Departamento de Engenharia Mecânica
Trabalho de Conclusão de Curso (TCC)/2020.2

CONTROLE APLICADO A ROBÔ INTELIGENTE MÓVEL

Júlio Freire Peixoto Gomes
julio.juliofreire@gmail.com

Jonatha Wallace da Silva Araújo
jonatha_wallace@hotmail.com

Resumo: *Este projeto consistiu em um estudo de caso usando conhecimentos da robótica, controle e aprendizado de máquinas a partir de uma simulação de um robô móvel diferencial em um ambiente virtual com o uso da linguagem de programação Python. Para isto, foram desenvolvidas as equações cinemáticas do robô e logo após utilizado um controlador proporcional para a sua movimentação a locais de forma precisa. O objetivo foi de mover-se aos pontos principais de uma rosa dos ventos (Norte, Sul, Leste e Oeste) no intuito de verificar a existência de recompensas. Nessa busca por recompensa, o algoritmo de aprendizado por reforço, ε-greedy, foi utilizado, de modo a possibilitar ao robô a tomada de decisões baseadas no parâmetro ε do modelo. A análise dos resultados permitiu observar que utilizando o algoritmo de aprendizagem houve um aumento de cerca de 0,6 na recompensa média quando comparado à escolhas sempre aleatórias, melhorando significativamente as decisões tomadas pelo robô.*

Palavras chaves: *Robô móvel diferencial, dinâmica, controlador proporcional, ε-greedy.*

1. INTRODUÇÃO

A busca por mecanismos que acelerem a produção de determinado objeto, ou ainda, que desempenhem funções atribuídas ao homem tem sido uma ambição da humanidade há algum tempo. Diante desse cenário, a robótica, através dos mais diversos tipos de robôs, tem apresentado soluções efetivas que se estendem sob um grande leque de aplicações, sejam elas pequenas ou até mesmo tarefas de alta complexidade, por exemplo, o controle da dinâmica de voo de uma aeronave.

Segundo Corke (2017), a palavra *robot*, de origem Checa, criada pelo irmão do escritor Karel Čapek, significa trabalho forçado, trabalho duro ou labuta. Essa conceituação pode ser expandida para definir um robô como uma máquina que é capaz de realizar uma tarefa pré-programada ou por conta própria.

Diversos estudos têm apresentado a inteligência artificial e o aprendizado de máquinas como ferramentas que permitem aos robôs tomar decisões, classificar e agrupar, por exemplo, de forma rápida e com altas probabilidades de acerto. De acordo com Sutton e Barto (2018), o aprendizado pode ser dividido em aprendizado supervisionado, não-supervisionado e aprendizado por reforço.

Esse conhecimento é utilizado incansavelmente nas mais diversas aplicações do dia-a-dia e até em atividade de alta complexidade, podendo ser citados: aperfeiçoamento de buscas no Google, melhoria na recomendação de vídeos no Youtube e filmes na Netflix, como também reconhecimento de imagens, transações fraudulentas, predições de preços de imóveis e entre diversas outras. Sendo assim, a utilização de aprendizado de máquinas com robótica seria só mais uma entre essas distintas aplicações.

Na robótica é necessário um passo anterior a finalidade da máquina, a movimentação. Ela é prevista pela modelagem dinâmica, que consiste no estabelecimento de referenciais inerciais e móveis, sob os quais se descreve a cinemática (posição, velocidade e aceleração) de todos as partes que compõem o mecanismo. Em seguida, o aprendizado de máquinas permite que o robô analise determinados dados e através deles extraia informações capazes de possibilitar o reconhecimento de padrões que o ajudem na execução de determinada ação. De acordo com Mohri *et al.* (2018), esse campo de estudo possibilita aos computadores e robôs a habilidade de aprender sem serem explicitamente programados, como proposto Arthur Samuel em sua conceituação de aprendizado de máquinas.

Diante do exposto, o presente trabalho consistiu na simulação de um robô móvel diferencial, que possui duas rodas paralelas com velocidades independentes de modo a permitir uma movimentação no plano 2D. O movimento do robô é descrito pela solução das equações cinemáticas diferenciais desse tipo de robô em conjunto a um sistema de controle proporcional (P), que especifica as condições cinemáticas que o robô deve desenvolver para chegar a qualquer ponto do plano e por fim a aplicação do algoritmo de aprendizado de máquinas. A escolha do robô tipo móvel diferencial e desse algoritmo foi feita devido à simplicidade, custo e por conseguir realizar a tarefa de percorrer 4 pontos básicos de uma rosa dos ventos, e pela capacidade de checar em cada um dos pontos a presença de uma recompensa.

2. REVISÃO BIBLIOGRÁFICA

Diversos trabalhos associados a controle e automação de robôs e mecanismos utilizando os mais diversos tipos de aprendizado são apresentados na literatura. Alguns trabalhos relacionados a esses tópicos são apresentados.

Na área de controle, Bessa *et al.* (2008) descreve o desenvolvimento de um sistema de controle de profundidade para veículos subaquáticos operados remotamente. A estratégia utilizada no trabalho consistiu no controle por modos deslizantes aprimorado por um algoritmo *fuzzy* adaptativo para compensação de incertezas e perturbações. Por fim, resultados numéricos são apresentados de modo a demonstrar o desempenho do sistema de controle proposto.

Já Dröder *et al.* (2018) apresenta uma abordagem de aprendizado de máquina para permitir que robôs industriais consigam contornar obstáculos ou pessoas na área de trabalho. A estratégia adotada para controle da movimentação do robô consistiu no uso de redes neurais artificiais, análise de agrupamento e abordagem de vizinho mais próximo para planejamento do caminho a ser traçado.

Com contextos mais próximos do que é apresentado neste trabalho, Carrascosa e Bellalta (2020) utilizaram o algoritmo *ε-greedy* e uma variação desse método que é chamado de *opportunistic ε-greedy with stickness*, que evita constante emparelhamentos desnecessários, para selecionar um ponto de acesso WiFi por um usuário diante um ambiente dinâmico.

Bem como a utilização por Mendoza et al. (2012) do *ε-greedy* para conjunto ao método de algoritmo genérico para melhorar a produção de novas gerações acelerando a convergência dos resultados, a precisão deles e permitindo que o usuário controle a velocidade de evolução.

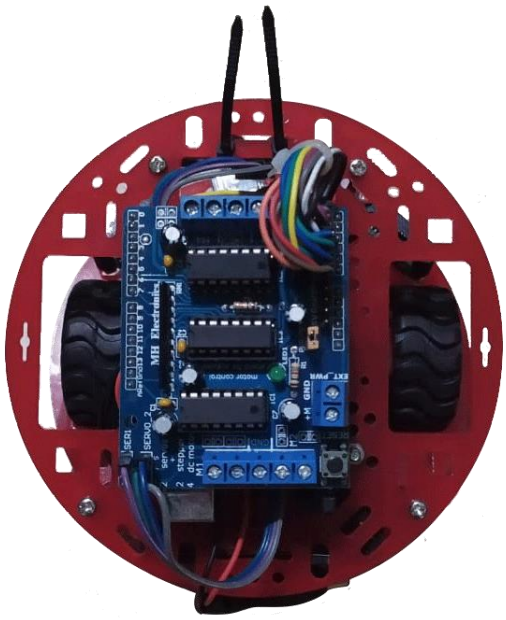
Kuang e Leung (2019) utilizaram o duas *ε-greedy*, *EGSE-A* e *EGSE-B* para melhorar a pesquisa de informações de instrumentos musicais e permitiu descobrir, através da política de exploração, informações que eram relevantes e não eram descobertas por outros meios.

Na abordagem de uma simulação em um espaço 2D, Zoltán (2012) faz uso do *Q-learning*, algoritmo de aprendizado por reforço, para o agente ser guiado em um ambiente desconhecido e chegar ao alvo. Para identificar o espaço, o agente utiliza de sensores que medem a distância entre ele e os pontos no seu alcance, o que possibilita a identificação possíveis obstáculos ou a do seu objetivo.

2.1 Robô Móvel Diferencial

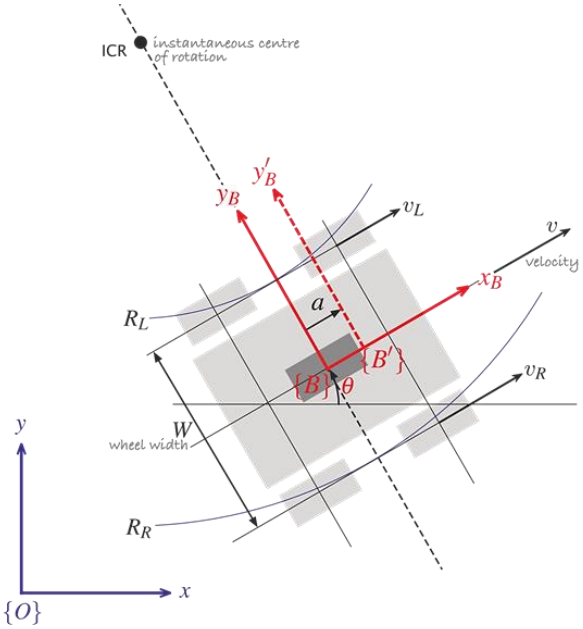
Escolheu-se esse robô pela simplicidade da modelagem e pelo fato de ser possível uma futura fabricação de um modelo experimental com uso de Arduino. Sua movimentação é bastante simples e suficiente para realizar a tarefa o qual fora projetado. O que caracteriza esse modelo é a disposição de suas rodas que são paralelas pertencentes a um mesmo eixo perpendicular e controladas independentemente, ou seja, cada uma pode ter velocidade diferente da outra. Este modelo pode ser visualizado na Figura 1, na qual tem-se representado um modelo em Arduino de um robô móvel diferencial.

Figura 1. Robô móvel diferencial montado em placa de Arduino.



Segundo Corke (2017), a manipulação separada de cada roda pode resultar em 3 tipos de movimentos, um de translação, um de rotação ou a combinação dos dois. Se suas rodas tiverem a mesma velocidade com mesmo sentido, o robô pode se locomover para frente ou para trás, quando tiverem a mesma velocidade e sentidos opostos a rotação será em torno de si e quando forem diferentes tanto em magnitude quanto em sentido combinar-se-á os dois movimentos anteriores. A Figura 2 mostra uma forma esquemática desenvolvimento equações cinemáticas.

Figura 2. Representação esquemática do robô diferencial.



Fonte: Adaptado de Corke (2017).

A posição do robô pode ser definida em termos do vetor de coordenadas generalizadas **q** = (*x*, *y*, *θ*). A velocidade do robô medida no referencial {B} é dada por *v* na direção x_B e zero na direção y_B, já que não ocorre deslizamento das rodas nessa direção. A Eq. (1) descreve o vetor velocidade no referencial {B}.

$$^{B}v = (v, 0) \tag{1}$$

Com base na Figura 2, utilizando a definição de centro instantâneo de rotação (ICR) tem-se a seguinte relação, Eq. (2).

$$\dot{\theta} = \frac{v_L}{R_L} = \frac{v_R}{R_R} \tag{2}$$

Sendo $\dot{\theta}$ a velocidade angular do referencial móvel, v_L a velocidade da roda esquerda, v_R a velocidade da roda esquerda, R_L e R_R os raios formados a partir do ICR para roda esquerda e direita, respectivamente. Como temos a relação de que $R_R = R_L + W$ em que W é a distância entre as duas rodas, obtendo-se a Eq. (3).

$$\dot{\theta} = \frac{v_R - v_L}{W} \tag{3}$$

As equações de velocidade descritas para referencial inercial $\{O\}$ são apresentadas nas equações abaixo.

$$\dot{x} = v\cos\theta \tag{4}$$

$$\dot{y} = v\sin\theta \tag{5}$$

$$\dot{\theta} = \omega \tag{6}$$

No qual $\dot{x}$ e $\dot{y}$ são as velocidades do referencial móvel ao longo dos eixos *x* e *y*, respectivamente, *v* a velocidade linear do referencial móvel e ômega a velocidade angular. Segundo Corke (2017), essa modelagem cinemática é frequentemente chamada de *unicycle model* e pode ser relacionado com a modelagem do robô diferencial pelas equações seguintes:

$$v = \frac{v_R + v_L}{2} \tag{7}$$

$$\omega = \frac{(v_R - v_L)}{W} \tag{8}$$

As equações cinemáticas acima foram implementadas e garantiram a correta movimentação do modelo proposto na simulação.

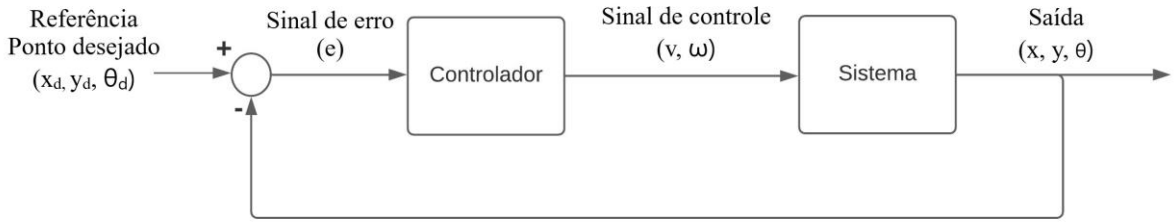
2.2 Controlador Proporcional

Teoricamente, movimentar o robô é uma tarefa condicionada a solucionar as equações cinemáticas do tópico anterior, no entanto é difícil afirmar com precisão se o robô executará o movimento correto para o qual foi projetado. Nesse caso, se faz necessário de técnicas de controle em sistema de malha fechada para que se possa saber se o robô está de fato está indo para o local indicado. O erro definido como sendo a diferença entre a posição de referência e a posição atual do robô é a variável que norteará a atuação do controlador de modo que à medida que o erro tende a zero, a posição do robô tenderá a posição de referência.

Dentro dessa sistemática, é preciso realizar a comunicação do sinal de erro com o sistema e assim atualizar os sinais de entrada e orientá-lo ao destino corretamente. O elemento responsável por essa ponte é o controlador, que se baseia nas leis de controle, que especifica quais os sinais de controle ao passo que o erro é atualizado. Então, combinando a solução cinemática do robô obtida via modelo matemático juntamente com a ação de controle executado por um controlador tem-se a movimentação correta do corpo.

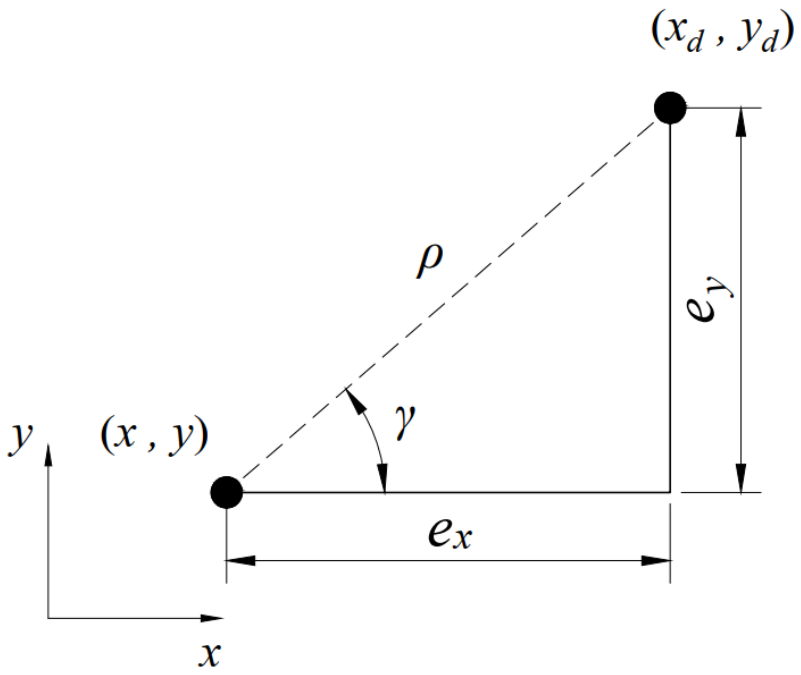
Nesse trabalho foi utilizado um controlador proporcional, que segundo Campos e Texeira (2010), o esse tipo de controlador utiliza a estratégia de controle baseada na obtenção do sinal de saída do controlador como uma proporção do erro entre o valor de referência e o valor medido. Corke (2017) apresenta esse tipo de controlador como um sistema básico na solução de problemas onde se deseja controlar a movimentação de um corpo a uma posição especificada. A Figura 3 apresenta o esquema em malha fechada da atuação do controlador

Figura 3. Sistema em malha fechada para o robô.



Através das equações da cinemática encontradas, observa-se que as posições são determinadas pelas velocidades linear e angular (ou as velocidades de cada roda), então controlar essas velocidades é controlar a posição em que o robô está. Porém, uma forma para que o controlador interprete o sinal de erro e dê uma resposta afim de minimizá-lo é usando as coordenadas polares, as quais podem ser esquematizadas na Figura 4 abaixo.

Figura 4. Esquema para mudança de coordenadas.



Na Figura 4, são apresentados os pontos atual (x, y) e de destino (x_d, y_d) bem como as quantidadades e_x e e_y dadas nas Eqs. (9) e (10).

$$e_x = x_d - x \tag{9}$$

$$e_y = y_d - y \tag{10}$$

A variável ρ consiste na distância euclidiana entre os pontos e a variável γ representa o ângulo polar representados respectivamente pelas Eqs. (11) e (12).

$$\rho = \left(e_x^2 - e_y^2\right) \tag{11}$$

$$\gamma = \tan^{-1}\left(\frac{e_y}{e_x}\right) \tag{12}$$

O erro é definido pelas Eqs. (9) e (10), nas quais é feita a diferença entre cada sinal de referência e as coordenadas de saída. Já a Eq. (11) é a utilização dos erros para calcular a distância euclidiana ρ entre a localização atual e o ponto de referência. As Eqs. (13) e (14) dizem respeito a qual ângulo α o robô deve rotacionar para ir ao ponto destino e qual ângulo β o robô deve girar para chegar no ângulo de referência.

$$\alpha = \gamma - \theta \tag{13}$$

$$\beta = \theta_d - \gamma \tag{14}$$

Nesse caso, as equações em sistema polar consistem no próprio sistema em malha fechada. O controlador responderá ao sinal do erro de forma proporcional a essa entrada, logo:

$$v = k_\rho \rho \tag{15}$$

$$\omega = k_\alpha \alpha + k_\beta \beta \tag{16}$$

Sendo v a velocidade linear e ω a velocidade angular que o robô deve tomar para se direcionar ao ponto de referência.

A interpretação feita das Eqs. (15) e (16) é que quanto mais distante do ponto de referência maior será a velocidade de forma a minimizar mais rapidamente a distância e o mesmo ocorrem para os ângulos, girando o robô mais rapidamente quanto maior forem os ângulos α e β.

Segundo Corke (2017), para que o robô possua erros tendendo a zero, os parâmetros de controle devem seguir a teoria de estabilidade de Lyapunov, a qual estabelece que $k_\rho > 0$, $k_\alpha > k_\rho$ e $k_\beta < 0$.

Esses valores advêm da solução das equações linearizadas em torno do ponto de interesse ($\rho = 0$, $\alpha = 0$ e $\beta = 0$), quando os erros tendem a zero. As equações abaixo mostram o sistema cinemático para coordenadas polares, obtida através da derivada completa em relação ao tempo das Eqs. (11), (13) e (14).

$$\dot{\rho} = -k_\rho \rho \cos\alpha \tag{17}$$

$$\dot{\alpha} = k_\rho \sin\alpha - k_\alpha \alpha - k_\beta \beta \tag{18}$$

$$\dot{\beta} = -k_\rho \sin\alpha \tag{19}$$

E sua forma linearizada pode ser escrita pela matriz Jacobiana **J**, a qual representa uma boa aproximação linear de funções diferenciais em torno de um ponto, assim obtém-se o sistema linear da forma $\dot{x} = \mathbf{J}x$.

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \underbrace{\begin{bmatrix} -k_\rho & 0 & 0 \\ 0 & k_\rho - k_\alpha & -k_\beta \\ 0 & -k_\beta & 0 \end{bmatrix}}_{\mathbf{J}} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} \tag{20}$$

Os autovalores da matriz **J** são os modos característicos do sistema, ou seja, aqueles que ditam a sua estabilidade. Ao serem determinados com a imposição de que assumam valores negativos a condição de Lyapunov é satisfeita.

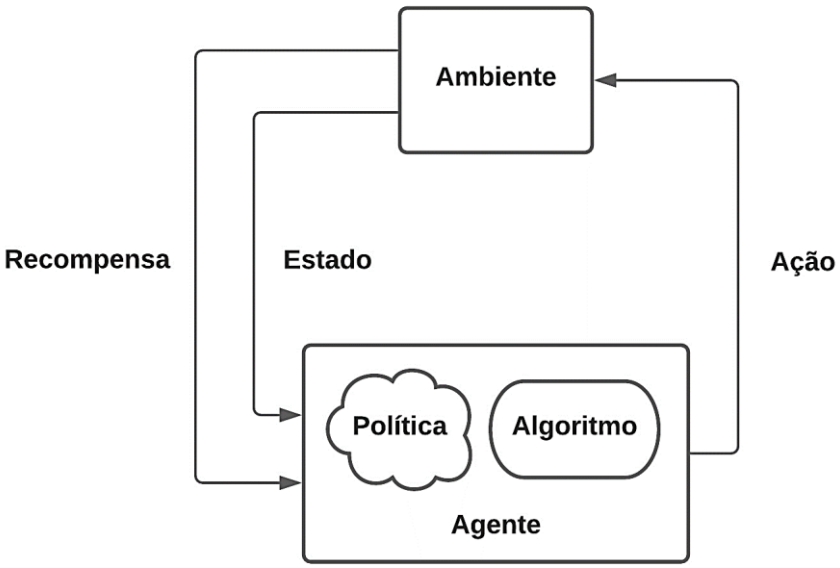
2.3 Aprendizagem por reforço

O algoritmo de aprendizado de máquinas, em específico o de aprendizado por reforço teve como objetivo estabelecer para qual ponto deve-se ir e mensurar com que frequência ir de modo a obter sempre o melhor resultado possível. Nesse processo a máquina interage com o ambiente e para cada ação tomada ela pode ser penalizada ou recompensada, e ainda, cada ação pode influenciar como o ambiente responderá dali para frente. Segundo Sutton e Barto (2018), aprender através de interações é a ideia base mais próxima de todas as teorias de aprendizagem e inteligência.

Contudo, pode-se identificar dois elementos, aquele que aprende, o agente, e aquele cujo será aprendido, o ambiente. No entanto, ainda existem subelementos que vão orientar como o agente vai se comportar e como o ambiente irá responder, eles são: a política, o sinal de recompensa, função de valor e o modelo do ambiente. A primeira irá definir como o agente se comportará na tomada de decisões no ambiente, podendo ser um valor ou uma função. Já o segundo, será a resposta do ambiente ao agente dada a ação tomada, entretanto uma recompensa imediata não significa que aquela é a melhor opção a longo prazo, já que a recompensa daquela ação pode variar ao longo do tempo, portanto, um parâmetro que melhor pode representar a satisfação do agente ao decorrer do aprendizado é a função de valor, no qual deve ser maximizada. Por fim, o modelo do ambiente é usado pelo robô para predizer como o ambiente se comportaria, reagiria e se modificaria a cada ação tomada antes mesmo de escolher determinada ação. A Figura 5 esquematiza o processo.

Visto que esse tipo de aprendizado tem uma política e valor de função definidas pelo método de aprendizagem de reforço, ou seja, tais parâmetros se modificam de acordo com o algoritmo de cada um dos métodos que podem ser usados para expressar um aprendizado por reforço. No tópico a seguir, define-se quais são esses parâmetros para o caso do *ε-greedy*.

Figura 5. Esquema de como acontece o aprendizado por reforço.



2.4 ε-greedy

Nesse método a política e função de valor, os subelementos responsáveis pela forma que o agente deverá escolher uma ação e pela avaliação da recompensa e otimização delas durante a interação agente-ambiente, são caracterizadas como ações baseadas em *explore* ou *exploit* e acumulação da recompensa média. A exploração ocorrerá com uma probabilidade igual a ε e essa atitude de explorar é entendida como uma forma de procurar melhores recompensas tomando ações aleatórias, já a atitude de *exploit*, traduzida como prospectar, é aquela em que o agente vai realizar a ação que lhe deu, até o momento, maior recompensa e ocorre com probabilidade complementar a exploração, $(1 - \varepsilon)$. Portanto, o agente irá explorar à procura de melhores recompensas e também irá agir de forma gananciosa prospectando a melhor recompensa até então.

Para a recompensa média, onde a cada ação sua recompensa é acumulada e dividida pela quantidade de ações tomadas até então, Eq. (21).

$$Q_n \doteq \frac{1}{n-1}\sum_{i=1}^{n-1} R_i \tag{21}$$

Sendo Q a recompensa média, n a quantidade de ações, R_i a recompensa encontrar na i-ésima ação.

Uma forma de otimizar a implementação quanto ao uso da memória é implementar o termo de recompensa média na forma incremental como sugere Sutton e Barto (2018), Eq. (22).

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n) \tag{22}$$

Dessa forma, para cada passo basta adicionar o valor da recompensa recolhida corrigida para a forma de recompensa média.

O algoritmo do *ε-greedy* pode ser implementado usando o pseudocódigo sugerido por Sutton e Barto (2018) mostrado na Figura 6.

Figura 6. Pseudocódigo do algoritmo de *ε-greedy*.

Algoritmo E - greedy :

Iniciar $a = [1, 2, 3, ...k]$

$Q(a) = 0$

$N(a) = 0$

Loop while até n == valor de passos

$$A = \begin{cases} \arg\max\ Q(a) \text{com probabilidade} (1-\varepsilon) \\ \text{ação aleatória com probabilidade } \varepsilon \end{cases}$$

$R = \text{Recompensa de } A$

$N(A) = n$

$$Q(A) = Q(A) + \frac{1}{n}\left[R - Q(A)\right]$$

3. METODOLOGIA

Primeiramente, a implementação de toda a parte teórica foi precedida pela criação do ambiente em que foi simulado, em seguida tornou-se possível a implementação das equações cinemáticas, controle e o *ε-greedy* fazendo as devidas adaptações para o ambiente simulado. As implementações foram criadas utilizando a linguagem de programação *Python*, junto a bibliotecas auxiliares (*numpy*, *random*, *pygame*), as quais possibilitaram a realização operações matemáticas, geração de números aleatórios e criação um ambiente virtual de simulação gráfica do projeto, respectivamente.

Inicialmente, foram criadas duas classes, a primeira, responsável por criar os objetos do ambiente, logo, criar um plano de coordenadas e as recompensas dos locais, e a segunda classe que ficou responsável pelo robô e suas atribuições, movimento, controle e aprendizado. Seguiu-se a ordem de exposição da parte teórica, primeiro foi implementado um método que realizaria o movimento do robô utilizando as Eqs. (4), (5) e (6), depois complementado com o controle onde utilizou-se as leis de controle da Eqs. (15) e (16) com os seguintes parâmetros mostrados na Tabela 1. Por último, implementou-se a parte do aprendizado.

Tabela 1. Parâmetros utilizado na simulação

k_ρ	k_α	k_β
0.2	0.5	-0.2

Resumidamente, o experimento consiste em o robô estar centrado entre suas opções de escolhas, norte, sul, leste e oeste, então é selecionado sua atitude baseada no algoritmo *ε-greedy*, ele vai até a localização da ação, recolhe a recompensa, calcula sua recompensa média enquanto retorna ao centro e dá início ao novo ciclo. Vale ressaltar que esse tipo de algoritmo de aprendizado combina perfeitamente com o caso em que foi estudado, uma vez que se trata de um problema puramente de decisões.

Algumas características devem ser deixadas claras durante o processo de testes, a recompensa é sorteada para uma das direções ao início do teste e mantida nesse local até o final, com a reinicialização do programa a recompensa é sorteada novamente, além disso tem valor unitário e está presente em um só local. Outra informação relevante é a de que o robô sabe quais as posições ele pode ir, mas desconhece o local de recompensa e seu valor.

O esquema em que uni todas as implementações e mapeia a ordem de execução do programa é visualizado na Figura 7 e executado na Figura 8 com sua imagem da simulação. O código utilizado está disponível no apêndice A do trabalho.

Figura 7. Esquema completo da simulação, local de início, escolha da ação, mobilidade sempre passando pelo controlador, cálculo da recompensa média e reinicialização do ciclo.

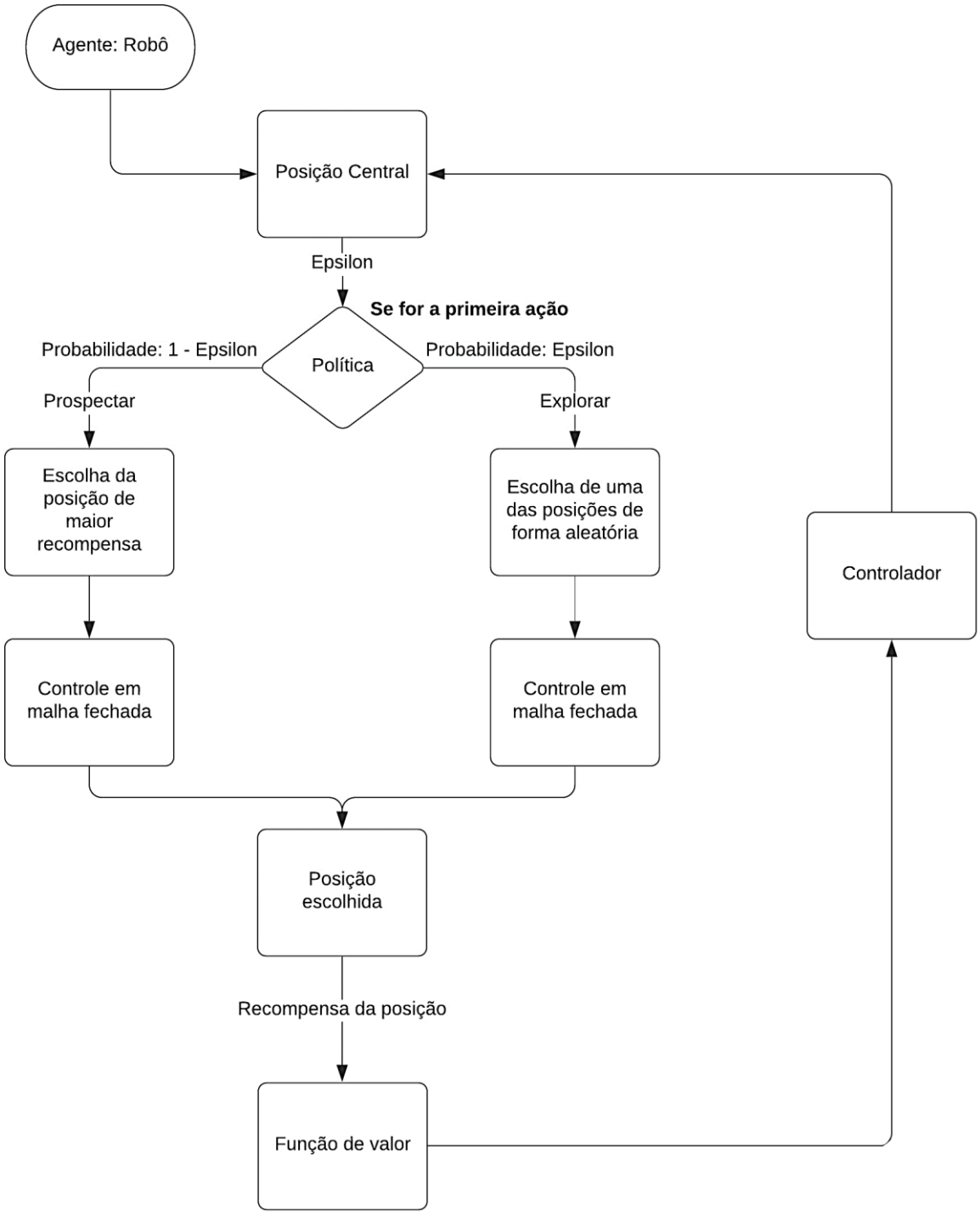
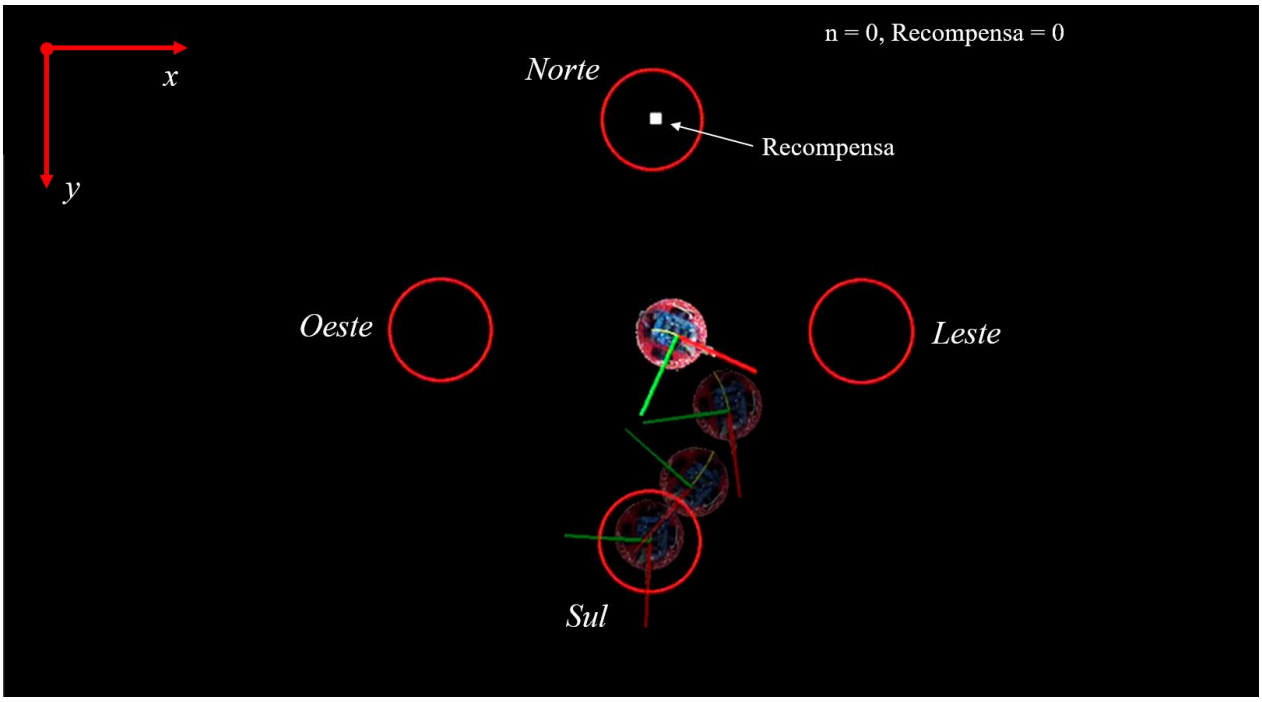


Figura 8. Imagem da simulação com informações pertinentes a execução. A recompensa está marcada com cor branca apenas para visualização.

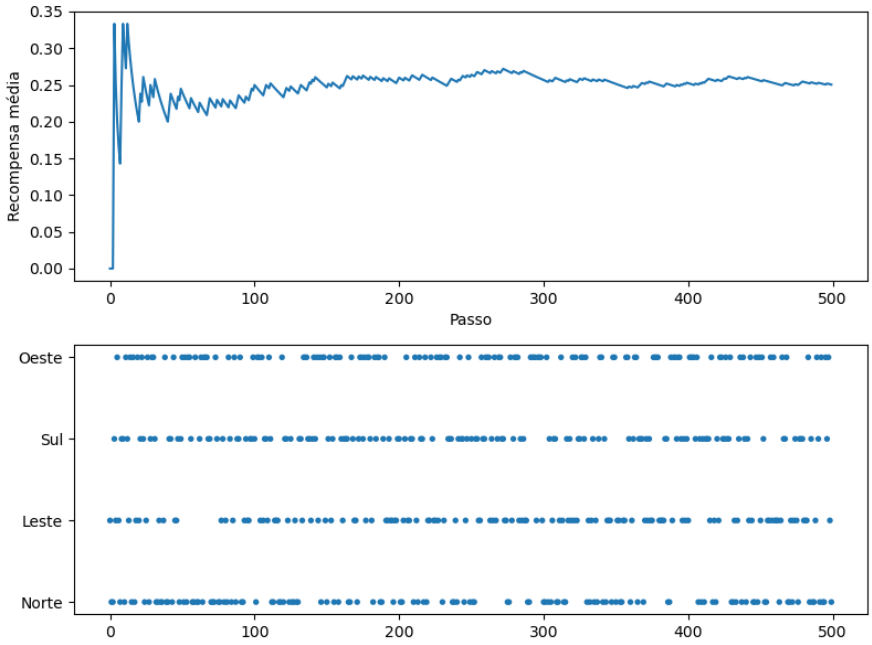


4. RESULTADOS E DISCUSSÕES

4.1 Totalmente Explorador

Para o primeiro teste, foi utilizado um robô com decisões sempre aleatórias, ou seja, seu parâmetro ε foi igual a 1 e tinha caráter totalmente exploratório, contudo registrou-se uma recompensa de média de 0.25, o que era esperado, uma vez que a probabilidade de ir a cada um dos pontos eram a mesma e igual a ¼. A Figura 9 apresenta o comportamento das recompensas médias ao longo das buscas realizadas pelo robô.

Figura 9. Gráfico da recompensa média e das ações ao longo da execução

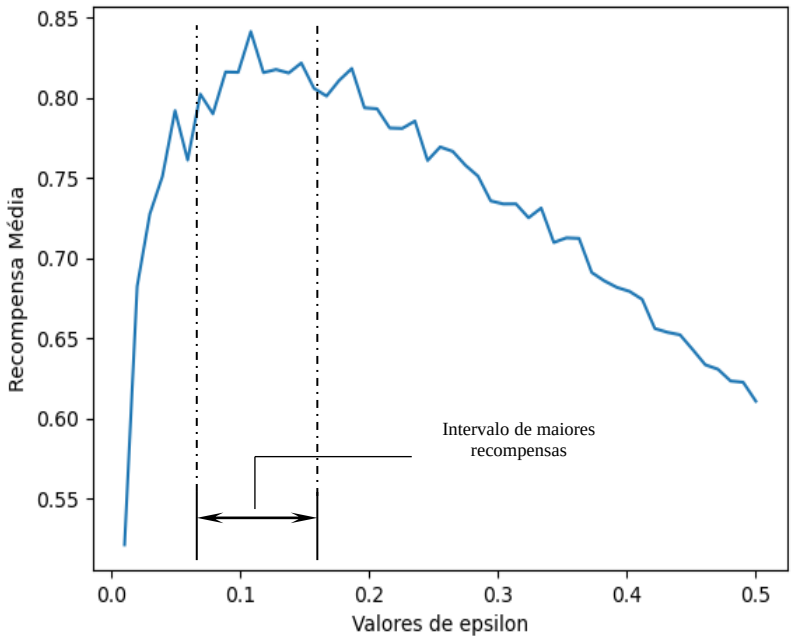


Na Figura 9, o gráfico mais embaixo mostra as ações tomadas ao longo do teste, tornando viável a visualização da influência de cada ação com a recompensa média, exposto primeiro, além do mais cada ação é mutualmente excludente, ou seja, a cada passo só pode ser escolhido um local e por isso cada ponto dado pelo gráfico só pode pertencer a uma das direções naquele determinado passo. Ainda no gráfico de ações tomadas, pode-se perceber que a densidade de pontos entre cada um dos locais é bem parecida, o que torna visível a distribuição uniforme de 0,25 para cada um dos pontos, o que nos explica a aproximação da recompensa média também ao valor de 0,25. Contudo, o próximo passo é dado com o ajuste do parâmetro ε a fim de maximizar a recompensa média.

4.2 Balanceamento da política de explorar e prospectar

Para tal balanceamento, foram feitas uma bateria de testes variando o ε em um intervalo de]0, 0,5] dividido em 50 partes iguais, ou seja, com incremento de 0,01, e para cada valor tomado foram feitos 50 testes de 1000 passos. Verificou-se os valores de recompensa média pelas médias em cada valor de ε que está mostrado na Figura 10.

Figura 10. Gráfico das recompensas média para o intervalo]0, 0.5] nos valores de epsilon.



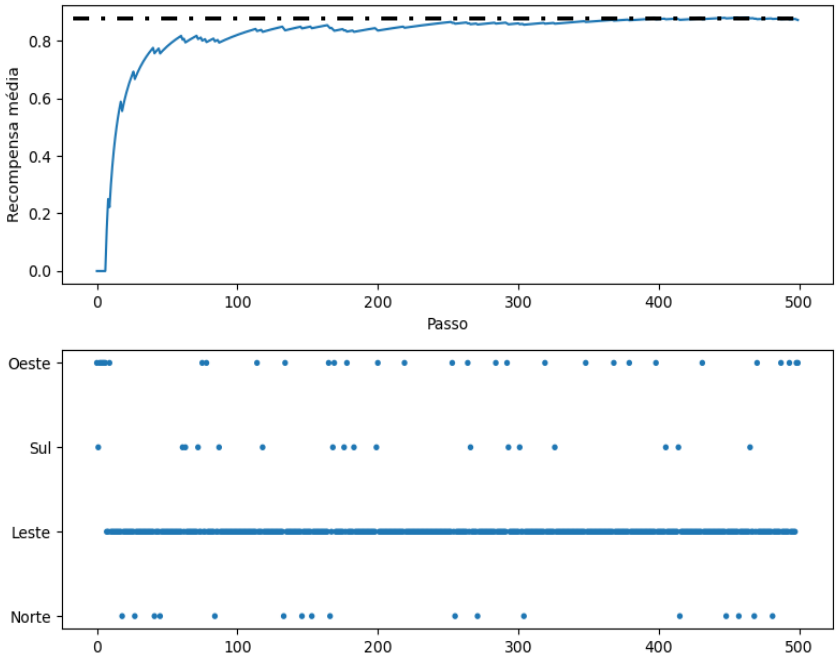
Nesse teste evidenciou-se que as recompensas médias têm os maiores valores quando o ε está dentro do intervalo as margens do pico de 0.1, ainda durante esse teste, pôde-se ser observado que 500 passos já mostrava uma tendência a convergência da recompensa média.

4.3 Efetividade do método

Como o método também depende de uma certa aleatoriedade nas situações de exploração para que o agente encontrasse novos locais e novas recompensas, foram feitos diversos testes usando os resultados evidenciados no tópico anterior, em que se utilizou o número de passos de convergência de 500 e o valor de ε igual 0.15, no qual esse valor de ε pertence ao intervalo encontrado para maiores recompensas média. Para as simulações foram utilizados os parâmetros de controle exposto na Tab. 1.

As Figuras 11, 12 e 13 mostram, no início de cada teste, a quantidade de passos que o agente demorou até encontrar o local de recompensa e, no gráfico inferior, uma densidade de pontos maior no local de maior recompensa.

Figura 11. Agente com descoberta rápida.



Analisando-se o gráfico da Figura 11 é possível observar que o robô encontrou a primeira recompensa após verificar 7 posições aproximadamente. Após esse ponto, é possível verificar um aumento significativo na taxa de sucesso do robô, haja vista que a recompensa média ainda não possui uma grande quantidade de ações do robô.

Na Figura 8, tem-se a situação na qual o agente descobriu a primeira recompensa após 40 tentativas aproximadamente. Comparando as Figura 11 e 12 é possível inferir que a taxa na qual as recompensas médias são descobertas é inferior, isso é justificado pela quantidade de tentativas iniciais que o robô teve que realizar para descobrir. Na Figura 13, o agente levou ainda mais tempo que os outros dois testes para descobrir o local de recompensa, cerca de 70 passos.

Figura 12. Agente com descoberta em tempo médio.

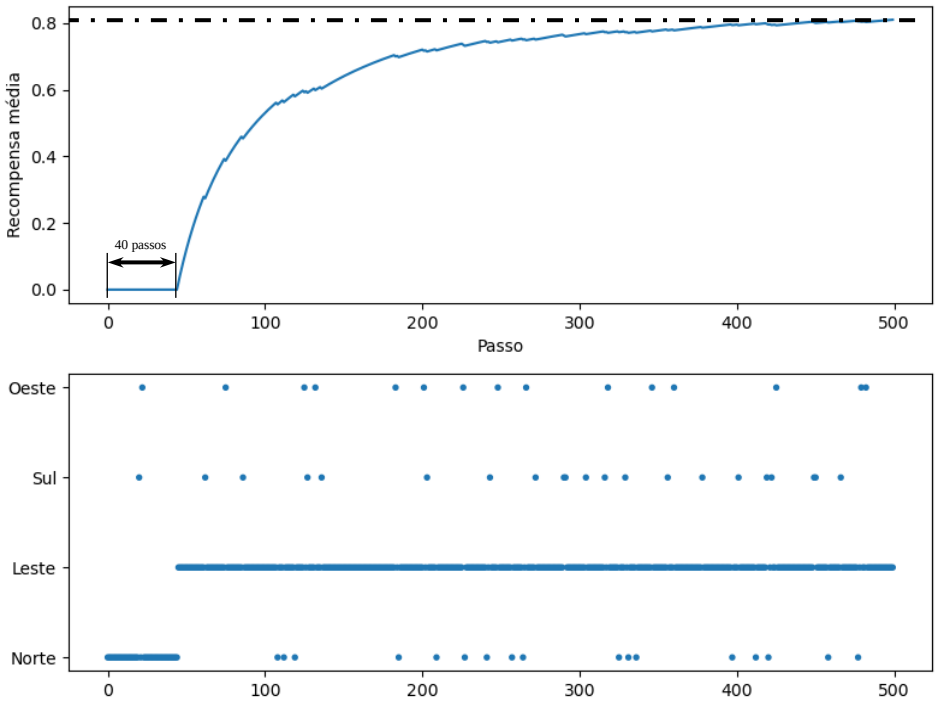
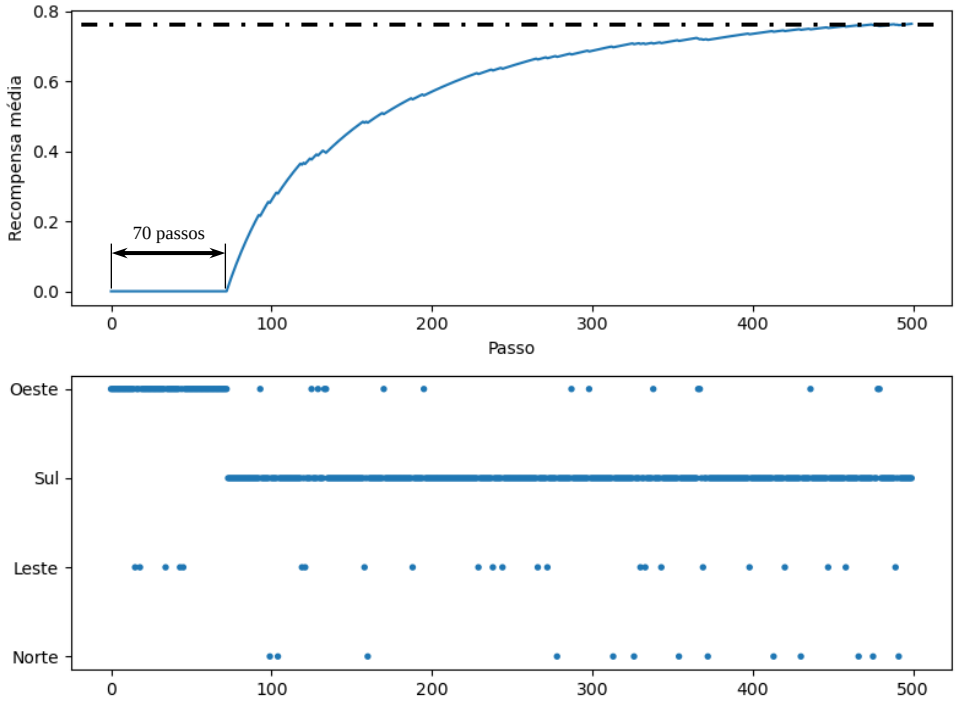


Figura 13. Agente com descoberta lenta.



Na Figura 11, a recompensa foi encontrada mais prematuramente, portanto havia mais passos restantes em que o agente poderia prospectar esse local, logo é de se esperar que a compensa média fosse maior que os demais testes. Nas Figura 12 e Figura 13, apesar de tempos diferentes na busca da melhor opção, as recompensas médias foram bem próximas, sendo assim, quando o número de passos aumenta, independente da demora para encontrar o ponto favorável, essa recompensa média tenderá a valores bem próximos.

Nas Figuras 11, 12 e 13 é possível ver que as recompensas médias foram alteradas conforme a "sorte" em que o agente descobriu o local de recompensa, além de que a cada ponto de exploração, nesse caso, influi com efeitos negativos na recompensa média, já que foi um passo que ele tomou uma ação e não obteve retorno.

O valor de recompensa máxima em cada situação descrita foi ligeiramente alterado pelo tempo de descoberta da recompensa, atingindo um valor de aproximadamente 0.85 no caso de identificação da recompensa mais rápido, 0.8 no caso de tempo médio e 0.77 em relação ao último caso.

5. CONCLUSÕES

Nesse trabalho foi desenvolvido um estudo de caso no qual realizou-se a simulação em ambiente virtual do controle de um robô móvel diferencial aliado ao desenvolvimento de um método de aprendizagem por reforço, *ε-greedy*.

O trabalho possibilitou a interdisciplinaridade de vários conceitos relacionados a mecânica, robótica e computação, tais como, modelagem de equações cinemáticas, controle dessas equações e aplicação de algoritmo de aprendizagem por reforço.

Com a análise dos dados, encontrou-se um intervalo de valores de ε que possibilitaram a obtenção das maiores taxas de recompensas, o qual se situou entre 0.07 a 0.16 aproximadamente.

Percebeu-se uma melhora significativa em termos de ganho no desempenho do robô quando comparadas as situações de busca totalmente exploratórias com a busca balanceada entre prospecção e exploração com a utilização do método *ε-greedy*. Em termos de recompensa média, o valor obtido com a utilização do método atingiu valores de aproximadamente 0.8, ao passo que na ausência do método, ou seja, caráter exploratório, a expectativa esteve limitada ao valor 0.25.

Em trabalhos futuros pretende-se aprofundar o e-greedy com modificações, bem como o estudo de novos métodos de aprendizagem por reforço.

6. REFERÊNCIAS

Mohri, M.; Rostamizadeh, A.; Talwalkar, A. **Foundations of Machine Learning**. Ed. 2. Cambridge, MA: The MIT Press. 2018.

Carrascosa, M; Bellata, B. **Multi-Armed Bandits for Decentralized AP selection in Enterprise WLANs**. *Computer Comunications*. Vol. 159, pp 108-123. 2020.

Mendoza, M. R.; Werhli, A. V; Bazzan, A. L. C. **An Epsilon-Greedy Mutation Operator Based on Prior Knowledge for GA Convergence and Accuracy Improvement: An Application to Networks Inference.** 2012 Brazilian Symposium on Neural Network, pp. 67-72. 2012

Kuang, N. L.; Leung, C. H. C. **Performance Effectiveness of Multimedia Information Search Using the Epsilon-Greedy Algorithm.** 18th IEEE international conference on machine learning and applications, pp 929-936. 2019

Dröder, K.; Bobka, P.; Germann, T.; Gabriel, F.; Dietrich, F. **A Machine Learning-Enhanced Digital Twin Approach for Human-Robot-Collaboration**. *Procedia CIRP*. Vol. 76, pp. 187-192. 2018.

Bessa, W. M.; Dutra, M. S.; Kreuzer, E. **Depth control of remotely operated underwater vehicles using an adaptive fuzzy sliding mode controller**. *Robotics and Autonomous Systems*. Vol. 56, Ed. 8, pp. 670-677. 2008.

Corke, P. **Robotics, Vision and Control: Fundamental Algorithms in MATLAB®**. *Springer*. Vol. 118. 2011.

Sutton, R. S.; Barto, A. G. **Reinforcement Learning: An Introduction.** Cambridge, Massachusetts: The MIT Press. 2018.

Zoltán, S. **Robot Control Using Q-Learning**. Scientific Bulletin of the Petru Maior. Vol 9. 2012.

Campos, M. C. M. M.; Texeira, H. C. G. **Controles Típicos de Equipamentos e Processos Industriais**. Editora: Blucher; 2ª edição. 2010.

7. APÊNDICE

Apêndice A – Código desenvolvido em linguagem de programação Python. Disponível em: https://github.com/juliofreire/TCC/blob/main/lady_bug_oficial.py

```
import pygame
import math
import random
import numpy as np
import matplotlib.pyplot as plt

class Envir:
    def __init__(self, dimensions):
        # colors
        self.black = (0, 0, 0)
        self.white = (255, 255, 255)
        self.green = (0, 255, 0)
        self.blue = (0, 0, 255)
        self.red = (255, 0, 0)
        self.yellow = (255, 255, 0)
        # maps dims
        self.height = dimensions[0]
        self.width = dimensions[1]
        # window settings
        self.MapWindowName = "DDR Controlled"
        pygame.display.set_caption(self.MapWindowName)
        self.map = pygame.display.set_mode((self.width,
                                            self.height))
        # Text Variables
        self.font = pygame.font.Font('freesansbold.ttf', 20)
        self.text = self.font.render('default', True, self.white, self.black)
        self.textRect = self.text.get_rect()
```

```
        self.textRect.center = (dimensions[1] - 400, dimensions[0] -
dimensions[0]/7)
        self.textRect_cur = self.text.get_rect()
        self.textRect_cur.center = (dimensions[1] - 400, dimensions[0] -
dimensions[0]/7 - 40)
        self.textRect_goal = self.text.get_rect()
        self.textRect_goal.center = (dimensions[1] - 400, dimensions[0] -
dimensions[0]/7 - 20)
        self.textRect_reward = self.text.get_rect()
        self.textRect_reward.center = (dimensions[1] - 400, dimensions[0] -
dimensions[0] + 20)

        # Trail
        self.trail_set = []
        # Points
        self.point0 = (600, 100, math.radians(270))
        self.point1 = (800, 300, math.radians(0))
        self.point2 = (600, 500, math.radians(90))
        self.point3 = (400, 300, math.radians(180))
        self.points = (self.point0, self.point1, self.point2, self.point3)
        self.b = random.randint(0, 3)
        # self.b = 1
        self.point_ch = self.points[self.b]
        self.bandit = np.zeros(4)
        self.bandit[self.b] = 1
        self.point_size = (10, 10)

    def write_info(self, v, omega):
        txt = f"v = {v}, omega = {omega}"
        self.text = self.font.render(txt, True, self.white, self.black)
        self.map.blit(self.text, self.textRect)

    def write_info_cur(self, x, y, theta):
        txt_cur = f"x = {x}, y = {y}, theta = {int(math.degrees(theta))}"
        self.text = self.font.render(txt_cur, True, self.white, self.black)
        self.map.blit(self.text, self.textRect_cur)

    def write_info_goal(self, g_x, g_y, g_theta):
        txt_goal = f"goal_x = {g_x}, goal_y = {g_y}, goal_theta =
{int(math.degrees(g_theta))}"
        self.text = self.font.render(txt_goal, True, self.white, self.black)
        self.map.blit(self.text, self.textRect_goal)

    def write_info_reward(self, n, reward):
        txt_goal = f"n = {n}, reward = {reward}"
        self.text = self.font.render(txt_goal, True, self.white, self.black)
        self.map.blit(self.text, self.textRect_reward)

    def trail(self, pos):
        for i in range(0, len(self.trail_set) - 1):
            pygame.draw.line(self.map, self.yellow, (self.trail_set[i][0],
self.trail_set[i][1]),
                             (self.trail_set[i + 1][0], self.trail_set[i +
1][1]))
        if self.trail_set.__sizeof__() > 2000:
            self.trail_set.pop(0)
        self.trail_set.append(pos)

    def robot_frame(self, pos, rotation):
        n = 80
```

```
        centerx, centery = pos

        x_axis = (centerx + n * math.cos(rotation), centery + n *
math.sin(rotation))
        y_axis = (centerx + n * math.cos(rotation + math.pi/2), centery + n *
math.sin(rotation + math.pi/2))
        pygame.draw.line(self.map, self.red, (centerx, centery), x_axis, 3)
        pygame.draw.line(self.map, self.green, (centerx, centery), y_axis, 3)

    def coord(self):
        x, y = 0, 0
        pygame.draw.line(self.map, self.white, (x, y), (x+100, y), 3)
        pygame.draw.line(self.map, self.white, (x, y), (x, y+100), 3)

    def point(self):
        pygame.draw.rect(self.map, self.white,
self.point_ch[0:2]+self.point_size)


class Robot:
    def __init__(self, startpos, goalpos, robotImg, width):
        self.m2p = 3779.52  # meter to pixels

        # Robot dimension
        self.w = width
        self.x = startpos[0]
        self.y = startpos[1]
        self.theta = startpos[2]
        self.goalx = goalpos[0]
        self.goaly = goalpos[1]
        self.goaltheta = goalpos[2]
        self.a = 20
        self.v = 0 * self.m2p
        self.omega = 1 * self.m2p
        self.maxspeed = 1 * self.m2p
        self.minspeed = - 1 * self.m2p
        self.reward = 0
        # self.nodes = 0

        # Parameters of egreedy

        self.last_atitude = "Random"
        self.action = 4
        self.n = 0
        self.A = []
        self.R = []
        self.Q = []
        self.N = []
        self.n_arms = np.arange(4)
        self.OCC = np.zeros(4)

        # Graphics
        self.img = pygame.image.load(robotImg)
        self.rotated = self.img
        self.rect = self.rotated.get_rect(center=(self.x, self.y))

        # goals
        self.centerpoint = (600, 300)
        self.goal0 = (600, 100, math.radians(270))
        self.goal1 = (800, 300, math.radians(0))
        self.goal2 = (600, 500, math.radians(90))
```

```
        self.goal3 = (400, 300, math.radians(180))
        self.goals = (self.goal0, self.goal1, self.goal2, self.goal3)

    def ajusteAngulo(self, angulo):
        angulo = np.mod(angulo, 2 * np.pi)
        if angulo > np.pi:
            angulo = angulo - 2 * np.pi
        return angulo

    def Controller(self, Krho=2/11, Kalpha=345/823, Kbeta=-2/11, forward=False):
#Krho=2/11, Kalpha=345/823, Kbeta=-2/11,

        self.Krho = Krho
        self.Kalpha = Kalpha
        self.Kbeta = Kbeta

        Dx = self.goalx - self.x
        Dy = self.goaly - self.y
        Dth = self.goaltheta - self.theta

        # Control just for line
        #self.v = Dx * math.cos(self.theta) + Dy * math.sin(self.theta)
        #self.omega = (-1/self.a) * math.sin(self.theta) * Dx + (1/self.a) *
math.cos(self.theta) * Dy

        # P control

        rho = np.sqrt(Dx**2 + Dy **2)
        gamma = self.ajusteAngulo(math.atan2(Dy, Dx))

        alpha = self.ajusteAngulo(gamma - self.theta)

        beta = self.ajusteAngulo(self.goaltheta - gamma)

        self.v = min(self.Krho * rho, self.maxspeed)
        # Walk forward or backward
        if forward == False:
            if np.abs(alpha) > math.pi / 2:
                self.v = -self.v
                alpha = self.ajusteAngulo(alpha + math.pi)
                beta = self.ajusteAngulo(beta + math.pi)
        self.omega = self.Kalpha * alpha + self.Kbeta * beta

    def Reinforcement(self, pointok, bandit, epsilon):

        self.Controller()

        self.epsilon = epsilon
        nmax = 500

        Dx = self.goalx - self.x
        Dy = self.goaly - self.y

        self.point_x = pointok[0]
        self.point_y = pointok[1]
        dx_point = self.x - self.point_x
        dy_point = self.y - self.point_y
        rho_point = np.sqrt(dx_point**2 + dy_point**2)

        rho = np.sqrt(Dx**2 + Dy**2)
```

```
        rho_center = np.sqrt((self.x - self.centerpoint[0])**2 + (self.y -
self.centerpoint[1])**2)

        if self.goalx == self.goal0[0] and robot.goaly == self.goal0[1]:
            self.action = 0
        elif self.goalx == self.goal1[0] and robot.goaly == self.goal1[1]:
            self.action = 1
        elif self.goalx == self.goal2[0] and robot.goaly == self.goal2[1]:
            self.action = 2
        elif self.goalx == self.goal3[0] and robot.goaly == self.goal3[1]:
            self.action = 3

        if rho <= 0.01 and rho_center >= 0.01:
            # self.Controller(forward=False)
            self.goalx = self.centerpoint[0]
            self.goaly = self.centerpoint[1]

            self.A.append(self.action)
            self.R.append(bandit[self.action])
            self.N.append(self.n)
            if self.n == 0:
                self.Q.append((1 / (self.n + 1)) * self.R[self.n])
            else:
                self.Q.append(self.Q[self.n - 1] + (1 / self.n) *
(self.R[self.n] - self.Q[self.n - 1]))
            self.OCC[self.action] += 1
            self.n = self.n + 1

            if rho_point <= 0.02:
                self.reward = self.reward + 1

        if rho <= 0.1 and rho_center <= 0.01:
            # self.Controller(forward=True)
            choice = random.uniform(0, 1) < epsilon #0.14~
            if choice == 0:
                self.last_atitude = "exploit"
            else:
                self.last_atitude = "explore"

            if choice == 0:
                b = max(self.R)
                a1 = self.R.index(b)
                self.action = self.A[a1]
            else:
                self.action = random.randint(0, 3)

            print(self.action)
            print(self.last_atitude)
            print(self.A)

            ch_goal = self.goals[self.action]
            self.goalx = ch_goal[0]
            self.goaly = ch_goal[1]
            self.goaltheta = ch_goal[2]

        if self.n >= nmax:
            pygame.quit()

    def draw(self, map):
        map.blit(self.rotated, self.rect)
```

```
    def move(self, event=None):

        self.x += (self.v * math.cos(self.theta)) * dt #- (self.a *
math.sin(self.theta) * self.omega) * dt
        self.y += (self.v * math.sin(self.theta)) * dt #+ (self.a *
math.cos(self.theta) * self.omega) * dt
        self.theta += self.omega * dt
        # Reset theta over 1 lap complete
        if self.theta > 2 * math.pi or self.theta < - 2 * math.pi:
            self.theta = 0

        self.rotated = pygame.transform.rotozoom(self.img, math.degrees(-
self.theta), 1)
        self.rect = self.rotated.get_rect(center=(self.x, self.y))

        # self.Controller()
        self.Reinforcement(environment.point_ch, environment.bandit, 1)

        if event is not None:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_KP4:
                    self.goalx += 0.001 * self.m2p
                elif event.key == pygame.K_KP1:
                    self.goalx -= 0.001 * self.m2p
                elif event.key == pygame.K_KP6:
                    self.goaly += 0.001 * self.m2p
                elif event.key == pygame.K_KP3:
                    self.goaly -= 0.001 * self.m2p
                elif event.key == pygame.K_KP7:
                    self.goaltheta += math.degrees(0.00033)
                elif event.key == pygame.K_KP9:
                    self.goaltheta -= math.degrees(0.00033)

# Initialisation
pygame.init()

# Start Position
start = (600, 300, math.radians(0))
goal = (600, 100, math.radians(-90))

# Dimensions
dims = (600, 1200)

# Score Points
point_size = (10, 10)
point0 = (600, 100, math.radians(270))
point1 = (800, 300, math.radians(0))
point2 = (600, 500, math.radians(90))
point3 = (400, 300, math.radians(180))
points = (point0, point1, point2, point3)
center_point = (600, 300, math.radians(0))
sorted_point = random.choice(points)
goal = sorted_point

# Running or not
running = True

# The enviroment
environment = Envir(dims)

# The robot
```

```
robot = Robot(start, goal,
"/home/julio/Documentos/Projetos/TCC/testerobozinho.png", 0.01 * 3779.52)

dt = 0
lasttime = pygame.time.get_ticks()
# Simulation loop
while running:
    # activate the quit button
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
        robot.move(event)

    dt = (pygame.time.get_ticks() - lasttime) / 100
    lasttime = pygame.time.get_ticks()
    pygame.display.update()
    environment.map.fill(environment.black)
    robot.move()
    robot.draw(environment.map)
    environment.point()
    environment.trail((robot.x, robot.y))
    environment.robot_frame((robot.x, robot.y), robot.theta)
    environment.write_info('%.3f'%(robot.v), '%.4f'%(robot.omega))
    environment.write_info_cur(int(robot.x), int(robot.y), robot.theta)
    environment.write_info_goal(int(robot.goalx), int(robot.goaly),
robot.goaltheta)
    environment.write_info_reward(robot.n, robot.reward)
    environment.coord()

# Plot do robô
fig, (ax1, ax2) = plt.subplots(2, figsize=(8, 6))
    # Plot de recompensa ao longo da run
ax1.set_xlabel("Passo")
ax1.set_ylabel("Recompensa média")
ax1.plot(robot.N, robot.Q)
    # Plot ações ao longo da run
labels = ["Norte", "Leste", "Sul", "Oeste"]
ticks = np.arange(len(labels))
ax2.set_yticks(ticks)
ax2.set_yticklabels(labels)
s = 8 * np.ones(len(robot.A))
ax2.scatter(robot.N, robot.A, s=s)

fig.tight_layout()
plt.show()

# Plot de ocorrência por direção
fig, ax = plt.subplots()
width = 0.35
ax.set_title("Ocorrência em cada ponto")
labels = ["Norte", "Leste", "Sul", "Oeste"]
ticks = np.arange(len(labels))
ax.set_xticks(ticks)
ax.set_xticklabels(labels)
ax.bar(robot.n_arms, robot.OCC, width=width)

fig.tight_layout()
plt.show()
```

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
COORDENAÇÃO DO CURSO DE ENGENHARIA MECÂNICA

Natal, 30 de abril de 2021.

Ao(s) **trinta** dia(s) do mês de **abril** do ano de **dois mil e vinte e um,** às **treze horas e trinta minutos, na plataforma google meet,** instalou-se a banca examinadora do Trabalho de Conclusão de Curso do aluno **JÚLIO FREIRE PEIXOTO GOMES,** matrícula **20170010473,** do curso de Engenharia Mecânica**.** A banca examinadora foi composta pelos seguintes membros: **JONATHA WALLACE DA SILVA ARAÚJO,** orientador; **ÍTALO SABINO ARRAIS BEZERRA,** examinador interno; **LUCAS SOLANO CADENGUE,** examinador externo; **FELIPE PINHEIRO MAIA**, examinador externo. Deu-se início à abertura dos trabalhos pelo **JONATHA WALLACE DA SILVA ARAÚJO,** que após apresentar os membros da banca examinadora, solicitou a (o) candidato (a) que iniciasse a apresentação do trabalho de conclusão de curso, intitulado **"CONTROLE INTELIGENTE APLICADO À ROBÓTICA MÓVEL"**, marcando um tempo de trinta minutos para a apresentação. Concluída a exposição, **JONATHA WALLACE DA SILVA ARAÚJO,** orientador, passou a palavra aos examinadores para arguirem o(a) candidato(a); após o que fez suas considerações sobre o trabalho em julgamento; tendo sido Aprovado, o(a) candidato(a), conforme as normas vigentes na Universidade Federal do Rio Grande do Norte. A versão final do trabalho deverá ser entregue à Coordenação do Curso de Engenharia Mecânica, no prazo de 7 dias; contendo as modificações sugeridas pela banca examinadora e constante na folha de correção anexa. Conforme o que rege o Projeto Político Pedagógico do Curso de Engenharia Mecânica da UFRN, o(a) candidato(a) não será o aprovado(a) se não cumprir as exigências acima.

Jonatha Wallace da Silva Araújo
JONATHA WALLACE DA SILVA ARAÚJO
Orientador

Ítalo Sabino Arrais Bezerra
ÍTALO SABINO ARRAIS BEZERRA
Examinador interno

Lucas Solano Cadengue
LUCAS SOLANO CADENGUE
Examinador interno

Felipe Pinheiro Maia
FELIPE PINHEIRO MAIA
Examinador interno