



TRABALHO DE GRADUAÇÃO

**INTELIGÊNCIA COMPUTACIONAL  
PARA AGENTE ÚNICO DE FUTEBOL DE ROBÔS**

Bruno Andregretti Dantas  
Samuel Venzi Lima Monteiro de Oliveira

Brasília, Maio de 2021



**ENGENHARIA  
MECATRÔNICA**  
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA  
Faculdade de Tecnologia  
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

**INTELIGÊNCIA COMPUTACIONAL  
PARA AGENTE ÚNICO DE FUTEBOL DE ROBÔS**

**Bruno Andreghetti Dantas**  
**Samuel Venzi Lima Monteiro de Oliveira**

*Relatório submetido como requisito parcial de obtenção  
de grau de Engenheiro de Controle e Automação*

Banca Examinadora

Prof. Dr. Alexandre Ricardo Soares Romariz  
*Orientador*

---

**Brasília, Maio de 2021**

## Dedicatórias

*um beijo pra minha mãe, pro meu pai, e pra  
você*

*Samuel Venzi Lima Monteiro de Oliveira*

*Dedicatória do autor 2.*

*Bruno Andreghetti Dantas*

## Agradecimentos

*Agradecimentos!*

*Bruno Andregretti Dantas*

*A inclusão desta seção de agradecimentos é opcional e fica à critério do(s) autor(es), que caso deseje(em) inclui-la deverá(ão) utilizar este espaço, seguindo esta formatação.*

*Samuel Venzi Lima Monteiro de Oliveira*

---

## RESUMO

O cenário da aprendizagem de máquina tem crescido cada vez mais nos últimos anos. Junto a isso, iniciativas como a RoboCup buscam incentivar a aplicação dessas técnicas fomentando um cenário competitivo de futebol de robôs. Nesse trabalho foi desenvolvida uma nova plataforma de desenvolvimento voltada para a categoria *RoboCup Soccer Simulation 2D* e são aplicadas técnicas de aprendizagem por reforço a fim de validá-la. Foram realizados experimentos com técnicas estabelecidas como Sarsa e Q-Learning e tiveram como objetivo realizar o maior número de gols possíveis em uma partida.

Palavras Chave: Aprendizagem por reforço, RoboCup, Futebol de robôs.

---

## ABSTRACT

Abstract, in English ofc!

Keywords: Reinforcement learning, RoboCup, Robot soccer

# SUMÁRIO

<b>1</b>	<b>Introdução.....</b>	<b>1</b>
1.1	ROBÓTICA.....	1
1.2	APRENDIZAGEM POR REFORÇO.....	2
1.3	FUTEBOL DE ROBÔS.....	2
1.3.1	ROBOCUP SOCCER SIMULATION 2D.....	3
1.3.2	SERVIDOR DA PARTIDA.....	3
1.3.3	CLIENTE.....	4
1.3.4	SENSORES.....	5
1.3.5	AÇÕES.....	5
1.3.6	ABORDAGENS UTILIZADAS NA CATEGORIA.....	5
1.4	CARACTERIZAÇÃO DO PROBLEMA.....	6
1.4.1	OBJETIVOS.....	6
<b>2</b>	<b>Fundamentação Teórica.....</b>	<b>7</b>
2.1	PROCESSOS DE DECISÃO DE MARKOV.....	7
2.1.1	MDP EPISÓDICO E CONTÍNUO.....	8
2.1.2	RECOMPENSA E RETORNO.....	8
2.1.3	POLÍTICAS E FUNÇÕES DE VALOR.....	8
2.2	APRENDIZAGEM POR REFORÇO.....	9
2.2.1	APRENDIZAGEM ON-POLICY E OFF-POLICY.....	9
2.2.2	SOLUÇÕES TABULARES E APROXIMADAS.....	10
2.2.3	SARSA.....	10
2.2.4	Q-LEARNING.....	11
2.2.5	Q-LEARNING DUPLO.....	11
<b>3</b>	<b>Desenvolvimento.....</b>	<b>13</b>
3.1	BIBLIOTECA.....	13
3.1.1	ARQUITETURA DO CÓDIGO.....	13
3.1.2	DECODIFICAÇÃO E CODIFICAÇÃO DE MENSAGENS.....	14
3.2	DEFINIÇÃO DOS ESTADOS.....	14
3.2.1	SENSORES.....	15
3.2.2	ESTIMAÇÃO E TRATAMENTO DE ESTADOS.....	16
3.3	DEFINIÇÃO DE AÇÕES.....	17

3.3.1	ARRANCAR .....	17
3.3.2	CHUTAR .....	17
3.3.3	VIRAR.....	17
3.3.4	VIRAR PESCOÇO .....	18
3.3.5	MOVER-SE.....	18
3.3.6	AGARRAR.....	18
3.3.7	FALAR.....	19
3.4	AMBIENTE DE TREINAMENTO .....	19
3.4.1	LAÇO DE TREINAMENTO .....	19
3.5	EXPERIMENTOS .....	19
3.5.1	SARSA APROXIMADO E COMPORTAMENTOS PRÉ-PROGRAMADOS .....	20
3.5.2	DOUBLE Q-LEARNING TABULAR E COMPORTAMENTOS PRÉ-PROGRAMADOS..	22
3.5.3	DOUBLE Q-LEARNING TABULAR E AÇÕES PURAS .....	23
<b>4</b>	<b>Resultados Experimentais.....</b>	<b>25</b>
4.1	AGENTE ÚNICO COM SARSA APROXIMADO E COMPORTAMENTOS PRÉ-PROGRAMADOS	25
4.2	AGENTE ÚNICO COM DOUBLE Q-LEARNING TABULAR E COMPORTAMENTOS PRÉ-PROGRAMADOS .....	25
4.3	AGENTE ÚNICO COM DOUBLE Q-LEARNING TABULAR E AÇÕES PURAS.....	26
<b>5</b>	<b>Conclusões.....</b>	<b>29</b>
5.1	IMPLEMENTAÇÃO DA INTERFACE COM O SERVIDOR.....	29
5.2	TREINAMENTO DE EQUIPE PARA PARTICIPAÇÃO EM COMPETIÇÕES .....	29
	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>30</b>

# LISTA DE FIGURAS

1.1	Visualização de uma partida em andamento .....	4
1.2	Esquema ilustrando a arquitetura de um cliente e sua comunicação com o servidor do jogo.....	5
2.1	Interação agente-ambiente em um MDP [1].....	7
3.1	Indicadores espalhados pelo campo [2]. ....	16
3.2	Visualização da área agarrável pelo goleiro [2]. ....	18
4.1	Curva de aprendizado do agente com comportamentos pré-programados utilizando Sarsa aproximado. ....	26
4.2	Curva de aprendizado do agente com comportamentos pré-programados. ....	27
4.3	Sobreposição de sequência de imagens do agente fazendo gol.....	27
4.4	Curva de aprendizado do agente com ações puras. ....	28
4.5	Curva de aprendizado para treinamento longo.....	28



# LISTA DE TABELAS

3.1	Estrutura da rede neural utilizada no experimento .....	20
-----	---	----

# LISTA DE ALGORITMOS

3.1	Pseudo-código que descreve a rotina de treinamento . . . . .	19
3.2	Algoritmo do comportamento de se aproximar da bola . . . . .	21
3.3	Algoritmo geral do comportamento de carregar bola . . . . .	21
3.4	Algoritmo geral do comportamento de chutar bola para o gol . . . . .	21

# LISTA DE SÍMBOLOS

## Símbolos

$\pi$	Política de tomada de ação
$\gamma$	Fator de desconto
$\alpha$	Fator de aprendizagem
$\epsilon$	Fator de exploração

## Subscritos

$t$	Ciclo $t$ de treinamento
$terminal$	Ciclo que encerra episódio
$q_\pi$	Função de esperança do retorno de acordo com a política $\pi$

## Sobrescritos

$*$	Política ótima
-----	----------------

## Siglas

IA	Inteligência artificial
RL	Aprendizagem por reforço - <i>Reinforcement learning</i>
RCSS	<i>RoboCup Soccer Simulation 2D</i>
MDP	Processo de decisão de Markov - <i>Markov decision process</i>

# Capítulo 1

## Introdução

O interesse humano em criar artefatos para facilitar seu próprio trabalho ou realizar uma tarefa sem interferência remonta os tempos mais antigos. No Egito Ptolomaico, Ctesíbio (285-222 AC) descreveu um relógio d'água com a presença de um sistema de engrenagens, um indicador e o primeiro sistema de retroalimentação registrado. Por volta de 1495, Leonardo da Vinci, concebeu o projeto de um autômato mecânico de um guerreiro em armadura medieval que podia ficar em pé, sentar-se, levantar o visor e mover os braços. [3]

O estudo da união de sistemas eletromecânicos e inteligência teve começo há pelo menos 70 anos. A *cibernética*, área inaugurada por Norbert Wiener na década de 1950, descreve o estudo científico de controle e comunicação no animal e na máquina. Wiener começou, então, a desenvolver sistemas que replicassem comportamentos animais.[4] Somado a isso, a teoria da informação de Claude Shannon e a teoria de computação de Alan Turing abriram espaço para pesquisas que iriam desenvolver Inteligências Artificiais (IA). [5]

### 1.1 Robótica

A robótica se apoia em conhecimentos de vários campos para criar uma das áreas de estudos mais amplas da ciência. Desde a metade do século XX, a robótica vêm reunindo noções dessas áreas, e pouco a pouco as tornando partes essenciais de si: sistemas mecânicos, eletromecânicos, teoria de controle, IA e outras. As aplicações existentes são inúmeras e se renovam a todo momento. Dentre as principais, é possível citar sistemas de manufatura, robótica médica e robótica agrícola. [6]

Sistemas robóticos construídos para automatizar tarefas repetitivas são interessantes. Entretanto, o avanço das indústrias e aumento da complexidade das tarefas a serem realizadas criou um ambiente catalisador para o desenvolvimento de processos de tomada de decisão autonomamente.

É importante notar a complexidade do problema de se desenvolver a tomada de decisão de um sistema autônomo. Tal sistema precisa mapear seu ambiente por meio de sensores, extrair um significado do seu estado atual, usá-lo para decidir uma ação e determinar se tal ação foi a

melhor a ser tomada. Sensores, porém, são imprecisos e limitados fisicamente. A representação dos estados, frequentemente, não é completamente conhecida. E o processo de mapeamento de estado para ação não é trivial.

Neste contexto surgiu a área de estudo conhecida como *aprendizagem por reforço*, que formaliza os elementos citados anteriormente para prover uma base de como agentes devem tomar ações para cumprir um objetivo pré-definido. [1]

## 1.2 Aprendizagem por Reforço

A aprendizagem por reforço (do inglês, *reinforcement learning* ou RL) tem como inspiração a maneira como o aprendizado acontece com seres-humanos: interagindo com o ambiente. [1] Se uma criança está aprendendo a andar, por exemplo, ela toma certas ações no ambiente e, ainda que inconscientemente, está atenta aos resultados que essa ação causa.

A teoria por trás da aprendizagem por reforço formaliza a ideia de aprender através da interação e a aplica em um contexto computacional. Os principais elementos de um sistema de RL são: uma política de decisão, um sinal de recompensa e uma função valor. O primeiro diz respeito à decisão de qual ação se tomar a partir de uma situação, o segundo quantifica quão boa foi a ação escolhida naquele momento e o terceiro quantifica quão boa é a ação considerando o longo prazo.[1]

Apesar de recente, a técnica de RL já se mostrou promissora em diversas áreas, com destaque para seu uso em jogos. Em 2016, o programa AlphaGo mostrou resultados significativos ao jogar contra o campeão europeu de Go, superando-o nos 5 jogos que foram disputados. [7]

Em 2018, pesquisadores do grupo OpenAI utilizaram técnicas de RL para treinar um time de 5 agentes colaborativos no jogo *Dota 2*, um jogo de estratégia em tempo real onde 2 times batalham para destruir a base inimiga. O jogo provê um ambiente extremamente complexo, com espaços de estados e ações contínuos. São 20.000 números ponto-flutuante para estados e 1.000 ações possíveis em um dado ciclo. A duração usual de uma partida é de pelo menos 1 hora. Inicialmente, foi feito um sistema do tipo um contra um (1v1) e o agente resultante deste treinamento foi capaz de derrotar jogadores profissionais. Em 2019, o sistema com 5 agentes foi capaz de derrotar um time profissional. [8] Um resultado dessa magnitude foi possível devido, entre outras razões, ao número altíssimo de amostras coletadas pelos agentes: 300 anos de experiência por dia para o agente singular e 180 anos por dia por agente para o time contendo 5 membros.

## 1.3 Futebol de Robôs

A ideia de robôs jogando futebol foi proposta pela primeira vez em 1992 por Alan Mackworth[9]. Desde então a comunidade científica tem criado iniciativas buscando por soluções que tornem isso realidade. Desde então a comunidade científica tem criado iniciativas buscando por soluções que tornem isso realidade.

Uma delas é a *Robot World Cup Initiative* [10], abreviada como *RoboCup*, que teve sua primeira

edição em 1997 com mais de 40 equipes distribuídas entre as diversas categorias do evento.

O objetivo da iniciativa, definido pela *RoboCup Federation*, é que por volta da metade do século XXI, um time de robôs humanóides autônomos vençam uma partida contra os campeões da Copa do Mundo mais recente. Mesmo que o objetivo pareça ambicioso, ele guia as pesquisas e motiva o avanço no campo.

Atualmente, a RoboCup conta com mais de 10 categorias, incluindo robôs humanoides, robôs com rodas e simulações. Entre elas há a *RoboCup Soccer Simulation 2D*, abreviada RCSS, objeto de estudo deste projeto.

### 1.3.1 RoboCup Soccer Simulation 2D

A RCSS possui grande relevância internacional, sendo uma das principais categorias disputadas na RoboCup, com equipes do mundo inteiro.

A categoria apresenta, também, grande relevância no cenário brasileiro. Desde 2005, a RCSS está presente na maior competição de robótica da América Latina, a *Latin American Robotics Competition*, LARC.

Nessa categoria, duas equipes de 11 jogadores autônomos e independentes jogam futebol em um ambiente virtual bidimensional. Um servidor é responsável por esse ambiente e possui informação absoluta sobre o estado do jogo e suas regras. Os jogadores, por sua vez, recebem dele informação incompleta e ruidosa de seus sensores virtuais, podendo executar comandos a fim de atuar sobre o estado do jogo. [2]

### 1.3.2 Servidor da partida

Um servidor que executa a partida é disponibilizado pelos organizadores da competição e este pode ser utilizado, também, para desenvolvimento. O servidor, portanto, apresenta, internamente, algumas das regras da partida bem como um juiz autônomo que age para determinar gols, faltas e demais situações de uma partida de futebol. Caso necessário, um juiz humano poderá intervir em situações não contempladas pelas regras do servidor.

O servidor simula todos os movimentos e ações dos jogadores e da bola. Clientes externos se conectam ao servidor e cada cliente controla um único jogador. A comunicação entre o cliente e o servidor é feita a partir do protocolo UDP por meio de mensagens com sintaxe específica e definida pelo servidor.

De forma a permitir o acompanhamento visual da partida, um monitor também é disponibilizado, porém não é necessário para que uma partida ocorra com sucesso.

O servidor, ainda, possui o modo *trainer* para utilização durante treinamentos de algoritmos de inteligência computacional. Este modo permite a conexão de um cliente do tipo treinador que tem acesso absoluto às informações da partida e pode mudar modos de jogo e ainda mover arbitrariamente jogadores e bola. Adicionalmente, é possível acelerar os ciclos da partida permitindo

o treinamento em tempo hábil.



Figura 1.1: Visualização de uma partida em andamento

### 1.3.3 Cliente

Os jogadores são controlados por clientes externos conectados ao servidor. Como já foi dito, um cliente corresponde a um único jogador e os clientes só podem ser comunicar com mensagens mandadas através do servidor da partida.

O cliente pode ser desenvolvido em qualquer linguagem desde que se comunique com o servidor pelo protocolo UDP e utilize a sintaxe de mensagens reconhecida pelo sistema. Há várias escolhas disponíveis para a construção do cliente, sendo decisão de cada equipe competidora como fazê-lo.



Figura 1.2: Esquema ilustrando a arquitetura de um cliente e sua comunicação com o servidor do jogo.

### 1.3.4 Sensores

Cada jogador presente na partida possui um conjunto de sensores de onde são tiradas todas as informações sobre o ambiente. Em uma partida usual, um jogador tem informações visuais dos jogadores do seu time e do time adversário, da bola e de uma série de marcadores fixos no campo, como bandeiras e linhas, que servem para situar o jogador em coordenadas absolutas do campo. O jogador possui também informações “sonoras”, onde pode ouvir mensagem do árbitro, treinador e de outros jogadores. Por último, tem acesso a informações do próprio corpo, como orientação do corpo e pescoço. [2]

Os sensores possuem características que os aproximam de sensores reais como perda de resolução da informação conforme a variável medida se afasta do sensor.

### 1.3.5 Ações

A cada ciclo de simulação, cada cliente conectado ao servidor pode realizar ações que terão efeito no ambiente.[2]

As ações englobam mover-se, virar-se, chutar a bola e até falar, permitindo troca de mensagens entre os jogadores. As ações disponíveis serão detalhadas no decorrer do texto.

### 1.3.6 Abordagens utilizadas na categoria

Uma pesquisa sobre as abordagens para o desenvolvimento das estratégias dos times participantes da RCSS revelou o uso recorrente de métodos de inteligência computacional.

A equipe chinesa *WrightEagle*, campeã do principal evento internacional da categoria diversas vezes, utiliza Processos de Decisão de Markov ou MDPs para modelar a partida[11].



A equipe japonesa *HELIOS*, campeã de 2018 da categoria na RoboCup, divide seus jogadores em categorias “chutadores” e “não-chutadores”. Os chutadores são responsáveis por realizar o planejamento de sequência de ações, utilizando métodos de valor de ação. Os não-chutadores, por sua vez, não tem conhecimento do planejamento feito pelos chutadores, e devem obter o máximo de informações relevantes para tentar gerar a mesma sequência de ações que jogador chutador[12].

A equipe brasileira *ITAndroids*, atual campeã da LARC, utiliza a abordagem de sequência de ações, similar à *HELIOS*, explorando uma árvore de ações criada dinamicamente de forma a maximizar o valor de cada ação. Além disso, utilizam Otimização por Enxame de Partículas [13] para adequar os parâmetros que calculam o valor da ação. A *ITAndroids* também vem desenvolvendo o uso de Aprendizagem por Reforço Profunda [14].

Muitas equipes, ainda, desenvolvem seus agentes utilizando o agente base da equipe *HELIOS*, *Agent2d* com a biblioteca *Librcsc*, escritas em C++. Por isso, é comum que haja semelhança na construção dos agentes dessas equipes.

## 1.4 Caracterização do Problema

Deseja-se, então, explorar o problema de se fazer gols com um agente único no ambiente descrito utilizando-se de técnicas de aprendizagem por reforço.

Para isso, foi desenvolvida uma biblioteca de interfaceamento com o servidor da partida como adaptação do ambiente. Após isso o agente foi treinado com a utilização de técnicas de RL em duas abordagens - ações puras e comportamentos - de forma a compará-las.

No Capítulo 2, descreve-se os fundamentos teóricos da aprendizagem por reforço, com destaque para os algoritmos de *Q-learning* e *Q-learning* duplo. No Capítulo 3, descreve-se o desenvolvimento da biblioteca e a definição de ações e comportamentos. Além disso, propõe-se o algoritmo de treinamento e os experimentos a serem realizados. No Capítulo 4, os resultados desses experimentos e suas análises são apresentados. Finalmente, o Capítulo 5 expõe conclusões e trabalhos futuros.

### 1.4.1 Objetivos

De acordo com o contexto apresentado, o presente trabalho se propõe a cumprir as seguintes etapas:

- Implementar uma biblioteca de interfaceamento para comunicação com o servidor
- Utilizar técnicas de aprendizagem por reforço para treinar a escolha de ações puras
- Utilizar técnicas de aprendizagem por reforço para treinar a escolha de comportamentos
- Comparar as diferentes configurações de treinamento e seus resultados

## Capítulo 2

# Fundamentação Teórica

### 2.1 Processos de Decisão de Markov

O problema abordado neste trabalho pode ser descrito simplificadaamente como um Processo de Decisão de Markov (MDP). MDP é uma forma clássica de representação matemática de processos de decisão sequenciais. Nesse modelo de representação, cada ação realizada por um agente que interage com o ambiente transforma o estado do processo e determina a recompensa que o agente recebe. Em um MDP, a codificação do estado atual deve conter toda a informação sobre a interação entre o agente e o ambiente que seja relevante para a dinâmica futura do processo. Nesse caso, é dito que o estado possui a "propriedade de Markov".



Figura 2.1: Interação agente-ambiente em um MDP [1].

Assim, dado um espaço de estados  $\mathcal{S}$ , um espaço de ações  $\mathcal{A}$  e um espaço de recompensas  $\mathcal{R}$ , para cada par  $(S, A)$  com  $S \in \mathcal{S}$  sendo o estado atual do processo e  $A \in \mathcal{A}$  a ação tomada pelo agente existe uma determinada probabilidade de atingir o estado  $S' \in \mathcal{S}$  e receber a recompensa  $R \in \mathcal{R}$  [1].

Essa abordagem é bastante flexível e torna possível modelar a dinâmica do futebol virtual de robôs de diversas maneiras de modo que cada agente possa construir um estado percebido a partir de seus sensores e tomar decisões acerca de qual a melhor ação a fim de maximizar a recompensa recebida.

### 2.1.1 MDP Episódico e Contínuo

Um MDP pode ser caracterizado quanto à presença de um estado terminal. Caso o MDP tenha um ou mais estados que determinem o fim do processo, ele é dito episódico, e a totalidade do processo até o estado que determina o fim é chamada de episódio. A simulação de futebol de robôs tratada neste trabalho é um exemplo de MDP episódico, uma vez que o MDP termina ao se encerrar o tempo de jogo.

Em contrapartida, há MDPs onde não está bem definido nenhum estado terminal. Nesses casos, o MDP pode continuar indefinidamente até que uma ação externa ao MDP determine a sua parada. Um exemplo disso é um MDP que controle um robô numa linha de produção. Caso o sistema de automação supervisor desse robô não determine sua parada (por falta de insumos, por exemplo), o MDP pode seguir operando indefinidamente.

### 2.1.2 Recompensa e Retorno

Como definido acima, para cada ação tomada em um MDP é atribuída uma recompensa  $R \in \mathcal{R}$ . Essa recompensa é sempre referente ao instante de tempo anterior, ou seja, não depende de qualquer outro fator que não o par  $(S_t, A_t)$  executados no instante  $t$  e a função de probabilidade associada pelo MDP a esse par. Por isso, é comum utilizar a notação  $R_{t+1}$  para se referir à recompensa obtida após tomar a ação  $A_t$  no instante de tempo  $t$ .

Porém em muitos casos é esperado de um agente que ele tome decisões que maximizem a recompensa total a longo prazo, ou seja, em um MDP episódico é esperado que se escolha  $A_t$  a fim de maximizar não apenas  $R_{t+1}$  mas sim o retorno  $G_t$  tal que:

$$G_t = R_{t+1} + R_{t+2} + \dots + R_{terminal}. \quad (2.1)$$

Caso o MDP seja contínuo,  $G_t$  poderia divergir, uma vez que seja uma soma infinita de parcelas. Para solucionar o problema, basta adicionar à equação um fator de desconto ( $\gamma$ ), tal que seja possível ajustar a relevância de recompensas futuras para o cálculo do retorno:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \quad (2.2)$$

Observa-se que, para  $0 \leq \gamma < 1$ , o somatório converge. Para  $\gamma = 0$ , o retorno leva em consideração apenas a recompensa imediata  $R_{t+1}$ . Em contrapartida, para  $\gamma \rightarrow 1$ , as recompensas mais distantes no futuro são cada vez menos descontadas, ou seja, o agente tende a levar mais em consideração os ganhos futuros.

### 2.1.3 Políticas e Funções de Valor

É dado o nome de política para qualquer função  $\pi : (S, A) \rightarrow \mathbb{R}$  que associe um estado qualquer do MDP e uma ação a uma probabilidade de se tomar essa ação diante desse estado, ou

seja,  $\pi(A|S) = p$  onde  $p$  é a probabilidade de selecionar a ação  $A$  diante do estado  $S$ .

Para cada política  $\pi$  existe uma função  $v_\pi : (S) \rightarrow \mathbb{R}$  que, para cada estado, define a esperança de retorno caso o agente continue seguindo a política  $\pi$ . A função  $v_\pi$  é conhecida como função de valor sob a política  $\pi$ .

Similarmente, existe uma função  $q_\pi : (S, A) \rightarrow \mathbb{R}$  que, para cada par de estado e ação, define a esperança de retorno. Neste caso, a função  $q_\pi$  é conhecida como função de valor da ação sob a política  $\pi$ .

É possível comparar duas políticas  $\pi$  e  $\pi'$  a respeito de suas funções  $q$ . A política  $\pi$  é considerada melhor ou igual a  $\pi'$ , ou  $\pi \geq \pi'$ , caso  $q_\pi(S, A) \geq q_{\pi'}(S, A)$  para todo par  $(S, A)$ .

Sempre há ao menos uma política melhor ou igual a todas as outras, denominada política ótima. Qualquer política que cumpra esse requisito é denominada  $\pi_*$  e, caso haja mais de uma, todas devem possuir a mesma função  $q$  denominada  $q_*$  [1]. No decorrer deste trabalho serão utilizadas técnicas que buscam estimar  $q_*$ .

Uma política que toma sempre o caminho de maior retorno é denominada gulosa, e uma política que toma o caminho de maior retorno mas escolhe uma ação aleatoriamente com probabilidade parametrizada  $\epsilon$  é denominada  $\epsilon$ -gulosa.

## 2.2 Aprendizagem por Reforço

Dada uma modelagem do problema como um MDP, resta obter uma maneira de estimar as probabilidades que determinam a dinâmica desse MDP para determinar a política capaz de maximizar a recompensa a longo prazo recebida pelo agente. O conjunto de técnicas que resolvem esse tipo de problema é chamado de Aprendizagem por Reforço.

No campo da aprendizagem de máquina, ela se difere da Aprendizagem Supervisionada por não haver um conjunto de pares  $(s, a)$  considerados como verdade fundamental. Nesse tipo de aprendizagem, o objetivo é extrapolar uma solução genérica a partir de exemplos de um conjunto de treinamento dado como correto, o que não é prático em problemas em que não se tem exemplos de comportamentos esperados e que representem bem o conjunto total de situações possíveis. Ela também se diferencia da Aprendizagem Não-Supervisionada, que tradicionalmente visa encontrar estrutura em conjuntos de dados não classificados, enquanto a Aprendizagem por Reforço visa maximizar um sinal de recompensa [1].

Desse modo, as técnicas de Aprendizagem por Reforço serão aplicadas a fim de buscar políticas capazes de maximizar o desempenho dos jogadores virtuais, ou seja, obter políticas que tornem os agentes capazes de fazer gols e evitar que os jogadores do time adversário façam gols.

### 2.2.1 Aprendizagem On-policy e Off-policy

Entre as técnicas de aprendizagem por reforço existe uma divisão entre a aprendizagem on-policy e a aprendizagem off-policy, referentes à relação entre a política executada durante o apren-

dizado e a política sobre a qual se quer aprender.

Nos algoritmos de aprendizagem on-policy o agente aprende a respeito da política  $\pi$  enquanto navega o MDP de acordo com a própria política  $\pi$ , ou seja, a política executada durante a aprendizagem é a mesma que se quer estudar.

Em contrapartida, nos algoritmos de aprendizagem off-policy o agente aprende a respeito da política alvo  $\pi$  enquanto navega o MDP de acordo com a política  $b$ , ou seja, ele estima a função  $q_\pi$  enquanto executa a política  $b$ . Esses métodos costumam introduzir variância no processo, tornando o aprendizado ruidoso e muitas vezes divergente.

Além disso, é possível observar que a aprendizagem on-policy é apenas um caso particular da aprendizagem off-policy em que  $b = \pi$ .

### 2.2.2 Soluções Tabulares e Aproximadas

A maioria dos métodos de aprendizagem por reforço são testados e validados em MDPs cujos espaços de estados  $\mathcal{S}$  e de ações  $\mathcal{A}$  são suficientemente pequenos. Para esses MDPs é possível utilizar uma solução tabular, ou seja, a função  $Q$  pode ser armazenada em uma tabela de tamanho razoável e sua imagem para cada par estado-ação pode ser atualizado individualmente.

Infelizmente em diversas aplicações a quantidade de estados possíveis é grande demais ou até mesmo infinita, como é o caso de sistemas em que determinada característica do estado é medida como uma grandeza contínua. Nesses casos, é impossível esperar que se obtenha soluções ótimas mesmo com tempo infinito, portanto o objetivo é obter uma solução aproximada que seja boa o suficiente para a aplicação desejada.

A ferramenta matemática utilizada para viabilizar soluções aproximadas é o conceito de aproximadores de função, muito utilizados na aprendizagem supervisionada. Entre os aproximadores mais utilizados estão os aproximadores lineares e as redes neurais multicamada.

### 2.2.3 Sarsa

Entre os algoritmos de aprendizagem on-policy está o Sarsa. Seu nome é derivado da quintupla  $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$  utilizada como entrada da sua fórmula de atualização:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2.3)$$

Uma vez que a política adotada pelo agente não pode ser gulosa, uma vez que ela inibiria a exploração do espaço de estados e de ações, é interessante que o agente siga uma política  $\epsilon$ -gulosa baseada na estimativa de  $Q$ . Nesse caso, para  $Q$  tabular, a convergência para a política ótima é garantida desde que todos os pares  $(S, A)$  sejam visitados infinitas vezes e a política convirja para a política gulosa em  $t \rightarrow \infty$ .

Para o caso em que se usa um aproximador para estimar  $Q$  não há garantias, porém a natureza on-policy do Sarsa reduz a variância da aprendizagem. Isso pode tornar essa abordagem mais

estável do que o Q-learning, descrito a seguir.

### 2.2.4 Q-Learning

Um dos algoritmos mais populares no campo da aprendizagem por reforço é o Q-Learning. Trata-se de um método off-policy que aproxima diretamente a função  $q_*$  independente da política que estiver sendo adotada pelo agente durante o treinamento.

O algoritmo é também muito simples. Dada uma representação tabular  $Q : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}$  da função  $q_*$ , para cada instante de tempo  $t$  é realizada a seguinte atualização a fim de aproximar  $Q$  de  $q_*$ :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (2.4)$$

Sendo  $\alpha$  o fator de aprendizagem, responsável por suavizar o impacto de cada atualização da tabela, e  $\gamma$  o fator de desconto, responsável por reduzir a relevância de recompensas muito distantes no tempo.

Após iterações suficientes, espera-se que  $Q$  convirja para  $q_*$ . Em certas condições a convergência é matematicamente garantida.

Uma vez estimada a função  $q_*$ , é simples obter a política ótima. Basta escolher a ação que maximiza  $q_*$  no estado atual, ou seja:

$$A_t = \max_a q_*(S_t, a) \quad (2.5)$$

É comum, mas não obrigatório, que a política  $b$  seguida durante o aprendizado seja *epsilon*-gulosa em relação à aproximação  $Q$ .

### 2.2.5 Q-Learning Duplo

Apesar de popular o Q-Learning possui um problema de viés de maximização. Uma vez que a aproximação  $Q$  é imprecisa no início do treinamento, é possível que o retorno esperado estimado seja enviesado para um valor maior do que o real.

Como solução para esse problema é utilizada a abordagem do Q-Learning Duplo. Nela são utilizadas duas aproximações,  $Q_1$  e  $Q_2$ , e a atualização de  $Q$  é dada da seguinte forma:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)] \quad (2.6)$$

Em metade das iterações (através de um sorteio, por exemplo), as aproximações  $Q_1$  e  $Q_2$  são trocadas. Com isso é anulado o viés de maximização gerado pelo uso de  $\max_a Q$  como estimativa de retorno para os estados seguintes.

A vantagem desse método é que apesar de dobrar os requisitos de memória do algoritmo, afinal será preciso armazenar os dados referentes a duas aproximações, ele não aumenta o custo computacional por iteração.

## Capítulo 3

# Desenvolvimento

### 3.1 Biblioteca

O servidor da partida apresenta, como já mencionado na Subseção 1.3.2, um protocolo de comunicação e sintaxe de mensagens específicos. Uma biblioteca de interfaceamento foi desenvolvida com o objetivo de abstrair os detalhes de comunicação e de construção de mensagens e facilitar, assim, o desenvolvimento dos programas jogadores. Esta abordagem já é comum na categoria e existem soluções de código aberto como a *librcsc*, utilizada por várias equipes, usualmente atreladas ao agente base *agent2d*, desenvolvidas pela equipe *HELIOS*, como visto na Subseção 1.3.6.

A biblioteca própria foi desenvolvida em linguagem Go como forma de modernização e diversificação da base de código utilizada pelas equipes. A biblioteca cobre uma parte considerável das possibilidades previstas no protocolo de comunicação e foi programada de modo a ser facilmente expansível de acordo com o lançamento de atualizações do servidor.

#### 3.1.1 Arquitetura do código

A biblioteca possui três pacotes internos: *playerclient*, *trainerclient* e *rcsscommon*. Os dois primeiros dizem respeito aos dois tipos de programas que podem se conectar ao servidor da partida: jogadores e treinadores. O terceiro engloba todas as funcionalidades utilizadas por ambos clientes, além de informações gerais sobre parâmetros da partida, como coordenadas de bandeiras do campo e modos de jogo.

Os dois clientes desenvolvidos possuem as funcionalidades necessárias para se conectar ao servidor, ouvir mensagens via protocolo UDP, decodificá-las e então executar uma ação em forma de mensagem codificada e enviada ao servidor.



### 3.1.2 Decodificação e Codificação de Mensagens

A decodificação de mensagens, por sua vez, foi feita em duas camadas: um analisador léxico e um analisador sintático. O analisador léxico passa pelas mensagens em formato *string* e identifica os símbolos que ela contém. O analisador sintático extrai desses símbolos as informações e as organiza em estruturas de dados para que possam ser utilizadas fora da biblioteca. As informações recebidas e decodificadas são, em sua maioria, dados dos sensores do jogador.

```
(see 37 ((f c b) 16.6 -1 -0 -0.8))
```

Exemplo de mensagem codificada.

```
SightSymbols{
  Time: 37,
  ObjMap: map[string][]string{
    "f c b": {"16.6", "-1", "-0", "-0.8"},
  },
}
```

Mensagem após passar pelo analisador léxico.

```
SightData{
  Time: 37,
  Ball: nil,
  Lines: LineArray{},
  Flags: FlagArray{
    {
      ID:          rcsscommon.FlagCenterBot,
      Distance: 16.6,
      Direction: -1,
    },
  },
}
```

Mensagem após passar pelo analisador sintático.

## 3.2 Definição dos Estados

As informações de estado são fornecidas pelos sensores do jogador. Há três sensores presentes que entregam informações em forma de mensagens para o agente: sensor auditivo, sensor visual e sensor corporal.

Além de ser necessário decodificar as mensagens recebidas, como demonstrado na Subseção 3.1.2, é necessário processar os dados dos sensores para extrair mais características do estado atual do agente.

### 3.2.1 Sensores

Os sensores são responsáveis por todas as informações que o jogador tem do ambiente. Eles são modelados de forma a emular características de sensores reais, portanto um sensor pode “perder” informações caso o objeto da variável medida esteja longe, como será evidenciado no modelo do sensor visual.

#### 3.2.1.1 Sensor Auditivo

As mensagens do sensor auditivo são do seguinte formato:

*(hear Tempo Remetente "mensagem")*

Onde *Tempo* é o número do ciclo em que a mensagem foi ouvida e *Remetente* é descrição de quem enviou a mensagem. O *Remetente* pode ser o árbitro, outros jogadores, um dos treinadores ou o próprio jogador.

No escopo deste trabalho, apenas as mensagens do árbitro serão consideradas, não sendo implementada nenhuma forma de comunicação direta entre os agentes.

#### 3.2.1.2 Sensor Visual

As mensagens do sensor visual contém as posições relativas referentes a cada objeto dentro do campo de visão do jogador. Esses objetos podem ser outros jogadores, marcadores como bandeiras e linhas (Figura 3.1) e a bola. O formato genérico é este:

*(see (Objeto1)(Objeto2)(Objeto3)...(ObjetoN))*

Onde cada objeto tem o seguinte formato:

*((NomeDoObjeto) Distância Direção VariaçãoDeDistância VariaçãoDeDireção DireçãoDoCorpo DireçãoDaCabeça)*

Sendo a distância e a direção dados em coordenadas polares, assim como suas variações. As informações de direção do corpo e da cabeça só aparecem quando o objeto em questão é um outro jogador.

O campo é marcado com vários indicadores com posição fixa conhecida para que o agente possa estimar sua posição absoluta, como vista na Figura 3.1.

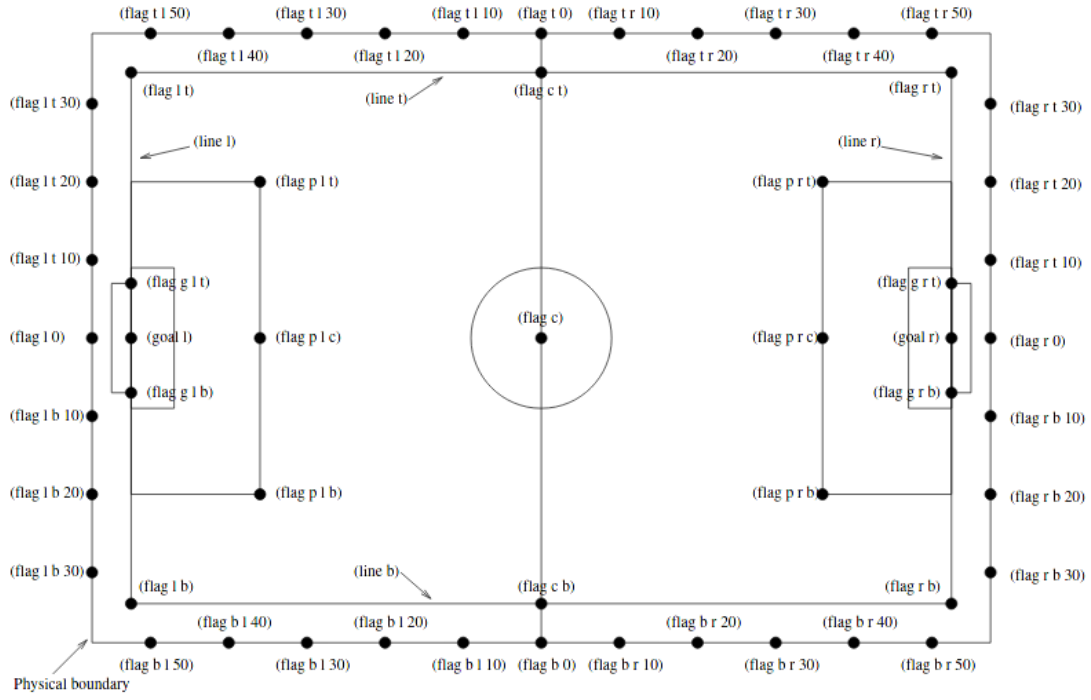


Figura 3.1: Indicadores espalhados pelo campo [2].

A riqueza de detalhes a respeito das informações obtidas depende da distância entre o objeto e o jogador. Por exemplo, caso um outro jogador esteja sendo visto a uma distância muito grande, pode ser impossível determinar o número de sua camisa ou até mesmo a qual time ele pertence. Em contrapartida, para jogadores próximos, é fornecido até mesmo a direção para a qual ele está olhando.

### 3.2.1.3 Sensor Corporal

O sensor corporal contém informações sobre o estado físico do jogador. Entre elas sua energia, que é consumida a cada ação tomada como chute ou arrancada (Subseção 1.3.5), sua própria velocidade e direção de movimento, a direção de sua cabeça e a quantidade de cartões de advertência recebidos.

### 3.2.2 Estimação e Tratamento de Estados

Os sensores do jogador fornecem somente informações em coordenadas polares relativas ao próprio jogador. Desta forma, ele possui informações de distância e direção para a bola, demais jogadores, bandeiras do campo e linhas do campo.

Com esses dados é possível estimar as coordenadas cartesianas absolutas no campo de todas as entidades de interesse. As bandeiras do campo são fixas e possuem coordenadas conhecidas, detalhado na subseção 3.2.1.2. Logo, a transformação da informação polar e relativa para uma cartesiana e absoluta da posição do próprio jogador é direta utilizando trigonometria básica.

A partir da informação de posição absoluta do próprio jogador é possível calcular as posições absolutas para o resto das entidades.

### 3.3 Definição de Ações

Os jogadores possuem diversas ações possíveis mapeadas pelo servidor da partida. As ações recebem parâmetros em sua maioria contínuos. Dessa forma a escolha de quais ações devem ser executadas se dá em um domínio discreto, porém cada ação exige uma escolha de parâmetros em um domínio contínuo.

#### 3.3.1 Arrancar

O comando de arrancar faz com que haja uma aceleração do jogador na direção da arrancada. O parâmetro “potência” determina o valor da aceleração e o parâmetro “direção” é relativo à direção do corpo do jogador. É importante salientar que o comando de arranque é o jeito padrão de movimentar um jogador.

Cada jogador possui uma certa quantidade de energia e o arranque tem um custo sobre ela. Ao começo de cada partida, a energia do jogador é colocada no máximo. Se o jogador acelera para frente, a energia é reduzida em  $1 \times$  a potência. Se o jogador acelera para trás, o custo é maior e a energia é reduzida em  $2 \times$  a potência.

Se a energia disponível é menor que a necessária para a realização com comando, o valor de “potência” é reduzido para que a quantidade necessária de energia seja a disponível.

#### 3.3.2 Chutar

O comando de chute recebe dois parâmetros: a força e a direção do chute. Para realizar o comando, a bola precisa ser “chutável”, ou seja, estar a uma certa distância do jogador.

Caso a bola não esteja diretamente à frente do jogador, a força efetiva será reduzida por um fator dependente da posição relativa da bola.

#### 3.3.3 Virar

O comando virar recebe como parâmetro o momento angular a ser aplicado pelo jogador sobre si mesmo. Se o jogador não estiver em movimento, o seu ângulo é apenas incrementado com o momento.

Porém, caso o jogador esteja em movimento, o resultado do comando é influenciado pelo momento de inércia do jogador (definido aleatoriamente pelo servidor no início da partida) e sua velocidade linear.

### 3.3.4 Virar pescoço

O jogador pode virar seu pescoço de maneira semi-independente de seu corpo. O ângulo do pescoço é relativo ao ângulo do corpo, então caso um comando virar seja executado, o ângulo absoluto do pescoço também mudará. Os ângulos máximo e mínimo em relação ao corpo são definidos na configuração do servidor, sendo o padrão +90 e -90 graus, respectivamente.

### 3.3.5 Mover-se

O comando “mover-se” é utilizado para mover diretamente jogadores para coordenadas. Não pode ser utilizado no decorrer de uma partida, exceto quando o goleiro tem a posse da bola. O comando fica disponível no início dos tempos da partida para posicionamento dos jogadores.

Para movimentar os jogadores durante a partida, o arranque (Subseção 3.3.1) deve ser utilizado.

### 3.3.6 Agarrar

O único jogador com permissão para agarrar ou pegar a bola é o goleiro. O goleiro pode pegar a bola em qualquer direção, desde que ela esteja dentro da área agarrável (Figura 3.2), definida como um retângulo com comprimento 2.0 e largura 1.0 na direção em que se deseja tentar agarrar.

Se o comando de agarrar falhar o goleiro fica incapacitado de agarrar a bola durante um número pré-determinados de ciclos do jogo. Se o comando tiver sucesso o goleiro pode usar o comando "mover-se"(Seção 3.3.5) para se mover segurando a bola até um limite máximo de vezes estabelecido nas configurações do servidor.

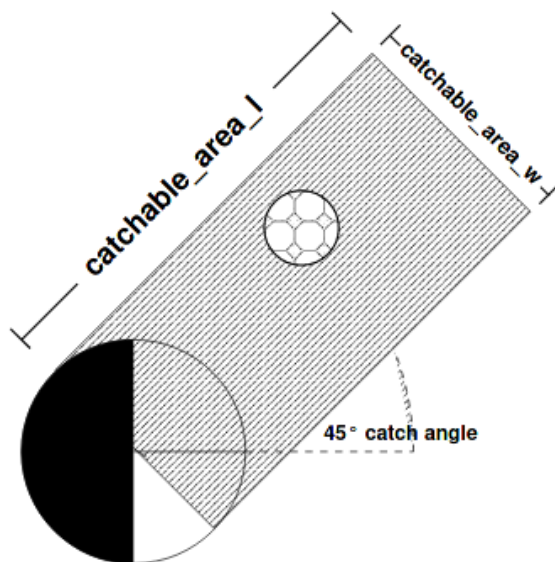


Figura 3.2: Visualização da área agarrável pelo goleiro [2].

### 3.3.7 Falar

Usado para transmitir mensagens aos outros jogadores. As mensagens devem ter comprimento menor que um valor pré-determinado pelo servidor. Os jogadores que estiverem a uma distância audível da mensagem receberão a mensagem do servidor imediatamente.

Não será implementada nenhuma comunicação entre os jogadores neste trabalho.

## 3.4 Ambiente de Treinamento

O ambiente de treinamento consiste em uma base de código que importa a biblioteca detalhada na Seção 3.1. Um formato geral foi definido e desenvolvido a fim de tornar os experimentos fáceis de adaptar, bastando mudar alguns trechos do código.

### 3.4.1 Laço de Treinamento

De forma geral o laço de treinamento obedece ao pseudo-código apresentado.

```
definir parâmetros
inicializar pesos de treinamento
enquanto o estado não é terminal:
  conectar jogador
  para cada ciclo da partida:
    escolher ação de acordo com a política e o estado
    observar novo estado e recompensa
    treinar pesos
  estado <- novo estado
```

Algoritmo 3.1: Pseudo-código que descreve a rotina de treinamento

O laço interno é onde efetivamente os algoritmos de treinamento são implementados, portanto este trecho é alterado a depender da técnica de aprendizagem por reforço utilizada.

## 3.5 Experimentos

O objetivo inicial para teste do sistema como um todo foi o de realizar o treinamento de um agente único capaz de executar gols estando sozinho em campo.

Para isso foram realizadas três abordagens diferentes.

- Sarsa com método aproximado e comportamentos pré-programados
- Double Q-Learning tabular e comportamentos pré-programados
- Double Q-Learning tabular e ações puras

### 3.5.1 Sarsa Aproximado e Comportamentos Pré-Programados

Inicialmente, desejou-se realizar um treinamento utilizando um método aproximado, devido ao grande número de estados contínuos possíveis para o agente. Portanto, foi usada uma rede neural multicamada com estrutura detalhada na Tabela 3.1.

camada	nº neurônios
1	5
2	256
3	256
4	128
5	64
6	8

Tabela 3.1: Estrutura da rede neural utilizada no experimento

A utilização de comportamentos permite que sejam inseridos conhecimentos prévios a respeito do que é esperado de um agente jogador de futebol, ou seja, é possível simplificar o aprendizado substituindo as ações puras como chutar ou correr por ações abstratas como perseguir a bola e chutar para o gol.

#### 3.5.1.1 Codificação dos Estados

- **Coordenada X do Jogador:** dentro do intervalo  $[-1, 1]$  e normalizada em relação ao comprimento do campo.
- **Coordenada Y do Jogador:** dentro do intervalo  $[-1, 1]$  e normalizada em relação à largura do campo.
- **Orientação do Jogador:** dentro do intervalo  $[-1, 1]$  e proporcional ao ângulo do corpo do jogador em relação ao eixo X entre  $-180^\circ$  e  $180^\circ$ .
- **Distância até a bola:** normalizada em relação à distância máxima visível.
- **Direção da bola:** dentro do intervalo  $[-1, 1]$  e proporcional ao ângulo do corpo do jogador em relação ao vetor que o liga à bola entre  $-180^\circ$  e  $180^\circ$ .

#### 3.5.1.2 Codificação dos Comportamentos

Os comportamentos pré-programados foram desenvolvidos de forma a cobrir ações que permitissem a realização de gols, que é o objetivo do treinamento.

- **Comportamento nulo:** O agente apenas espera até o próximo ciclo.

- **Localizar bola:** Caso o agente não veja a bola, ele gira no seu próprio eixo na direção em que a bola foi vista pela última vez. Se a bola não foi vista por mais de 30 ciclos, o agente gira em seu eixo  $45^\circ$ , a fim de achar a bola mais rápido.
- **Se aproximar da bola:** Caso o agente veja a bola, ele se move em direção a ela com velocidade constante
- **Se afastar da bola:** Caso o agente veja a bola, ele se move em direção oposta a ela com velocidade constante
- **Carregar bola para o gol direito:** Caso a distância até a bola seja menor ou igual a 0.7 metros, o agente chuta levemente a bola e anda de forma a conduzi-la ao gol direito.
- **Chutar bola para gol direito:** Caso a distância até a bola seja menor ou igual a 0.7 metros, o agente chuta a bola na direção do gol direito.
- **Carregar bola para o gol esquerdo:** Caso a distância até a bola seja menor ou igual a 0.7 metros, o agente chuta levemente a bola e anda de forma a conduzi-la ao gol esquerdo.
- **Chutar bola para gol esquerdo:** Caso a distância até a bola seja menor ou igual a 0.7 metros, o agente chuta a bola na direção do gol esquerdo.

A seguir, são descritos os algoritmos dos comportamentos de forma geral. Com poucas adaptações chega-se aos comportamentos específicos para direita ou esquerda do campo.

```
função AproximarBola(s):
  se s.BolaVisível:
    ângulo da bola <- s.DireçãoBola + s.DireçãoPescoço
    comando <- Arrancar(60, ângulo da bola)
  retorna comando
```

Algoritmo 3.2: Algoritmo do comportamento de se aproximar da bola

```
função CarregarBola(s):
  se s.DistânciaBola < 0.7:
    ângulo <- -s.Orientação
    comando <- Chutar(5, ângulo)
  retorna comando
```

Algoritmo 3.3: Algoritmo geral do comportamento de carregar bola

```
função CarregarBola(s):
  Xgol, Ygol <- Gol.çãPosio
  se s.DistânciaBola < 0.7:
    ângulo <- atan2(s.Y - Ygol, s.X - Xgol) - s.Orientação
    comando <- Chutar(60, ângulo)
  retorna comando
```

Algoritmo 3.4: Algoritmo geral do comportamento de chutar bola para o gol



### 3.5.1.3 Parâmetros

- **Fator de desconto ( $\gamma$ ):** Apesar do ambiente ser episódico, foi utilizado um fator de desconto de 0.99 devido ao fato de que a condição de término do episódio (fim de jogo) não ser observável através da discretização do estado utilizada.
- **Fator de aprendizagem ( $\alpha$ ):** O fator de aprendizagem foi definido inicialmente como 0.1 e foi reduzido exponencialmente multiplicando-o por 0.99999 ao final de cada partida.
- **Fator de exploração ( $\epsilon$ ):** Para incentivar a exploração das possibilidades, o fator de exploração foi definido inicialmente como 0.9 e reduzido exponencialmente multiplicando-o por 0.99996 ao final de cada partida. A cada ação tomada, o agente tem probabilidade  $\epsilon$  de escolher uma ação aleatória. Além disso, para favorecer a exploração, no início de cada partida era também sorteada uma posição inicial para o agente em seu lado do campo.

### 3.5.1.4 Recompensa

A recompensa foi definida em 3 partes a fim de guiar o agente na direção do aprendizado desejado. Todas as medições de recompensa foram feitas utilizando dados da simulação e não da percepção do agente.

- **Proximidade da bola  $R_1$ :** Para que o agente tenha tendência a se aproximar da bola, foi definida uma recompensa negativa proporcional à distância  $d$  do agente em relação à bola, ou seja:  $R_1 = -d * 0.001/6000$ .
- **Velocidade da bola  $R_2$ :** Para que o agente adquira o comportamento de chutar a bola em direção ao gol adversário, foi definida uma recompensa positiva proporcional à velocidade instantânea da bola em X ( $v_x$ ), ou seja:  $R_2 = v_x/6000$ .
- **Gol  $R_3$ :** Por fim, para incentivar que o agente fizesse gols, foi definida uma recompensa esparsa de valor 1 para cada gol realizado e -1 para gols contra, ou seja:

$$R_3 = \begin{cases} 1 & \text{se foi feito um gol no ciclo} \\ -1 & \text{se foi feito um gol contra no ciclo} \\ 0 & \text{caso não houver gol no ciclo} \end{cases}$$

Deste modo, a cada instante de tempo foi atribuída uma recompensa  $R = R_1 + R_2 + R_3$ .

## 3.5.2 Double Q-Learning Tabular e Comportamentos Pré-Programados

Além da solução através de um aproximador de funções, foi testada uma solução utilizando o método tabular com os mesmos comportamentos e uma discretização dos estados descrita abaixo.

A codificação dos comportamentos, parâmetros  $\alpha$ ,  $\epsilon$  e  $\gamma$  e a recompensa foram mantidos como descrito nas subseções 3.5.1.2, 3.5.1.3 e 3.5.1.4 respectivamente.

### 3.5.2.1 Codificação dos Estados

O estado percebido pelo agente é dado pela combinação dos seguintes fatores:

- **Distância até a bola.** A distância  $D$  até a bola foi discretizada de acordo com a seguinte função:

$$\begin{cases} 0 & \text{se } D < 0.7 \\ \lfloor \log_2(\frac{D}{0.7}) \rfloor & \text{se } 0.7 \leq D \text{ e } D < 0.7 \times 2^6 \\ 6 & \text{se } D \geq 0.7 \times 2^6 \end{cases} \quad (3.1)$$

Ou seja, a distância percebida até a bola varia entre 0 e 6 com resolução cada vez menor à medida que o agente se afasta da bola. O fator 0.7 foi inserido na função devido ao fato de que esta é a distância mínima que permite que o agente chute a bola.

Caso o jogador possa enxergar a bola, a distância  $D$  é recebida diretamente do sensor. Caso contrário, a distância  $D$  é estimada com base na última posição percebida da bola.

- **Direção da bola:** A direção da bola foi dividida em 24 fatias de  $15^\circ$  cada. O ângulo de visão do jogador é de  $\pm 30^\circ$ . Caso a bola não esteja visível, a direção da bola é estimada com base na última posição percebida.
- **Posição do jogador em X:** A posição estimada do jogador em X foi discretizada em 10 janelas de tamanho 11.5.
- **Posição do jogador em Y:** A posição estimada do jogador em Y foi discretizada em 7 janelas de tamanho aproximado 11.14.
- **Direção do jogador:** A direção estimada do jogador em relação ao eixo horizontal foi discretizada em 24 fatias de  $15^\circ$  cada.

Com isso, temos que o número total de estados possíveis é dado pelo produtório da quantidade de possibilidades em cada um dos itens acima totalizando 282240 estados.

### 3.5.3 Double Q-Learning Tabular e Ações Puras

Após o resultado positivo obtido utilizando comportamentos pré-programados, foi testada uma abordagem *end-to-end* utilizando a discretização de estados descrita na subseção 3.5.2.1 e a recompensa descrita na subseção 3.5.1.4.

#### 3.5.3.1 Codificação das Ações

Para simplificar o vasto espaço de ações disponíveis, foi selecionado um conjunto discreto de 13 ações:

- **Ação nula:** O agente apenas espera até o próximo ciclo.

- **Virar-se:** O agente tem a opção de virar-se  $7^\circ$ ,  $15^\circ$  ou  $31^\circ$  para ambas as direções, totalizando 6 ações de rotação possíveis.
- **Correr:** É possível correr em frente ( $0^\circ$ ) ou a  $30^\circ$  em ambas as direções, sempre com potência 50, totalizando 3 ações de corrida possíveis.
- **Chutar:** Caso a distância até a bola seja menor ou igual a 0.7 metros, o jogador tem a opção de chutá-la em frente ou em um ângulo de  $45^\circ$  em ambas as direções, totalizando 3 ações de chute possíveis. Caso a bola não esteja próxima o suficiente, nada acontece.

### 3.5.3.2 Parâmetros

- **Fator de desconto ( $\gamma$ ):** Foi mantido tal como descrito na subseção 3.5.1.3.
- **Fator de aprendizagem ( $\alpha$ ):** O fator de aprendizagem foi definido como  $\frac{6000}{6000+N}$  sendo  $N$  o número de partidas experienciadas pelo jogador.
- **Fator de exploração ( $\epsilon$ ):** O fator de aprendizagem foi definido como  $\frac{8000}{8000+N}$  sendo  $N$  o número de partidas experienciadas pelo jogador.

## Capítulo 4

# Resultados Experimentais

### 4.1 Agente Único com Sarsa Aproximado e Comportamentos Pré-Programados

O treinamento de Sarsa realizado contou com XXXXX partidas utilizando a rede neural multicamadas descrita na Subseção 3.5.1 e salvando os retornos obtidos em cada partida.

O gráfico da Figura 4.1 mostra em conjunto o retorno a cada partida e a média do retorno a cada 100 partidas. É possível observar que há uma tendência de subida do retorno com o passar dos episódios experienciados, porém em grande parte das partidas o agente não realizou nenhum gol, refletido pelo baixo valor da média de 100 partidas.

É interessante ressaltar o alto custo de processamento deste tipo de treinamento, por utilizar redes neurais. Dessa forma, a quantidade de amostras possíveis de serem coletadas em tempo hábil foram dramaticamente reduzidas.

### 4.2 Agente Único com Double Q-Learning Tabular e Comportamentos Pré-Programados

Foi executado um treinamento de 100000 partidas e foram salvos a tabela Q completa e o histórico dos retornos obtidos pelo agente ao longo do treinamento.

O gráfico da Figura 4.2 a seguir mostra esse histórico. Observa-se que o desempenho dessa abordagem supera o da abordagem anterior rapidamente, com poucas amostras. Em contrapartida, há uma estagnação do retorno por volta das 70000 amostras, o que indica a existência de um limite superior para o desempenho do agente devido à menor flexibilidade da política aprendida.

A Figura 4.3 ilustra o agente conduzindo a bola e realizando um gol conforme a política aprendida.



Figura 4.1: Curva de aprendizado do agente com comportamentos pré-programados utilizando Sarsa aproximado.

### 4.3 Agente Único com Double Q-Learning Tabular e Ações Puras

Foram executados 3 treinamentos distintos de 100000 partidas a fim de suavizar o elemento sorte nos resultados. Após cada um dos treinamentos foram salvos a tabela Q completa e o histórico dos retornos obtidos pelo agente ao longo do treinamento.

A Figura 4.4 mostra esse histórico. É interessante observar que com o decaimento dos fatores de exploração e de aprendizagem, após 100000 partidas ambos eram  $\epsilon \approx 0.01648$  e  $\alpha \approx 0.03679$ , ou seja, o agente já executava na maior parte dos ciclos a política aprendida. Para cada jogo foi feita a média entre os 3 retornos observados em cada um dos treinamentos.

Além disso, foi executado um treinamento de 200000 partidas com os mesmos parâmetros, a fim de observar a aprendizagem por um período mais longo. Na Figura 4.5, observa-se a cessação de aprendizado com o decaimento do fator de exploração.

Apesar da grande quantidade de experiência a que o agente teve acesso, nota-se na Figura 4.5 que o crescimento de seu desempenho é bastante limitado, sequer atingindo a média de 1 gol por partida. Isso é um indicativo do altíssimo custo computacional de soluções *end-to-end* como a utilizada no experimento.

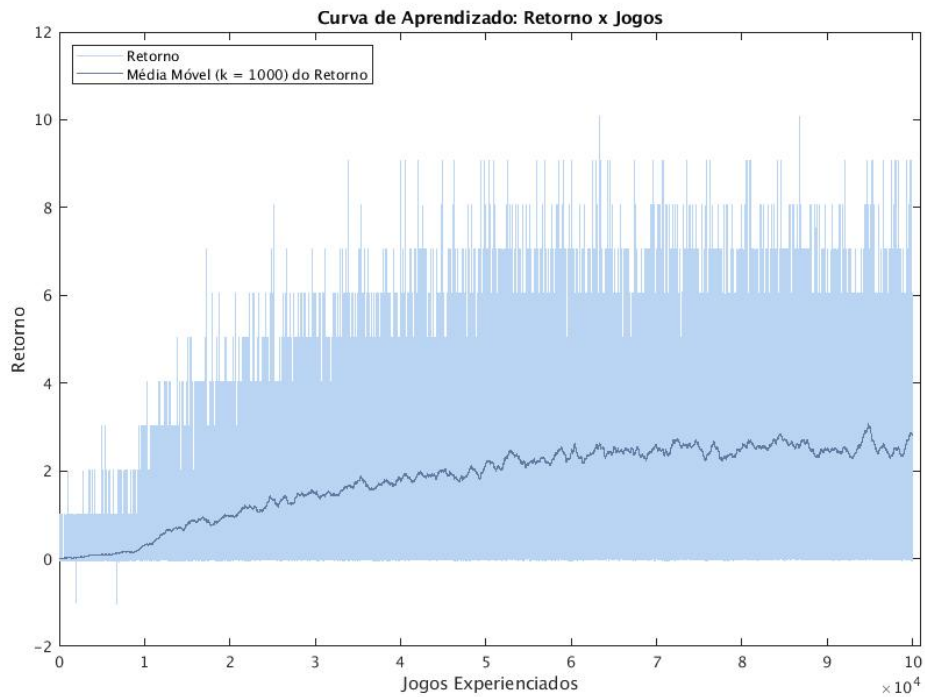


Figura 4.2: Curva de aprendizado do agente com comportamentos pré-programados.

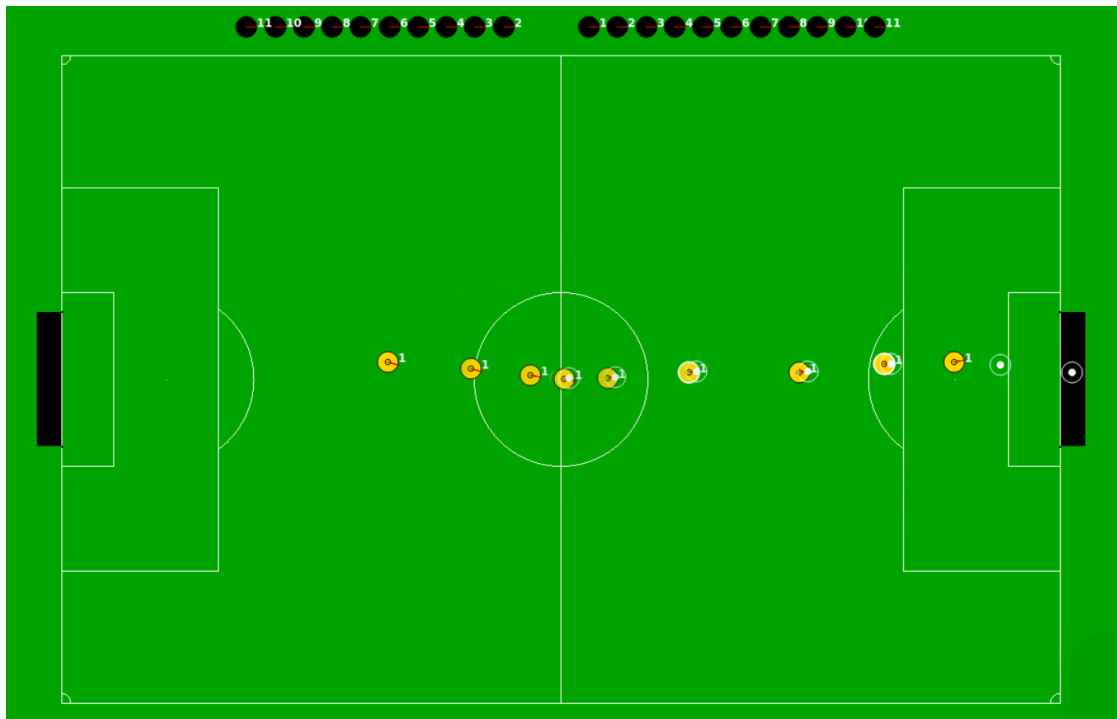


Figura 4.3: Sobreposição de sequência de imagens do agente fazendo gol

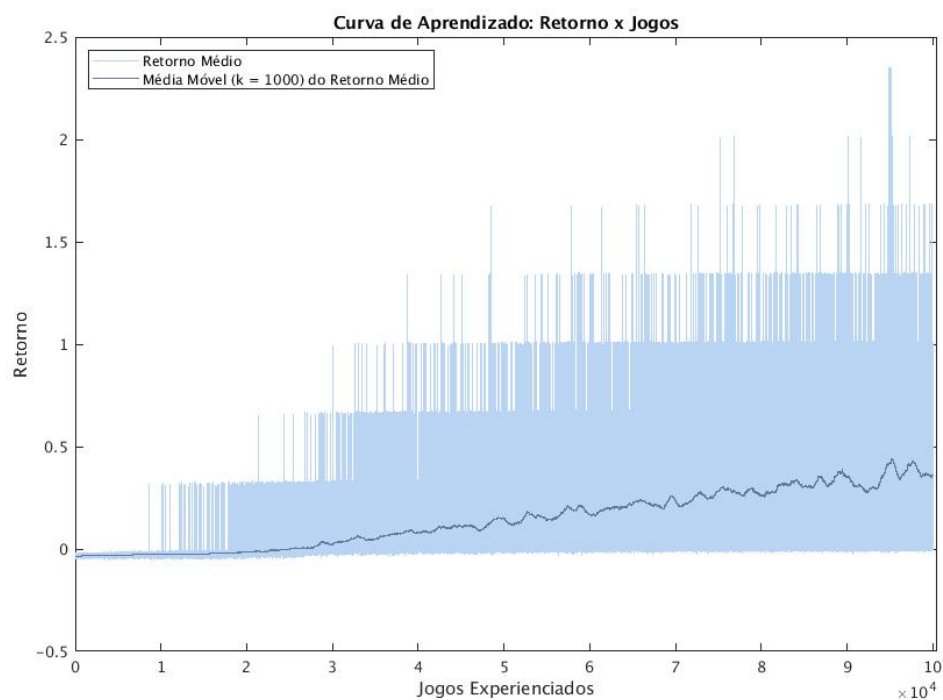


Figura 4.4: Curva de aprendizado do agente com ações puras.

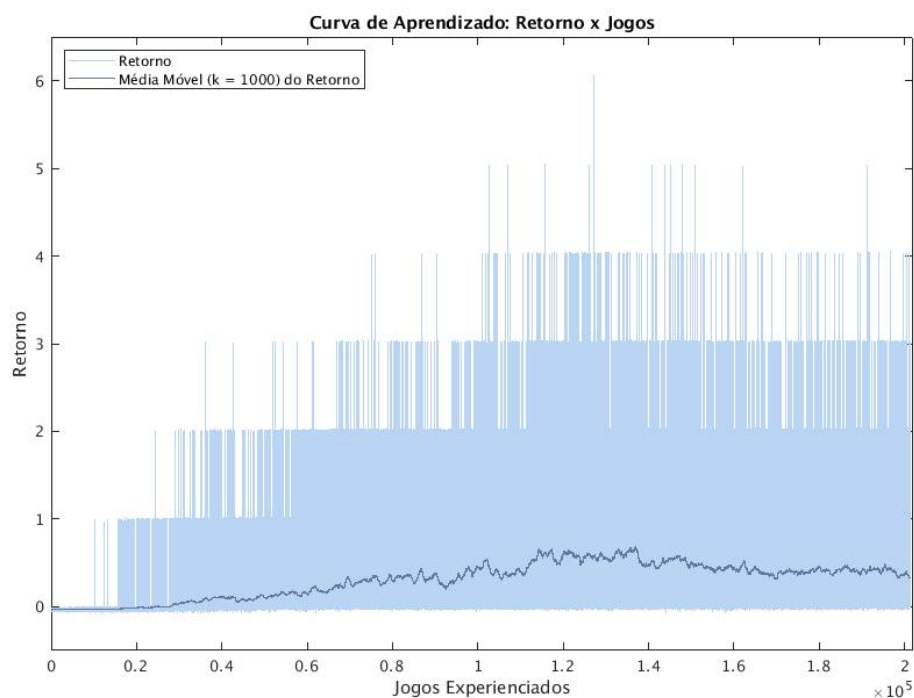


Figura 4.5: Curva de aprendizado para treinamento longo.

## Capítulo 5

# Conclusões

### 5.1 Implementação da Interface com o Servidor

Ao pesquisar sobre a comunidade e equipes participantes das edições nacionais da competição, notou-se que a biblioteca de interfaceamento com o servidor *librcsc* e o time base *agent2d* - ambos desenvolvidos no Japão por acadêmicos relacionados à equipe *HELIOS* - são amplamente utilizados. Entretanto, a documentação da biblioteca é escassa e há dificuldade de utilização dela, evidenciado por conversas com os participantes da comunidade.

Esse cenário demonstra a necessidade de modernização da base de código utilizada pelas equipes. É proposto, então, a reimplementação da interface de comunicação com o servidor da partida utilizando a linguagem Go.

### 5.2 Treinamento de Equipe para Participação em Competições

Este projeto propõe o treinamento de um time capaz de competir contra as principais equipes nacionais e internacionais da categoria. Serão estudados, avaliados e implementados métodos de inteligência computacional para o treinamento do time a fim de obter comportamentos adequados para os jogadores de modo que eles consigam atuar colaborativamente para vencer o time adversário.

A participação em competições proporciona um ambiente perfeito para validação do sistema, uma vez que é possível comparar qualitativa e quantitativamente seu desempenho contra as diversas equipes do país.



# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. [S.l.]: MIT press, 2018.
- [2] CHEN, M. et al. Users manual - robocup soccer server (for soccer server version 7.07 and later). 2003.
- [3] Guarnieri, M. The roots of automation before mechatronics [historical]. *IEEE Industrial Electronics Magazine*, v. 4, n. 2, p. 42–43, 2010.
- [4] WIENER, N. *he human use of human beings: Cybernetics and society*. Garden City, New York: Doubleday. [S.l.]: APA, 1950.
- [5] MCCORDUCK, P. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. [S.l.]: AK Peters Ltd, 2004. ISBN 1568812051.
- [6] SICILIANO, B.; KHATIB, O. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN 354023957X.
- [7] SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, v. 529, 2016. ISSN 0028-0836.
- [8] OPENAI. *OpenAI Five*. 2018. <https://blog.openai.com/openai-five/>.
- [9] MACKWORTH, A. K. On seeing robots. In: *Computer Vision: Systems, Theory and Applications*. [S.l.]: World Scientific, 1993. p. 1–13.
- [10] KITANO, H. et al. Robocup: The robot world cup initiative. In: *Proceedings of the first international conference on Autonomous agents*. [S.l.: s.n.], 1997. p. 340–347.
- [11] BAI, A.; WU, F.; CHEN, X. Online planning for large markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM, v. 6, n. 4, p. 45, 2015.
- [12] NAKASHIMA, T. et al. Helios2018: Team description paper. In: *RoboCup 2018 Symposium and Competitions: Team Description Papers, Montreal, Canada*. [S.l.: s.n.], 2018.
- [13] MELLO, F. et al. Itandroids 2d soccer simulation team description 2012.
- [14] MAXIMO, M. R. et al. Itandroids 2d soccer simulation team description 2019.