TRABALHO DE GRADUAÇÃO

INTELIGÊNCIA COMPUTACIONAL PARA FUTEBOL DE ROBÔS MULTI-AGENTE

Bruno Andreghetti Dantas

Samuel Venzi Lima Monteiro de Oliveira

Brasília, dezembro de 2019

SUMÁRIO

Capítulo 1

Introdução

1.1 RoboCup Soccer Simulation 2D

A ideia de robôs jogando futebol foi proposta pela primeira vez em 1992 por Alan Mackworth[1]. Desde então a comunidade científica tem criado iniciativas buscando por soluções que tornem isso realidade. Uma delas é a *Robot World Cup Initiative*, 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, entre elas a *RoboCup Soccer Simulation 2D*, abreviada RCSS, objeto de estudo deste projeto.

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.

1.1.1 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 comunição 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

1.1.2 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.

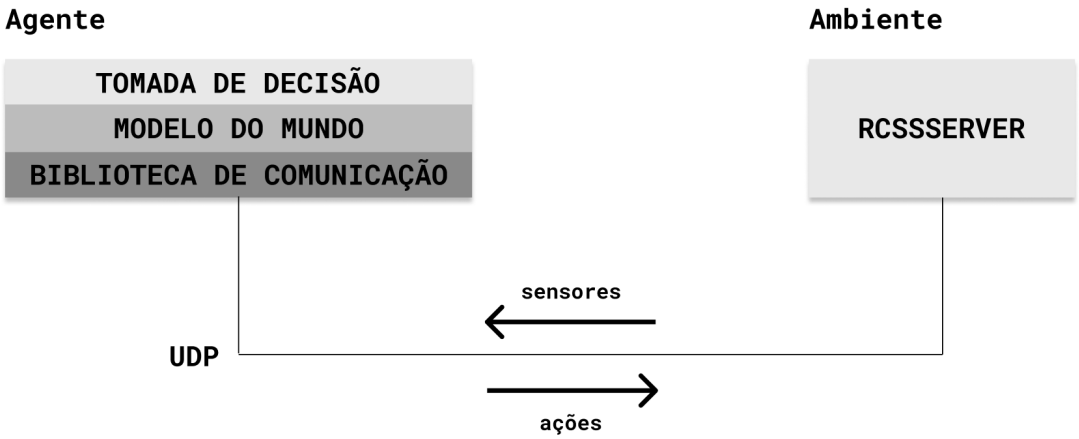

Figura 1.2: Esquema ilustrando a arquitetura de um cliente e sua comunicação com o servidor do jogo.

1.1.3 Sensores

Para receber informações a respeito da situação do jogo, cada cliente conectado recebe mensagens referentes a atualizações de três sensores virtuais descritos abaixo.

1.1.3.1 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 jogadores.

1.1.3.2 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 1.3) 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.

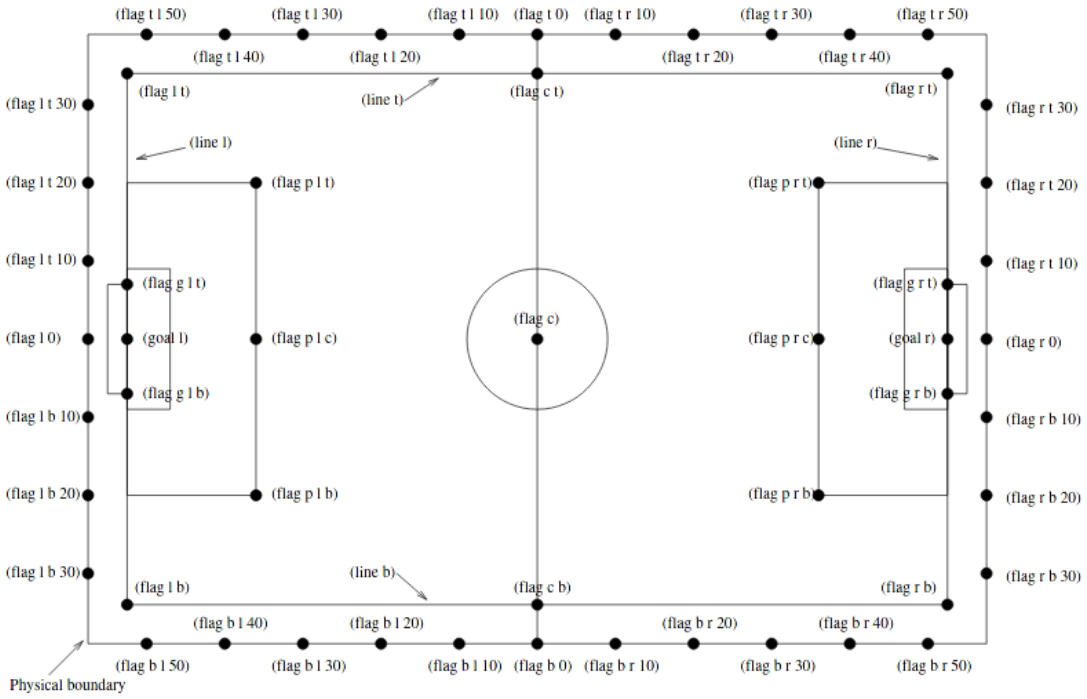

Figura 1.3: Indicadores espalhados pelo campo para que o agente possa estimar sua posição absoluta [2].

A riqueza de detalhes a respeito das informações obtidas depende da distância entre o objeto e o jogador. Por exemplo, caso esteja sendo visto outro jogador a uma distância muito grande, talvez seja 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.

1.1.3.3 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 (Seção 1.1.4), sua própria velocidade e direção de movimento, a direção de sua cabeça e a quantidade de cartões de advertência recebidos.

1.1.4 Ações

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

1.1.4.1 Arrancar

O comando de arrancar faz com que haja uma aceleração do jogador na direção da arrancada. Um parâmetro "potência"determina o valor da aceleração. É importante notar 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 tempo da partida, a energia do jogador é colocada no máximo. Se o jogador acelera para frente, a energia é reduzida pelo valor de "potência". Se o jogador acelera para trás, o custo é maior e a energia é reduzida pelo dobro da 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.

1.1.4.2 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.

1.1.4.3 Virar

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

1.1.4.4 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.

1.1.4.5 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 o jogadores durante a partida, o arranque Seção 1.1.4.1) deve ser utilizado.

1.1.4.6 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 1.4), 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 1.1.4.5) para se mover segurando a bola até um limite máximo de vezes estabelecido nas configurações do servidor.

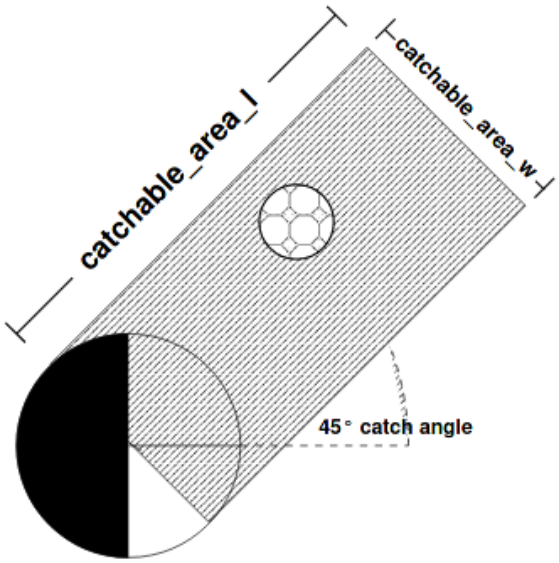

Figura 1.4: Visualização da área agarrável pelo goleiro [2].

1.1.4.7 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 à distância audível da mensagem receberão a mensagem do servidor imediatamente.

Como já dito, não será implementada nenhuma comunicação entre os jogadores neste trabalho.

1.2 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[3].

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[4].

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 [5] para adequar os parâmetros que calculam o valor da ação. A *ITAndroids* também vem desenvolvendo o uso de Aprendizagem por Reforço Profunda [6].

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.

Capítulo 2

Fundamentação Teórica

2.1 Processos de Decisão de Markov

O problema abordado neste trabalho pode ser descrito como um Processo de Decisão de Markov (MDP). MDP é uma forma clássica de representação matemática de processos de decisão sequenciais. Nessa representação, cada ação tomada por um agente que interage com o ambiente transforma o estado do processo e determina a recompensa que o agente recebe imediatamente. Esse estado também deve ser suficiente para conter toda a informação relevante para a dinâmica futura do processo.

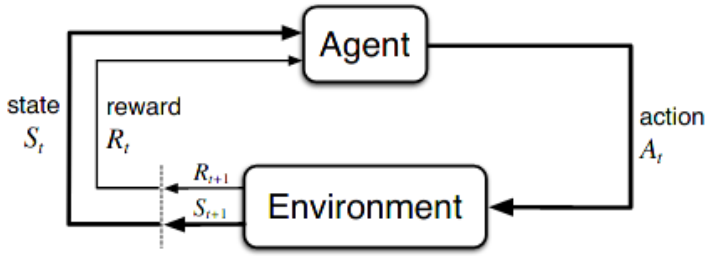

Figura 2.1: Interação agente-ambiente em um MDP [7].

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 imediata $R \in \mathcal{R}$ [7].

Essa abordagem é bastante flexível e torna possível a modelagem da 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 ação tomar diante desse estado 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. 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.

Nesta fundamentação, será tratado com mais atenção o caso episódico uma vez que é o caso que interessa para aplicação na simulação de futebol de robôs.

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 ao fim de um episódio, ou seja, é esperado que se escolha A_t a fim de maximizar não apenas R_{t+1} mas sim o retorno $G_{t+1} = R_{t+1} + R_{t+2} + \ldots + R_{terminal}$.

2.1.3 Políticas

É dado o nome de política para qualquer função $\pi(S) \rightarrow \mathcal{A}$ que leve de um estado qualquer do MDP para uma ação a ser tomada. Para cada política π, existe uma função $q_\pi(S, A)$ que, para cada par de estado e ação, define a esperança de retorno caso o agente continue seguindo a política π no restante do episódio.

É possível comparar duas políticas π e π' a respeito de suas funções q. A política π é considerada melhor ou igual a π, 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 π_* e, caso haja mais de uma, todas devem possuir a mesma função q denominada q_* [7].

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 ϵ é denominada ϵ-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, e com isso determinar um critério de decisão - denominado 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) dados como corretos. 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 corretos 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 [7].

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 aprendizado e a política sobre a qual se quer aprender.

Nos algoritmos de aprendizagem on-policy, o agente aprende a respeito da política π enquanto navega o MDP de acordo com a própria política π.

Já nos algoritmos de aprendizagem off-policy, o agente aprende a respeito da política alvo π enquanto navega o MDP de acordo com a política b, ou seja, ele estima a função q_π enquanto segue a política b.

Os métodos off-policy costumam introduzir variância no processo, tornando o aprendizado ruidoso e em alguns casos a garantia de convergência é provada apenas para o caso on-policy.

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 poda 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 infinito, 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.

Neste trabalho serão utilizados métodos de solução aproximada devido à grande quantidade de informações simultâneas às quais o jogador tem acesso.

2.2.3 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 $Q : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ da função q_*, para cada instante de tempo t é realizada a seguinte atualização a fim de aproximar Q de q_*:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.1}$$

Sendo α o fator de aprendizagem, responsável por diluir as atualizações de cada entrada da tabela, e γ 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 alguns casos a convergência é provada matematicamente.

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

$$A_{t+1} = \max_a q_*(S_t, a) \tag{2.2}$$

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

2.2.4 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:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t)] \tag{2.3}$$

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 aproximaçoes, ele não aumenta o custo computacional por iteração.

Capítulo 3

Desenvolvimento

3.1 Biblioteca

O servidor da partida apresenta, como já mencionado, um protocolo de comunicação e sintaxe de mensagens específica. Uma biblioteca de interfaceamento foi desenvolvida com o objetivo de abstrair os detalhes de comunicação 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*.

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 novas versões do servidor.

3.1.1 Arquitetura

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 de Codificação de Mensagens

A decodificação de mensagens, por sua vez, foi feita em duas camadas: um *lexer* e um analisador. O lexer passa pelas mensagens em formato string e retira todas as informações que ela

contém. O analisador estrutura essas informações 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 *lexer*.

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

Mensagem após passar pelo analisador.

3.2 Experimentos

3.2.1 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.

Além disso, para os dados disponíveis por meio da biblioteca foram estimados outros estados que pudessem ser úteis no treinamento: posições absolutas da bola, dos outros jogadores e do próprio jogador.

3.2.1.1 Laço de Treinamento

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

```
definir parametros
inicializar pesos de treinamento
laco para cada partida:
  conectar jogador
  laco para cada passo da partida:
    ver estado
    escolher acao de acordo com uma politica
    observar novo estado e recompensa
    treinar pesos
    estado <- novo estado
    se estado for terminal:
      quebra
```

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.

3.2.1.2 Estimação 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, entretanto, 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. Portanto, 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 e de sua direção é possível calcular as posições para o resto das entidades.

3.2.2 Agente Único com Double Q-Learning Tabular

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. Esse objetivo se provou mais difícil do que o esperado para a abordagem *end-to-end* desejada.

Foi utilizado o algoritmo Double Q-Learning tabular, possível através da discretização de diversas métricas obtidas através dos sensores do agente.

3.2.2.1 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:

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

  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° cada. O ângulo de visão do jogador é de ±30°. 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° 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.

3.2.2.2 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°, 15° ou 31° para ambas as direções, totalizando 6 ações de rotação possíveis.

- **Correr**: É possível correr em frente (0°) ou a 30° 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° em ambas as direções, totalizando 3 ações de chute possíveis. Caso a bola não esteja próxima o suficiente, nada acontece.

3.2.2.3 Parâmetros

- **Fator de desconto (γ)**: 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 (α)**: 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 (ϵ)**: 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 ϵ 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.

3.2.2.4 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 $R = R_1 + R_2 + R_3$.

3.2.2.5 Procedimentos e Resultados

Foram executados 2 treinamentos distintos de 100000 partidas a fim de suavizar o elemento sorte nos resultados. Após cada um dos treinamentos foi salva a tabela Q completa e o histórico dos retornos obtidos pelo agente ao longo do treinamento.

Os gráficos a seguir mostram 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 ótima aprendida.

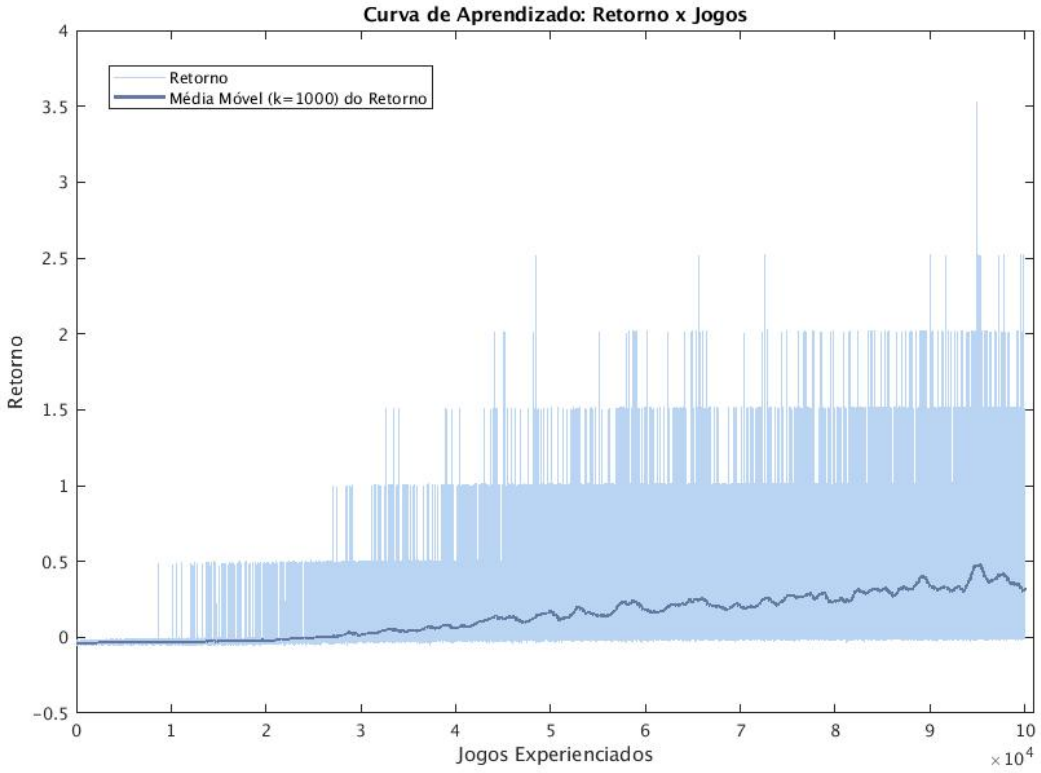

Figura 3.1: Curva de aprendizado. Para cada jogo foi feita a média entre os 2 retornos observados em cada um dos treinamentos.

3.3 Agentes Concorrentes

Após validação do sistema com agente único, é interessante experimentar com treinamento adversarial de apenas 2 jogadores em formato um-contra-um. A intenção dessa etapa é experimentar com o sistema o caso adversarial, no qual há um ou mais agentes com objetivo oposto ao do agente sendo treinado.

3.4 Múltiplos Agentes

Após validar os casos de agente único e de agentes concorrentes, propõe-se um treinamento completo em jogos 11 contra 11. O objetivo é, ao final do processo, termos um time capaz de jogar contra os principais times da atualidade na categoria RoboCup Soccer Simulation 2D.

Para isso, os agentes devem ser capazes de cooperar e reagir aos movimentos da equipe oposta a fim de marcar gols e evitar os gols do adversário.

Capítulo 4

Conclusões

4.1 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.

4.2 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] MACKWORTH, A. K. On seeing robots. In: *Computer Vision: Systems, Theory and Applications*. [S.l.]: World Scientific, 1993. p. 1–13.

[2] CHEN, M. et al. Users manual - robocup soccer server (for soccer server version 7.07 and later). 2003.

[3] 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.

[4] NAKASHIMA, T. et al. Helios2018: Team description paper. In: *RoboCup 2018 Symposium and Competitions: Team Description Papers, Montreal, Canada*. [S.l.: s.n.], 2018.

[5] MELLO, F. et al. Itandroids 2d soccer simulation team description 2012.

[6] MAXIMO, M. R. et al. Itandroids 2d soccer simulation team description 2019.

[7] SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. [S.l.]: MIT press, 2018.