

UNIVERSIDADE DE SÃO PAULO  
ESCOLA DE ENGENHARIA DE SÃO CARLOS  
DEPARTAMENTO DE ENGENHARIA MECÂNICA

RELATÓRIO FINAL  
INICIAÇÃO CIENTÍFICA

PROGRAMA UNIFICADO DE BOLSAS DE ESTUDO PARA APOIO E FORMAÇÃO DE  
ESTUDANTES DE GRADUAÇÃO (PUB)  
PROJETO 2020-444

---

**Utilização de Aprendizado de Máquina em um Simulador 3D na Nuvem**

---

**PALAVRAS-CHAVE:** Aprendizado Profundo; Aprendizado por Reforço; Nuvem; AWS

*Orientadora:* PROF.DR. MARCELO BECKER

*Aluno:* JOÃO MANOEL HERRERA PINHEIRO

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Veículos Autônomos e Computação em Nuvem . . . . .	1
1.2	Objetivos . . . . .	2
<b>2</b>	<b>Aprendizado Por Reforço</b>	<b>2</b>
2.1	Processo de Decisão de Markov . . . . .	3
2.2	Aprendizado por Reforço Framework . . . . .	4
2.3	Equação de Bellman . . . . .	5
2.4	Interação de Valor . . . . .	6
2.5	Q-learning . . . . .	7
2.6	Otimização de Políticas Próximas . . . . .	7
<b>3</b>	<b>DeepRacer</b>	<b>10</b>
3.1	Veículo . . . . .	10
3.2	Espaço de Ação . . . . .	11
3.3	Função Recompensa . . . . .	13
3.4	Hiperparâmetros . . . . .	16
3.5	Arquitetura do Ambiente . . . . .	17
3.6	Arquitetura da Rede . . . . .	19
<b>4</b>	<b>Implementação</b>	<b>19</b>
4.1	Simulação I . . . . .	19
4.2	Simulação II . . . . .	22
4.3	Simulação III . . . . .	24
<b>5</b>	<b>Conclusões</b>	<b>26</b>
	<b>Referências</b>	<b>27</b>

# 1 Introdução

## 1.1 Veículos Autônomos e Computação em Nuvem

Um sistema pode ser considerado “autônomo” quando todas as tarefas dinâmicas são realizadas pelo próprio sistema. Por exemplo, no caso de um carro autônomo, deve ser capaz de realizar as tarefas de direção, em todos os ambientes de direção usando apenas seu sistema automatizado.[1]

A pesquisa e o desenvolvimento de veículos autônomos cresceu na última década. Com o avanço da tecnologia, tivemos além do aumento do poder de processamento dos computadores a redução de custo de equipamentos essenciais para o desenvolvimento de veículos autônomos, como câmera, sensores e os próprios computadores. Acredita-se que os veículos autônomos são o futuro para um transporte mais eficiente e seguro.



Figura 1: Veículo Autônomo do Google

Conforme a tecnologia foi avançando a quantidade de dados gerados foi aumentando exponencialmente, mediante a isto o investimento para o processamento destes dados para utilizar em aplicações de Machine Learning pode ser altamente custoso para algumas empresas, por isso começaram a surgir soluções em nuvem como a Microsoft Azume, Amazon Web Services, Google Cloud, IBM Cloud dentre outros. Com isso não existe mais a necessidade de implementar algoritmos e métodos de Machine Learning na sua máquina local e sim utilizar uma solução em nuvem disponível no mercado.

Com isso surgiu a plataforma de DeepRacer fornecida pela Amazon Web Services (AWS). Esta plataforma tem como o objetivo juntar os desafios de Machine Learning em um carro de corrida autônomo dentro de um simulador 3D baseado totalmente na nuvem. Os resultados obtidos dentro do

simulador podem ser usados no modelo físico do carro na pista de corrida.



Figura 2: Carro da AWS DeepRacer na Pista.

## 1.2 Objetivos

Objetiva-se, com esse projeto de Iniciação Científica, o estudo e desenvolvimento de algoritmos para Aprendizado Por Reforço na nuvem para otimizar a performance de um veículo autônomo no ambiente do simulador 3D fornecido pela AWS.

## 2 Aprendizado Por Reforço

No aprendizado de máquina, distinguimos três tipos de aprendizado: supervisionado, não aprendizagem supervisionada e de reforço. A distinção geralmente é feita pelo feedback o agente recebe. A aprendizagem supervisionada é a tarefa de aprender uma função a partir de um conjunto de dados rotulado. De rotulados, queremos dizer que os dados consistem em exemplos de treinamento, as entradas, e seus valores de saída desejados. O problema pode ser uma classificação, se a saída for uma classe, ou um regressão, se a saída for um ou vários valores reais. Em ambos os casos, o feedback é o saída correta que corresponde à entrada fornecida. Um modelo treinado totalmente supervisionado serve como uma função que podemos usar para mapear novas entradas. O objetivo é ser capaz de rotular dados não vistos corretamente: exemplos que não estavam no conjunto de treinamento. A aprendizagem não supervisionada é semelhante à aprendizagem supervisionada, exceto que o treinamento os dados são “não rotulados”. Em outras palavras, ele deve aprender sem receber nenhum feedback. Dentro neste caso, uma técnica de aprendizagem não supervisionada pode ser aplicada para identificar, por exemplo, diferentes grupos de tipos de dados (ou seja, agrupamento).

Entre estes dois casos está a aprendizagem por reforço, onde existe feedback mas não indica se a ação realizada foi a correta. Depois de atuar, o agente recebe um feedback, uma recompensa imediata que pode ser positiva ou negativa. Isto cabe ao agente e, portanto, ao algoritmo de aprendizado, usar e interpretar essa recompensa.[2, 3, 4]

A aprendizagem por reforço pode ser vista como uma abordagem de tentativa e erro. Como o agente não é explicitamente informado sobre qual ação tomar, ele só pode avaliar as ações com as recompensas recebeu. Para que esta avaliação seja eficiente, o agente deve interagir continuamente com o meio ambiente e adaptar sua estratégia no que diz respeito às recompensas que obtém. A recompensa imediata depende da ação realizada e do estado em que o agente se encontra.

## 2.1 Processo de Decisão de Markov

Um dos modelos mais tradicionais para um problema de Aprendizado por Reforço com um único agente é o Processo de Decisão de Markov. Um processo de decisão de Markov (MDP - Markov Decision Process) é uma forma de modelar processos em que as transições de estado são probabilísticas. Cada ação tem uma recompensa que depende do estado em que o processo se encontra. Num MDP os efeitos das ações são probabilísticas e obedecem a propriedade de Markov, o efeito de uma ação depende apenas do estado atual do sistema e não de como o processo chegou em tal estado[5, 6, 7].

São ditos “Markovianos” porque os processos modelados obedecem a propriedade de Markov: o efeito de uma ação em um estado depende apenas da ação e do estado atual do sistema, e são chamados de processos “de decisão” porque modelam a possibilidade de um agente interferir periodicamente no sistema executando ações.

Um MDP pode ser definido formalmente como uma tupla  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$  em que:

- $\mathcal{S}$  é o conjunto finito de possíveis estados do ambiente.
- $\mathcal{A}$  é o conjunto finito de ações que podem ser executadas pelo agente.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  é uma função que retorna a probabilidade do sistema passar para um estado  $s_{t+1} \in S$ , em que o processo estava em um estado  $s_t \in S$  e o agente decidiu executar uma ação  $a_t \in A$ , no mesmo instante de tempo  $t$ .
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  é a função de recompensa por tomar uma decisão  $a \in A$  quando o processo está em um estado  $s \in S$ .

- $\gamma$  é o fator de desconto  $\gamma \in [0, 1]$

As distribuições de ações para um dado estado é definido pela política  $\pi$ .

$$\pi(s) = P(a|s) \quad (2.1)$$

## 2.2 Aprendizado por Reforço Framework

A partir de um MDP simples podemos construir o framework para o Aprendizado por Reforço. Um agente interage com o ambiente através de uma sequência de ações escolhidas em um estado observado, com o objetivo de maximizar um sinal de recompensa. Para cada ação realizada no estado  $s_t$  há uma recompensa  $r_{t+1}$  após o agente atingir determinado estado  $t + 1$ .

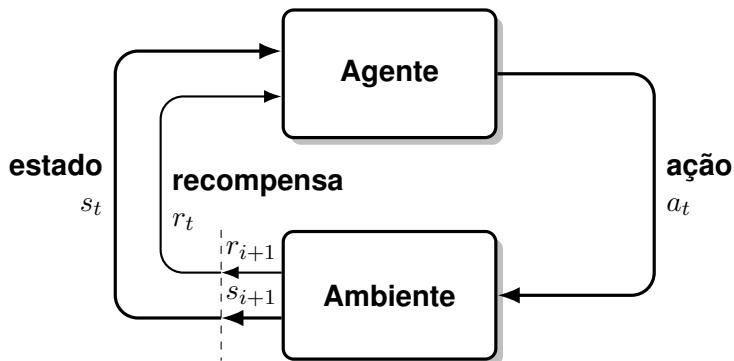


Figura 3: Framework de Aprendizado por Reforço [2]

O objetivo do agente é sempre maximizar a recompensa esperada. Este é o conceito da hipótese de recompensa que pode ser expressado como:

$$G_t = E_\pi[r] = r_{t+1} + r_{t+2} + \dots = \sum_{k=0}^{\infty} r_{t+k+1} \quad (2.2)$$

A equação 2.2 é válida quando as recompensas tem o mesmo peso para todos os instantes de tempo. Entretanto na prática isto não acontece por isso é adicionado um fator de desconto ( $\gamma$ ). Com isso a equação 2.2 fica da seguinte forma:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (2.3)$$

Ou seja, em termos gerais o aprendizado por reforço considera a seguinte função custo

$$\text{argmax}_{\theta} E_{s,a} \pi r \quad (2.4)$$

## 2.3 Equação de Bellman

A equação de Bellman [7] é uma equação recursiva que pode ser utilizada em algoritmos de aprendizado por reforço de modo que é possível trabalhar com as funções  $V(s_t)$  e  $Q(s_t, a_t)$  de maneira recursiva.

Dentro Processos de decisão de Markov, uma equação de Bellman é um recursão para recompensas esperadas. Por exemplo, a recompensa esperada por estar em um determinado estado  $s$  dado pela função  $V(s_t)$  e seguindo alguma política fixa  $\pi$  tem a seguinte equação de Bellman:

$$V_{\pi}(s_t) = E_{\pi}(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots s_t) \quad (2.5)$$

Também podemos definir uma função  $Q(s_t, a_t)$  para descrever a recompensa esperada dado a política  $\pi$  do estado e ação atual.

$$Q_{\pi}(s_t, a_t) = E_{\pi}(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots s_t, a_t) \quad (2.6)$$

Model Free é uma classe de algoritmos de aprendizado por reforço na qual o agente não tem acesso às informações da dinâmica do ambiente, ou seja, à função de transição de estados  $P$  ou a função de recompensa explícita  $R$ .

Utilizando a equação de Bellman no Model Free, temos que :

$$V_{\pi}(s_t) = E_{\pi}[r(s_t, a_t) + \gamma V_{\pi}(s_{t+1})] \quad (2.7)$$

$$Q_{\pi}(s_t, a_t) = E_{\pi}[r(s_t, a_t) + \gamma Q_{\pi}(s_{t+1}, a_{t+1})] \quad (2.8)$$

## 2.4 Interação de Valor

O algoritmo de interação de valor [5] utiliza programação dinâmica para determinar o valor de  $V^{(s)}$  para cada estado  $s$  do MDP calculando de forma iterativa. Pode ser interpretado como o valor esperado da recompensa que o agente pode obter  $t$  estágios para frente a partir do estado  $s$ . O algoritmo inicia com valores  $V$  aleatórios. Em cada iteração  $i$  calcula-se  $V_i$  baseado na função valor  $V_{i-1}$ . O algoritmo também faz uso da função  $Q(s, a)$ .

$$Q_t(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V_{t-1}(s') \quad (2.9)$$

E o valor do estado  $s$  no estágio de decisão  $t$  é igual ao valor da melhor ação:

$$V_t(s) = \max Q_t(s, a) \quad (2.10)$$

Com  $a \in A$ .

A complexidade do algoritmo é  $\mathcal{O}(h|A||S|^2)$ . O critério de parada do algoritmo para horizonte finito é que o número de iterações seja igual ao horizonte do problema. No caso de horizonte finito, o algoritmo gera uma política para cada estágio de decisão.

---

**Algorithm 1:** Interação de Valor ( $M, \epsilon$ )

---

**Input:**  $M$ : um MDP,  $\epsilon$ : erro máximo permitido

**Output:**  $\pi^*$ : uma política ótima

**for**  $s \in S$  **do**

$V_0 \leftarrow$  Valor Aleatório;

$t \leftarrow 1$

**for**  $s \in S$  **do**

$Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V_{t-1}(s');$

**repeat**

**for**  $s \in S$  **do**

**for**  $a \in A$  **do**

$Q^t(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V_{t-1}(s');$

$V^t(s) \leftarrow \max_{a \in A} Q^t(s, a);$

$\pi(s) \leftarrow_{a \in A} Q^t(s, a);$

**until**  $V_t(s) - V_{t-1}(s) < \epsilon \forall s \in S$ ;

**return**  $\pi$

---

## 2.5 Q-learning

Uma forma de resolver o MDP através de interações com o ambiente é por exemplo com o algoritmo Q-learning [8]. Este método estima de forma interativa uma função  $Q(s, a)$ , com isso, o agente busca aprender uma função que mapeia os valores de estar em um estado e de tomar uma ação com o objetivo de maximizar a soma das recompensas futuras. Neste modelo a política do agente vem de uma ação futura  $Q_\pi(s, a)$  e a sua política ótima é dado por :

$$Q_\pi^*(s_t, a_t) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t, a_t, \pi] \quad (2.11)$$

Utilizando a Equação de Bellman, podemos modificar a equação 2.11

$$Q_\pi^*(s_t, a_t) = e[r_t + \gamma \max_{a_{t+1}} Q_\pi^*(s_{t+1}, a_{t+1})] \quad (2.12)$$

O Algoritmo 2 mostra um exemplo de implementação do algoritmo Q-Learning. Uma propriedade deste algoritmo é que a convergência dos valores de  $Q$  para  $Q^*$  (valor ótimo) é garantida desde que o par (estado, ação) sejam visitados muitas vezes [9].

---

### Algorithm 2: Q-Learning ( $M, s_0, \alpha, \gamma$ )

---

**Input:**  $M$ : um modelo,  $s_0$ : um estado inicial,  $\alpha$ : fator de aprendizado,  $\gamma$  : fator de desconto  
**for each**  $s \in S$  **do**

**for each**  $a \in A$  **do**  
 |    |  $Q_o(s, a) \leftarrow$  valor aleatório

$t \leftarrow 0$ ;

$a_0 \leftarrow \arg \max_{a' \in A} Q_0(s, a')$ ;

**while** critério de parada não é satisfeito **do**

|  $[s_{t+1}, r] \leftarrow \text{executaAcao}(s_t, a_t)$ ;  
 |  $a_{t+1} \leftarrow \arg \max_{a' \in A} Q_t(s_{t+1}, a')$ ;  
 |  $V_t(s_{t+1}) \leftarrow \max_{a' \in A} Q_t(s_{t+1}, a')$ ;  
 |  $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha[r + \gamma V_t(s_{t+1})]$ ;  
 |  $t \leftarrow t + 1$ ;

---

## 2.6 Otimização de Políticas Próximas

Otimização de Políticas Próximas Proximal ou Policy Optimization (PPO) é um método gradiente de política utilizado em aprendizado por reforço. Os métodos de gradiente de política visam modelar e otimizar a política diretamente. A política é geralmente modelada com uma função parametrizada

em relação  $\pi_\theta(a|s)$ . O valor da função de recompensa (objetivo) depende dessa política e, então, vários algoritmos podem ser aplicados para otimizar  $\theta$  para a melhor recompensa.[2, 10].

A função recompensa é definida como:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a) \quad (2.13)$$

Onde  $d^\pi(s)$  é a distribuição estacionária da cadeia de Markov para  $\pi_\theta$  (distribuição de estado na política em  $\phi$ ). Para simplificar, o parâmetro  $\theta$  seria omitido para a política  $\pi_\theta$  quando a política está presente no subscrito de outras funções; por exemplo,  $d^\pi$  e  $Q^\pi$  devem ser  $d^{\pi_\theta}$  e  $d^{\pi_\theta}$ .

Imagine que você possa viajar ao longo dos estados da cadeia de Markov para sempre e, eventualmente, conforme o tempo passa, a probabilidade de você terminar com um estado se torna inalterada - esta é a probabilidade estacionária para  $d^\pi(s) = \lim_{t \rightarrow \infty} P(s_t = s | s_0, \pi_\theta)$  é a probabilidade de que  $s_t = s$  ao começar de  $s_0$  e seguir a política  $\phi_\theta$  para  $t$  etapas.

Utilizando o gradiente ascendente, podemos mover  $\theta$  em direção à direção sugerida pelo gradiente  $\nabla_\theta J(\theta)$  para encontrar o melhor  $\theta$  para  $\phi_\theta$  que produz o maior retorno.

Calcular o gradiente  $\nabla_\theta J(\theta)$  é complicado porque depende tanto da seleção de ação (determinada diretamente por  $\phi_\theta$ ) e da distribuição estacionária de estados seguindo o comportamento de seleção do alvo (determinada indiretamente por  $\phi_\theta$ ). Dado que o ambiente geralmente é desconhecido, é difícil estimar o efeito sobre a distribuição do estado por meio de uma atualização de política. Entretanto, temos o Teorema do gradiente de política, abordado em [2], basicamente ele faz uma reformulação no derivativo da função objetivo para que não haja a necessidade de envolver o derivativo do estado de distribuição  $d^\pi$ , simplificando assim o gradiente.

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \pi_\theta(a|s) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \nabla_\theta \pi_\theta(a|s) \quad (2.14)$$

PPO faz parte de uma família de métodos gradiente de política em que alternam entre dados de amostragem por meio de interação com o ambiente, e otimizando uma função objetiva utilizando gradiente estocástico. Embora os métodos realizam o cálculo de uma atualização de gradiente por amostra de dados PPO adota várias épocas de atualizações de 'minibatch'.

Primeiro vamos definir a razão entre a nova e a antiga política:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \quad (2.15)$$

A função Trust region policy optimization (TRPO) fica como:

$$J^{\text{TRPO}}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{\text{old}}}(s, a)] \quad (2.16)$$

Para evitar instabilidade do algoritmo PPO impõe uma restrição, esta restrição força  $r(\theta)$  a ficar próximo a um pequeno intervalo perto de 1, este intervalo pode ser expressado como  $[1 - \epsilon, 1 + \epsilon]$ , em que  $\epsilon$  é um hiperparâmetro. Logo a é a loss function of Clipped Surrogate Objective

$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_{\theta_{\text{old}}}(s, a))] \quad (2.17)$$

Ao aplicar PPO na arquitetura de rede com parâmetros compartilhados para ambas as funções de política e valor, além da recompensa cortada, a função objetivo é aumentada com um termo de erro na estimativa de valor ( $c_1(V_\theta(s) - V_{\text{target}})^2$ ) e um termo de entropia ( $c_2 H(s, \pi_\theta(.))$ ) para encorajar a exploração suficiente,  $c_1, c_2$  são hiperparâmetros contantes.

$$J^{\text{CLIP'}}(\theta) = \mathbb{E}[J^{\text{CLIP}}(\theta) - c_1(V_\theta(s) - V_{\text{target}})^2 + c_2 H(s, \pi_\theta(.))] \quad (2.18)$$

Com isso, conforme visto em [11] o algoritmo para PPO fica da seguinte forma:

---

### Algorithm 3: PPO-Clip

---

**Input:**  $\theta_0$ : parâmetros da política inicial,  $\phi_0$ : parâmetros iniciais da função de valor  
**for**  $k = 0, 1, 2, \dots$  **do**

Coletar as trajetórias $\mathcal{D}_k = \tau_i$ pela política $\pi_k = \pi(\theta_k)$ no ambiente Calcular as recompensas $R_t$ Calcular $\hat{A}_t$ baseado no valor da função $V_{\phi_k}$ Atualizar a política maximizando a PPO-Clip objetiva: $\theta_{k+1} = \theta \left( \frac{1}{ \mathcal{D}_k T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_\theta(a_t s_t)}{\pi_{\theta_k}(a_t s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right) \right)$	Ajuste o valor da função utilizando regressão $\phi_{k+1} = \arg \min_{\phi} \frac{1}{ \mathcal{D}_k T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_{\phi}(s_t) - R_t \right)^2$
---	---

---

### 3 DeepRacer

Nesta secção vamos abordar a plataforma da AWS, tanto o ambiente quanto quais algoritmos da secção 2 foram implementados no ambiente de simulação DeepRacer.

#### 3.1 Veículo

DeepRacer é um modelo totalmente autônomo em escala 1/18 de um carro de corrida que é impulsionado na pista por aprendizagem por reforço. Foi desenvolvido pela Amazon para que os usuários possam implementar os algoritmos de Aprendizagem Profunda em um veículo autônomo [12]. Por meio de seu Console é possível implementar modelos para o veículo e depois treina-los.



Figura 4: Veículo da AWS DeepRacer.

O veículo é capaz de funcionar de forma autônoma, executando a inferência com base no modelo de aprendizagem por reforço que é carregado pelo usuário. O veículo é movido por motor escovado e a sua velocidade é controlada por meio de um regulador de tensão que controla o motor. Ele possui uma câmera que captura os dados para a rede neural, é possível modificar esta camera para uma camera stereo e adicionar sensores LIDAR ao modelo do veículo. A direção é controlada pelo servomecanismo, no caso é utilizado Geometria de Ackermann.

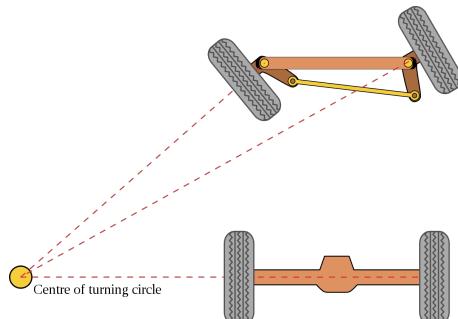


Figura 5: Geometria de Ackermann, modelo utilizado para direção do veículo.

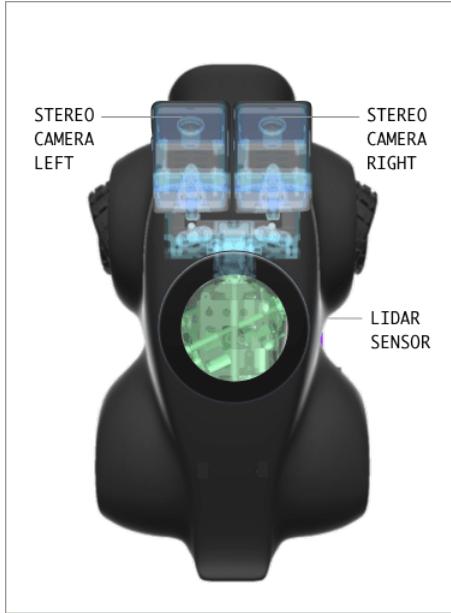


Figura 6: Modelo customizado do Veículo na plataforma da AWS.

### 3.2 Espaço de Ação

Em aprendizado profundo, o conjunto de todas as ações validas ou escolhas, disponíveis para o agente enquanto ele interagi com o ambiente é chamado de espaço de ações. As ações de espaço podem ser classificadas como espaço de ações discreto ou contínuo.

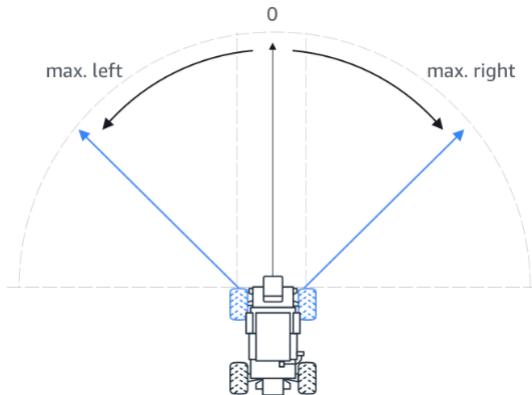


Figura 7: Espaço de Ações do Veículo da AWS DeepRacer[12]

A faixa de angulo que o veículo trabalha é entre  $-30^\circ$  e  $30^\circ$  equanto que a velocidade é entre  $0.4m/s$  e  $4m/s$

Um espaço de ação discreto representa todas as ações possíveis de um agente para cada estado em um conjunto finito. Isso significa que para cada situação diferente, a rede neural do agente seleciona uma velocidade e direção para o carro com base na entrada de sua câmera e sensor LiDAR. A escolha é limitada a um agrupamento de combinações predefinidas de ângulos de direção e de velocidade.

O espaço de ação discreta padrão do AWS DeepRacer contém as seguintes ações:

Tabela 1: AWS DeepRacer - Espaço de Ações Discreto[12]

Ação	Ângulo da Direção (graus)	Velocidade
0	-30°	0.4 m/s
1	-30°	0.4 m/s
2	-15°	0.4 m/s
3	-15°	0.8 m/s
4	0°	0.4 m/s
5	0°	0.8 m/s
6	15°	0.4 m/s
7	15°	0.8 m/s
8	30°	0.4 m/s
9	30°	0.8 m/s

Ao contrário do espaço de ação discreto, um espaço de ação contínua permite que o agente selecione uma ação de uma gama de valores para cada estado [9]. Semelhante ao discreto caso, o agente seleciona o par direção-velocidade com base na situação ambiental que é recebido da câmera e das entradas LiDAR. No entanto, no espaço de ação contínua, o agente tem uma gama de opções para escolher. Há uma troca entre tempo de atuação e treinamento para esses dois espaços de ação. Enquanto ação contínua o espaço fornece uma melhor otimização, pois há uma gama de valores para escolher e o agente pode selecionar os pares de valores ideais para melhorar o desempenho do que os comparados a um espaço de ação discreto onde o agente é forçado a escolher a partir de um conjunto de ações. No entanto, o fato de haver gama de valores em um espaço de ação contínua significa que o agente precisa treinar por mais tempo, o que aumenta a utilização de recursos.

Ao contrário do espaço de ação discreto, o espaço de ação contínuo permite definir uma gama de valores para que o agente execute sua ação diferentes da tabela 3.2.

Por exemplo, podemos escolher um espaço de ação contínua para quando o carro da AWS aproximar de uma curva ele pode optar por uma velocidade de  $0.7m/s$  á  $4m/s$ , podendo escolher um ângulo de direção de  $-20^\circ$  a  $20^\circ$ .

O benefício de usar um espaço de ação contínua é que você pode escrever funções de recompensa

que treinam modelos para incentivar ações de velocidade e direção em pontos específicos em uma pista, com isso é possível otimizar o desempenho nela. A escolha de uma série de ações também cria o potencial para mudanças suaves nos valores de velocidade e direção que, em um modelo bem treinado, podem produzir melhores resultados em condições da vida real.

Na configuração do espaço de ação discreta, limitar as escolhas de um agente a um número finito de ações predefinidas coloca sobre você a responsabilidade de entender o impacto dessas ações e defini-las com base no ambiente (pista, formato de corrida) e suas funções de recompensa. No entanto, em uma configuração de espaço de ação contínua, o agente aprende a escolher a velocidade ideal e os valores de direção dos limites mínimo e máximo fornecidos por meio do treinamento.

Embora fornecer uma gama de valores para o modelo escolher pareça ser a melhor opção, o agente precisa treinar mais para aprender a escolher as ações, com isso o sucesso também depende da função de recompensa.

### 3.3 Função Recompensa

A recompensa é a ideia central em aprendizado por reforço, em que o objetivo do agente é maximizar a recompensa esperada. Com isso a função recompensa é responsável por dar recompensas (ou punições) para o nosso agente. Ela é uma das mais importantes implementações da plataforma da AWS DeepRacer.

É uma função escrita em python que leva os parâmetros contendo as informações do estado presente do nosso agente e nos retorna uma estimativa numérica para a recompensa. De maneira geral a função implementa em python da seguinte forma:

```
1 def reward_function(params):
2     reward = ...
3     return reward
```

As informações do estado inicial, **params** é um dicionário que contém pares de key-value na tabela 2.

Tabela 2: Parâmetros Key-value [12]

Chave	Valor	Descrição
-------	-------	-----------

all_wheels_on_track	Boolean	indicar se o agente está na pista
x	float	coordenada em <i>x</i> do agente
y	float	coordenada em <i>y</i> do agente
closest_objects	[int, int]	índices dos 2 objetivos mais próximos da posição do agente
closest_waypoints	[int, int]	índices dos 2 ponto de rota mais próximos
distance_from_center	float	distancia, em metros, do centro da pista
is_crashed	Boolean	Booleano para indicar se ocorreu colisão com o agente
is_left_of_center	Boolean	Sinal para indicar se o agente está no lado esquerdo do centro da pista
is_offtrack	Boolean	Booleano para indicar se o agente saiu da pista
is_reversed	Boolean	Booleano para indicar se o agente esta em sentido horário (True) ou anti horário (False)
heading	float	angulo Yaw em grau do agente
objects_distance	[float, ]	lista da distancia de objetos entre 0 e o tamanho da pista
objects_heading	[float, ]	lista dos objetos entre $-180^\circ$ e $180^\circ$
objects_left_of_center	[Boolean, ]	uma lista de Booleanos para indicar se o objeto está (True) na esquerda do centro ou não (False)
objects_location	[float, float]	lista da localização dos objetos [(x,y),...]
objects_speed	[float, ]	lista da velocidade dos objetos [m/s]
progress	float	porcentagem da pista concluída
speed	float	velocidade do agente [m/s]
steering_angle	float	angulo da direção do agente
steps	int	numero de pacos completados
track_length	float	comprimento da pista em metros
track_width	float	largura da pista em metros
waypoints	[float, float]	lista de (x,y) como marcos ao longo do centro da pista

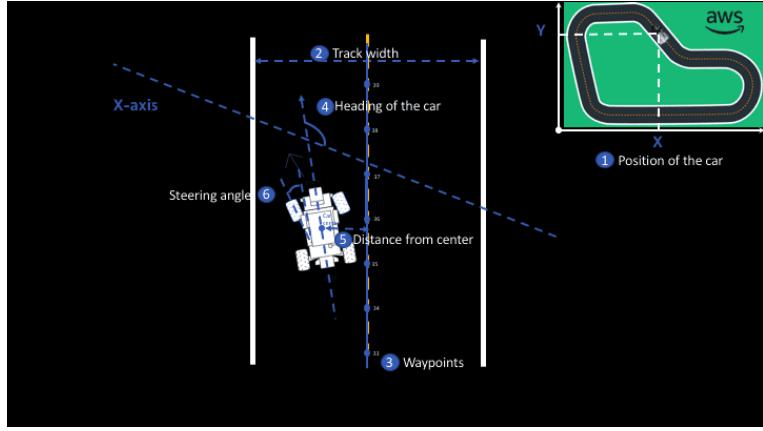


Figura 8: Exemplo de alguns parâmetros na pista. [12].

Não é necessário usar todos esses parâmetros na função de recompensa, uma função simples também pode levar a um desempenho satisfatório. No entanto, alguns parâmetros podem ser úteis ao implementar funções de recompensa para pistas mais complicadas que podem exigir que o agente se concentre em vários parâmetros diferentes simultaneamente para completar a corrida.

Um exemplo simples de função de recompensa que incentiva o agente a seguir de perto a linha central da pista é mostrado na própria documentação da AWS DeepRacer [12]. Esta função de recompensa determina o quanto longe o carro está da linha central da pista e atribui maior recompensa se estiver mais perto dela. Esta função de recompensa potencializa apenas dois parâmetros a largura da pista (track\_width) e a distância do centro (distance\_from\_center).

```

1 def reward_function(params):
2     """
3         Example of rewarding the agent to follow center line
4     """
5
6     # Read input parameters
7     track_width = params['track_width']
8     distance_from_center = params['distance_from_center']
9
10    # Calculate 3 markers that are increasingly
11    # further away from the center line
12    marker_1 = 0.1 * track_width
13    marker_2 = 0.25 * track_width
14    marker_3 = 0.5 * track_width

```

```

15
16     # Give higher reward if the car is
17     # closer to center line and vice versa
18     if distance_from_center <= marker_1:
19         reward = 1
20     elif distance_from_center <= marker_2:
21         reward = 0.5
22     elif distance_from_center <= marker_3:
23         reward = 0.1
24     else:
25         reward = 1e-3 # likely crashed/ close to off track
26
27     return reward

```

### 3.4 Hiperparâmetros

Hiperparâmetros são as variáveis que afetam o processo de treinamento e não são uma propriedade intrínseca do modelo, a escolha de hiperparâmetros ideais é um processo empírico. Ou seja, não há formulação para encontrar o melhor conjunto de hiperparâmetros e requer experimentações. Os hiperparâmetros da PPO junto com as descrições se encontra na tabela 3.

Tabela 3: Hiperparâmetros da PPO.

Hiperparâmetro	Descrição
Tamanho do Batch	Define o número de amostras a serem trabalhadas antes de atualizar os parâmetros internos do modelo.
Número de Epocas (epochs)	O número de passos dado pela rede neural durante o treinamento para atualizar os pesos.
Taxa de Aprendizado	Controla as Atualizações de pesos na Rede
Entropia	Grau de incerteza usado para determinar quando adicionar aleatoriedade ao distribuição de políticas.
Fator de Desconto	Um fator especifica quanto das recompensas futuras contribuem para a recompensa esperada. Ele define se o treinamento será lento ou rápido.

Tipo de Perda	Tipo de função objetivo usada para atualizar os pesos da rede.
---------------	--

### 3.5 Arquitetura do Ambiente

O principal método de treinamento de um modelo para operar o carro é por meio do console AWS DeepRacer. O console é uma plataforma interativa que permite os usuários monitorem o treinamento e a avaliação do modelo enquanto exibem o métricas de recompensa, porcentagem de conclusão durante o treinamento e avaliação.

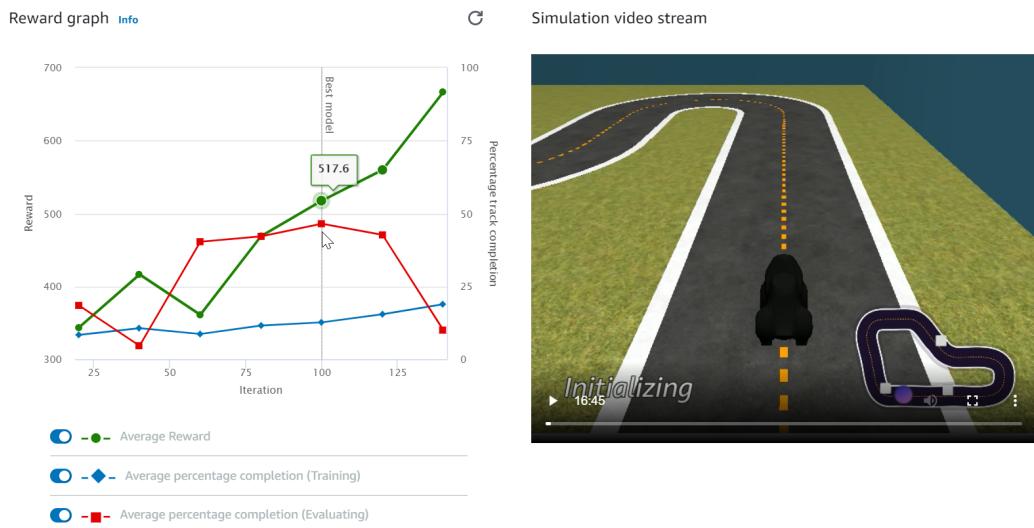


Figura 9: Treinamento do Modelo no Console da AWS DeepRacer [12].

Os principais componentes que os usuários possuem a liberdade de explorar e fazer as alterações de projetos são o Espaço de Ação e a Função de Recompensa, abordados em 3.2 e 3.3. O ambiente da AWS é composto por diversos serviços em nuvem da AWS, porém os dois principais são o Safe-Maker e RoboMaker [12]. O SageMaker é a plataforma que permite treinar os modelos de Machine Learning enquanto que o RoboMaker é o serviço de nuvem para o desenvolvimento e teste de soluções robóticas. Para armazenamento é utilizado a plataforma Amazon S3, são armazenados nela os modelos treinados junto com os registro de treinamento.

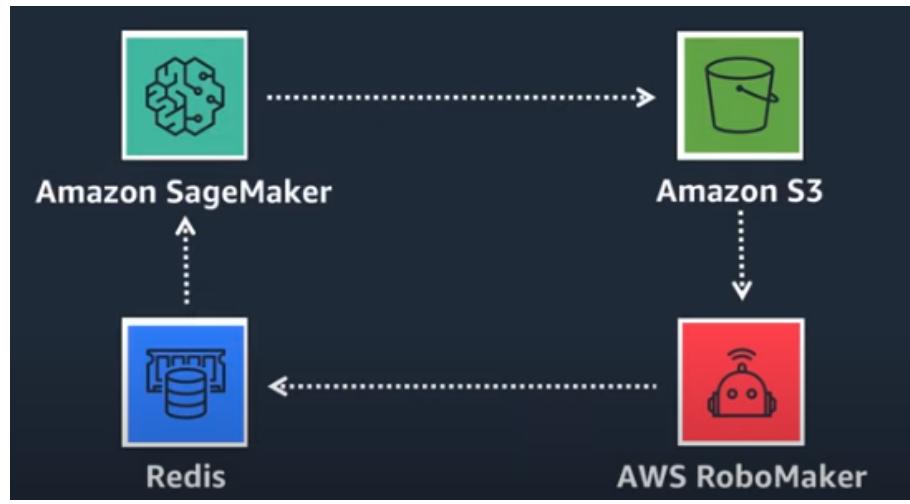


Figura 10: Comunicação entre os Serviços da AWS. [12].

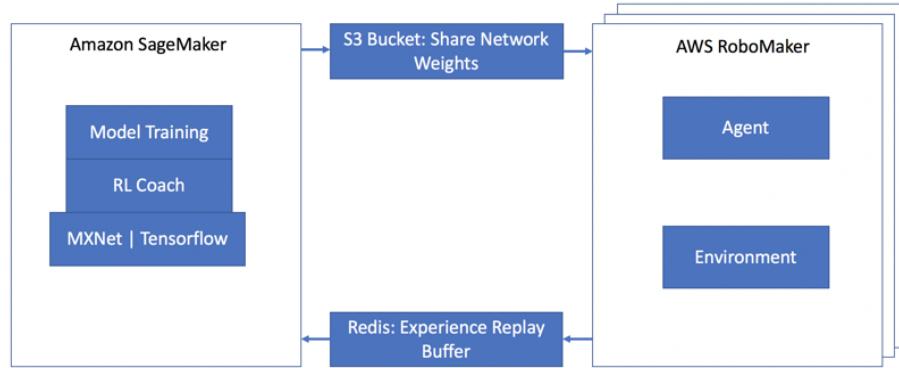


Figura 11: Arquitetura da AWS DeepRacer [12].

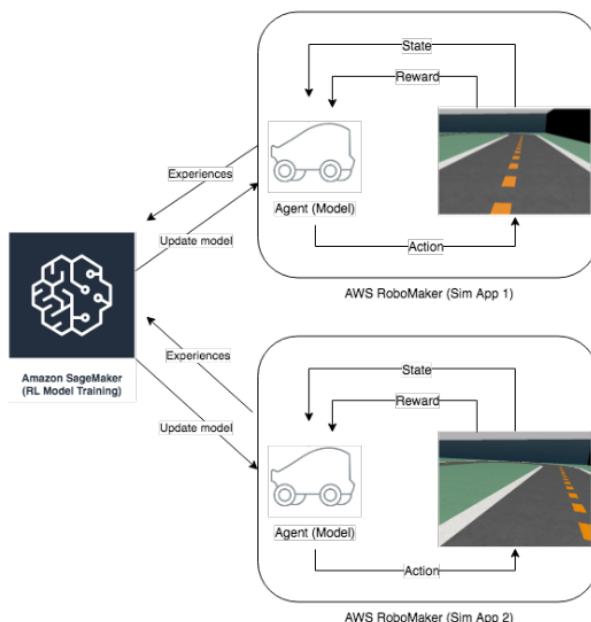


Figura 12: Esquemático da AWS DeepRacer [12].

### 3.6 Arquitetura da Rede

O algoritmo padrão no DeepRacer é o PPO que foi abordado em 2.6 algoritmo. Entretanto também existe a possibilidade de utilizar o Soft Actor-Critic (SAC).

O PPO usa duas redes neurais, uma rede de políticas e uma rede de valor. A rede de políticas, decide qual ação tomar com base na entrada (imagem da câmera e sensor LiDAR) [13]. A rede de valor, estima a recompensa cumulativa com base nas entradas. Das duas redes, a rede de política é a rede que interage e é implementada no carro. A Figura 13 mostra a arquitetura da rede.

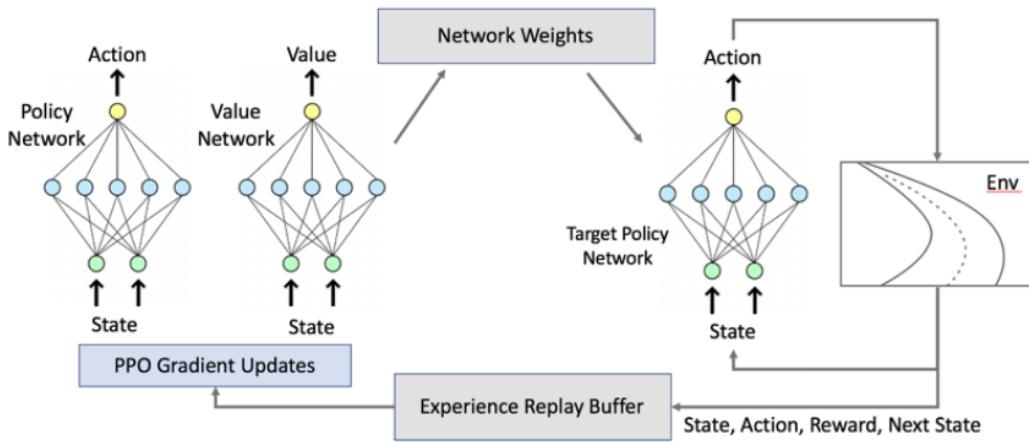


Figura 13: Arquitetura da Rede.

## 4 Implementação

O objetivo deste capítulo é estudar o comportamento de diferentes funções recompensa, para diversas pistas. Serão analisados as métricas de treinamento e as validações nas pistas. O Framework utilizado pelo SageMaker foi o TensorFlow [14] e o algoritmo de aprendizado por reforço foi o PPO, conforme abordado em 2.6. O tempo de treinamento foi definido como 1 hora e após o treinamento foi feito a validação em diferentes pistas, utilizando 3 tentativas.

### 4.1 Simulação I

O primeiro passo foi-se utilizar a função recompensa mostrada em 3.3. O sensor utilizado foi apenas a câmera e o espaço de estado foi o contínuo com velocidades entre  $0.5m/s$  e  $1m/s$  e o ângulo de direção entre  $-30^\circ$  e  $30^\circ$ . A pista escolhida foi a re:Invent 2018. Ela possui  $17.6m$  de comprimento e  $0.76m$  de largura.

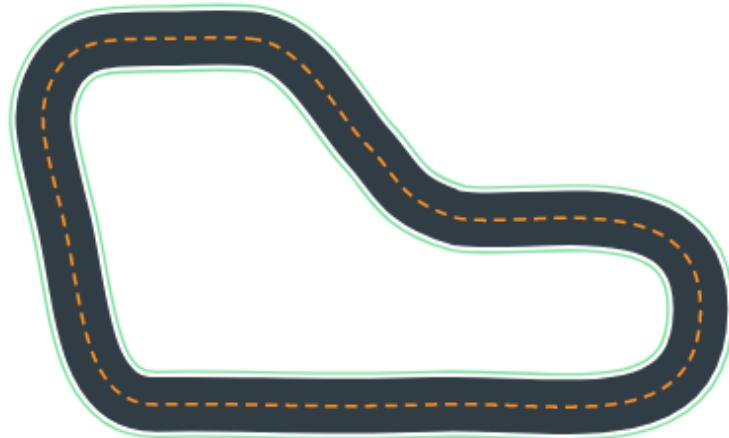


Figura 14: Pista Reivent 2018.

O Total de recompensa por episodio e a porcentagem de conclusão pode ser mostrado na Figura 15.

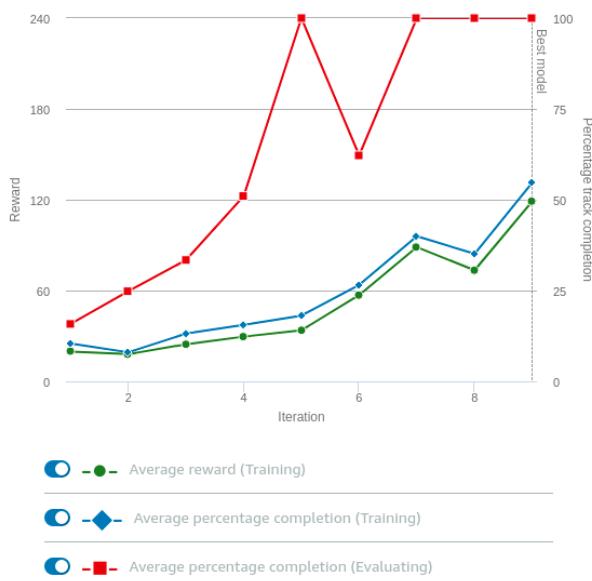


Figura 15: Métricas de Treinamento na Pista Reivent 2018 - Simulação I.

Com isso foi possível fazer a validação de cada tentativa, os resultados estão mostrados na tabela abaixo. A primeira coluna da Tabela é a tentativa, o tempo é dado em Minutos:Segundos:milissegundos, os resultados em % da pista concluída e o status, fora da pista ou volta completa.

Tabela 4: Validação na Pista Reivent 2018 - Simulação I.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	00:22:577	100%	Volta Completa

2	00:23:221	100%	Volta Completa
3	00:17:121	73%	Fora da Pista

Tabela 5: Hiperparâmetros na Pista Reivent 2018 - Simulação I.

Hiperparâmetro	Valor
Tamanho do Batch	64
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0003
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

Agora será feito a validação do mesmo modelo em outra pista, a pista escolhida foi a Cumulo Turnpike, esta pista possui um comprimento de  $60m$  e uma largura de  $1.06m$ , o objetivo vai ser analisar a performance do modelo em uma pista diferente da treinada.

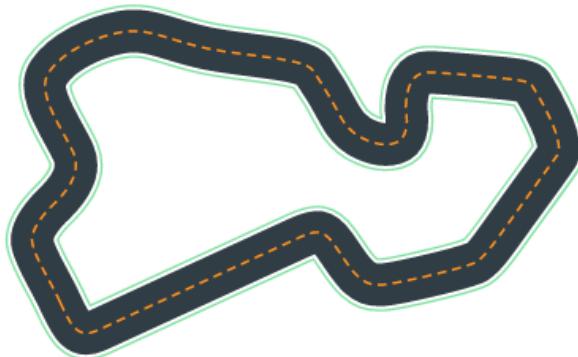


Figura 16: Pista Cumulo Turnpike.

Tabela 6: Validação na Pista Cumulo Turnpike - Simulação I.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	00:09:666	11%	Fora da Pista
2	00:04:132	4%	Fora da Pista
3	00:01.932	11%	Fora da Pista

Tabela 7: Hiperparâmetros na Pista Cumulo Turnpike - Simulação I.

Hiperparâmetro	Valor
Tamanho do Batch	64
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0003
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

## 4.2 Simulação II

A ideia foi realizar alterações na função recompensa e nos hiperparâmetros. Também foi alterado o Espaço de Estado para discreto, com os valores de velocidade e ângulo de direção sendo respectivamente  $[0.33, 0.67, 1] \frac{m}{s}$  e  $[-30, 0, 30]^{\circ}$ .

```

1 def reward_function(params):
2     reward = 0.001
3     if params["all_wheels_on_track"]:
4         reward += 1
5     if abs(params["steering_angle"]) < 5:
6         reward += 1
7     reward += (params["speed"] / 8)
8     return float(reward)

```

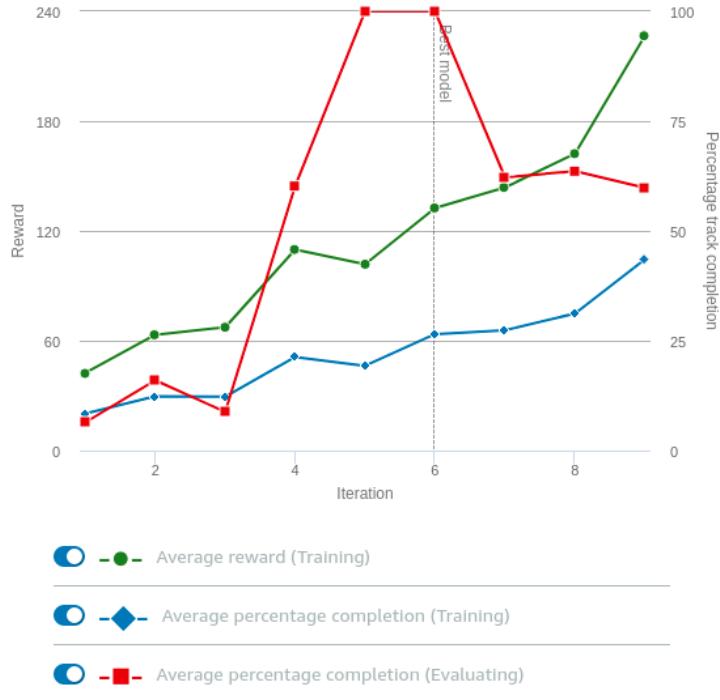


Figura 17: Métricas de Treinamento na Pista Reivent 2018 - Simulação II.

Tabela 8: Validação na Pista Reivent 2018 - Simulação II.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	00:28.782	100%	Volta Completa
2	00:27.148	100%	Volta Completa
3	00:13.224	47%	Fora da Pista

Tabela 9: Hiperparâmetros na Pista Reivent 2018 - Simulação II.

Hiperparâmetro	Valor
Tamanho do Batch	128
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0002
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

Tabela 10: Validação na Pista Cumulo Turnpike - Simulação II.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	00:01.321	9%	Fora da Pista
2	00:12.953	22%	Fora da Pista
3	00:08.926	17%	Fora da Pista

Tabela 11: Hiperparâmetros na Pista Cumulo Turnpike - Simulação II.

Hiperparâmetro	Valor
Tamanho do Batch	128
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0002
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

### 4.3 Simulação III

Nesta simulação, será repetido a Simulação I e II, com a única diferença é a pista de treinamento que será alterada para a Cumulo Turnpike.

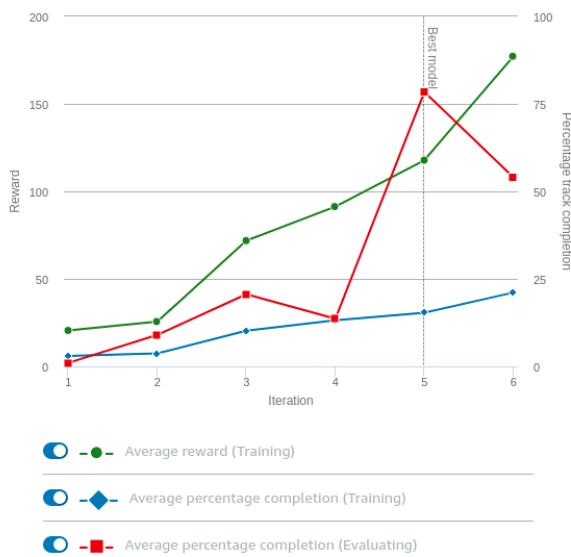


Figura 18: Métricas de Treinamento na Pista Cumulo Turnpike - Simulação III.

Tabela 12: Validação na Pista Cumulo Turnpike - Simulação II.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	01:21.523	100%	Volta Completa
2	00:46.592	56%	Fora da Pista
3	01:21.111	100%	Volta Completa

Tabela 13: Hiperparâmetros na Pista Cumulo Turnpike - Simulação III.

Hiperparâmetro	Valor
Tamanho do Batch	64
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0003
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

Tabela 14: Validação na Pista Cumulo Turnpike - Simulação III.

Tentativa	Tempo (MM:SS:mms)	Resultado	Status
1	00:08.321	100%	Fora da Pista
2	00:06.137	56%	Fora da Pista
3	00:21.332	92%	Fora da Pista

Tabela 15: Hiperparâmetros na Pista Reivent 2018 - Simulação III.

Hiperparâmetro	Valor
Tamanho do Batch	128
Número de Epocas (epochs)	10
Taxa de Aprendizado	0.0002
Entropia	0.01
Fator de Desconto	0.999
Tipo de Perda	Huber

## 5 Conclusões

Com isso podemos perceber que a AWS DeepRacer é uma excelente plataforma para entender o aprendizado por reforço e implementa-lo em um veículo autônomo. É possível treinar diferentes funções recompensas junto com algoritmos de otimização com o objetivo de rodar o carro físico de forma autônoma e não apenas o modelo virtual.

Também percebeu-se que para construir um modelo satisfatório pode ser bem desafiador e complexo pois o modelo requer uma alta precisão no método de treinamento e alguns casos precisamos lidar com os hiperparâmetros que são externos ao nosso modelos e obtidos de maneira empírica. Porém na pista apropriada, o carro é capaz de navegar com sucesso na pista de corrida que estava treinado exatamente como durante a avaliação simulada.

Conforme abordado abordado na Secção 4, não foi possível criar um modelo universal , para cada pista a função recompensa precisava ser ajustada e o modelo tinha que ser treinado naquela pista, modelos criados em pistas diferentes não obtiveram uma performance satisfatória em outras pistas e com isso o veículo não conseguia completar a volta. Entretanto notou-se que para pistas que fossem similares a pista de treinamento o modelo conseguia obter uma boa performance e em alguns testes até completar a volta, mas os melhores resultados se nas simulações feitas nas pistas em que o modelo foi treinado.

## Referências

- [1] M. Kamruzzaman G. Currie A. Faisal, T. Yigitcanlar. Understanding autonomous vehicles: A systematic literature review on capability, impact, planning and policy. *Journal of Transport and Land Use*, 12, 2019.
- [2] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press., 2018.
- [3] N. Akalin and A. Loutfi. Reinforcement learning approaches in social robotics. *arXiv*:,(2009.09689), 2020.
- [4] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial Neural Networks: A Practical Course*. Springer, 2017.
- [5] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.
- [6] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.
- [7] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.
- [8] Peter Watkins, Christopher J. C. H.; Dayan. Q-learning. machine learning. *Springer*, 8(5):679–684, 1992.
- [9] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv*:,(1707.06347), 2017.
- [11] Documentação OpenAI Spinning Up . <https://spinningup.openai.com/>. Acessado em Julho 2021.
- [12] Documentação AWS DeepRacer. <https://docs.aws.amazon.com/deepracer/latest/developerguide/what-is-deepracer.html>. Acessado em Julho 2021.
- [13] DeepRacer Car. [https://cse.buffalo.edu/~avereshc/rl\\_spring20/Xingtong\\_Li.pdf](https://cse.buffalo.edu/~avereshc/rl_spring20/Xingtong_Li.pdf). Acessado em Julho 2021.

[14] TensorFlow . <https://www.tensorflow.org/>. Acessado em Julho 2021.