Desenvolvimento do Sistema de Planejamento da Trajetória de um Robô Móvel Autônomo

Aluno: Jaime Estevam Santos de Queiroz

Orientador: Prof. Dr. Cláudio Rodrigo Torres

Coorientador: Prof. Dr. Wellington Batista de Sousa

Relatório Final referente ao período de vigência de 09/2020 a 08/2021.

INTRODUÇÃO

Segundo o dicionário online Michaelis uma das definições da palavra robô é:

“Aparelho automático, com aspecto humanoide, capaz de se movimentar e executar diferentes tarefas, inclusive algumas geralmente feitas pelo homem.” (Michaelis, 2021)

As áreas de atuação da robótica tendem a expandir cada vez mais com o passar dos anos e com avanços tecnológicos.

Robôs industriais já são comuns, podem ser controlados ou autônomos. Um modelo interessante está na robótica móvel, a diferença que eles proporcionam é bastante significativa, pois as máquinas desse tipo são capazes de transportar grandes cargas e podem alcançar um tempo de trabalho que um ser humano não suportaria sem chegar à exaustão e gerando lesões. É interessante também a capacidade de acesso em lugares que seriam perigosos a uma pessoa, no cenário pandêmico mundial atual essa tarefa é muito importante para ajudar a evitar contaminações em áreas de quarentena.

Estão presentes também em atividades físicas ondem acompanham ou ajudam no treino de artes marciais como o modelo RXT-1 produzido pela STRYK e o Unitree Go1 produzido pela Unitree Robotics, lembrando um cachorro, acompanha na corrida carregando água.

A inteligência artificial (I.A) é um software dotado de algum método teórico desenvolvido com a intenção de melhorar a capacidade autônoma de algum hardware. Uma I.A acrescentada a um robô aumenta exponencialmente sua capacidade de trabalho.

Dentre os robôs móveis, existem os autônomos e destacando aqui apensas dois tipos deles que são AGV (Autonomous Guided Vehicle) e AMR (Autonomous Moble Robots). Referênte ao primeiro, necessita ser guiado por trilhos ou fitas magnéticas, já o segundo pode se localizar no ambiente e criar suas próprias trajetórias.

Um projeto de pesquisa nessa área é extremamente gratificante e uma rica fonte de aprendizado e conhecimento.

FUNDAMENTAÇÃO TEÓRICA

Conceitos de lógica paraconsistente para formação das Células Neurais Artificiais Paraconsistentes de maximização (CNAPmax) e analítica (CNAPa)

Para iniciar este assunto é interessante lembrar da lógica comum onde se trabalha com dois estados, verdadeiro e falso.

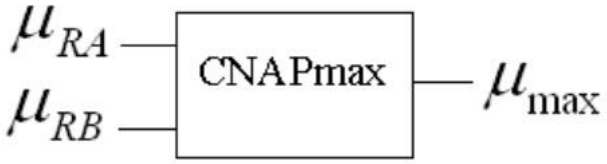
Com a lógica paraconsistente é possível trabalhar com contradições e incertezas.

Facilitando a explicação com exemplos, imaginamos um robô que desvia de obstáculos funcionando com a lógica tradicional, consideramos a seguinte proposição: Existe obstáculo? Pode-se encontrar dois resultados, existe obstáculo (verdadeiro) e não existe obstáculo (falso).

Para exemplificar a lógica paraconsistente consideramos a proposição: Está calor? Neste caso pode estar calor (verdadeiro), pode estar frio (falso), mas também pode estar calor e frio ao mesmo tempo como a sensação de não estar nem calor e nem frio, ainda dentro dessa proposição pode estar frio tendendo para o calor e calor tendendo para o frio, assim podemos trabalhar com muito mais possibilidades de estados diferentes para uma proposição.

De maneira simplificada a Célula Neural Artificial Paraconsistente de maximização analisa os resultados de várias proposições de entrada e resulta no maior, a Figura 1 demonstra graficamente esse conceito:

Figura 1 - Representação Gráfica da Célula Neural Artificial Paraconsistente de maximização.

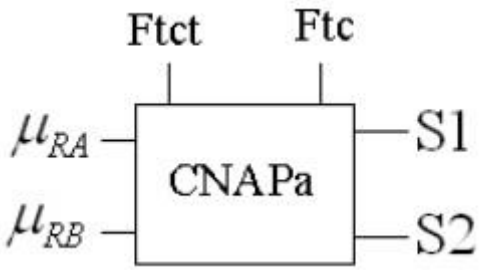


Fonte: TORRES (2010).

Na Figura 1, μ_{RA} e μ_{RB} são as entradas e μ_{max} representa a saída. Exemplo: se μ_{RA} for 0,2 e μ_{RB} 0,6, μ_{max} será 0,6.

A Célula Neural Artificial Paraconsistente analítica tem o aspecto gráfico mostrado na Figura 2:

Figura 2 - Representação gráfica da Célula Neural Artificial Paraconsistente analítica.



Fonte: TORRES (2010).

Para as necessidades do projeto apenas a saída S1 é relevante e os cálculos realizados são:

S1 = μ_E = (Gc+1)/2; onde Gc = μ - λ; μ = μ_{RA}; λ = 1 - μ_{RB};

Vcve = (1+Ftc)/2; Vcfa = (1- Ftc)/2; Vcic = (1+ Ftct)/2; Vcpa = (1- Ftct)/2; μ_{CTR} = (μ+λ)/2;

Para a saída assumir valor de μ_E a condição para S1 deve ser:

Vcic > μ_{CTR} >Vcpa e [(Vcve ≤ μ_E) ou (μ_E ≤ Vcfa)]

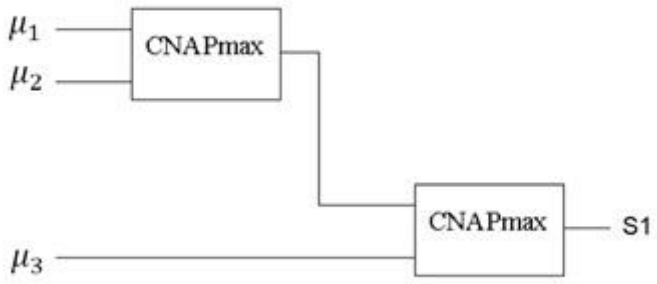
Caso contrário S1 = 0,5.

Onde: Ftct (Fator de tolerância a contradição) e Ftc (fator de tolerância a certeza) são constantes de valores entre 0 e 1; Vcfa (valor de controle de falsidade deve estar entre -1 e 0); Vcic (Valor de controle de inconsistência que deve ser entre 0 e 1); Vcpa (valor de controle de paracompleteza (indeterminado) que deve estar entre -1 e 0).

É possível existir mais que duas entradas em ambas as células utilizando o NAP (Nó de Análise Paraconsistente) interligando a saída de uma célula em uma das entradas de outra.

O exemplo da Figura 3 usa a CNAPmax, mas pode ser feito com outras além desta.

Figura 3 - Exemplo de um Nó de Análise Paraconsistente.



Fonte: TORRES (2010).

Banco de dados

Se trata de um sistema de armazenamento de dados organizados sobre um mesmo tema e de fácil acesso para consulta e manipulação.

Existem principalmente dois tipos de banco de dados, os primeiros são os bancos de dados relacionais, é o mais comum, com inserção de dados em tabelas e linhas, sua linguagem é o SQL (Structured Query Language). Banco de dados não relacionais vai além deste conceito podendo acrescentar dados que não podem ser adicionados em tabelas com colunas e linhas, então pode por exemplo trabalhar com imagens, além disso seus dados podem ser armazenados na nuvem, as linguagens deste tipo são NoSQL e Not Only SQL.

Entre os principais bancos de dados disponíveis estão: Oracle, SQL Server, MySQL, MariaDB, PostgreSQL, NoSQL, MongoDB e Redis.

Linguagem Python

Python é uma linguagem de programação de alto nível, é acessível, gratuita, simplificada, possui uma vasta biblioteca e foi criada com intenções de agilizar a produtividade por ser uma linguagem mais pensada para o humano do que a máquina.

É uma linguagem orientada a objetos que é um modelo de estrutura da programação visando facilitar e organizar programas complexos.

É bastante utilizada para sistemas de inteligência artificial.

METODOLOGIA

O ambiente de desenvolvimento PyCharm utilizando linguagem Python é o elemento central do projeto onde é desenvolvido a programação que controla o sistema de planejamento para as trajetórias.

A programação pede valores para serem digitados de coordenadas baseadas num plano cartesiano com abscissa (x) e ordenada (y), sendo eles a coordenada de origem e a coordenada destino.

Neste ambiente também é feito a lógica que corresponde Célula Neural Artificial Paraconsistente de maximização (CNAPmax) que é responsável por analisar valores que vão de 0 a 1 obtidos do banco de dados e definir o melhor caminho a ser seguido, sendo definido arbitrariamente que a partir de 0,7 é considerado obstáculo e abaixo disso é caminho livre. Os cálculos da célula em questão são:

μ_E = (μ - λ + 1)/2; onde μ = μ_{RA} e λ = μ_{RB}

Se $\mu_E \geq 0.5$, a saída vale μ_{RA}, caso contrário a saída vale μ_{RB}.

A Célula Neural Artificial Paraconsistente analítica (CNAPa) também é programada e funciona como um refinamento do sistema, Suas equações são as seguintes:

Grau de evidencia resultante = S1 = μ_E = (Gc+1)/2; onde Gc = μ - λ; μ = μ_{RA}; λ = 1 - μ_{RB};

Vcve = (1+Ftc)/2; Vcfa = (1- Ftc)/2; Vcic = (1+ Ftct)/2; Vcpa = (1- Ftct)/2; μ_{CTR} = (μ+λ)/2;

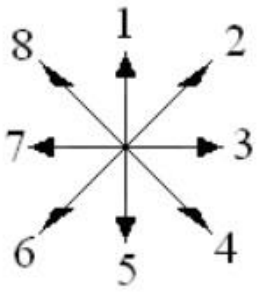
Para a saída assumir valor de μ_E a condição para S1 deve ser:

Vcic > μ_{CTR} >Vcpa e [(Vcve ≤ μ_E) ou (μ_E ≤ Vcfa)]

Caso contrário S1 = 0,5.

Através da ferramenta chamada XAMPP o servidor de banco de dados MariaDB fica disponível, é aqui onde fica os valores que teoricamente seriam coletados pelo subsistema de sensoriamento, como se trata de outro projeto os valores para o teste neste são adicionados manualmente para serem analisados, então finalmente expressa o resultado mostrando-o na tela do PyCharm. O resultado é uma sequência numérica baseada nos movimentos que o robô pode fazer de acordo com a direção, no total são 8 como mostra a Figura 4:

Figura 4 - Representação das direções da sequência numérica emitida pelo sistema.



Fonte: TORRES (2010).

Exemplo de funcionamento:

Passo 1: Pede as coordenadas de origem e destino;

Passo2: Coleta valores do banco de dados por onde é possível se locomover;

Passo 3: Analisa os valores e decide o melhor caminho a ser seguido baseado onde tem ou não obstáculos;

Passo 4: Expressa o resultado em forma de sequência numérica. Exemplo 1: 11111111; aqui o robô avançará 8 coordenadas na direção “1” que foi definido como sendo “andar para frente”. No formato (x, y), se a origem for (0, 0) o destino será (0,8).

Exemplo 2: 33335555; neste o robô virá para a direita, avança 4 coordenadas para a direita, vira para baixo e anda 4 coordenadas para baixo. Se origem (0, 0), destino será (4, -4).

Descrição das principais funções do código

A partir deste ponto será relatado um descritivo das principais etapas da programação com o objetivo de relatar mais detalhadamente o código para possibilitar e facilitar a continuidade e/ou expansão do projeto.

Apesar de não se tratar de um robô físico, e sim um sistema de planejamento de sua trajetória será adotado um método de explanação onde é considerado que realmente há um robô, facilitando assim a explicação e entendimento do sistema.

E com o objetivo de facilitar a didática do texto, os nomes das funções do programa serão citados em negrito e entre aspas, por exemplo: **"função"**.

É possível escolher o tamanho do plano cartesiano a ser considerado para o trajeto do robô, por convenção e para testes é interessante escolher 11, assim ele criará virtualmente um ambiente simulado com 11 valores nas abscissas e ordenadas e o teste acontece calculando o trajeto do robô até no máximo 8 casas do plano. Neste momento também é criado através da função feita na programação "**CriarBanco"** uma simulação do sistema de sensoriamento onde tal função coloca valores de obstáculos aleatórios a todos os pontos do plano cartesiano, podendo ter os filtros de criar apenas valores considerados obstáculos, apenas valores não considerados obstáculos e ambos.

É possível escolher o número de pontos no plano a serem analisadas pelas células paraconsistentes, por costume usa-se 3.

As duas etapas citadas não são obrigatórias executar em todos os testes.

O passo seguinte é onde será escolhido a origem e o destino do sistema.

Então começa a análise do algoritmo planejador. Dependendo do resultado da análise seguirá para ramificações onde o programa é semelhante, então será descrito apenas uma delas pois é valido para ambas, a não ser quando o destino é igual a origem, assim o programa finaliza.

Primeiro é chamada à função **"fixador"**, esta é responsável por criar um valor adicional no banco de dados colocando obstáculos pelo trajeto que o robô faz para evitar conflitos.

Na próxima etapa a função **“mov_contrario”**, inverte os valores de movimentos pois o programa precisa realizar a análise do destino para a origem. Por exemplo, se o movimento for 1 que significa frente, ele considera 5 ou seja, para traz ou ré.

Agora é o momento em que começará as análises paraconsistentes, primeiro chamando **“banco”** para obter o valor do ponto do plano cartesiano, então a **“CNAPmax”**, será considerado o maior valor dos três pontos à frente da direção desejada a ser analisada, por exemplo: se a localização do robô for X = 0, Y = 0 e o movimento for 1, o sistema fará a varredura dos pontos X = 0, Y = 1; X = 0, Y = 2; X = 0, Y = 3. O maior valor presente nestes pontos será considerado para a continuação da análise.

Sobre a função **“analise1”**: Por enquanto não foi mencionado obstáculos, mas e se houver? O robô deve buscar dentro dos 8 movimentos possíveis os 2 “vizinhos” da direção em que se encontra o obstáculo. Exemplo: se for necessário realizar movimento 1 e ele estiver com obstruções, o robô analisará os movimentos 2 e 8, caso for 3 então o robô analisará 2 e 4, com a Fig. 4 fica mais fácil visualizar o conceito.

Caso o robô necessite realizar o movimento 3 e este contém um obstáculo, ela analisará os movimentos 2 e 4, aqui entra a função **“CNAPa”** que faz um filtro analisando os dois “vizinhos” do 3 e escolhe o melhor caminho, ou seja, com menos obstruções, sempre analisando três pontos do plano cartesiano à frente para realização dos cálculos. E se os dois estiverem com obstáculos? A análise seguirá analisando os próximos “vizinhos” que agora seriam 1 e 5, depois 8 e 6 e por último o 7 que seria o caminho contrário ao 3. Caso estiver impossibilitado de seguir, o robô ficará imóvel.

É necessário ativar “**mov_contrario”** com finalidade gráfica no momento de mostrar o código da sequência numérica de movimentos correta e também para seguir o método de análise programado.

A função **“planejamento_de_movimentos”** realiza os cálculos necessários baseado no número do movimento para obter o próximo.

O programa realizará todos esses passos até o planejamento da trajetória chegar ao seu destino.

Por fim a função **“deletador”** zera todos os valores que **“fixador”** adicionou no banco de dados para evitar conflitos e que a longo prazo tudo seja considerado um obstáculo.

RESULTADOS E DISCUSSÃO

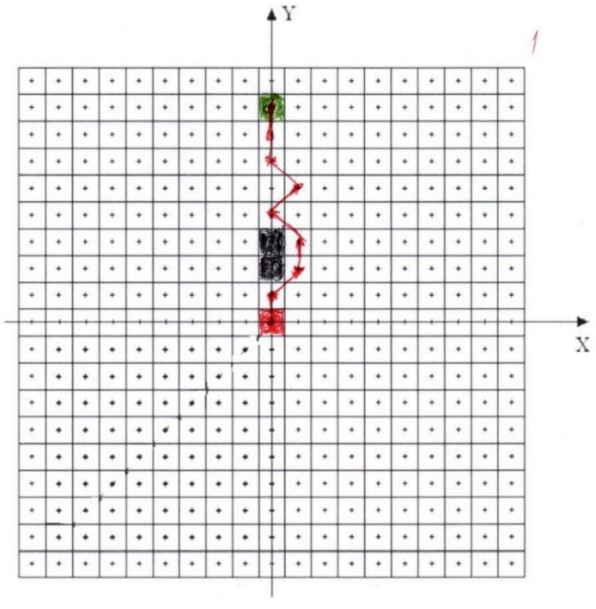
Para demonstrar os resultados de um sistema de planejamento para trajetórias de um robô móvel autônomo feito em software sem uma plataforma mecânica foi realizado em forma de simulação onde é possível traduzir os resultados expressados pelo sistema, isso é feito com base na representação da sequência numérica que representam as direções (Fig. 4). Desenhando manualmente em um papel com um mapa baseado no sistema cartesiano fica nítido o caminho que um robô seguiria equipado com a versão do software desenvolvido até agora.

A seguir serão apresentados os testes mais relevantes feitos, onde os quadrados representam os pontos das coordenadas, sendo pintado de vermelho é a origem, preto são obstáculos, verde é o destino e as setas em vermelho representam a trajetória que a plataforma mecânica faria.

As coordenadas que representam a origem, o destino e demonstram os obstáculos serão expressas no formato (X, Y).

O teste 1 (Figura 5) tem como origem (0, 0), destino (0, 8) e obstáculos em (0,2) e (0,3). A sequência numérica gerada pelo sistema foi: [1, 2, 1, 8, 2, 8, 1, 1].

Figura 5 – Teste 1.

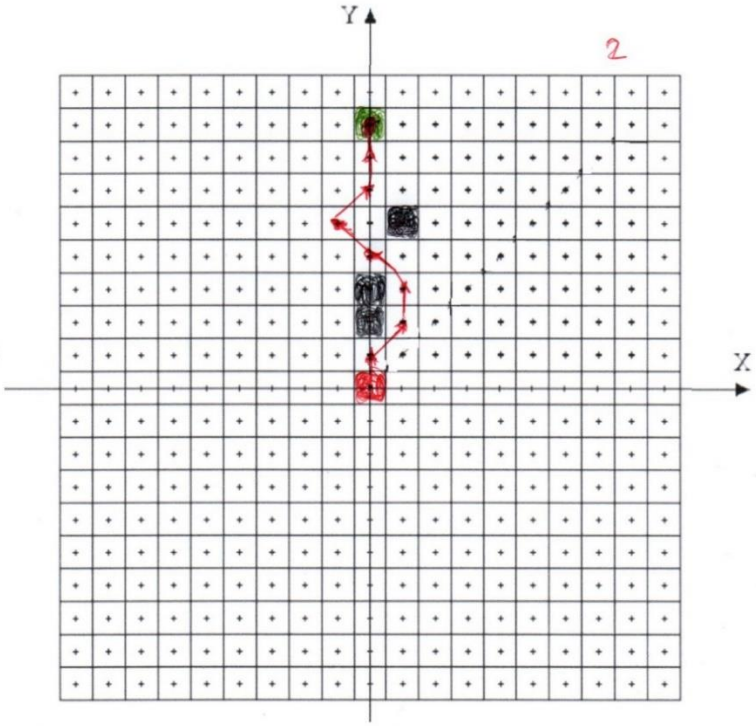


Fonte: Do autor (2021).

Teste 2 (Figura 6) origem (0, 0), destino (0, 8) e obstáculos em (0, 2), (0, 3) e (1, 5).

Sequência gerada pelo sistema: [1, 2, 1, 8, 8, 2, 1, 1].

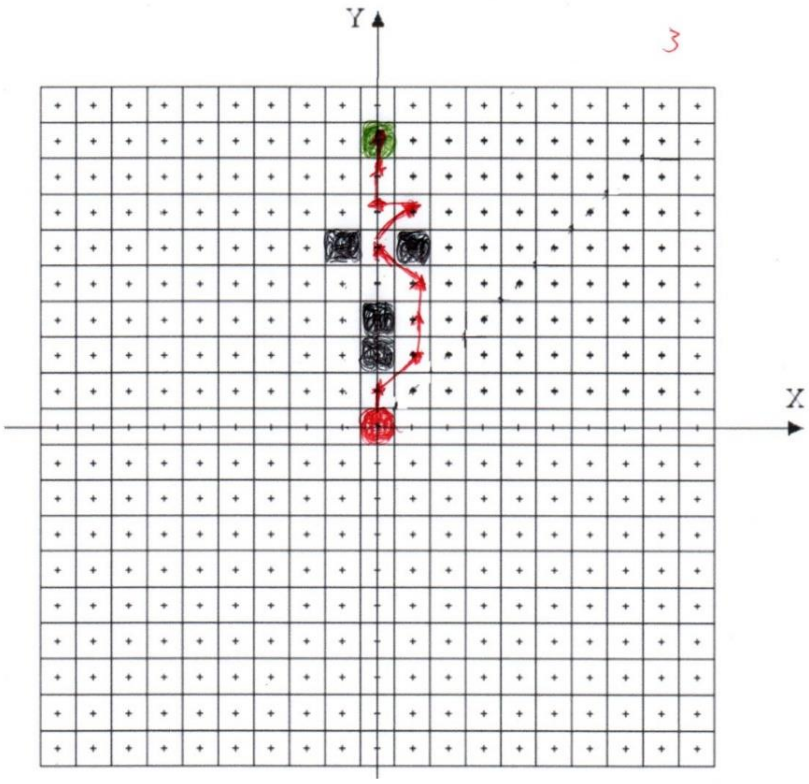
Figura 6 – Teste 2.



Fonte: Do autor (2021).

Teste 3 (Figura 7) origem (0, 0), destino (0, 8) e obstáculos em (0, 2), (0, 3), (1, 5) e (-1, 5). Sequência gerada pelo sistema: [1, 2, 1, 1, 8, 2, 7, 1, 1].

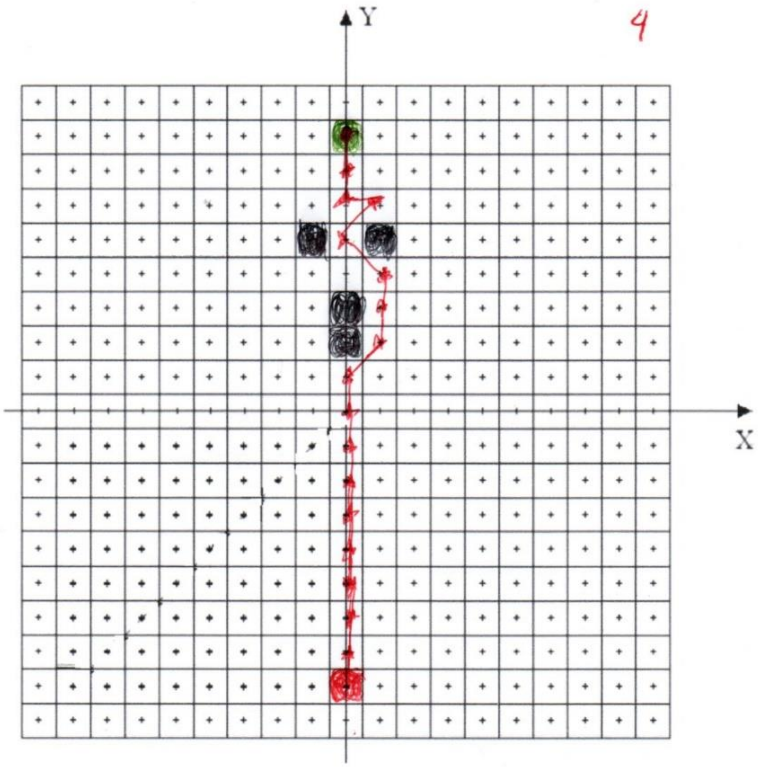
Figura 7 – Teste 3.



Fonte: Do autor (2021).

Teste 4 (Figura 8) origem (0, -8), destino (0, 8) e obstáculos em (0, 2), (0, 3), (1, 5) e (-1, 5). Sequência gerada pelo sistema: [1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 8, 2, 7, 1, 1].

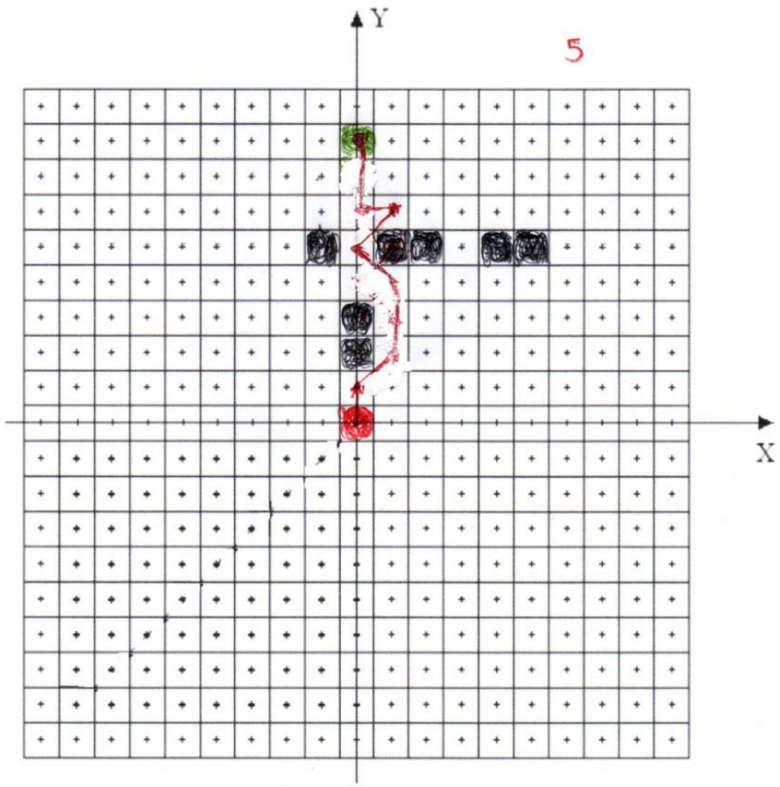
Figura 8 – Teste 4.



Fonte: Do autor (2021).

Teste 5 (Figura 9) origem (0, 0), destino (0, 8) e obstáculos em (0, 2), (0, 3), (1, 5), (2, 5), (4, 5), (5, 5) e (-1, 5). Sequência gerada pelo sistema: [1, 2, 1, 1, 8, 2, 7, 1, 1].

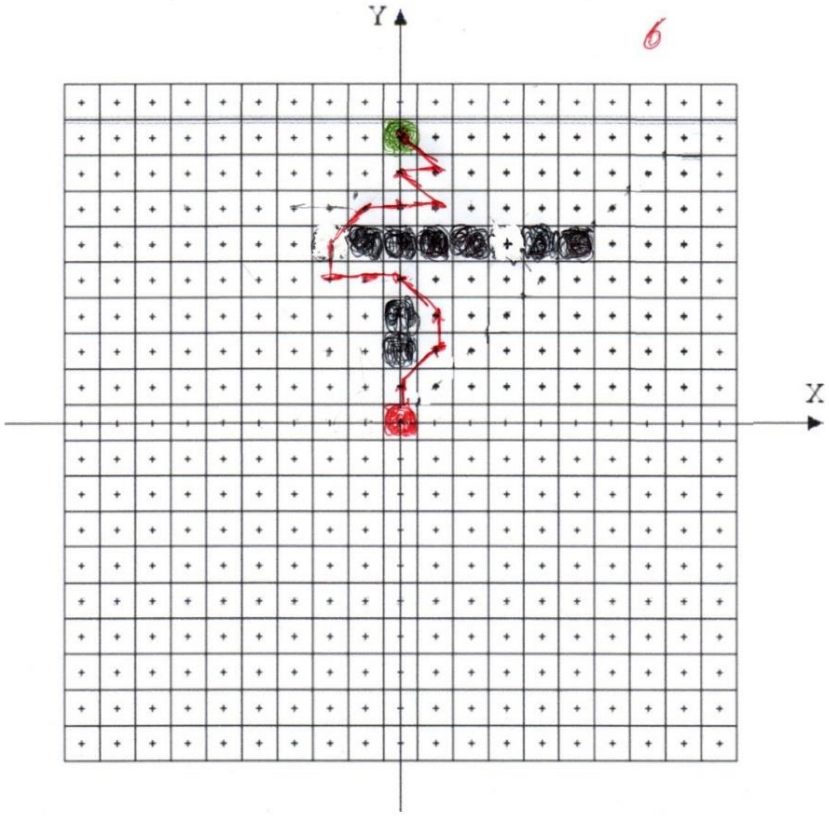
Figura 9 – Teste 5.



Fonte: Do autor (2021).

Teste 6 (Figura 10) origem (0, 0), destino (0, 8) e obstáculos em (0, 2), (0, 3), (0, 5), (1, 5), (2, 5), (4, 5), (5, 5) e (-1, 5). Sequência gerada pelo sistema: [1, 2, 1, 8, 7, 7, 1, 2, 3, 3, 8, 3, 8].

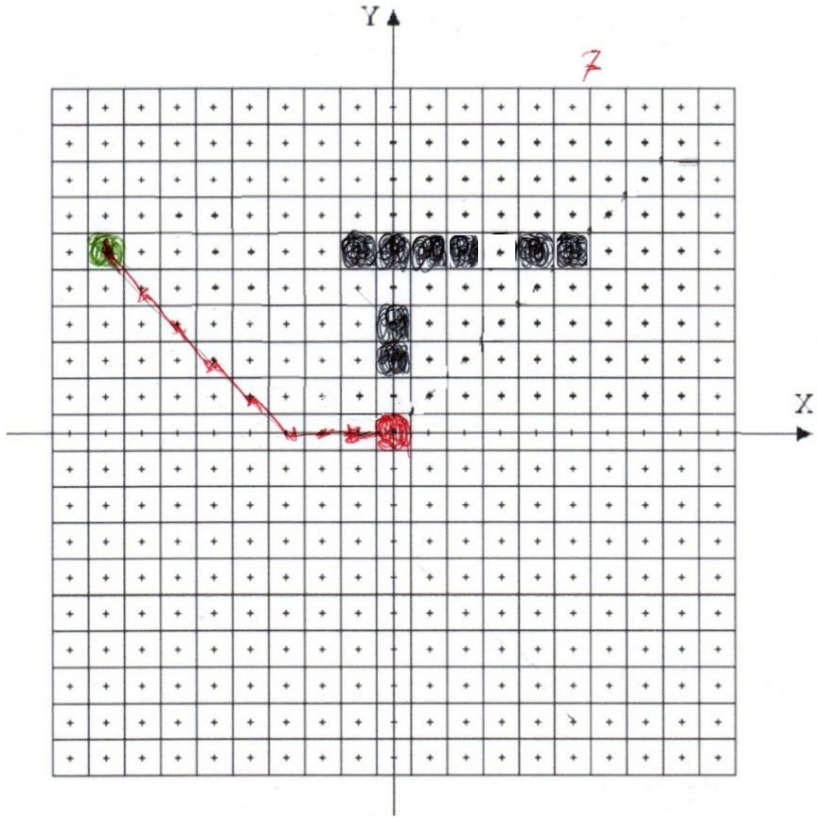
Figura 10 – Teste 6.



Fonte: Do autor (2021).

Teste 7 (Figura 11) origem (0, 0), destino (-8, 5) e obstáculos em (0, 2), (0, 3), (0, 5), (1, 5), (2, 5), (4, 5), (5, 5) e (-1, 5). Sequência gerada pelo sistema: [7, 7, 7, 8, 8, 8, 8, 8].

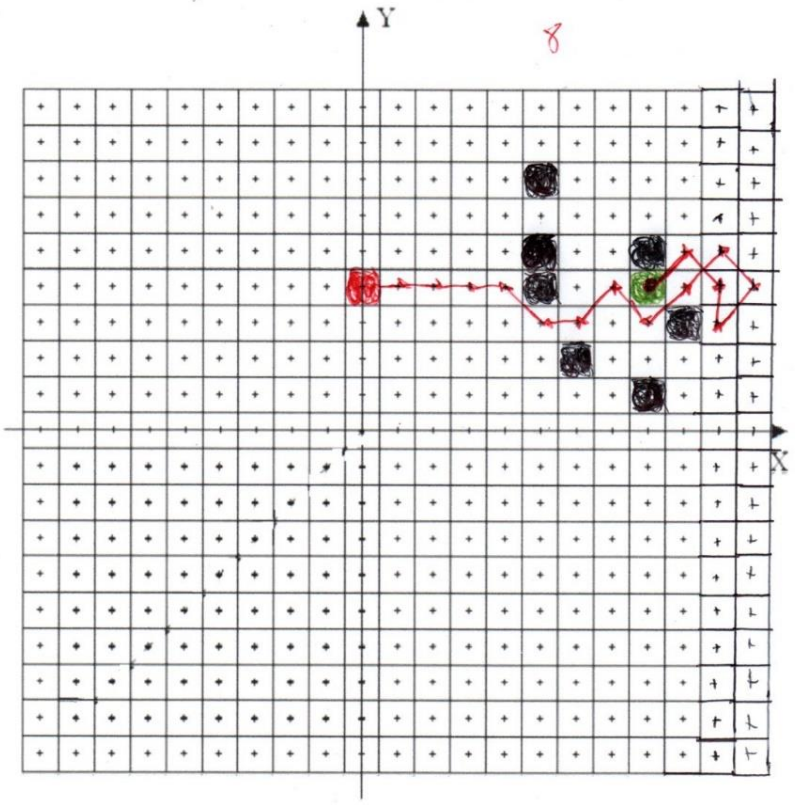
Figura 11 – Teste 7.



Fonte: Do autor (2021).

Teste 8 (Figura 12) origem (0, 4), destino (8, 4) e obstáculos em (5, 4), (5, 5), (5, 7), (6, 2), (8, 1), (8, 5) e 9, 3). Sequência gerada pelo sistema: [3, 3, 3, 3, 4, 3, 2, 4, 2, 2, 4, 6, 1, 8, 6].

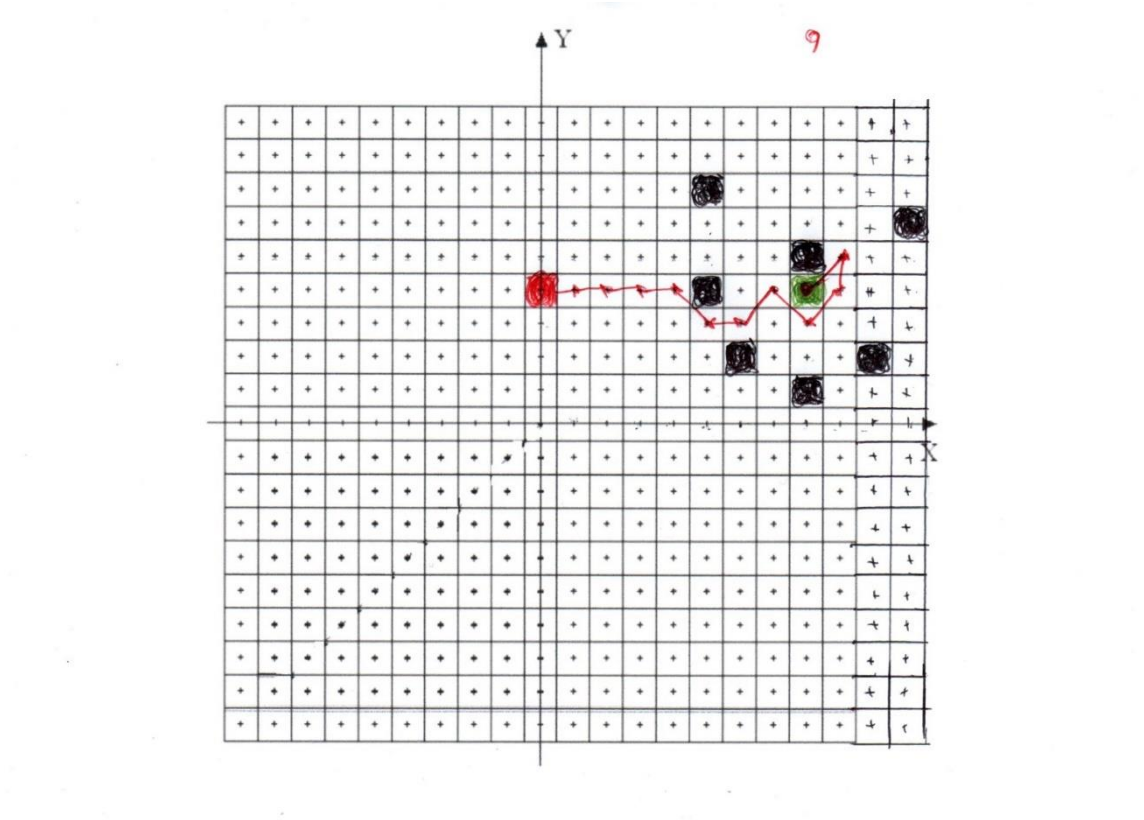
Figura 12 – Teste 8.



Fonte: Do autor (2021).

Teste 9 (Figura 13) origem (0, 4), destino (8, 4) e obstáculos em (5, 4), (5, 7), (6, 2), (8, 1), (8, 5), (10, 2) e (11, 6). Sequência gerada pelo sistema: [3, 3, 3, 3, 4, 3, 2, 4, 2, 1, 6].

Figura 13 – Teste 9.



Fonte: Do autor (2021).

Nas simulações de teste é possível notar rotas desnecessárias para chegar até a origem, em estudos sobre o caso, a solução encontrada foi adicionar uma programação que faça com que realize todo o planejamento e use somente o primeiro valor da sequência, depois rodar novamente todo o planejamento e fazer o mesmo até que o planejamento encontre o destino.

CONCLUSÕES

Visando a intenção primária do projeto que é criar um sistema em software de trajetórias para robôs móveis autônomos, os resultados encontrados satisfazem a condição de partir de uma origem e chegar ao destino.

O detalhe sobre a trajetória conter movimentos desnecessários, que apesar da solução aparentemente fácil não foi possível ser resolvida, a programação adicionada não causava efeitos desejados, sendo assim foi preferível manter como estava. Apesar de ser interessante concertar esse pequeno problema no sistema de planejamento, não é estritamente necessário, pois isso pode ser realizado no projeto de uma plataforma mecânica.

Algumas sugestões para uma possível continuidade do projeto seriam: concerto dos movimentos desnecessários da trajetória no planejamento, melhorar o código deixando-o mais simples e fácil de ser interpretado, adicionar um sistema de sensoriamento e projetar uma plataforma mecânica capaz de comportar toda a lógica do atual projeto para um teste em ambiente real.

A realização deste projeto de pesquisa proporcionou um riquíssimo conhecimento que pode ser usado em muitas outras aplicações e ajudou a fortalecer grandiosamente o processo de graduação em um curso superior.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Paulo. **Coronavírus: China utiliza robôs para atender pacientes em quarentena.** Disponível em: <https://www.techtudo.com.br/noticias/2020/01/robos-ajudam-pacientes-em-quarentena-na-china-por-conta-do-coronavirus.ghtml>. Acesso em 12 fev. 2021.

ARAÚJO, Aurélio. **Pet eletrônico: cão-robô chinês pode carregar sua água enquanto você corre**. Disponível em: <https://www.uol.com.br/tilt/noticias/redacao/2021/06/10/pet-eletronico-cao-robo-chines-pode-carregar-sua-agua-enquanto-voce-corre.htm>. Acesso em 10 ago. 2021.

HEWLETT PACKARD ENTERPRISE. **OQUE É INTELIGÊNCIA ARTIFICIAL?** Disponível em: <https://www.hpe.com/br/pt/what-is/artificial-intelligence.html#resources>. Acesso em 10 ago. 2021.

MICHAELIS. **Dicionário Brasileiro da Língua Portuguesa**. Disponível em: <https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/robo>. Acesso em 10 ago. 2021.

PORAZZA, Rafael. **Robôs móveis: qual sua importância na indústria?** Disponível em: <https://www.pollux.com.br/blog/robos-moveis-qual-sua-importancia-na-industria/>. Acesso em: 12 fev. 2021.

SOUZA, Ivan. **Banco de dados: saiba o que é, os tipos, e a importância para o site da sua empresa**. Disponível em: <https://rockcontent.com/br/blog/banco-de-dados/>. Acesso em 13 fev.2021.

STRYK. **RXT-1 CAN YOU DEFEND A STRIKE?** Disponível em: <https://www.strykusa.com/>. Acesso em 10 ago. 2021.

TORRES, Cláudio R. **Sistema Inteligente Baseado Na Lógica Paraconsistente Anotada Evidencial Eτ Para Controle E Navegação De Robôs Móveis Autônomos Em Um Ambiente Não Estruturado.**2010. Tese (Doutorado) - Universidade Federal De Itajubá.2010.

Assinatura do aluno

ANEXO – Código em Python da programação desenvolvida

```
# Funções que determinam os cálculos dos movimentos
def Movimento1():
    global x
    global y

    #    x += 0
    y += 1
    print(f'X{x}; Y{y}')
    return


def Movimento2():
    global x
    global y

    x += 1
    y += 1
    print(f'X{x}; Y{y}')
    return


def Movimento3():
    global x
    global y

    x += 1
    #    y = 0
    print(f'X{x}; Y{y}')
    return


def Movimento4():
    global x
    global y

    x += 1
    y -= 1
    print(f'X{x}; Y{y}')
    return


def Movimento5():
    global x
    global y

    #    x = 0
    y -= 1
    print(f'X{x}; Y{y}')
    return


def Movimento6():
    global x
    global y

    x -= 1
```

```
    y -= 1
    print(f'X{x}; Y{y}')
    return


def Movimento7():
    global x
    global y

    x -= 1
    #    y = 0
    print(f'X{x}; Y{y}')
    return


def Movimento8():
    global x
    global y

    x -= 1
    y += 1
    print(f'X{x}; Y{y}')
    return


def planejamento_de_movimentos():
    global Mov
    global xd
    global yd
    global x
    global y
    global xm
    global ym


    if Mov == 1:
        print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

        yd = yd - 1

        ym -= 1

    elif Mov == 2:
        print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

        xd = xd - 1
        yd = yd - 1

        xm -= 1
        ym -= 1

    elif Mov == 3:
        print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

        xd = xd - 1

        xm -= 1

    elif Mov == 4:
        print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')
```

```
            xd = xd - 1
            yd = yd + 1

            xm -= 1
            ym += 1

        elif Mov == 5:
            print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

            yd = yd + 1

            ym += 1

        elif Mov == 6:
            print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

            xd = xd + 1
            yd = yd + 1

            xm += 1
            ym += 1

        elif Mov == 7:
            print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

            xd = xd + 1

            xm += 1

        elif Mov == 8:
            print(f'MOVIMENTO ---|||||||(((((((( {Mov} ))))))))|||||||---')

            xd = xd + 1
            yd = yd - 1

            xm += 1
            ym -= 1


    def mov_contrario():
        global Mov

        if Mov == 1:
            Mov = 5
        elif Mov == 2:
            Mov = 6
        elif Mov == 3:
            Mov = 7
        elif Mov == 4:
            Mov = 8
        elif Mov == 5:
            Mov = 1
        elif Mov == 6:
            Mov = 2
        elif Mov == 7:
            Mov = 3
        elif Mov == 8:
            Mov = 4


    # Função para determinar a busca de novos caminhos caso encontre algum
```

```
obstáculo
def analise1():
    global valores
    global Mov
    global Mov2
    global movdir
    global movesq
    global s
    global s0
    global saida
    global saida0
    global l2
    global l2a
    global l3
    global x
    global y
    global xd
    global yd

    if s >= 0.7:
        Mov2 = Mov

        if Mov2 == 8:
            movdir = Mov2 - 7
            movesq = Mov2 - 1

        elif Mov2 == 1:
            movesq = Mov2 + 7
            movdir = Mov2 + 1
        else:
            movdir = Mov2 + 1
            movesq = Mov2 - 1

        # PRIMEIRA ANÁLISE
        print(f'Análisando movimento {movdir} e {movesq}')
        Mov = movdir

        x = xd
        y = yd

        banco()

        x = xd
        y = yd

        print(valores)
        l2 = l3  # mantém variavel do laço "for" da função CNAPmax
para voltar a análise de células
        CNAPmax()
        s0 = s  # variavel para diferenciar dois caminhos diferentes
        l2a = l3
        CNAPa()
        saida0 = saida
        Mov = movesq

        x = xd
        y = yd

        banco()

        x = xd
```

```
        y = yd

        print(valores)
        l2 = l3  # mantém variavel do laço "for" da função CNAPmax
para voltar a análise de célular
        CNAPmax()
        l2a = l3
        CNAPa()
        print(f'Movimento {movdir} = {s0}; Movimento {movesq} = {s}')
        if s0 >= 0.7 and s >= 0.7:
            print(f'Movimento {movdir} e Movimento {movesq} estão
obstruidos')
            print('Buscando outros movimentos...')

            # SEGUNDA ANÁLISE

            if Mov2 == 8:
                movdir = Mov2 - 6
                movesq = Mov2 - 2

            elif Mov2 == 1:
                movesq = Mov2 + 6
                movdir = Mov2 + 2

            elif Mov2 == 7:
                movdir = Mov2 - 6
                movesq = Mov2 - 2

            elif Mov2 == 2:
                movesq = Mov2 + 6
                movdir = Mov2 + 2
            else:
                movdir = Mov2 + 2
                movesq = Mov2 - 2
            print(f'Análisando movimento {movdir} e {movesq}')
            Mov = movdir

            x = xd
            y = yd

            banco()

            x = xd
            y = yd

            print(valores)
            l2 = l3  # mantém variavel do laço "for" da função CNAPmax
para voltar a análise de célular
            CNAPmax()
            s0 = s  # variavel para diferenciar dois caminhos
diferentes
            l2a = l3
            CNAPa()
            saida0 = saida
            Mov = movesq

            x = xd
            y = yd

            banco()
```

```
            x = xd
            y = yd

            print(valores)
            l2 = l3  # mantém variavel do laço "for" da função CNAPmax
para voltar a análise de célular
            CNAPmax()
            l2a = l3
            CNAPa()

            if s0 >= 0.7 and s >= 0.7:
                print(f'Movimento {movdir} = {s0}; Movimento {movesq}
= {s}')
                print(f'Movimento {movdir} e Movimento {movesq} estão
obstruidos')
                print('Buscando outros movimentos...')

                # TERCEIRA ANÁLISE

                if Mov2 == 8:
                    movdir = Mov2 - 5
                    movesq = Mov2 - 3

                elif Mov2 == 1:
                    movesq = Mov2 + 5
                    movdir = Mov2 + 3

                elif Mov2 == 7:
                    movdir = Mov2 - 5
                    movesq = Mov2 - 3

                elif Mov2 == 2:
                    movesq = Mov2 + 5
                    movdir = Mov2 + 3

                elif Mov2 == 6:
                    movdir = Mov2 - 5
                    movesq = Mov2 - 3

                elif Mov2 == 3:
                    movesq = Mov2 + 5
                    movdir = Mov2 + 3
                else:
                    movdir = Mov2 + 3
                    movesq = Mov2 - 3
                print(f'Análisando movimento {movdir} e {movesq}')
                Mov = movdir

                x = xd
                y = yd

                banco()

                x = xd
                y = yd

                print(valores)
                l2 = l3  # mantém variavel do laço "for" da função
CNAPmax para voltar a análise de célular
                CNAPmax()
                s0 = s  # variavel para diferenciar dois caminhos
```

```
diferentes
                l2a = l3
                CNAPa()
                saida0 = saida
                Mov = movesq

                x = xd
                y = yd

                banco()

                x = xd
                y = yd

                print(valores)
                l2 = l3  # mantém variavel do laço "for" da função
CNAPmax para voltar a análise de célular
                CNAPmax()
                l2a = l3
                CNAPa()

                if s0 >= 0.7 and s >= 0.7:
                    print(f'Movimento {movdir} = {s0}; Movimento
{movesq} = {s}')
                    print(f'Movimento {movdir} e Movimento {movesq}
estão obstruidos')
                    print('Buscando outros movimentos...')

                    # QUARTA ANÁLISE (ULTIMA)

                    if Mov2 >= 1 and Mov2 < 5:
                        Mov2 += 4
                    elif Mov2 >= 5 and Mov2 < 9:
                        Mov2 -= 4
                    print(f'Análisando movimento {Mov2}')
                    #                Mov = movdir
                    Mov = Mov2

                    x = xd
                    y = yd

                    banco()

                    x = xd
                    y = yd

                    print(valores)
                    l2 = l3  # mantém variavel do laço "for" da função
CNAPmax para voltar a análise de célular
                    CNAPmax()
                    if s >= 0.7:
                        print(f'Movimento {Mov2} = {s}')
                        print('ROBÔ TRAVADO!')
                    else:
                        print(f'Movimento {Mov2} = {s}')
                        print(f'Movimento {Mov2} está livre')
                        Mov = Mov2

                elif s0 < 0.7 and s < 0.7:
                    print(f'Movimento {movdir} = {s0}; Movimento
{movesq} = {s}')
```

```
                    print(f'Movimento {movdir} e movimento {movesq}
estão livres')
                    if saida0 < saida:
                        print(f'Movimento {movdir} = CNAPa: {saida0} é
melhor')
                        Mov = movdir
                    else:
                        print(f'Movimento {movesq} = CNAPa: {saida} é
melhor')
                        Mov = movesq
                elif s0 < 0.7:
                    print(f'Movimento {movdir} = {s0}; Movimento
{movesq} = {s}')
                    print(f'Movimento {movdir} está livre')
                    Mov = movdir
                elif s < 0.7:
                    print(f'Movimento {movdir} = {s0}; Movimento
{movesq} = {s}')
                    print(f'Movimento {movesq} está livre')
                    Mov = movesq

            elif s0 < 0.7 and s < 0.7:
                print(f'Movimento {movdir} = {s0}; Movimento {movesq}
= {s}')
                print(f'Movimento {movdir} e movimento {movesq} estão
livres')
                if saida0 < saida:
                    print(f'Movimento {movdir} = CNAPa: {saida0} é
melhor')
                    Mov = movdir
                else:
                    print(f'Movimento {movesq} = CNAPa: {saida} é
melhor')
                    Mov = movesq
            elif s0 < 0.7:
                print(f'Movimento {movdir} = {s0}; Movimento {movesq}
= {s}')
                print(f'Movimento {movdir} está livre')
                Mov = movdir
            elif s < 0.7:
                print(f'Movimento {movdir} = {s0}; Movimento {movesq}
= {s}')
                print(f'Movimento {movesq} está livre')
                Mov = movesq

        elif s0 < 0.7 and s < 0.7:
            print(f'Movimento {movdir} e movimento {movesq} estão
livres')
            if saida0 < saida:
                print(f'Movimento {movdir} = CNAPa: {saida0} é
melhor')

                Mov = movesq

            else:
                print(f'Movimento {movesq} = CNAPa: {saida} é melhor')

                Mov = movesq

        elif s0 < 0.7:
            print(f'Movimento {movdir} está livre')
```

```
            Mov = movdir
        elif s < 0.7:
            print(f'Movimento {movesq} está livre')
            Mov = movesq

#    else:
#        print(Mov)


# Função responsável por criar um banco de dados do tamanho da
coordenada desejada e coloca valores automáticos para
# definição de obstáculos
def CriarBanco():
    global x
    global y
    global l
    global n
    global l1
    x = 0
    y = 0
    import pymysql
    conexao = pymysql.connect(host='localhost', user='root',
passwd='', database='amr5')
    cursor = conexao.cursor()

    # Primeiro quadrante e eixo Y positivo

    l1_2 = l1  # variável para ccntrole da criação da matriz do banco
de dados
    for quad1 in range(-1, l1, +1):
        for quad1 in range(0, l1_2, +1):
            y += 1
            aleatorio()
            com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
            valor = (x, y, n)
            cursor.execute(com_sql, valor)
            conexao.commit()
        x += 1
        y = 0
    x = 0
    y = 0

    # Segundo quadrante e eixo y negativo

    l1_2 = l1  # variável para ccntrole da criação da matriz do banco
de dados
    for quad2 in range(-1, l1, +1):
        for quad2 in range(0, l1_2, +1):
            y -= 1
            aleatorio()
            com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
            valor = (x, y, n)
            cursor.execute(com_sql, valor)
            conexao.commit()
        x += 1
        y = 0
    x = -1
    y = 0
```

```
    # Terceiro quadrante

    l1_2 = l1  # variável para ccntrole da criação da matriz do banco
de dados
    for quad3 in range(0, l1, +1):
        for quad3 in range(0, l1_2, +1):
            y -= 1
            aleatorio()
            com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
            valor = (x, y, n)
            cursor.execute(com_sql, valor)
            conexao.commit()
        x -= 1
        y = 0

    x = -1
    y = 0

    # Quarto quadrante

    l1_2 = l1  # variável para ccntrole da criação da matriz do banco
de dados
    for quad1 in range(0, l1, +1):
        for quad1 in range(0, l1_2, +1):
            y += 1
            aleatorio()
            com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
            valor = (x, y, n)
            cursor.execute(com_sql, valor)
            conexao.commit()
        x -= 1
        y = 0
    x = 0
    y = 0

    # Eixo X positivo
    l1  # variável para ccntrole da criação da matriz do banco de
dados
    for quad1 in range(-1, l1, +1):
        aleatorio()
        com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
        valor = (x, y, n)
        cursor.execute(com_sql, valor)
        conexao.commit()
        x += 1
    x = -1
    y = 0

    # Eixo X negativo e coordenada x0, y0
    l1  # variável para ccntrole da criação da matriz do banco de
dados
    for quad1 in range(0, l1, +1):
        aleatorio()
        com_sql = "INSERT INTO plano (x, y, valor) VALUES (%s, %s,
%s)"
        valor = (x, y, n)
        cursor.execute(com_sql, valor)
        conexao.commit()
```

```
            x -= 1
        x = 0
        y = 0

        print(cursor.rowcount, "inserida com sucesso")


# Função para buscar valores no banco de dados
def banco():
    global l
    global x
    global y
    global valores
#     travacoordenada()
    #     x = 0
    #     y = 0
    import pymysql

    conexao = pymysql.connect(host='localhost', user='root',
passwd='', database='amr5')

    cursor = conexao.cursor()
    valores = []  # list(range(0, l)) # criando lista que vai de 0 até
l
    for mov in range(0, l, +1):  # laço for começando em 0,
finalizando em l e com passo acrescentando 1
        if Mov == 1:
            Movimento1()
        elif Mov == 2:
            Movimento2()
        elif Mov == 3:
            Movimento3()
        elif Mov == 4:
            Movimento4()
        elif Mov == 5:
            Movimento5()
        elif Mov == 6:
            Movimento6()
        elif Mov == 7:
            Movimento7()
        elif Mov == 8:
            Movimento8()
        query = 'SELECT valor FROM plano WHERE x = %s and y = %s'  #
dizendo que vai pegar o valor x e y do bando de dados
        cursor.execute(query, (x, y,))  # executando a query do banco
de dados com as variaveis x e y
        a = cursor.fetchone()  # valor do banco de dados de forma
string colocado em a
        #print(x)
        #print(y)
        #        print(a)

        query = 'SELECT valor2 FROM plano WHERE x = %s and y = %s'  #
dizendo que vai pegar o valor x e y do bando de dados
        cursor.execute(query, (x, y,))  # executando a query do banco
de dados com as variaveis x e y
        b = cursor.fetchone()  # valor do banco de dados de forma
string colocado em b
        # print(x)
        # print(y)
        #        print(a)
```

```
        for w in a:  # transformando string do valor do bando de dados
em numero inteiro
            print(f' W = {w}')
            #pass

        for u in b:
            print(f' U = {u}')
            #pass

        if u == 1:
            valores += [u]

        else:

            valores += [w]  # acrescenta valor "w" buscado no banco de
dados ao final da lista valores


#         l = ll


# Função que gera valores aleatorios para definição de obstaculos
def aleatorio():
    global n
    import random
    #aleatorio = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
1.0]

    aleatorio = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

    #aleatorio = [0.7, 0.8, 0.9, 1.0]

    #aleatorio = [0.0]
    n = random.choice(aleatorio)
    print(n)



def fixador():
    global x
    global y
    import pymysql
    conexao = pymysql.connect(host='localhost', user='root',
passwd='', database='amr5')
    cursor = conexao.cursor()

    query = "UPDATE plano SET valor2  = '1' WHERE x = %s and y = %s"

    cursor.execute(query, (x, y,))
    conexao.commit()

    #query = "UPDATE plano SET valor  = '1' WHERE x = %s and y = %s"

    #cursor.execute(query, (x, y,))
    #conexao.commit()

def deletador():
    import pymysql
    conexao = pymysql.connect(host='localhost', user='root',
passwd='', database='amr5')
```

```
    cursor = conexao.cursor()

    #query = "UPDATE plano SET valor  = '0' WHERE valor2 = '1'"

    #cursor.execute(query)
    #conexao.commit()

    query = "UPDATE plano SET valor2  = '0'"

    cursor.execute(query)
    conexao.commit()


def backup():
    import pymysql
    conexao = pymysql.connect(host='localhost', user='root',
passwd='', database='amr5')
    cursor = conexao.cursor()

    query = "UPDATE plano SET valor = backup"

    cursor.execute(query)
    conexao.commit()


# Função da rede com a Célula Neural Artificial Paraconsistente
maximizadora com capacidade de gerar multicelulas
def CNAPmax():
    global valores
    global l2
    global s
    global inc1
    global c
    global vmax
    inc1 = c - 2  # variável de incremento do atributo da lista
valores

    print(valores)

    ue = (valores[c - 1] - valores[inc1] + 1) / 2
    if ue >= 0.5:
        s = valores[c - 1]
        l2 -= 2  # controle da variavel de análise de células, na
primeira análise já se usam duas
        inc1 -= 1
        for uuu in range(l2, 0, -1):
            ue = (s - valores[inc1] + 1) / 2
            if ue >= 0.5:
                print('S = {}'.format(s))
                inc1 -= 1
            else:
                s = valores[inc1]
                print('S = {}'.format(valores[inc1]))
                inc1 -= 1

    else:
        s = valores[inc1]
        l2 -= 2  # controle da variavel de análise de células, na
primeira análise já se usam duas
        inc1 -= 1
        for uuu in range(l2, 0, -1):
```

```
            ue = (s - valores[inc1] + 1) / 2
            if ue >= 0.5:
                print('S = {}'.format(s))
                inc1 -= 1
            else:
                s = valores[inc1]
                print('S = {}'.format(valores[inc1]))
                inc1 -= 1
    l2 = vmax
    print(f'lllllllllllllll2 = {l2}')


# Função da rede com a Célula Analitica

def CNAPa():
    global valores
    global l2a
    global saida
    global va

    ftct = 1
    ftc = 0

    inc1 = c - 2

    Vcve = (1 + ftc) / 2
    Vcfa = (1 - ftc) / 2
    Vcic = (1 + ftct) / 2
    Vcpa = (1 - ftct) / 2

    Ue = (valores[c - 1] - (1 - valores[inc1]) + 1) / 2
    Uctr = (valores[c - 1] + (1 - valores[inc1])) / 2

    if (Vcic > Uctr > Vcpa and ((Vcve <= Ue) or (Ue <= Vcfa))):
        saida = Ue
        print(f'saida CNAPa = {saida}')

        l2a -= 2  # controle da variavel de análise de células, na
primeira análise já se usam duas
        inc1 -= 1
        for uuu in range(l2a, 0, -1):
            Ue = (saida - (1 - valores[inc1]) + 1) / 2
            Uctr = (saida + (1 - valores[inc1])) / 2
            if (Vcic > Uctr > Vcpa and ((Vcve <= Ue) or (Ue <=
Vcfa))):
                saida = Ue
                print(f'saida CNAPa = {saida}')
                inc1 -= 1
            else:
                saida = 0.5
                print(f'saida CNAPa = {saida}')
                inc1 -= 1

    else:
        saida = 0.5
        l2a -= 2  # controle da variavel de análise de células, na
primeira análise já se usam duas
        inc1 -= 1
        print(f'saida CNAPa = {saida}')
        for uuu in range(l2a, 0, -1):
            Ue = (saida - (1 - valores[inc1]) + 1) / 2
```

```
            Uctr = (saida + (1 - valores[inc1])) / 2
            if (Vcic > Uctr > Vcpa and ((Vcve <= Ue) or (Ue <=
Vcfa))):
                saida = Ue
                print(f'saida CNAPa = {saida}')
                inc1 -= 1
            else:
                saida = 0.5
                print(f'saida CNAPa = {saida}')
                inc1 -= 1
    l2a = va



def travacoordenada():
    global x
    global y
    global xx
    global yy

    x = xx
    y = yy


#    x = 0
#    y = 0

############################################ PROGRAMA PRINCIPAL
#############################################################

l = 3  # Automatização da variável de análise de células (numero de
entradas, numero de celulas = entradas - 1)

#l1 = int(input(f'Digite o tamanho desejado do plano cartesiano para
criaçao do banco de dados e geração automática de valores para testar
obstáculos: '))
#CriarBanco()

# l = int(input(f'Digite o número de células desejado para analise
(Deve ser no mínimo3): '))

# travacoordenada()
global x
global y
deletador()
xx = int(input(f'Digite o valor de Xorigem): '))
yy = int(input(f'Digite o valor de Yorigem): '))
xd = int(input(f'Digite o valor de Xdestino): '))
yd = int(input(f'Digite o valor de Ydestino): '))

x = xd
y = yd
xm = 0
ym = 0
lista = []




if l >= 3:
```

```
    l2 = l  # atribuição de variavel para uso da função CNAPmax que
deve ser baseado no número de céluas a serem analisadas
    l2a = l2
    l3 = l  # atribuição de variavel para manter o valor escolhido de
análise de células sem ser alterado por laços "for"
    c = l # variavel de controle para rede maximizadora executar os
calculos ao contrario
    c2 = l # # variavel de controle para rede de analise executar os
calculos ao contrario
    vmax = l
    va = l
    Mov2 = 0  # variável para auxiliar função analise1()

    while (yd != yy and xd != xx) and (xd > xx or xd < xx or yd > yy
or yd < yy) or (xd == xx and yd != yy) or (yd == yy and xd != xx):
        if yd >= yy:
            if xd == xx:
                if yd == yy:
                    print('FIM')
                else:
                    fixador()
                    Mov = 1
                    print(f'Movimento {Mov}')
                    mov_contrario()
                    print(f'Esta em X{xd}, Y{yd}')



                    banco()
                    CNAPmax()
                    analise1()

                    print(f'Esta em X{xd}, Y{yd}')
                    mov_contrario()


                    planejamento_de_movimentos()
                    lista += [Mov]  # lista com sequência dos
movimentos
                    x = xd
                    y = yd
            else:
                if yd == yy:
                    if xd > xx:
                        fixador()
                        Mov = 3
                        print(f'Movimento {Mov}')
                        mov_contrario()
                        print(f'Esta em X{xd}, Y{yd}')


                        banco()
                        CNAPmax()
                        analise1()

                        print(f'Esta em X{xd}, Y{yd}')
                        mov_contrario()


                        planejamento_de_movimentos()
```

```
                        lista += [Mov]  # lista com sequência dos
movimentos
                        x = xd
                        y = yd
                    else:
                        fixador()
                        Mov = 7
                        print(f'Movimento {Mov}')
                        mov_contrario()
                        print(f'Esta em X{xd}, Y{yd}')


                        banco()
                        CNAPmax()
                        analise1()

                        print(f'Esta em X{xd}, Y{yd}')
                        mov_contrario()


                        planejamento_de_movimentos()
                        lista += [Mov]  # lista com sequência dos
movimentos
                        x = xd
                        y = yd
                else:
                    if xd > xx:
                        Mov = 2
                        fixador()
                        print(f'Movimento {Mov}')
                        mov_contrario()
                        print(f'Esta em X{xd}, Y{yd}')


                        banco()
                        CNAPmax()
                        analise1()

                        print(f'Esta em X{xd}, Y{yd}')
                        mov_contrario()


                        planejamento_de_movimentos()
                        lista += [Mov]  # lista com sequência dos
movimentos
                        x = xd
                        y = yd
                    else:
                        fixador()
                        Mov = 8
                        print(f'Movimento {Mov}')
                        mov_contrario()
                        print(f'Esta em X{xd}, Y{yd}')

                        banco()
                        CNAPmax()
                        analise1()

                        print(f'Esta em X{xd}, Y{yd}')
                        mov_contrario()
```

```
                        planejamento_de_movimentos()
                        lista += [Mov]  # lista com sequência dos
movimentos
                        x = xd
                        y = yd
        else:
            if xd == xx:
                fixador()
                Mov = 5
                print(f'Movimento {Mov}')
                mov_contrario()
                print(f'Esta em X{xd}, Y{yd}')


                banco()
                CNAPmax()
                analise1()

                print(f'Esta em X{xd}, Y{yd}')
                mov_contrario()


                planejamento_de_movimentos()
                lista += [Mov]  # lista com sequência dos movimentos
                x = xd
                y = yd
            else:
                if xd < xx:
                    fixador()
                    Mov = 6
                    print(f'Movimento {Mov}')
                    mov_contrario()
                    print(f'Esta em X{xd}, Y{yd}')


                    banco()
                    CNAPmax()
                    print(f'Esta em X{xd}, Y{yd}')
                    analise1()

                    mov_contrario()


                    planejamento_de_movimentos()
                    lista += [Mov]  # lista com sequência dos
movimentos
                    x = xd
                    y = yd
                else:
                    fixador()
                    Mov = 4
                    print(f'Movimento {Mov}')
                    mov_contrario()
                    print(f'Esta em X{xd}, Y{yd}')




                    banco()
                    CNAPmax()
```

```
                    print(f'Esta em X{xd}, Y{yd}')
                    analise1()

                    mov_contrario()


                    planejamento_de_movimentos()
                    lista += [Mov]  # lista com sequência dos
movimentos
                    x = xd
                    y = yd

else:
    print('Número de análise de células deve ser no mínimo 3')

deletador()

lista.reverse()
print(lista)
```