Comparison of Reinforcement and Imitation Learning Algorithms in Autonomous Sailboat Digital Twins

Rodrigo P. Méxas, Fabiana R. Leta and Esteban W. G. Clua

***Abstract*—This project aims to study the performance of two reinforcement machine learning algorithms, namely the Proximal Policy Optimization and Soft Actor Critic, in the simulation of autonomous sailboats and their response to different wind directions while avoiding obstacles detected by image analysis and following defined target check-points. Also, the effect of the imitation learning algorithms Behavioral Cloning and Generative Adversarial Imitation Learning combined with the first mentioned algorithms is studied. The proposed scenarios consist of areas filled with random static or moving obstacles and with the presence of favorable or crosswinds. The motivation for the project comes from the lack of studies of the mentioned algorithms in autonomous sailboats, issue which the current study tries to address. The Unity® platform and ML-Agents machine learning toolkit are used for development and the methodology that guides the project can be similarly applied to other reinforcement learning problems. Through agent training, it is possible to compare the results and observe that the Proximal Policy Optimization obtains better performance within the proposed scenarios, both with and without the support of imitation learning algorithms.**

***Index Terms*—reinforcement learning, imitation learning, autonomous sailboat, unmanned surface vehicle.**

I. Introdução

O uso de técnicas de inteligência artificial tem se tornado padrão para programação e desenvolvimento de robôs e veículos autônomos [1]. Recentemente, técnicas de aprendizagem por reforço tem ganhado bastante espaço dentro da área de Inteligência Artificial, demonstrando-se bastante eficientes para a programação de robôs e veículos autônomos. Embora haja uma ampla literatura atualmente sobre sua aplicação nestes contextos, ainda há pouca literatura no que se refere ao seu uso em veleiros autônomos.

No contexto de veleiros autônomos, encontram-se referencias e pesquisas com abordagens distintas, tais como controladores baseados em lógica *fuzzy* [2] [3]. Estes trabalhos mostram soluções eficientes para embarcações que devem atingir um alvo de curta distância [4], tendo sido também propostos algoritmos de planejamento de caminho reativo [5].

O aprendizado por reforço é um método baseado em agentes que são capazes de descobrir o que deve ser feito por tentativas e erros, mediante interação com o ambiente [6]. Assim sendo, para evitar danos a um veículo real, torna-se importante o uso de gêmeos digitais para o treinamento, antes que o aprendizado seja transferido ao modelo físico real. Gêmeos digitais são representações dinâmicas virtuais que representam um objeto ou processo físico [7].

Após realizar-se a pesquisa bibliográfica, observa-se a escassez de aplicações que estudam os algoritmos de aprendizado por reforço *Proximal Policy Optimization* [8] (PPO) e *Soft Actor Critic* [9] (SAC) e algoritmos de aprendizado por imitação em barcos a vela. Algoritmos como *Q-Learning*, *Deep Q-Learning* [10], *Deep Deterministic Policy Gradient* [11] (DDPG), *Deep Sarsa* [12] e *Asynchronous Advantage Actor-Critic* [13] (A3C) foram encontrados na literatura. Porém, a maioria dessas aplicações foram desenvolvidas em barcos autônomos motorizados e sem vela, como será apresentado na seção II.

Deste modo, o estudo aqui proposto busca simular o uso do PPO e do SAC em veleiros autônomos considerando quatro ambientes virtuais distintos, constituídos por cenários com obstáculos estáticos e móveis para o vento favorável e o vento de través, e por fim verificar qual solução apresenta o melhor desempenho para os desafios apresentados. Também é experimentado o uso do aprendizado por imitação com os algoritmos *Behavioral Cloning* (BC) e *Generative Adversarial Imitation Learning* [14] (GAIL) junto ao PPO e SAC.

PPO e SAC tem se tornado as técnicas mais recorrentes para abordagens de aprendizagem com reforço e aprendizagem por imitação. Diferentes ferramentas para construção de gêmeos digitais implementam e são atualizados com as mesmas, tal como ocorre com a plataforma *Unity*® [15] e o pacote *ML-Agents* [16]. A implementação deste trabalho adotou esta ferramenta e, portanto, utilizamos estas soluções listadas.

O projeto foi baseado na simulação gráfica de Lytsus [17] e se difere por implementar as inteligências artificiais no veleiro, pela criação dos cenários propostos e pela utilização apenas da vela mestra. O ponto forte da pesquisa é a análise do desempenho de algoritmos pouco estudados até o momento na literatura para veleiros autônomos. Entretanto, uma simulação que reproduz melhor a realidade, com forças hidrodinâmicas mais realistas e uma dinâmica mais precisa do veleiro, é

R. P. Méxas is with the undergraduate program in Mechanical Engineering, Federal Fluminense University (UFF), Niterói, Brazil (e-mail: rodrigopicinini@id.uff.br)

F. R. Leta is with the Mechanical Engineering Department, Federal Fluminense University (UFF), Niterói, Brazil (e-mail: fabianaleta@id.uff.br)

E. W. G. Clua is with the Computer Science Department, Federal Fluminense University (UFF), Niterói, Brazil (e-mail: esteban@ic.uff.br).

necessária para uma aplicação satisfatória no mundo real, sendo esta a maior limitação do estudo. Ao final da pesquisa, observou-se que o algoritmo PPO obteve o melhor desempenho.

II. Trabalhos relacionados

Meyer et al. [18] apresentam um estudo de barcos autônomos em um ambiente simulado sem vento. Os autores definem um caminho pré-determinado ao mesmo tempo em que a embarcação deve desviar de obstáculos estáticos, utilizando-se de vários sensores telêmetros para detecção em simulação bidimensional. Para desviar de tais obstáculos, utiliza-se o PPO, que apresenta bons resultados, mesmo em situações onde há uma densa quantidade de obstáculos.

Zhou et al. [19], além de estudarem o desvio de obstáculos em ambientes de mar calmo e sem influências externas, abordaram a navegação de barcos em formação, ou seja, múltiplos barcos que navegam em conjunto. Os autores utilizaram o planejamento de movimento, baseado em *Deep Q-Learning* [10] e ambiente de simulação bidimensional. O planejamento de movimento, ou caminho, é o cálculo de um caminho livre de colisão de um ponto inicial até um ponto final, de acordo com a avaliação no ambiente de obstáculos [20].

Woo, Yu e Kim [21] treinaram o barco de forma que ele aprendesse a seguir uma trajetória em um simulador e depois o testaram em ambiente real. O algoritmo DDPG foi utilizado para treinar o controlador de direção. Silva Junior et al. [22] estudaram o planejamento de caminho e desvio de obstáculos estáticos para um veleiro utilizando o algoritmo *Q-Learning*, dando importância para o efeito do vento no veículo. Neste trabalho foram encontrados bons resultados para todas as direções do vento e novos caminhos conseguem ser gerados conforme há a mudança de direção do vento, em tempo real, sendo que a simulação do mundo real foi feita em uma estrutura matricial.

Wang et al. [23] simularam um barco em um ambiente tridimensional no *Unity*®, utilizando imagens coloridas como dados de entrada e testaram o modelo em ambientes com diferentes climas, demonstrando grande impacto no desempenho de acordo com o visual do ambiente. Os autores utilizaram o algoritmo *Prior knowledge-based USV reinforcement learning obstacle avoidance* [23], sendo este baseado no PPO.

Polvara et al. [24] simularam o pouso de um veículo aéreo não-tripulado em um veículo de superfície não-tripulado submetido a perturbações da corrente do mar. Neste cenário, apenas o veículo aéreo foi treinado, com a utilização de duas redes neurais baseadas em *Deep Q-Learning*, uma rede para a detecção do veículo de superfície e outra para a descida vertical.

Lin e Guo [25] utilizaram a plataforma *Pygame* para testar o planejamento de caminho na navegação marítima em um ambiente 2D. Para isso, os algoritmos *Q-Learning* e *Q-Learning* baseado em otimização de estratégia são utilizados e comparados entre si na geração dos caminhos, demonstrando a superioridade do Q-Learning aprimorado em cenários complexos.

Zhou et al. [26] utilizaram o algoritmo A3C [13] aprimorado e integrado à GPU, com a simulação da navegação em ambientes tridimensional, como labirintos e obstáculos estáticos não aleatórios e labirintos aleatórios. A observação foi feita por imagens e foi demonstrada a superioridade do algoritmo proposto com o A3C tradicional.

Na pesquisa de [27], o algoritmo COLREGs *inteligent collision avoidance* (CICA) foi proposto e a embarcação foi treinada em um ambiente tridimensional construído no *Unity*® em que deveria desviar de obstáculos dinâmicos. O algoritmo CICA foi comparado com os algoritmos *Artificial Potential Field* (APF) [28] e *Velocity Obstacle* (VO) [29], e a capacidade de escolher caminhos mais curtos do CICA foi verificada.

Wang et al. [30] estudaram múltiplos barcos que navegam em conjunto e devem alcançar os pontos definidos enquanto mantêm a formação. O algoritmo DDPG foi utilizado em tal estudo.

Wu et al. [31], propuseram o algoritmo *Autonomous Navigation and Obstacle Avoidance* (ANOA) e o comparou com o *Deep Q-Learning* e o *Deep Sarsa* [12], mostrando a superioridade do ANOA em relação a eles em um cenário com obstáculos estáticos móveis e observação por imagens.

Meyer et al. [32] estudaram a navegação de um barco com rota pré-definida e capaz de desviar de obstáculos estáticos e móveis gerados aleatoriamente. O algoritmo implementado foi o PPO e três ambientes digitais foram construídos baseados na área real Trondheim Fjord.

III. Fundamentação teórica

Para esta pesquisa foram usadas técnicas de *Proximal Policy Optimization, Soft Actor Critic e* Algoritmos de Imitação, conforme detalhados na sequência.

A. Proximal Policy Optimization

O PPO é um método baseado em ator-crítico de vantagem proposto por Schulman et al. [8] e desenvolvido originalmente pela *Open AI*. Este método já foi aplicado, por exemplo, em Problemas de Reabastecimento em Conjunto [33]. A ideia do algoritmo consiste em evitar que uma atualização de política seja muito expressiva e cause distúrbios e erros no cálculo do modelo.

O algoritmo utiliza um método chamado de *clip* para evitar que a atualização da política se desvie demasiadamente da anterior. Se os valores são menores ou maiores que os extremos determinados, estes ganham o valor do extremo ultrapassado. A função de perda do algoritmo, que ilustra a ideia apresentada, é dada abaixo:

$$L^{CLIP}(\theta) = E_{\tau\sim\pi}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t,\ clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A_t\right) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \quad (1)$$

em que: θ é a parametrização da política e representa os pesos de uma rede neural, t é o passo de tempo, τ é a trajetória, A é a vantagem, π é a política e representa uma probabilidade, *old* denota a política de pré-atualização, a é a ação, s é o estado, ϵ é um hiper parâmetro que regula o tamanho da atualização da razão probabilística advinda da nova e da antiga política, L_t^{VF} é uma perda de erro quadrático, S é um bônus de entropia e, c_1 e c_2 são constantes.

B. Soft Actor Critic

Sua conceitualização foi dada por Haarnoja et al. [9]. A ideia por trás deste algoritmo é de maximizar a recompensa da política, bem como maximizar a entropia da política. A entropia elevada estimula a exploração do agente, pois evita que o agente siga sempre por um mesmo caminho já descoberto. A imposição da aleatoriedade estimula a saída de uma tática gananciosa. A eficiência da política que deve ser maximizada, e que ilustra a ideia principal do algoritmo, é dada pela equação abaixo:

$$J(\pi_\theta) = E_{\tau\sim\pi}\big[r(\tau) + \mu H\big(\pi(.\,|s_t)\big)\big], \tag{2}$$

em que: r é a recompensa, H é a entropia e μ é o coeficiente de compensação, que indica a importância da entropia em relação à recompensa e determina o quanto a política deve ser estocástica.

C. Algoritmos de Imitação

Dentro do universo de aprendizado por imitação é possível citar duas abordagens: o BC e o GAIL [14]. O BC consiste em tentar imitar exatamente a demonstração fornecida e é o algoritmo de imitação mais simples aqui estudado. Por sua simplicidade, ele é comumente utilizado como um pré-treino para que o algoritmo principal, usado após ele, já conheça as recompensas do ambiente. Já o GAIL é uma técnica baseada em *Inverse Reinforcement Learning* (IRL). A ideia do IRL é aprender a função de recompensa baseada nas demonstrações feitas por experientes.

O GAIL, além de ser um algoritmo de IRL, também é baseado em *Generative Adversarial Nets* [34] (GAN). O GAN consiste em dois modelos: o gerador e o discriminador. O gerador é um modelo gerado pela utilização da distribuição de dados e o discriminador é um modelo que calcula a probabilidade de a amostra ter vindo dos dados recebidos como entrada ao invés do gerador. Desse modo, o meio para se treinar o gerador é a tentativa de maximizar a probabilidade de o discriminador cometer um erro. Assim, no GAIL, o gerador tenta estimular o erro do discriminador para que seja confundido com os dados fornecidos pela demonstração do experiente. Com isso, uma função de recompensa e política são geradas. No *ML-Agents*, o GAIL é usado como uma forma de recompensa intrínseca e precisa ser utilizado em conjunto com o PPO ou o SAC. Ou seja, enquanto o PPO ou SAC aprendem suas políticas com as recompensas extrínsecas, que são recompensas obtidas ao explorar o ambiente, o GAIL recompensa o modelo por seguir as demonstrações fornecidas e ajuda a moldar as funções de recompensa e políticas.

IV. Metodologia

A. Ferramentas Utilizadas

Para a implementação das técnicas propostas foram utilizados o *Unity*® e o *ML-Agents*. O *Unity*® é uma plataforma de desenvolvimento de jogos, mas que também é utilizada para simulações. O *Unity*® *Machine Learning Agents Toolkit*, também conhecido como *ML-Agents*, é um *toolkit* de código aberto que implementa soluções de aprendizado de máquina ao *Unity*® por meio de uma *API* em *Python*, com implementações em *PyTorch*® [35], que é uma biblioteca de aprendizado de máquina. Juliani et al. [36] argumentam a efetividade e inovação do *Unity*® como plataforma de desenvolvimento para inteligência artificial. Essas ferramentas são interessantes, pois o *ML-Agents* traz algoritmos de aprendizado por reforço e imitação já implementados para serem utilizados em ambientes ricos construidos no *Unity*®.

Para este projeto, a plataforma de desenvolvimento utilizada foi o *Unity*® em sua edição gratuita, versão 2020.3.8.f1. Realizou-se a programação em C# dos scripts no *Visual Studio*® 2019 [37], que é a linguagem de programação para o *Unity*®. O *toolkit* do *ML-Agents* utilizado estava em sua *release* 17. O pacote usado dentro do *Unity*® estava na versão 2.0.0, e, o pacote utilizado no *Python* era a versão 0.26.0. O *Pytorch*® 1.7.1 foi utilizado.

Com o objetivo de comparar o uso de ações discretas e contínuas junto ao algoritmo SAC e validar uma hipótese, como será discutido na seção V, houve uma mudança nas versões utilizadas em decorrência de um erro no cálculo da entropia para ações contínuas na *release* 17 do *ML-Agents*. Assim, para este caso específico, do SAC com ações contínuas, foi utilizado o *ML-Agents* na *release* 18. Nesta situação, há o pacote no *Unity*® na versão 2.1.0, o pacote *Python* na versão 0.27.0, e o *Pytorch*® na versão 1.9.0.

B. Configuração do Ambiente de Treinamento

Com o intuito de partir de uma simulação gráfica já desenvolvida, e fazer a implementação de algoritmos de inteligência artificial, utilizou-se o projeto de código aberto [17]. Esta simulação é constituída de um veleiro, com uma resposta física simplificada ao vento e ao movimento da água. Colocou-se apenas uma vela (vela mestra) no veleiro, já que se pretende criar um gêmeo digital para barcos que tenham esta característica.

O treinamento ocorreu de maneira episódica. Um episódio se caracteriza como uma sequência de estados, ações e recompensas que sempre possuem um fim. Ou seja, caso algum gatilho seja acionado, o episódio termina e outro é iniciado, reiniciando os estados, ações e recompensas acumuladas. Em um treinamento de episódio, o agente enfrenta vários episódios no decorrer do mesmo. A reinicialização constante do ambiente com aleatoriedades no cenário torna o uso de episódios um grande atrativo para generalizar o modelo para ambientes mais diversos.

Na área de treinamento, fronteiras foram posicionadas nos limites da esquerda, da direita e de trás. De forma a marcar a região para o qual o barco deveria alcançar, uma fronteira foi posicionada à frente. Essas fronteiras são invisíveis e são acionadas em forma de gatilho, ou seja, quando houver a interseção do agente com as fronteiras, este evento é detectado e o episódio é reiniciado. A representação das fronteiras pode ser vista na Fig. 1.

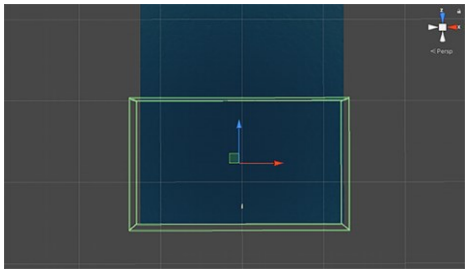

Fig. 1. Representação das fronteiras no ambiente de treinamento.

Dois ambientes de treinamento foram criados. São eles ambientes com obstáculos estáticos e obstáculos móveis. A obtenção dos modelos dos obstáculos estáticos foi feita através do site *BlenderNation* [38] e os modelos das rochas utilizadas como obstáculos foram modelados por Gyzen [39]. Os obstáculos móveis foram obtidos no site *cgtrader* com os barcos pescadores modelados pelo usuário Seemlyhasan [40].

No cenário de obstáculos estáticos utilizaram-se nove obstáculos para situações em que o vento estava a favor do movimento do barco e cinco obstáculos quando o vento estava em través. Esta decisão foi feita, pois o agente possuía maior dirigibilidade com vento a favor de seu movimento do que quando estava em través. A cada início de episódio os obstáculos surgiam em posições aleatórias e em rotações aleatórias que podiam tomar de 0º a 360º. Essa aleatoriedade tem o intuito de evitar que o agente se vicie em cenários específicos e tenha dificuldade de vencer cenários mais diversos, evitando-se desta maneira o *overfitting*. Um exemplo de cenário de obstáculos estáticos em situação de vento em popa pode ser conferido na Fig. 2.

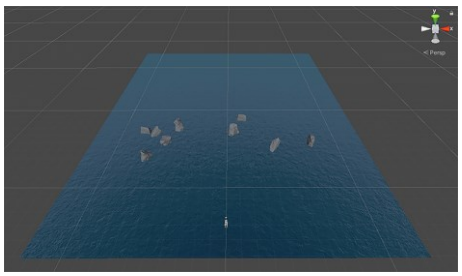

Fig. 2. Exemplo de cenário aleatório com obstáculos estáticos gerado para uma a situação de vento favorável, ou seja, com nove obstáculos.

No ambiente de obstáculos móveis, três obstáculos foram utilizados, tanto para o vento a favor do movimento, quanto para o vento de través, com o surgimento aleatório à esquerda ou à direita. O surgimento de obstáculo no centro do cenário não foi interessante, pois, devido ao avanço constante em direção às fronteiras laterais, o mesmo se distanciaria demais do agente ao longo do episódio. O número menor de obstáculos, se comparado com o ambiente de cenários estáticos, teve o objetivo de balancear a dificuldade do desafio. Os obstáculos, também, receberam velocidades aleatórias dentro de uma faixa determinada. No cenário de vento de través, essas velocidades foram reduzidas, pois nesta situação o veleiro tem uma velocidade reduzida e dirigibilidade menor. A Fig. 3 representa um cenário com obstáculos móveis.

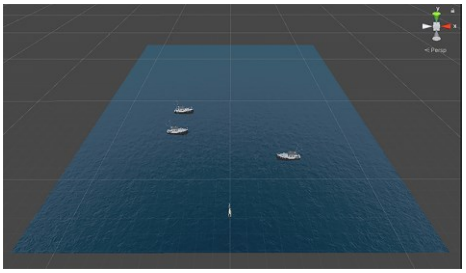

Fig. 3. Exemplo de cenário aleatório gerado para obstáculos móveis.

C. *Observações, Atuadores e Recompensas*

Para a detecção dos obstáculos, foi utilizada a observação por imagens. Uma câmera foi usada para alimentar a rede neural convolucional do agente e sua posição referente ao barco pode ser verificada na Fig. 4. Redes neurais convolucionais automaticamente identificam características relevantes e são utilizadas em áreas como visão computacional, processamento de fala e reconhecimento de face [41]. No estudo, a imagem do ambiente tridimensional captada pela câmera é dada como entrada a cada novo quadro da simulação. Por não ser necessária uma resolução muito elevada para detectar a presença de obstáculos, utilizaram-se imagens com resolução de 84x47 pixels. Além disso, as imagens foram convertidas para tons de cinza, já que as cores não são necessárias para a detecção destes obstáculos. A rede convolucional considerou duas camadas convolucionais, por não ser preciso um reconhecimento de imagens muito elaborado, reduzindo assim o custo computacional. A câmera utilizada possui um campo de visão de 60º, sendo este valor responsável por alterar a amplitude da cena detectada, conforme ilustrado na Fig. 5. Na Fig. 6 é possível visualizar a exibição da câmera em tons de cinza, representando a imagem que é alimentada na rede neural artificial convolucional.

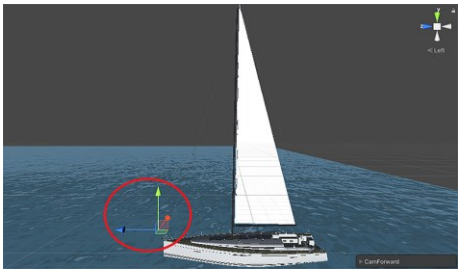

Fig. 4. Posição da câmera referente ao barco.

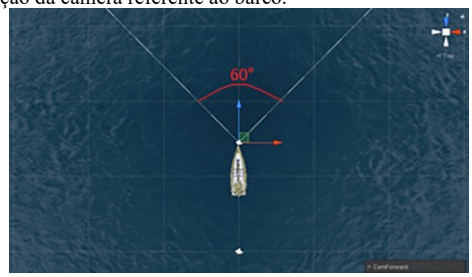

Fig. 5. Representação visual do campo de visão da câmera.

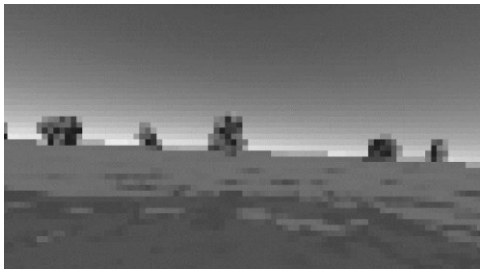

Fig. 6. Exibição da câmera em escala de cinza.

Na situação em que há obstáculos móveis, houve a adição de mais um sensor de observação. No *Unity*® pode-se incluir uma componente conhecida como "sensor de percepção por raios". Por meio desta, é possível calcular a distância do emissor do raio e do objeto atingido, o que permite fazer uma analogia com o funcionamento de um Lidar, que também calcula esta distância. Foram, então, adicionados estes sensores de percepção por raios nas duas laterais do veículo. Essa configuração se mostrou necessária para prever a aproximação de obstáculos pelas laterais. A Fig. 7 mostra a representação dos sensores nas laterais com o campo de visão da câmera também marcado.

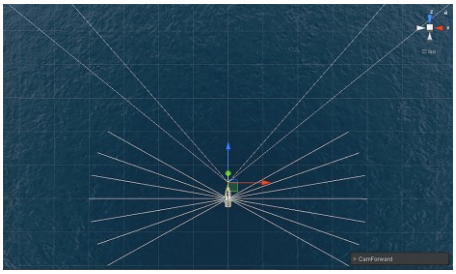

Fig. 7. Representação visual dos sensores de percepção por raios e do campo de visão da câmera.

Além da observação visual, as seguintes observações foram fornecidas ao agente: distância entre o agente e a chegada; balanço e cabeceio do veleiro; sinal do produto escalar entre o eixo longitudinal do agente e o eixo longitudinal da chegada, ou seja, se o veleiro aponta para a chegada ou não; sinal da velocidade do agente no eixo longitudinal; direção do vento; ângulo de rotação da vela; rotação do leme. Os eixos do veleiro estão representados na Fig. 8. e os eixos da chegada estão representados na Fig. 9. Com base nas recompensas recebidas, o agente aprendeu os valores das observações que trazem os maiores retornos.

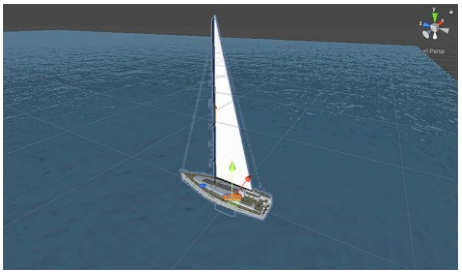

Fig. 8. Eixos do veleiro.

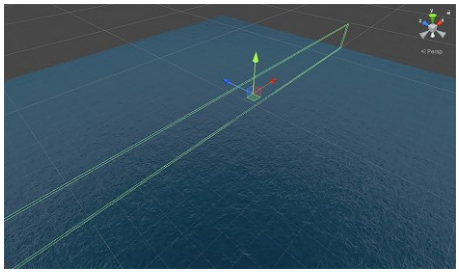

Fig. 9. Eixos da chegada.

Com a contabilização das observações citadas, há oito observações definidas à parte da observação visual. Ressalta-se que os valores foram normalizados, podendo estar no alcance de -1 e 1. Para obter os valores normalizados, dividiu-se a variável pelo seu maior valor possível. Por exemplo, a distância até a chegada normalizada consiste na divisão da distância atual do barco pela sua distância máxima, ou seja, a distância inicial. A rotação por outro lado, consiste na sua rotação momentânea dividida por 360º. Entretanto, ao revisar o projeto, percebeu-se que os ângulos fornecidos vieram de um quatérnio, ao invés do ângulo de Euler tradicional. Cada componente de um quatérnio possui o valor modular máximo igual a 1, e, ao ser feita a divisão por 360º, os números saem do intervalo de -1 e 1. Contudo, esta questão não prejudicou o aprendizado, pois a escala do alcance obtido não se difere muito da escala do intervalo de -1 e 1, que é uma boa prática, porém não obrigatória, recomendada pelo pacote *ML-Agents*

As ações do agente podem tomar forma contínua ou discreta, dependendo da aplicação. Para este trabalho, foram utilizadas ações discretas. São elas: rotação da vela em 1° ou 3° em sentido horário ou anti-horário; não rotacionar a vela; rotação do leme em sentido horário ou anti-horário; não rotacionar o leme.

Assim, contabilizando todos os casos, há um total de oito possíveis ações que podem ser tomadas por passo de tempo. Também, é interessante mencionar que o movimento do leme dita a direção do veleiro e a vela se ajusta para obter a eficiência máxima trazida pela força do vento. Assim, destaca-se que o efeito de rotação do leme foi caracterizado pela aplicação de uma força constante no extremo anterior do veleiro.

Nas ocasiões em que houve o alcance do objetivo, que é o atingimento da chegada, ou a colisão com obstáculos e fronteiras, uma recompensa era recebida e o episódio reiniciava-se. Definiram-se as recompensas, conforme recomendado pelo pacote *ML-Agents*, da seguinte forma: +1 para o alcance no objetivo; -0.5 para a colisão em algum obstáculo; -1 para a colisão em alguma fronteira; -0.5 se o agente não apontasse para a chegada; -0.5 se o balanço do veleiro ultrapassasse 90º; -0.0001 para cada passo de tempo de velocidade negativa do agente. Com exceção do caso da velocidade negativa, todo recebimento de recompensa aqui definido foi utilizado como gatilho para reinicialização dos episódios.

D. Treinamento

Após a definição das observações, atuadores e recompensas, a etapa do treinamento pôde ser iniciada. Com os parâmetros configurados, os testes foram feitos, e, por meio de um processo

iterativo de eventuais não-convergências, características do ambiente eram ajustadas. O treinamento ocorreu para o vento em popa e de través para desafios com obstáculos estáticos e móveis, totalizando quatros cenários para cada algoritmo. Os algoritmos PPO e SAC, além de serem utilizados sozinhos, também foram utilizados com técnicas de aprendizado por imitação, o BC, como um pré-treino e o GAIL. Assim, os modelos utilizados foram: PPO, SAC, PPO + BC + GAIL e SAC + BC + GAIL. No pacote *ML-Agents*, o algoritmo GAIL pode apenas ser utilizado em conjunto com o PPO e o SAC. Todos os treinamentos ocorreram ao longo de 5.000.000 passos de tempo, e quando houve a utilização do BC, seu uso perdurou por 150.000 passos de tempo, tendo sido fornecidos 10 episódios para a solução de imitação, em que a gravação desses episódios foi feita dentro do *Unity®*. Por meio de arquivos de configuração de formato *yaml*, determinaram-se os parâmetros, os algoritmos utilizados e a combinação dos algoritmos de reforço com os de imitação. A Tabela I representa um esquema dos algoritmos e cenários estudados. Para o treinamento de cada cenário, foram utilizadas redes neurais artificiais diferentes, pois verificou-se experimentalmente que houve dificuldade de uma mesma rede aprender para direções de vento diferentes. Com o intuito de analisar as recompensas, gráficos que retratam o desempenho dos treinamentos foram gerados no *Tensorboard®*.

TABELA I
ESQUEMA DOS ALGORITMOS E CENÁRIOS ESTUDADOS

Algoritmos	Obstáculos		Vento	
PPO SAC PPO + BC + GAIL SAC + BC + GAIL	Estáticos	Móveis	Em popa	De través

V. RESULTADOS

Os resultados obtidos estão divididos a seguir conforme as situações estudadas: a) Vento a favor com obstáculos estáticos; b) vento de través com obstáculos estáticos; c) vento a favor com obstáculos móveis e d) vento de través com obstáculos móveis. A Fig. 10 mostra um desenho esquemático dos cenários listados. Os pontos representados nos gráficos das subseções *A*, *B*, *C* e *D* são as médias de recompensas obtidas do ambiente a cada 20.000 passos. Além dos gráficos que comparam os algoritmos estudados entre si, estão também demonstrados gráficos que trazem as recompensas obtidas por meio do uso de ações contínuas do SAC. Esta abordagem foi necessária, pois durante o treinamento do SAC, observou-se um comportamento irregular, que é contrário à teoria proposta para tal algoritmo, levantando-se a hipótese que tal problema podia estar relacionado ao tipo ação utilizada. Para confirmar esta hipótese, estudou-se o seu comportamento diante de ações contínuas, que representa a modelagem original para o algoritmo. Com isso, notou-se a regularização de seu comportamento e teve-se a confirmação da hipótese. Esta utilização de ações contínuas consistia no agente gerar valores contínuos entre -3 a 3 para o movimento da vela e -1 a 1 para a rotação do leme. Dos valores gerados, sua parte inteira era utilizada, logo, a vela possuiu movimentos de 0°, 1°, 2° e 3° para ambos os sentidos e o leme podia rotacionar no sentido horário e anti-horário ou não rotacionar, assim como definido para ações discretas.

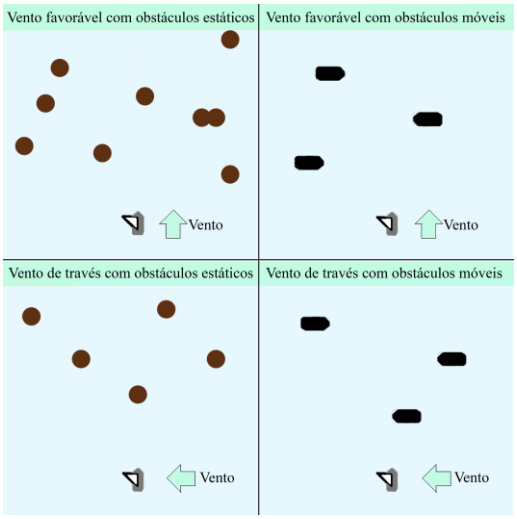

Fig. 10. Desenho esquemático dos quatro cenários estudados.

A. *Vento a Favor com Obstáculos Estáticos*

A Fig. 11 mostra as médias de recompensas obtidas ao longo do treinamento para o cenário de vento a favor com obstáculos estáticos e a Fig. 12 traz as médias do SAC treinado com ações contínuas.

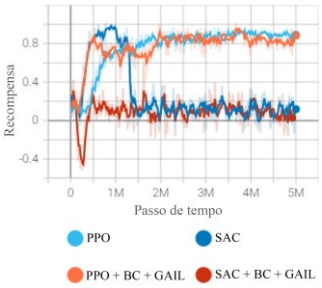

Fig. 11. Gráfico das médias de recompensas para o cenário de vento a favor com obstáculos estáticos.

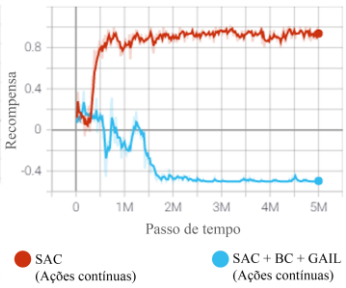

Fig. 12. Gráfico das médias de recompensas para o cenário de vento a favor com obstáculos estáticos com o uso do SAC para ações contínuas.

As últimas médias de recompensas obtidas para o PPO, PPO + BC + GAIL, SAC e SAC + BC + GAIL respectivamente foram: 0.9114, 0.8977, 0.1654, 0.09069. E para o SAC e SAC + BC + GAIL com ações contínuas foram: 0.9314 e -0.5012.

B. *Vento de Través com Obstáculos Estáticos*

A Fig. 13 mostra as médias de recompensas obtidas ao longo do treinamento para o cenário de vento de través com obstáculos estáticos e a Fig. 14 traz as médias do SAC treinado com ações contínuas.

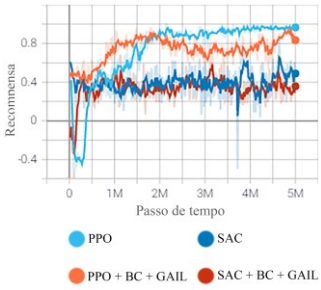

Fig. 13. Gráfico das médias de recompensas para o cenário de vento de través com obstáculos estáticos.

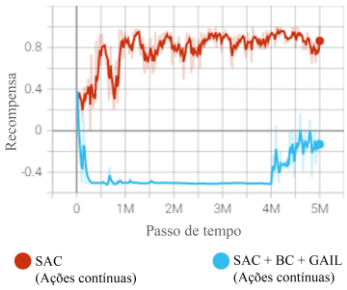

Fig. 14. Gráfico das médias de recompensas para o cenário de vento de través com obstáculos estáticos com o uso do SAC para ações contínuas.

As últimas médias de recompensas obtidas para o PPO, PPO + BC + GAIL, SAC e SAC + BC + GAIL respectivamente foram: 0.9659, 0.7899, 0.4992, 0.4317. E para o SAC e SAC + BC + GAIL com ações contínuas foram: 0.9986 e -0.1636.

C. *Vento a Favor com Obstáculos Móveis*

A Fig. 15 mostra as médias de recompensas obtidas ao longo do treinamento para o cenário de vento a favor com obstáculos móveis e a Fig. 16 traz as médias do SAC treinado com ações contínuas.

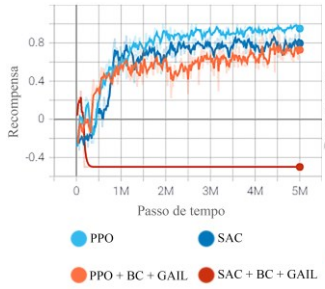

Fig. 15. Gráfico das médias de recompensas para o cenário de vento a favor com obstáculos móveis.

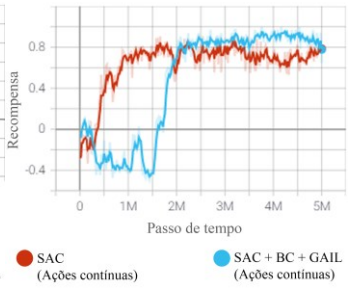

Fig. 16. Gráfico das médias de recompensas para o cenário de vento a favor com obstáculos móveis com o uso do SAC para ações contínuas.

As últimas médias de recompensas obtidas para o PPO, PPO + BC + GAIL, SAC e SAC + BC + GAIL respectivamente foram: 0.9434, 0.7128, 0.73, -0.5. E para o SAC e SAC + BC + GAIL com ações contínuas foram: 0.8019 e 0.6455.

D. *Vento de Través com Obstáculos Móveis*

A Fig. 17 mostra as médias de recompensas obtidas ao longo do treinamento para o cenário de vento de través com obstáculos móveis e a Fig. 18 traz as médias do SAC treinado com ações contínuas.

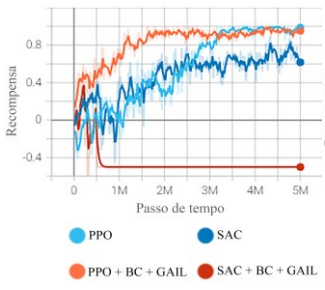

Fig. 17. Gráfico das médias de recompensas para o cenário de vento de través com obstáculos móveis.

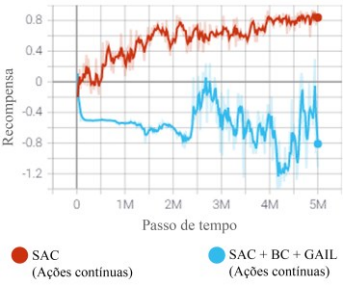

Fig. 18. Gráfico das médias de recompensas para o cenário de vento de través com obstáculos móveis com uso do SAC para ações contínuas.

As últimas médias de recompensas obtidas para o PPO, PPO + BC + GAIL, SAC e SAC + BC + GAIL respectivamente foram: 1, 0.9423, 0.6099, -0.5017. E para o SAC e SAC + BC + GAIL com ações contínuas foram: 0.8959 e -1.114.

E. *Considerações Gerais Sobre os Algoritmos*

Nesta seção discutem-se as considerações sobre o desempenho dos algoritmos estudados. O tempo de treinamento para cada algoritmo, ordenado do menor para o maior foi: PPO, PPO + BC + GAIL, SAC e SAC + BC + GAIL.

PPO: observou-se que para todas as situações esta abordagem conseguiu obter um bom desempenho e um resultado consistente. Essa consistência durante o aprendizado mostra-se condizente com a teoria por trás do PPO, em que o algoritmo possui um limiar máximo e mínimo na atualização da política, não permitindo que recompensas muito altas ou muito baixas desequilibrem a política aprendida até então.

PPO + BC + GAIL: em geral, observou-se um comportamento semelhante ao PPO com convergências ocorrendo em um número reduzido de passos. Este ponto é condizente com o fato de o algoritmo já ser alimentado com demonstrações prévias. Entretanto, nos casos de obstáculos estáticos observou-se uma queda no aprendizado após a convergência inicial e para o vento a favor com obstáculos móveis, notou-se um desempenho pior que o do PPO. Isso pode ser associado ao GAIL, afinal o BC foi apenas uma forma de pré-treino. Entende-se que isto pode ter relação com demonstrações sub-ótimas, que normalmente são fornecidas por humanos, e demonstrações que não cubram ações a serem realizadas em muitos estados, devido a aleatoriedade dos ambientes gerados nessas situações e o número limitado de ambientes utilizados nas demonstrações.

SAC: verificou-se um comportamento irregular com o uso de ações discretas. No cenário de vento a favor com obstáculos estáticos, observou-se um rápido aprendizado, mas em seguida ocorreu um desaprendizado, e, no cenário de vento de través com obstáculos estáticos o aprendizado permaneceu baixo. Para solucionar este problema, utilizou-se o treinamento com ações contínuas, que mostrou a estabilização do treinamento e melhorou o resultado para os casos citados, porém o

desempenho para os casos de obstáculos móveis continuou similar. Este algoritmo demonstrou um comportamento, em geral, inferior quando comparado ao PPO, salvo a situação de navegação com vento favorável e obstáculos estáticos. Ressalta-se que o SAC necessitou de maior tempo para treinar com o número de passos estipulado, em razão do buffer de experiência, que traz mais atualizações de política e maior custo computacional. Mesmo que o SAC apresentasse maior demora para treinar, esperava-se que a convergência acontecesse com menos passos de tempo se comparado ao PPO, porém isto só ocorreu no cenário de vento a favor com obstáculos estáticos.

SAC + BC + GAIL: observou-se desempenho ruim com este algoritmo, tanto para o uso de ações discretas quanto para contínuas. Assim, percebe-se que pode haver uma incompatibilidade destes algoritmos para determinados cenários. Para fortalecer esta argumentação, analisou-se o artigo de [36] em que há o estudo do PPO e SAC em diversos cenários exemplos do *ML-Agents* e percebeu-se que para o cenário *Pyramids*, que também tem característica episódica, utilizou-se o GAIL junto ao SAC e obteve-se um péssimo desempenho. Ainda se observa que o desempenho nos passos iniciais, em geral, não é negativo, indicando mais uma vez que o mal desempenho está mais relacionado ao GAIL, pois o BC interrompe sua influência após os 150.000 primeiros passos. E, por fim, percebe-se o grande tempo gasto quando comparado aos outros algoritmos. Grande parte da demora foi observada enquanto o BC estava ativo, afinal, neste treinamento, foi utilizado o tamanho do *buffer* como número máximo de amostras para cada atualização da imitação do BC, que é o valor padrão associado ao parâmetro que regula esta característica no pacote utilizado, e, o SAC possui um *buffer* naturalmente grande. Quando o BC é desativado ao longo do treinamento, percebe-se uma diminuição no tempo para realizar cada passo de tempo.

Por fim, ao comparar-se o estudo realizado com a revisão bibliográfica, percebe-se que assim como [18], o PPO trouxe bons resultados. Além disso, ressalta-se que na literatura atual grande parte dos estudos são realizados em simuladores 2D e gerando caminhos livres de colisão no mapa bidimensional. Desta forma, o atual projeto destaca-se em simular o barco autônomo em ambientes tridimensionais, o que também permitiu o uso de análise de imagens para a detecção de obstáculos. Finalmente, a maior parte da literatura não estuda barcos autônomos à vela, evita o uso de ventos em simulações e não aborda o uso de obstáculos móveis. Assim, além de sobressair-se nesses aspectos, o presente estudo traz a análise dos algoritmos SAC, BC e GAIL, raramente encontrados na literatura atual para o tema em questão. E, no que tange a comparação direta de resultados com outras pesquisas, apesar de algumas poucas utilizarem ambiente tridimensional com obstáculos, é difícil realizar tal comparação, pois não utilizam vento e as recompensas são relativas ao estudo.

VI. Conclusão

O projeto foi desenvolvido na plataforma de desenvolvimento de jogos *Unity®* e baseou-se em uma simulação de veleiros. A partir desta simulação, modificações foram realizadas, quatro cenários diferentes foram montados e foi feita a implementação dos algoritmos de aprendizado de máquina, tendo o uso do pacote *ML-Agents*, que permite a utilização de técnicas de inteligência artificial no *Unity®*.

Com o treinamento realizado, percebeu-se que o PPO demonstrou superioridade em relação ao SAC, sem e com utilização do aprendizado por imitação. O PPO com a limitação da atualização de sua política conseguiu se adaptar a vários tipos de cenários. O SAC, apesar de possuir uma adaptação da modelagem original para ações discretas, sofreu grande impacto com o tipo de ação utilizada. Analisando-se o uso de ações contínuas, notou-se que conseguiu convergir de maneira adequada e estável, estando de acordo com a teoria por trás do SAC. Mesmo assim, o PPO ainda conseguiu obter melhores resultados ao longo do treinamento.

Sugere-se em trabalhos futuros a utilização de cenários mais complexos e próximos da realidade. Estes consistiriam em: situações de mar agitado e mudança de luminosidade acarretada pelo momento do dia, por exemplo. No trabalho realizado, o mar permaneceu calmo. Também é necessário criar uma simulação mais precisa da dinâmica do movimento do veleiro, bem como incluir forças hidrodinâmicas mais realistas.

Por fim, seria interessante o estudo mais aprofundado da relação do SAC com algoritmos de imitação, mais especificamente o GAIL. No estudo, foi observado que os resultados dessa combinação não foram satisfatórios. Porém, podem existir casos em que esta junção não tenha resultados ruins. Uma análise de em quais casos pode-se obter desempenhos positivos e negativos com a combinação desses algoritmos seria de grande valia.

Referências

[1] B. G. Buchanan, "A (Very) Brief History of Artificial Intelligence," *AI Mag.*, vol. 26, no. 4, pp. 53–53, Dec. 2005, doi: 10.1609/AIMAG.V26I4.1848.

[2] T. W. Vaneck, "Fuzzy Guidance Controller for an Autonomous Boat," *IEEE Control Syst.*, vol. 17, no. 2, pp. 43–51, 1997, doi: 10.1109/37.581294.

[3] J. Abril, J. Salom, and O. Calvo, "Fuzzy control of a sailboat," *Int. J. Approx. Reason.*, vol. 16, no. 3–4, pp. 359–375, Apr. 1997, doi: 10.1016/S0888-613X(96)00132-6.

[4] R. Stelzer and T. Pröll, "Autonomous sailboat navigation for short course racing," *Rob. Auton. Syst.*, vol. 56, no. 7, pp. 604–614, Jul. 2008, doi: 10.1016/J.ROBOT.2007.10.004.

[5] C. Pêtrès, M. A. Romero-Ramirez, and F. Plumet, "Reactive path planning for autonomous sailboat," *IEEE 15th Int. Conf. Adv. Robot. New Boundaries Robot. ICAR 2011*, pp. 112–117, 2011, doi: 10.1109/ICAR.2011.6088585.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd, in prog ed. 2015.

[7] A. Stanford-Clark, E. Frank-Schultz, and M. Harris, "What are digital twins? – IBM Developer," 2019. https://developer.ibm.com/articles/what-are-digital-twins/ (accessed Apr. 03, 2022).

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*. arXiv, Jul. 19, 2017.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 5, pp. 2976–2989, Jan. 2018.

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, 2015, doi: 10.1038/nature14236.

[11] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, Sep. 2015.

[12] M. Andrecut and M. K. Ali, "Deep-sarsa: A reinforcement learning algorithm for autonomous navigation," *Int. J. Mod. Phys. C*, vol. 12, no. 10, pp. 1513–1523, Dec. 2001, doi: 10.1142/S0129183101002851.
[13] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 4, pp. 2850–2869, Feb. 2016.
[14] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, Jun. 2016, pp. 4572–4580.
[15] Unity Technologies, "Unity," 2021. https://unity.com/ (accessed Sep. 11, 2021).
[16] Unity Technologies, "GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit," 2021. https://github.com/Unity-Technologies/ml-agents (accessed May 10, 2021).
[17] V. Lytsus, "GitHub - vlytsus/unity-3d-boat: Unity Yacht Simulator," 2020. https://github.com/vlytsus/unity-3d-boat (accessed May 15, 2021).
[18] E. Meyer, H. Robinson, A. Rasheed, and O. San, "Taming an Autonomous Surface Vehicle for Path following and Collision Avoidance Using Deep Reinforcement Learning," *IEEE Access*, vol. 8, pp. 41466–41481, 2020, doi: 10.1109/ACCESS.2020.2976586.
[19] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to Navigate: Cooperative Path Planning for Unmanned Surface Vehicles Using Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 165262–165278, 2019, doi: 10.1109/ACCESS.2019.2953326.
[20] Z. Shi, H. Zhang, J. Zhou, and J. Wei, "An Adaptive Path Planning Based on Improved Fuzzy Neural Network for Multi-robot Systems," pp. 319–343, Jan. 2016, doi: 10.4018/978-1-4666-9572-6.CH012.
[21] J. Woo, C. Yu, and N. Kim, "Deep reinforcement learning-based controller for path following of an unmanned surface vehicle," *Ocean Eng.*, vol. 183, pp. 155–166, Jul. 2019, doi: 10.1016/j.oceaneng.2019.04.099.
[22] A. G. da S. Silva Junior, D. H. dos Santos, A. P. F. de Negreiros, J. M. V. B. de S. Silva, and L. M. G. Gonçalves, "High-Level Path Planning for an Autonomous Sailboat Robot Using Q-Learning," *Sensors*, vol. 20, no. 6, p. 1550, Mar. 2020, doi: 10.3390/s20061550.
[23] W. Wang, X. Luo, Y. Li, and S. Xie, "Unmanned surface vessel obstacle avoidance with prior knowledge-based reward shaping," *Concurr. Comput. Pract. Exp.*, p. e6110, Dec. 2020, doi: 10.1002/cpe.6110.
[24] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Autonomous Vehicular Landings on the Deck of an Unmanned Surface Vehicle using Deep Reinforcement Learning," *Robotica*, vol. 37, no. 11, pp. 1867–1882, Nov. 2019, doi: 10.1017/S0263574719000316.
[25] X. Lin and R. Guo, "Path planning of unmanned surface vehicle based on improved q-learning algorithm," in *2019 IEEE 3rd International Conference on Electronic Information Technology and Computer Engineering, EITCE 2019*, Oct. 2019, pp. 302–306, doi: 10.1109/EITCE47263.2019.9095038.
[26] Z. Zhou, Y. Zheng, K. Liu, X. He, and C. Qu, "A Real-time Algorithm for USV Navigation Based on Deep Reinforcement Learning," Dec. 2019, doi: 10.1109/ICSIDP47821.2019.9173280.
[27] X. Xu, Y. Lu, X. Liu, and W. Zhang, "Intelligent collision avoidance algorithms for USVs via deep reinforcement learning under COLREGs," *Ocean Eng.*, vol. 217, p. 107704, Dec. 2020, doi: 10.1016/j.oceaneng.2020.107704.
[28] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 1398–1404, 1991, doi: 10.1109/ROBOT.1991.131810.
[29] P. Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using Velocity Obstacles:," *http://dx.doi.org/10.1177/027836499801700706*, vol. 17, no. 7, pp. 760–772, Jul. 1998, doi: 10.1177/027836499801700706.
[30] S. Wang, F. Ma, X. Yan, P. Wu, and Y. Liu, "Adaptive and extendable control of unmanned surface vehicle formations using distributed deep reinforcement learning," *Appl. Ocean Res.*, vol. 110, p. 102590, May 2021, doi: 10.1016/j.apor.2021.102590.
[31] X. Wu *et al.*, "The autonomous navigation and obstacle avoidance for USVs with ANOA deep reinforcement learning method," *Knowledge-Based Syst.*, vol. 196, p. 105201, May 2020, doi: 10.1016/j.knosys.2019.105201.
[32] E. Meyer, A. Rasheed, A. Heiberg, and O. San, "COLREG-COMPLIANT COLLISION AVOIDANCE FOR UNMANNED SURFACE VEHICLE USING DEEP REINFORCEMENT LEARNING," *arXiv*. arXiv, Jun. 16, 2020, doi: 10.1109/access.2020.3022600.
[33] N. Vanvuchelen, J. Gijsbrechts, and R. Boute, "Use of Proximal Policy Optimization for the Joint Replenishment Problem," *Comput. Ind.*, vol. 119, p. 103239, Aug. 2020, doi: 10.1016/J.COMPIND.2020.103239.
[34] I. J. Goodfellow *et al.*, "Generative Adversarial Nets," in *Proceedings of the International Conference on Neural Information Processing Systems*, 2014, pp. 2672–2680.
[35] Facebook, "PyTorch," 2021. https://pytorch.org/ (accessed Sep. 11, 2021).
[36] A. Juliani *et al.*, "Unity: A General Platform for Intelligent Agents," *arXiv*, Sep. 2018.
[37] Microsoft, "Visual Studio: IDE e Editor de Código para Desenvolvedores de Software e Teams," 2021. https://visualstudio.microsoft.com/pt-br/ (accessed Sep. 11, 2021).
[38] Blender Foundation, "blender.org - Home of the Blender project - Free and Open 3D Creation Software," 2021. https://www.blender.org/ (accessed Sep. 11, 2021).
[39] K. Gyzen, "Rock Pack Vol.1 Free - BlenderNation," 2020. https://www.blendernation.com/2020/03/14/rock-pack-vol-1-free/ (accessed Jul. 02, 2021).
[40] Seemlyhasan, "Fisher Boat free VR / AR / low-poly 3D model," 2020. https://www.cgtrader.com/free-3d-models/watercraft/industrial/fisher-boat-96631d80-50ba-4b41-a11d-2bea68e1db64 (accessed Jul. 02, 2021).
[41] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *J. Big Data*, vol. 8, no. 1, p. 53, Dec. 2021, doi: 10.1186/s40537-021-00444-8.

Rodrigo Picinini Méxas holds a bachelor's degree in Mechanical Engineering from the Universidade Federal Fluminense (UFF). His interests include artificial intelligence and visual computing. Currently works at Extreme Digital Solutions as a developer.

Fabiana Rodrigues Leta is Full Professor of Mechanical Engineering at Universidade Federal Fluminense (UFF). She develops research mainly in the following areas: Visual Computing, Metrology by Image, Technological Innovation, Oil and Gas Industry and Engineering Education. She is part of the scientific committee of IWSSIP International Conference on Systems, Signals and Image Processing and of ACE-X - International Conference on Advanced Computational Engineering and Experimenting since 2010. She published more than 270 papers in conferences and journals and edited 5 books.

Esteban Clua is professor at Universidade Federal Fluminense and coordinator of UFF Medialab, Scientist of the State of Rio prize in 2019. His main research and development areas are Digital Games, Virtual Reality, GPUs, Simulation and Data Science.