FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FCPortugal - Path Planning for Simulated Soccer using Reinforcement Learning

Edgar dos Santos de Matos

FOR JURY EVALUATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor Luís Paulo Gonçalves dos Reis

25 de Fevereiro de 2022

Resumo

Esta dissertação foi desenvolvida dentro do contexto da equipa FCPortugal que participa na liga de simulação 3D da RoboCup. O objetivo deste trabalho passa por projetar e implementar um método de planeamento de caminho para controlar robôs humanóides num ambiente simulado de futebol. Para a implementação do mesmo serão usadas técnicas de *Reinforcement Learning* para escolher e otimizar o percurso do robô até à posição final desejada sendo o algoritmo usado o PPO.

Neste documento é apresentado o estado de arte de *Reinforcement Learning* assim como vários algoritmos que ate hoje foram usados.

É também descrito o ambiente geral da RoboCup para a liga de simulação 3D assim como os seus componentes e o robô atualmente usado na competição. É abordado também algum do papel que *Reinforcement Learning* teve nesta competição com foco na equipa FCPortugal.

De modo a resolver o problema desta dissertação são usadas as ferramentas FCPgym e Stable Baselines sendo que neste documento existe também uma breve descrição dos mesmos. Com a ajuda destas ferramentas é resolvido o problema de planeamento de caminho para algumas situações sendo elas divididas em duas: com ou sem obstáculo entre o agente e a posição final havendo por fim uma comparação dos resultados dos problemas.

Abstract

This dissertation was developed within the context of the FCPortugal team that participates in the RoboCup Simulation 3D League. The aim of this work is to design and implement a path planning method to control humanoid robots in a simulated football environment. For its implementation, *Reinforcement Learning* techniques will be used to choose and optimize the robot's path to the desired final position, the PPO is the chosen algorithm to this thesis.

This document presents the state of the art of *Reinforcement Learning* as well as several algorithms that have been used until today.

It also describes RoboCup's general environment for the Simulation 3D League as well as its components and the robot currently used in the competition. Some of the role that *Reinforcement Learning* played in this competition focusing on the FCPortugal team is also addressed.

In order to solve the problem of this dissertation, the FCPgym and Stable Baselines tools are used, and in this document there is also a brief description of them. With the help of these tools, the path planning problem for some situations is solved and they are divided into two: with or without an obstacle between the agent and the final position, finally having a comparison of the results of the problems.

Agradecimentos

Inicialmente gostava de agradecer aos membros da equipa FCPorgugal por toda a ajuda que me foi concedida durante estes meses. Quero também agradecer ao meu orientador, Luís Paulo Reis por estar sempre disponível e estar disposto a reunir sempre que precisei de esclarecer alguma dúvida. Por fim e não menos importante gostava de agradecer à minha família e aos meus amigos, pois sem o suporte deles não teria conseguido atingir este objetivo.

Edgar dos Santos de Matos

"Those who dream by day are cognizant of many things
which escape those who dream only by night."

Edgar Allan Poe

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

RL	Reinforcement Learning
TPRO	Trust Region Policy Optimization
PPO	Proximal policy optimization
TD3	Twin Delayed Deep Deterministic policy gra-dient
SAC	Soft actor-critic
CoM	Center of Mass
ZMP	Zero Momento Point
IPM	Inverted Pendulum Mode

Chapter 1

Introdução

1.1 Enquadramento

FCPortugal é um projeto conjunto entre a Universidade do Porto e a Universidade de Aveiro para a participação na *RoboCup World-Championship*. *RoboCup* é uma competição global de robótica organizada anualmente, onde as equipas podem competir umas contra as outras em várias ligas. Uma destas ligas baseia-se na simulação de jogos de futebol em 3D, sendo cada equipa constituída por 11 robôs humanóides que se defrontam em campo num ambiente simulado. A intenção da *RoboCup*[15] é ser usada como um meio para promover a robótica e a pesquisa de inteligência artificial, oferecendo um desafio publicamente atraente e formidável. O grande objetivo do projeto *RoboCup* é:

> *"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.[15]"*

1.2 Objetivos

O objetivo principal desta dissertação é projetar, implementar e testar um método de planeamento de caminho para controlar robôs humanóides num ambiente simulado de futebol. Cada agente deve ser capaz de chegar ao seu destino da maneira mais eficiente, evitando objetos estáticos (como os postes das balizas) e objetos dinâmicos (bola, membros da própria equipa ou da equipa adversaria). O mecanismo de planeamento do caminho deve levar em consideração a velocidade e o raio de girar do comportamento atual (andar ou correr), bem como a orientação final desejada e para tal efeito será usado *reinforcement learning*. Os objetivos passam por desenvolver um método de planeamento de caminho através *reinforcement learning* e que seja capaz de controlar os jogadores de forma eficiente, uma forma flexível de adaptar o comportamento às diferentes condições e métricas de avaliação de desempenho para diferentes cenários e diferentes abordagens.

1.3 Motivação

Inicialmente a liga de futebol simulado em 3D passava pelo comportamento mais básico dos robôs humanóides como correr e chutar, mas como é de esperar a cada ano é exigido um maior nível de performance por parte das equipas, portanto, é necessário atualizar e melhorar os métodos e algoritmos da equipa para tentar atingir o objetivo da simulação ser mais parecida com a realidade e para a obtenção de bons resultados. Um dos aspetos mais básicos no futebol é a movimentação dos jogadores, qualquer ser humano a jogar futebol antes de se movimentar planeia o caminho que vai percorrer para ir ao encontro da bola ou em movimentações defensivas, ou ofensivas. Estas movimentações envolvem muitas vezes diferentes velocidades durante a progressão deste mesmo caminho, também envolve um planeamento que evite ir contra os restantes elementos de ambas equipas evitando cometer falta no caso de ser um elemento da equipa adversaria ou apenas obstrução a membros da própria equipa prejudicando assim a sua equipa. Sendo um dos aspectos mais básicos num jogo de futebol também o torna um dos mais importantes, pois sem este planeamento de caminho o jogo perderia qualidade e tornar-se-ia uma confusão, e equipas em que os jogadores se movimentem melhor normalmente são as que têm um jogo mais refinado e, ao mesmo tempo, são as mais capazes. Em suma se os robôs forem capazes de planear o caminho que irão percorrer ao longo do jogo de forma eficiente a equipa será também ela mais eficiente conseguindo melhores resultados e, em simultâneo, dando um passo ao objetivo de tornar o futebol simulado 3D mais parecido ao real.

1.4 Estrutura do Documento

- **Capítulo 1:** Introdução à dissertação onde há uma breve descrição do enquadramento, objetivos e motivação desta mesma.
- **Capítulo 2:** Estado da arte onde descrevo trabalho já feito e relacionado com tema em que o foco é *Reinforcement Learning* e alguns dos algoritmos existentes
- **Capítulo 3:** Este capitulo apresenta o ambiente geral usado na *RoboCup 3d Simulation League* e analisa os seus componentes, desde o simulador usado e o agente. Por fim descreve algum do papel que o *Reinforcement Learning* teve até agora na *RoboCup* tendo em foco a equipa FCPortugal.
- **Capítulo 4:** Ferramentas de treino descreve as principais ferramentas usadas durante o desenvolvimento da dissertação.
- **Capitulo 5:** Descrição do trabalho prático realizado durante a dissertação assim como a apresentação dos resultados
- **Capitulo 6:** Conclusões do trabalho realizado assim como algumas sugestões para trabalho futuro.

Chapter 2

Reinforcement Learning

Reinforcement learning consiste em aprender a interagir com um ambiente. Um agente aprende com as consequências das suas ações, em vez de ser explicitamente ensinado, procurando maximizar as suas recompensas. Seleciona as suas ações com base nas experiências passadas (*exploitation*) e por novas escolhas (*exploration*).

Os sistemas *Reinforcement learning* são treinados pelas próprias experiências, e em princípio podem exceder as capacidades humanas, e operar em domínios onde há falta de perícia humana [16]. Deve haver um bom equilíbrio entre *exploitation* e *exploration*, embora o primeiro par estado/ação gratificante dependa do acaso e de total *exploration*, o agente deve explorar outras ações no mesmo estado, que podem ou não serem gratificantes, e somente após algumas tentativas deve decidir explorar o par estado/ação mais gratificante.

2.1 Estrutura de Reinforcement Learning

O modelo geral da estrutura do *reinforcement learning* [3] envolve *state*, *reward*, *agent* e *policy* como mostra na figura 2.1. A *policy* define a maneira do agente aprender um comportamento num determinado momento. É o mapeamento de estados percebidos do ambiente para as ações. A função de *reward* define o objetivo em um problema de *reinforcement learning*. A interação do *agent* com o meio ambiente (*environment*) resulta na obtenção da *reward*. O *agent* define a sua localização no meio ambiente enquanto escolhe uma acção para mudança de *state*, o *agent* executa uma ação aleatória no seu meio ambiente. A ação a_t atualiza o *state* original do meio ambiente para mudar a abordagem do *agent* para um novo *state* s_{t+1} no instante t+1. No próximo *state* a *reward* do meio ambiente dá uma resposta r_{t+1}; o *agent* no *state* s_{t+1} gera um novo *feedback* para implementar a acção a_{t+1}, agora o *state* s_{t+1} é considerado o *state* inicial. O *agent* seleciona a ação de forma a receber uma recompensa imediata. A criação do próximo *state* e a interação do *agent* serão repetidas para sempre.

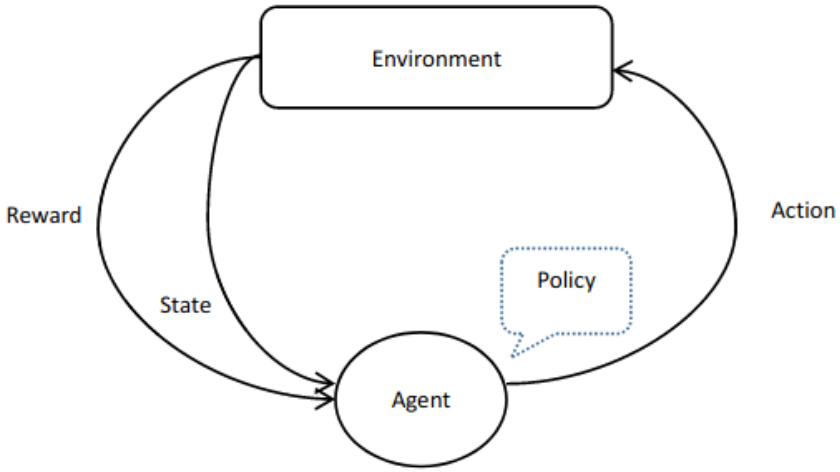

Figure 2.1: Estrutura do *Reinforcement Learning*

2.1.1 Action Spaces

Diferentes ambientes permitem diferentes ações. O conjunto de todas as ações válidas num determinado ambiente é chamado *action space*[17]. Espaços de ação discretos são definidos quando existe apenas um número finito de movimentos que o agente pode utilizar, como no trabalho desenvolvido no Go[18]. Já no caso de espaços de ação contínuos como, por exemplo, o agente controlar um robô no mundo físico, as ações são vetores de valores reais.

2.2 Algoritmos

Em *reinforcement learning* há vários algoritmos, desde os que operam em espaços discretos como o algoritmo *Q-learning*, até aqueles que operam em espaços contínuos que têm maior aplicação em robótica como, por exemplo, o *Trust Region Policy Optimization* (TPRO) e o *Proximal Policy Optimization* (PPO). Existem muitos outros que podem ser classificados como tendo um espaço de estado discreto ou contínuo, ou um espaço de ação discreto, ou contínuo.

2.2.1 Taxonomia dos algoritmos de *Reinforcement Learning*

Podemos ver na figura 2.2 a taxonomia dos algoritmos de *reinforcement learning*.

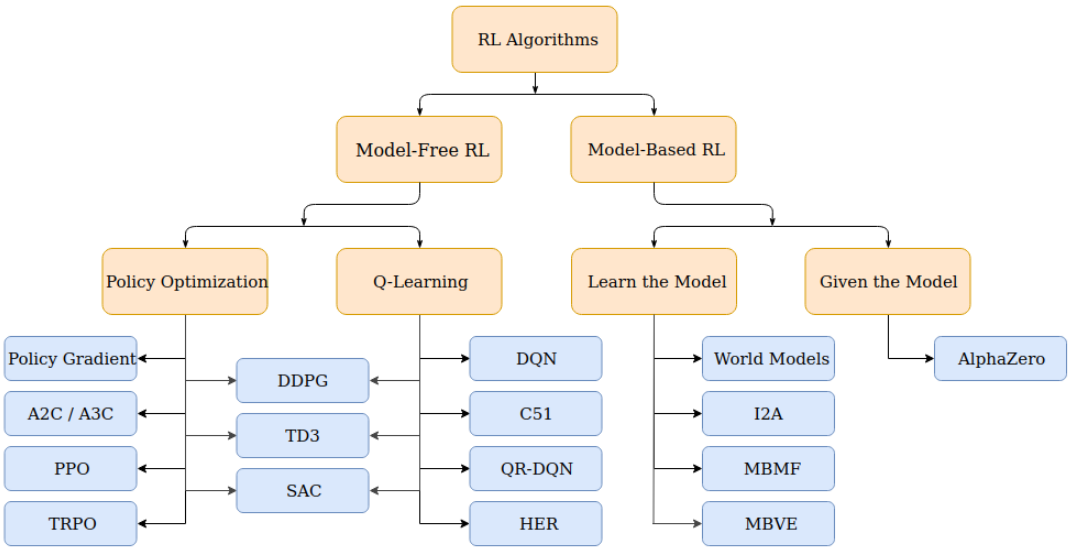

Figure 2.2: Taxonomia dos algoritmos de *Reinforcement Learning*[1]

2.2.1.1 Model-Free vs Model-Based

Um dos pontos mais importantes da ramificação nos algoritmos de *reinforcement learning*[1] é a saber se o agente tem acesso ou aprende um modelo do ambiente (função que prevê transições de estado e recompensas).

A principal vantagem em ter um modelo é que permite que o agente planeie pensando no futuro, isto é, sabe o que aconteceria em um série de escolhas possíveis e decide explicitamente entre as opções. Um exemplo desta abordagem é o *AlphaZero*[18]. Quando este método funciona, pode resultar numa melhoria na eficiência da amostra em relação aos métodos sem um modelo. A principal desvantagem é que o modelo de verdade do ambiente geralmente não está disponível para o agente, se este quiser usar um modelo deve aprender o modelo apenas por experiência o que pode criar vários desafios. o maior desafio é que a tendência do modelo pode ser explorada pelo agente o que resulta em um bom desempenho neste modelo mas no ambiente real terá um comportamento não muito bom ou até mesmo péssimo. Aprendizagem através de modelo é fundamentalmente difícil o que leva a muito esforço e tempo para calcular esse mesmo modelo e não é certo que resulte como mencionado em cima.

Figure 2.3: Eficiência de amostra[2]

2.3 Model-Free RL

2.3.1 Q-learning

Q-learning[3] é uma família de algoritmos *off-policy* de *reinforcement learning*. Este algoritmo é uma estrutura tipicamente livre de modelos. *Exploration* é usada no início e a *exploitation* é usada no final do processo de aprendizagem.

A função Q [19] define o quão útil é uma ação num determinado estado, a função Q é a seguinte:

$$\delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma * (max) * Q_t(s_{t+1}, a_t) - Q_t(s_t, a_t)]$$

Em *Q-learning*, o agente seleciona a próxima etapa pela atual tabela(*Q-table*) de recompensas, a *Q-table* é o núcleo do algoritmo. Esta é atualizada de acordo com os valores $\alpha e \gamma$.

2.3.1.1 DQN

Um dos exemplos da família de *Q-learning* é o *Deep Q-Networks* onde o agente é capaz de aprender como jogar Atari sem nenhum conhecimento anterior do mesmo.[20] DQN é capaz de combinar reinforcement learning com redes neuronais profundas[21], onde é usada uma rede convolucional profunda que usa camadas hierárquicas de filtros convolucionais para imitar os efeitos de campos recetivos.

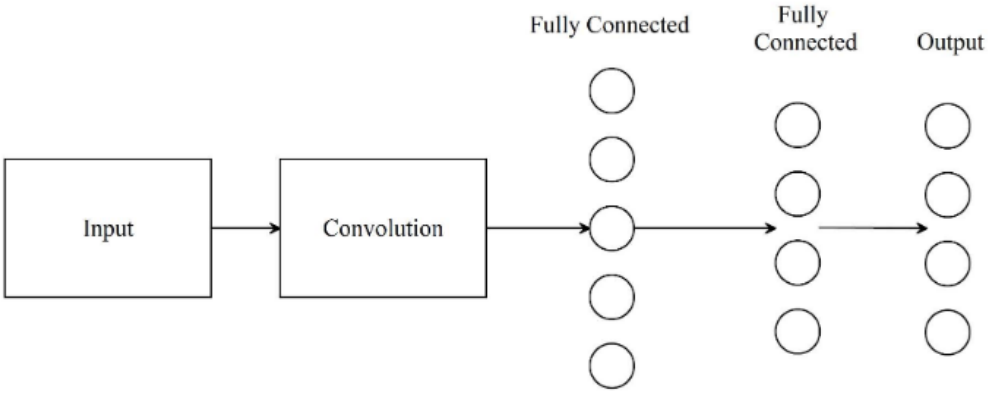

Figure 2.4: Modelo DNQ proposto por Mnih[3]

2.3.2 Policy Optimization

Nesta família de algoritmos[1] a otimização é quase sempre realizada na política, o que significa que cada atualização usa apenas os dados recolhidos enquanto age de acordo com a versão mais recente da política por isso não voltar a usar dados antigos, exigindo uma maior quantidade de dados para treinar.

2.3.2.1 Policy Gradient Methods

Os métodos de gradiente de políticas[4] calculam uma estimativa do gradiente de política e juntam-no a um algoritmo de ascensão de gradiente estocástico. A forma de estimar o gradiente mais usual é a seguinte:

$$\hat{g} = \hat{E}_t[\nabla_\theta log \pi_\theta(a_t|s_t)\hat{A}_t]$$

Onde o π_θ é uma política estocástica e $\hat{A}_t$ é uma estimativa da junção no *timestep* t. A expectativa $\hat{E}_t[...]$ representa a média empírica de um lote finito de amostras num algoritmo que alterna entre amostragem e otimização.

2.3.2.2 TRPO

O algoritmo *Trust Region Policy Optimization* (TRPO) [22] unifica a política de gradiente e a política de métodos de interação, e otimiza um determinado objetivo sujeito a uma restrição de região de confiança. TRPO é relacionado com métodos passados, por exemplo, com o *natural policy gradient* mas faz diversas alterações, principalmente porque usa uma divergência de Kullback-Leibler fixa em vez de um coeficiente de penalidade fixo.

2.3.2.3 PPO

Proximal policy optimization[4] tem alguns dos benefícios do algoritmo *Trust Region Policy Optimization* (TRPO), mas é mais simples de implementar, mais geral e têm melhor complexidade de amostra (empiricamente). PPO é uma família de métodos de otimização de política que usa vários tempos de ascensão de gradiente estocástico para realizar cada atualização de política. Em suma, PPO atinge um equilíbrio entre a facilidade de implementação, complexidade de amostra e facilidade de ajuste, tentando calcular uma atualização em cada etapa que minimiza a função de custo enquanto garante o desvio da política anterior seja relativamente pequeno[23].

O algoritmo PPO penaliza mudanças à política que afasta $r_t(\theta)$ de 1. Sendo $r_t(\theta)$ o rácio de probabilidade em que:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

e

$$r_t(\theta_{old}) = 1$$

Para atingir este objetivo usa uma função a seguinte função CLIP:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

A figura em baixo 2.5 mostra o algoritmo PPO que usa tamanhos fixos de segmentos de trajetória.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2.5: Algoritmo PPO[4]

Como podemos ver na imagem 2.6 o algoritmo PPO tem melhor desempenho que vários outros que são considerados terem um bom desempenho num domínio continuo.

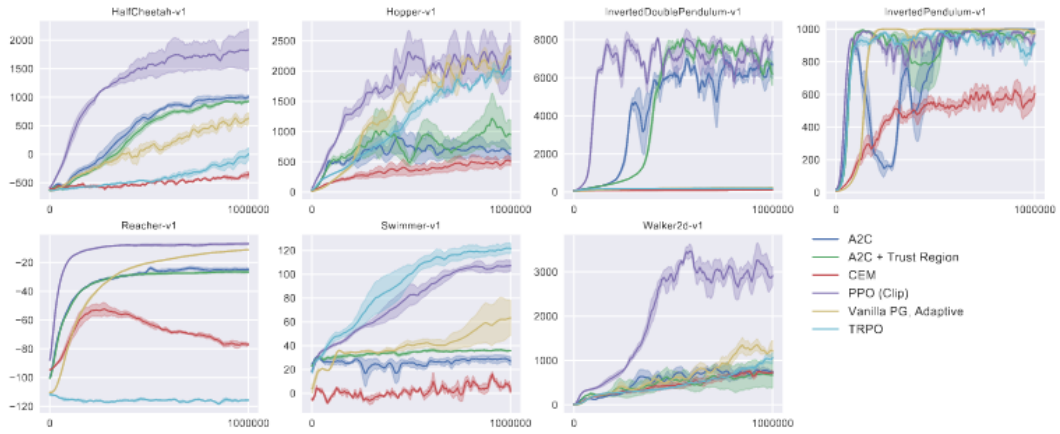

Figure 2.6: Comparação de vários algoritmos em vários ambientes de MuJoCo[4]

2.3.3 Interpolação entre policy optimization e Q-learning

As famílias *model-free* descritas em cima(*policy optimizaition* e *Q-learning*) podem ser combinadas para juntar o melhor de ambas.

2.3.3.1 TD3

Twin Delayed Deep Deterministic policy gra-dient algorithm[24] é um algoritmo que melhorou a velocidade de aprendizagem e a performance do algoritmo *Deep Deterministic Policy Gradient*, conseguiu este objetivo resolvendo os problemas que envolvem o excessivo apreço dos *Q-values* que podem levar à violação da política.

2.3.3.2 SAC

Soft actor-critic (SAC)[25] é um algoritmo de *off-policy deep reinforcement learning* e de entropia máxima que fornece uma aprendizagem eficiente da amostra enquanto retém os benefícios da maximização de entropia e estabilidade. *Soft actor-critic* é uma melhoria do algoritmo *Deep Deterministic Policy Gradient* mas em vez de otimizar uma política determinística, otimiza uma estocástica.

2.4 Model-Based RL

2.4.1 Learn the Model

Um dos tipos de métodos baseados em modelos são os *learn the model*, estes métodos têm que aprender o modelo conforme treinam. Resumindo é um método de tentativa e erro.

2.4.1.1 World Models

World models[26] é um algoritmo que pode ser treinado de uma maneira não supervisionada para aprender uma representação espacial e temporal do ambiente. Este método usa recursos extraídos do modelo do mundo como entradas para o agente e consegue treinar uma política compacta e simples que possa resolver a tarefa necessária.

2.4.2 Given the Model

O outro tipo de métodos baseados em modelos são os *given the model*, neste caso é dado o modelo do mundo logo inicialmente. Inicialmente é dado um modelo e usando a técnica do *Monte Calo Tree Search* avalia a qualidade do estado atual e planeia o próximo movimento.

2.4.2.1 AlphaZero

Alphazero[16] é um dos algoritmos mais famosos de aprendizagem por reforço. A rede neuronal deste algoritmo é treinada através de auto-jogos. Um dos exemplos é o famoso AlphaGo, podemos ver na figura a baixo a técnica de pesquisa de Monte Carlo aplicada ao Go.

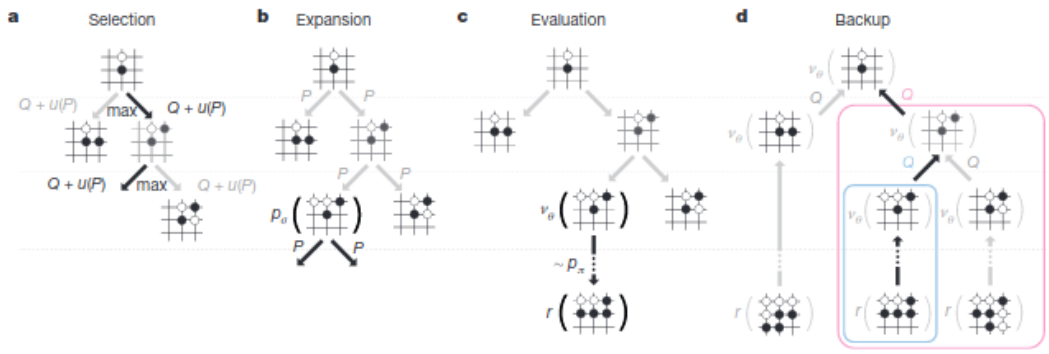

Figure 2.7: Pesquisa em árvore do Monte Carlo no AlphaGo [5]

2.5 Conclusão

Este capitulo cobre o estado de arte relacionado com *reinforcement learning*. É revista a estrutura de RL bem como os vários modelos existentes e um sumário dos vários algoritmos existentes. É esperado que todos os anos, novas técnicas e novos algoritmos surjam visto ser uma área com muita margem para progresso e atrai cada vez mais a curiosidade das diversas pessoas.

Chapter 3

RoboCup

3.1 RoboCup

Robocup [27] visa promover a inteligência artificial e a pesquisa robótica, fornecendo um problema padrão na forma de competições de futebol robótico. O objetivo do projeto é, até 2050, fazer com que uma equipa de robôs humanóides derrote os campeões humanos de futebol dessa época.

3.1.1 Simulador Simspark

Simspark[28] é um sistema genérico de simulador multi-agente físico para agentes em ambientes 3D, é baseado na estrutura flexível do aplicativo Spark. Este simulador é usado como servidor oficial de simulação 3D da RoboCup. As dimensões e a disposição do campo é a seguinte [27]:

- As dimensões do campo de futebol São de 30 por 20 metros;
- Cada baliza tem 2,1 por 0,6 metros, com uma altura de 0,8 metros;
- A grande área é de 3,9 por 1,8 metros;
- A área de penalti é a soma da largura da baliza e da grande área, perfazendo um total de 6,0 por 1,8 metros;
- O círculo central tem um raio de 2 metros;
- O campo de futebol é circundado por uma borda de 10 metros, os agentes não têm acesso para além desta borda;
- A bola de futebol tem um raio de 0,04 metros e uma massa de 26 gramas.

Em cada canto do campo e nos postes da baliza é colocado um marcador distinto. As posições destes marcadores são fixas e conhecidas por cada agente, estes percebem a posição relativa de um subconjunto de marcados juntamente com as linhas de campo no seu campo de visão, capazes de se localizarem no campo de futebol. Embora as linhas de campo tenham a mesma aparência

que estes marcadores por ser identificado por meio de um identificador como por ser observado na figura a baixo3.1.

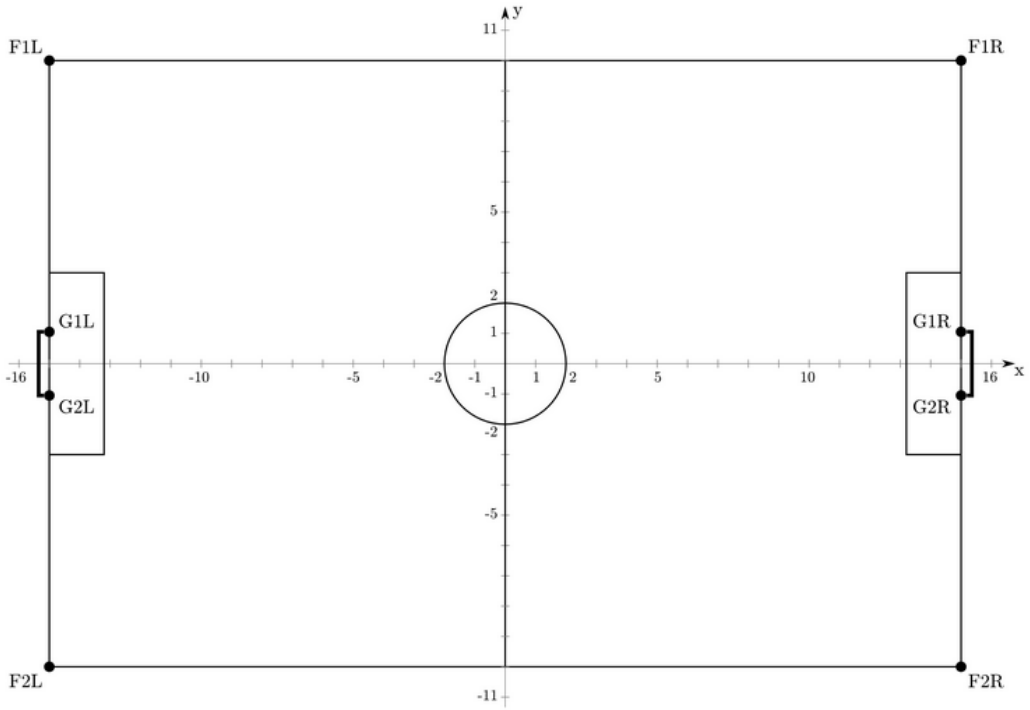

Figure 3.1: Dimensões do campo de futebol e os marcadores de campo que são percebidos por um agente

3.1.2 Ambiente da competição

O ambiente de competição[6] é composto por:

- Dois computadores de clientes, um de cada equipa, com os nomes de *client1* e *client2*;
- Um computador *server*;
- Um computador monitor(roboviz)

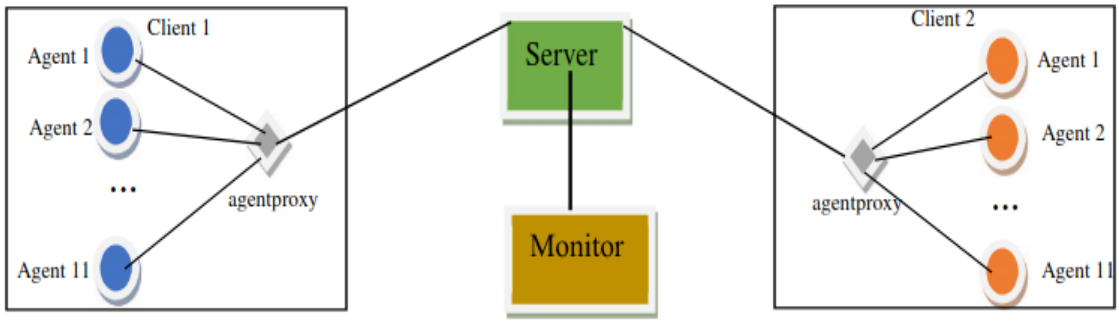

Figure 3.2: Configuração do sistema para 2021[6]

3.1.3 RoboViz

RoboViz[7] é o monitor e a ferramenta de visualização oficial para a RoboCup 3D Soccer Simulation League. Foi desenvolvida pela equipa *magmaOffenburg*, que é uma das equipas que habitualmente participa no RoboCup. Esta ferramenta faz o processamento 3D do ambiente e dos agentes, é um recurso útil para fazer *debugging* ou implementar novas estratégias porque permite a visualização das trajetórias e posições dos vários agentes como tambem da bola.

Figure 3.3: Interdace do RoboViz[7]

3.1.4 NAO Robot

As competições de RoboCup usam um modelo simulado do NAO Robô como agente e tem as seguintes características[29]:

- 25 graus de liberdade que lhe permitem mover-se e adaptar-se ao seu ambiente;
- 7 sensores de toque localizados na cabeça, mãos e pés, sonares e uma unidade inercial para perceber o seu ambiente e localizar-se no espaço;
- 4 microfones direcionais e altifalantes para interagir com humanos;
- Reconhecimento de voz e diálogo disponível em 20 idiomas;
- Duas câmaras 2D para reconhecer formas, objetos e até pessoas;
- Plataforma aberta e totalmente programável.

Há um total de 22 articulações na versão do robô NAO sem dedos, como mostra a figura 3.4. A inclinação e a rotação da cabeça são excluídos do *space of state*. O modelo de caixa pode ser visto na figura 3.5.

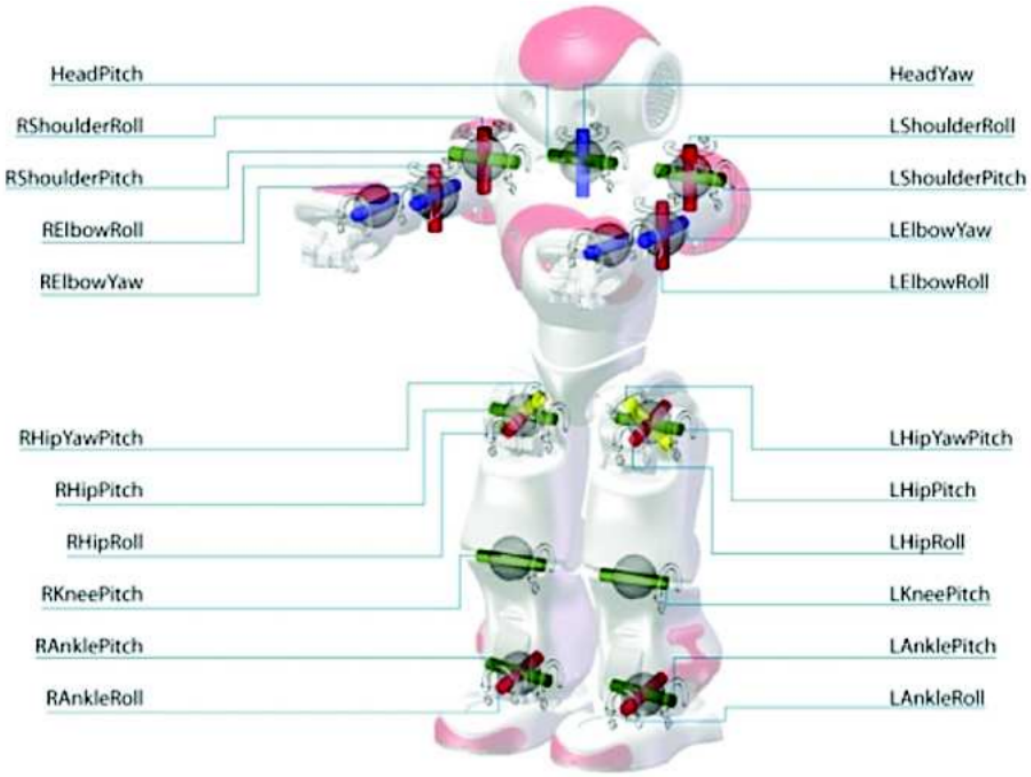

Figure 3.4: Articulações e Actuadores do Robô NAO [8]

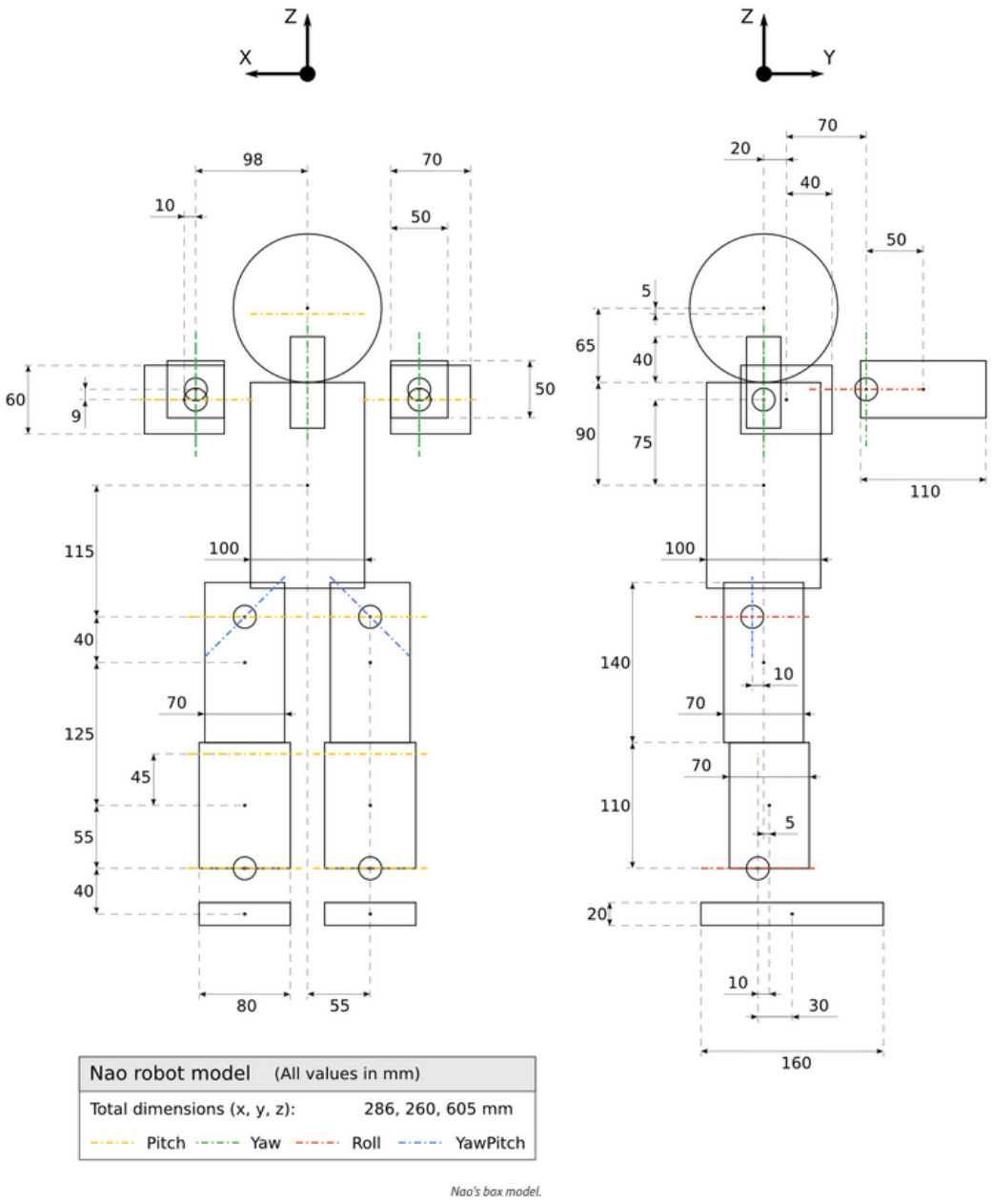

Figure 3.5: Modelo caixa do NAO robô [9]

3.2 RL na Robocup e equipa FCPortugal

Na RoboCup, em que a equipa FCPortugal compete, a comunidade cientifica continua principalmente focada em chutar[30], no guarda redes [31] e detecção de objetos sejam eles parados ou em movimento [32], [33]. Mas outro processo importante é o planeamento do caminho que deve ter em conta a velocidade e o raio de girar do estado actual, seja ele andar ou correr, também deverá ter atenção os restantes elementos das equipas em jogo.

3.2.1 Multi-agent System

Muitas das ações estudadas para RoboCop são através de algoritmos de reinforcement learning para permitir um único agente conseguir desempenho de nível humano. Porém, na maioria das situações, os agentes interagem com outros agentes para resolver um determinado problema. *Multi-agent systems* concentra-se na construção de um sistema com vários agentes independentes e em como coordená-los [34].

3.2.2 Agente FCPortugal

O agente da equipa FCPortugal é dividido em divido em vários pacotes que controlam uma parte diferente do agente, desde a comunicação com o simulador às diferentes habilidades de alto ou baixo nível desenvolvidas ate ao momento. Estes vários pacotes são os seguintes:

- **Estado do mundo** — Contem a informação sobre o meio ambiente, do estado do jogo e da posição e características dos objectos
- **Modelo do agente** — Contem a informação do modelo do agente, isto é, as posições das várias partes do corpo do robô e também é responsável por controlar as articulações do mesmo.
- **Geometria** — Este pacote contem como o nome diz as classes que definem entidades geométricas desde operação matemáticas a vetores
- **Habilidades** — Este pacote contem todas as habilidades ate agora desenvolvidas pela equipa e os motores necessários para tornas as redes neuronais que foram treinadas utilizáveis pelo agente
- **Utilidades** — Contem classes necessárias ao agente para funcionar e também contem classes dedicadas à comunicação entre os agentes e o servidor.
- **Estratégia** — Como o nome diz é responsável pela estratégia que a equipa usará durante os vários jogos e torneios que possa disputar.

3.2.3 Omnidirectional Walking

Omnidirectional Walking foi implementado pela equipa FCPortugal[10]. Para conseguir implementar foi necessario decompor em vários modulos e abordar cada módulo de forma independente, na seguinte figura 3.6 podemos ver a interação entre os diferentes módulos.

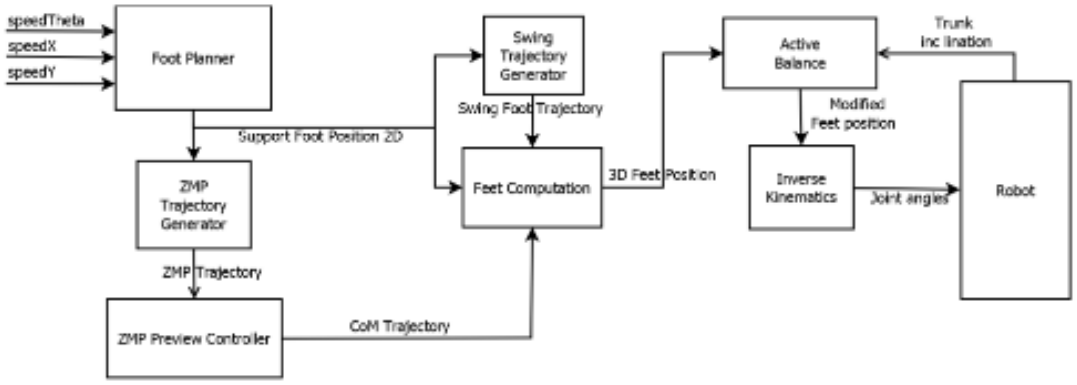

Figure 3.6: Arquitetura dos módulos do motor de andar[10]

3.2.3.1 Estabilização do andar

Geralmente para controlar o equilíbrio da locomoção bípede é usado o indicador de estabilidade ZMP como foi mostrado pela equipa FCPortugal[11]. ZMP não consegue gerar diretamente a referência para a trajetória do andar, mas consegue indicar se a trajetória gerada ira fazer com que o robô perca ou não o equilíbrio. O andar bípede pode ser representado como um problema de equilíbrio de um ZMP, visto que na fase em que só há um suporte o andar humano pode ser representado por um pêndulo invertido.

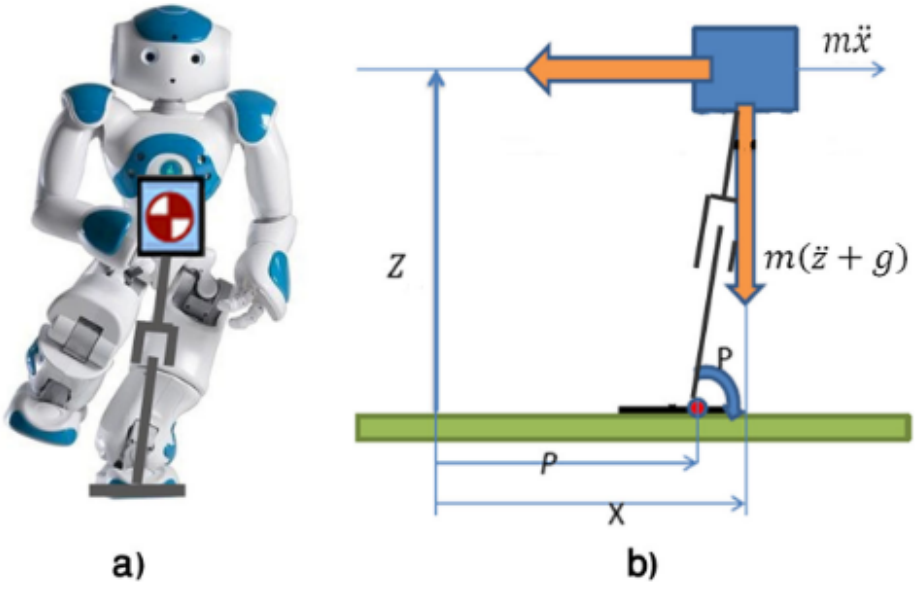

Figure 3.7: a)Vista frontal do robô NAO e o IPM b)Visão esquemática do IPM [11]

Usando este modelo cada futuro passo é calculado baseado na posição atual do pé e a direção pretendida durante o andar.

3.2.4 Walk faster

Walk faster foi apresentado pela equipa FCPortugal [11]. Foi apresentada uma abordagem numérica para gerar a trajetória CoM horizontal com base na trajetória ZMP predefinida e na trajetória CoM vertical. Nesta abordagem andar é considerado um movimento dinâmico e não se presume que seja estaticamente estável no início e no final de cada passo do andar, portanto a abordagem pode gerar um andar rápido e dinâmico. Para gerar um andar a considerar o movimento vertical CoM foi estudar o movimento da altura do quadril, otimizando os parâmetros do movimento modelado a essa altura. O agente foi capaz de aprender a andar rápido tanto para a frente como para o lado. O motor de andar desenvolvido foi testado num robô NAO em que se verificou um aumento de velocidade de 0.119 m/s para 0,34 m/s. Pode ser constatado na imagem 3.8 uma visão geral da abordagem e todos os componentes.

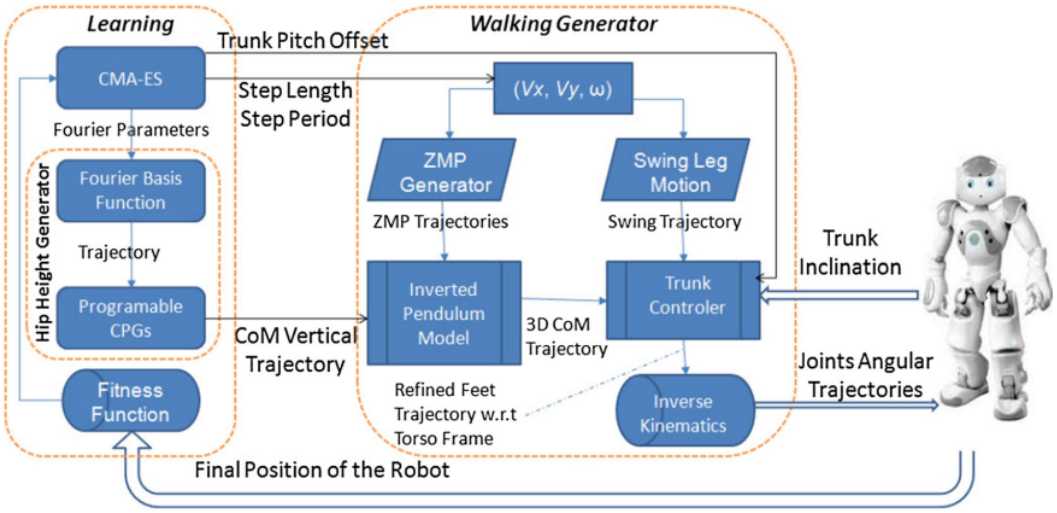

Figure 3.8: A interação entre cada componente da abordagem proposta

3.2.5 Sprinting

A equipa FCPortugal conseguiu treinar os comportamentos de correr e parar [8], esta solução foi obtida a usar o algoritmo Proximal Policy Optimization. O desempenho de correr é bastante estável e bastante superior aos antigos campeões dos desafios de correr, e também consideravelmente melhor que os algoritmos atuais usados pelas equipas do RoboCup 3D Soccer Simulation. As velocidades que os agentes conseguiram obter são na ordem dos 2,45m/s que são muito superiores às velocidades do *walk fast*.

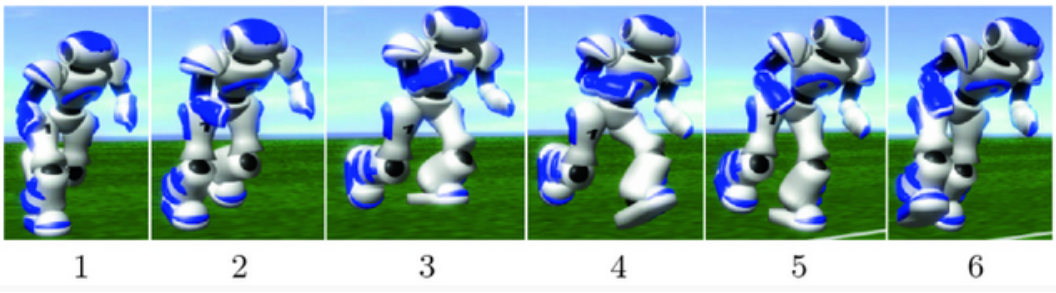

Figure 3.9: Execução do *sprint* por parte do robô NAO [8]

3.3 Conclusão

Este capitulo apresenta o ambiente geral usado na *RoboCup 3D Simulation League* e analisa os seus componentes, desde o simulador usado, o agente(NAO Robot), etc. É abordado o do papel do RL no *Robocup* descrevendo algumas das habilidades que os robôs conseguem fazer atualmente, focando no que foi desenvolvido pela equipa FCPortugal.

Chapter 4

Ferramentas de treino

4.1 Open AI Gym e Stable Baselines

O *Open AI Gym*[35] tornou-se um padrão para *benchmarking* de algoritmos de *reinforcement learning*, a grande aceitação da sua interface levou ao aparecimento de várias coleções de implementações de grande qualidade dos mais populares algoritmos de *reinforcement learning*, tais como a *Open AI Baselines* e a *Stable-Baselines*[36], estas ferramentas permitem o desenvolvimento e poder comparar algoritmos de RL.

4.1.1 Ambientes

Open AI Gym contem vários ambientes pré-feitos e, ao longo do tempo, prevê-se que irão aumentar. Podemos observar nas imagens seguintes[37] alguns exemplos como os de Atari, andar bipede entre outros.

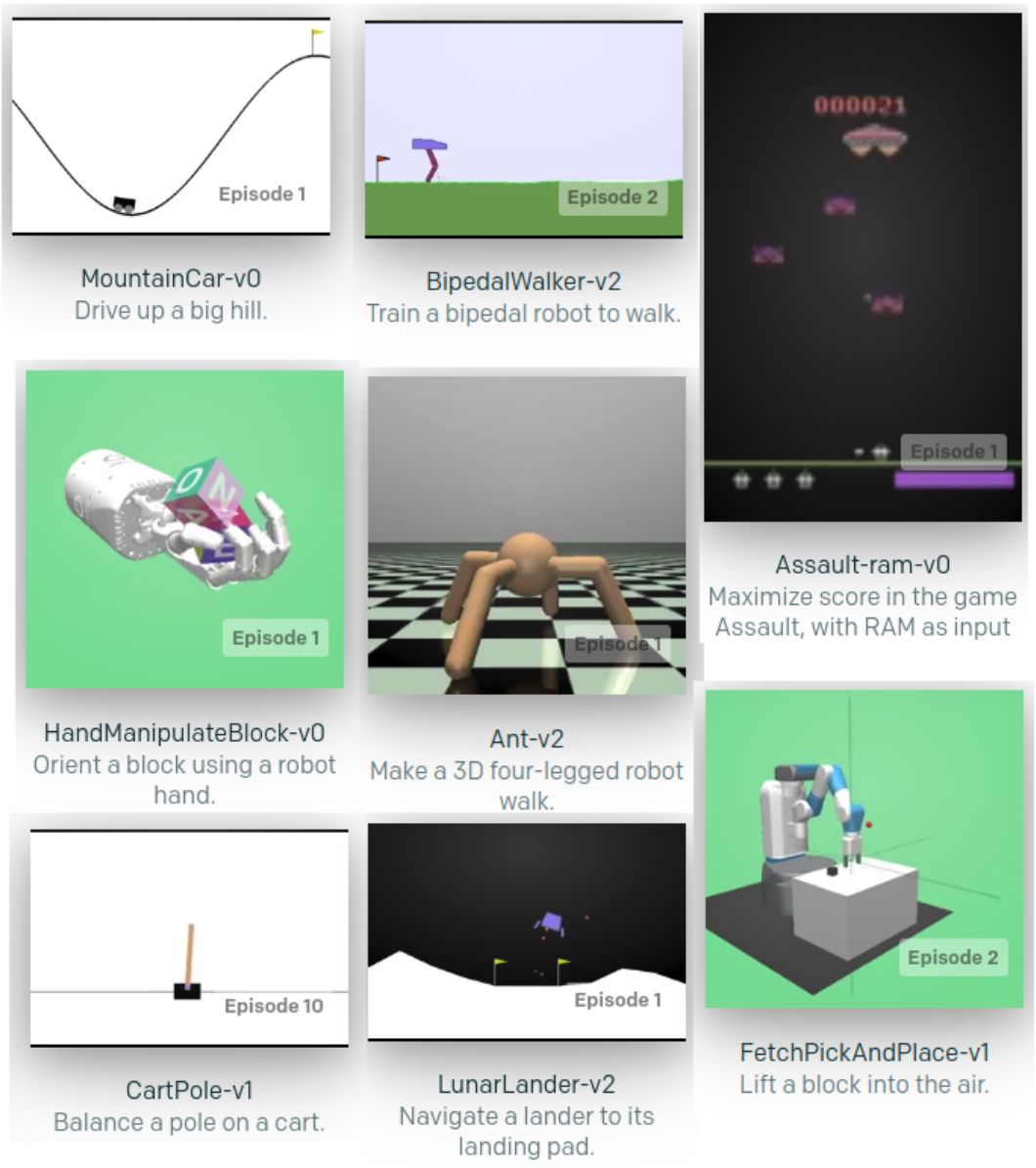

Figure 4.1: Exemplos de ambientes pré-feitos do *Open AI Gym*

Para usar um ambiente[38] inicialmente é preciso usar a função *make* para criar o ambiente em que o argumente é o nome do mesmo. Logo a seguir é chamada a função *reset* que reinicia o ambiente, isto é, todos os valores e posições voltam ao valor inicial e retorna uma observação. Esta observação representa o estado do ambiente. De seguida, as funções *render* e *step* são chamadas ate o episódio acabar(*done* ser igual a *true*) criando assim um ciclo como podemos ver em 4.2. A função *step* realiza uma ação e retorna quatro valores, que são:

- ***observation*** — um objeto específico do ambiente que representa a observação do ambiente;
- ***reward*** — quantidade de recompensa que a ação anterior conseguiu atingir;
- ***done*** — indica se é tempo de dar *reset* ao ambiente;

- ***info*** — informação útil para *debugging*.

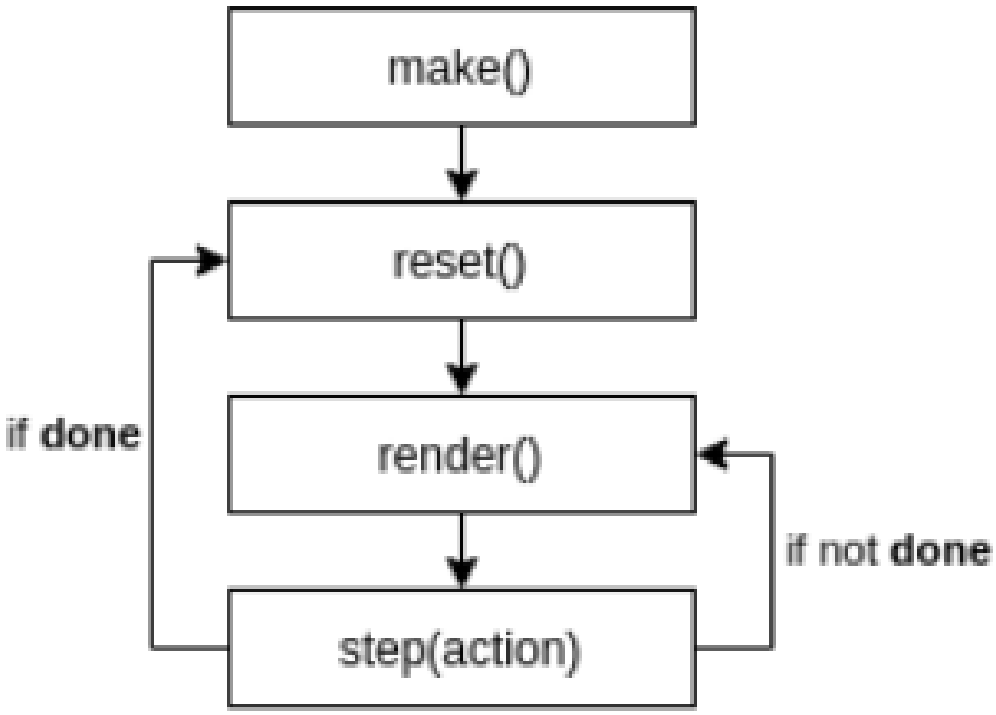

Figure 4.2: Ciclo de execução do ambiente no *Open AI Gym*

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

Figure 4.3: Exemplo de código usando um ambiente pré-feito

4.1.2 Algoritmos de RL

Existem vários algoritmos de *reinforcement learning* como foi explicado no capítulo anterior. O projeto *stable baselines* tem implementado vários algoritmos como podemos ver na figura 4.4 onde também podemos ver as características dos mesmos tais como as políticas recorrentes, o tipo de espaço e o multiprocessamento.

Name	Refactored [1]	Recurrent	`Box`	`Discrete`	Multi Processing
A2C	✓	✓	✓	✓	✓
ACER	✓	✓	✗ [4]	✓	✓
ACKTR	✓	✓	✓	✓	✓
DDPG	✓	✗	✓	✗	✓ [3]
DQN	✓	✗	✗	✓	✗
HER	✓	✗	✓	✓	✗
GAIL [2]	✓	✓	✓	✓	✓ [3]
PPO1	✓	✗	✓	✓	✓ [3]
PPO2	✓	✓	✓	✓	✓
SAC	✓	✗	✓	✗	✗
TD3	✓	✗	✓	✗	✗
TRPO	✓	✗	✓	✓	✓ [3]

Figure 4.4: Algoritmos presentes no *Stable Baselines*[12]

Como podemos verificar diferentes algoritmos têm diferentes tipos de espaços. Podemos classificar os espaços em quatro tipos [12]:

- ***Box*** — Uma caixa N-dimensional que contém todos os pontos do espaço de ação;
- ***Discrete*** — Uma lista de ações possíveis, em que cada intervalo de tempo apenas uma das ações pode ser usada;
- ***MultiDiscrete*** — Uma lista de ações possíveis, onde cada intervalo de tempo, apenas uma ação de cada conjunto discreto pode ser usada.
- ***MultiBinary*** — Uma lista de ações possíveis, onde cada intervalo de tempo de qualquer uma das ações pode ser usado em qualquer combinação.

4.1.3 Ambientes Vectorizados

Alguns dos algoritmos implementados no *Stable Baselines* usam ambientes vectorizados.

Ambientes vectorizados[13] são um método para empilhar vários ambientes independentes num único ambiente conseguindo correr vários episódios em sincronismo. Desta forma em vez de treinar um agente usando *Reinforcement Learning* num único ambiente por *step*, usando ambientes vectorizados é possível treinar este agente em n ambientes por *step*. Portanto, as ações, as observações, as recompensas passam a ser em forma de vetores, neste caso de dimensão n. Havendo sincronismo e se um ambiente acabar primeiro terá que esperar que todos os ambientes acabem os seus *steps*.

No *Stavle Baselines* existem dois tipos de ambientes vectorizados sendo eles o *DummyVecEnv* e o *SubprocVecEnv*. Podemos ver as diferentes características na imagem seguinte.

Name	Box	Discrete	Dict	Tuple	Multi Processing
DummyVecEnv	✓	✓	✓	✓	✗
SubprocVecEnv	✓	✓	✓	✓	✓

Figure 4.5: Características dos 2 tipos de ambientes vectorizados no *Stable Baselines*[13]

O *DummyVecEnv* [13] chama cada ambiente em sequência num único processo *Python*, sendo mais útil para ambientes simples e com menor peso computacional. O *SubprocVecEnv* distribui cada ambiente pelo seu processo *Python* conseguindo assim um aumento significativo da velocidade quando usados ambientes mais complexos.

4.2 FCPgym

A grande variedade de problemas do contexto da *3D Simulation League* em que a equipa FCPortugal e este trabalho se inserem são adequados para serem resolvidos através de algoritmos e técnicas de *reinforcement learning* sendo por isso útil e adequado usar as ferramentas *Open AI Gym* e *Stable-Baselines*. A equipa FCPortugal[39] desenvolveu uma *framework* para treinar e partilhar os algoritmos de RL sendo que se baseou na interface da *Open AI Gym*, sendo o nome desta FCP Gym. O FCP Gym foi construído em cima do código base da equipa para ser possível usar os comportamentos já existentes. O código base da equipa está escrito em C++ e o *Stable-Baselines* está implementado em python, logo para usar esta coleção de algoritmos assim como outras bibliotecas de computação numérica como o *TensorFlow* entre outras que têm interfaces em python foi usado a biblioteca *pybind11* que permite operar entre as duas linguagens de programação.

4.2.1 Processo de optimização

O processo de otimização[14] começa com a criação de N ambientes do *FCP Gym* em paralelo, cada desses ambientes cria uma instância do *SimSpark* onde o agente treina. A comunicação feita entre o agente e o optimizador é efetuada através de um *socket* TCP. Um episódio começa quando o optimizador envia uma ação ao agente, de seguida o agente executa a ação dada e informa o optimizador sobre o novo estado da simulação, isto é, envia uma observação. Depois, o optimizador avalia a ação que foi efectuada baseado na observação que recebeu e por fim o ciclo repete-se em que novamente o optimizador envia uma nova ação ao agente. Como disse inicialmente existe a criação de N ambientes em paralelo logo este ciclo é executado em cada um destes ambientes em simultâneo.

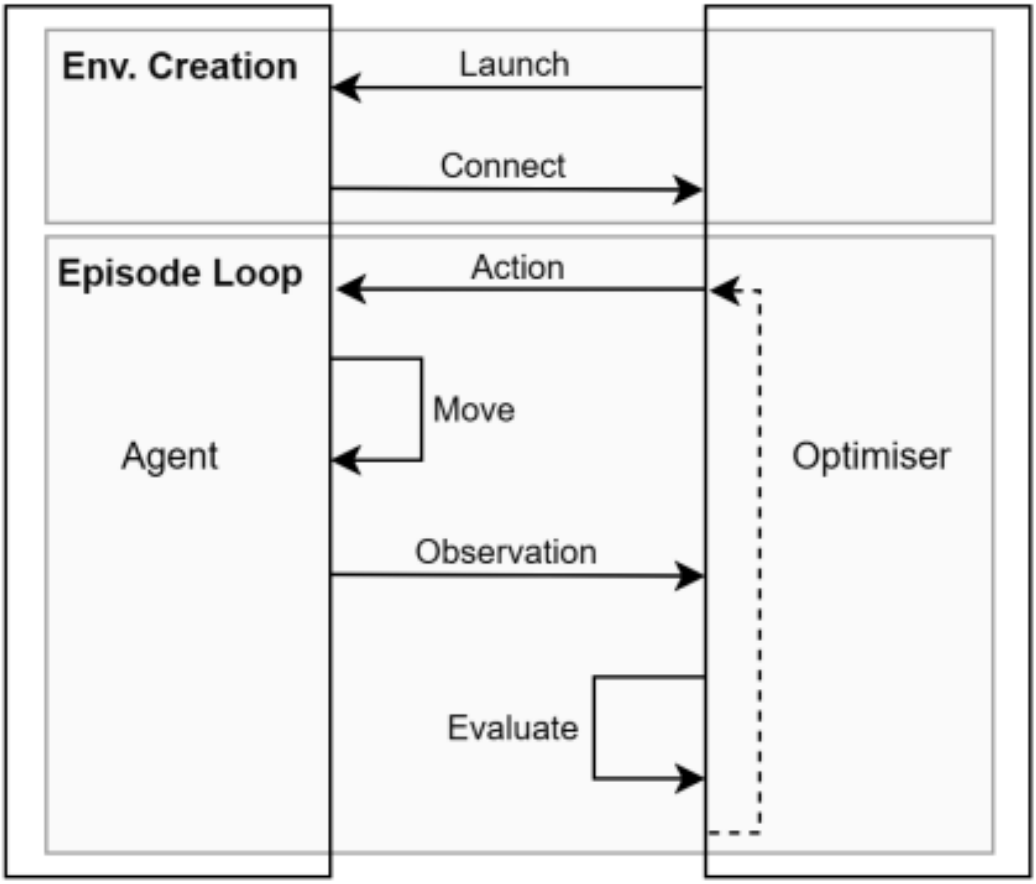

Figure 4.6: Comunicação entre o agente e o optimizador[14]

4.3 Conclusão

Neste capítulo é explicado as ferramentas de treino usadas durante a parte prática. Inicialmente é explicado como as ferramentas *Open AI Gym* e *Stable Baselines* funcionam e podem ser usadas para treinar usando algoritmos de *reinforcement learning*. Por último é abordado a ferramenta desenvolvida pela equipa FCPortugal que usa para o desenvolvimento de novas habilidades e por consequência o melhoramento do desempenho da equipa.

Chapter 5

Planeamento de caminho

Como dito anteriormente o objetivo desta dissertação é treinar usando *Reinforcement Learning* o robô humanoide a planear um caminho para conseguir alcançar um determinado local. Irá ser usado o comportamento de andar já treinado pela equipa e descrito previamente. Atualmente a equipa FCPortugal ao usar este comportamento de andar planeia o caminho através de condições *hard-coded*. Este capitulo vai descrever dois diferentes problemas e a maneira como foram resolvidos cada um deles usando as ferramentas de treino *FCPgym* e *Stable Baselines*. Os problemas estão divididos em duas categorias:

- Planear o caminho ate a um determinado ponto sem qualquer obstáculo.
- Planear o caminho ate a um determinado ponto com obstáculos.

5.1 PPO2

Para todos os treinos foram usados o algoritmo PPO2 disponível no *Stable Baselines* com a ajuda da ferramenta *FCPgym*. Os parâmetros deste algoritmo estão predefinidos na documentação da [40].

- ***Policy*** = MlpPolicy(2 layers of 64 nodes [41])
- ***Gamma*** = 0,99
- ***Entropy coefficient*** = 0,01
- ***Learning rate*** = 0,00025
- ***Value function coefficient*** = 0,5
- ***Maximum Gradient Norm*** = 0,5
- ***Lam*** = 0,95
- ***Number of training minibatches per update*** = 4

- ***Number of epoch when optimizing the surrogate*** = 4
- ***Clip Range*** = 0,2

Para a resolução dos problemas a seguir foi usado multiprocessamento sendo usados 25 ambientes em paralelo usando ambientes vectorizados.

5.2 Planeamento de caminho sem obstáculo

5.2.1 Experiência 1

Esta primeira experiência consiste em usar o comportamento de andar em que o agente apenas terá que andar em linha reta para ser bem sucedido.

5.2.1.1 Action Space

A *skill* usada tem como entrada três velocidade, uma segundo o eixo das abcissas, a segunda referente ao eixo das ordenadas e a última é uma velocidade angular.

Os eixos das velocidades são relativas ao agente sendo que a primeira velocidade (Vx) permite o agente andar para a frente ou para trás, a segunda (Vy) permite o agente andar para os lados, isto é, esquerda e direita, e a última (Vtheta) permite o agente descrever curvas.

Não foi usada a velocidade Vy porque com a combinação das outras duas velocidades é possível descrever qualquer trajetória e a velocidade do agente ao andar para os lados é muito menor que em frente.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.2.1.2 *Observation Space*

O espaço de observação neste primeiro problema tem os seguintes valores:

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final

5.2.1.3 Episódio

O agente começa na posição inicial (-5, 0) e o *target* final na posição (0,0) como pode ver na imagem seguinte.

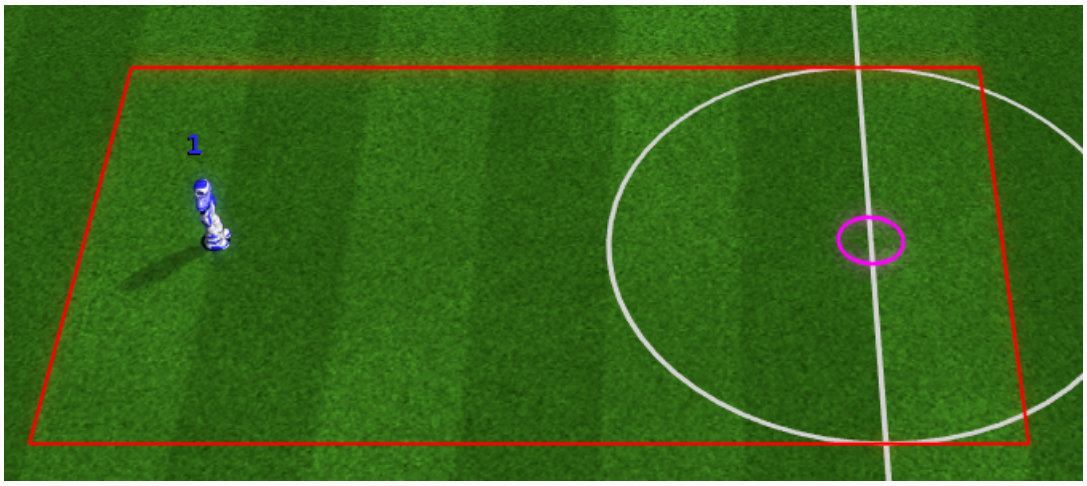

Figure 5.1: Situação inicial do episódio da experiência 1

As condições para o episódio acabar são as seguintes:

- A posição do agente não sair fora da área que pode ser vista na figura 5.1. Isto é, o valor da coordenada x da posição do agente não ser menor que -6 e maior que 1 e o valor da coordenada y não ser menor que -2 e maior que 2.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25. Pois, a partir deste valor o agente cai.
- A distância entre o robô e a posição final ser menor que 0,25
- Duração do episódio ser menor que 2000 *timesteps*

5.2.1.4 Recompensa

Normalmente existe uma recompensa evidente para qualquer tipo de otimização em que se conseguir atingir o objetivo, sendo neste caso conseguir chegar à posição final a recompensa seria 1 e se não conseguisse a recompensa seria 0. Mas muito provavelmente o agente iria falhar porque neste caso iria estar a divagar pelo campo sem saber se estaria perto ou não de atingir o objetivo, pois a cada ação que tomasse iria receber o valor 0.

Por este motivo a recompensa usada para este treino poderia ser a diferença entre a distância atual do agente à posição final pretendia e a distância no *timestep* anterior entre a posição do agente e a posição final pretendida. Mas esta função recompensa continua com dois problemas, o primeiro é que poderia ser negativa ao se afastar da posição final pretendida e recompensas negativas podem levam ao agente evitar por completo determinadas ações resultando em comportamentos diferentes

do pretendido em determinadas situações. O segundo problema passa por não haver um incentivo ao agente atingir a posição final no menor tempo possível, pois o total da soma das recompensas no final de um episódio seria igual à distância inicial do agente para a posição pretendida.

A resolução destes dois problemas é fácil, pois para o primeiro basta criar uma condição em que só nos casos em que o agente se aproximou da posição pretendida é calculada a recompensa, isto é, a distância para a posição final atual ser menor que a distância anterior, nos outros casos a recompensa tem um valor igual a zero. Para o segundo problema se for aplicada o quadrado à diferença das distâncias o somatório das recompensas no final do episódio será tanto maior quando maior for a distância que o agente percorreu entre dois *timesteps*.

Desta forma temos:

$$se \quad D(t-1) - D(t) > 0 :$$

$$R(t) = [D(t-1) - D(t)]^2$$

Em que, R é a recompensa, D a distância e t o *timestep*

5.2.1.5 Resultados

Sendo um problema relativamente simples em que o robô só teria que aprender a andar em frente até atingir a posição final desejada é esperado que a percentagem de sucesso seja alta, mas o robô devido a tentar atingir a velocidade máxima possível e descrever o trajeto cai várias vezes. Foi possível verificar que um dos motivos de cair é devido à taxa de atualização de valores de velocidade (Vx e Vtheta) como podemos constatar na tabela a seguir. O ambiente atualiza a cada vez que é feito um *sync*, sendo que cada *sync* corresponde a 0,02 segundos.

Timesteps p/ completar treino	*sync*	Nº episódios	Nº ep. com sucesso	Nº de vezes que Cai
900k	1	100	0	100
650k	10	100	50	50
400k	20	100	88	12
200k	30	100	83	17

Table 5.1: Resultados da experiência 1, caminho sem obstáculo

Podemos verificar que a melhor taxa de atualização dos valores de velocidade é a cada 20 *syncs*, isto é, a cada 0,4 segundos e que para completar o treino é tanto mais rápido o quanto menor for esta taxa de atualização.

A velocidade média do robô ao longo do percurso usando os 20 *syncs* foi de 0,70009 m/s.

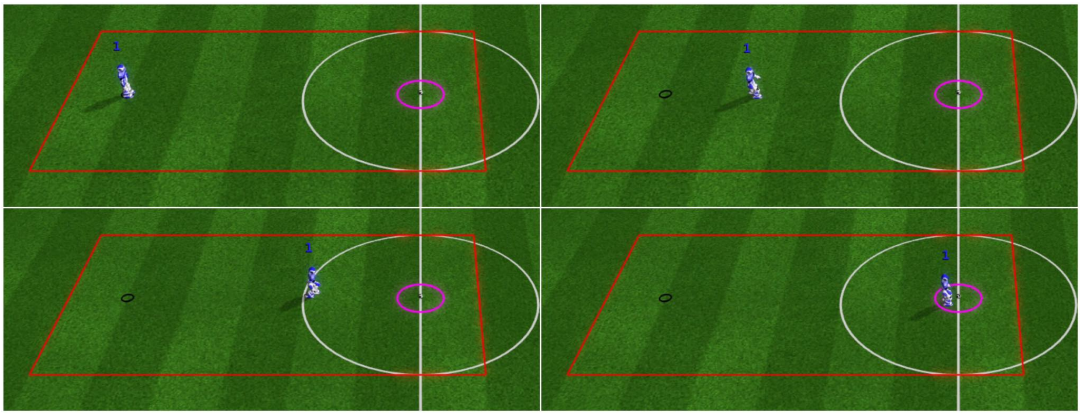

Figure 5.2: Exemplo do percurso do robô após treino na experiência 1

5.2.2 Experiência 2

Nesta experiência a posição e a orientação inicial do robô irá variar.

5.2.2.1 Action Space

O *action space* é o mesmo que na experiência 1.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.2.2.2 *Observation Space*

Assim como o *action spcace*, o *Observation Space* também é o mesmo.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final

5.2.2.3 Episódio

Neste problema a posição final pretendida será novamente nas coordenadas (0,0) e o agente ir aparecer aleatoriamente à volta do *target* a uma distância que varia entre 3 metros e 5 metros, na figura 5.3 podemos observar a área onde o robô pode ter posição inicial, sendo ela delimitada pelas duas linhas azuis claras. A orientação inicial do agente em relação à posição final pretendida será de 0 primeiro e num segundo caso também será aleatória.

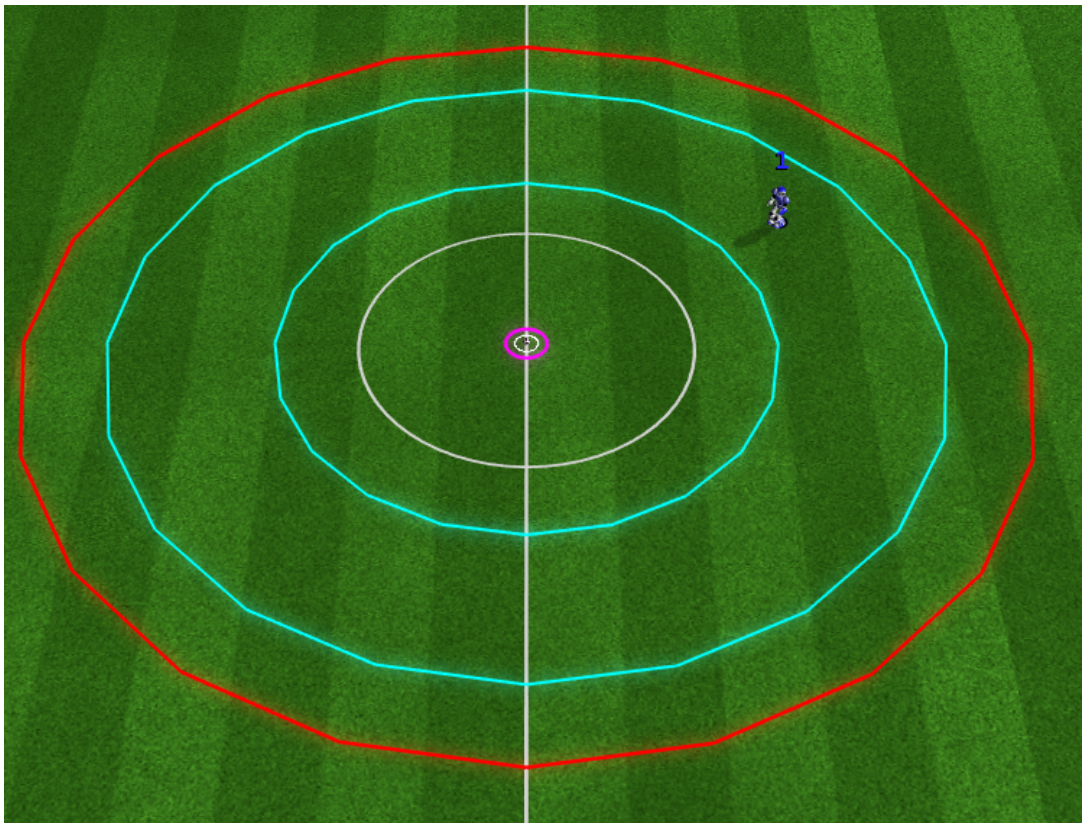

Figure 5.3: Situação inicial do episódio da experiência 2, posição inicial do agente variável

As condições para o episódio acabar são as seguintes: Apenas há alteração na área permitida para o agente percorrer e na duração do episódio sendo esta maior devido à complexidade da experiência também ser maior.

- A posição do agente não sair fora da área delimitada pela linha vermelha que pode ser vista na figura 5.3. Isto é, a distância do robô à posição final pretendida não poderá ser maior que 6 metros.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25. Pois, a partir deste valor o agente cai.
- A distância entre o robô e a posição final ser menor que 0,25
- Duração do episódio ser menor que 3000 *timesteps*

5.2.2.4 Recompensa

A recompensa também será a mesma da experiência anterior sendo ela a seguinte:

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = [D(t-1) - D(t)]^2$$

Em que, R é a recompensa, D a distância e t o *timestep*

5.2.2.5 Resultados

Como podemos observar na figura 5.4 a distribuição da posição inicial do agente em 100 episódios podendo verificar uma boa distribuição pelos quatro quadrantes.

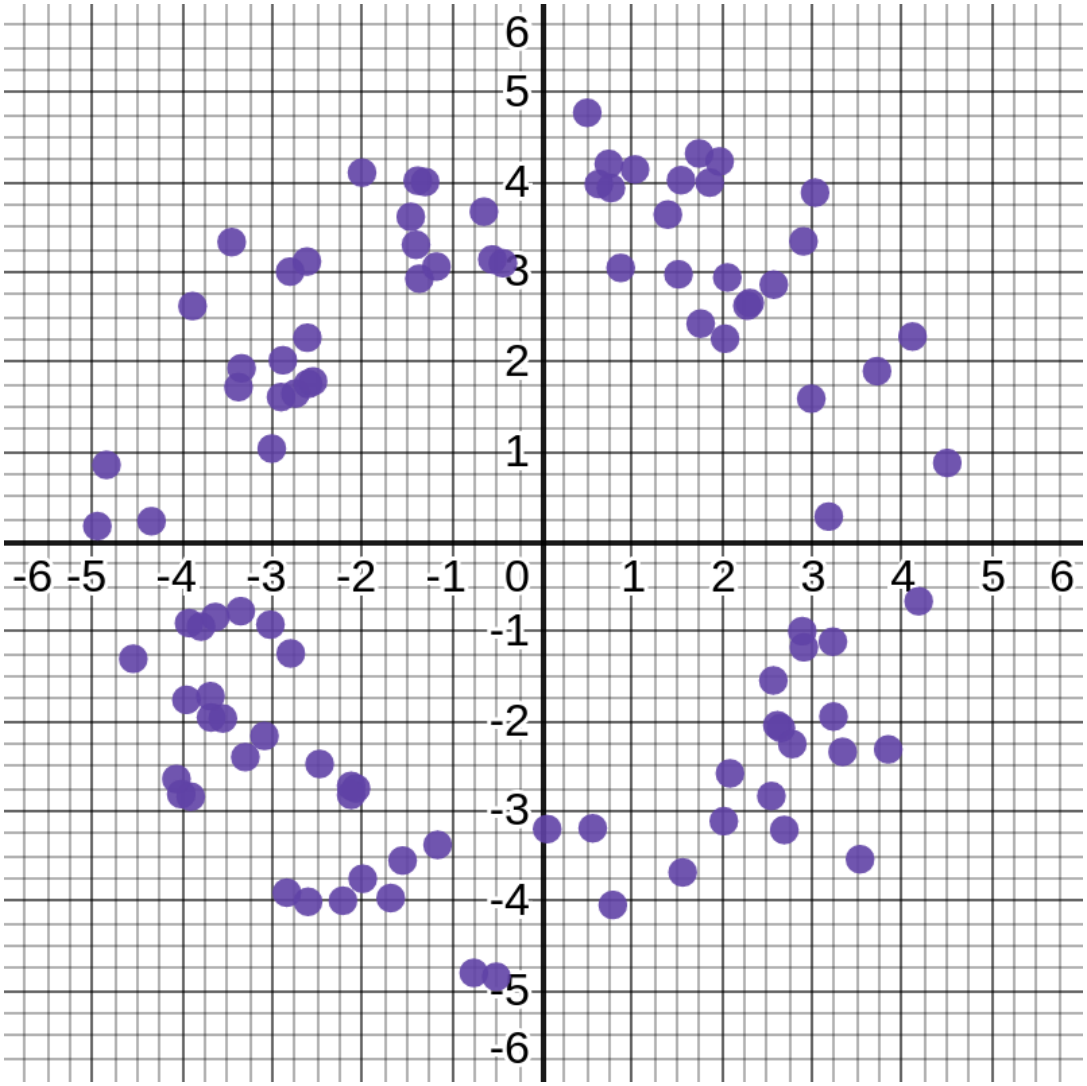

Figure 5.4: Distribuição da posição inicial do agente em 100 episódios

Inicialmente foram obtidos os resultados de uma versão mais simples do problema em que a orientação relativa inicial é igual a zero, com estas condições o problema é muito parecido à experiência 1 onde o agente apenas tem que andar em frente ate ao *target*, obtendo por isso percentagem de sucesso muito parecidas.

Numa segunda fase foi alterada a condição inicial da orientação relativa para aleatória onde se verificou percentagens de sucesso na mesma ordem de valores da orientação relativa inicial a zero, a única variação verificada foi o tempo que demorou a treinar ate atingir bons resultados.

Timesteps Treino	ORI	Nº de episódios	Nº ep. c/ sucesso	Nº Cai	Teste nº2 (%)
2,050M	0	100	80	20	81
2,450M	0	100	90	10	87
2,500M	Aleatória	100	83	17	84
3,000M	Aleatória	100	87	13	86
3,250M	Aleatória	100	87	13	90
3,500M	Aleatória	100	89	11	86

Table 5.2: Resultados da experiência 2 de planeamento de caminho sem obstáculo

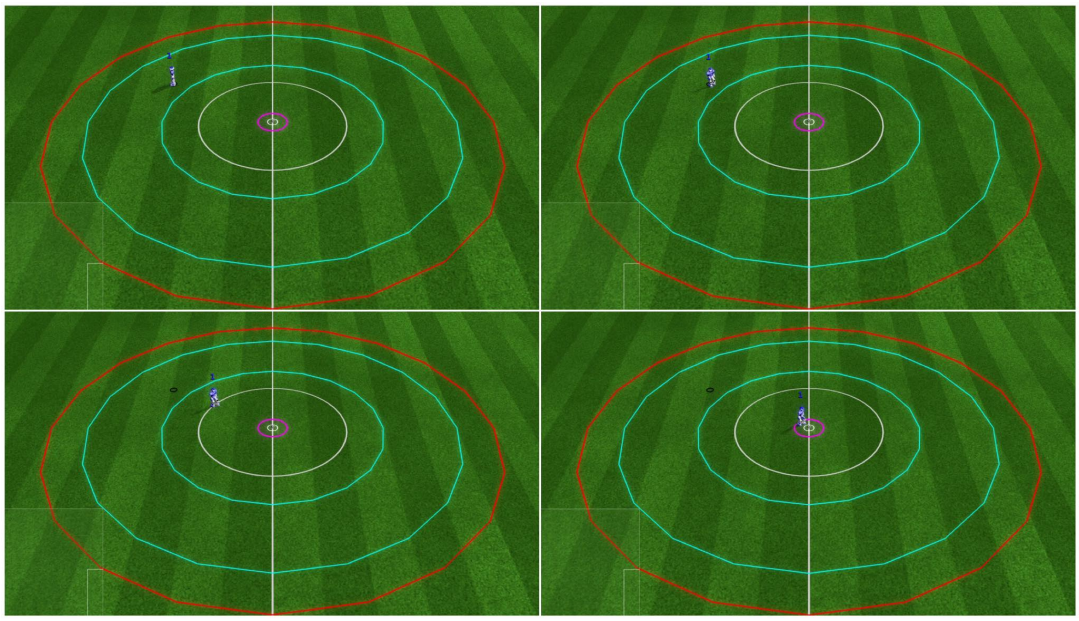

Figure 5.5: Exemplo do percurso do robô após treino na experiência 2

5.2.3 Experiência 3

Nesta experiência o objetivo é melhorar o tempo de reação, pois nas experiências anteriores as velocidades Vx e Vtheta são atualizadas a cada 20 *syncs* que correspondem a 0,4 segundos.

Como um dos grandes objetivos é o robô conseguir desviar-se de obstáculos sejam eles estáticos ou dinâmicos este tempo de reação é muito grande.

5.2.3.1 Action Space

O *action space* é o mesmo que nas experiências anteriores.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.2.3.2 *Observation Space*

Assim como o *action spcace*, o *Observation Space* também é o mesmo.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final

5.2.3.3 Episódio

Neste problema é usado o mesmo modelo de episódio que na experiência anterior em que a posição final pretendida será novamente nas coordenadas (0,0) e o agente ir aparecer aleatoriamente à volta do *target* a uma distância que varia entre 3 metros e 5 metros e com uma orientação inicial aleatória, na figura 5.6 podemos ver a área onde o robô pode ter posição inicial, sendo ela delimitada pelas duas linhas azuis claras.

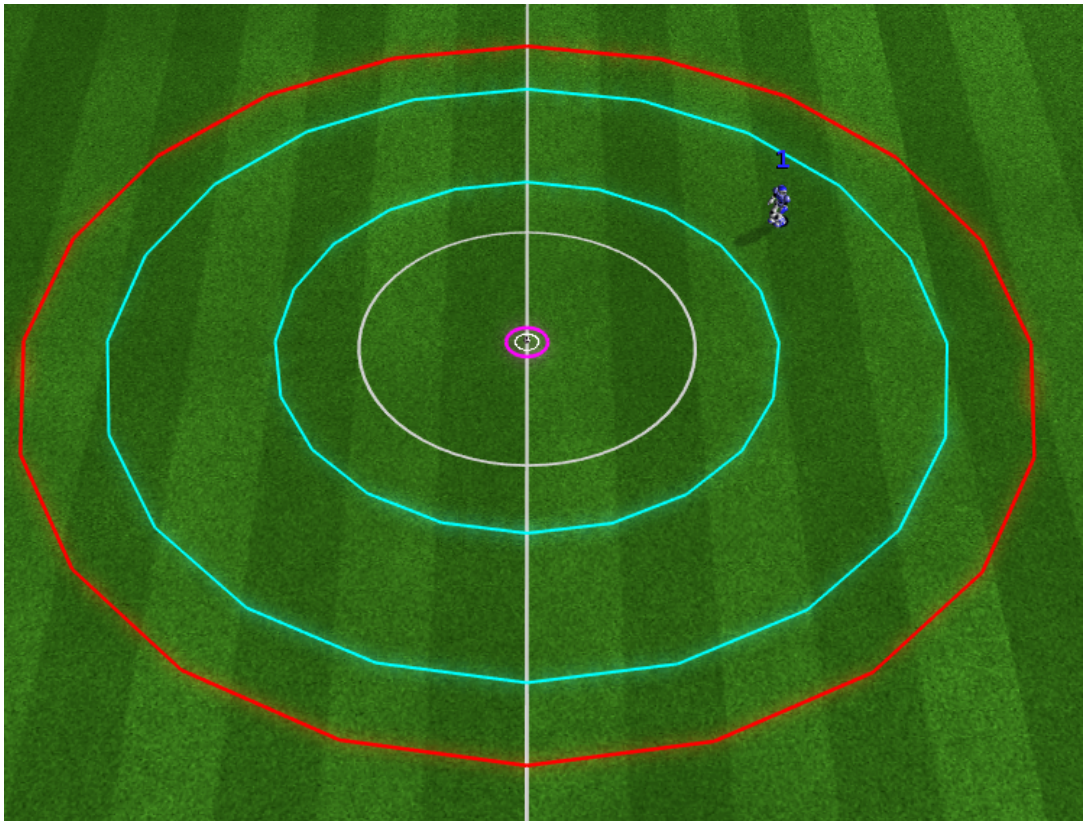

Figure 5.6: Situação inicial do episódio da experiência 3, posição inicial do agente variável

As condições para o episódio acabar novamente as mesmas.

- A posição do agente não sair fora da área delimitada pela linha vermelha que pode ser vista na figura 5.3. Isto é, a distância do robô à posição final pretendida não poderá ser maior que 6 metros.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25. Pois, a partir deste valor o agente cai.
- A distância entre o robô e a posição final ser menor que 0,25
- Duração do episódio ser menor que 3000 *timesteps*

5.2.3.4 Recompensa

A recompensa também será a mesma da experiência anterior sendo ela a seguinte:

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = [D(t-1) - D(t)]^2$$

Em que, R é a recompensa, D a distância e t o *timestep*

5.2.3.5 Resultados

Como visto anteriormente, o que dificulta ter um tempo de reação mais baixo é o agente cair mais vezes, pois durante o treino existem variações bruscas de grandes velocidades provocando o agente a se desequilibrar. Partindo deste princípio foi aumentado o intervalo das velocidades e diminuído o número de *syncs*, pois mantendo os parâmetros do algoritmo PPO2 a variação absoluta ao longo do treino do valor das velocidades será menor. Por fim foi comparado com as experiências anteriores, como podemos constatar na tabela seguinte.

Nº Exp.	Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
1	[0:1]	[-1:1]	1	100	0	100
1	[0:1]	[-1:1]	10	100	50	50
2	[0:1]	[-1:1]	20	100	89	11
3	[0:10]	[-20:20]	10	100	68	32
3	[0:5]	[-10:10]	5	100	65	35
3	[0:2]	[-5:5]	5	100	63	37

Table 5.3: Diminuição do número de *syncs* e aumento do intervalo das velocidades e comparação com as experiências 1 e 2

Como podemos ver as percentagens de sucesso são mais baixas que na experiência anterior, mas se for comparado com a experiência 1 em que o agente apenas tinha que andar em frente ate à posição final nota-se uma melhoria de resultados. Pois, com uma capacidade de reação de 10

syncs, que correspondem a 0,2 segundos a percentagem de sucesso é de 50%, com esta alteração as percentagens melhoraram mesmo com uma capacidade de reação de 0,1 segundos (5 *syncs*).

Apesar da melhoria dos resultados, ainda não são suficientemente satisfatórios, por isso, a recompensa foi alterada e feita uma interpolação com o valor da velocidade usado anteriormente evitando assim variações mais bruscas. A mudança da recompensa é pequena, apenas se multiplicou por uma constante, pois os valores a cada iteração eram muito baixos.

Nº Exp.	Inter. vX	Inter. vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai	Observações
1	[0:1]	[-1:1]	1	100	0	100	
1	[0:1]	[-1:1]	10	100	50	50	
2	[0:1]	[-1:1]	20	100	89	11	
3	[0:10]	[-20:20]	10	100	68	32	
3	[0:1]	[-1:1]	1	100	18	82	20 * Recompensa
3	[0:1]	[-1:1]	1	100	15	85	20 * Recompensa e interpolação 75% valor anterior
3	[0:0,65]	[-1:1]	1	100	83	17	20 * Recompensa e interpolação 50% valor anterior

Table 5.4: Aumento da recompensa, interpolação com a velocidade anterior e comparação com as experiências 1 e 2

O aumento da recompensa ajudou a melhorar os resultados como se pode ver ao comparar com a experiência 1. O uso da interpolação com o valor da velocidade anterior não mostrou melhorias aos resultados, mas a diminuição da velocidade Vx mostrou um aumento drástico da percentagem de sucesso. Sabendo que o aumento da recompensa e a limitação da velocidade do agente ajuda na obtenção de percentagens de sucesso mais altas foi usado uma combinação de ambos.

Nº Exp.	Inter. vX	Inter. vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai	Observações
1	[0:1]	[-1:1]	1	100	0	100	
1	[0:1]	[-1:1]	10	100	50	50	
2	[0:1]	[-1:1]	20	100	89	11	
3	[0:10]	[-20:20]	10	100	68	32	
3	[0:1]	[-1:1]	1	100	18	82	$20 * R$
3	[0:0,5]	[-1:1]	5	100	98	2	$\frac{20 * R}{syncs}$
3	[0:0,7]	[-1:1]	5	100	85	15	$\frac{20 * R}{syncs}$
3	[0:0,5]	[-1:1]	1	100	99	1	$\frac{20 * R}{syncs}$
3	[0:0,7]	[-1:1]	1	100	84	16	$\frac{20 * R}{syncs}$

Table 5.5: Aumento da recompensa, limitação do valor de Vx e comparação com as experiências 1 e 2

Como se pode ver na tabela 5.5 as percentagens de sucesso quando há uma limitação da velocidade Vx a 0,5 m/s e um aumento da recompensa são de perto de 100% tanto no caso em que se usa um tempo de reação de 0,1 segundos (5 *syncs*) ou de 0,02 segundos (1 *sync*). Foi usada uma limitação da velocidade a 0,7 m/s, pois como podemos ver na experiência 1 é a velocidade máxima que o agente conseguiu atingir quando tinha sucesso em descrever a trajetória pretendida. Assim, deste modo é o robô consegue atingir essa mesma velocidade máxima, mas não é forçado a ultrapassa-la conseguindo deste modo atingir percentagens de sucesso na ordem dos 85% parecidos às percentagens obtidas na experiência 2, mas com tempos de reação muito melhores.

A diferença entre usar um tempo de reação de 0,1 segundos (5 *syncs*) ou de 0,02 segundos (1 *sync*) está no tempo necessário de treino. A tabela 5.6 mostra que usar um tempo de reação de 0,02 segundos em vez de 0,1 aumenta o tempo necessário de treino, pois a sua dificuldade também aumenta. No caso da limitação da velocidade de 0,5 m/s o treino é bastante mais rápido, pois a esta velocidade o robô raramente cai o que diminui a dificuldade do treino.

timesteps Treino.	Inter. vX	Inter. vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
2M	[0:0,5]	[-1:1]	5	100	98	2
0,9M	[0:0,7]	[-1:1]	5	100	85	15
5,6M	[0:0,5]	[-1:1]	1	100	99	1
2,25M	[0:0,7]	[-1:1]	1	100	84	16

Table 5.6: Comparação dos tempos de treino

5.3 Planeamento de caminho com obstáculo

5.3.1 Experiência 4

Esta experiência é pela primeira vez colocado um obstáculo entre o agente e a posição final pretendida.

5.3.1.1 Action Space

O *action space* é o mesmo que nas últimas experiências sendo ele o seguinte.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.3.1.2 *Observation Space*

No caso do *observation space* não será o mesmo que nas experiências anteriores, pois existe desta vez um obstáculo e o agente terá que ter informação do mesmo para o poder evitar.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final
- **Dist_obst** — distância para o obstáculo
- **Ori_obst** — Orientação relativa do agente ao obstáculo

5.3.1.3 Episódio

Sendo esta a primeira experiência com um obstáculo o episódio será o mais simples possível devido ao tempo que dos treinos. Assim como na experiência 1 o agente começa na posição inicial (-5, 0) e o *target* final na posição (0,0) e o obstáculo, com um raio de 0,4 metros, é posicionado a meio entre a posição inicial do agente o *target* final como pode observar na imagem seguinte.

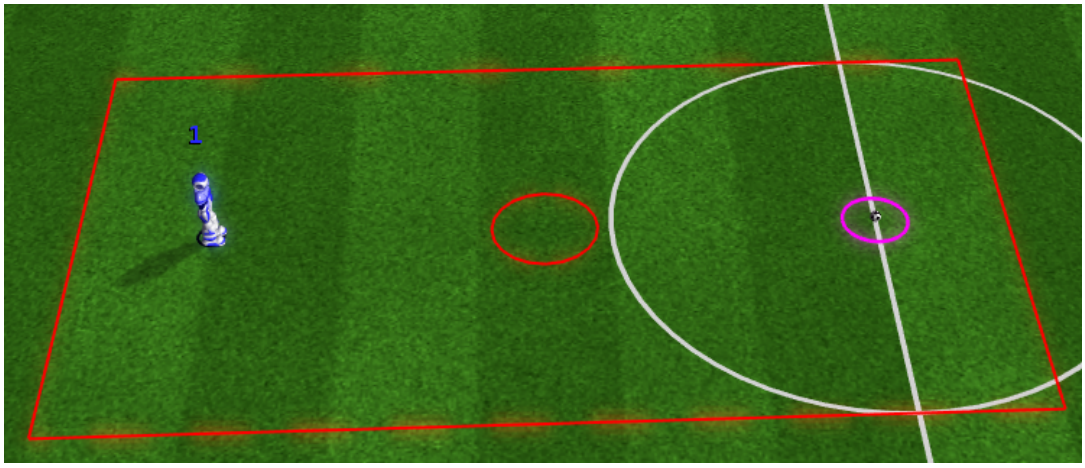

Figure 5.7: Situação inicial do episódio com obstáculo, experiência 4

As condições para o episódio acabar são as seguintes:

- A posição do agente não sair fora da área que pode ser vista na figura 5.1. Isto é, o valor da coordenada x da posição do agente não ser menor que -6 e maior que 1 e o valor da coordenada y não ser menor que -2 e maior que 2.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25 metros, pois a partir deste valor o agente cai.
- A distância entre o robô e a posição final ser menor que 0,25 metros
- Duração do episódio ser menor que 3000 *timesteps*
- A distância entre o robô e a obstáculo ser menor que 0,4 metros

5.3.1.4 Recompensa

A recompensa é divida em duas partes, a primeira relaciona o agente a posição final pretendida como nas experiências anteriores, mas desta vez multiplicada por uma constante, pois foi possível verificar uma melhoria de resultados na experiência 3 com esta nova função recompensa. Deste modo temos:

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = \frac{20}{syncs} * [D(t-1) - D(t)]^2$$

Em que, R é a recompensa, D a distância para a posição final pretendida, t é o *timestep* e os *syncs* é o número de *syncs* em que não se muda os valores das velocidades Vx e Vtheta.

A segunda parte da recompensa relaciona o agente e o obstáculo. Para tal a solução proposta é o uso de funções do tipo sigmoide em que um dos ramos tende para zero e o outro para um valor na mesma ordem de grandeza que a função recompensa entre o agente e a posição final pretendida. Deste modo temos a seguinte função:

$$R(t) = \frac{1}{200 * (e^{-70(D_{obst}(t)-0{,}55)} + 1)}$$

Em que, R é a recompensa, D_obst a distância para o obstáculo e t é o *timestep*

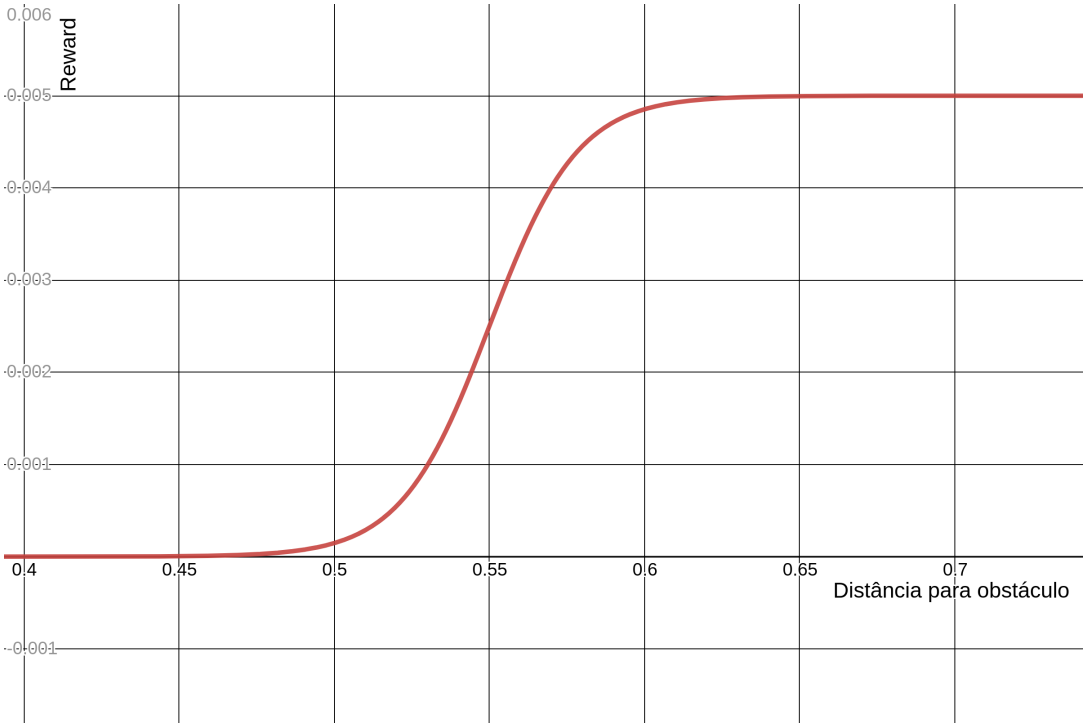

Figure 5.8: Função sigmoide

Como podemos ver pelo gráfico 5.8 o agente recebe uma recompensa de 0,005 se tiver afastado do obstáculo. Essa recompensa diminui se o agente se aproximar a acaba por atingir o valor de zero quando a distância para ao obstáculo é de 0,4575 deste modo evitando o agente colidir com o obstáculo.

A recompensa final será igual à soma entre as duas recompensas sendo ela a seguinte:

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = \frac{20}{syncs} * [D(t-1) - D(t)]^2 + \frac{1}{200 * (e^{-70(D_{obst}(t)-0{,}55)} + 1)}$$

5.3.1.5 Resultados

Sendo o problema mais simples envolvendo um obstáculo em que o robô só teria que aprender a andar em frente e desviar-se do obstáculo até atingir a posição final desejada é esperado que a percentagem de sucesso seja alta. Inicialmente foi usado um limite na velocidade Vx de 0,5 m/s com um tempo de reação de 0,1 segundos que correspondem a 5 *syncs*.

Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
[0:0,5]	[-1:1]	5	100	90	1

Table 5.7: Resultados de treino com obstáculo

Como podemos observar na tabela 5.7 a percentagem de sucesso ronda os 90%, seriam bons resultados, mas o comportamento descrito pelo agente não é o pretendido, pois o robô após evitar o obstáculo não se dirige ao encontro da posição final pretendida e apenas ficava a andar à volta dessa posição ate que eventualmente a atingia tendo por isso a percentagem de sucesso de 90%. O motivo de tal comportamento é a função da recompensa em relação ao obstáculo ter valores tão ou maiores que a recompensa em relação à posição final pretendida. Por este motivo foi altera da função recompensa em relação ao obstáculo e usada uma nova função sigmoide, sendo ela a seguinte:

$$R(t) = \frac{1}{1000 * (e^{-70(D_{obst}(t)-0{,}55)} + 1)}$$

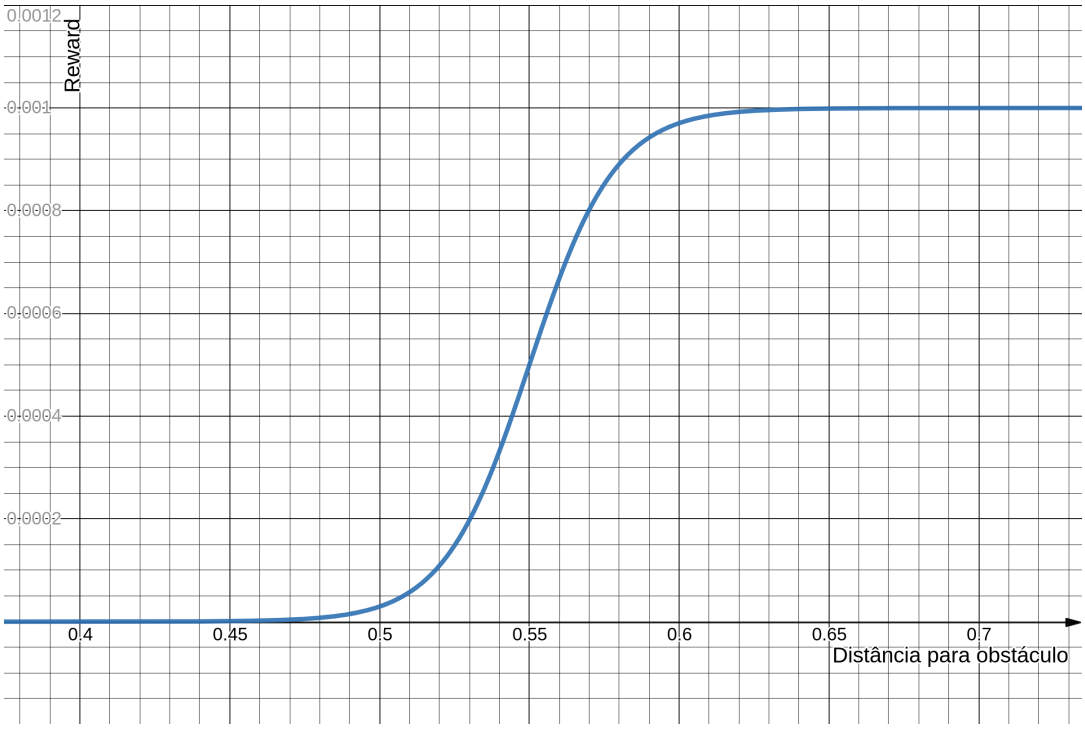

Figure 5.9: Função sigmoide corrigida

Como podemos ver na imagem 5.9 a única alteração é o valor para que tende a função, tendo ele diminuído de 0,005 para 0,001.

Com esta mudança os resultados melhoraram como podemos ver na tabela 5.8 e o comportamento do agente é o pretendido. Por diminui-se o tempo de reação para 0,02 segundos(1 *sync*) e aumentou-se a velocidade máxima para 0,7m/s

Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
[0:0,5]	[-1:1]	5	100	98	0
[0:0,7]	[-1:1]	1	100	98	1

Table 5.8: Resultados de treino com obstáculo, recompensa corrigida e aumento da velocidade

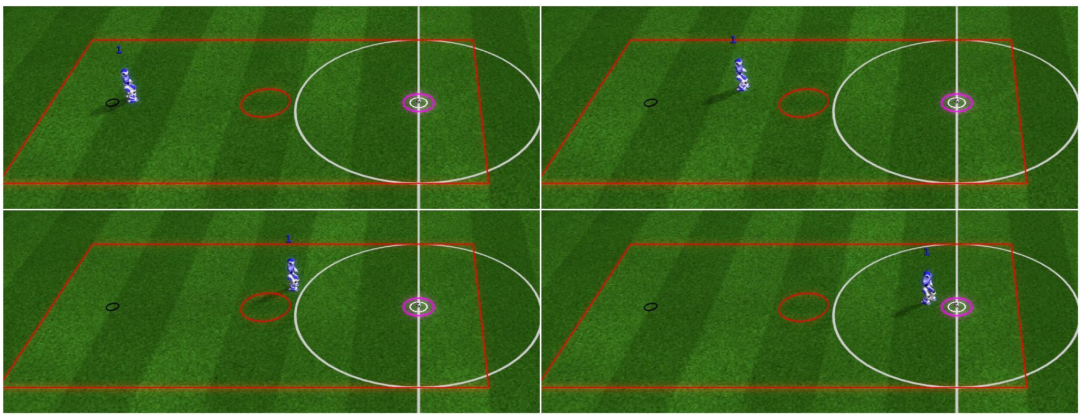

Figure 5.10: Exemplo do percurso do robô após treino com obstáculo na experiência 4

5.3.2 Experiência 5

Nesta experiência com obstáculo, a posição e a orientação inicial do robô irá variar.

5.3.2.1 Action Space

O *action space* é o mesmo que nas experiências anteriores.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.3.2.2 *Observation Space*

No caso do *observation space* será o mesmo que na experiência 4 que também envolve um obstáculo.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior

- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final
- **Dist_obst** — distância para o obstáculo
- **Ori_obst** — Orientação relativa do agente ao obstáculo

5.3.2.3 Episódio

Neste problema a posição final pretendida será novamente nas coordenadas (0,0) e o agente irá aparecer aleatoriamente à volta do *target* a uma distância que varia entre 3 metros e 5 metros, na figura 5.3 podemos constatar a área onde o robô pode ter posição inicial, sendo ela delimitada pelas duas linhas azuis claras. A orientação inicial do agente em relação à posição final pretendida será aleatória. O obstáculo ira estar sempre no ponto médio entre a posição inicial do agente e a posição final pretendida, tendo este um raio de 0,4 metros.

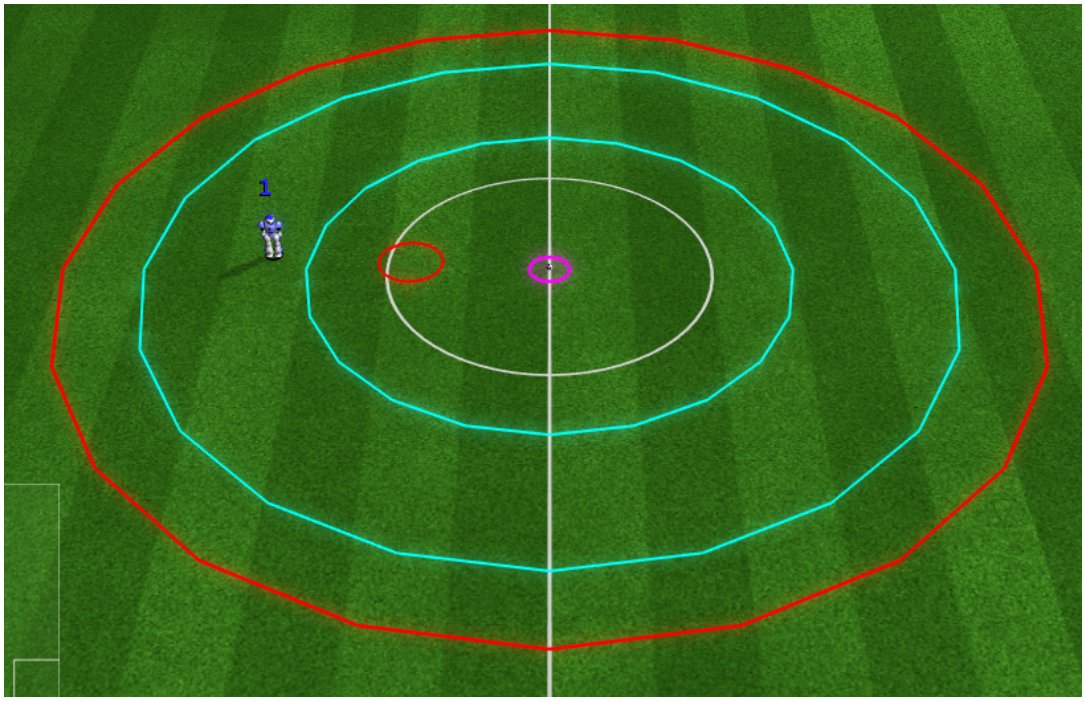

Figure 5.11: Situação inicial do episódio da experiência 5, posição inicial do agente variável e com obstáculo

As condições para o episódio acabar são as seguintes:

- A posição do agente não sair fora da área delimitada pela linha vermelha que pode ser vista na figura 5.3. Isto é, a distância do robô à posição final pretendida não poderá ser maior que 6 metros.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25 metros. Pois, a partir deste valor o agente cai.

- A distância entre o robô e a posição final ser menor que 0,25 metros

- Duração do episódio ser menor que 3000 *timesteps*

- A distância entre o robô e a obstáculo ser menor que 0,4 metros

5.3.2.4 Recompensa

A recompensa é igual à experiência anterior após a verificação de resultados muito positivos. Portanto, é a soma entre a recompensa relativa ao agente e a posição final e a recompensa, já melhorada na última experiência, relativa ao agente e ao obstáculo.

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = \frac{20}{syncs} * [D(t-1) - D(t)]^2 + \frac{1}{1000 * (e^{-70(D_{obst}(t)-0{,}55)} + 1)}$$

Em que, R é a recompensa, D a distância até à posição final pretendida e t o *timestep* e D_obst é a distância do agente ao obstáculo.

5.3.2.5 Resultados

Sendo uma versão mais complexa da experiência anterior e envolvendo um obstáculo foi inicialmente usado um limite na velocidade Vx de 0,5 m/s com um tempo de reação de 0,1 segundos que correspondem a 5 *syncs*. Depois foi diminuído o tempo de reação e por fim aumentada o limite de velocidade máxima para 0,7m/s, como podemos ver na tabela 5.9

Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
[0:0,5]	[-1:1]	5	100	98	0
[0:0,5]	[-1:1]	1	100	94	0
[0:0,7]	[-1:1]	1	100	82	7

Table 5.9: Resultados de treino com obstáculo, posição e orientação inicial do agente aleatórias

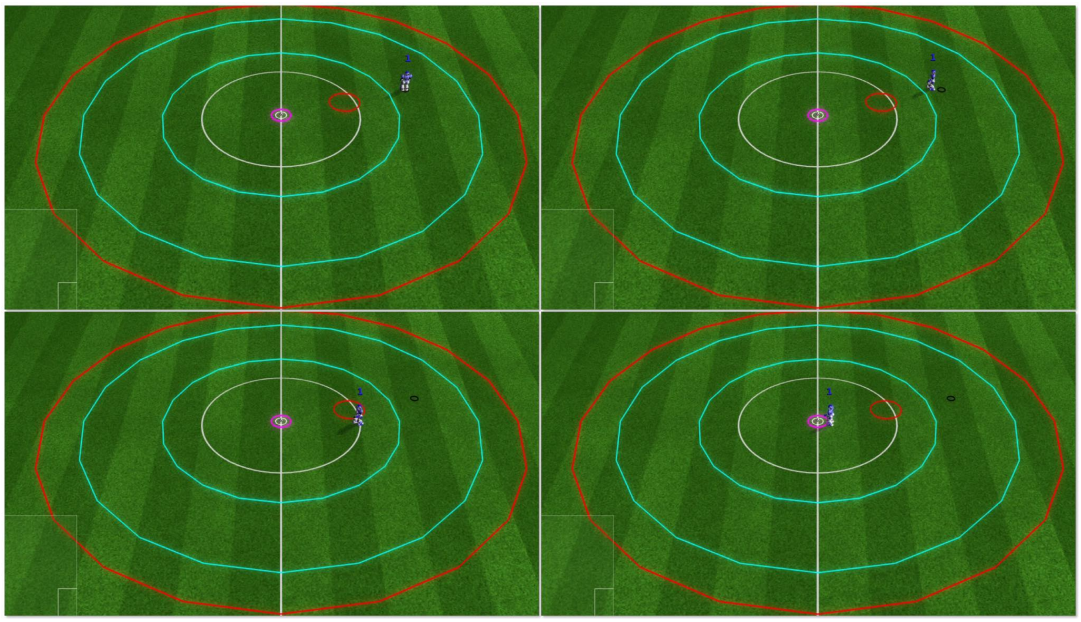

Figure 5.12: Exemplo do percurso do robô após treino na experiência 5 com obstáculo e posição inicial variável

5.3.3 Experiência 6

Nesta experiência o agente aprende a desviar-se de um obstáculo será dinâmico.

5.3.3.1 Action Space

O *action space* é o mesmo que nas últimas experiências sendo ele o seguinte.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.3.3.2 *Observation Space*

No caso do *observation space* será o mesmo que nas experiências 4 e 5 que também envolvem um obstáculo.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final
- **Dist_obst** — distância para o obstáculo
- **Ori_obst** — Orientação relativa do agente ao obstáculo

5.3.3.3 Episódio

Sendo esta a primeira experiência com um obstáculo o episódio dinâmico será o mais simples possível devido ao tempo que dos treinos. Assim como na experiência 4 o agente começa na posição inicial (-5, 0), o *target* final na posição (0,0) e o obstáculo, com um raio de 0,4 metros, é posicionado a meio entre a posição inicial do agente o *target* final como pode ver na imagem seguinte. A diferença para a experiência 4 é que o obstáculo irá se mover em direção a posição final pretendida pelo agente com uma velocidade de 0,1 m/s.

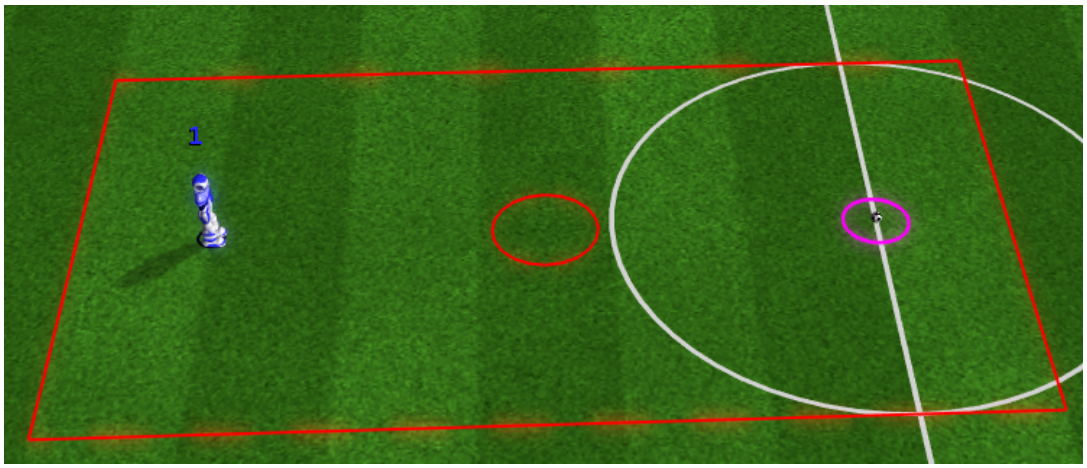

Figure 5.13: Situação inicial do episódio 6 com obstáculo dinâmico

As condições para o episódio acabar são as seguintes:

- A posição do agente não sair fora da área que pode ser vista na figura 5.1. Isto é, o valor da coordenada x da posição do agente não ser menor que -6 e maior que 1 e o valor da coordenada y não ser menor que -2 e maior que 2.
- O valor da coordenada z da cabeça do agente não ser menor que 0,25 metros, pois a partir deste valor o agente cai.
- A distância entre o robô e a posição final ser menor que 0,25 metros
- Duração do episódio ser menor que 3000 *timesteps*
- A distância entre o robô e a obstáculo ser menor que 0,4 metros

5.3.3.4 Recompensa

A recompensa é igual à experiência 5. Portanto, é a soma entre a recompensa relativa ao agente e a posição final e a recompensa, já melhorada na última experiência, relativa ao agente e ao obstáculo.

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = \frac{20}{syncs} * [D(t-1) - D(t)]^2 + \frac{1}{1000 * (e^{-70(D_{obst}(t)-0,55)} + 1)}$$

Em que, R é a recompensa, D a distância até à posição final pretendida e t o *timestep* e D_obst é a distância do agente ao obstáculo.

5.3.3.5 Resultados

Sendo uma versão mais complexa da experiência anterior e envolvendo um obstáculo dinâmico e com uma velocidade constante de 0,1 m/s foi inicialmente usado um limite na velocidade Vx de 0,5 m/s com um tempo de reação de 0,1 segundos que correspondem a 5 *syncs*. Depois foi aumentado a velocidade para 0,7m/s e por fim diminuído o tempo de reação para 0,02 segundos.

Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
[0:0,5]	[-1:1]	5	100	99	0
[0:0,7]	[-1:1]	5	100	98	1
[0:0,7]	[-1:1]	1	100	96	2

Table 5.10: Resultados de treino com obstáculo com velocidade constante de 0,1m/s

Figure 5.14: Exemplo do percurso do robô após treino na experiência 6 com obstáculo dinâmico

5.3.4 Experiência 7

Experiência com obstáculo dinâmico, a posição e a orientação inicial do robô irá variar.

5.3.4.1 Action Space

O *action space* é o mesmo que nas experiências anteriores.

- **Vx** velocidade do agente segundo o eixo das abcissas
- **Vtheta** velocidade angular do agente

5.3.4.2 *Observation Space*

No caso do *observation space* será o mesmo que na experiência com obstáculo.

- **Vx antigo** — valor da velocidade Vx anterior
- **Vtheta antigo** — valor da velocidade Vtheta anterior
- **Dist** — distância para o *target* final
- **Ori** — Orientação relativa do agente ao *target* final
- **Dist_obst** — distância para o obstáculo
- **Ori_obst** — Orientação relativa do agente ao obstáculo

5.3.4.3 Episódio

O episódio será igual à experiência 5 com posição inicial e orientação inicial do agente em relação à posição final pretendida aleatória. O obstáculo começa sempre no ponto médio entre a posição inicial do agente e a posição final pretendida, tendo este um raio de 0,4 metros, após o início do episódio o obstáculo irá mover-se em direção à posição final pretendida pelo agente com uma velocidade constante de 0,1m/s

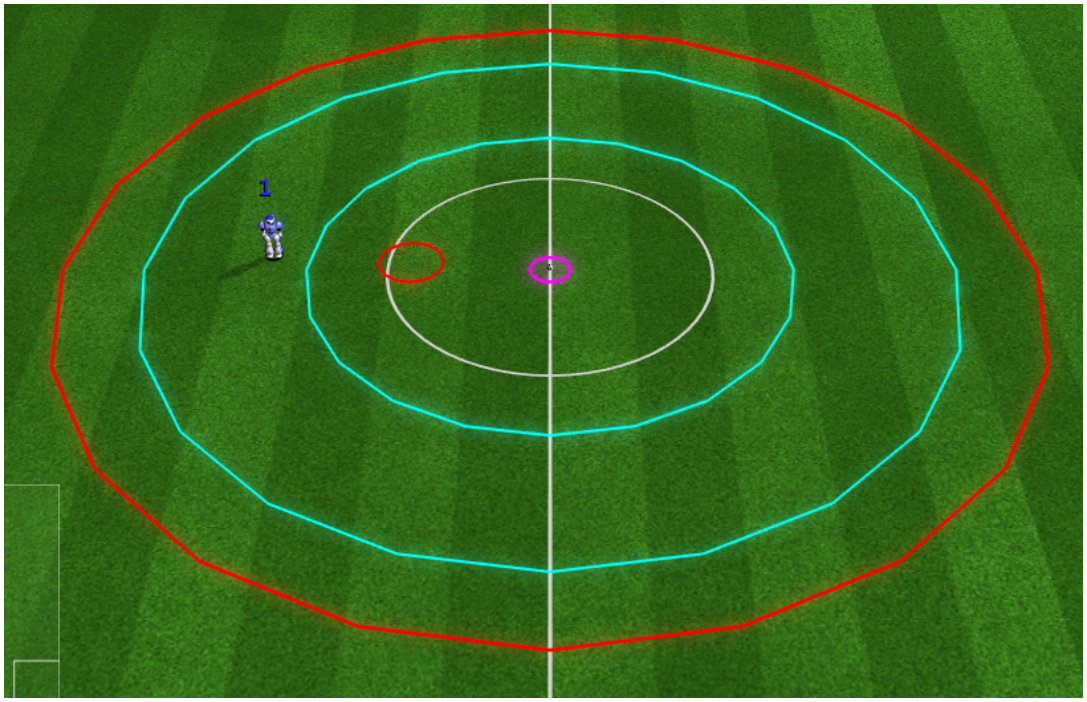

Figure 5.15: Situação inicial do episódio da experiência 7, posição inicial do agente variável e com obstáculo dinâmico

As condições para o episódio acabar são as seguintes:

- A posição do agente não sair da área delimitada pela linha vermelha que pode ser observada na figura 5.3. Isto é, a distância do robô à posição final pretendida não poderá ser maior que 6 metros.

- O valor da coordenada z da cabeça do agente não ser menor que 0,25 metros. Pois, a partir deste valor o agente cai.

- A distância entre o robô e a posição final ser menor que 0,25 metros

- Duração do episódio ser menor que 3000 *timesteps*

- A distância entre o robô e a obstáculo ser menor que 0,4 metros

5.3.4.4 Recompensa

A recompensa é igual às experiências 5 e 6. Portanto, é a soma entre a recompensa relativa ao agente e a posição final e a recompensa, já melhorada na última experiência, relativa ao agente e ao obstáculo.

$$se \quad D(t-1) - D(t) > 0:$$

$$R(t) = \frac{20}{syncs} * [D(t-1) - D(t)]^2 + \frac{1}{1000 * (e^{-70(D_{obst}(t)-0{,}55)} + 1)}$$

Em que, R é a recompensa, D a distância até à posição final pretendida e t o *timestep* e D_obst é a distância do agente ao obstáculo.

5.3.4.5 Resultados

Mais uma vez a abordagem passa por começar com um tempo de reação de 0,1 segundos (5 *syncs*) e a velocidade Vx limitada e 0,5 m/s. Após a verificação dos resultados é diminuído o tempo de reação e por fim aumentada a velocidade máxima para 0,7m/s, foram obtidos resultados que estão de acordo com os das experiências anteriores como pode ser visto na tabela seguinte.

Intervalo vX	Intervalo vTheta	Nº *Syncs*	Nº Ep.	Nº sucesso	Nº Cai
[0:0,5]	[-1:1]	5	100	98	0
[0:0,5]	[-1:1]	1	100	95	0
[0:0,7]	[-1:1]	1	100	80	15

Table 5.11: Resultados de treino com obstáculo dinâmico, posição e orientação inicial do agente aleatórias

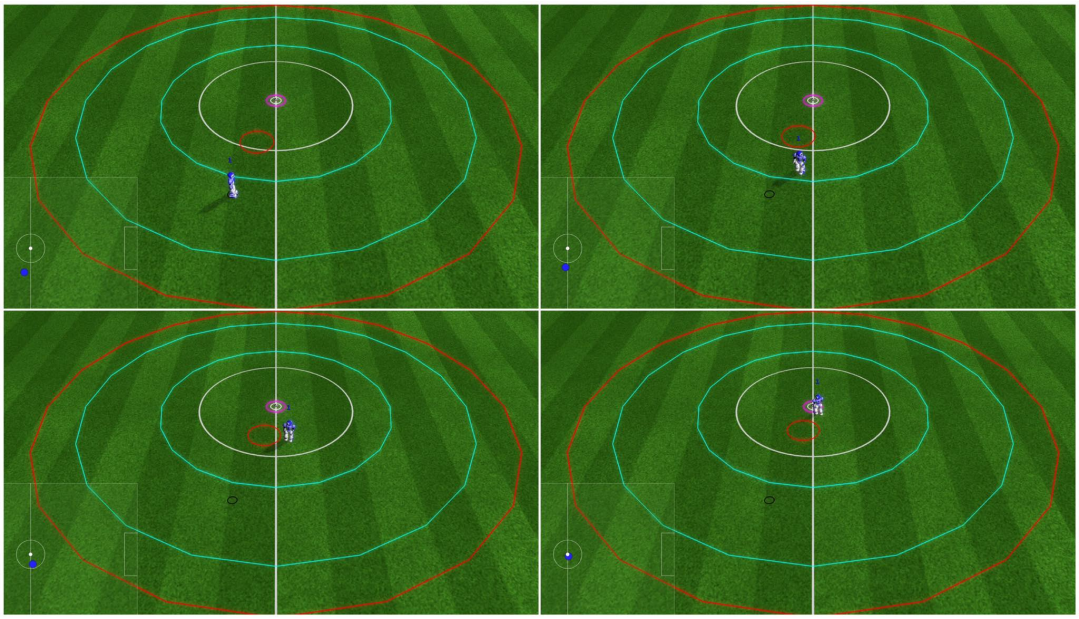

Figure 5.16: Exemplo do percurso do robô após treino na experiência 7, com obstáculo dinâmico e posição inicial aleatória

5.4 Conclusão

Este capitulo é sobre a parte prática da dissertação em que se focou em criar um plano de caminho para o agente com e sem obstáculo durante o percurso do mesmo. Deste modo o problema foi dividido em dois, o primeiro sem obstáculo e o segundo com, sendo que a segunda parte também foi dividida em duas, sendo a primeira com obstáculo estático e a segunda com obstáculo dinâmico. Tendo isto em conta é descrito como foi resolvido esses mesmo problemas desde da escolha do *action space*, à construção do episódio, à formulação da recompensa e por fim são comparados os resultados.

Chapter 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Inicialmente no primeiro capítulo é feita uma introdução geral a este documento o enquadramento desta dissertação bem como os objetivos e a motivação para a realização do mesmo. Uma das partes mais importante é o grande objetivo que a comunidade da RoboCup tem para o futuro. Este objetivo é ganhar à seleção campeã do mundial a meio do presente século. transmitindo uma esperança enorme pelo futuro da robótica.

O segundo capítulo cobre o estado de arte relacionado com *reinforcement learning* podendo verificar que ainda é um tema bastante recente e com muita margem para se desenvolver e melhorar sendo uma boa aposta para o futuro da robótica.

O ambiente da RoboCup usado nesta dissertação é descrito no capítulo três começando o simulador SimsSparK, o visualizador 3D Roboviz e por fiz a descrição do robô humanoide usado que é o robô NAO. No fim deste capítulo é abordado algum do papel do *reinforcement learning* na RoboCup focando-se principalmente do trabalho desenvolvido pela equipa FCPortugal, com esta descrição é percetível a importância do RL na equipa visto ter conseguido o desenvolvimento de comportamentos de andar e correr muito interessantes em relação às restantes equipas da Robocup.

No quarto capítulo é explicado as ferramentas de treino usadas durante a parte pratica sendo elas o FCPgym desenvolvido pela equipa FCPortugal tendo como base o *Open AI Gym*, a outra ferramenta descrita é o *Stable Baselines* que em conjunto com o FCPgym é usado para treinar usando algoritmos de RL. A ferramenta criada pela equipa facilita o trabalho para o desenvolvimento de novos comportamentos sendo por isso uma enorme ajuda para a equipa e para o desenvolvimento desta dissertação.

O capítulo 5 é sobre a parte prática da dissertação e a resolução do problema de planeamento de caminho. O problema foi dividido em dois sub-problemas sendo um deles sem obstáculos durante o percurso e o outro com, o segundo problema também foi dividido em duas partes sendo que na primeira é usado um obstáculo estático e na segunda um obstáculo dinâmico. No fim foram comparados os resultados. é possível verificar que as percentagens de sucesso são altas e o comportamento final do robô é o pretendido.

Os resultados desta dissertação podem ser considerados interessantes, mas ainda e estão prontos para serem usados durante uma partida da competição Robocup.

6.2 Trabalho Futuro

Em relação ao tema desta dissertação há várias coisas que pode ser feitas e melhoradas, pois apenas um algoritmo foi usado e os ambientes de treino podem ser mais complexos e com mais desafios para o agente ultrapassar. Com isto em mente sugiro algumas ideais para trabalhos futuros:

- A primeira sugestão é a mais óbvia que seria tentar usar outros algoritmos para haver mais resultados para comparar e tentar atingir resultados melhores do que aqueles que foram obtidos durante esta dissertação.
- Os ambientes usados apenas incluem um obstáculo seja ele estático ou dinâmico nas seria bom testar em outros ambientes em que houvessem mais obstáculos, alguns destes obstáculos sejam dinâmicos, pois durante um jogo um robô adversário é considerado um obstáculo e eles normalmente estão em movimento e são mais que um.
- Foi apenas usado o comportamento de andar desenvolvido pela equipa FCPortugal, seria interessante poder usar outros comportamentos como o correr, pois, apesar do agente atingir velocidades muito maiores terá maior dificuldade em virar e mudar de trajetória.

References

[1] OpenAI. Part 2: Kinds of RL algorithms — Spinning up documentation. *Openai.com*, page 38665, 2018. URL: https://spinningup.openai.com/en/latest/spinningup/rl{_}intro2.html.

[2] Lex Fridman. MIT 6.S091: Introduction to Deep Reinforcement Learning (Deep RL), 2019. URL: https://www.youtube.com/watch?v=zR11FLZ-O9M&list=PLrAXtmErZgOeiKm4sgNOknGvNjby9efdfhttps://www.youtube.com/watch?v=zR11FLZ-O9M [last accessed 2021-03-02].

[3] Zhengqiao Wang, Yufan Zeng, Yue Yuan, and Yibo Guo. Refining co-operative competition of robocup soccer with reinforcement learning. In *Proceedings - 2020 IEEE 5th International Conference on Data Science in Cyberspace, DSC 2020*, pages 279–283. Institute of Electrical and Electronics Engineers Inc., jul 2020. doi:10.1109/DSC50466.2020.00049.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. Technical report, 2017. arXiv:1707.06347.

[5] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. doi:10.1038/nature16961.

[6] Maziar Palhang, Nuno Lau, and David Simões. RoboCup Soccer Simulation 3D League, 2019. URL: https://ssim.robocup.org/3d-simulation/3d-rules/ [last accessed 2021-02-08].

[7] MagmaOffenburg. GitHub - Monitor and visualization tool for the RoboCup 3D Soccer Simulation League, 2019. URL: https://github.com/magmaOffenburg/RoboViz [last accessed 2021-02-10].

[8] Miguel Abreu, Luis Paulo Reis, and Nuno Lau. Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11531 LNAI, pages 3–15. Springer, jul 2019. URL: https://doi.org/10.1007/978-3-030-35699-6{_}1, doi:10.1007/978-3-030-35699-6_1.

[9] S.Glaser. Models · Wiki · RoboCup Simulation / SimSpark · GitLab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Models [last accessed 2021-02-07].

[10] Nima Shafii, Abbas Abdolmaleki, Rui Ferreira, Nuno Lau, and Luís Paulo Reis. Omnidirectional walking and active balance for soccer humanoid robot. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8154 LNAI:283–294, 2013. doi:10.1007/978-3-642-40669-0_25.

[11] Nima Shafii, Nuno Lau, and Luis Paulo Reis. Learning to Walk Fast: Optimized Hip Height Movement for Simulated and Real Humanoid Robots. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 80(3-4):555–571, feb 2015. URL: https://link.springer.com/article/10.1007/s10846-015-0191-5, doi:10.1007/s10846-015-0191-5.

[12] RL Algorithms — Stable Baselines 2.10.2 documentation. URL: https://stable-baselines.readthedocs.io/en/master/guide/algos.html [last accessed 2021-09-03].

[13] Vectorized Environments — Stable Baselines 2.10.2 documentation. URL: https://stable-baselines.readthedocs.io/en/master/guide/vec_envs.html.

[14] Tiago Silva. Humanoid low-level skills using machine learning for robocup. Technical report, Faculdade de Engenharia da Universidade do Porto, Jul 2019. URL: https://repositorio-aberto.up.pt/handle/10216/122505.

[15] Objective. URL: https://www.robocup.org/objective [last accessed 2021-02-13].

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, Fan Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676), oct 2017. URL: https://www.engineeringvillage.com/share/document.url?mid=inspec{_}683817db1624ed0a21bM6d3f1017816339{&}database=ins, doi:10.1038/nature24270.

[17] Josh Achiam. Part 1: Key Concepts in RL — Spinning Up documentation, 2019. URL: https://spinningup.openai.com/en/latest/spinningup/rl{_}intro.html [last accessed 2021-02-12].

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, dec 2017. URL: https://arxiv.org/abs/1712.01815v1, arXiv:1712.01815.

[19] Yuepeng Hu, Lehan Yang, and Yizhu Lou. Path Planning with Q-Learning. In *Journal of Physics: Conference Series*, volume 1948, page 12038. IOP Publishing, 2021. doi:10.1088/1742-6596/1948/1/012038.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. 2013. URL: http://arxiv.org/abs/1312.5602, arXiv:1312.5602.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig

Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015. URL: https://www.nature.com/articles/nature14236, doi:10.1038/nature14236.

[22] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 1889–1897, 2015. arXiv:1502.05477.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL: https://openai.com/blog/openai-baselines-ppo/ [last accessed 2021-02-13], arXiv:1707.06347.

[24] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *35th International Conference on Machine Learning, ICML 2018*, volume 4, pages 2587–2601, feb 2018. URL: http://arxiv.org/abs/1802.09477, arXiv:1802.09477.

[25] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *35th International Conference on Machine Learning, ICML 2018*, volume 5, pages 2976–2989, 2018. arXiv:1801.01290.

[26] D. C. J. de Jongh. World models. In *World Models*, volume 1, pages 85–86. mar 1978. URL: https://worldmodels.github.io/, doi:10.1007/bfb0007224.

[27] Stefan Glaser. SimSpark Wiki - Soccer Simulation, 2017. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Soccer-Simulation [last accessed 2021-02-07].

[28] S.Glaser. About SimSpark · Wiki · RoboCup Simulation / SimSpark · GitLab. URL: https://gitlab.com/robocup-sim/SimSpark/-/wikis/About-SimSpark [last accessed 2021-02-07].

[29] SoftBank Robotics. Pepper the humanoid and programmable robot | SoftBank Robotics, 2020. URL: https://www.softbankrobotics.com/emea/en/naohttps://www.softbankrobotics.com/emea/en/pepper [last accessed 2021-02-07].

[30] Henrique Teixeira, Tiago Silva, Miguel Abreu, and Luis Paulo Reis. Humanoid robot kick in motion ability for playing robotic soccer. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*, pages 34–39. Institute of Electrical and Electronics Engineers Inc., apr 2020. doi:10.1109/ICARSC49921.2020.9096073.

[31] Aijun Bai, Stuart Russell, and Xiaoping Chen. Concurrent hierarchical reinforcement learning for robocup keepaway. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11175 LNAI, pages 190–203. Springer Verlag, jul 2018. URL: https://doi.org/10.1007/978-3-030-00308-1{_}16, doi:10.1007/978-3-030-00308-1_16.

[32] Sha Luo, Weijia Yao, Qinghua Yu, Junhao Xiao, Huimin Lu, and Zongtan Zhou. Object detection based on GPU parallel computing for RoboCup Middle Size League. In *2017*

IEEE International Conference on Robotics and Biomimetics, ROBIO 2017, volume 2018-Janua, pages 86–91. Institute of Electrical and Electronics Engineers Inc., mar 2018. doi:10.1109/ROBIO.2017.8324399.

[33] Jacob Menashe, Josh Kelle, Katie Genter, Josiah Hanna, Elad Liebman, Sanmit Narvekar, Ruohan Zhang, and Peter Stone. Fast and precise black and white ball detection for robocup soccer. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11175 LNAI, pages 45–58. Springer Verlag, jul 2018. URL: https://doi.org/10.1007/978-3-030-00308-1{_}4, doi:10.1007/978-3-030-00308-1_4.

[34] David Simões, Nuno Lau, and Luís Paulo Reis. Multi-agent double deep Q-networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10423 LNAI, pages 123–134. Springer Verlag, 2017. doi:10.1007/978-3-319-65340-2_11.

[35] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016. URL: http://arxiv.org/abs/1606.01540, arXiv:1606.01540.

[36] Openai Baselines. Welcome to Stable Baselines docs ! - RL Baselines Made Easy Main differences with OpenAI Baselines. URL: https://stable-baselines.readthedocs.io/en/master/ [last accessed 2021-09-03].

[37] Gym. URL: https://gym.openai.com/envs/#classic_control [last accessed 2021-09-03].

[38] Gym. URL: https://gym.openai.com/envs/#classic_control [last accessed 2021-09-03].

[39] Miguel Abreu, Tiago Silva, Henrique Teixeira, Luís Paulo Reis, and Nuno Lau. 6D Localization and Kicking for Humanoid Robotic Soccer. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 102(2), 2021. URL: https://doi.org/10.1007/s10846-021-01385-3, doi:10.1007/s10846-021-01385-3.

[40] PPO2 — Stable Baselines 2.10.2 documentation. URL: https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html.

[41] Policy Networks — Stable Baselines 2.10.2 documentation. URL: https://stable-baselines.readthedocs.io/en/master/modules/policies.html.