son un caso particular de *Dataset shift* que engloba las situaciones en donde la distribución conjunta de las entradas y salidas difiere entre los escenarios de entrenamiento y prueba [44]. Es decir, BC funciona dentro de la distribución para la cual fue entrenada, sin embargo, para problemas con un espacio de estados tan amplio como lo es AD, posiblemente se encuentre situaciones en donde el sistema probablemente falle, pudiendo ocurrir situaciones que puedan ser fatales. Otra limitación presente en BC es la parcialidad en el conjunto de los datos de las muestras expertas [45]. Por lo general, cuando se recolectan datos no se capturan todos los comportamientos posibles para una situación en específico, por lo que las acciones que se toman no representan la mejor acción posible. Aunque no son las únicas limitaciones, ya que BC también sufre de confusión causal y alta varianza [45].

Si bien lo anterior supone un problema para el pipeline de IL, también se han investigado diversas técnicas para que el agente aprenda continuamente durante toda su vida en el

ambiente, sin importar la cantidad de entrenamiento con IL haya tenido, permitiendo que el agente aprenda de situaciones nuevas que nunca vio en el entrenamiento por parte de un experto. Uno de los trabajos que aplica esta premisa a su investigación es el trabajo de Goecks, et al. [52], quien propone un aprendizaje combinado entre IL y RL para el continuo aprendizaje del agente sin desviarse de lo aprendido por parte de un experto. El algoritmo planteado en este trabajo fue denominado como *Cycle-of-Learning*(COL) y combina en proporciones fijas demostraciones expertas más demostraciones que se vayan generando de manera *online* para el entrenamiento del agente. En la figura 2.10 se muestra el diagrama de flujo del algoritmo CoL. En esta se puede ver como el agente se entrena de manera supervisada con demostraciones expertas, para después mantener proporciones fijas entre demostraciones expertas y demostraciones nuevas que el agente vaya generando dentro de este nuevo ambiente. Lo interesante de este nuevo planteamiento es el hecho que posibilita el entrenamiento supervisado de una red neuronal crítica mediante BC, hecho que no ocurre en planteamientos clásicos de algoritmos de BC.

3. Metodología

En este capítulo se describe la solución propuesta para el problema de evasión de colisión en un ambiente urbano. Primero se describe el problema, después los módulos neuronales ocupados para el aprendizaje por representación y el modelo de aprendizaje del controlador. Por otro lado, también se describe y detalla la formulación de la función de recompensa. Finalmente, se detallan los distintos detalles de la configuración experimental a ocupar en la experimentación de esta tesis.

3.1. Formulación

Formulamos el problema de navegación autónoma a base de un objetivo a alcanzar dentro de un ambiente de manejo adversario por parte de exo-agentes presentes en el mismo. A su vez, el agente deberá evitar colisiones con exo-agentes, al mismo tiempo que optimiza ciertas variables de manejo, tales como velocidad y alineación de ruta. El escenario en donde estará ubicado el agente se presenta gráficamente en la figura 3.1 la cual muestra un ego-agente, representado por un vehículo de color azul, que deberá conducir, en forma rectilínea, hasta el objetivo delineado por el cuadrado rojo al final de la ruta. El escenario presentado está inspirado por [71]. En este, se colocarán exo-vehículos en distintas posiciones, orientaciones y diferentes situaciones de conducción adversaria en donde el agente deberá evitar colisionar y tratar de arribar a la meta. Implementando métricas de evaluación tales como porcentaje de llegada y de colisión (*Success and Collision Rate*), se espera evaluar el desempeño del ego-agente en las distintas situaciones que se pueden dar en el escenario descrito. Entrenamos y validamos la estrategia planteada en el simulador de alta fidelidad CARLA (0.9.11) [55].

3.2. Espacio de acciones

El espacio de acciones contempla el uso de acciones continuas para el control del ego-vehículo. Motivado por una serie de investigaciones en el campo del *Autonomous Driving* [58], [74], [64], [49] se establecen dos salidas del controlador las cuales son el *Steering Angle* y *Throttle* para la conducción del mismo. Ambas acciones que se denotarán conjuntamente como $a_t = \{Steering, Throttle\}$, las cuales tienen rangos de operación entre los -1 a 1 para la primera variable, y 0 a 1 para la segunda variable del espacio de acción.

3.3. Espacio de estados

El espacio de estados será definido por las observaciones O_t que se obtendrán desde una sola cámara frontal. Debido a distintas investigaciones que han demostrado que el uso de imágenes semánticas permiten una aceleración y mejor desempeño del modelo de conducción [58][72], se opta por usar imágenes semánticas que son dadas por el simulador CARLA [55] y que están codificadas según las etiquetas del dataset *Cityscapes* [77].

Figura 3.1: Ambiente en donde esta ubicado el agente, que esta representado por el vehículo de color azul, y que tendrá la misión de conducir hasta la meta, delineada por el cuadrado de color rojo.

3.4. Modelo del agente

Para el modelo del agente se propone usar una arquitectura neuronal generativa, tal como se planteó en [58][70] que aprenda representaciones espaciales compactas que ayudan en la estabilidad y rapidez del aprendizaje. Además, de contemplar el aprendizaje de cada uno de sus módulos de forma individual. Esto último permitirá que la toma de decisión sea más fácil y rápida de aprender, ya que los módulos de representación se entrenan de manera supervisada *offline*, dejando únicamente al controlador en un aprendizaje *online*, alivianando la carga que conlleva el aprendizaje de una red neuronal de manera *online*.

La arquitectura propuesta tendrá una primera etapa de representación espacial compuesta por un β-*VAE-Encoder* [75] que permitirá una representación compacta y desagregada de la observación O_t. Este *Encoder* deberá ser entrenado de manera supervisada y separada del resto de la arquitectura [64][70]. Se debe considerar también que en su etapa de entrenamiento se usará toda la arquitectura *VAE* mientras que en el entrenamiento del controlador se ocupará únicamente el *Encoder*. La figura 3.2 muestra la arquitectura completa de una red β-VAE al momento de su entrenamiento.

Una vez obtenida esta representación latente z_t que compacta la observación del estado, se concatenan las variables de control del vehículo a_{t-1}, es decir el *steering* y *throttle*. Luego, el vector resultante es entrada de una red neuronal MLP que representará el papel de controlador del vehículo. En la figura 3.3 se presenta el modelo del ego-agente controlador.

3.5. Función de recompensa

En la tabla 3.1 se muestran las funciones de recompensa a considerar para el entrenamiento del agente. Las funciones de recompensas que llevan por nombre “Colisión a Vehículo” y

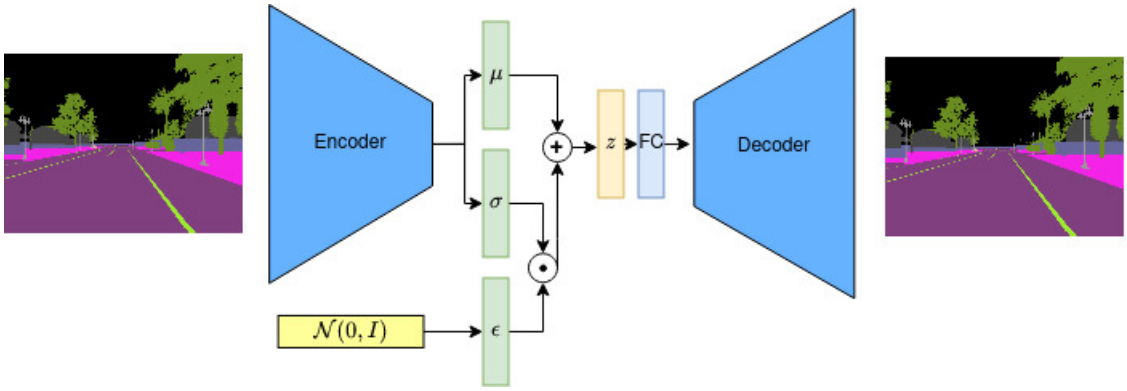


Figura 3.2: VAE en entrenamiento supervisado *Offline*

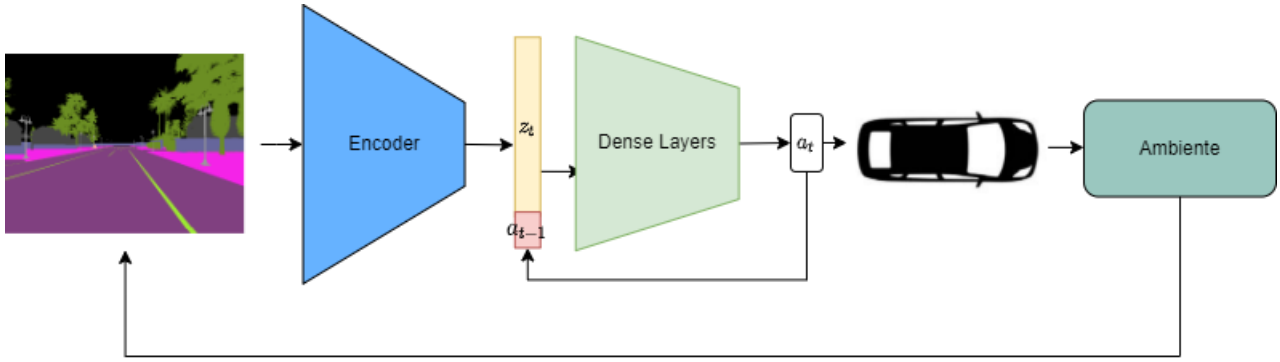


Figura 3.3: Modelo neuronal del Ego-Agente

"Colisión con otro obstáculo" apuntan a la seguridad en el manejo del vehículo, por lo que penalizan colisiones que pueda sufrir el agente con respecto a un exo-vehículo o cualquier otro obstáculo que no sea un vehículo. Estas funciones fueron tomadas desde [74]. Además, se proponen las funciones de recompensa "Llegada a la meta" y "Distancia a la meta". La primera incentiva al agente a llegar a cierto objetivo, mientras que la segunda recompensa por la distancia que posee el agente con respecto al objetivo. Esto último con el fin de guiar al vehículo hacia su objetivo final. Esta función considera una distancia máxima igual a la distancia que hay entre la posición inicial del agente con respecto a la meta. Es necesario especificar que si el vehículo no alcanza el objetivo planteado, no suma nada en ese ítem de recompensa, pero se trata como un caso diferente a aquellos en donde hay colisión con algún objeto en la ruta.

Las funciones de recompensas "Límite de velocidad", "Alineación con respecto a ruta" y "Centralización de línea" son una modificación de [72][74]. La primera apunta a que el agente aprenda a conducir dentro de un rango de velocidades. Este rango está dado por el conjunto $v_{min} = 10[km/h]$ y $v_{target} = 40[km/h]$. Creciendo y decreciendo de manera exponencial desde 0 a v_{min} y desde v_{target} a $v_{max} = 60[km/h]$, respectivamente. Si se llegase a superar la velocidad máxima establecida se considera una recompensa negativa. A su vez, la segunda función apunta a penalizar al vehículo por desviar su orientación paralela a la orientación de la ruta. Además, esta posee una lógica similar a la función de velocidad, en donde se establece un decaimiento exponencial de recompensa positiva desde un ángulo 0 a hasta un ángulo $\alpha_{min} = 20$. Si llegase a sobrepasar el ángulo mencionado, se empieza a

Tabla 3.1: Función de Recompensa

Nombre	Función	Peso(w)
Colisión a Vehículo	$\begin{cases} -1, & \text{Si } Pos^A \leqslant Pos_i^E \\ 0, & \text{De otra forma} \end{cases}$	1000
Colisión con otro obstáculo	$\begin{cases} -1, & \text{Si } Pos^A \leqslant Obs_i \\ 0, & \text{De otra forma} \end{cases}$	1000
Llegada a la meta	$\begin{cases} 1, & \text{Si } Pos^A \leqslant Pos^{Goal} \\ 0, & \text{De otra forma} \end{cases}$	500
Distancia a la meta	$\begin{cases} 1 - (\frac{d}{d_{max}})^\beta, & \text{if } d <= d_{max} \\ -1, & \text{De otra forma} \end{cases}$	5
Límite de velocidad	$\begin{cases} (\frac{v}{v_{min}})^\beta, & \text{if } v <= v_{min} \\ 1, & \text{if } v_{min} < v < v_{target} \\ 1 - (\frac{v - v_{target}}{v_{max} - v_{target}})^\beta, & \text{if } v_{target} < v < v_{max} \\ -1, & \text{De otra forma} \end{cases}$	1
Alineación con respecto a ruta	$\begin{cases} 1 - (\frac{\alpha}{\alpha_{min}})^\beta, & \text{if } \alpha <= \alpha_{min} \\ -1 + (\frac{\alpha_{max} - \alpha}{\alpha_{max} - \alpha_{min}})^\beta, & \text{if } \alpha_{min} < \alpha < \alpha_{max} \\ -1, & \text{De otra forma} \end{cases}$	1
Centralización de línea	$\begin{cases} 1 - (\frac{d}{d_{min}})^\beta, & \text{if } d <= d_{min} \\ -1 + (\frac{d_{max} - d}{d_{max} - d_{min}})^\beta, & \text{if } d_{min} < d < d_{max} \\ -1, & \text{De otra forma} \end{cases}$	1
Distancia a exo-agentes	$\sum_{i=1}^{N} \begin{cases} (\frac{d_i}{R})^\beta - 1, & \text{if } d_i <= R \\ 0, & \text{De otra forma} \end{cases}$	1

penalizar con recompensa negativa hasta un ángulo de $\alpha_{max} = 30$. Más allá de este ángulo límite, penaliza continuamente con -1. Mientras que la tercera función de recompensa tiene como objetivo penalizar la distancia del vehículo con respecto al centro de la ruta, teniendo como limites una distancia $d_{min} = 1.5$ y $d_{max} = 3.0$. Por último, se propone una función de recompensa que fue denominada "Distancia a exo-agentes" que penaliza la cercanía del agente con respecto a los exo-vehículos en ruta. Estableciéndose una área de penalización de 2 metros con respecto al centro del exo-vehículo.

Al igual como se hace en [74] se opta por ponderar las funciones de recompensa por parámetros w_i para asignar importancia a las funciones de recompensa. Por lo que la función de recompensa final tiene la siguiente forma: $R = \sum_i w_i r_i$. Es importante mencionar que no es necesario una normalización de recompensas, tal como se hace en [74], ya que todas tienen un dominio mínimo y máximo entre -1 y 1, por lo que influyen de igual manera en la recompensa total, diferenciándose únicamente por la ponderación de importancia. Por

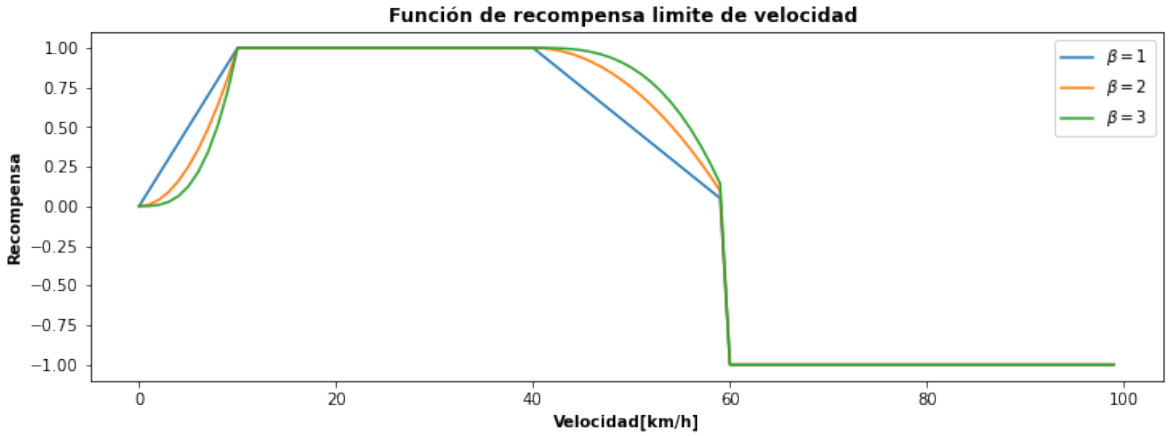


Figura 3.4: Función de recompensa "Límite de velocidad"

lo demás, es relevante destacar que el exponente β ayuda a controlar la aceleración del crecimiento o decrecimiento en la velocidad, posición, ángulo, etc. penalizando o tolerando de manera exponencial las distintas funciones de recompensa. Este parámetro se estableció en un valor $\beta = 3$ para todos los experimentos realizados. Para entender este valor y como influye en la función de recompensa, considérese la figura 3.4 en donde se muestra la función de recompensa denominada "Límite de velocidad" para distintos valores de β. De esta figura, se puede verificar como la función de recompensa se vuelve más suave a medida que el β aumenta. Esto permite que las velocidades que estén por debajo de la velocidad mínima, obtengan más recompensa o menos según su cercanía al límite inferior. Mientras que por el límite superior, la recompensa disminuye de manera más agresiva a medida que la velocidad está más alejada del umbral objetivo. Permitiendo que el agente penalice de mayor forma a medida que se aleja de los umbrales establecidos. Finalmente, se debe mencionar que los valores de los parámetros de las distintas funciones de recompensa fueron elegidos de manera propia con respecto al problema que se está intentando solucionar en esta tesis.

3.6. Algoritmos de Aprendizaje Reforzado

Para el entrenamiento del agente se propone un algoritmo de aprendizaje reforzado que hace uso de muestras expertas. Este algoritmo denominado TD3CoL e inspirado por [52], posee dos fases de aprendizaje, una primera fase de aprendizaje por imitación y otra fase que combina BC más RL para el aprendizaje del agente. La ventaja del uso de un algoritmo de este tipo radica en la eficiencia, estabilidad y rapidez de aprendizaje del agente, ya que posibilita el entrenamiento de manera *offline* dada demostraciones expertas, y que se sigue reentrenando en un escenario nuevo con la reutilización de demostraciones expertas más las demostraciones generadas desde este nuevo ambiente. Además, y a diferencia de *CoL*, se propone añadir las mejoras que se introdujeron a *DDPG* en *TD3* [66] para tratar problemas de sesgo de sobreestimación (*Overestimation Bias*) y reducción de varianza en la actualización de la red actor. El primero lleva al modelo a estimar de manera pobre la función de valor que lleva a políticas subóptimas. Mientras que el segundo, causa divergencia en el actor debido a la actualización en estados altamente erróneos que son dados por la sobre-estimación por parte de la red neuronal que parametriza la función de valor Q. En resumen, las mejoras que introduce TD3CoL con respecto a CoL, son las siguientes:

- Modificación del algoritmo de aprendizaje reforzado de DDPG a TD3. Algoritmo que ha demostrado su superioridad por sobre DDPG en varios trabajos[66][67][68][69].
- Reducción en la sobre-estimación de la función de valor Q mediante la adición de un critico adicional.
- Aumento en la calidad de las actualizaciones de la política del actor mediante una mejor estimación de los valores por parte de las redes objetivos.

Para lograr lo anterior, se deben modificar las funciones de pérdida del agente para combinar esta idea de imitación y aprendizaje *online*. Por lo mismo, se establecen las siguientes funciones de pérdida:

- **Expert Behavior Cloning Loss:** Dada demostraciones expertas:

$$\mathcal{D}^E = \{(s_t^E, a_t^E)\}_{t=1}^N \tag{3.1}$$

En donde s_t^E y a_t^E representan el estado del ambiente y la acción del agente experto en el tiempo t, respectivamente. Luego, la función de perdida de imitación viene dada por la minimización entre la acción tomada por la red del actor $\pi_\phi(s_t)$, parametrizada por parámetros ϕ, y la acción tomada por el agente experto a_t^E.

$$\mathcal{L}_{BC}(\phi) = \frac{1}{2}(\pi_\phi(s_t) - a_t^E)^2 \tag{3.2}$$

- **1-step return Q-learning loss:** El calculo del *1-step return* se calcula en términos de las dos redes objetivas críticas $Q_{\theta_i'}$, parametrizadas por θ_i':

$$R_1 = r_t + \gamma \min_{i=1,2} Q_{\theta_i'}(s_{t+1}, \pi_{\phi'}(s_{t+1})) \tag{3.3}$$

En donde $\pi_{\phi'}$ es la red objetivo del actor. Esta minimización que se introdujo en [66] denominada *Clipped Double Q-learning* ayudará en el control de la sobre-estimación de la función de valor del par estado-acción. Luego, para cumplir con la ecuación de Bellman se debe minimizar la función de valor del par estado-acción con respecto al *1-step return* considerando un batch de estados muestreados $\mathbf{s}$.

$$\mathcal{L}_Q(\theta_i) = \frac{1}{2}(R_1 - Q_{\theta_i}(\pi_\phi(\mathbf{s})))^2 \tag{3.4}$$

- **Actor Q-loss:** Para la función de pérdida del actor se debe maximizar la función de valor del par estado-acción con respecto al estado actual, lo cuál significa que se debe optimizar con respecto al negativo del *Q-value* predicho por el primer crítico:

$$\mathcal{L}_A(\phi) = -Q_{\theta_1}(s_t, \pi_\phi(s_t)) \tag{3.5}$$

- **L2 *regularization***: Además de las funciones de pérdida tradicionales se añade regularización a los modelos del actor y ambos críticos.

$$\mathcal{L}_{L2}(\phi) = \phi^T\phi \tag{3.6}$$
$$\mathcal{L}_{L2}(\theta_i) = \theta_i^T\theta_i \tag{3.7}$$

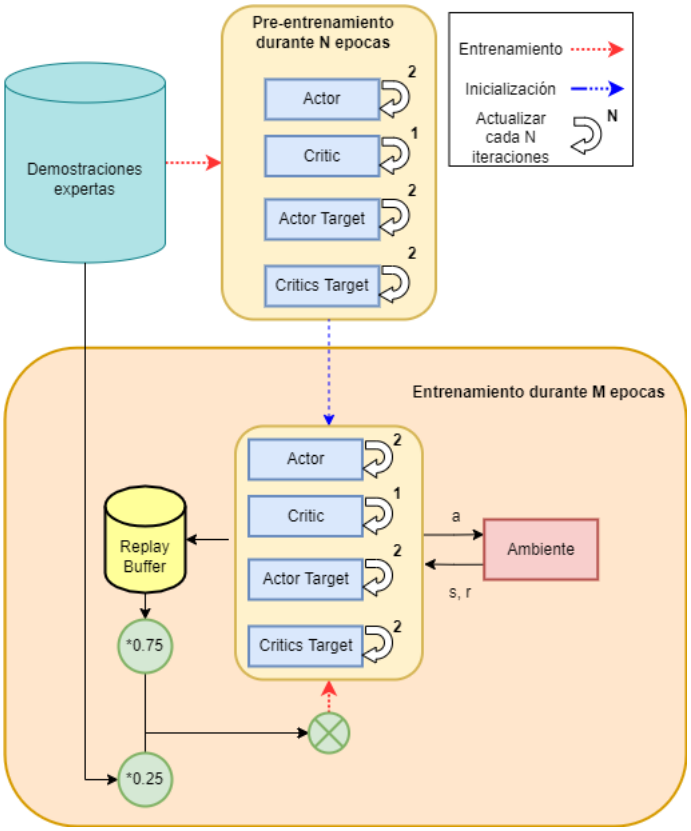


Figura 3.5: Diagrama de flujo del algoritmo TD3CoL.

Cabe destacar que la nomenclatura de nombres no es propia sino que se tomo desde [52] que es de donde se propuso el algoritmo base original más las adiciones correspondientes que contempla el nuevo algoritmo. Finalmente, la funciones de pérdida del actor y los críticos vienen dadas por las ecuaciones 3.8 y 3.9, respectivamente.

$$\mathcal{L}_{actor}(\phi) = \lambda_{BC}\mathcal{L}_{BC}(\phi) + \lambda_{A}\mathcal{L}_{A}(\phi) + \lambda_{L2}\mathcal{L}_{L2}(\phi) \tag{3.8}$$

$$\mathcal{L}_{critic}(\theta_1, \theta_2) = \lambda_{Q} \sum_{i=1,2} \mathcal{L}_{Q}(\theta_i) + \lambda_{L2} \sum_{i=1,2} \mathcal{L}_{L2}(\theta_i) \tag{3.9}$$

El pseudo-código de como se establecería el algoritmo propuesto se muestra en el algoritmo 9. De este algoritmo, se pueden verificar las dos fases mencionadas con anterioridad, la fase de pre-entrenamiento o entrenamiento *offline* y la fase de entrenamiento *online*. En la primera fase se actualiza el actor durante M épocas con la data recolectada desde un agente experto, para después, pasar a un entrenamiento general de T episodios usando la data recolectada del agente experto en una proporción de C_{exp} con respecto al *batch* de tamaño N y la data recolectada en el escenario de turno en una proporción C_{agent} con respecto al mismo *batch*.

En la figura 3.5 se muestra el diagrama de flujos del algoritmo TD3CoL.

Para analizar la complejidad computacional del algoritmo se debe recordar que el modulo de representación de estados se entrena de manera aislada al del controlador neuronal, acelerando el aprendizaje de este último. Esta separación de entrenamiento de ambos módulos

neuronales significa una ventaja considerable en los tiempos de computo y convergencia del modelo, con respecto a aquellas arquitecturas neuronales que consideran el entrenamiento de una red convolucional en conjunto al controlador neuronal [58]. En consecuencia a lo anterior, TD3CoL no considera la complejidad del entrenamiento del modulo de representación de estados, ya que no forma parte del entrenamiento del controlador neuronal, pero si debe considerar el costo por inferencia de este modulo. Dicho aquello, se tiene que la complejidad del modulo de representación de estados es lineal con respecto a la entrada $\mathcal{O}(N)$ en tiempo de inferencia. En cambio, para el controlador neuronal se tiene que el tiempo de inferencia de una sola observación contempla el computo de las 6 redes neuronales presentes: la red actor y su red objetivo, y las 2 criticas más sus correspondientes redes neuronales objetivas. Lo anterior, deriva en tiempos de computo dependiente de los datos de entrada y de las arquitecturas neuronales que se elijan para la solución del problema. Asumiendo que el vector de entrada de las redes neuronales tiene una dimensión z_{dim} y que las redes neuronales implementadas no tienen más de 3 capas ocultas, en donde cada una de estas tiene, a lo más, h_x neuronas por capa. Donde x hace mención a la red neuronal de donde pertenece esa capa. Por ejemplo, h_a corresponde a la cantidad de neuronas que posee una capa oculta del actor. Además, y por fines de simplificación de notación, se compactan las complejidades de las redes objetivas con sus respectivas redes neuronales, denotando $h_{(a,a')}$ y $h_{(c,c')}$ como la cantidad de neuronas que tiene una capa oculta del actor y la red objetivo del actor, y lo mismo con las redes neuronales del critico. No se hicieron distinción entre las redes neuronales normales y sus objetivos, ya que entre ellas la cantidad de parámetros no cambia, por lo que no sufre modificación la complejidad del algoritmo, a la vez que no se pierde generalidad en la expresión obtenida.

En las ecuaciones 3.10 y 3.11 se muestran las complejidades del algoritmo TD3CoL en inferencia y entrenamiento, respectivamente.

$$T_{cont}^{inference} = \mathcal{O}(N + z_{dim}h_a + h_a^2) = \mathcal{O}(N + z_{dim}W_a^2) \tag{3.10}$$

$$\begin{aligned} T_{cont}^{training} &= \mathcal{O}(N + BL(z_{dim}h_{(c,c')} + h_{(c,c')}^2 + z_{dim}h_{(a,a')} + h_{(a,a')}^2) + B\frac{L}{w}(z_{dim}h_a + h_a^2)) \\ &= \mathcal{O}(N + BLz_{dim}(W_{(c,c')}^2 + W_{(a,a')}^2) + B\frac{L}{m}z_{dim}W_a^2) \end{aligned} \tag{3.11}$$

En donde W_x es la cantidad total de neuronas en la red neuronal x, B es la cantidad de ejemplos usados para el entrenamiento, L es la cantidad de épocas por la cual se entrena el controlador y m es la frecuencia de actualización de la red neuronal del actor y las redes objetivos. En las ecuaciones anteriores, se simplifico la sumatoria que corresponde a los productos matriciales ejecutados en las sucesivas capas ocultas: $\sum_{i=1}^{H-1} h_i h_{i+1}$ por h^2. Esto se debe a que se asume, una cantidad pequeña de capas ocultas, lo cual permite: $\mathcal{O}(\sum_{i=1}^{H-1} h_i h_{i+1}) = \mathcal{O}(h^2)$. Dado lo anterior, y nuevamente por las propiedades de la función $O(\cdot)$, se puede asumir una cota superior establecida por el cuadrado de la cantidad total de neuronas en la red neuronal en las ecuaciones 3.10 y 3.11.

Por como se puede verificar de las ecuaciones anteriores, la complejidad computacional obtenida no es una expresión cerrada porque no depende únicamente de la cantidad de datos de entrada, sino que también de la cantidad de parámetros que posean las arquitecturas neuronales, cantidad de épocas de entrenamiento, largo del batch, etc. Ahora, si se quiere

estimar la complejidad en función de los datos de entrada, se puede asumir una arquitectura neuronal del controlador que sea fija. En las ecuaciones 3.12 y 3.13 se muestra las complejidad computacional para entrenamiento e inferencia de TD3CoL considerando un controlador neuronal con W_a neuronas, un critico con W_c neuronas y redes objetivos con $W_{a'}$ y $W_{c'}$ neuronas.

$$T_{cont}^{training} = \mathcal{O}(N + C) = \mathcal{O}(N) \tag{3.12}$$

$$T_{cont}^{inference} = \mathcal{O}(N + z_{dim} W_a^2) = \mathcal{O}(N) \tag{3.13}$$

En la ecuación 3.12 se juntaron todas las constantes en la variable C para simplificar notación. Con lo cual se tiene que el algoritmo es lineal con respecto a los datos de entrada, o más bien, a la cantidad de píxeles que contenga la imagen de entrada. Aunque por lo general, en este tipo de problemas los datos de entrada mantienen dimensiones fijas, por lo que la complejidad del algoritmo depende exclusivamente de la cantidad de parámetros que posean las redes neuronales utilizadas. En ese caso, la complejidad del algoritmo en tiempo de inferencia corresponde a $\mathcal{O}(W_a^2)$, en donde, se asume una red neuronal del actor con una cantidad W_a de neuronas. Mientras que en el caso de entrenamiento se podría llegar a poseer una complejidad de $\mathcal{O}(W^3)$.

Algorithm 9 TD3CoL

Require: Ambiente Env, M pasos de pre-entrenamiento, T episodios de entrenamiento, $StepsLimit$ pasos limites, J pasos de entrenamiento, Constantes de ponderación de funciones de pérdida de TD3CoL λ_1, PolFreqUpdate frecuencia de actualización de la red actor y objetivos, y Base de datos de experiencias expertas $\mathcal{D}^E$

Ensure: Redes Actor y Críticas entrenadas

Inicializar red actor π_ϕ y redes críticas Q_{θ_1}, Q_{θ_2}, parametrizadas por ϕ, θ_1 y θ_2, respectivamente
Inicializar las redes objetivos: $\phi' \leftarrow \phi$, $\theta'_1 \leftarrow \theta_1$ y $\theta'_2 \leftarrow \theta_2$
Inicializar ruido correlacionado Ornstein-Uhlenbeck $X_{OU} = X(\mu, \theta, \sigma)$
Inicializar el replay buffer del agente $\mathcal{B}^A$ y el replay buffer del experto $\mathcal{B}^E$
Cargar demostraciones expertas $\mathcal{D}^E$ en $\mathcal{B}^E$
for i = 1: M actualizaciones de pre-entrenamiento **do**
    Llamar a UpdateStep(It=i, es_pre-entrenamiento=True)
end for
for Episodios de Entrenamiento i = 1:T **do**
    $s_0 = EnvReset()$
    while step < StepsLimit **do**
        Elegir acción acorde a la política $a_t = \pi_\phi(s_t)$
        Agregar Ruido Ornstein-Uhlenbeck $a_t = a_t + X_{OU}(i)$
        $s_{t+1}, r_t, d_t = Env(a_t)$
        Guarda la transición $(s_t, a_t, r_t, s_{t+1}, d_t)$ en $\mathcal{B}^A$
    end while
    for j = 1:J Pasos de entrenamiento **do**
        UpdateStep(It=j, es_pre-entrenamiento=False)
    end for
end for
function UpdateStep(It, es_pre-entrenamiento)
    if es_pre-entrenamiento **then**
        Muestrear de manera aleatoria N tuplas expertas (s_i^E, a_i^E, r_i^E, s_{i+1}^E, d_i^E)
    else
        Muestrear de manera aleatoria $N * C_{exp}$ tuplas expertas (s_i^E, a_i^E, r_i^E, s_{i+1}^E, d_i^E) desde $\mathcal{B}^E$
        Muestrear de manera aleatoria $N * C_{agent}$ tuplas de agente (s_i^A, a_i^A, r_i^A, s_{i+1}^A, d_i^A) desde $\mathcal{B}^A$
    end if
    Se computa las funciones de pérdida $\mathcal{L}_{BC}(\phi)$, $\mathcal{L}_Q(\theta_1, \theta_2)$, $\mathcal{L}_A(\phi)$, $\mathcal{L}_{L2}(\phi)$, $\mathcal{L}_{L2}(\theta_1)$ y $\mathcal{L}_{L2}(\theta_2)$
    Se actualiza la redes críticas de acuerdo a la ecuación (3.9)
    if It % PolFreqUpdate == 0 **then**
        Se actualiza la red actor de acuerdo a la ecuación (3.8)
        Se actualiza la red objetivo del actor:
        $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
        Se actualiza las redes objetivos de los críticos:
        $\theta'_1 \leftarrow \tau\theta_1 + (1 - \tau)\theta'_1$
        $\theta'_2 \leftarrow \tau\theta_2 + (1 - \tau)\theta'_2$
    end if
end function

3.7. Configuración experimental

La toma de muestras por parte de un agente experto vendrá dada por la manipulación del agente por parte de un humano y tendrá la misión de controlar al vehículo para que pueda llegar al destino sin colisión alguna. La cantidad de exo-vehículos en ruta como sus posiciones se muestrearán de manera continua durante el tiempo de colección de muestras. La cantidad de exo-vehículos va desde 0 a 5 y la posición para cada uno de los exo-vehículos podrá ser cualquiera dentro del perímetro del escenario especificado en la imagen 3.1. Para esta fase de recolección de muestras expertas se arman dos bases de datos diferentes: una con muestras de exo-vehículos estáticos, es decir, que la posición de los exo-vehículos en ruta, para un episodio cualquiera, no cambia en el tiempo. Y otra base de datos que contenga muestras expertas de escenarios con exo-vehículos en movimiento. Este movimiento estará dado por coordenadas muestreadas de manera aleatoria. Por lo demás, el seguimiento de estas coordenadas aleatorias estará dado por un controlador PID que se encargará de seguir los *waypoints* establecidos para el exo-vehículo en cuestión. Cabe mencionar también que, en caso de que exista una colisión, la situación se reinicia en el mismo punto de partida. En la figura 3.6 se muestran estos escenarios de entrenamiento. El cuadrado rojo muestra el objetivo al cual debe llegar el ego-agente sin ninguna colisión.

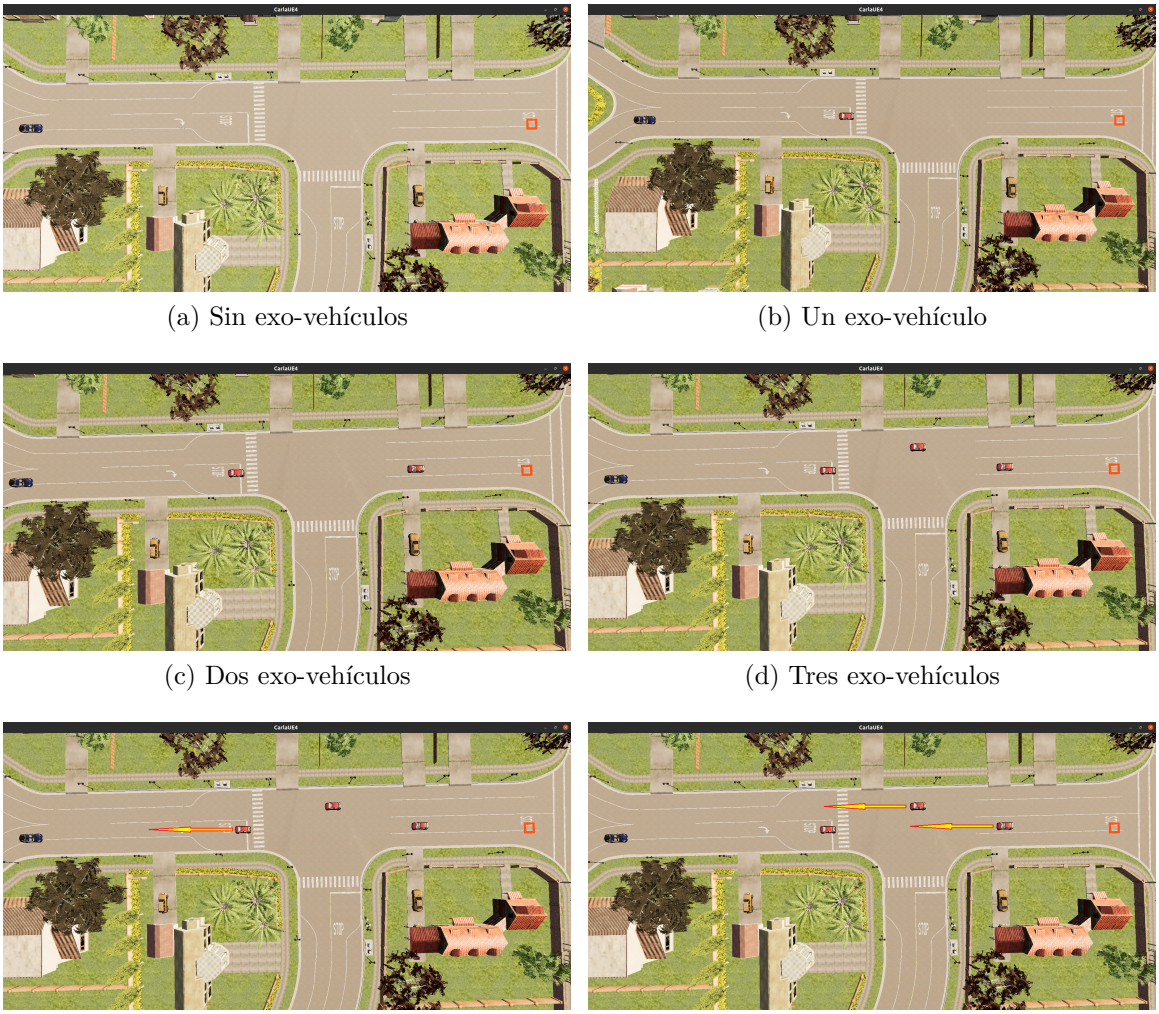
Las muestras obtenidas fueron 100 episodios de escenarios con exo-vehículos estáticos y 100 episodios de escenarios con exo-vehículos con movimientos aleatorios. Ambas bases de datos suman 15Gb de datos de imágenes semánticas más las señales de control de conducción tal como *throttle* y *steering*; además de estadísticas de conducción tal como la posición, velocidad, dirección del agente y exo-agente.

Antes de entrenar el controlador, se debe entrenar el modelo VAE con las imágenes tomadas en la recolección de experiencias de un experto. El *Encoder* y *Decoder* de este modelo están conformados por cuatro capas convolucionales de 32, 64, 128 y 256 canales, considerando un tamaño de kernel de 4 y un *stride* de 2. Además de hacer uso de una función de activación $ReLU$ entre capas. En el caso del *Decoder* el orden de las capas convolucionales es inversa. Las capas lineales que modelan la media, desviación estándar y entrada al *Decoder* tienen dimensiones acordes a los requerimientos de las capas convolucionales y una dimensión de 128 para el vector latente z.

Sé entreno el modelo VAE durante 1000 épocas con una función de pérdida MSE con reducción suma, $\beta = 1$, radio de aprendizaje inicial en 0.001 y con una reducción a las 500 épocas de 0.1, y un optimizador Adam [78] con *weight-decay* de 0.05.

Para el controlador se considera una arquitectura de dos capas lineales ocultas de 512 y 128 neuronas. Como función de activación se ocupa $ReLU$ para las capas ocultas y una función de activación de salida *Tanh*. Tanto el actor como el crítico poseen la misma arquitectura, diferenciándose únicamente en la entrada. En el caso del primero, hace uso del vector latente que es salida del *Encoder* más el *steering* y *throttle* del *timestamp* pasado. Mientras que el crítico hace uso de lo anterior más el *steering* y *throttle* actual.

Para entrenar el modelo del controlador se elige un escenario fijo. Aunque la cantidad de exo-agentes en ruta no es fija ni tampoco su movilidad dentro de la misma. Para verificar que



(a) Sin exo-vehículos

(b) Un exo-vehículo

(c) Dos exo-vehículos

(d) Tres exo-vehículos

(e) Tres exo-vehículos con uno en movimiento

(f) Tres exo-vehículos con dos en movimiento

Figura 3.6: Escenarios para el entrenamiento y validación de los distintos algoritmos de aprendizaje. El ego-agente es el vehículo azul ubicado al costado izquierdo de los distintos escenarios, mientras que los exo-agentes son los vehículos rojos que aparecen en distintas ubicaciones en la imagen. Las flechas de color amarillo reflejan el movimiento que seguirá un exo-agente durante el entrenamiento.

el modelo aprende de distintos obstáculos presentes en ruta, se entrena al mismo en distintas situaciones que contiene una, dos, tres y cuatro exo-vehículos en ruta, todos estáticos. Por lo demás se entrena al agente en dos situaciones que contemplan a exo-vehículos en movimiento. Las distintas situaciones de entrenamiento se pueden visualizar en la figura 3.6.

Sé entreno el modelo del controlador usando 1000 actualizaciones de pre-entrenamiento con experiencia experta. En esta fase se considera un entrenamiento *Behavior-Cloning*, por lo cual se cumple que $\lambda_A = 0$ y $\lambda_{BC} = 1$. Además de utilizar *Prioritized Replay Buffer* [65] con un tamaño de 20000 muestras, $\alpha = 1$, $\beta = 1$ y un *batch* de tamaño 64. Mientras que para el optimizador se emplea *Adam* [78]. El radio de aprendizaje tanto de los críticos como

del actor es de 10^{-3}.

Para la fase *online* de entrenamiento se sigue ocupando una configuración similar, cambiando las ponderaciones de las funciones de pérdida a: $\lambda_{BC} = 0.1$ y $\lambda_A = 0.9$. La actualización del agente se hace a partir de la décima época y se considera solo un paso de actualización por época. Esto con el objetivo de acumular muestras de este nuevo escenario sin la necesidad de entrenar en este nuevo *batch* de experiencias y seguir con muestras expertas, además de no desviar en demasía la política conseguida en las fases de pre-entrenamiento, sino que calibrar la política de conducción, aprovechando el conocimiento a priori que se tiene. Para ambas fases de entrenamiento se decide usar una constante de regularización $\lambda_{L2} = 10^{-5}$.

La actualización de la política se hace cada dos pasos, es decir, que las frecuencias de las actualizaciones de los críticos a las del actor están en un radio de 2:1. Además, que el radio de aprendizaje del actor disminuye a 10^{-4} y la de los críticos se mantiene igual con respecto a las épocas de pre-entrenamiento. También, y al igual como se hizo en [73], se hace uso de un *replay buffer* que guarde situaciones traumáticas o de colisión. La proporción de estas experiencias en la muestra de un *batch* son de 0.2 y el restante 0.8 se reparte en un 0.25 para experiencias expertas y un 0.75 para experiencias actuales. Además, se hace uso de ruido correlacionado *ornstein uhlenbeck* con parámetros $\mu = 0$, $\theta = 0.6$, $\theta_{max} = 0.4$ y $\theta_{min} = 0.005$ durante 350 épocas de entrenamiento. Con el objetivo de minimizar la variabilidad en el entrenamiento del agente y no presentar comportamientos de divergencia, se hace uso de un *scheduler* que a cada 350 pasos de actualización pondere el radio de aprendizaje del actor y de los críticos por 0.01.

Para el testeo de las diferentes políticas obtenidas desde el entrenamiento, se establecen 7 pruebas diferentes a superar. Estas pruebas se enumeran a continuación:

1. **Static**: Escenario con una cantidad de exo-vehículos estáticos muestreados de manera aleatoria desde 0 a 7.

2. **2Cars1LeadM**: Escenario con dos exo-vehículos, uno estático y otro con movimiento recto y adelantado del ego-agente (*Lead Exo-Vehicle*). Al igual que el anterior escenario, las muestras de posiciones son aleatorias, en donde las muestras para el exo-vehículo adelantado están restringidas a la pista del ego-agente y con la misma dirección de este.

3. **2Cars2RandomM**: Escenario con dos exo-vehículos; ambos con un movimiento aleatorio en contra de la dirección del ego-agente.

4. **3Cars3RandomM**: Escenario con tres exo-vehículos; todos con un movimiento aleatorio en contra de la dirección del ego-agente.

5. **4Cars4RandomM**: Escenario con cuatro exo-vehículos; todos con un movimiento en contra de la dirección del ego-agente.

6. **3Cars2StraightM**: Escenario con tres exo-vehículos, uno estático y dos en movimientos rectos en contra de la dirección del agente y con una posición inicial aleatoria.

7. **3Cars3LeadM**: Escenario con tres exo-vehículos; todos con un movimiento recto y adelantado con respecto al agente.

Cada uno de estos escenarios se muestrean de manera aleatoria episodio a episodio, tratando de abarcar la mayor cantidad de configuraciones posibles para cada uno de los escenarios planteados. Esto ayuda a comprobar situaciones borde que se podrían dar en la realidad, y que, por lo general, no se tienen en cuenta en situaciones de emergencia debido a su baja probabilidad de ocurrencia en el mundo real. Esto último con el objetivo de obtener y verificar la política más robusta posible dado un modelo neuronal y paradigma de aprendizaje.

4. Resultados y análisis

4.1. Variational AutoEncoder

En esta sección se presentan algunos resultados obtenidos directamente del entrenamiento y posterior prueba del *Variational AutoEncoder*.

4.1.1. Entrenamiento

En la figura 4.1 se muestra el decaimiento de la función de perdida obtenida a partir del entrenamiento de la red VAE. En esta se puede verificar como en las primeras iteraciones se concentra el mayor decrecimiento de dicha función, mientras que en los subsecuentes pasos se mantiene decreciendo en menor medida.

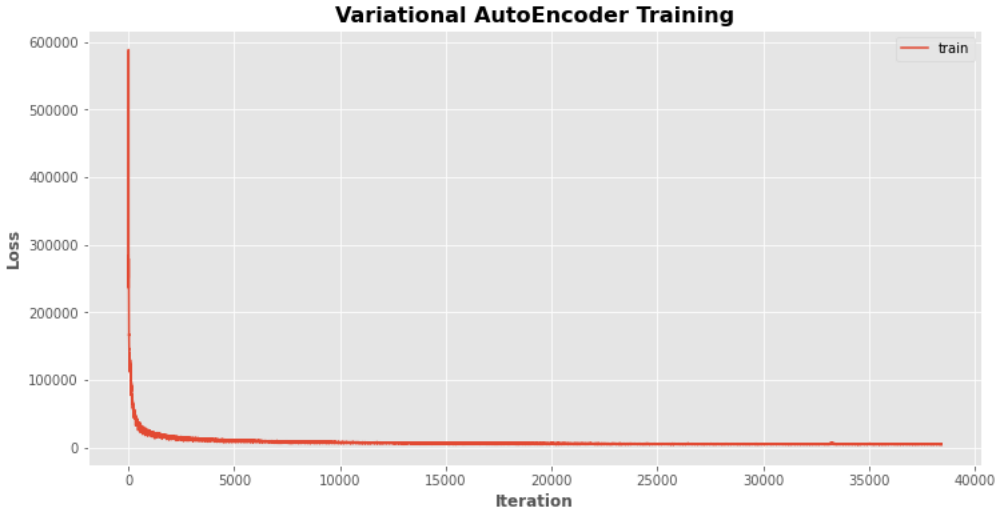
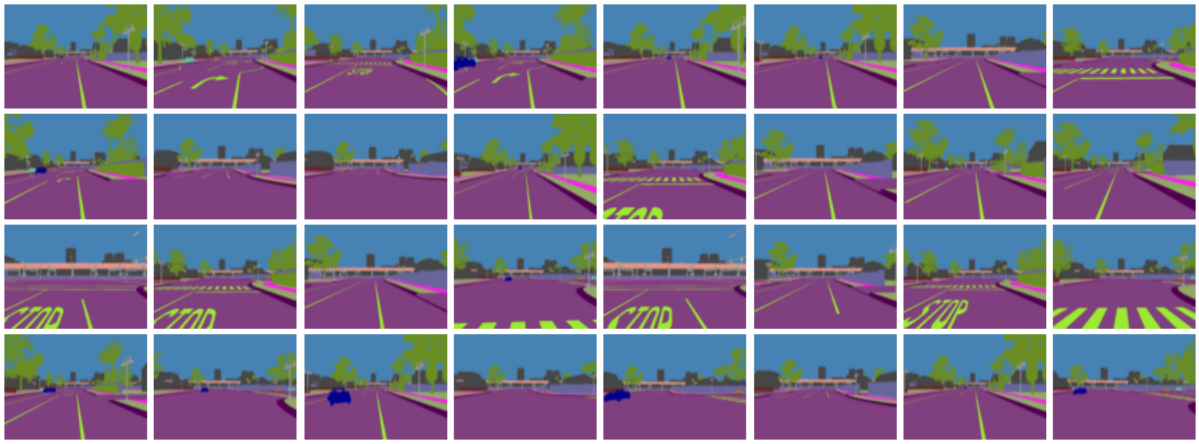


Figura 4.1: Función de perdida obtenida desde el entrenamiento de la VAE

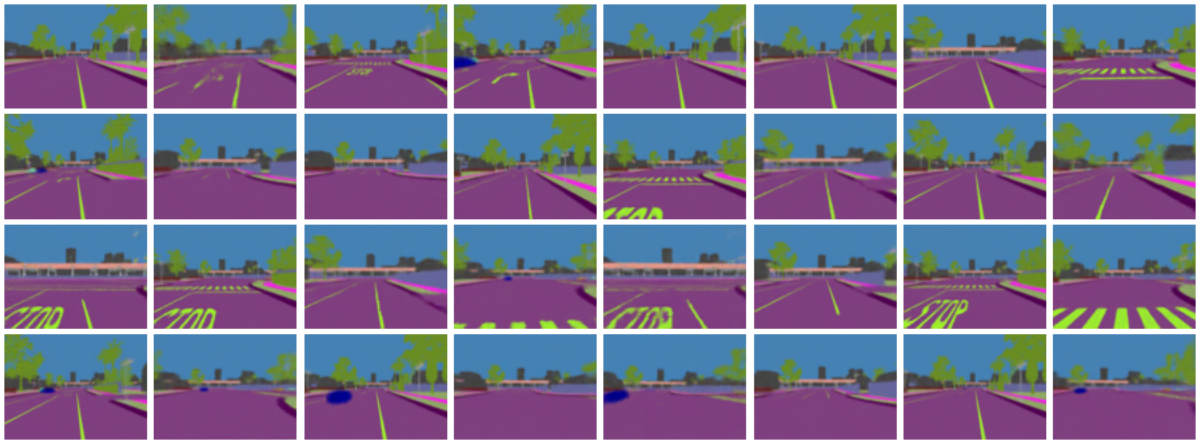
4.1.2. Reconstrucción

Para verificar que la red VAE está funcionando bien, se deben reconstruir algunas imágenes de prueba y verificar las reconstrucciones de tales imágenes de manera cualitativa.

En la figura 4.2.b se pueden visualizar algunas reconstrucciones que el modelo VAE obtiene dadas imágenes de entrada de la figura 4.2.a. Por cómo se puede comprobar de las imágenes, las reconstrucciones que son aprendidas por el modelo VAE tienen una similitud bastante alta con respecto a las imágenes que percibe de entrada. Este resultado permitirá a posteriori usar de manera eficiente la compresión de información y representaciones espaciales de las imágenes de entrada para entrenar el controlador propuesto en vectores de menor dimensionalidad. Aunque esto último se cumple solo si el módulo VAE aprende características desagregadas de las imágenes.



(a) Imágenes Semánticas tomadas desde CARLA



(b) Reconstrucción de imágenes por parte del modelo VAE

Figura 4.2: Comparativa entre imágenes tomadas desde la cámara frontal del vehículo y su posterior Reconstrucción

4.1.3. Generación

Para verificar que el modelo esté aprendiendo efectivamente características desagregadas, se debe verificar que el vector latente z esté codificando de manera desagregada las características de las entradas. Para comprobar aquello, se deben modificar las distintas dimensiones de este vector latente z por valores constantes y reconstruir las imágenes a partir de esta modificación.

En las figuras 4.3, 4.4, 4.5 y 4.6 se muestran las distintas reconstrucciones de una misma imagen para los valores de modificación -10, -5, 0, 5 y 10 para cada una de las dimensiones del vector latente. En donde la modificación 0 corresponde a la imagen original sin modificación alguna. Los valores en el eje y indican la dimensión que se está modificando, mientras que los valores del eje x, indican la cantidad que se añadió a la dimensión especificada.

En la figura 4.3 se muestran las reconstrucciones de las primeras 32 dimensiones del vector latente. Es posible verificar que varía de estas dimensiones codifican características únicas o, incluso, un conjunto de ellas. Por ejemplo, la tercera dimensión codifica la proximidad

que posee el agente con respecto al exo-vehículo, pudiéndose comprobar que al restar -10, el exo-vehículo acortaba su distancia al agente de manera notoria. Además de alejarse en caso contrario al sumarse 5 y 10. Por otro lado, en la dimensión 14 se codifica la posición del exo-vehículo que está más lejano al agente. En donde, al restarse, se conseguía trasladar el exo-vehículo más lejano a la vecindad del exo-vehículo más cercano al agente. Mientras que al sumarse, el exo-vehículo más lejano permanece, prácticamente, inmóvil. En la quinceava dimensión se puede constatar una característica similar, pero la traslación del exo-vehículo más lejano también hace generarse otro junto a los ya posicionados en la ruta. Aunque lo más interesante ocurre, cuando al sumarse las cantidades definidas, el exo-vehículo más lejano tiende a desaparecer de la ruta. En varias dimensiones se puede verificar como la vereda izquierda se modifica, ya sea cambiando algo de posición o distorsionando los colores de dicha sección, pero en la veintitresava dimensión se puede verificar un notorio traslado o generación de la vereda derecha al restarse 10.

En la figura 4.4 se muestran las reconstrucciones de las dimensiones 32 a 63 del vector latente. De esta figura se pueden distinguir algunas dimensiones que codifican características que son notorias a simple vista. Una de estas dimensiones es la dimensión 37, la cual codifica información sobre la escena que aparece al lado derecho de la imagen. En esa imagen se puede verificar como al restar se genera un árbol al lado derecho, mientras que al sumar aparece una pared. Ambos objetos no son parte de la escena actual, tal como se puede constatar en la imagen original.

Otra dimensión de la misma figura es la dimensión 38, la cual codifica información sobre la proximidad de un exo-vehículo imaginario, el cual se encuentra alrededor del exo-vehículo real. Esto se puede comprobar, ya que, al restarse cierta cantidad, se genera un exo-vehículo al lado izquierdo del exo-vehículo más cercano en la imagen. Esta misma información se codifica en la dimensión 51, sin embargo, en esta dimensión se codifica la información de un posible exo-vehículo al lado derecho del exo-vehículo más próximo. Por último, la dimensión 60 codifica la información del exo-vehículo más cercano, pudiéndolo hacer más presente en la imagen, coloreando la imagen de una manera más notoria, o haciéndolo desaparecer al sumar cierta cantidad.

En la figura 4.5 se presentan las reconstrucciones que se obtuvieron al modificar las dimensiones 64 a 95 del vector latente. Entre las dimensiones que se pueden mencionar en esa figura están las dimensiones 65 y 68. Ambas dimensiones codifican información sobre los árboles que están a los extremos laterales de la ruta, haciéndolos desaparecer o aparecer. Otra dimensión a destacar, es la dimensión 95 la cual codifica información sobre el movimiento lateral del exo-vehículo más cercano. A medida que se aumenta la cantidad a sumar, este vehículo se traslada en la escena hacia la derecha de la ruta.

En la figura 4.6 se presentan las últimas 32 dimensiones que se modificaron del vector latente. Una de las dimensiones que destaca en esta figura es la dimensión 102 que codifica la información de ambos lados de la ruta, generando una muralla y extendiendo la vereda más allá de lo que realmente abarca dicha estructura, además de hacer desaparecer ambas veredas si se resta cierta cantidad. Otra dimensión a destacar de esta figura es la dimensión 115 la cual codifica información sobre el fondo de la imagen. En la figura se puede verificar como el fondo toma una mayor proporción de la imagen a medida que se resta una mayor cantidad

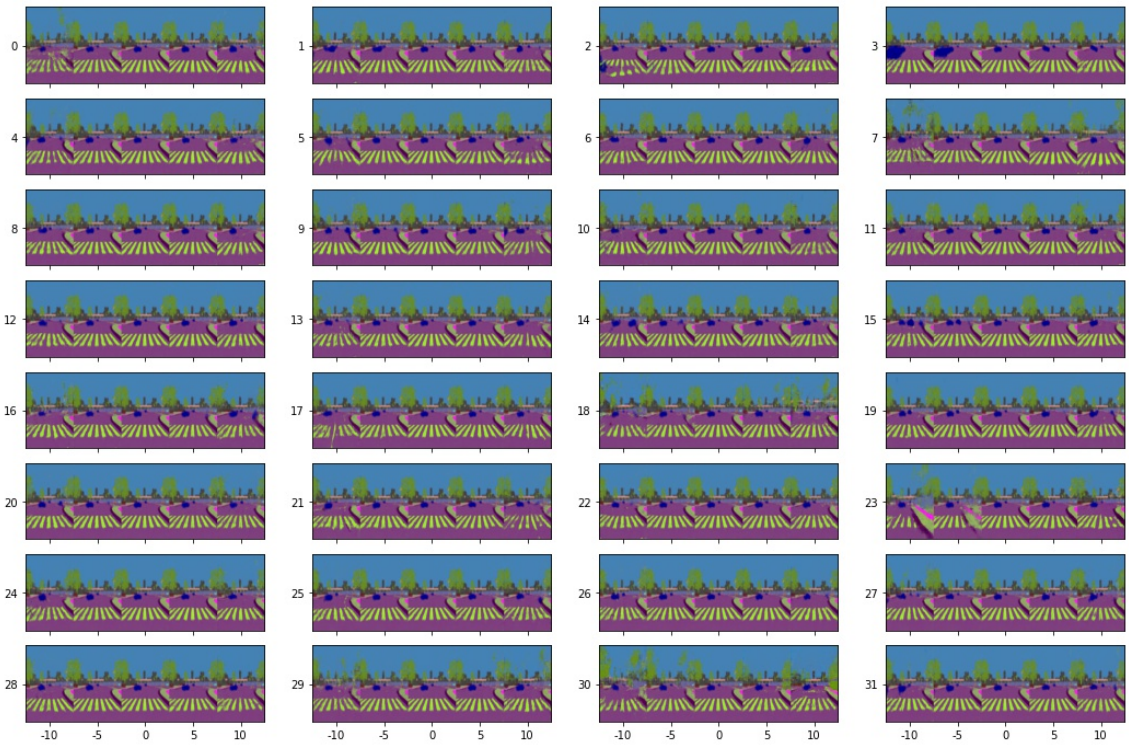


Figura 4.3: Reconstrucciones generadas a partir de una imagen de prueba para 5 distintos valores de distorsión para las dimensiones 0-31

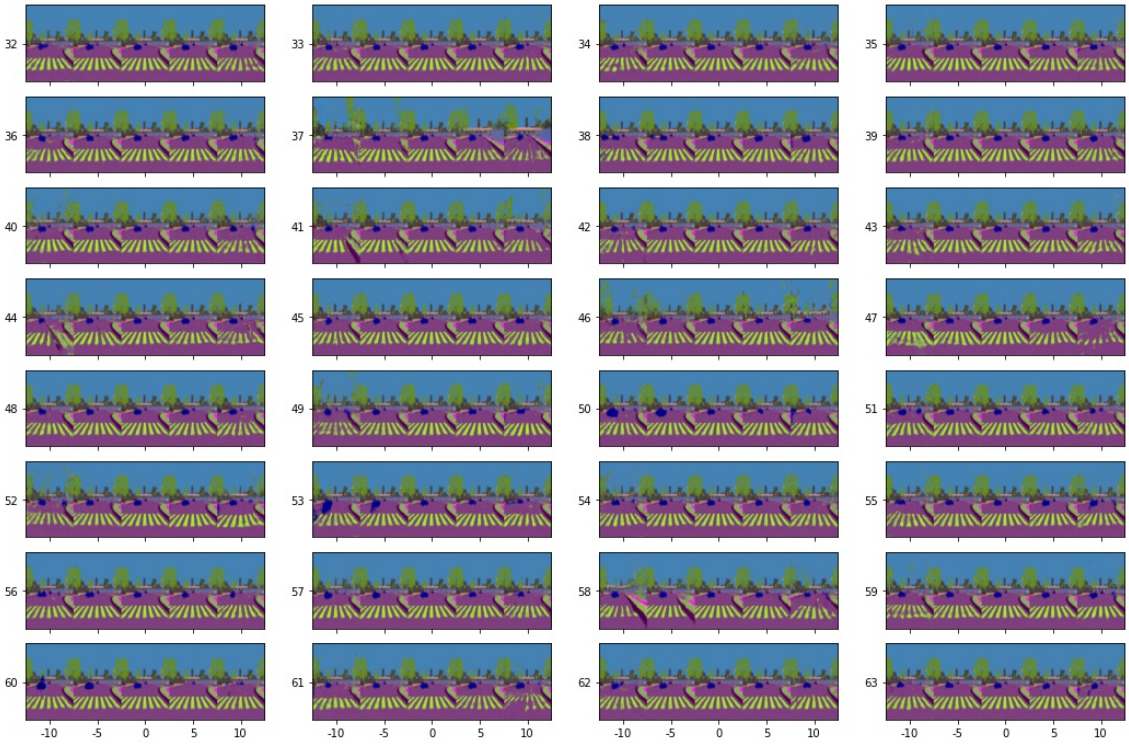


Figura 4.4: Reconstrucciones generadas a partir de una imagen de prueba para 5 distintos valores de distorsión para las dimensiones 32-63

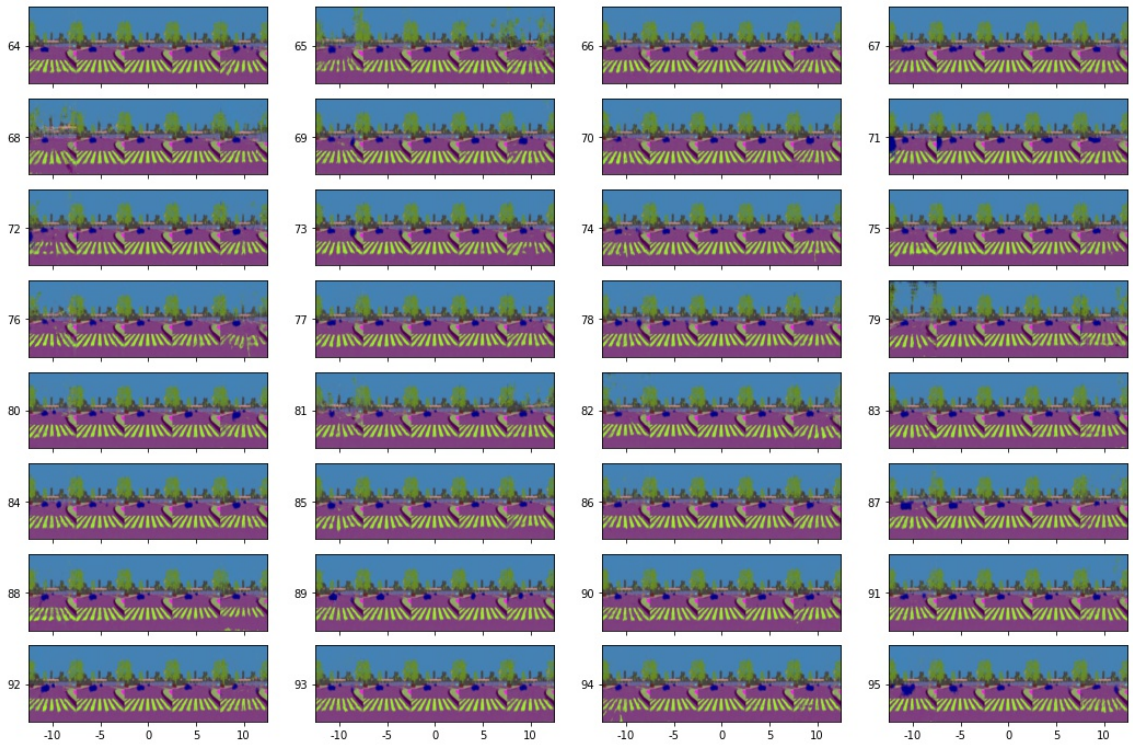


Figura 4.5: Reconstrucciones generadas a partir de una imagen de prueba para 5 distintos valores de distorsión para las dimensiones 64-95

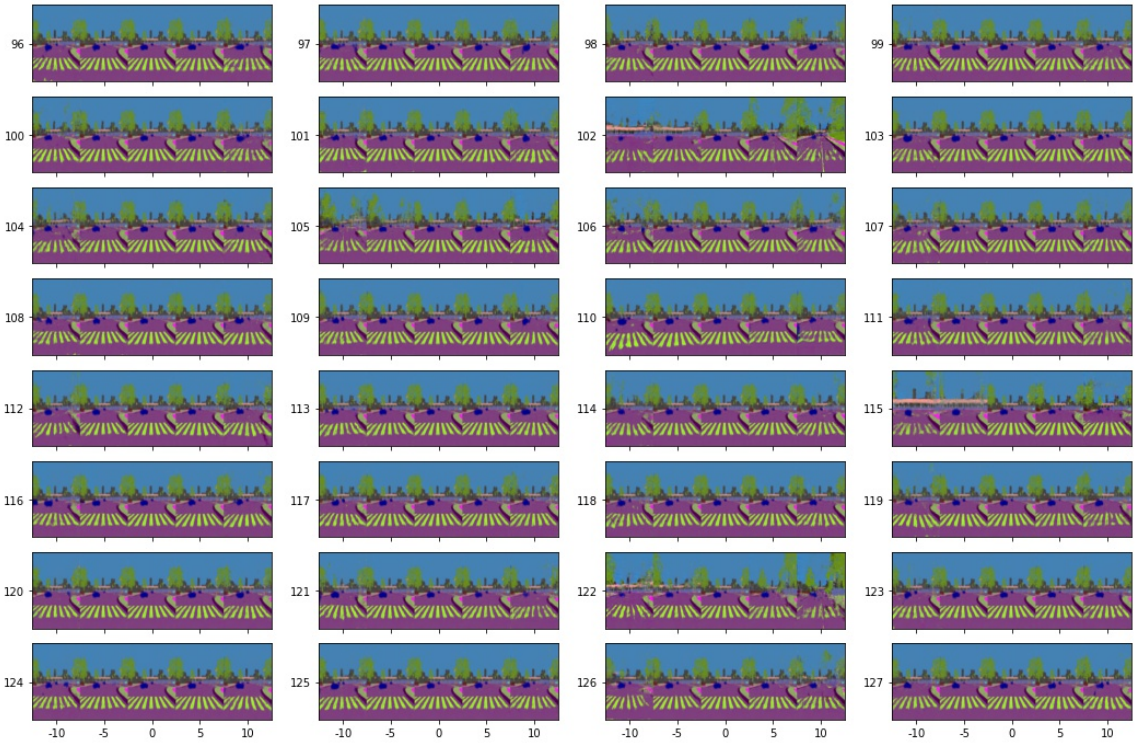


Figura 4.6: Reconstrucciones generadas a partir de una imagen de prueba para 5 distintos valores de distorsión para las dimensiones 96-123

a la dimensión mencionada. Y ocurre todo lo contrario al sumar una mayor cantidad. La dimensión 122 codifica una información similar, pero de manera más difusa y no tan notoria como lo hace la dimensión señalada anteriormente.

Debido a lo descrito anteriormente y a las figuras expuestas, se puede concluir, a base de un análisis cualitativo, que el modelo construido está aprendiendo características desagregadas de la ruta. Esto permitirá a posteriori utilizar este módulo como un extractor de características que optimizará el aprendizaje del controlador. Además de las figuras anteriores, también se adjunta un enlace[1] que contiene generaciones de imágenes en un formato *gif* para que se puedan ver transiciones de manera continua.

Ya que el entendimiento y la medición del modelo VAE no es la idea central de este documento, no se procederá a confirmar que las características desagregadas aprendidas por el modelo VAE sean correctas con base en algún análisis cuantitativo. Sin embargo, se insta a estudiar y evaluar un futuro trabajo teniendo en cuentas estudios sobre análisis cuantitativos en la medición de representaciones desagregadas, tal como lo hacen en [80], [81], [82], [83] y [84].

4.1.4. Explicabilidad

Además de confirmar la desagregación de variables, es interesante verificar la explicabilidad de las predicciones o generaciones realizadas por la red VAE. Por lo mismo, se recurre al método implementado en [79] para la visualización de mapas atencionales. El trabajo mencionado implementa un método basado en gradiente para obtener mapas atencionales. Resumidamente, ellos propagan el gradiente de cada dimensión del vector latente z hacia la última capa convolucional del encoder resultando en un mapa convolucional por dimensión del vector latente. Para finalmente, promediar estos y obtener un mapa atencional absoluto por imagen de entrada.

En la figura 4.7 se adjuntan algunos ejemplos de mapas atencionales que se generaron. En estas imágenes se puede verificar como la VAE pone atención en diferentes elementos claves de la ruta, tales como el paso de cebra, señalética de viraje y frenado, y exo-vehículos. Además de las líneas del centro de la ruta y características laterales de la ruta como lo son los edificios y árboles. Este resultado tiene relación directa con las imágenes generadas en las figuras 4.3, 4.4, 4.5 y 4.6, en donde se analizaban la codificación de información por parte de cada dimensión, la cual se debe a la metodología ocupada en ambos métodos, ya que ambos hacen uso del vector latente. Lo interesante de este segundo método, es que se necesitan menos imágenes que deben ser generadas para obtener un indicio de cuáles son las características de la imagen en donde la VAE intenta desagregar al momento de codificar una imagen de entrada.

4.2. Entrenamiento del controlador

Tal como se dijo en la sección anterior, se entrena el controlador en un ambiente de exo-vehículos estáticos, en donde la cantidad de exo-vehículos y sus posiciones iniciales se muestrean de manera aleatoria. Para esta etapa de entrenamiento, se considera muestrear desde cero a cinco exo-vehículos en cada episodio de entrenamiento.

[1] https://drive.google.com/drive/folders/12f-UmGNiBqg1UEqEtwj9ITEA1W1Ct4cn?usp=sharing

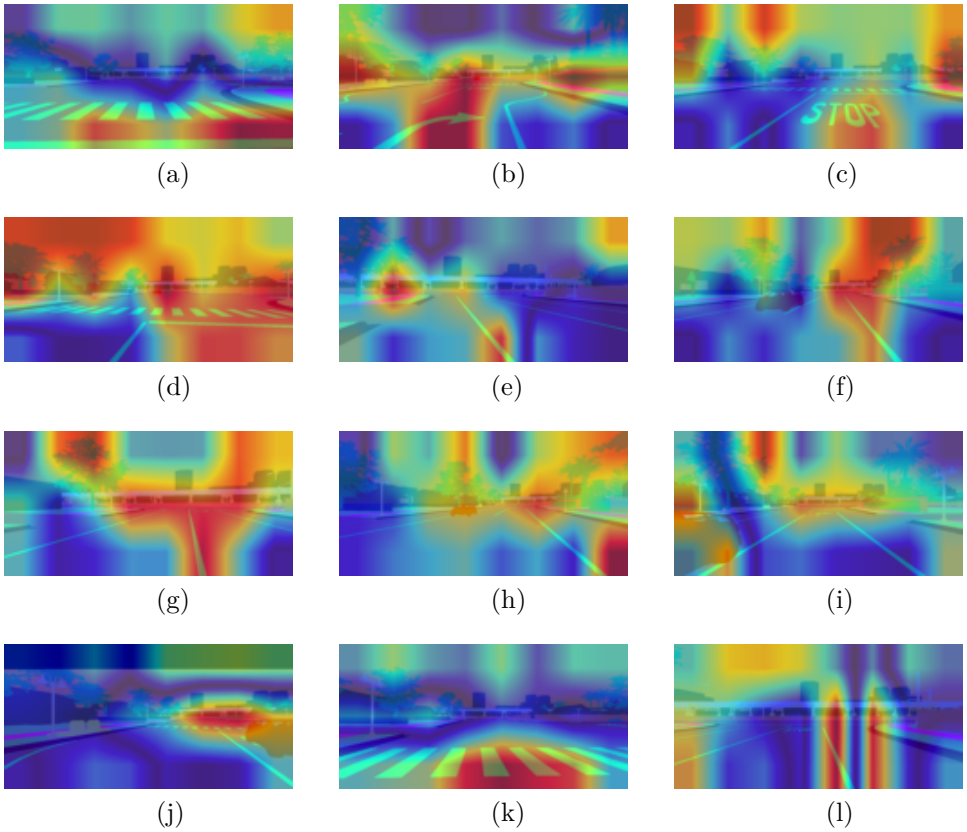


Figura 4.7: Mapas de atención generados a partir de la VAE

En la figura 4.8 se muestran el radio de éxito promedio y recompensa promedio a través de los distintos episodios de entrenamiento del controlador. Estas curvas fueran obtenidas a partir de los episodios de prueba, cuyos episodios no contemplan la adición de ruido correlacionado, testeándose el controlador cada 5 episodios de entrenamiento. En total fueron 500 episodios de entrenamiento y cada uno de ellos poseía una configuración diferente del ambiente.

Antes de cualquier análisis a realizar es importante señalar que por cómo están configuradas las recompensas del agente, una recompensa positiva significa, mayoritariamente, la llegada del agente a la línea de término de la pista, no significando necesariamente la llegada al objetivo planteado. Dicho lo anterior se puede verificar como los algoritmos de aprendizaje asistido demuestran el aprendizaje en los distintos escenarios aleatorios. Más en detalle, en la figura 4.8 se puede verificar como CoL y TD3CoL superan por amplio margen el desempeño del agente entrenado con DDPG. Además, TD3CoL supero a su par CoL en las métricas de radio de éxito promedio y recompensa promedio.

Un detalle interesante para mencionar es la baja en el rendimiento del algoritmo TD3CoL para los primeros 150 episodios de entrenamiento, viéndose mermada el rendimiento del controlador obtenido en la fase de pre-entrenamiento. Aunque este comportamiento se ve corregido por la etapa de *Reinforcement Learning* subiendo desde un radio de éxito promedio de 0.55 hasta 0.9 y de -100 hasta 600 en recompensa promedio. Por otro lado, CoL no sufrió esta caída en los primeros episodios, sin embargo, su rendimiento fue en picada desde

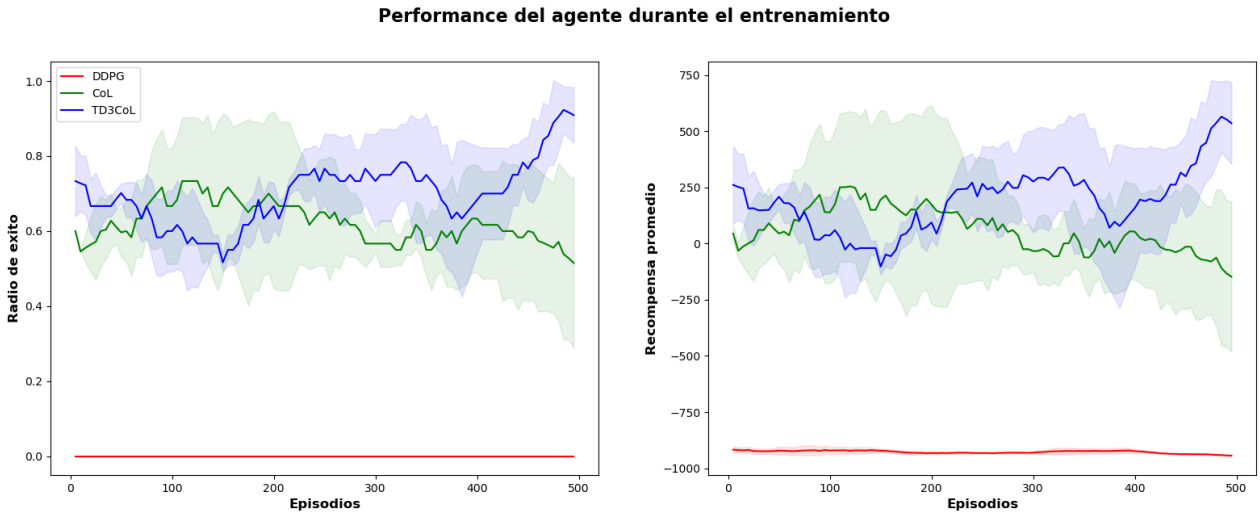


Figura 4.8: Radio de éxito y Recompensa promedio obtenida durante en el entrenamiento del agente

el centésimo episodio, desde un radio de éxito de 0.7 hasta 0.5 y de 250 a -50 en recompensa promedio. En cuanto a DDPG no se pudo apreciar evolución alguna.

El desempeño de DDPG en la evaluación realizada fue prácticamente nula en ambas métricas. Esto da a entender que el agente no aprendió, prácticamente, nada a lo largo del entrenamiento y no pudo llegar al objetivo planteado en el ambiente. Sin embargo, y debido a que el entrenamiento considera configuraciones aleatorias episodio a episodio, no es posible saber si DDPG no aprende, absolutamente, de ninguna configuración del ambiente.

Independiente del entrenamiento anterior, se decide entrenar al agente en distintas situaciones fijas para comprobar la evolución de la política de aprendizaje a lo largo de los episodios de entrenamiento en escenarios fijos. Esto último se debe a que el paradigma anterior no nos permite verificar y comprobar como el agente aprende de los distintos escenarios a los cuales se está viendo enfrentado. Los escenarios fijos que se implementaron para este entrenamiento son seis y estos se muestran en la figura 3.6. La configuración de estos escenarios son los siguientes: Sin exo-vehículos, un solo exo-vehículo, dos exo-vehículos, tres exo-vehículos, tres exo-vehículos con uno de ellos con movimiento longitudinal (flecha amarilla) y tres exo-vehículos con dos de ellos con movimientos longitudinales. Además, se muestra un cuadrado rojo, el cual es el objetivo a alcanzar. Mismo objetivo que se ocupó para el entrenamiento anterior.

En las figuras 4.9 y 4.10 se muestran los resultados obtenidos en el entrenamiento realizado en los escenarios fijos. De estas figuras, se puede verificar como el algoritmo de aprendizaje propuesto TD3CoL resolvió todas las situaciones planteadas. Su par CoL, si bien resolvió 5 de las 6 situaciones, registró peor desempeño que el algoritmo propuesto en los 6 escenarios. Mientras que DDPG resolvió 3 de las 6 situaciones, siendo su desempeño mucho menor a los algoritmos de aprendizaje asistido.

En la primera situación de entrenamiento se puede verificar como los tres algoritmos de

aprendizaje aprendieron a conducir rápidamente. Tanto CoL como TD3CoL tenía aprendida esta tarea antes de iniciar el entrenamiento general, mientras que DDPG le basto solo un par de episodios.

En la segunda situación, el agente entrenado por el algoritmo DDPG logró converger únicamente a un valor de recompensa promedió 0 y un valor de radio de éxito 0.5. Por otro lado, CoL convergió a una recompensa de 300 y radio de éxito de 0.8. Mientras que TD3CoL tuvo un desempeño más estable, convergiendo a una recompensa de 550 y un radio de éxito de 0.83. Lo interesante de esta situación, es que ambos algoritmos de imitación, más perceptible en CoL que en TD3CoL, mejoraron su rendimiento con respecto a la política pre-entrenada, logrando converger a una mejor política que solucionase la situación del momento.

La tercera situación presenta un agente entrenado con DDPG que alcanza una convergencia de recompensa promedio de 50 y un radio de éxito de 0.4. A diferencia de la situación anterior, el aprendizaje es más patente y se logra apreciar que desde la época 150 empieza a mejorar su rendimiento. CoL presento un aprendizaje notorio a lo largo de los episodios de entrenamiento, convergiendo a una recompensa promedio de 300 y un radio de éxito promedio de 0.8. Mientras que TD3CoL fluctuó con el rendimiento obtenido en la fase de pre-entrenamiento, pero hacia el final logró converger hacia una recompensa promedio de 600 y un radio de éxito promedio de 0.98.

Para la cuarta situación de entrenamiento se puede verificar como DDPG ya no registra aprendizaje absoluto, sino que se mantiene en una recompensa promedio de -750 a lo largo del entrenamiento. Por su parte, los algoritmos de aprendizaje asistido lograron mejores rendimientos. Al igual que en la situación anterior, ambos algoritmos aprendieron a mejorar su rendimiento inicial obtenido en la fase de pre-entrenamiento. Donde CoL logró converger a una recompensa promedio de 250 y un radio de éxito de 0.7. Por su lado, TD3CoL convergió a una recompensa promedio de 500 y un radio de éxito promedio de 0.85.

Para la quinta situación de entrenamiento, situación que tenía el añadido de incluir uno de los exo-vehículos con movimiento, se pudo registrar, nuevamente, que DDPG no pudo aprender en lo absoluto de la situación que tenía en frente. A diferencia de las situaciones anteriores, ambos algoritmos de aprendizaje asistido lograron resultados notoriamente disímiles, viendo CoL mermado su desempeño a lo largo de la fase de entrenamiento, convergiendo a una recompensa promedio de -250 y un radio de éxito promedio de 0.4. Mientras que TD3CoL mantuvo un rendimiento plano a lo largo de la fase de entrenamiento, en cuanto al radio de éxito promedio se refiere, manteniendo el rendimiento conseguido en la fase de pre-entrenamiento durante los 1000 episodios, logrando converger a una recompensa promedio de 150 y un radio de éxito promedio de 0.8.

Finalmente, en el último escenario de entrenamiento, DDPG mantuvo su rendimiento obtenido en los dos últimos escenarios de entrenamiento, mientras que las políticas de entrenamiento asistido lograron mejores resultados. Particularmente, CoL tuvo una variabilidad de aprendizaje bastante alta a lo largo de los episodios de entrenamiento, llegando a converger a una recompensa promedio de 50 y un radio de éxito promedio de 0.62. TD3CoL fue algo más consistente con su aprendizaje, pero no pudo mantener o mejorar su rendimiento conseguido en la fase de pre-entrenamiento, llegando a una recompensa promedio final de

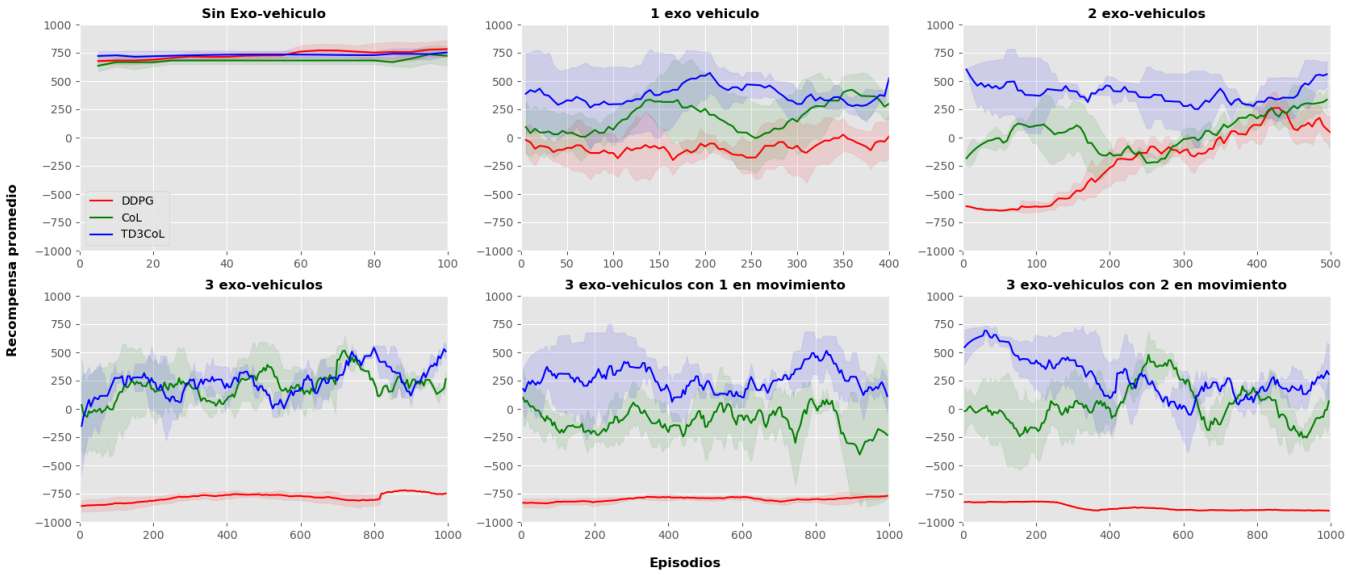


Figura 4.9: Recompensa promedio obtenida en los distintos escenarios de entrenamiento

300 y radio de éxito promedio de 0.8. Ambos resultados más bajos que los conseguidos al empezar el entrenamiento, que son de 520 en recompensa promedio y 0.85 en radio de éxito promedio.

Algo interesante a mencionar de estos dos entrenamientos, es el hecho de que DDPG si aprende de algunos escenarios, pero a medida que se complejiza la tarea no responde bien y diverge en su aprendizaje. Esto quiere decir que, si se pretende obtener una política de evasión de colisión con respecto a diferentes escenarios y que el agente aprenda a generalizar de ellos, DDPG no es buen algoritmo, tal como está configurado el problema. Y se hace necesarias demostraciones expertas para controlar y regularizar el entrenamiento del agente, tal como se demuestra con los algoritmos de aprendizaje asistido (CoL y TD3CoL). Aunque no es gratis, ya que se necesitan demostraciones expertas, cuyos datos no son fácil de conseguir y menos en la vida real. Estas conclusiones impactan directamente en el desempeño de DDPG en el entrenamiento realizado con escenarios aleatorios que se muestran en la figura 4.8. Esto debido a que el agente entrenado con DDPG demostró aprender, solo de algunas configuraciones de exo-agentes, por lo que intentar obtener una política con escenarios aleatorios que incluyen escenarios hasta 5 exo-agentes complejiza de sobremanera el entrenamiento para DDPG, a la vez, que se evalúa la política en escenarios aleatorios, muchos de ellos que no van a ser favorables para el conocimiento que acumulo el agente entrenado con DDPG.

Para verificar la evolución del aprendizaje del agente a lo largo de los episodios de entrenamiento, se deben considerar las figuras 4.11, 4.12 y 4.13 en donde se muestra la evolución en la posición y velocidad para la política de conducción durante los episodios de entrenamiento para los tres distintos algoritmos en las situaciones de un, dos y tres exo-vehículos en ruta, respectivamente. La situación expuesta corresponde al escenario con dos exo-vehículos estáticos. Los trazos azules simbolizan las posiciones del agente hasta llegar a la meta. Los

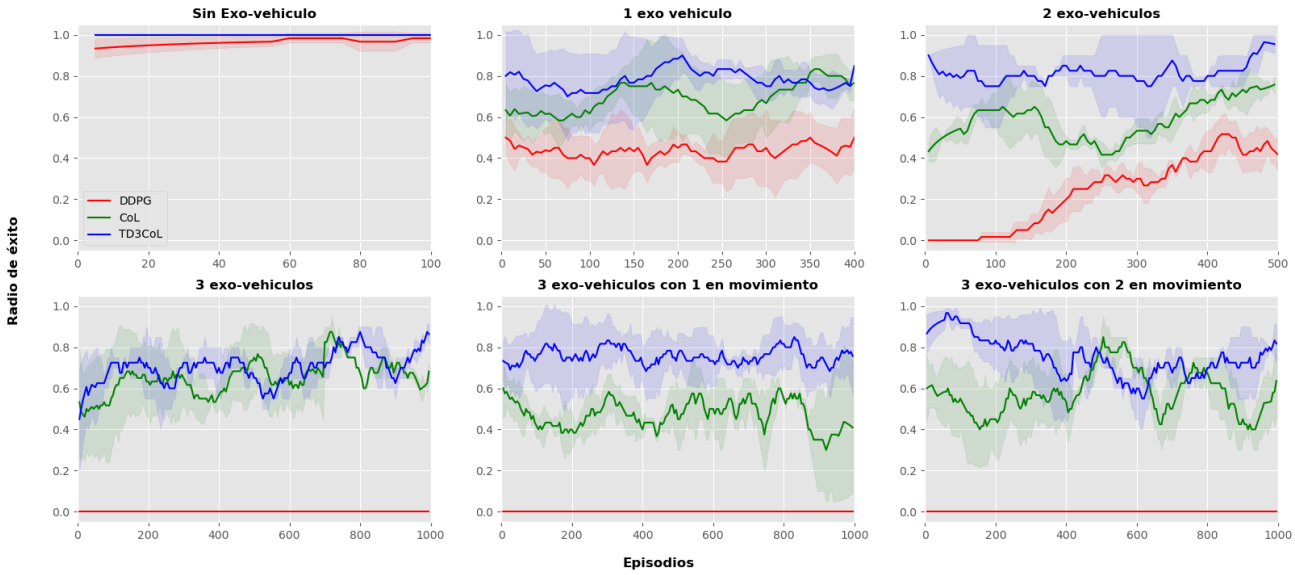


Figura 4.10: Radio de éxito para diferentes escenarios

trazos rojos y verdes simbolizan colisiones con exo-vehículos u otros obstáculos, respectivamente. Además, los colores se codifican de tal manera que posean más intensidad a medida que avanzan los episodios, por lo que las líneas más oscuras simbolizan episodios tempranos y los más intensos y coloridos codifican episodios más tardíos. Además, el rectángulo púrpura simboliza a los exo-vehículos en ruta. Con esto en mente, se puede verificar como la política de conducción converge hacia una ruta viable hacia el objetivo en los tres algoritmos probados a medida que el entrenamiento avanza.

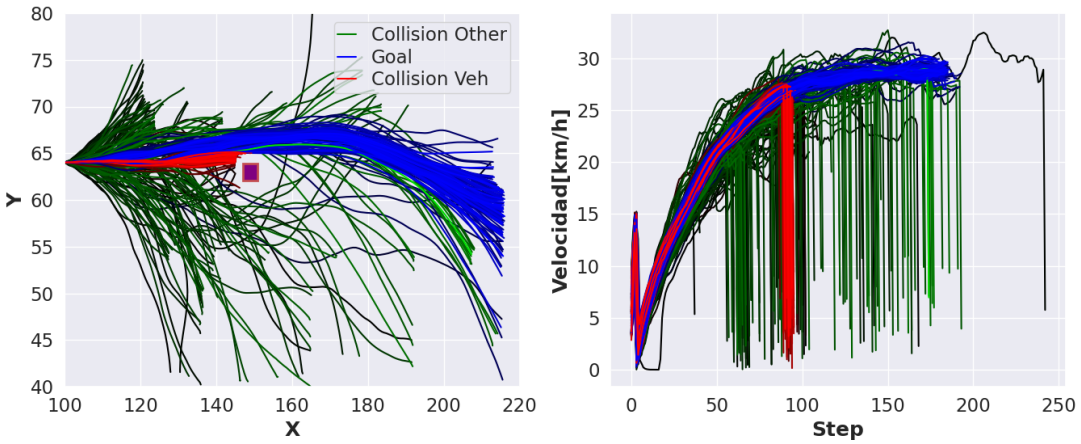
Particularmente, se puede verificar como en la figura 4.11 los tres algoritmos muestran la evolución de la trayectoria del agente a lo largo de los episodios de entrenamiento. Lo más interesante de esta figura es que la política obtenida mediante DDPG converge a una serie de trayectorias bien definidas y evidentes, además de tener episodios de colisión con el exo-agente en ruta para episodios finales. Además, la velocidad converge a una velocidad de 30 km/hr después de los 100 pasos. En el caso de CoL y TD3CoL ambos algoritmos muestran la convergencia de resolución del problema a diferentes trayectorias alrededor del exo-vehículo, alcanzando mayores velocidades a medida que avanzaba el entrenamiento.

En el caso de la figura 4.12 es posible verificar como se obtienen conclusiones similares a las obtenidas de la figura anterior. Se vuelve a verificar como DDPG converge a una sola trayectoria de solución, mientras que CoL y TD3CoL mantienen diferentes trayectorias en los distintos episodios de entrenamiento. Además de alcanzar mayores velocidades que DDPG.

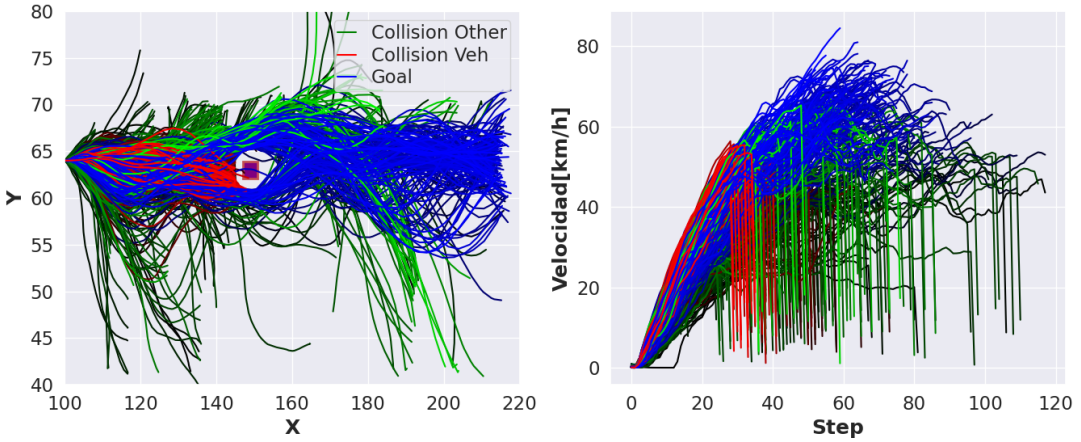
En cambio, en la figura 4.13 se verifican comportamientos diferentes a los vistos en las dos figuras anteriores. En esta ocasión, DDPG no muestra comportamientos en donde llegue a la meta, sino que puramente comportamientos en donde colisiona con algún obstáculo en el ambiente y con los exo-vehículos presentes en la ruta. También las velocidades de conducción son menores, bajando de los 30 $[km/h]$ a los 25 $[km/h]$. En el caso de los algoritmos de CoL y

TD3CoL, se vuelve a verificar las diferentes trayectorias de arribo a la meta, las velocidades de conducción bajan, lo cual es lógico debido a los obstáculos que debe enfrentar en la ruta.

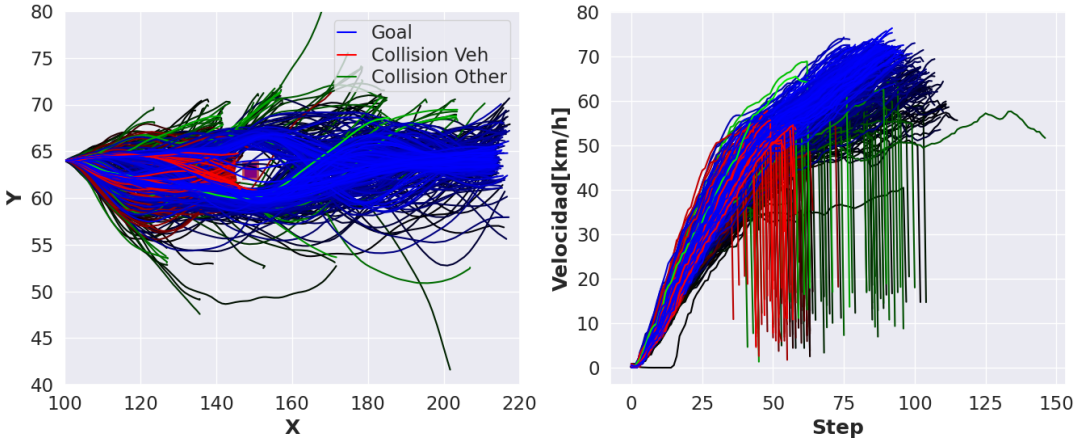
Basándose en lo expuesto en estas figuras, es posible verificar como los algoritmos de aprendizaje asistido convergen a distintas soluciones o trayectorias de llegada a la meta, lo cual es consecuencia del ruido correlacionado que afecta a la política de conducción en los episodios tempranos del entrenamiento, haciendo que la política sea robusta a una mayor cantidad de situaciones en ruta. Demostrando por lo demás la resiliencia de la política de manejo. También las mayores velocidades alcanzadas en los algoritmos de aprendizaje asistido demuestran que estos algoritmos poseen un mayor sentido del espacio.



(a) DDPG

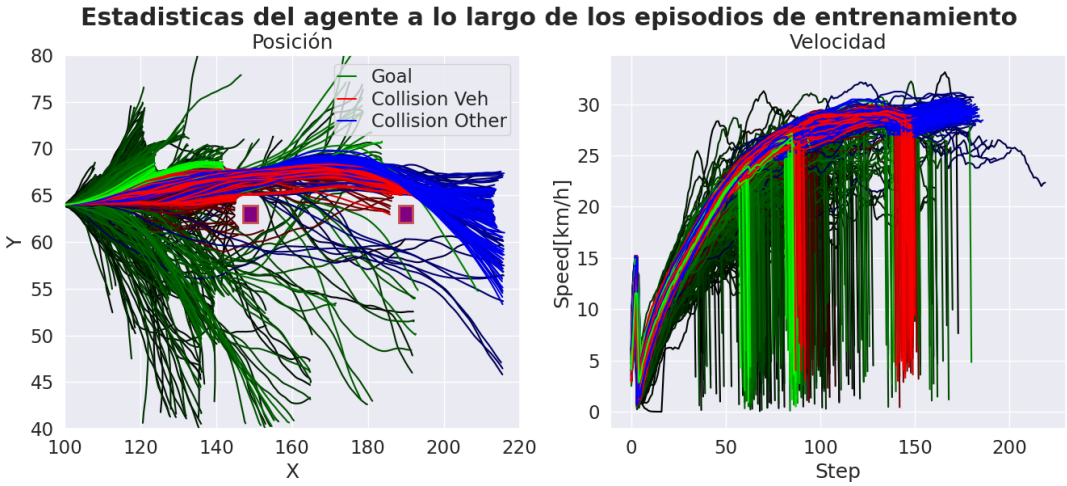


(b) CoL

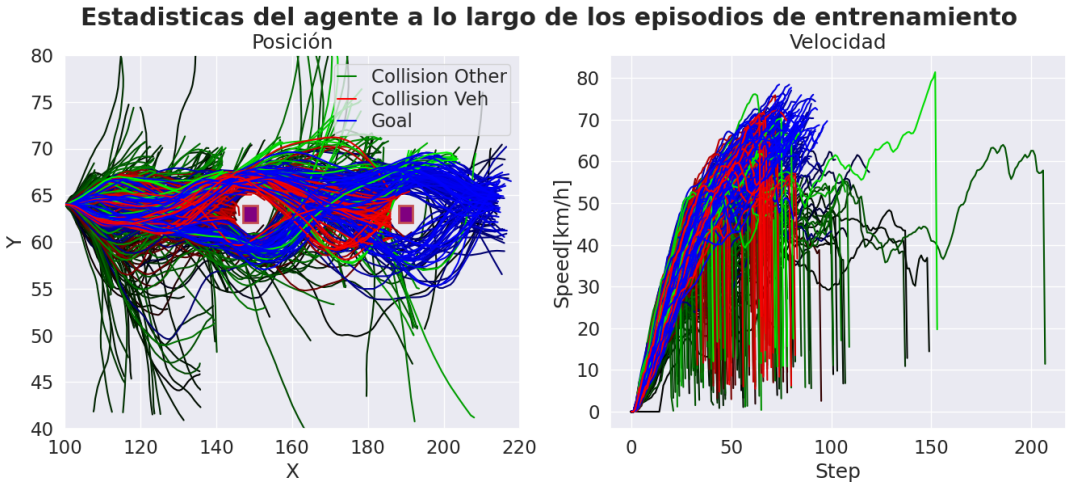


(c) TD3CoL

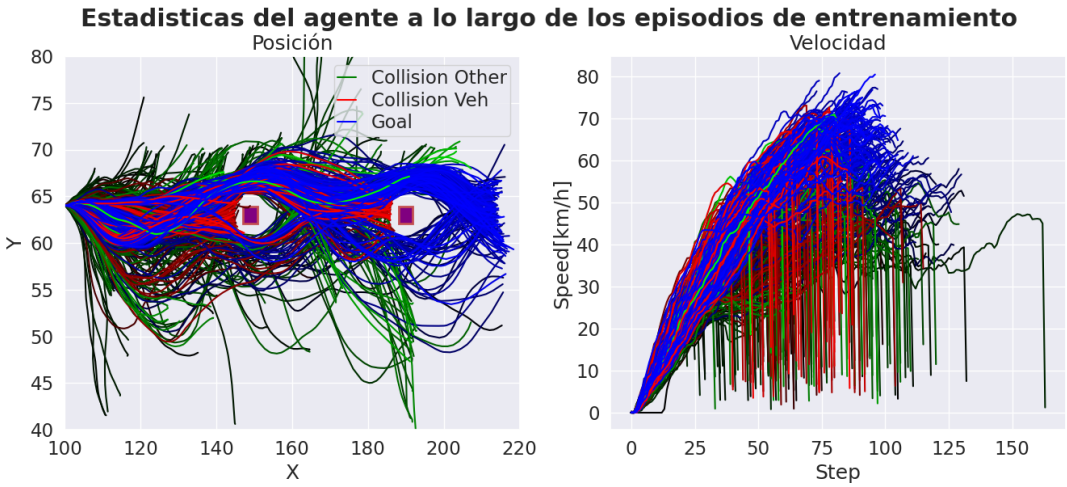
Figura 4.11: Evolución de la política de conducción a lo largo del entrenamiento para el escenario de 1 exo-vehículo estático.



(a) DDPG

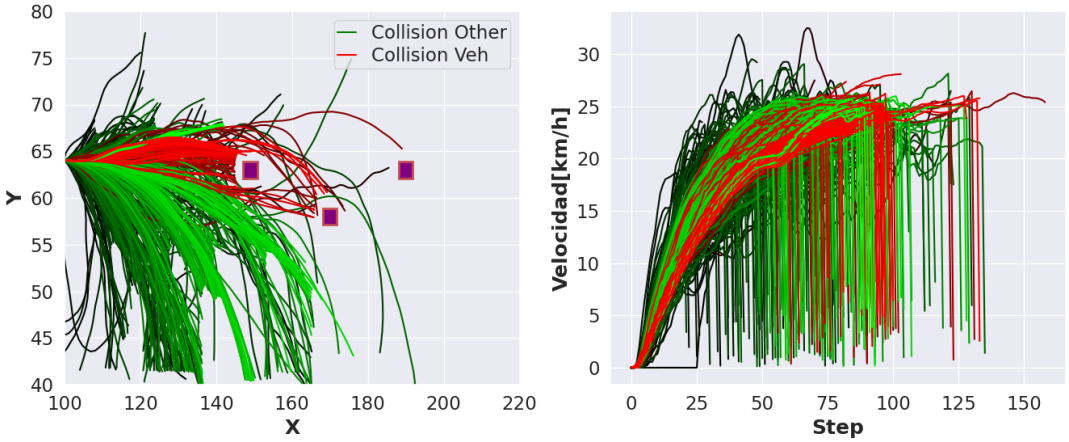


(b) CoL

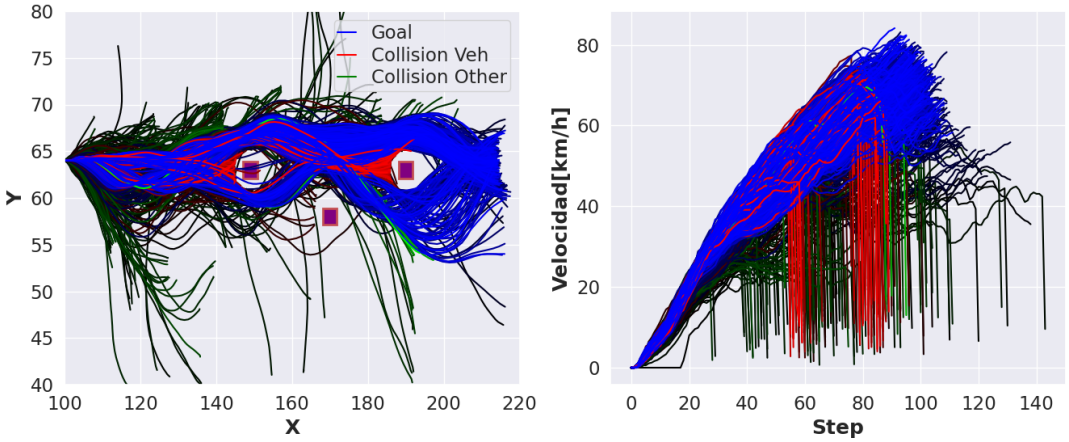


(c) TD3CoL

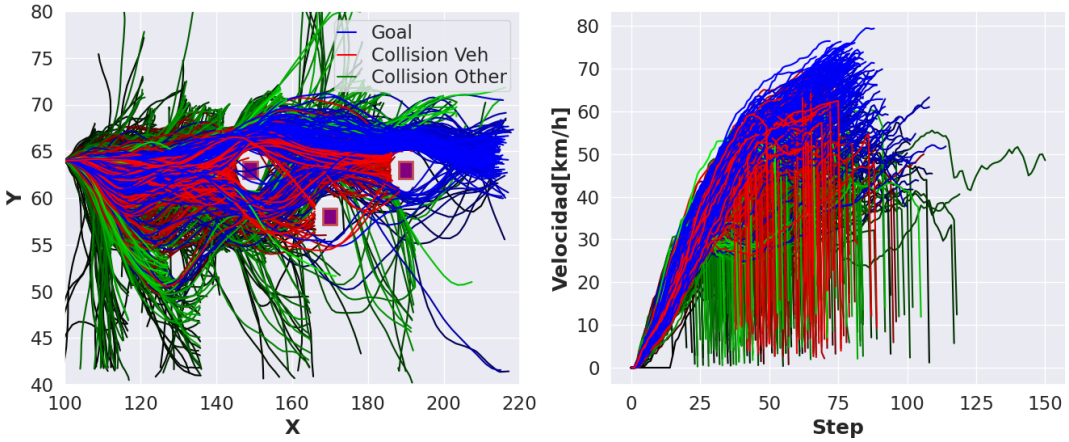
Figura 4.12: Evolución de la política de conducción a lo largo del entrenamiento para el escenario de 2 exo-vehículos estáticos.



(a) DDPG



(b) CoL



(c) TD3CoL

Figura 4.13: Evolución de la política de conducción a lo largo del entrenamiento para el escenario de 3 exo-vehículos estáticos.

4.3. Prueba de las políticas de conducción

Para evaluar la política obtenida por cada algoritmo de aprendizaje se establecen 7 tipos de prueba, las cuales fueron enunciadas y explicadas en la sección de metodología. Es importante mencionar que la política ocupada para evaluar el desempeño de cada algoritmo fue aquella que se obtuvo en el entrenamiento con escenarios aleatorios.

Antes de probar el sistema en los escenarios de pruebas propuestos más arriba, debemos verificar la variedad y completitud de posiciones que poseen los exo-vehículos dentro del ambiente propuesto manteniendo un muestreo aleatorio. En la figura 4.14 se muestran todas las posiciones iniciales (puntos rojos) de los exo-vehículos a lo largo de los 300 episodios de prueba para el escenario de exo-vehículos estáticos. En esta se puede comprobar como el espacio completo fue visitado, a lo menos una vez, en el escenario mencionado. Es conveniente mencionar que el muestreo de las posiciones, se realizó manteniendo coordenadas enteras, por lo que es fácil verificar patrones horizontales a lo largo de la ruta que son consecuencia de la escala del gráfico y del tipo de datos que se maneja, más que una intencionada distribución de las posiciones.

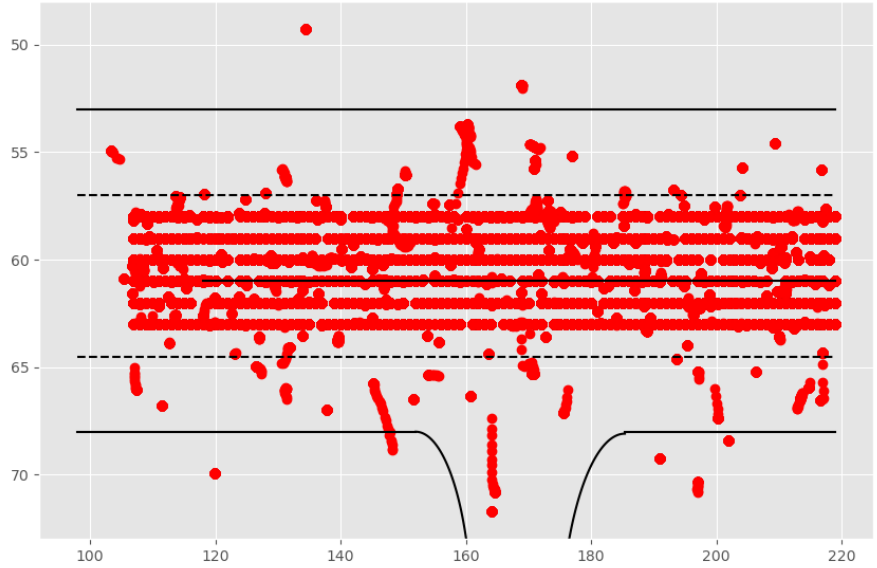


Figura 4.14: Diferentes posiciones adoptadas por exo-agentes para el Escenario estático

En la tabla 4.1 se muestran los radios de éxito, colisión en contra de exo-vehículos y en contra de algún otro obstáculo en ruta, tal como murallas, postes, árboles, etc. que obtuvieron los distintos agentes para los escenarios de prueba. De esta tabla se puede verificar como TD3CoL alcanza las mejores tasas de éxito en 5 de los 7 escenarios de prueba, mientras que DDPG y CoL alcanzan mejores rendimientos en solo un tipo de escenario. Específicamente, TD3CoL obtuvo los mejores rendimientos en los escenarios *Static*, *2Cars1LeadM* y *2Cars2RandomM*. Estos dos últimos escenarios eran de esperarse debido a que son situaciones dinámicas en donde se ubican menos exo-vehículos en ruta, haciendo ambos escenarios

		DDPG	CoL	TD3CoL
Static	Goal	0.38	0.8	**0.89**
	Exo-veh Collision	**0.61**	0.19	0.1
	Other Colllision	0.01	0.01	0.01
2Cars1LeadM	Goal	0.47	0.89	**0.93**
	Exo-veh Collision	**0.38**	0.1	0.04
	Other Colllision	**0.15**	0.01	0.03
2Cars2RandomM	Goal	0.65	0.73	**0.75**
	Exo-veh Collision	0.24	**0.25**	0.24
	Other Colllision	**0.13**	0.02	0.01
3Cars3RandomM	Goal	0.62	0.57	**0.63**
	Exo-veh Collision	0.31	**0.37**	0.33
	Other Colllision	**0.07**	0.06	0.04
4Cars4RandomM	Goal	**0.58**	0.46	0.49
	Exo-veh Collision	0.36	**0.46**	0.43
	Other Colllision	0.06	0.08	**0.09**
3Cars2StraightM	Goal	0.52	**0.57**	0.55
	Exo-veh Collision	0.35	0.37	**0.38**
	Other Colllision	**0.13**	0.06	0.07
3Cars3LeadM	Goal	0.33	0.59	**0.66**
	Exo-veh Collision	**0.57**	0.31	0.26
	Other Colllision	0.1	0.1	0.08

Tabla 4.1: Radios de exito y colisión para diferentes escenarios de prueba

más fáciles que los demás. En el caso del escenario *Static*, el que haya registrado un buen rendimiento no es tan evidente considerando que este escenario especificaba 7 exo-vehículos en ruta. Además, registró su peor rendimiento en el escenario *4Cars4RandomM*, siendo también el peor escenario para CoL, pero no así para DDPG. Esto último se debe a que la política obtenida mediante DDPG es más conservadora que las obtenidas por los algoritmos de aprendizaje asistido, haciendo que el agente conduzca mayoritariamente por el centro de la pista, arriesgando poco para evitar las colisiones. Justamente esta característica es la que ayuda a DDPG, ya que el movimiento aleatorio de los exo-vehículos estresa en demasía a las políticas que si intentan evitar las colisiones y terminan fallando debido a los movimientos aleatorios de los exo-vehículos, como los son los algoritmos de aprendizaje asistido. Aunque tampoco es algo de fiar, puesto que, según las cifras obtenidas a partir de las pruebas realizadas, la política obtenida desde el entrenamiento con DDPG no tienen la capacidad suficiente como política evasora de colisiones.

Por otro lado, TD3CoL registró los peores rendimientos en solo dos métricas de las 21 posibles, siendo estas las colisiones con otros objetos en el escenario *4Cars4RandomM* con un 0.09 de la trayectoria totales para tal escenario y un radio de colisión con exo-vehículos del 0.38 para el escenario *3Cars2StraightM*. Por su lado, CoL registró resultados similares a

	DDPG	CoL	TD3CoL
Radio de Éxito	0.51	0.66	**0.7**
Radio de Colisión a algún Exo-vehículo	0.4	0.29	**0.25**
Radio de Colisión para con algún otro objeto	0.09	0.05	**0.03**

Tabla 4.2: Promedio del radio de éxito, radio de colisión con algún exo-vehículo y algún otro objeto para los distintos escenarios aleatorios de prueba

los que presentó TD3CoL, pero siendo peores que este último en todos los escenarios, exceptuando en el escenario *3Cars2StraightM*.

En la tabla 4.2 se muestran los promedios de las métricas presentadas en la tabla 4.1 para cada algoritmo de aprendizaje. Esta tabla confirma que el algoritmo TD3CoL tiene mejor rendimiento que los otros dos algoritmos, presentando un radio de éxito de 0.7 en promedio para los 7 escenarios de prueba. Por su lado, CoL registró un promedio de 0.66 de radio de éxito promedio para los siete escenarios de prueba, mientras que DDPG registró únicamente un 0.51. En el caso del radio de colisión con algún exo-vehículo, TD3CoL registró un radio de 0.25, siendo menor que los 0.29 y 0.4 de CoL y DDPG, respectivamente. Por último, se tiene que TD3CoL obtuvo un radio de colisión con objetos que no son exo-vehículos de 0.03, mientras que sus pares CoL y DDPG obtuvieron radios de 0.05 y 0.09, respectivamente.

Además, en la tabla 4.3 se muestran las recompensas obtenidas por cada algoritmo de aprendizaje para cada escenario de prueba. Lo interesante de esta tabla es que muestra que DDPG tiene la mayor cantidad de recompensa para 3 de los 7 escenarios de prueba, siendo estos *2Cars2RandomM*, *3Cars3RandomM* y *4Cars4RandomM*. Este último escenario tiene lógica, ya que fue justamente el escenario en donde DDPG consiguió la mayor cantidad de trayectorias exitosas. Sin embargo, los escenarios con 2 y 3 exo-vehículos con movimientos aleatorios no fueron escenarios en donde DDPG obtuvo la mayor cantidad de trayectorias exitosas, sino que fue TD3CoL. Por ejemplo, en el escenario con 2 exo-vehículos con movimientos aleatorios, DDPG obtuvo, 1071 de recompensa promedio, mientras que TD3CoL logro, únicamente, 757 de recompensa promedio. Es decir, DDPG obtuvo una diferencia de recompensa de 314 con respecto a TD3CoL. Sin embargo, DDPG consiguió únicamente un radio 0.65 de trayectorias exitosas, mientras que TD3CoL alcanzó un radio de trayectorias exitosas de 0.75. Una posible causa de estos resultados correspondería, nuevamente, a una política más conservadora de conducción, la cual aumentaría la cantidad de recompensa obtenida, por ejemplo, las recompensas de centralización de ruta y alineación con respecto a la ruta, además y por sus características de conducción permitiría que el algoritmo alcanzase con una mayor frecuencia la meta.

Además de las pruebas realizadas anteriormente, también se adjuntan varios vídeos[2] en donde se muestra el rendimiento de las distintas políticas de conducción para escenarios definidos. Esto con el objetivo de hacer una comparación cualitativa de las políticas de conducción obtenidas para los distintos métodos de aprendizaje.

[2] https://drive.google.com/drive/folders/1SmsTulTxrlQSvQk5qLTsHYKIU2siNELN?usp=sharing

	DDPG	CoL	TD3CoL
Static	-424.28	696.49	**753.2**
2Cars1LeadM	-476.19	733.09	**767.65**
2Cars2RandomM	**1071.81**	723.04	757.34
3Cars3RandomM	**936.99**	639.63	711.83
4Cars4RandomM	**570.89**	-824.48	-737.95
3Cars2StraightM	591.31	**623.85**	312.29
3Cars3LeadM	-722.31	704.84	**733.11**

Tabla 4.3: Mediana de la recompensa promedio obtenida por lo distintos agentes para todos los escenarios de prueba

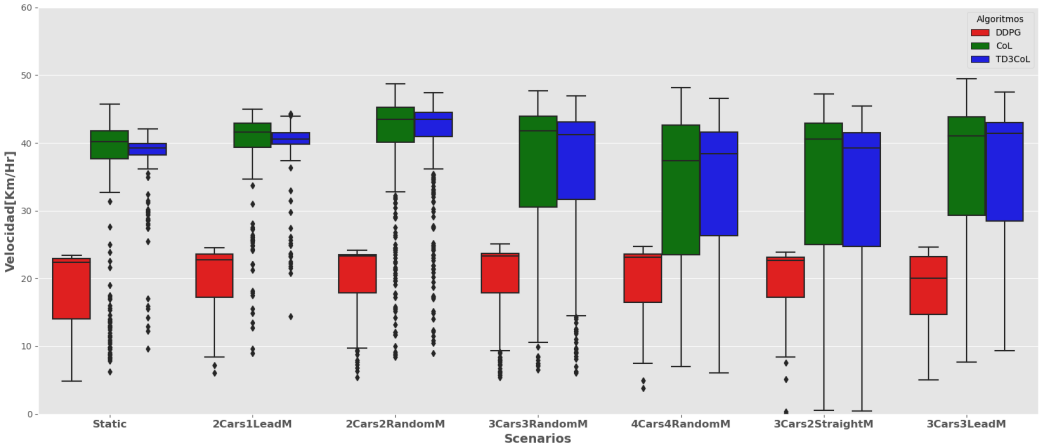


Figura 4.15: Distribución de velocidades para los tres algoritmos de aprendizaje en las distintas pruebas de evaluación

Otra estadística interesante para analizar es la velocidad promedio que alcanzó el agente en los distintos escenarios de prueba a los que se vio enfrentado. En la figura 4.15 se muestra un gráfico de cajas de la velocidad alcanzada por el agente, entrenado por los distintos tres algoritmos de aprendizaje. De este gráfico se desprende el hecho de que DDPG maneja una velocidad muy parecida para cada uno de los escenarios propuestos, promediando una velocidad de 22$[km/h]$, confirmando el hecho de que DDPG consiguió obtener una política de conducción más conservadora. Por otro lado, las velocidades para los otros dos algoritmos son similares entre ellos, promediando cercano a los 40$[km/h]$ entre todos los escenarios de prueba. Lo más interesante de este último resultado, es que coincide casi perfectamente a la velocidad objetiva que se estableció como parte de la función de recompensa.

Un caso interesante para mencionar son los escenarios *3Cars3RandomM*, *4Cars4RandomM*, *3Cars2StraightM* y *3CarsLeadM* en donde se aprecia como la velocidad posee una mayor varianza de la mediana. Esto es consecuencia directa de la mayor cantidad de obstáculos a evadir y las diferentes maniobras que debe realizar el agente para evitar alguna colisión.

Se acaban de presentar los resultados obtenidos para el entrenamiento del agente con tres distintos algoritmos de aprendizaje, tal como lo son DDPG, CoL y TD3CoL. Este último, propuesto en este documento. También se presentaron los resultados obtenidos en los diferentes escenarios de evaluación planteados, los cuales contemplaban diferentes configuraciones con el objetivo de evaluar la robustez de las políticas de conducción obtenidas.

Los resultados obtenidos dan cuenta que los algoritmos de aprendizaje asistido (CoL y TD3CoL) pudieron obtener una política de conducción más robusta que aquella obtenida por el algoritmo DDPG. Además, se pudo verificar que DDPG si aprende de algunas configuraciones, pero no de aquellas que estresan en demasía la política de conducción. Configuraciones que si fueron aprendidas por las políticas de conducción obtenidas desde CoL y TD3CoL. Además, se demostró que el algoritmo propuesto TD3CoL logró superar al algoritmo CoL en seis de los siete escenarios de prueba para la métrica de radio de éxito. Además, se logró comprobar que TD3CoL registra una radio de recompensa promedio mayor a su contraparte CoL en todos los escenarios de prueba. Demostrando así su superioridad en cuanto a evasión de colisión y forma de conducción.

Considerando las condiciones actuales de aprendizaje y los resultados obtenidos, cabe preguntarse: **¿Qué tan necesario será entrenar al agente en configuraciones adversas de conducción que tengan una probabilidad baja de ocurrencia en la vida real?**. Por tal como se entrenó al agente y evaluó este, se consideran configuraciones que son poco realistas, y que, por lo general, no se darían en escenarios comunes de la vida real. Sin embargo, si se pretende entrenar un agente que pueda manejar en cualquier escenario adverso, se hace necesario considerar la mayor cantidad de escenarios posibles. Aunque se debe mencionar el hecho de que en configuraciones adversas que sean extremas para el agente, sería más abordable manejar el freno más que la evasión. Cuya característica no se implementó en este documento debido a que complejiza aún más el problema, pero que se hace necesario si se pretende llevar este sistema a uno que sea más realista.

Una de las limitaciones del modelo propuesto es la incapacidad para capturar dependencias temporales de los datos de entrada. Esto le permitiría al agente tener sentido del tiempo y adelantar las consecuencias venideras de la ruta actual que lleva el vehículo. Este acercamiento ha sido exitoso en agentes desplegados en videojuegos, tal como lo demostró el trabajo de Ha. [70].

5. Conclusiones y trabajo futuro

En esta tesis se ha implementado, probado y cuantificado el rendimiento de un controlador neuronal basado en imágenes y entrenado bajo distintos algoritmos de *Reinforcement Leaning*. También, y con el objetivo de optimizar el entrenamiento del controlador, se optó por hacer uso de representaciones compactas de imágenes semánticas captadas desde la cámara frontal del vehículo. Con ello se pudo demostrar que una política que integra demostraciones expertas más las bondades de *Reinforcement Learning*, como lo son los algoritmos CoL y el algoritmo propuesto en este documento denominado TD3CoL, resultan en políticas de conducción exitosas en, a lo menos, un 66 % y 70 % de un total de 2100 escenarios de prueba, respectivamente. Escenarios de pruebas que se configuraron para que tuviesen la mayor cantidad de situaciones que desafiasen y comprobasen la robustez de la política obtenida para los algoritmos de aprendizaje implementados.

Uno de los beneficios que se tiene al usar modelos generativos en el *pipeline* de control es la explicabilidad que concede al modelo del agente. Permitiendo saber a que le presta más atención el modelo para el control del vehículo. Esto mismo fue demostrado en esta tesis a través de un modelo VAE, que permite capturar dependencias espaciales de la escena y poder obtener un vector latente z de menor dimensionalidad y que contiene toda la información relevante de la imagen de entrada. Mediante algunos experimentos realizados se pudo mostrar las características desagregadas que aprende este modelo neuronal, tales como distintos elementos de la ruta, como señaletica, arboles, líneas de la calle y exo-vehículos presentes en la ruta.

Con el objetivo de entender el aprendizaje de la política de conducción por partes de los algoritmos revisados, se entrenó al agente en escenarios fijos, registrándose que los algoritmos de aprendizaje asistido (CoL y TD3CoL) tomaron alrededor de 100 a 300 episodios en aquellos escenarios con menos de dos exo-vehículos en ruta y 700 a 900 episodios para aquellos escenarios que poseían más de dos exo-vehículos en ruta y con conducciones aleatorias. En contra parte, DDPG demostró aprender y superar escenarios que poseían a lo más dos exo-vehículos en ruta, ya que una vez aumentaba el número de exo-vehículos o se añadía conducción adversaria, DDPG fracasaba categóricamente en el aprendizaje de una política de conducción para la evasión de colisiones.

Desde este mismo experimento, se pudo comprobar como los algoritmos de aprendizaje asistido aprendían a evadir alguna colisión con el exo-vehículo mediante diferentes trayectorias que rodeaban a los exo-vehículos, mientras que DDPG convergía a una trayectoria definida. Aunque este último colapsaba cuando se veía enfrentado a escenarios más desafiantes.

A modo de trabajo futuro se debiese considerar la implementación de un módulo de representación temporal tal como se hace en [64][70], para verificar el aporte que tendría un módulo de estas características en el controlador propuesto para la evasión de colisión. Además, se debiese considerar la integración de un módulo de toma de decisiones jerárquicas, como se hizo

en [74], donde demostrarón que tal sistema ayuda a aumentar los casos de éxito en cambios de línea. Esto último motivado por la similitud en el movimiento que el agente debe realizar al momento de adelantar un vehículo o para evadir una colisión, sin la necesidad de frenar.

Por otro lado, sería interesante investigar comportamientos más sofisticados, tal como sería la posibilidad de frenar de manera repentina el vehículo, o dicho de otro modo permitir aceleraciones negativas. Abriendo un sin fin de situaciones nuevas a estudiar y que podrían robustecer la política actual de conducción.

Otro punto que considerar es la integración de este sistema de representación visual dentro de un sistema de sensores para una percepción más completa de la ruta, permitiendo que la eficiencia alcanzada por las representaciones de las imágenes semánticas puedan ser parte de un sistema que integre otro tipo de sensores como pueden ser LIDAR's, RADAR's o un arreglo de cámaras en diferentes posiciones del vehículo, permitiendo crear diferentes módulos que compacten la información de su sensor correspondiente, y hacer uso de aprendizaje por representación para una conducción completamente autónoma. Más en detalle, para la correcta integración de cada uno de estos módulos se debiese definir la metodología de procesamiento de los datos provenientes de los distintos sensores que conforman el sistema perceptor del vehículo autónomo. Por ejemplo, para el procesamiento de los datos provenientes del LIDAR se pueden implementar diferentes formas de procesamiento de la nube de puntos, tales como el procesamiento crudo de los datos [85] o el procesamiento de imágenes formadas por la proyección de la nube de puntos a un espacio euclidiano [86]. De la misma forma que esta última, se pueden procesar los datos provenientes de los sensores RADAR's, proyectando las mediciones obtenidas a el espacio de las imágenes para su posterior procesamiento mediante una red convolucional. Una vez procesada toda la información y representada en un espacio de características, se pueden procesar estas en forma conjunta para la posterior toma de decisión, tal como lo hacen en [86], [87] y [88].

Además del procesamiento de los datos, una extensión de este trabajo y del controlador realizado, sería la implementación de un sistema más completo y complejo como sería la inclusión de redes de comunicación intra-vehicular para mantener un sistema consciente de los actores presentes alrededor del vehículo autónomo. Específicamente, se podrían utilizar VANETs (*Vehicular Ad-Hoc Networks*) para la comunicación entre vehículos, vehículos y peatones o vehículos e infraestructura [89]. Con estas redes de comunicación se pueden mantener comunicaciones constantes sobre diferentes datos del vehículo, peatón o infraestructura que le permitiría al sistema tomar decisiones con respecto a seguridad, congestión, etc. Por ejemplo, dadas las posiciones, velocidad, aceleración y comandos de control de cada vehículo, permitirían verificar intersecciones entre las rutas de los distintos agentes y establecer alguna acción de seguridad, tal como activar el freno de mano inmediatamente después de superados ciertos umbrales de distancia y velocidad entre los agentes críticos. También, se podría obtener información global de los agentes que se encuentren alrededor del vehículo autónomo, concatenarla y/o procesarla para la inclusión de esta información contextual en la predicción de los comandos de control mediante el controlador neuronal definido en esta tesis.

Bibliografía

[1] K. Schwab, The Fourth Industrial Revolution, *Currency*, 2017.

[2] SAE International. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. J3016. 2021

[3] I. Meidute-Kavaliauskiene, B. Yıldız, Ş. Çiğdem and R. Činčikaitė, Do people prefer cars that people don't drive? A survey study on Autonomous Vehicles, *Energies, vol. 14, no. 16*, p. 4795, 2021.

[4] X. Guang, Y. Gao, H. Leung, P. Liu and G. Li, An autonomous vehicle navigation system based on inertial and visual sensors, *Sensors, vol. 18, no. 9*, p. 2952, 2018.

[5] X. Sun, S. Cao and P. Tang, Shaping driver-vehicle interaction in autonomous vehicles: How the new in-vehicle systems match the human needs, *Applied Ergonomics, vol. 90*, p. 103238, 2021.

[6] C. V. Baccarella, T. F. Wagner, C. W. Scheiner, L. Maier and K.-I. Voigt, Investigating consumer acceptance of autonomous technologies: The case of self-driving automobiles, *European Journal of Innovation Management, vol. 24, no. 4*, pp. 1210–1232, 2020.

[7] M. Massar, I. Reza, S. M. Rahman, S. M. Abdullah, A. Jamal and F. S. Al-Ismail, Impacts of autonomous vehicles on greenhouse gas emissions—positive or negative?, *International Journal of Environmental Research and Public Health, vol. 18, no. 11*, p. 5567, 2021.

[8] J. Wu, H. Liao and J.-W. Wang, Analysis of consumer attitudes towards autonomous, connected, and electric vehicles: A survey in China, *Research in Transportation Economics, vol. 80*, p. 100828, 2020.

[9] A. Severino, S. Curto, S. Barberi, F. Arena and G. Pau, Autonomous vehicles: An analysis both on their distinctiveness and the potential impact on urban transport systems, *Applied Sciences, vol. 11, no. 8*, p. 3604, 2021.

[10] B. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. Sallab, S. Yogamani, P. Pérez, Deep Reinforcement Learning for Autonomous Driving: A Survey, *CoRR, vol. abs/2002.00444*, 2020.

[11] P. Launonen, A. O. Salonen and H. Liimatainen, Icy Roads and urban environments. passenger experiences in autonomous vehicles in Finland, *Transportation Research Part F: Traffic Psychology and Behaviour, vol. 80*, pp. 34–48, 2021.

[12] T. Mitchell, Machine Learning, *McGraw Hill*, 1997.

[13] K. Murphy, Probabilistic Machine Learning: An Introduction, *MIT Press*. 2022.

[14] I. Goodfellow and Y. Bengio and A. Courville, Deep Learnig, *MIT Press*, 2016.

[15] C. Anfinsen, Studies on the principles that govern the folding of protein chains. Nobel

Lecture, December 11, 1972. National Institutes of Health Bethesda, Maryland.

[16] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli and D. Hassabis, Highly accurate protein structure prediction with alphafold, *Nature, vol. 596, no. 7873*, pp. 583–589, 2021.

[17] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T.J. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, Language Models are Few-Shot Learners, *ArXiv, abs/2005.14165*, 2020

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, Mastering the game of go with deep neural networks and Tree Search, *Nature, vol. 529, no. 7587*, pp. 484–489, 2016.

[19] T. Hastie, R. Tibshirani and J. H. Friedman, The elements of statistical learning: Data mining, inference, and prediction. *New York: Springer*, 2001.

[20] M. Nielsen, Neural Networks and Deep Learning. *Determination Press*, 2015.

[21] S. Deekshith, V. Jayanth, N. Shrishail and A. Mohammed, Diving deep into deep learning:history, evolution, types and applications, *International Journal of Innovative Technology and Exploring Engineering, vol. 9, no. 3*, pp. 2835–2846, 2020.

[22] I. Sutskever, O. Vinyals and Q. Le. Sequence to sequence learning with neural networks. *Proceedings of the 27th International Conference on Neural Information Processing Systems - vol. 2, NIPS'14*, *MIT Press*, Cambridge, MA, USA, 3104–3112. 2014

[23] X. Guo, X. Liu, E. Zhu and J. Yin, Deep clustering with convolutional autoencoders, *Neural Information Processing*, pp. 373–382, 2017.

[24] E. Pinho and C. Costa, Unsupervised learning for concept detection in medical images: A comparative analysis, *Applied Sciences, vol. 8, no. 8*, p. 1213, 2018.

[25] R. Sutton and A. Barto, Reinforcement learning: An introduction. *MIT press Vol. 1. No. 1. Cambridge*, 1998.

[26] J. Yu, W. Yu and J. Gu, Online Vehicle Routing With Neural Combinatorial Optimization and Deep Reinforcement Learning, *IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 10*, pp. 3806-3817, 2019.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, *CoRR, abs/1312.5602*, 2013.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller. Deterministic policy gradient algorithms, *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*, 2014.

[29] P. Kebria, R. Alizadehsani, S. Salaken, I. Hossain, A. Khosravi, D. Kabir, A. Koohestani,

H. Asadi, S. Nahavandi, E. Tunsel and M. Saif, Evaluating Architecture Impacts on Deep Imitation Learning Performance for Autonomous Driving, *IEEE International Conference on Industrial Technology (ICIT)*, pp. 865-870, 2019.

[30] B. Zheng, S. Verma, J. Zhou, I.W. Tsang and F. Chen. Imitation Learning: Progress, Taxonomies and Opportunities. *ArXiv, abs/2106.12177*, 2021.

[31] J. Hua, L. Zeng, G. Li, and Z. Ju, Learning for a Robot: Deep Reinforcement Learning, Imitation Learning, Transfer Learning, *Sensors, vol. 21, no. 4*, p. 1278, 2021.

[32] S. Arora and P. Doshi, A survey of inverse reinforcement learning: Challenges, methods and progress, *Artificial Intelligence, vol. 297*, p. 103500, 2021.

[33] P. Abbeel, Inverse Reinforcement Learning, CS 287: Advanced Robotics. Fall 2012. University of California at Berkeley

[34] S. Levine. Inverse Reinforcement Learning. CS 294-112: Deep Reinforcement Learning. Fall 2017. University of California at Berkeley

[35] C. Lynch and P. Sermanet, Language conditioned imitation learning over unstructured data, *Robotics: Science and Systems XVII*, 2021.

[36] Z. Wang, J. Liu, H. Cui, C. Jin, M. Yang, Y. Wang, X. Li and R. Mao, Two-stage behavior cloning for spoken dialogue system in debt collection, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

[37] N. Ratliff, J. A. Bagnell and S. S. Srinivasa, Imitation learning for locomotion and manipulation, *7th IEEE-RAS International Conference on Humanoid Robots*, 2007.

[38] T. Zhang, Z. McCarthy, O. Jow, D. Lee, K. Goldberg and P. Abbeel, Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. *IEEE International Conference on Robotics and Automation (ICRA)*, 1-8, 2018.

[39] D. Pomerleau, ALVINN: An Autonomous Land Vehicle in a Neural Network. *Advances in Neural Information Processing Systems*, 1989.

[40] M. Bojarski, D.W. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao and K. Zieba, End to End Learning for Self-Driving Cars. *ArXiv, abs/1604.07316*, 2016

[41] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah and A. Kendall, Urban Driving with Conditional Imitation Learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 251-257, 2020.

[42] S. Ross, G. Gordon and D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning in Proceedings of the fourteenth international conference on artificial intelligence and statistics. *JMLR Workshop and Conference Proceedings*, pp. 627–635, 2011.

[43] S. Ross and D. Bagnell, Efficient reductions for imitation learning, *Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, pp. 661–668, 2010

[44] J. Quiñonero Candela, M. Sugiyama, A. Schwaighofer and N. D. Lawrence, Dataset Shift in Machine Learning Shift in Machine Learning, *The MIT Press*, 2009.

[45] F. Codevilla, E. Santana, A.M. López and A. Gaidon, Exploring the Limitations of Beha-

vior Cloning for Autonomous Driving, *IEEE/CVF International Conference on Computer Vision (ICCV)*, 9328-9337, 2019.

[46] Z. Qiao, Z. Tyree, P. Mudalige, J.G. Schneider and J.M. Dolan, Hierarchical Reinforcement Learning Method for Autonomous Vehicle Behavior Planning, *ArXiv, abs/1911.03799*, 2019.

[47] M. Hausknecht and P. Stone, Deep reinforcement learning in parameterized action space, *arXiv preprint arXiv:1511.04143*, 2015.

[48] E. Yurtsever, J. Lambert, A. Carballo and K. Takeda, A survey of autonomous driving: Common practices and emerging technologies, *IEEE Access, vol. 8*, pp. 58443–58469, 2020.

[49] Y. Fu, C. Li, R. Yu, T. H. Luan and Y. Zhang, A Decision-making Strategy for Vehicle Autonomous Braking in Emergency via Deep Reinforcement Learning, *IEEE Transactions on Vehicular Technology*, 2020.

[50] Y. Bengio, J. Louradour, R. Collobert and J. Weston, Curriculum learning, *In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, 2009.

[51] Z. Qiao, K. Muelling, J. M. Dolan, P. Palanisamy and P. Mudalige, Automatically Generated Curriculum based Reinforcement Learning for Autonomous Vehicles in Urban Environment, *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1233-1238, 2018.

[52] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek and N. R. Waytowich, Integrating BehaviorCloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Envi-ronments, *In Proceedings of the 19th International Conference on Autonomous Agents and MultiagentSystems (AAMS'20)*, pp. 465–473, 2020.

[53] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra. Continuous control with deep reinforcement learning, *International Conference on Learning Representations (ICLR)*, 2016.

[54] E. Espié, C. Guionneau, B. Wymann, C. Dimitrakakis, R. Coulom and A. Sumner. TORCS, The Open Racing Car Simulator. 2005. Software available at http://torcs.sourceforge.net. [Accessed: 13 - Abr - 2022].

[55] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun. CARLA: An Open Urban Driving Simulator. *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[56] G. Rong, B. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky and S. Kim, Lgsvl simulator: A high fidelity simulator for autonomous driving, *arXiv preprint arXiv:2005.03778*, 2020.

[57] Udacity: An open source self-driving car. [Online]. Available: https://udacity.com/self-driving-car. [Accessed: 13 - Abr - 2022].

[58] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley and A. Shah. Learning to drive in a day, *CoRR, abs/1807.00412*, 2018.

[59] J. Wang, Q. Zhang, D. Zhao and Y. Chen, Lane Change Decision-making through Deep Reinforcement Learning with Rule-based Constraints. *International Joint Conference on Neural Networks (IJCNN)*, 1-6, 2019.

[60] X. Pan, D. Seita, Y. Gao and J. Canny, Risk averse robust adversarial reinforcement learning, *International Conference on Robotics and Automation (ICRA)*, 2019.

[61] O. Anschel, N. Baram and N. Shimkin. Averaged-DQN: variance reduction and stabilization for deep reinforcement learning, *In Proceedings of the 34th International Conference on Machine Learning - vol. 70 (ICML'17)*, JMLR.org, 176–185, 2017.

[62] Y. Bengio,, A.C. Courville and P. Vincent, Representation Learning: A Review and New Perspectives, *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35*, 1798-1828. 2013.

[63] T. Lesort, N. Díaz-Rodríguez, Goudou Jean-Francois, and D. Filliat, State Representation Learning for Control: An overview, *Neural Networks, vol. 108*, pp. 379–392, 2018.

[64] H. Porav and P. Newman, Imminent Collision Mitigation with Reinforcement Learning and Vision, *21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 958-964, 2018.

[65] T. Schaul, J. Quan, I. Antonoglou and D. Silver, Prioritized experience replay, *International Conference on Learning Representations*, 2016.

[66] S. Fujimoto, H. van Hoof and D. Meger, Addressing function approximation error in actor-critic methods, *CoRR, vol. abs/1802.09477*, 2018.

[67] F. Zhang, J. Li and Z. Li, A TD3-based Multi-agent Deep Reinforcement Learning Method in Mixed Cooperation-Competition Environment. Neurocomputing - vol 411, pp 206-215, 2020.

[68] J. Lyu, X. Ma, J. Yan and X. Li, Efficient Continuous Control with Double Actors and Regularized Critics. ArXiv, abs/2106.03050, 2021.

[69] L. Pan, Q. Cai and L. Huang, Softmax Deep Double Deterministic Policy Gradients. ArXiv, abs/2010.09177, 2020.

[70] D. Ha and J. Schmidhuber, World models. *arXiv preprint arXiv:1803.10122*, 2018.

[71] Y. Abeysirigoonawardena, F. Shkurti and G. Dudek, Generating Adversarial Driving Scenarios in High-Fidelity Simulators, *International Conference on Robotics and Automation (ICRA)*, pp. 8271-8277, 2019.

[72] M. Vergara, Accelerating Training of Deep Reinforcement Learning-based Autonomous Driving Agents Through Comparative Study of Agent and Environment Designs. *NTNU Open Access*, 2019.

[73] H. Chae, C. Mook Kang, B. Kim, J. Kim, C. Choo Chung and J. Won Choi, Autonomous braking system via deep reinforcement learning, *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 1-6., 2017.

[74] Y. Chen, C. Dong, P. Palanisamy, P. Mudalige, K. Muelling and J. M. Dolan, Attention-Based Hierarchical Deep Reinforcement Learning for Lane Change Behaviors in Autonomous Driving, *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1326-1334, 2019.

[75] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed and A. Lerchner, β-VAE: Learning basic visual concepts with a constrained variational framework, *ICLR*, 2017.

[76] K. He and J. Sun, Convolutional neural networks at constrained time cost, 2015 IEEE

Conference on Computer Vision and Pattern Recognition (CVPR), 5353-5360, 2015.

[77] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, The Cityscapes Dataset for Semantic Urban Scene Understanding, *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[78] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[79] W. Liu, R. Li, M. Zheng, S. Karanam, Z. Wu, B. Bhanu, R.J. Radke and O.I. Camps, Towards Visually Explaining Variational Autoencoders, *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8639-8648, 2020.

[80] H. Kim and A. Mnih, Disentangling by Factorising, *ArXiv, abs/1802.05983*, 2018.

[81] C. Eastwood and C.K. Williams, A Framework for the Quantitative Evaluation of Disentangled Representations, *ICLR*, 2018.

[82] K. Do and T. Tran, Theory and Evaluation Metrics for Learning Disentangled Representations, *ArXiv, abs/1908.09961*, 2020.

[83] A. Sepliarskaia, J. Kiseleva and M. de Rijke, Evaluating Disentangled Representations, *ArXiv, abs/1910.05587*, 2019.

[84] J. Zaidi, J. Boilard, G. Gagnon and M. Carbonneau, Measuring Disentanglement: A Review of Metrics, *ArXiv, abs/2012.09276*, 2020.

[85] Z. Liu, A. Amini, S. Zhu, S. Karaman, S. Han and D. Rus, Efficient and Robust LiDAR-Based End-to-End Navigation, 2021 IEEE International Conference on Robotics and Automation (ICRA), 13247-13254, 2021.

[86] P. Cai, S. Wang, Y. Sun and M. Liu, Probabilistic End-to-End Vehicle Navigation in Complex Dynamic Environments With Multimodal Sensor Fusion, IEEE Robotics and Automation Letters, 5, 4218-4224, 2020.

[87] J. Maanpää, J. Taher, P. Manninen, L. Pakola, I. Melekhov and J. Hyyppä, Multimodal End-to-End Learning for Autonomous Steering in Adverse Road and Weather Conditions, 2020 25th International Conference on Pattern Recognition (ICPR), 699-706, 2021.

[88] A. Prakash, K. Chitta and A. Geiger, Multi-Modal Fusion Transformer for End-to-End Autonomous Driving, 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 7073-7083, 2021.

[89] M. Ahangar, Q. Ahmed, F. Khan, M. Hafeez, Survey of Autonomous Vehicles: Enabling Communication Technologies and Challenges, Sensors 21, 706, 2021.