UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

SAMUEL DE MAGALHÃES PEDROZA

UTILIZAÇÃO DE UMA REDE NEURAL ARTIFICIAL NO CONTROLE E TOMADA DE DECISÕES DE FUTEBOL DE ROBÔS DA CATEGORIA VSSS

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2023

SAMUEL DE MAGALHÃES PEDROZA

UTILIZAÇÃO DE UMA REDE NEURAL ARTIFICIAL NO CONTROLE E TOMADA DE DECISÕES DE FUTEBOL DE ROBÔS DA CATEGORIA VSSS

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Prof. Jadson Castro Gertrudes, Dr.

Coorientador: Prof. Alan Kardek Rego Segundo, Dr.

Ouro Preto
Escola de Minas – UFOP
2023

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Samuel de Magalhães Pedroza

Utilização de uma Rede Neural Artificial no Controle e Tomada de Decisões de Futebol de Robôs da Categoria VSSS

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 28 de março de 2023

Membros da banca

Prof. Dr. Jadson Castro Gertrudes – Orientador (Universidade Federal de Ouro Preto)
Prof. Dr. Alan Kardek Rêgo Segundo – Coorientador (Universidade Federal de Ouro Preto)
Profa. Dra. Karla Boaventura Pimenta Palmieri – Convidada (Universidade Federal de Ouro Preto)
Prof. Dr. Vander Luís de Souza Freitas – Convidado (Universidade Federal de Ouro Preto)

Alan Kardek Rêgo Segundo, coorientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 11/04/2023

Documento assinado eletronicamente por **Alan Kardek Rego Segundo**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 11/04/2023, às 14:46, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0506960** e o código CRC **16B53658**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

AGRADECIMENTOS

Agradeço primeiramente aos meus pais e minha família, que me deram toda a base e apoio para chegar onde cheguei. Agradeço a todos os amigos que fizeram parte da minha caminhada durante todo o período da graduação. Um agradecimento especial para a minha amiga Júlia, que sempre esteve ao meu lado em todos os momentos, independente das dificuldades e das ocasiões, além de todo apoio no desenvolvimento deste trabalho. Também agradeço ao professor Jadson por todo apoio desde o começo do desenvolvimento do TCC, ao professor Alan Kardek por todas oportunidades de aprendizado e crescimento concedidas durante toda a graduação, além de agradecer a todos os integrantes da equipe Rodetas que contribuiram para o desenvolvimento do projeto.

“Não podemos esperar construir o mundo melhor sem melhorar as pessoas. Cada um de nós deve trabalhar para o nosso próprio aprimoramento.” (Marie Curie)

RESUMO

Durante os últimos anos, a robótica vem sendo uma área em grande expansão, seja facilitando o desenvolvimento de tarefas pesadas, ou até mesmo na diminuição de tarefas repetitivas. Entretanto, a robótica não se limita somente à este tipo de tarefas, tendo uma gama muito maior de atuação. Neste trabalho é apresentada a implementação e uso de uma rede neural artificial em jogos de futebol de robôs da modalidade *IEEE Very Small Size Soccer*. A rede neural utilizada é baseada no algoritmo genético para neuroevolução de topologias, NEAT, com o objetivo de trazer uma nova abordagem para o time de robôs da equipe Rodetas Robo Clube. Ao utilizar esta tecnologia, o processo de tomada de decisões deixa de ser limitado pelos cenários pré-definidos pela equipe, passando a ter ações baseadas na percepção de jogo do próprio algoritmo, o que permite explorar cenários que de outra forma não seriam utilizados. Os resultados obtidos sugerem que o NEAT pode resolver tarefas simples na categoria, como seguir a bola, mas não consegue resolver tarefas complexas, como controlar todos os robôs de uma equipe em um jogo real.

Palavras-chaves: Inteligência Artificial. Redes Neurais. Futebol de Robôs

ABSTRACT

During the last few years, robotics has been an area in great expansion, either facilitating the development of heavy tasks, or even reducing repetitive tasks. However, robotics is not limited only to this type of tasks, having a much wider range of activities. This work describes the implementation and use of an artificial neural network in robot soccer matches of the *IEEE Very Small Size Soccer* category. The neural network used is based on the genetic algorithm for topology neuroevolution, NEAT, with the objective of bringing a new approach to the robot team of the Rodetas Robô Clube. By using this technology, the decision-making process is no longer limited by predefined scenarios, but rather based on the algorithm's own perception of the game, allowing exploration of scenarios that would otherwise not be used. The results obtained suggest that NEAT can solve simple tasks in the category, such as following the ball, but cannot solve complex tasks, such as controlling all robots on a team in a real match.

Key-words: Artificial Inteligence. Neural Networks. Robots Soccer

LISTA DE ILUSTRAÇÕES

Figura 1 – Campo com todas as dimensões da categoria VSSS. Fonte: (PINTO, 2020) . 14
Figura 2 – Exemplo da regra referente ao chanfro do robô. Fonte: (PINTO, 2017). . . . 15
Figura 3 – Modelagem 3D e imagem do robô da equipe Rodetas. Fonte: Arquivo Rodetas. 15
Figura 4 – Infrações da competição VSSS. Fonte: (PINTO, 2017). 16
Figura 5 – Estrutura da competição. Fonte: (PINTO, 2017). 17
Figura 6 – VSS-VIEWER. Fonte: Elaborado pelo autor. 18
Figura 7 – *Software* FIRASim. Fonte: Elaborado pelo autor. 20
Figura 8 – Modelo neuronal. Fonte: Elaborado pelo autor. 23
Figura 9 – Exemplos de ondas do algoritmo WWO. Fonte: Zhou et al. (2019). 24
Figura 10 – Operação de crossover. Adaptado de Kramer (2017). 25
Figura 11 – Estrutura básica de um indivíduo. Fonte: Adaptado de Stanley e Miikkulainen (2002). 27
Figura 12 – Operação de mutação por adição de nó. Fonte: Elaborado pelo Autor. 28
Figura 13 – Operação de mutação por adição de conexão. Fonte: Elaborado pelo Autor. . 28
Figura 14 – Operação de crossover. Fonte: Adaptado de Stanley e Miikkulainen (2002). . 29
Figura 15 – Relação para determinar movimentação do robô em direção a bola. Fonte: Elaborado pelo Autor. 38
Figura 16 – Resultados do treinamento de rede neural para seguir a bola. Fonte: Elaborado pelo autor. 41
Figura 17 – Resultados do treinamento de Martins et al. (2021) para que o robô siga a bola (DDPG em azul e SAC em laranja). O eixo X representa o número de passos e o eixo Y representa a *fitness*. 41
Figura 18 – Primeiro resultado de treinamento da rede Neural. Fonte: Elaborado pelo autor. 42
Figura 19 – Treinamento de rede neural com reinício por estagnação. Fonte: Elaborado pelo autor. 43
Figura 20 – Treinamento de rede neural com punição por ficar parado. Fonte: Elaborado pelo autor. 44
Figura 21 – Treinamento de rede neural com punição por ficar parado e executar círculos. Fonte: Elaborado pelo autor. 44
Figura 22 – Treinamento de rede neural sem estagnação após 60 gerações. Fonte: Elaborado pelo autor. 45
Figura 23 – Treinamento de rede neural sem estagnação após 300 gerações. Fonte: Elaborado pelo autor. 46

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivos gerais e específicos 13
1.2 Estrutura do trabalho 13

2 REVISÃO DA LITERATURA 14
2.1 O Ambiente da categoria *Very Small Size Soccer* 14
2.1.1 O campo 14
2.1.2 A partida 15
2.1.3 A estrutura física de uma partida 16
2.2 Estrutura atual da equipe Rodetas 17
2.3 Ferramentas utilizadas 19
2.3.1 RSoccer 20
2.4 Inteligência Artificial 21
2.4.1 Definição 21
2.4.2 Redes Neurais Artificiais 22
2.4.3 Aprendizado por reforço 23
2.5 Otimização e evolução de redes neurais 24
2.6 Algoritmos genéticos 25
2.6.1 Crossover 25
2.6.2 Mutação 25
2.6.3 Funcionamento geral de um algoritmo genético 26
2.7 Funcionamento do NEAT 26
2.7.1 Codificação genética 26
2.7.1.1 Operações genéticas 27
2.7.2 Acompanhamento de genes através de histórico de marcas 27
2.7.3 Garantia da diversidade por meio de especiação 29
2.7.4 Redução das dimensões da rede 30

3 DESENVOLVIMENTO 31
3.1 Tecnologias 31
3.2 Ferramentas 31
3.3 Problemas e dificuldades do sistema atual da equipe Rodetas 31
3.4 Abordagem Proposta 32
3.5 Configuração do NEAT 33
3.6 Treinamento da rede neural 35
3.6.1 Treinamento seguir a bola 36

3.6.2 *Treinamento da partida completa* *37*
3.6.2.1 Problemas identificados 38
3.6.2.2 Melhorias implementadas 38
3.6.2.3 Novos problemas 39

4 RESULTADOS 40
4.1 Treinamento para seguir a bola 40
4.2 Treinamento de partida completa 41
4.2.1 *Resultados de configuração inicial* *41*
4.2.2 *Resultados após alteração de* fitness *43*
4.2.3 *Resultados removendo estagnação* *45*

5 CONCLUSÃO 47
5.1 Conclusões 47
5.2 Trabalhos Futuros 47

REFERÊNCIAS 48

1 INTRODUÇÃO

Nos últimos anos, a robótica vem sendo uma grande aliada dos seres humanos, seja para facilitar o desenvolvimento de tarefas pesadas, ou até mesmo diminuir o esforço em tarefas repetitivas. Porém, nos estudos mais recentes, a robótica não vem se limitando somente ao desenvolvimento de tarefas pesadas. Há aplicações nas mais diversas áreas, como o estudo de robôs que simulam comportamentos humanos (HADDADIN; ALBU-SCHAFFER; HIRZINGER, 2008).

Construir robôs capazes de disputar uma partida de futebol, tomar decisões em tempo real e ter uma inteligência artificial requer uma pesquisa multidisciplinar que combina percepção, tomada de decisão dinâmica e interação entre o sistema robótico e o subsistema de controle cinemático. Por trás de uma partida de futebol de robôs ocorre um algoritmo complexo que envolve processamento de imagens em tempo real, cálculos de movimentos dos robôs, comunicação sem-fio e gerenciamento de múltiplos robôs, além de ser necessário projetar um sistema eletrônico robusto e capaz de executar todos os comandos com velocidade e precisão.

Neste contexto, o Rodetas Robô Clube é um projeto de extensão fundado em 2011, pela professora Karla Boaventura Pimenta Palmieri, na Universidade Federal de Ouro Preto com o objetivo de montar uma equipe multidisciplinar capaz de participar de competições de futebol de robôs da categoria *IEEE Very Small Size Soccer*.

A categoria *IEEE Very Small Size Soccer* (VSSS) consiste no desenvolvimento de robôs autônomos capazes de disputar partidas de futebol, controlados remotamente por um computador ou utilizando um processamento embarcado, como pode ser visto em toda a descrição apresentada no texto de Pinto (2020). Dentro de uma partida diversos desafios são apresentados, o que traz uma grande complexidade aos trabalhos desenvolvidos. Seguindo os padrões utilizados pela equipe. Pode-se dividir o processo realizado por uma equipe em 5 etapas: visão computacional, tomada de decisões (*Decision Making*), planejamento de caminhos (*Path Planning*), controle e transmissão.

A etapa de visão computacional consiste no sistema responsável pelo reconhecimento dos robôs em campo, por meio de uma câmera, trazendo para dentro de um *software* as informações coletadas em campo, tais quais velocidade de cada robô, posição em campo, além da posição e velocidade da bola. Para tal, utiliza-se o FIRASim[1], um simulador virtual de partidas do VSSS, que faz todo processamento de dados e, através do protocolo de comunicação do Google, o Protobuf, transmite as informações para o código utilizado por cada equipe.

É neste ponto que o RSoccer, um programa projetado para execução de treinamentos voltados para aprendizado de máquina em algumas categorias de futebol de robôs também toma

[1] Disponível em: https://github.com/robocin/FIRASim

espaço, uma vez que as partidas oficiais acontecem obtendo informações vindas do FiraSIM, o mesmo não permite uma execução de múltiplas partidas de forma simultânea e/ou com critérios de parada pré-definidos. Com isso, durante os treinamentos das redes neurais o RSoccer efetua o papel que o FIRASim executa em partidas oficiais, sendo a fonte de informações que simulam partidas reais, permitindo assim um maior volume de partidas em ambientes melhor controlados.

Com as informações recebidas do FIRASim, ou do RSoccer durante os treinamentos, se inicia a fase de tomada de decisões, onde cada equipe trata as informações de entrada da sua forma, com o fim de identificar cenários em campo e estabelecer as melhores estratégias para atingir o objetivo final da partida que é marcar o maior número de gols, sofrendo o mínimo de gols possíveis.

Em seguida, com as estratégias já definidas, faz-se necessário utilizar um algoritmo que trace as rotas (ou caminhos), já que não é necessário apenas definir o ponto a ser atingido, mas também a forma como o robô chega até ele. Com isso, algoritmos de busca em espaço, ou até mesmo algoritmos genéticos são utilizados a fim de traçar a melhor rota entre o robô e seu objetivo.

Na categoria VSSS a movimentação dos robôs no campo é um dos fatores mais importantes para o bom desempenho do time. Um ponto importante da movimentação é o planejamento da trajetória que o robô deve executar nas diferentes situações de jogo. Para este planejamento, o algoritmo deve considerar os outros robôs e abstraí-los como obstáculos, para que se possa chegar ao objetivo final sem colidir com os outros robôs (tanto adversários como a própria equipe).

A partir daí, em posse de todas informações necessárias, o algoritmo de controle dos veículos é iniciado, transformando os cálculos realizados pela estratégia e a rota a ser traçada pelos robôs definida pelo planejamento de caminhos, em valores de modulação por largura de pulso, que são enviados para as rodas dos robôs. Após todo o processo, o módulo de transmissão é acionado, enviando para os robôs com o objetivo de mover os robôs para o destino esperado.

Neste cenário, a utilização de novas tecnologias é crescente e alternativas aos métodos convencionais são introduzidas a fim de otimizar o desempenho das equipes. Assim, a inteligência artificial surge como uma opção que vem ganhando cada vez mais espaço, tendo em vista todo o investimento do cenário em softwares como o próprio RSoccer, disponibilizando cada vez mais ferramentas para o desenvolvimento nesta área, onde os robôs tendem a calcular jogadas e movimentações da forma mais autônoma possível e com uma grande capacidade de analisar cenários.

Uma alternativa no campo da inteligência artificial é a utilização de Redes Neurais. Segundo Khosravikia et al. (2018), as redes neurais artificiais (RNAs) baseiam-se no comportamento dos neurônios orgânicos presentes nos seres vivos. Uma forma especial para a construção de uma rede Neural é por meio do uso de algoritmos genéticos. Neste cenário, o algoritmo NEAT

(*Neuroevolution of Augmenting Topologies*) (STANLEY; MIIKKULAINEN, 2002) é um bom candidato. Ele permite a construção de uma arquitetura de rede neural ótima para o problema sob estudo, além de utilizar de funções objetivos para definir a melhor estrutura da rede proposta.

Com a utilização de RNAs, pode-se adquirir, de forma automática, decisões para movimentação dos robôs do time VSSS da equipe do Rodetas Robô Clube[2], dando um passo importante no desenvolvimento tecnológico do time, deixando um trabalho que possa ser objeto de estudo e base de conquistas para as próximas gerações de membros da equipe.

1.1 Objetivos gerais e específicos

O presente trabalho visa a implementação de redes neurais como solução para o problema de planejamento de caminhos dentro da equipe Rodetas Robô Clube na categoria VSSS.

Para alcançar nosso objetivo principal, foram estabelecidos os seguintes objetivos específicos:

- Recriar o código da equipe Rodetas utilizando da linguagem de programação Python;
- Avaliar o desempenho do algoritmo NEAT na integração de redes neurais artificiais para solucionar o problema de estratégia e controle na categoria VSSS;
- Integrar o algoritmo NEAT com RSoccer para disponibilizar um ambiente de treinamento de redes neurais para trabalhos futuros da equipe.

1.2 Estrutura do trabalho

O restante do trabalho está organizado na seguinte forma: O Capítulo 2 apresenta os conceitos básicos para compreensão do trabalho, além de trabalhos relacionados à presente monografia. No Capítulo 3 são apresentados o ambiente da categoria VSSS, assim como os *softwares*, tecnologias e metodologias utilizadas no desenvolvimento deste trabalho. O Capítulo 4 apresenta os resultados obtidos pelo experimento proposto. Por fim, no Capítulo 5 são apresentadas as conclusões e os trabalhos futuros.

[2] Endereço web: https://rodetas.ufop.br/

2 REVISÃO DA LITERATURA

Este capítulo traz a revisão da literatura e o embasamento teórico utilizado para execução deste trabalho.

2.1 O Ambiente da categoria *Very Small Size Soccer*

Para implementar um robô capaz de jogar uma partida da categoria Very Small Size Soccer é necessário ter atenção às diversas limitações do ambiente de jogo. A seguir são apresentadas as principais regras que regem a competição de acordo com as informações fornecidas na Copa Latino Americana de Robótica (LARC) 2020 presente no artigo de Pinto (2020).

2.1.1 O campo

O campo é o local onde as partidas da categoria VSSS são disputadas. Ele deve ser preto, não reflexível (para não influenciar nas imagens capturadas), de madeira e deve ter 1,50m de comprimento e 1,30m de largura. Além disso, o campo deve ter um gol em cada extremidade, com 40cm de comprimento e 10cm de profundidade.

No campo também existem marcações para a cobrança das diversas infrações presentes no jogo, como o *Free Ball* (FB), *Free Kick* (FK) e *Penalty* (PK). Todas as dimensões podem ser visualizadas na Figura 1, fornecida pela LARC no manual de regras da competição (PINTO, 2020). A bola utilizada na modalidade é uma bola de golf de cor laranja com 42,7mm de diâmetro e 46g de peso.

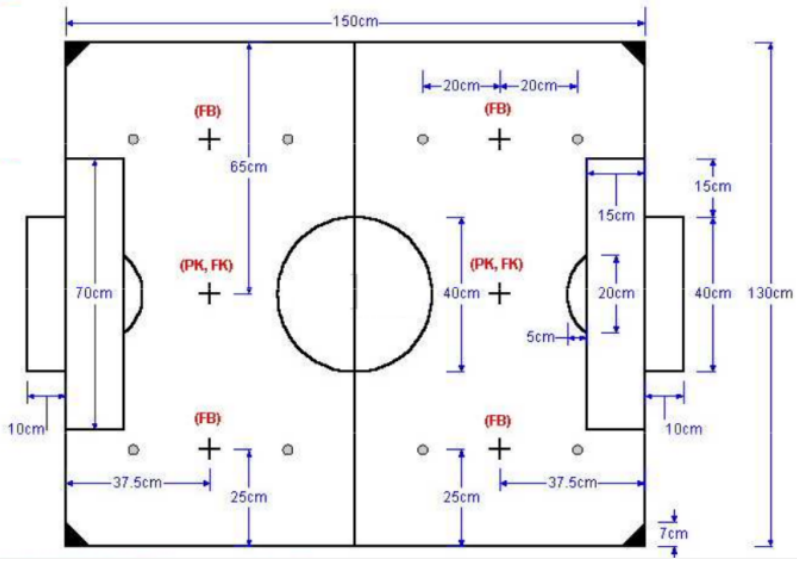

Figura 1 – Campo com todas as dimensões da categoria VSSS. Fonte: (PINTO, 2020)

.

As equipes são compostas por 3 robôs que, de acordo com Pinto (2017), cada robô pode dispor de braços, pernas ou dispositivos de chute desde que não ultrapasse o tamanho limitado de 75x75x75mm. Além disso, segundo a autora, é permitido ao robô possuir um chanfro que tenha o objetivo de conduzir a bola, desde que a profundidade final dele não compreenda mais que 30% do diâmetro da bola, como pode ser visto na Figura 2.

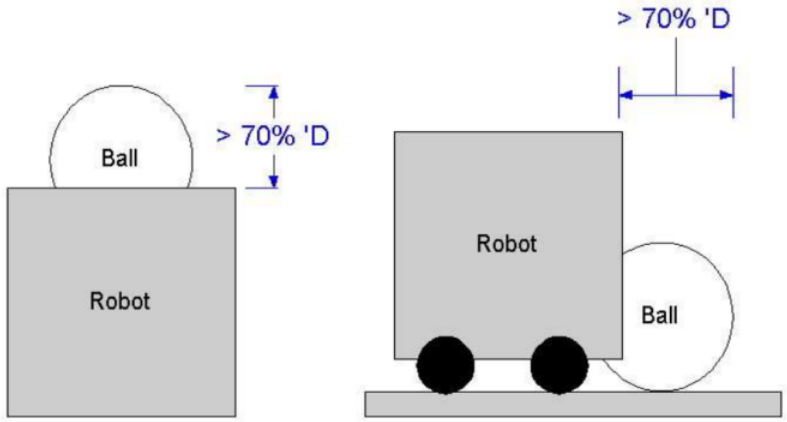

Figura 2 – Exemplo da regra referente ao chanfro do robô. Fonte: (PINTO, 2017).

Um exemplo prático é o robô da equipe Rodetas que pode ser visto na Figura 3.

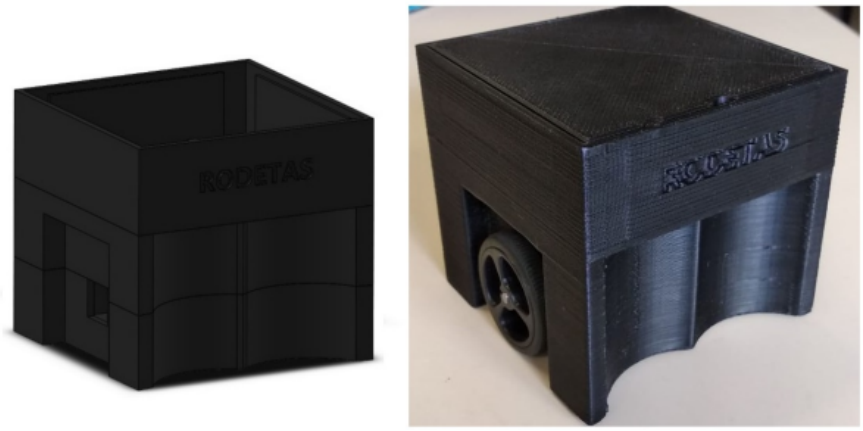

Figura 3 – Modelagem 3D e imagem do robô da equipe Rodetas. Fonte: Arquivo Rodetas.

Ademais, cada time deve possuir um identificador na parte superior de todos os robôs, que deve representar a cor do time (azul ou amarelo), a fim de diferenciá-los nas imagens capturadas em câmera.

2.1.2 A partida

Seguindo as regras da modalidade, cada jogo consiste de dois tempos de cinco minutos, com um intervalo de dez minutos entre eles. Cada equipe tem a possibilidade de solicitar dois

tempos de dois minutos cada um, os quais podem ser solicitados em qualquer momento durante a partida. Além disso, sempre que ocorre uma infração, o cronômetro da partida é pausado e reiniciado após a cobrança da falta.

De acordo com o estudo realizado por Pinto (2017), a competição VSSS estabelece as seguintes infrações: *Penalty-Kick* (PK), que ocorre quando a bola permanece dentro da área de defesa por mais de 10 segundos e o goleiro não consegue removê-la, quando a equipe defensora possui mais de um robô dentro da área do gol, ou quando um membro humano toca em um dos robôs sem a permissão do árbitro; *Free Kick* (FK), que acontece quando um robô empurra outro robô do time adversário; *Goal Kick* (GK), que é sinalizado quando o robô atacante empurra o goleiro na área do time adversário, mais de um atacante entra na área do time adversário simultaneamente, um robô oponente bloqueia o caminho do goleiro na área do gol, ou quando há um impasse na área do gol por mais de 10 segundos; e, por fim, a infração de *Ball Holding* (BH), que ocorre quando a bola fica parada fora das áreas do gol por mais de 10 segundos. A Figura 4 apresenta cada uma dessas infrações de acordo com as regras da competição VSSS.

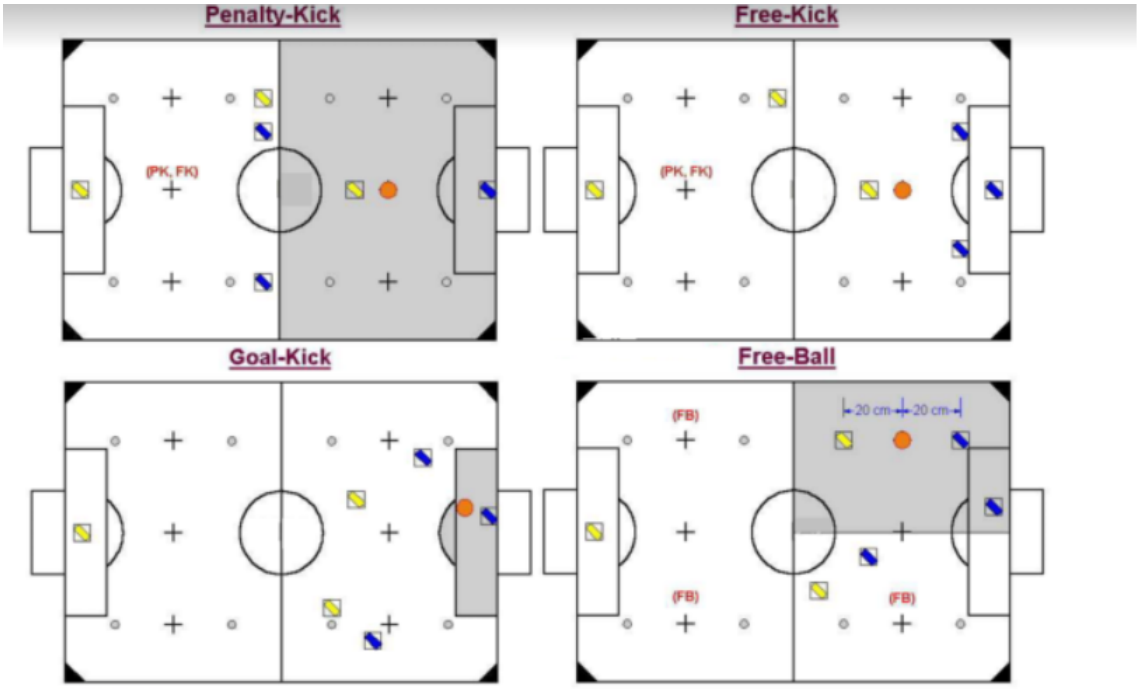

Figura 4 – Infrações da competição VSSS. Fonte: (PINTO, 2017).

2.1.3 A estrutura física de uma partida

A estrutura de jogo é formada pelo campo, onde os jogadores e a bola são posicionados, duas câmeras posicionadas acima do campo (uma para cada equipe) e um computador responsável por receber as imagens registradas pela câmera, processá-las e enviar instruções para os robôs de acordo com as informações recebidas. A distribuição pode ser vista na Figura 5.

Por se tratar de um ambiente muito dinâmico, a velocidade de processamento e a

eficiência da programação de cada equipe é fator determinante no desempenho na modalidade, onde um milésimo de segundo pode ser o suficiente para alterar todo o curso de uma partida.

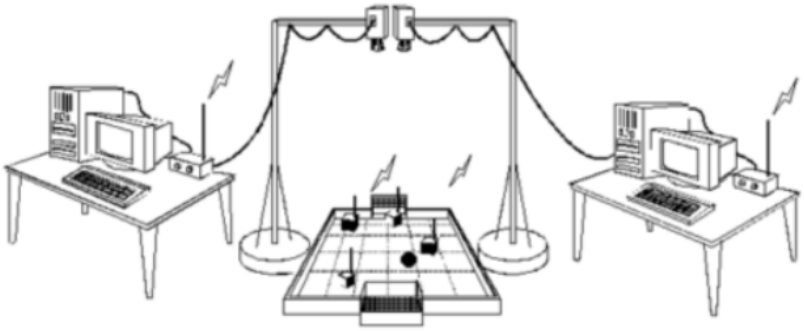

Figura 5 – Estrutura da competição. Fonte: (PINTO, 2017).

Inicialmente, todas as competições da categoria VSSS eram jogadas em ambiente físico, e a estrutura das partidas seguia o apresentado na seção anterior. Contudo, durante a pandemia, uma nova modalidade emergiu, e com ela, novas oportunidades surgiram. Com a necessidade do isolamento social, que praticamente impediu a execução de eventos presenciais, partidas *online* começaram a ser disputadas.

Estas partidas são centralizadas em um simulador, que será apresentado na seção 2.3, que elimina as diferenças físicas entre robôs e premia a equipe com melhor preparo estratégico.

2.2 Estrutura atual da equipe Rodetas

A equipe Rodetas utiliza atualmente o VSS-SDK, um conjunto de ferramentas de desenvolvimento projetado especificamente para a categoria IEEE *Very Small Size Soccer*. O VSS-SDK possui três módulos principais: VSS-Vision, responsável pela visão computacional; VSS-Simulator, que realiza a simulação dos robôs; e VSS-Viewer, utilizado para visualização de posições e depuração. Na Figura 6, é possível observar uma captura de tela do VSS-Viewer exibindo as posições atuais e finais dos robôs, bem como as trajetórias planejadas em uma determinada simulação.

Graças ao uso do SDK, a equipe é responsável apenas pelo desenvolvimento dos algoritmos de controle, movimentação dos robôs e estratégia de jogo. Com essas ferramentas disponíveis, não é necessário desenvolver algoritmos complexos para reconhecimento dos robôs em campo (visão computacional), já que essa funcionalidade é implementada de forma estável na plataforma. Isso permite que a equipe concentre seus esforços em inovações, como desvio de obstáculos e a implementação de estratégias baseadas em inteligência artificial, por exemplo.

O VSS-SDK é um projeto *open source* e a equipe Rodetas teve grandes contribuições no desenvolvimento dele. O projeto pode ser acessado no *Github* por meio do link https://vss-

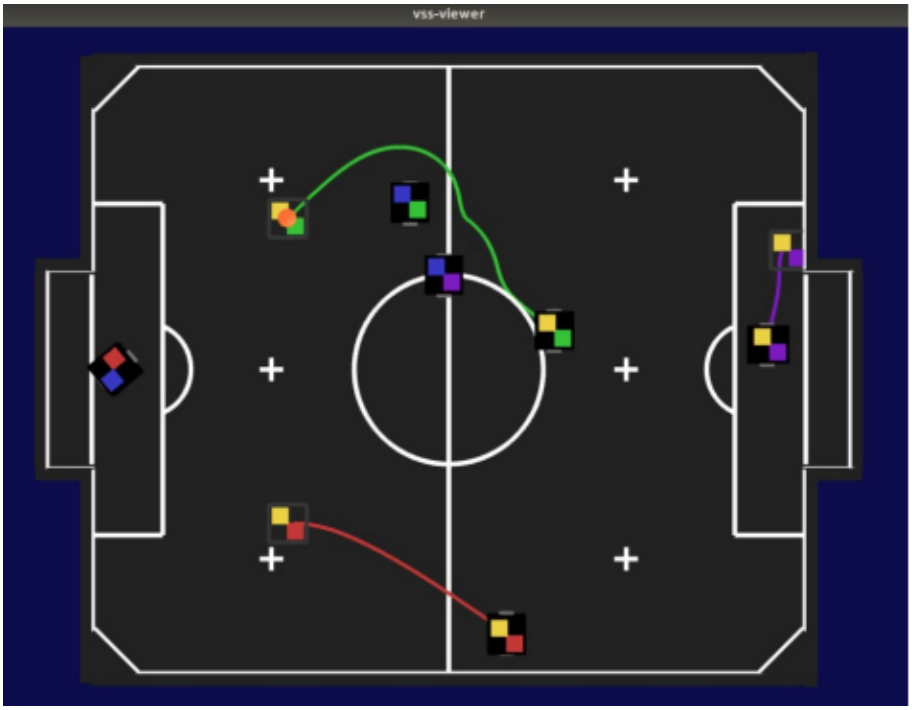

Figura 6 – VSS-VIEWER. Fonte: Elaborado pelo autor.

sdk.github.io/book/general.html.

A estratégia de jogo da equipe Rodetas é bem definida, e pode ser vista no seguinte trecho do trabalho de Paulo Henrique dos Santos (2019):

> O código da estratégia é implementado em C++ com os paradigmas de orientação a objeto. O desenvolvimento do algoritmo permite, em tempo real, a troca de funções entre os jogadores, dando desta forma maior dinâmica ao jogo. As estratégias criadas estão diretamente ligadas às posições dos jogadores e da bola no campo, criando condições e estados que o programa identifica e gerencia de acordo com o pré-estabelecido no código. Deste modo, gera-se uma certa inteligência do computador na tomada de decisões durante a partida.

As estratégias são separadas por funções em campo, onde cada robô assume uma função diferente na partida, sendo estas as funções de goleiro, defensor e atacante. Cada função possui seu grupo de estratégias específico.

O controle dos robôs é dividido em duas partes: a primeira trata do problema de planejamento do caminho que cada robô deve seguir. Esse caminho é calculado a todo instante e leva em consideração os demais robôs no campo como obstáculos a serem evitados. O caminho é gerado por meio de um campo vetorial. Portanto, para uma dada posição do robô, existe um vetor que aponta para a direção que o robô deve seguir. A segunda parte diz respeito ao controle do robô em si. Este é responsável por permitir que o robô execute o caminho planejado. A entrada do controlador é o vetor de direção que aponta para a direção que o robô deve ir. Assim, com base na orientação desejada e na orientação do robô medida pelo sistema de visão, calcula-se o erro de orientação. As saídas do controlador são as velocidades de cada roda do robô. As velocidades são calculadas por meio de um controle proporcional, integral e derivativo (PID) do erro de

orientação. Posteriormente, as velocidades são transformadas em sinais de PWM (Modulação por Largura de Pulso) e são transmitidas para cada robô.

O código embarcado possui duas camadas: comunicação (para receber dados) e execução de comandos para controle cinemático, que são transmitidos pelo transmissor. O computador responsável por tomar decisões sobre as ações de cada robô agrega uma mensagem que é posteriormente enviada por meio de um módulo Xbee, permitindo a comunicação remota com um Arduino. Cada robô recebe uma mensagem específica, decodifica-a por meio de seu código de identificação e atualiza o valor PWM de suas saídas para alterar a velocidade dos motores. A comunicação entre computador e robôs ocorre a uma taxa de 60 mensagens por segundo.

A equipe ainda enfrenta um desafio não resolvido em relação à combinação de velocidade e precisão no controle dos robôs. Embora ambas as tarefas funcionem bem individualmente, aplicá-las simultaneamente é um desafio. O robô pode atingir altas velocidades, enquanto o controle mostra-se preciso ao seguir o caminho desejado. No entanto, quando é solicitado que o robô percorra um caminho em alta velocidade, a precisão é prejudicada e, muitas vezes, ocorrem colisões devido ao controle não funcionar adequadamente em curvas de alta velocidade.

2.3 Ferramentas utilizadas

Com base no GRSim (MONAJJEMI; KOOCHAKZADEH; GHIDARY, 2011), o FIRASim, ilustrado na Figura 7 é um simulador de partidas da categoria VSSS que se baseia no protocolo de comunicação Protobuf (*Protocol Buffers*) desenvolvido pela Google. Ele atua como um servidor que centraliza a comunicação das equipes com o campo através de uma porta da máquina onde está hospedado. Esse simulador é a principal ferramenta disponibilizada para o desenvolvimento das competições *online*. No cenário atual, é necessário que ambos os times estabeleçam uma conexão via SSH com a máquina hospedeira e executem o projeto da equipe nessa máquina a fim de se comunicar com o sistema.

O pacote de informações fornecido pelo FIRASim para as equipes é composto de informações sobre a partida, o campo, a bola e cada um dos robôs presentes na partida. Os dados enviados seguem a seguinte estrutura:

- *Environment* (Informações do Ambiente/Partida)
  - *step*: Total de gols da equipe azul
  - *goals_blue*: Total de gols da equipe amarela
  - *goals_yellow*: Total de pacotes enviados durante a partida
- *Field* (Campo)
  - *width*: largura do campo
  - *length*: comprimento do campo

  - *goal_width*: largura do gol
  - *goal_depth*: profundidade do gol

- *Frame* (Objeto responsável por armazenar informações dos robôs e da bola)
  - *Ball* (Bola)
    - *x*: posição da bola no eixo X
    - *y*: posição da bola no eixo Y
    - *z*: posição da bola no eixo Z
    - *vx*: velocidade da bola no eixo X
    - *vy*: velocidade da bola no eixo Y
    - *vz*: velocidade da bola no eixo Z
  - *Robot* (Robô - um item deste é criado para cada um dos robôs do campo)
    - *robot_id*: número identificador do robô
    - *x*: posição do robô no eixo X
    - *y*: posição do robô no eixo Y
    - *vx*: velocidade do robô no eixo X
    - *vy*: velocidade do robô no eixo Y
    - *vorientation*: velocidade angular do robô

Os pacotes de informação enviados pelas equipes para o simulador são definidos por um vetor de objetos, onde cada objeto representa um robô da equipe, possuindo dois parâmetros, que são as velocidades angulares de cada roda, em radianos por segundo (rad/s).

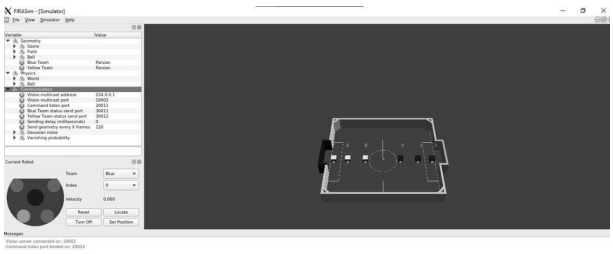

Figura 7 – *Software* FIRASim. Fonte: Elaborado pelo autor.

2.3.1 RSoccer

Com a evolução das partidas da categoria VSSS para o âmbito *online*, a busca pelo desenvolvimento de novas tecnologias que aprimorassem a estratégias das equipes se tornou cada vez maior. Uma das grandes áreas exploradas foi a área da Inteligência artificial, e este é o foco do RSoccer. Desenvolvido por Martins et al. (2021), este simulador *open-source* busca fornecer ambientes e cenários otimizados para experimentos de aprendizado de máquina por reforço nas categorias VSSS e SSL. Para a categoria VSSS dois ambientes são disponibilizados:

- VsssV0: Ambiente onde o foco é o treinamento de um único robô, visando otimizar tarefas individuais. Durante o treinamento, todos os outros robôs perfomam instruções aleatórias. A única mudança no pacote de informações enviado para o simulador, comparado com o FIRASim, é que neste se faz necessário somente o envio de informações de um único robô.
- VsssMA(MultiAgent): Ambiente onde o foco é o treinamento dos três robôs da equipe, visando otimizar o desempenho da equipe como um todo. O pacote de informações enviado para o simulador é igual ao do FIRASim.

O funcionamento do RSoccer pode ser divido nas seguintes etapas:

- *step*: é o recorte de tempo que o simulador irá utilizar para enviar pacotes de informação para o agente. Por padrão, cada *step* tem a duração de 0,25 segundos, ou seja, a cada 0,25 segundos um pacote de informações é enviado;
- *action*: ao receber um pacote de informações o código da equipe deve processar este dados e enviar um pacote de informações para o simulador. Este pacote enviado recebe o nome de *action*, uma vez que ele representa a ação que será executada na partida naquele momento;
- *reward*: após uma *action* ser enviada para o ambiente o simulador irá verificar o resultado da ação. A partir disso, uma recompensa será atribuída à jogada e retornada para o agente. Esta recompensa ajuda o código a interpretar se a ação realizada foi ou não eficiente, e pode servir de métrica para os algoritmos de aprendizado por reforço;
- *episode*: Um episódio nada mais é que uma partida jogada no simulador. Ele tem a duração de 30 segundos, ou até que aconteça um gol. Após este episódio é retornado para o ambiente a recompensa total do agente naquela partida, o que torna possível avaliar se o código executado teve uma boa performance. Ao final de cada episódio um novo é iniciado.

2.4 Inteligência Artificial

2.4.1 Definição

Este trabalho utiliza inteligência artificial (IA) para operar os robôs da equipe Rodetas Robô Clube. De acordo com Fernandes (2003), IA é a parte da Ciência da Computação voltada para o desenvolvimento de sistemas de computadores inteligentes, ou seja, sistemas que exibem características relacionadas à inteligência humana, como compreensão da linguagem, aprendizado, raciocínio e resolução de problemas.

Por outro lado, Fernandes (apud LUGER; STUBBLEFIELD, 1993) define inteligência artificial como o ramo da computação que se preocupa com a automação de comportamentos inteligentes.

A inteligência artificial tem sido amplamente explorada atualmente, com vários trabalhos sendo desenvolvidos com grande sucesso. A empresa iFlytek criou um robô que passou no exame nacional de licenciamento de médicos da China (SARACCO, 2017). Em Silver et al. (2018), é possível observar o desempenho da tecnologia desenvolvida pela Google para aprender Xadrez, Shoji e Go, jogos com uma complexidade muito elevada, nos quais os algoritmos propostos foram capazes de vencer os campeões mundiais das modalidades apresentadas.

2.4.2 Redes Neurais Artificiais

Aprofundando no conceito de inteligência artificial, o modelo escolhido para este trabalho foi o de redes neurais artificiais (RNA).

As RNAs são comumente utilizadas na resolução de problemas complexos, onde o comportamento das variáveis não é rigorosamente conhecido (FLECK et al., 2016). Este modelo tem como objetivo simular computacionalmente o comportamento do cérebro humano.

De acordo com Haykin (2000), a capacidade de um ser humano em realizar tarefas complexas e, sobretudo, aprender, é resultado do processamento paralelo e distribuído da rede de neurônios em seu cérebro. O córtex cerebral, camada mais externa do cérebro, é responsável pelo processamento cognitivo. Cada nova informação ou experiência pode ocasionar mudanças na estrutura do cérebro, como o fortalecimento ou enfraquecimento de conexões entre neurônios. Tais mudanças são geradas a partir do rearranjo das redes de neurônios no cérebro.

Sendo assim, uma rede neural artificial é definida por uma rede de neurônios, que são capazes de aprender e tomar decisões por conta própria, utilizando modelos matemáticos aplicados em seus neurônios, que se comunicam entre si, a fim de gerar uma saída para determinados estímulos de entrada. Segundo Fleck et al. (2016), uma RNA corresponde a um esquema de processamento capaz de armazenar conhecimento baseado em aprendizagem e disponibilizar esse conhecimento para a aplicação à qual se destina.

A Figura 8 apresenta um modelo de neurônio artificial proposto por Haykin (2000). À esquerda do neurônio (X_1 a X_m) podem ser vistos os sinais de entrada que o mesmo recebe. Esses sinais podem vir do ambiente em que a rede neural se encontra, caso o neurônio seja um neurônio da camada de entrada da rede neural, ou então de outro neurônio, caso este neurônio seja de uma camada oculta ou da camada de saída da rede neural. Além disso, tem-se uma função matemática, normalmente representada pelo somatório do bias (b_k) e das entradas multiplicadas pelo peso (W_{k1} a W_{km}) definido para cada conexão, que é a função responsável por manipular os valores de entrada e fornecer um valor que será enviado para uma função de ativação, que finalmente será responsável por fornecer a saída do neurônio (Y_k).

A partir disso, montam-se estruturas utilizando esses neurônios até obter uma rede neural completa, formada por camadas de entrada, camadas ocultas e camadas de saída, como apresentado na Figura 8. Essa estrutura constitui uma rede neural artificial(RNA), cujaa disposição

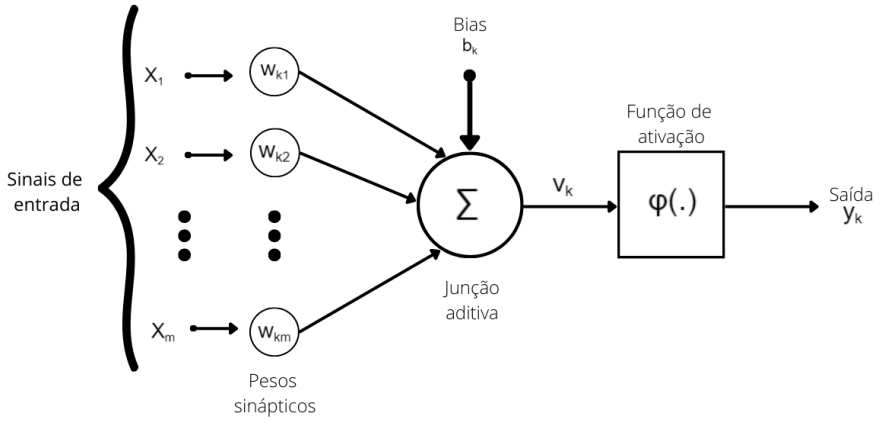

Figura 8 – Modelo neuronal. Fonte: Elaborado pelo autor.

de seus neurônios constitui a *topologia da rede*. As informações são transmitidas através das conexões entre os neurônios, onde cada neurônio processa o sinal de entrada e transmite para todos os neurônios da próxima camada.

No trabalho de Quintero et al. (2014), é possível acompanhar a utilização de redes neurais no treinamento de robôs da categoria *Small Size Soccer*. A abordagem utilizada foca em Redes Neurais do tipo *feedforward*, cujo objetivo é prever e impedir a utilização de jogadas ensaiadas realizadas pela equipe adversária. O algoritmo recebe informações referentes à posição e velocidade dos robôs e da bola, e verifica se a situação atual pode ser classificada como um cenário de jogada ensaiada esperado. Caso um cenário seja identificado pela rede neural, o programa executa uma estratégia para impedir que essa jogada seja executada com sucesso.

Em Pena et al. (2020), é possível notar uma abordagem de redes neurais com aprendizado não supervisionado para robôs da categoria VSSS. O artigo apresenta o uso de uma Q-Network. Este tipo de rede neural funciona, resumidamente, como um classificador de ações predefinidas pelo desenvolvedor. No trabalho citado, a Q-Network atua como treinadora da equipe, selecionando qual será o comportamento ideal do robô em um determinado momento, escolhendo se este atuará como goleiro, defensor ou atacante com base em um cenário.

2.4.3 Aprendizado por reforço

Existem várias maneiras de treinar uma rede neural. Neste trabalho, foi escolhida a abordagem de aprendizado por reforço. O aprendizado por reforço pode ser definido como uma sub-área do aprendizado de máquina em que o modelo aprende por tentativa e erro. Em cada estado, o agente realiza uma ação no ambiente e é recompensado de acordo com o seu desempenho (PENA et al., 2020). Com base nessas recompensas, o algoritmo é capaz de identificar se está trabalhando de maneira eficiente no ambiente e converge para o resultado desejado.

2.5 Otimização e evolução de redes neurais

O sucesso da solução de qualquer problema utilizando redes neurais depende de um fator muito importante: a configuração de pesos, funções de ativação e topologia da rede. A otimização dessas configurações é foco de muitos estudos e alguns deles serão apresentados nesta seção.

No trabalho de Whitley, Starkweather e Bogart (1990), são apresentadas algumas abordagens que focam no uso de algoritmos genéticos para a otimização dos pesos entre conexões. A ideia central se concentra no fato de que uma rede neural totalmente conectada precisa apenas de uma combinação ideal de pesos, e o algoritmo genético é responsável por manipulá-los até encontrar tal combinação. Já o artigo de Zhou et al. (2019) demonstra a aplicação do algoritmo chamado Water Wave Optimization (WWO), que foca em otimizar os pesos e o número de neurônios das camadas ocultas. Este algoritmo representa cada solução como uma onda, onde soluções com uma *fitness* alta (alta energia) têm um comprimento de onda mais curto e exploram uma área menor. O objetivo é encontrar uma onda que equilibre esses parâmetros. Esse comportamento pode ser visto na Figura 9.

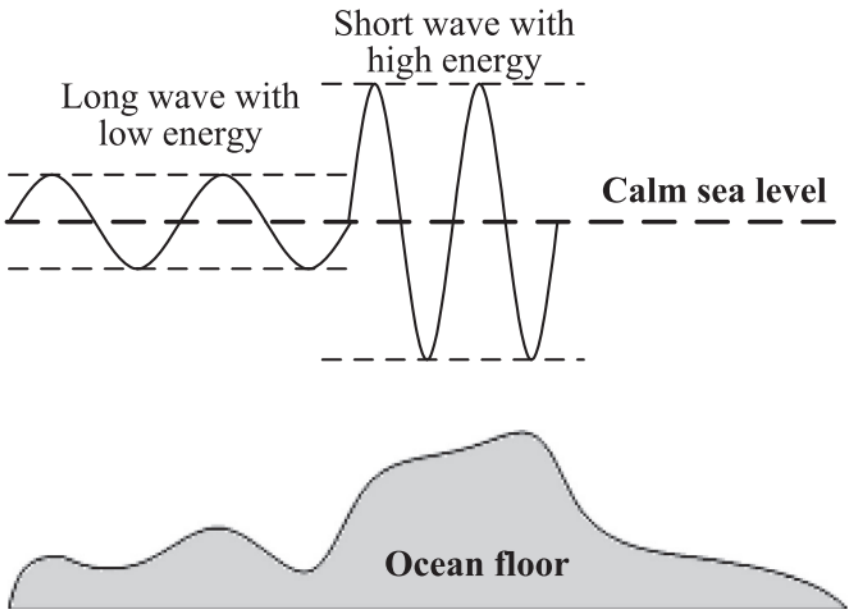

Figura 9 – Exemplos de ondas do algoritmo WWO. Fonte: Zhou et al. (2019).

No trabalho de Sinha, Haidar e Verma (2018), nota-se a utilização de algoritmos de Otimização por Enxame de Partículas (PSO), do inglês *Particle Swarm Optimization* - PSO), focados em otimizar os parâmetros de uma rede neural convolucional. Nessa abordagem, cada partícula representa uma possível solução que atravessa o espaço de busca se comunicando com outras partículas a fim de encontrar a solução ótima do problema proposto.

No presente trabalho, a Neuroevolução de Topologias Aumentadas (NEAT) é a proposta de otimização de redes neurais abordada. Ela foi apresentada por Stanley e Miikkulainen (2002)

em 2002 e baseia-se na utilização de algoritmos genéticos para evoluir pequenas redes neurais em redes neurais mais complexas, através da manipulação de seus pesos e conexões (MENDES, 2021). Na seção 2.6, tem-se a discussão do conceito de algoritmos genéticos para, posteriormente, na seção seção 2.7, expandir-se os conceitos principais e o funcionamento do NEAT.

2.6 Algoritmos genéticos

Algoritmos genéticos são amplamente conhecidos e baseados no processo biológico de evolução. Foram inicialmente propostos por Holland (1992) e imitam a teoria da seleção natural apresentada por Charles Darwin, onde os indivíduos mais adaptados ao ambiente sobrevivem. Sendo assim, a base destes algoritmos é definida pela representação da solução na forma de cromossomos, normalmente com valores binários, uma função de *fitness* que é responsável por avaliar o desempenho de cada solução, além de operações genéticas, que têm como objetivo a diversificação da população. Nas subseções a seguir serão discutidas as principais operações genéticas realizadas nesses algoritmos, que são baseadas em operações biológicas.

2.6.1 *Crossover*

O operador de crossover, também conhecido como cruzamento, é uma operação genética responsável por gerar novos indivíduos dentro da população. Nessa operação, dois indivíduos são selecionados, geralmente dando prioridade aos indivíduos com melhores avaliações, e suas soluções são recombinadas. A partir dessa recombinação, novos indivíduos são gerados e adicionados à população, com o objetivo de diversificar as soluções presentes. A Figura 10 ilustra essa operação.

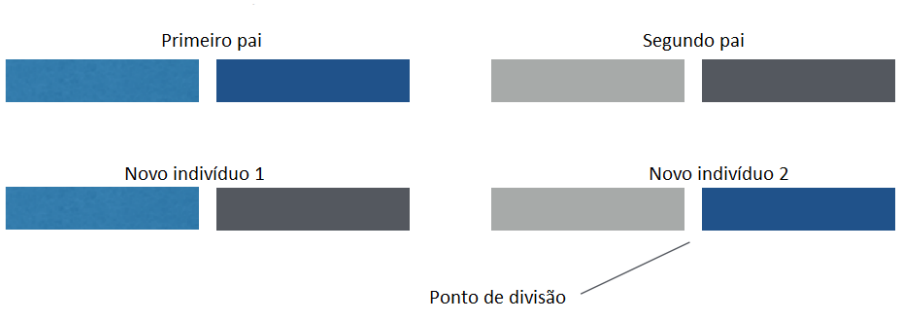

Figura 10 – Operação de crossover. Adaptado de Kramer (2017).

2.6.2 Mutação

O operador genético de mutação é o operador responsável por inserir pequenas mudanças aleatórias nos indivíduos da população. Tais alterações tem como objetivo impedir que a população não evolua e/ou fique estagnada. Para cumprir com esta tarefa este operador modifica um gene de um cromossomo selecionado a partir de uma probabilidade de mutação previamente estabelecida.

2.6.3 Funcionamento geral de um algoritmo genético

O Algoritmo 1 apresenta o fluxo de funcionamento de um algoritmo genético clássico, adaptado de um algoritmo apresentado no trabalho de Katoch, Chauhan e Kumar (2020).

Algorithm 1 Algoritmo genético clássico

Require: Tamanho da população -> N, Número máximo de iterações MAX
Ensure: Melhor solução global Y_m
  Gera população inicial de N indivíduos Y_i (i=1,2...N)
  j $\leftarrow 0$
  Computa fitness de cada indivíduo e atualiza melhor solução global
  while $j \leq MAX$ **do**
    Seleciona um par de indivíduos da população inicial baseado na fitness
    Aplica operação de crossover no par selecionado através da probabilidade de crossover
    Aplica mutação nos novos indivíduos a partir da probabilidade de mutação
    Troca a população antiga pela nova população gerada
    Computa fitness de cada indivíduo e atualiza melhor solução
    Incrementa j em 1
  end while
  retorna a melhor solução Y_m

2.7 Funcionamento do NEAT

Após estabelecer os princípios fundamentais dos algoritmos genéticos, é viável expor o modo de operação do algoritmo NEAT. Este processo é dividido em quatro fases:

1. Codificação genética;
2. Acompanhamento de genes através de histórico de marcas;
3. Garantia da diversidade por meio de especiação;
4. Redução das dimensões da rede.

As próximas subseções oferecem uma análise minuciosa de cada uma dessas etapas, com base no artigo original de Stanley e Miikkulainen (2002).

2.7.1 Codificação genética

No algoritmo NEAT, cada indivíduo é caracterizado por um conjunto de genes que representa a estrutura completa da rede neural. Cada gene representa uma conexão e contém informações sobre o nó de entrada, o nó de saída, o peso da conexão e um valor binário que indica se ela está ativa ou não. Além disso, cada gene possui um número de inovação que será explicado na seção sobre o histórico. A Figura 11 apresenta a estrutura básica de um indivíduo no NEAT.

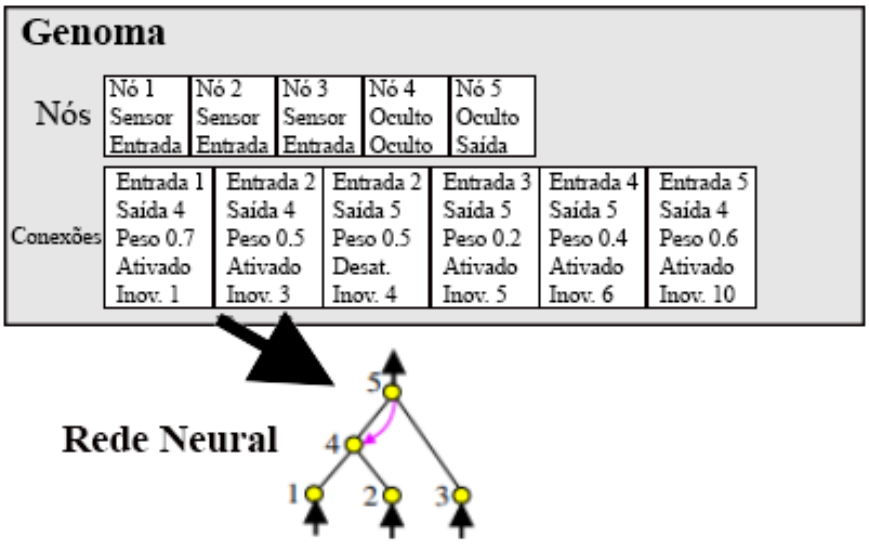

Figura 11 – Estrutura básica de um indivíduo. Fonte: Adaptado de Stanley e Miikkulainen (2002).

.

2.7.1.1 Operações genéticas

Com base na definição do indivíduo descrito anteriormente, o algoritmo NEAT propõe três operações genéticas, que consistem em dois tipos de mutação e uma operação de *crossover*.

A primeira operação de mutação é conhecida como *mutação de adição de nós*. O objetivo dessa mutação é alterar a estrutura da rede adicionando nós entre conexões já existentes. Para realizar essa mutação, uma conexão é selecionada e dividida em duas partes, criando uma nova interseção onde um novo nó é adicionado, conforme ilustrado na Figura 12. A segunda operação de mutação é a *mutação de adição de conexão*, que cria uma nova conexão entre dois nós que não estavam previamente conectados, alterando a configuração da estrutura da rede, como pode ser visto na Figura 13. Essas mutações são responsáveis por modificar o tamanho da rede e manipular sua topologia, a fim de explorar novas possibilidades para a solução do problema proposto.

A operação de crossover será discutida na subseção 2.7.2, que mostrará como o NEAT lida com a grande diversidade de indivíduos.

2.7.2 Acompanhamento de genes através de histórico de marcas

Para efetuar o cruzamento de indivíduos, é necessário que o NEAT saiba exatamente quais genes estão presentes em cada um deles. Identificar e comparar indivíduos em cada operação de cruzamento seria computacionalmente custoso. Por esse motivo, o NEAT propõe a adição do *número de inovação* em cada gene de um indivíduo. Esse número é atribuído ao gene em sua criação e é exclusivo para cada novo gene.

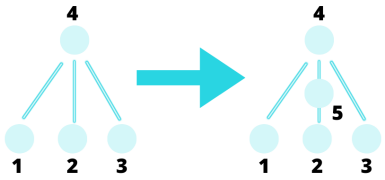

Figura 12 – Operação de mutação por adição de nó. Fonte: Elaborado pelo Autor.

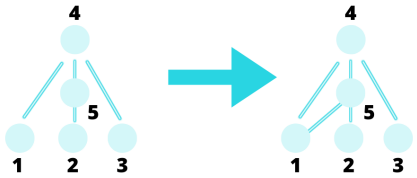

Figura 13 – Operação de mutação por adição de conexão. Fonte: Elaborado pelo Autor.

Ao efetuar as operações genéticas, o número de inovação de um gene é repassado para a nova geração, permitindo assim que todo o histórico de evolução de um indivíduo seja acompanhado por essas "marcas genéticas". Dessa forma, o processo de cruzamento de indivíduos é realizado da seguinte maneira: dois indivíduos são selecionados e seus genes são comparados. Genes com o mesmo *número de inovação* são sobrepostos, herdando conexões de ambos os pais. Já os genes que não possuem correlação são herdados do pai com maior valor de *fitness* ou aleatoriamente, caso ambos tenham uma avaliação igual. Essa abordagem de registro de histórico dos genes permite que qualquer par de indivíduos seja combinado de forma simples e rápida, sem o uso de algoritmos de análise de topologias. A Figura 14 apresenta de forma gráfica o funcionamento dessa operação genética.

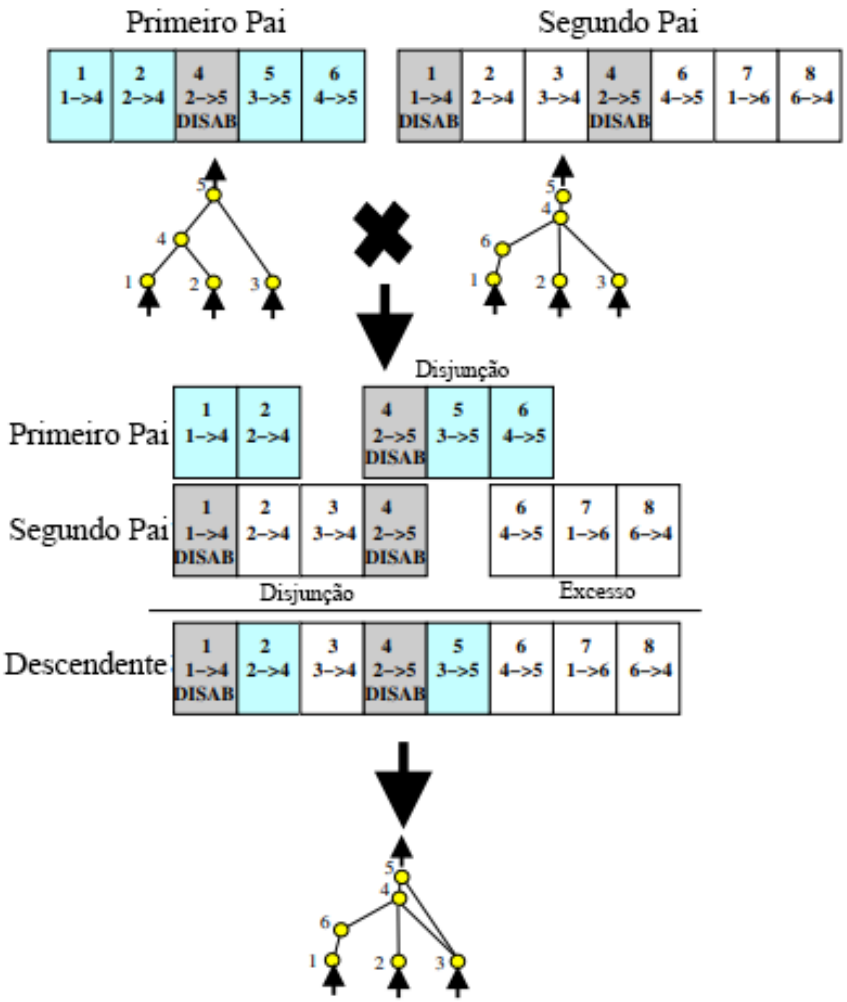

Figura 14 – Operação de crossover. Fonte: Adaptado de Stanley e Miikkulainen (2002).

2.7.3 Garantia da diversidade por meio de especiação

De forma geral, ao introduzir uma nova estrutura em uma rede neural, o seu comportamento padrão inicial é de piora no desempenho. A longo prazo, essa mudança estrutural pode se pagar e, por isso, uma das propostas do NEAT é o conceito de especiação dos indivíduos. Indivíduos com uma certa proximidade genética são considerados parte da mesma espécie. A partir dessa separação, a competição dos indivíduos ao longo das gerações acontece somente com indivíduos da mesma espécie. Dessa forma, possíveis inovações adicionadas nas redes neurais são protegidas, garantindo o tempo necessário para uma evolução desses indivíduos.

Um ponto negativo a ser considerado nesse mecanismo pode ser o tempo gasto para evolução de uma espécie que não esteja progredindo. Para evitar que isso aconteça, o algoritmo propõe um parâmetro de estagnação. Um contador de estagnação da espécie é incrementado a cada rodada em que a espécie não demonstra melhora no *fitness*. Quando esse contador atinge um valor predefinido, por padrão 20, a espécie é removida e novos indivíduos aleatórios são

inicializados para tomarem seu lugar. Com todas essas ferramentas, o NEAT garante a proteção à diversidade e inovação dos indivíduos.

2.7.4 Redução das dimensões da rede

Grande parte dos algoritmos de otimização de topologia de redes neurais inicializa a sua população com topologias aleatórias. Ao começar desta maneira, muito processamento é gasto em redes pouco otimizadas e topologias muito complexas podem ser criadas para resolver um problema que não necessita de tanta complexidade. Para solucionar este problema, o NEAT inicia sua execução com um conjunto de indivíduos idênticos e com topologia mínima: redes neurais cuja topologia é formada apenas pelas entradas e pelas saídas do problema proposto, e nada mais. A partir daí, as redes são incrementadas através dos cruzamentos e mutações mencionados anteriormente. Dessa forma, o número de gerações necessárias para chegar à solução do problema é reduzido, uma vez que o NEAT percorre um número mínimo de pesos e conexões entre neurônios.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as ferramentas e metodologias utilizadas para o desenvolvimento deste trabalho.

3.1 Tecnologias

Este trabalho foi desenvolvido utilizando a linguagem de programação Python. A escolha dessa linguagem deveu-se, em grande parte, ao ecossistema que ela oferece para o desenvolvimento de soluções de Inteligência Artificial, bem como ao suporte necessário para o uso da biblioteca base do RSoccer, o OpenAI Gym. Além disso, o algoritmo de otimização de redes neurais escolhido para solucionar o problema foi o NEAT, apresentado na seção 2.7. A biblioteca principal disponível para utilizar esta metodologia é a *Neat-Python* (MCINTYRE et al., 2015), que fornece métodos e mecanismos de integração, evitando a necessidade de desenvolver a base do NEAT dentro do código. A existência desta biblioteca é mais um fator que justifica a escolha do Python como linguagem de programação.

3.2 Ferramentas

Para o desenvolvimento do trabalho utilizou-se a arquitetura computacional e as ferramentas listadas a seguir:

- Notebook AMD Ryzen 5 3500U QuadCore com 16GB RAM para desenvolvimento de código;
- Servidor do Laboratório de Computação de Sistemas Inteligentes (CSILab) do Departamento Departamento de Computação da Universidade Federal de Ouro Preto, equipado com Intel i9-10900 decacore (20 threads) 2.80GHz, 128GB RAM DDR4 e GPU RTX 3090, de 24GB de RAM GDDR6X com aproximadamente 10 mil cuda cores;
- *Software* Visual Studio Code para desenvolvimento de código;
- Windows Subsystem for Linux (WSL) para execução de ferramentas do Ubuntu dentro do Windows.

3.3 Problemas e dificuldades do sistema atual da equipe Rodetas

Na seção 2.2, foi apresentado que a equipe Rodetas tem uma estrutura bem definida das responsabilidades do código, o que facilita o entendimento e manutenção. No entanto, o projeto enfrenta algumas limitações que serão discutidas abaixo.

Uma das limitações é a definição manual das estratégias de jogo, que são baseadas em cenários predefinidos e podem limitar a equipe quando um adversário usa uma tática não prevista. Além disso, a criação de novos cenários pode aumentar a complexidade do código. Adicionalmente, tratar situações específicas pode ser demorado e levar a verificações complexas, que afetam o desempenho da equipe se feitas incorretamente.

No que diz respeito ao controle do robô, o processo de geração de caminhos pode ser um desafio. A complexidade estratégica pode aumentar o tempo para definir o alvo a ser atingido, enquanto a complexidade do algoritmo de geração de caminhos pode afetar o tempo de processamento. Como a equipe envia 60 pacotes por segundo, o tempo de entrada da informação no código até a geração do pacote de saída não pode ultrapassar 0,017 segundos. Portanto, a geração de caminhos é um problema que deve ser abordado pela equipe.

3.4 Abordagem Proposta

Este trabalho propõe a implementação de uma rede neural para substituir os módulos de estratégia, movimentação e controle da equipe Rodetas, com o objetivo de solucionar alguns problemas apresentados na seção 3.3. O projeto tem como base dois pilares principais: otimizar o tempo de execução de código e minimizar a atuação humana na definição de estratégia e na tomada de decisão dos robôs. Para atingir o resultado esperado, diversas metodologias foram utilizadas até atingir a abordagem final. Alguns testes e experimentos realizados são descritos a seguir para fornecer material para estudos futuros.

Inicialmente, foi tentada a implementação de redes neurais em C++ com o objetivo de resolver o problema proposto. Algumas bibliotecas foram utilizadas, e uma implementação manual de novos módulos foi realizada. A primeira abordagem do projeto foi voltada para o uso do TensorFlow, por ser uma das principais plataformas disponíveis (ABADI et al., 2015). No entanto, a implementação da RNA se mostrou bastante complexa, além de que o custo computacional seria elevado ao utilizar esta biblioteca.

A segunda plataforma utilizada foi a Torch (PASZKE et al., 2019), um *framework* de código aberto focado em aprendizado de máquina que agiliza o desenvolvimento de ferramentas em pesquisa. Esta plataforma se mostrou muito mais simples e objetiva do que o TensorFlow, e foram encontradas implementações semelhantes ao objetivo deste trabalho, portanto existiam referências de projetos que ajudaram muito no desenvolvimento. Entretanto, a falta de suporte fez com que o Torch deixasse de ser uma opção para o desenvolvimento do projeto.

Após as duas implementações citadas anteriormente, outras bibliotecas foram testadas no projeto, como a Shark[1] e a Caffe [2], mas a falta de documentação e suporte fez com que as alternativas fossem deixadas de lado e dois módulos fossem desenvolvidos dentro do projeto: um módulo de construção de redes neurais e um módulo de execução de um algoritmo genético.

[1] disponível em https://github.com/Shark-ML/Shark
[2] Disponível em https://github.com/BVLC/caffe

Nesta abordagem, o algoritmo genético ficou responsável pela mutação dos pesos da rede neural definida a fim de chegar a uma configuração capaz de resolver o problema. O grande desafio encontrado foi o de treinar esta rede neural a partir do algoritmo genético. Com as ferramentas disponíveis, havia uma grande limitação de número de indivíduos por geração a ser treinada, e a execução de múltiplas instâncias do programa causava um grande custo computacional, além de depender da disponibilidade de portas TCP/IP para comunicação.

Dadas as limitações e dificuldades encontradas no C++, a linguagem de programação Python foi escolhida para a continuação do trabalho. A escolha se deu por diversos fatores, como a facilidade de integração com as mais diversas tecnologias focadas em aprendizado de máquina e a compatibilidade com o FIRASim. A partir dessa escolha, o projeto evoluiu e chegou à implementação final usando a biblioteca Neat-Python.

Desta forma, a primeira etapa de implementação deste trabalho foi a criação de um código base em Python capaz de receber informações provenientes do Rsoccer (MARTINS et al., 2021), tratá-las, e fornecê-las como entrada da rede neural a ser utilizada. Na sequência, este código deve receber a saída da rede neural e enviar para o Rsoccer a ação a ser realizada. Esta implementação segue o apresentado no Algoritmo 2.

Algorithm 2 Código base para disputa de partidas utilizando o novo módulo de redes neurais

Require: Dados fornecidos pelo Rsoccer
Ensure: Comandos para os robôs
  Recebe dados do Estado atual do campo pelo Rsoccer
  Concatena dados em um vetor compatível à entrada da rede neural
  Entrega dados para a entrada da rede neural
  Recebe saídas da rede
  Envia saídas para cada robô

3.5 Configuração do NEAT

Para iniciar a utilização do NEAT é necessário configurar uma série de parâmetros em um arquivo de configurações que dita o funcionamento da rede neural e também da execução do algoritmo genético. Esta subseção tem como objetivo apresentar os principais parâmetros e discorrer sobre a importância dos mesmos na execução desta ferramenta.

- *pop_size*: é o parâmetro responsável pela definição do número de indivíduos (soluções) em cada geração do algoritmo genético executado. Para garantir uma grande diversidade e diminuir o número de gerações necessárias para convergência, utilizou-se neste trabalho um *pop_size* de 500 indivíduos;

- *fitness_criterion*: é a função usada como critério de parada do algoritmo genético. Seus valores possíveis são: *min*, representando uma parada por menor fitness da geração; *max* representando uma parada por maior fitness da geração; *mean* representando uma parada

por valor médio da geração. O valor escolhido foi o critério *max*, uma vez que o interesse deste trabalho está em encontrar o melhor indivíduo para solucionar o problema;

- *fitness_threshold*: é o limite da fitness a ser atingida pela função *fitness_criterion*. Ao atingir este valor, o algoritmo genético finaliza sua execução. O valor escolhido foi o de 100 pontos, para que a execução do algoritmo só seja finalizada de forma manual;
- *species_fitness_func*: função usada para calcular a fitness de uma espécie. Seus valores possíveis são: *min*, representando uma cálculo por menor fitness da geração; *max* representando um cálculo por maior fitness da geração; *mean* representando um cálculo do valor médio da geração. O valor escolhido foi o critério *max*, uma vez que o interesse deste trabalho está em encontrar o melhor indivíduo para solucionar o problema;
- *max_stagnation*: representa o máximo de gerações que a fitness de uma espécie pode ficar estagnada. Caso este valor seja atingido esta espécie é excluída. Caso não existam mais espécies, entra-se no cenário de extinção. O valor escolhido para este parâmetro foi o de 15 gerações;
- *species_elitism*: representa o número de espécies a serem preservadas contra a extinção. Desta forma o algoritmo ganha tempo para geração de novas espécies mesmo que todas estejam estagnadas. O valor escolhido foi o de 0 espécies;
- *reset_on_extinction*: define se o algoritmo será reiniciado com uma nova população em caso de extinção da população atual. Caso definido como falso, o algoritmo se encerra lançando uma exceção. O valor escolhido foi o de "verdadeiro";
- *elitism*: representa o número de melhores indivíduos por espécie que irão sobreviver de uma geração para outra. O valor escolhido foi o de 2 indivíduos por espécie;
- *survival_threshold*: define a taxa de reprodução (*crossover*) por geração de cada espécie. O valor escolhido foi o de 0,2, o que significa que 20% da população irá realizar *crossover* para a geração de novos indivíduos;
- *conn_add_prob*: a probabilidade de mutação do tipo adição de conexão entre dois nós existentes. O valor escolhido foi o de 0,15 ou 15%;
- *conn_delete_prob*: a probabilidade de mutação do tipo remoção de conexão entre dois nós existentes. O valor escolhido foi o de 0,1 ou 10%;
- *node_add_prob*: a probabilidade de mutação do tipo adição de nó em uma conexão existente. Ao efetuar esta adição, a conexão anterior é desabilitada. O valor escolhido foi o de 0,15 ou 15%;

- *node_delete_prob*: a probabilidade de mutação do tipo remoção de um nó. Ao efetuar esta operação, todas as conexões deste nó são desabilitadas. O valor escolhido foi o de 0,1 ou 10%;
- *num_inputs*: o número de nós de entrada na rede neural. O valor escolhido foi o de 40 nós, que representa o total de informações fornecidas pelo Rsoccer;
- *num_outputs*: o número de nós de saída da rede neural. O valor escolhido foi o de 6 nós, representando a velocidade de cada roda de cada um dos 3 robôs;
- *activation_default*: a função de ativação padrão fornecida para cada novo nó inserido na rede. a função escolhida foi a tangente hiperbólica, pois seus valores variam de -1 a 1, sendo assim compatíveis ao intervalo de valores esperado por cada um dos robôs;
- *weight_max_value*: o valor máximo permitido para o peso das conexões. Pesos acima deste valor serão fixados neste máximo. O valor escolhido foi 1;
- *weight_min_value*: o valor minimo permitido para o peso das conexões. Pesos abaixo deste valor serão fixados neste mínimo. O valor escolhido foi -1;
- *feedforward*: caso definido como 'verdadeiro' este parâmetro força as conexões a irem sempre para camadas a frente, não permitindo a retroalimentação da rede. Caso contrário, não haverá nenhum tipo de monitoramento em relação a este parâmetro. O valor escolhido foi o de verdadeiro;
- *response_max_value*: valor máximo do nó de saída da rede neural. Definido em 1 para corresponder com o intervalo esperado pelo Rsoccer;
- *response_min_value*: valor mínimo do nó de saída da rede neural. Definido em -1 para corresponder com o intervalo esperado pelo Rsoccer.

3.6 Treinamento da rede neural

Após a configuração de todos os parâmetros necessários torna-se possível a execução do treinamento da rede neural. Para tal, foi implementado um código em Python que faz uso do NEAT junto do Rsoccer. O funcionamento geral deste código pode ser visto no algoritmo 3.

A partir deste código, dois cenários de treinos foram propostos:

- Seguir a bola: nesse treinamento o único objetivo da rede neural é o de garantir que um dos robôs seja capaz de chegar até a bola. Cada execução do treinamento é finalizada assim que o robô chega até a bola;

- Partida completa: nesse treinamento o objetivo da rede neural é controlar todos os robôs para garantir que cumpram o objetivo de fazer um gol. Uma execução é finalizada quando um gol acontece ou após um determinado número de passos é atingido.

Algorithm 3 Estrutura geral de código para treinamento de rede Neural utilizando NEAT e Rsoccer

Require: Arquivo de configuração do NEAT
Ensure: Melhor rede neural capaz de solucionar o problema proposto

```
Inicializa neat com arquivo de configuração
pop ← neat.pop                          ▷ atribui população inicial do neat a variável pop
solucionado ← falso
while solucionado = falso do
    recompensa ← []                     ▷ Inicializa um vetor de zeros do tamanho da população
    j ← 0
    for individuo i de pop do           ▷ loop executado de forma paralela para cada indivíduo
        Inicia simulação do Rsoccer
        while simulação não for finalizada do
            i.entradas ← Rsoccer.state  ▷ Entrada da RN recebe estado atual do Rsoccer
            comandos = i.saidas
            envia comandos para Rsoccer
            recompensa[j] ← recompensa[j] + Rsoccer.recompensaDaJogada
        end while
        if recompensa[j] ≥ fitnessThreshold then
            solucionado ← verdadeiro
            melhorIndividuo ← i
        end if
        j ← j + 1
    end for
    Efetua mutações e crossover nos indivíduos de acordo com a recompensa de cada um
deles
    Gera próxima geração com novos indivíduos gerados de acordo com critérios definidos
nas configurações do NEAT
end while
retorna melhorIndividuo
```

3.6.1 Treinamento seguir a bola

Para este treinamento o único objetivo da rede neural é o de garantir que um robô seja capaz de ir até a bola. O objetivo é verificar se o NEAT é capaz de realizar tarefas mais simples e somente na sequência prosseguir para situações mais complexas.

Para tal, a seguinte *fitness* foi utilizada no Rsoccer:

$$\begin{cases} 0 & dist > 0,1m \\ 1 & dist \leq 0,1m \end{cases} \tag{3.1}$$

em que *dist* é a distância entre o robô e a bola em metros. A metodologia de avaliação escolhida foi baseada na disponibilizada no artigo de Martins et al. (2021), com o objetivo de confrontar os resultados do presente trabalho com os obtidos no *paper* original do *Rsoccer*.

3.6.2 Treinamento da partida completa

Para este treinamento é necessário configurar a forma que se deseja avaliar cada execução de uma partida. Desta forma, os seguintes parâmetros são utilizados para avaliar os indivíduos:

- Movimentação do robô em direção a bola ($moveReward$): Para calcular esta recompensa efetua-se o cálculo do cosseno do ângulo formado pelo vetor da velocidade do robô e o vetor posição do robô - posição da bola. Quanto menor o ângulo formado por estes vetores, maior será o valor do cosseno, e com isso maior a recompensa, uma vez que quanto menor for este ângulo, mais correta será a movimentação do robô em relação à bola. Esta relação pode ser visualizada na Figura 15. Em azul tem-se o vetor de velocidade do robô, em vermelho o vetor robô-bola. O cosseno do ângulo θ formado por estes dois vetores fornece a recompensa recebida pelo robô em questão;
- Energia potencial da bola ($ballPotentialEnergy$): Nesta recompensa a pontuação é concedida pela movimentação da bola em campo. Colocar a bola em movimento concede uma recompensa positiva ao robô enquanto jogadas que causem uma desaceleração na mesma concedem uma recompensa negativa. Para tal, os módulos de velocidade da bola do quadro atual são somados, e subtraídos da somatória destes módulos do quadro anterior. O resultado desta operação representa a recompensa a ser concedida para o indivíduo como um todo;
- ocorrência de gol: caso aconteça um gol na partida, 10 pontos são somados à recompensa caso o gol seja marcado pela rede em treinamento. Caso a rede sofra um gol, 10 pontos são descontados da recompensa.

Sendo assim, cada jogada do time de robôs como um todo é avaliada seguindo estes parâmetros e a pontuação da rede neural na partida é definida pelo somatório das pontuações de cada jogada durante a partida.

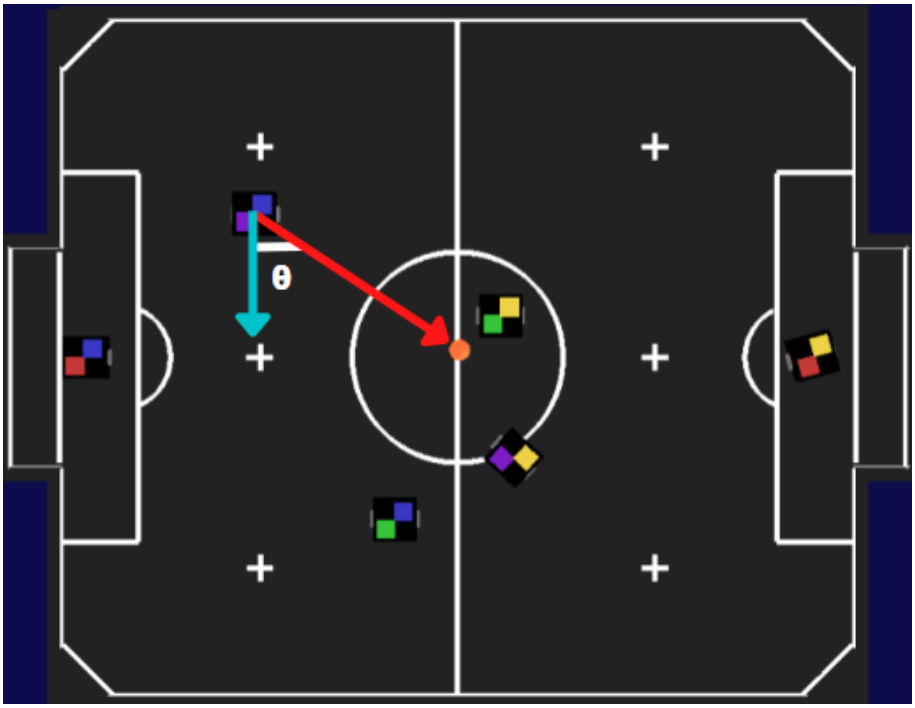

Figura 15 – Relação para determinar movimentação do robô em direção a bola. Fonte: Elaborado pelo Autor.

3.6.2.1 Problemas identificados

Após as primeiras execuções dos treinamentos, cujos resultados são apresentados na subseção 4.2.1, um comportamento padrão foi observado: inicialmente, toda a população inicial apresenta um desempenho ruim na resolução do problema proposto. Como resultado, a *fitness* desses indivíduos é bastante baixa, uma vez que eles são penalizados por executarem mal a tarefa. No entanto, qualquer pontuação, seja positiva ou negativa, exige que o robô execute a tarefa. Com isso, uma oportunidade encontrada pela rede neural foi manter os robôs sempre parados, pois uma *fitness* total de 0 ponto é maior do que qualquer *fitness* negativa.

Com isso, todos os testes realizados com as configurações iniciais convergiram para um mesmo valor: 0 ponto de *fitness*. Dessa forma, o melhor indivíduo de toda execução sempre será aquele que executar o menor número de movimentos, uma vez que este comportamento é muito mais fácil de ser atingido do que o comportamento de realmente desempenhar bem em campo.

3.6.2.2 Melhorias implementadas

A fim de solucionar o problema apresentado na subseção anterior, uma nova métrica foi adicionada a função de avaliação do desempenho do indivíduo: a cada passo que este mantiver a velocidade dos robôs igual a zero, 50 pontos são descontados. Desta forma, explorar o ambiente torna-se uma opção muito mais interessante do que simplesmente ficar parado enquanto a partida acontece. Com isso, a nova *fitness* avalia todos os pontos citados na seção 3.6.2 com este desconto de pontuação para evitar que os robôs repitam o comportamento citado anteriormente.

3.6.2.3 Novos problemas

A nova metodologia para o cálculo da *fitness* foi capaz de resolver parcialmente o problema. Notou-se que o número de indivíduos que não executavam nenhuma ação em campo se tornou drasticamente menor, mas o algoritmo encontrou uma nova forma de convergir para uma *fitness* de 0 ponto: andando em círculos. Cada robô continua sem interagir diretamente na partida e evita a penalidade implementada por ficar parado. Esse comportamento também permite uma convergência precoce, já que é muito mais fácil do que uma convergência por bom desempenho. Por isso, é necessária uma nova adaptação no cálculo da *fitness*.

Para isso, é necessário entender a movimentação do robô enquanto anda em círculos. Para andar em círculos sem sair do lugar, é necessário que a velocidade de uma das rodas seja 0, enquanto a outra roda acelera o robô, fazendo com que o mesmo gire em torno do próprio eixo. Desta forma, a regra estabelecida foi a seguinte: caso uma das rodas de um robô fique com velocidade igual a 0 por mais de 6 quadros consecutivos, uma penalidade é aplicada. Essa penalidade é reaplicada para cada quadro subsequente que faça com que o comportamento se mantenha. Com isso, a penalidade pode ser aplicada tanto para o caso do robô se manter completamente parado quanto para os casos da movimentação em círculo.

4 RESULTADOS

Este capítulo tem como objetivo apresentar os resultados dos treinamentos da rede neural proposta de acordo com variadas configurações e funções para cálculos de rede.

4.1 Treinamento para seguir a bola

Esta seção tem como objetivo apresentar os resultados do primeiro treinamento utilizando o NEAT para fazer com que os robôs sigam a bola. Neste treinamento, os robôs adversários permanecem estáticos, o que torna o ambiente um pouco menos dinâmico e mais fácil de ser explorado. Contudo, a cada execução, os robôs e a bola são posicionados em locais aleatórios.

Na Figura 18, é possível observar a evolução do algoritmo em uma execução de 500 gerações. No eixo X, tem-se o registro do número de gerações, e no eixo Y, a *fitness*. A linha em azul representa a média da evolução da *fitness* das gerações, e a linha vermelha representa a evolução da *fitness* do melhor indivíduo de cada geração. Como a *fitness* desse treinamento é definida como 0 (zero) caso o robô não alcance a bola e 1 (um) caso o robô atinja a bola, pode-se dizer que, apenas pela análise gráfica, na última geração, a rede neural é capaz de resolver o problema de alcançar a bola em aproximadamente 70% das vezes.

O trabalho de Martins et al. (2021) também implementou redes neurais que efetuam um treinamento para fazer com que os robôs sigam a bola. Os resultados obtidos por ele podem ser vistos na Figura 17. Para obter tais resultados, Martins et al. (2021) utilizaram redes neurais mais complexas, como o Soft Actor Critic (SAC) e o Deep Deterministic Policy Gradient (DDPG), que exigem um maior esforço de configuração e um maior custo computacional para execução. Ao comparar os resultados, é possível notar que a taxa de sucesso das redes mais complexas foi maior, na faixa dos 90%.

A rede gerada pelo NEAT foi capaz de solucionar parcialmente o problema com uma complexidade de desenvolvimento e configuração extremamente reduzida. A rede gerada foi uma rede neural simples e não precisou ser configurada, uma vez que o NEAT foi o responsável pela sua geração. Um ponto importante para garantir uma boa comparação entre os gráficos é o número de passos utilizados por cada um dos algoritmos. Nos resultados obtidos neste trabalho, tem-se o gráfico baseado em número de gerações. Cada geração possui 500 indivíduos que, por sua vez, jogam partidas de até 30 segundos. Como cada passo de uma partida tem a duração de 0,25 segundo, pode-se dizer que foram necessários aproximadamente 30 milhões de passos para obter o resultado presente neste trabalho. Esse número é maior se comparado ao trabalho de Martins et al. (2021), o que é justificado pela diferença de complexidade das redes.

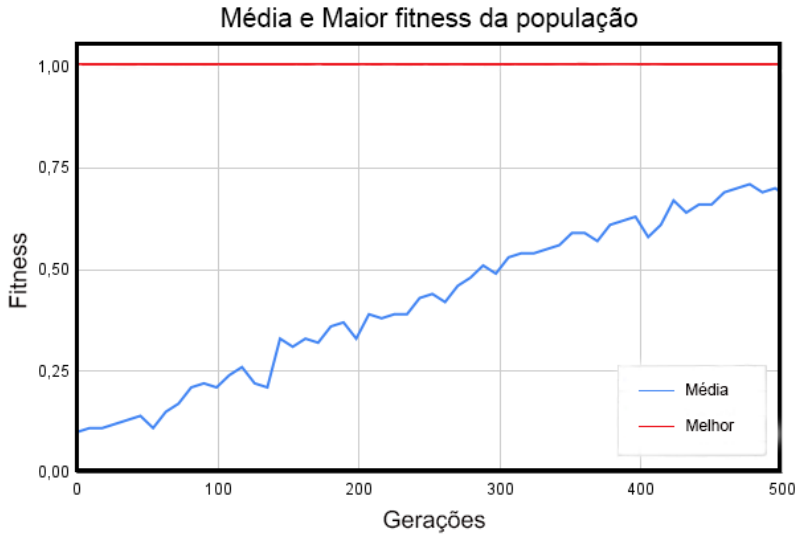

Figura 16 – Resultados do treinamento de rede neural para seguir a bola. Fonte: Elaborado pelo autor.

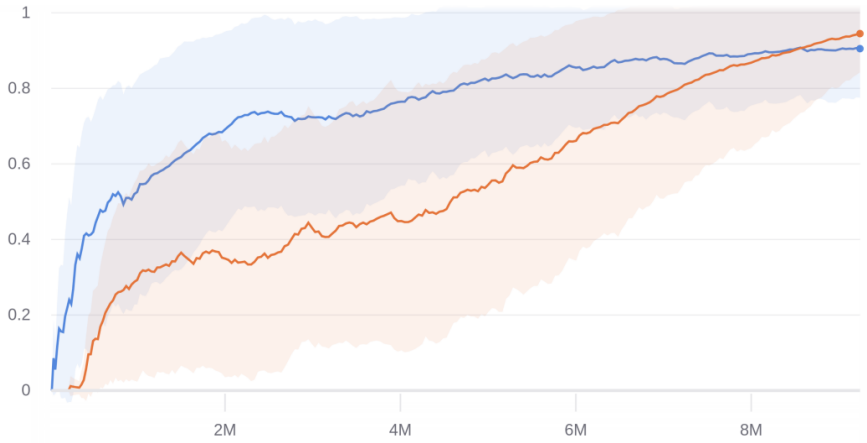

Figura 17 – Resultados do treinamento de Martins et al. (2021) para que o robô siga a bola (DDPG em azul e SAC em laranja). O eixo X representa o número de passos e o eixo Y representa a *fitness*.

4.2 Treinamento de partida completa

4.2.1 Resultados de configuração inicial

Utilizando os parâmetros definidos na seção 3.5 e a avaliação de *fitness* definida na subseção 3.6.2, a Figura 18 apresenta os resultados do treinamento utilizando destes parâmetros. A linha vermelha representa a evolução do melhor indivíduo de cada geração, a linha azul representa a *fitness* média da geração e a linha verde o desvio padrão de cada geração. Na

imagem pode-se notar uma convergência precoce do algoritmo, levando a um resultado onde o melhor indivíduo sempre é o de *fitness* igual a zero. Esse comportamento ocorreu conforme o abordado na seção Problemas identificados, onde um indivíduo que faz com que os robôs fiquem parados se torna o melhor indivíduo da população.

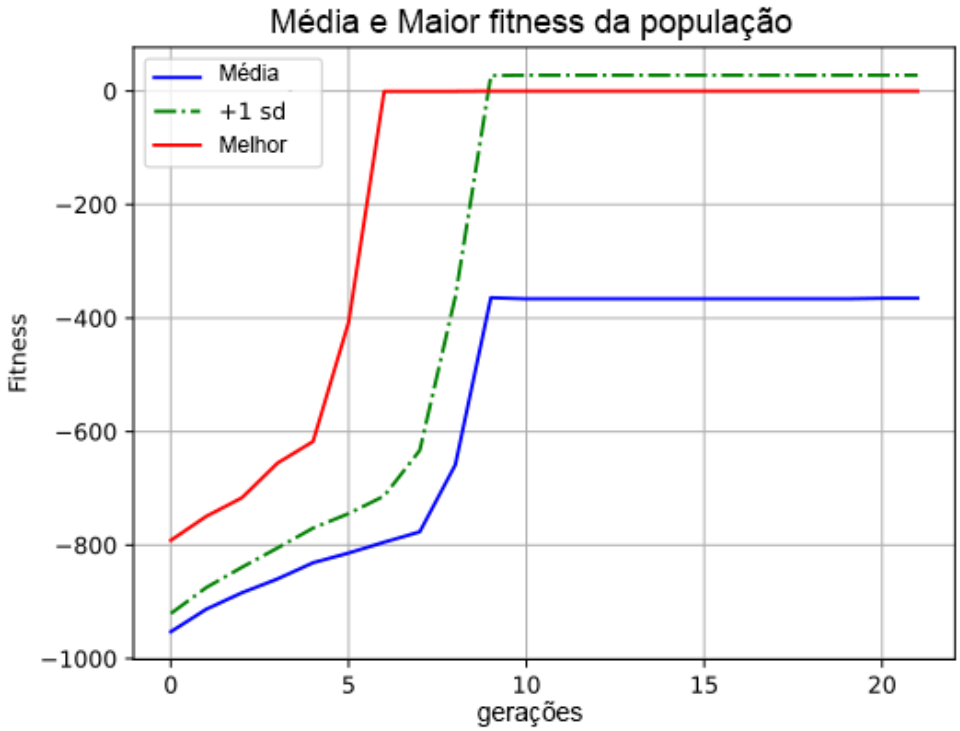

Figura 18 – Primeiro resultado de treinamento da rede Neural. Fonte: Elaborado pelo autor.

Focando em um dos parâmetros do NEAT que é a possibilidade de reiniciar o processo de treinamento no caso de estagnação dos resultados, o segundo teste com estas configurações teve como objetivo permitir um reinício da população após um período de estagnação maior que 15 gerações. A Figura 19 apresenta os resultados deste experimento. Nela é possível verificar o comportamento padrão da população com os parâmetros fornecidos, que é o de sempre convergir para uma *fitness* de zero pontos com indivíduos que fiquem parados. Mesmo após o reinício da população, que aconteceu por volta da geração 25, o comportamento se repete.

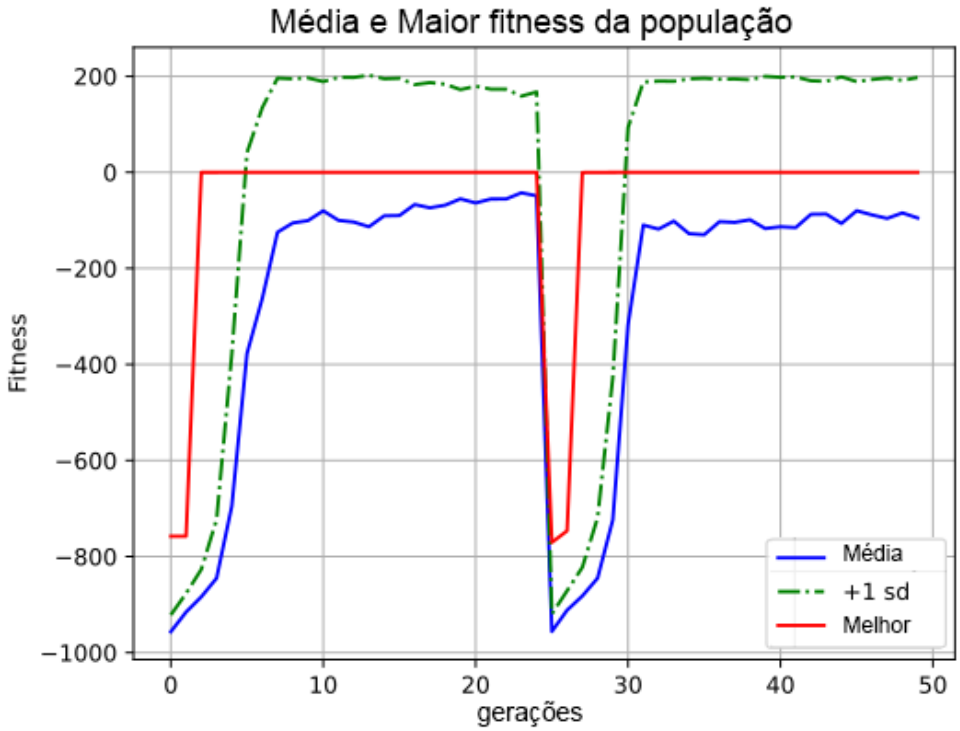

Figura 19 – Treinamento de rede neural com reinício por estagnação. Fonte: Elaborado pelo autor.

4.2.2 Resultados após alteração de *fitness*

Após alterar a função de *fitness*, como citado na subseção 3.6.2.2, pode-se notar um comportamento similar a citado anteriormente. Na Figura 20 fica claro um comportamento similar ao obtido no experimento anterior: uma convergência precoce para uma *fitness* que tende a zero. Desta vez o robô não ficou parado, ele passou a andar em círculos, de forma que sempre se manteve em movimento, ignorando a penalidade referente ao robô parado, porém seguindo sem interagir com o campo.

Ao aplicar as alterações citadas na subseção 3.6.2.3, o comportamento também se manteve, dessa vez com o robô efetuando semi-círculos e alterando seu sentido entre horário e anti-horário regularmente. A Figura 21 mostra que este comportamento também leva para uma *fitness* de zero pontos.

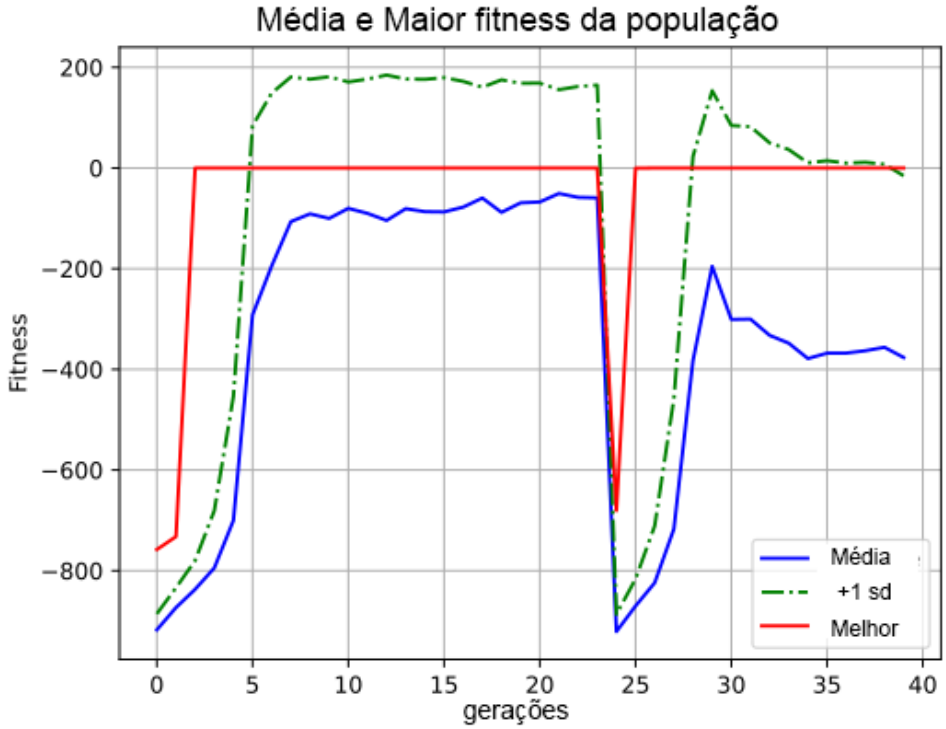

Figura 20 – Treinamento de rede neural com punição por ficar parado. Fonte: Elaborado pelo autor.

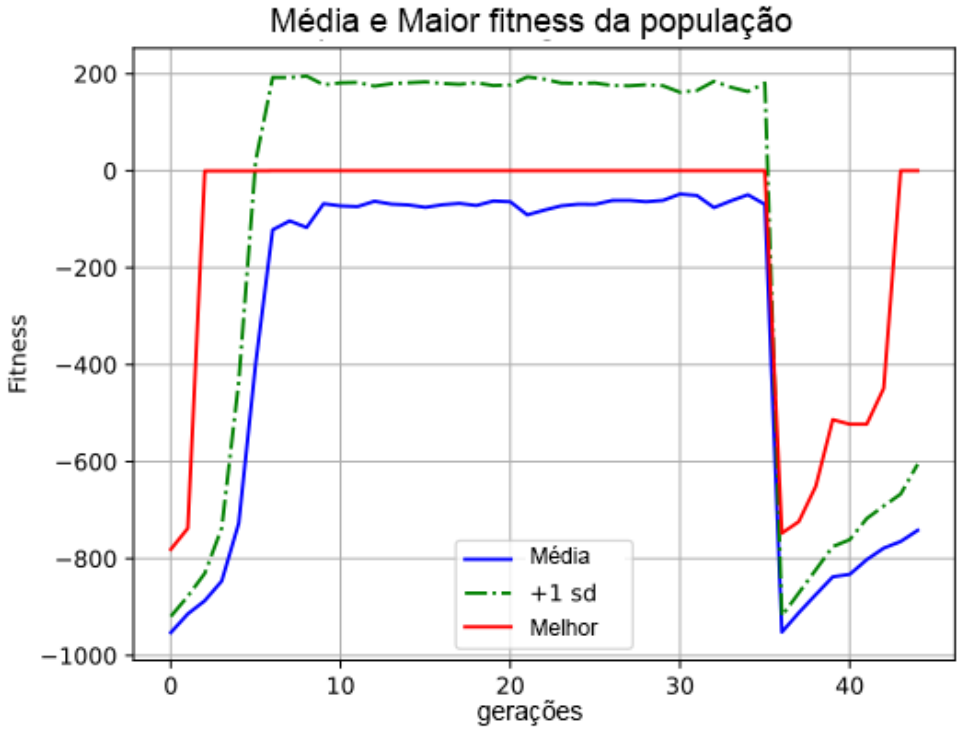

Figura 21 – Treinamento de rede neural com punição por ficar parado e executar círculos. Fonte: Elaborado pelo autor.

4.2.3 Resultados removendo estagnação

Para garantir que o algoritmo tenha tempo para se desenvolver, a propriedade de estagnação foi removida, dando ao algoritmo o tempo necessário para uma possível convergência. Porém, este não foi o resultado obtido. Como pode ser observado na Figura 22, o algoritmo não mostrou nenhum tipo de reação, mantendo a mesma média obtida nos testes anteriores e o melhor indivíduo com uma *fitness* tendendo a zero.

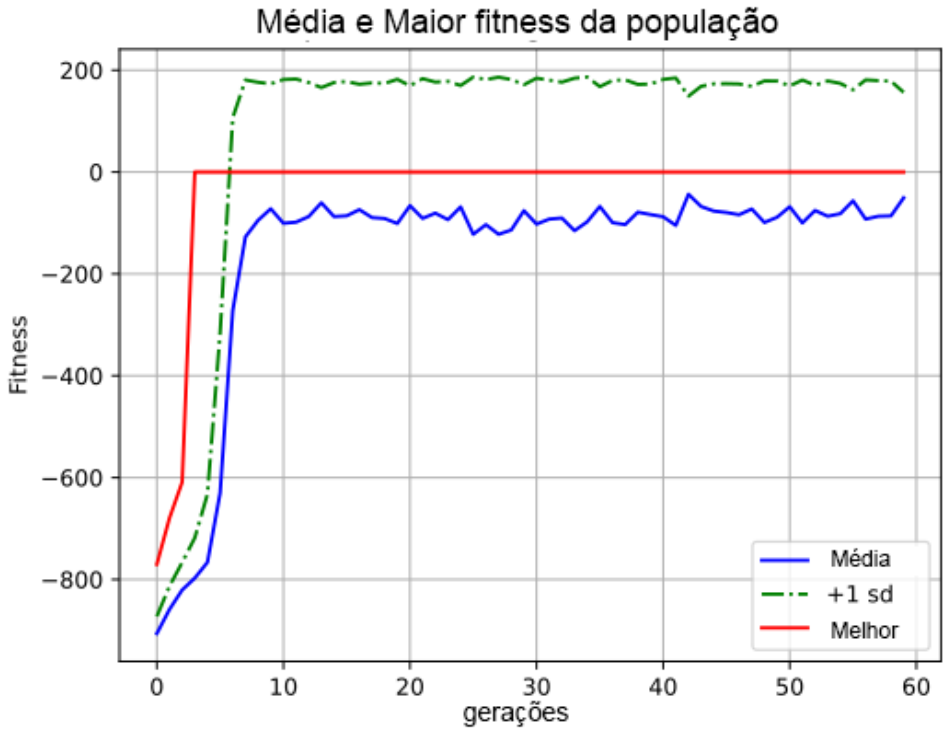

Figura 22 – Treinamento de rede neural sem estagnação após 60 gerações. Fonte: Elaborado pelo autor.

Um teste mantendo a execução por muito mais tempo foi realizado e, ainda assim, a rede neural não foi capaz de solucionar o problema. Nele também foi adicionada uma punição maior por efetuar movimentos circulares. Com isso, a convergência do algoritmo foi para um valor um pouco menor que zero, como pode ser visto na Figura 23 pois a melhor solução ainda foi a de realizar movimentos circulares, porém agora este comportamento recebeu uma pontuação menor.

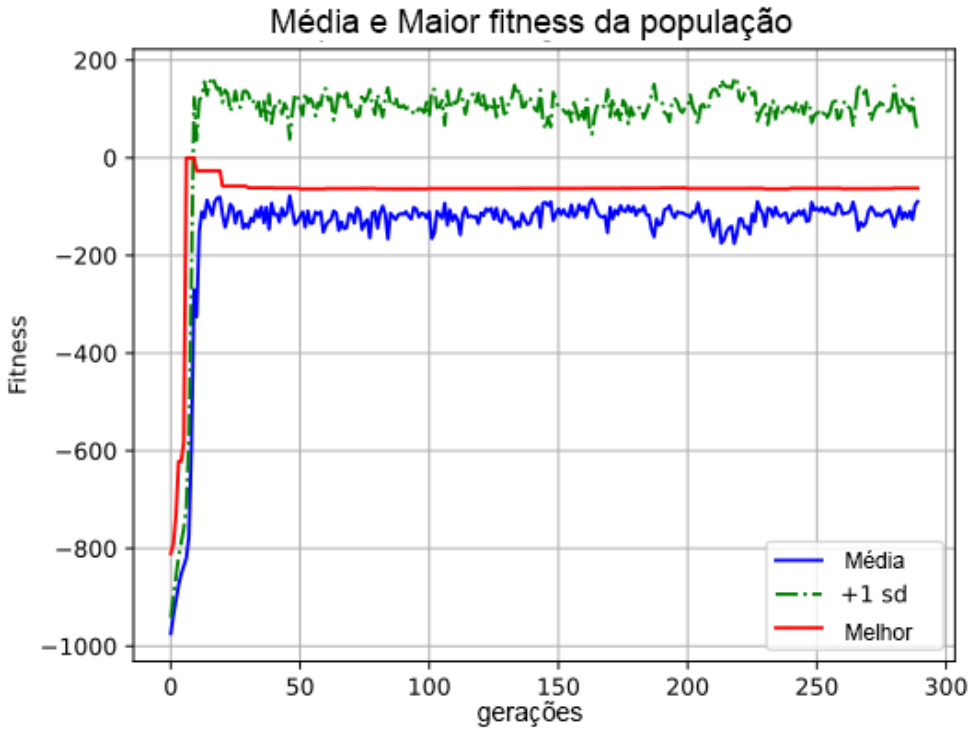

Figura 23 – Treinamento de rede neural sem estagnação após 300 gerações. Fonte: Elaborado pelo autor.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Conclusões

Este trabalho desenvolveu e avaliou uma rede neural baseada no NEAT para resolver problemas no ambiente da categoria *Very Small Size Soccer* da equipe Rodetas Robô Clube. Embora o NEAT tenha apresentado bom desempenho em tarefas simples, como seguir a bola, encontrou dificuldades em tarefas mais complexas, como jogar uma partida completa.

Apesar das limitações encontradas, os objetivos propostos foram alcançados, uma vez que foi criado um ambiente de treinamento de redes neurais integrado à equipe Rodetas e avaliado o desempenho de redes neurais evolutivas em tarefas complexas.

Observou-se que o NEAT gera redes neurais relativamente simples, o que pode limitar seu desempenho em tarefas dinâmicas e complexas. Uma possível abordagem para melhorar o desempenho seria utilizar uma rede neural mais robusta ou quebrar o problema principal em subtarefas menores, permitindo que o NEAT possa resolvê-las de forma mais eficiente.

5.2 Trabalhos Futuros

Como trabalhos futuros, propõe-se a utilização o NEAT na tomada de decisões, fornecendo como saída os pontos onde cada robô deve estar, e deixando para um algoritmo de planejamento de caminhos a tarefa de traçar a rota a ser atingida. Outra possibilidade de trabalho futuro seria explorar outras técnicas de otimização para redes neurais evolutivas, a fim de encontrar topologias mais complexas e eficientes.

Por fim, vale destacar que modelos mais complexos de aprendizado por reforço podem ter a capacidade de resolver a tarefa de partida completa. Desta forma, trabalhos focados em modelos baseados em políticas (policy-based models) ou outros modelos mais robustos podem chegar ao resultado final esperado.

REFERÊNCIAS

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>. Citado na página 32.

FERNANDES, A. M. d. R. *Inteligência artificial: noções gerais*. [S.l.]: Visual Books, 2003. Citado na página 21.

FLECK, L. et al. Redes neurais artificiais: princípios básicos. *Revista eletrônica científica de inovação e tecnologia*, 2016. v. 7, n. 15, p. 47–57, 2016. Disponível em: <https://periodicos.utfpr.edu.br/recit/article/view/4330/Leandro>. Citado na página 22.

HADDADIN, S.; ALBU-SCHAFFER, A.; HIRZINGER, G. The role of the robot mass and velocity in physical human-robot interaction - part i: Non-constrained blunt impacts. In: *2008 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 2008. p. 1331–1338. Citado na página 11.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. Bookman Editora, 2000. ISBN 9788577800865. Disponível em: <https://books.google.com.br/books?id=bhMwDwAAQBAJ>. Citado na página 22.

HOLLAND, J. H. Genetic algorithms. *Scientific American*, 1992. Scientific American, a division of Nature America, Inc., v. 267, n. 1, p. 66–73, 1992. ISSN 00368733, 19467087. Disponível em: <http://www.jstor.org/stable/24939139>. Citado na página 25.

KATOCH, S.; CHAUHAN, S. S.; KUMAR, V. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 2020. Springer Science and Business Media LLC, v. 80, n. 5, p. 8091–8126, out. 2020. Disponível em: <https://doi.org/10.1007/s11042-020-10139-6>. Citado na página 26.

KHOSRAVIKIA, F. et al. *Neural Network-Based Equations for Predicting PGA and PGV in Texas, Oklahoma, and Kansas*. 2018. Citado na página 12.

KRAMER, O. Genetic algorithms. In: ____. *Genetic Algorithm Essentials*. Cham: Springer International Publishing, 2017. p. 11–19. Citado 2 vezes nas páginas 8 e 25.

LUGER, G.; STUBBLEFIELD, W. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Benjamin/Cummings Publishing Company, 1993. (Artificial intelligence). ISBN 9780805347807. Disponível em: <https://books.google.com.br-/books?id=UHVQAAAAMAAJ>. Citado na página 21.

MARTINS, F. B. et al. *rSoccer: A Framework for Studying Reinforcement Learning in Small and Very Small Size Robot Soccer*. 2021. Citado 6 vezes nas páginas 8, 20, 33, 37, 40 e 41.

MCINTYRE, A. et al. *neat-python*. 2015. Citado na página 31.

MENDES, R. *Abordagens evolutivas para otimização de redes neurais convolucionais baseadas em algoritmos genéticos*. Tese (Doutorado), 2021. Disponível em: <https://doi.org/10.14393-/ufu.di.2021.694>. Citado na página 25.

MONAJJEMI, V.; KOOCHAKZADEH, A.; GHIDARY, S. S. grsim - robocup small size robot soccer simulator. In: *RoboCup*. [S.l.: s.n.], 2011. Citado na página 19.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. Citado na página 32.

Paulo Henrique dos Santos. *Planejamento de movimento utilizando campos vetoriais para robôs diferenciais futebolistas*. 2019. Disponível em: http://www.monografias.ufop.br/handle/35400000/2355. Acesso em: 10 de outubro 2022. Citado na página 18.

PENA, C. H. C. et al. An analysis of reinforcement learning applied to coach task in ieee very small size soccer. In: *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. [S.l.: s.n.], 2020. p. 1–6. Citado na página 23.

PINTO, A. H. M. *Regras IEEE Very Small Size Soccer (VSSS)*. 2020. Disponível em: http://200.145.27.208/cbr/wp-content/uploads/2020/07/vssRegras_Portugues.pdf. Acesso em: 10 de outubro 2022. Citado 3 vezes nas páginas 8, 11 e 14.

PINTO Érica S. *Desenvolvimento de robôs capazes de disputar uma partida de futebol da categoria IEEE very small size*. 2017. Disponível em: http://www.monografias.ufop.br/handle/35400000/491. Citado 4 vezes nas páginas 8, 15, 16 e 17.

QUINTERO, C. et al. *Learning Soccer Drills for the Small Size League of RoboCup*. 2014. Disponível em: https://fei.edu.br/rcs/2014/regularpapers/robocupsymposium2014_submission_54.pdf. Acesso em: 08 de outubro 2022. Citado na página 23.

SARACCO, R. *Congrats Xiaoyi. You are now a medical doctor.* 2017. Disponível em: https://cmte.ieee.org/futuredirections/2017/12/02/congrats-xiaoyi-you-are-now-a-medical-doctor/. Acesso em: 10 de outubro 2022. Citado na página 22.

SILVER, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 2018. v. 362, n. 6419, p. 1140–1144, 2018. Citado na página 22.

SINHA, T.; HAIDAR, A.; VERMA, B. Particle swarm optimization based approach for finding optimal values of convolutional neural network parameters. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018. Disponível em: <https://doi.org/10.1109/cec-.2018.8477728>. Citado na página 24.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 2002. v. 10, n. 2, p. 99–127, 2002. Citado 6 vezes nas páginas 8, 13, 24, 26, 27 e 29.

WHITLEY, D.; STARKWEATHER, T.; BOGART, C. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 1990. Elsevier BV, v. 14, n. 3, p. 347–361, ago. 1990. Disponível em: <https://doi.org/10.1016/0167-8191(90)90086-o>. Citado na página 24.

ZHOU, X.-H. et al. Shallow and deep neural network training by water wave optimization. *Swarm and Evolutionary Computation*, 2019. Elsevier BV, v. 50, p. 100561, nov. 2019. Disponível em: <https://doi.org/10.1016/j.swevo.2019.100561>. Citado 2 vezes nas páginas 8 e 24.