UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Lucas Ronco

Aplicação de algoritmos evolutivos em rotas de Drones com obstáculos

São Carlos
2019

Lucas Ronco

Aplicação de algoritmos evolutivos em rotas de Drones com obstáculos

Monografia apresentada ao Curso de Engenharia Elétrica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Eduardo do Valle Simões

São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Ronco, Lucas
R769a Aplicação de algoritmos evolutivos em rotas de Drones com obstáculos / Lucas Ronco; orientador Eduardo do Valle Simões. São Carlos, 2019.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. Algoritmo evolutivo. 2. Drone. 3. Obstáculo. 4. Rota. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Lucas Ronco

Título: "Aplicação de algoritmos evolutivos em rotas de Drones com obstáculos"

Trabalho de Conclusão de Curso defendido e aprovado em 07/06/2019,

com NOTA 5,7 (cinco, sete), pela Comissão Julgadora:

Prof. Dr. Eduardo do Valle Simões - Orientador - SSC/ICMC/USP

Mestre Pedro Henrique Aquino Barra - Doutorando - SEL/EESC/USP

Mestre Valdemar Abrão Pedro Anastácio Devesse - Doutorando - ICMC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

A minha noiva e aos meus pais que sempre me incentivaram nesse percurso.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por sempre estar ao meu lado em todas as batalhas. Agradeço também aos meus pais, Wagner e Cláudia, que sempre me incentivaram a buscar o melhor de mim. À Aline Takata, minha noiva e companheira, pelo apoio, principalmente nos momentos difíceis: obrigado por estar ao meu lado. Por fim, ao Prof. Dr. Eduardo Simões, pelo acolhimento e disposição em me orientar com entusiasmo neste trabalho.

“Por que d’Ele e por meio d’Ele, e para Ele, são todas as coisas. Glória, pois, a Ele eternamente. Amém”.

(Romanos, 11:36)

RESUMO

RONCO, L. **Aplicação de algoritmos evolutivos em rotas de Drones com obstáculos.** 2019. 72p. Monografia – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

O crescente desenvolvimento dos Drones tem aumentado suas aplicações, como enviá-los a locais perigosos para o ser humano ou de difícil acesso. Tarefas como essas tornam ainda maiores a necessidade de se produzir um sistema de alta qualidade, com a menor taxa de erro possível, aumentando então a necessidade de pesquisas por trás da eficiência no cumprimento dessas atividades. Este trabalho apresenta o estudo de um algoritmo evolutivo para traçar rotas adequadas a um Drone dada a posição de partida e o objetivo, desviando de obstáculos. Foi utilizado como base para modificação um sistema pré-implementado, que simula a aplicação de algoritmos evolutivos ao funcionamento de um braço robótico. O algoritmo foi implementado utilizando a mesma linguagem do original: C, e foi utilizada a biblioteca gráfica OpenGL para visualização dos resultados de uma forma didática. Os resultados demonstraram as tentativas de desviar dos obstáculos, e assim, obter um caminho ideal até o objetivo, que fosse o mais curto e não colidisse com algum dos obstáculos pré-determinados.

Palavras-chave: Algoritmo evolutivo. Drone. Obstáculo. Rota.

ABSTRACT

RONCO, L. **Application of evolutionary algorithms on drone routes with obstacles.** 2019. 72p. Monografia – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

The increasing development of Drones has expanded its possible applications, like sending them to dangerous or difficult to access places. Tasks such as these make it even more necessary to produce a high-quality system with the lowest error rate possible, increasing the need for research behind efficiency in accomplishing these activities. This work presents the study of an evolutionary algorithm to draw suitable drone routes given the starting position and objective, avoiding obstacles. An existent system that simulates the application of Genetic Algorithms of a robotic arm was used as a basis for improvement. The algorithm was implemented using the same language as the original: C, and the OpenGL grafic library to show the results in a didactic way. The results show the comparison of different attempts to avoid obstacles, and thus, to obtain an ideal path to the goal, being that the shortest path that did not collide with any of the pre-determined obstacles

Keywords: Evolutionary algorithm. Drone. Obstacle. Route.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação e Relevância do Trabalho

A Era da informação, também conhecida como Era da Tecnologia se deu após a era Industrial (após a década de 1980) em que surgiram inovações como o microprocessador, redes de computadores, fibra ótica e computadores pessoais. Essa foi fomentada pelos avanços dados durante a Segunda Guerra Mundial e Guerra Fria, em que diversas tecnologias foram criadas para utilizar-se contra o inimigo. Uma dessas tecnologias consiste em Veículos Aéreos Não Tripulados (VANTs), mais conhecidos como Drones. Sua criação foi vista inicialmente como uma inovação não muito confiável, gerando discussões e questionamentos sobre os riscos envolvidos em sua operação sobre regiões povoadas. A segurança de voo é definida pela Organização de Aviação Civil Internacional (ICAO) como "estado em que a possibilidade de danos a pessoas ou propriedades é reduzida a um nível aceitável, através de um processo contínuo de identificação de perigos e gestão de riscos de segurança" (ICAO, 2013).

Criados para fins militares, os Drones possuem dois propósitos típicos nesse meio:

- Vigilância de Combate
- Reconhecimento Tático

No primeiro, um piloto utiliza um controle de rádio para pilotar o VANT em diferentes pontos de vista para escanear e marcar posições inimigas. Já no segundo, um mini-drone voa em modo piloto automático para alvos pré-determinados, a fim de retornar com dados da área desejada, como por exemplo: fotos do terreno. Segundo (MATTEI et al., 2013; FONSECA; MATTEI; CUNHA, 2013; FIGUEIRA et al., 2013), com a ausência dos pilotos na aeronave, faz-se necessário estudar novas formas de garantir segurança, que é o fator mais importante na integração dos VANTs ao espaço aéreo.

Com a melhora da tecnologia dos Drones nas áreas militares, esses poderiam passar a ser usados em setores privados. A partir de 2006, agências governamentais passaram a utilizá-los para vigilância de fronteiras e incêndios florestais, enquanto corporações começaram a usá-los para inspecionar dutos e pulverizar pesticidas em fazendas. Nesse mesmo ano, a AFA (Administração Federal de Aviação dos Estados Unidos) passou a determinar medidas

regulatórias apropriadas e emitir licenças para utilização de VANTs, fato que engatou o desenvolvimento de Drones para novas aplicações (muitas delas de uso cotidiano) e surgimento de desafios da atualidade, que tem por finalidade aumentar cada vez mais a qualidade da realização de suas tarefas.

Figura 1 – Exemplos de Aplicações de Drones na Atualidade

Fonte: mydronelab.com/blog/drone-uses.html

A Figura 1 mostra alguns exemplos de utilização de Drones de forma benéfica à sociedade como uso na agricultura, entrega de pacotes e auxílio no controle de incêndios. Dado o último exemplo, observa-se que as aplicações de VANTs se tornam cada vez mais úteis quando se referem a utilizar da vantagem de enviá-los a locais de difícil acesso ou de perigo eminente ao ser humano. Outro bom exemplo se dá na detecção de vítimas em áreas de desastres. Tarefas como essa tornam ainda maiores a necessidade de se produzir um sistema de alta qualidade, com a menor taxa de erro possível, aumentando então a necessidade de pesquisas por trás da eficiência no cumprimento dessas tarefas.

O presente trabalho se relaciona com a dissertação do ex-aluno de mestrado Jesimar Arantes (ARANTES, 2016) do Instituto de Ciências Matemáticas e de Computação – ICMC-USP, que apresenta pesquisas sobre o desenvolvimento de algoritmos que efetuem o planejamento de rotas na ocorrência de situações críticas para obter maior segurança aérea aos VANTs. Este trabalho se relaciona também com a tese de doutorado do ex-aluno Márcio Arantes (ARANTES, 2017) do mesmo instituto, que traz uma abordagem para o planejamento de missões com VANTs.

1.2 Objetivo

O objetivo desse projeto é explorar os conceitos de Algoritmos Evolutivos e aplicá-los no âmbito de roteamento de Drones. Tal exploração será feita e testada com base em códigos testes - utilizando a linguagem C e a biblioteca OpenGL para visualização dos dados e resultados - onde serão inseridos novos elementos, como obstáculos, a fim de simular um cenário em 2 dimensões onde um Drone necessite ir de um ponto a outro escolhendo a melhor rota para tal tarefa. Caso o algoritmo se mostre viável para a solução de encontrar uma rota ideal para o Drone desviando dos obstáculos, o código será considerado para ser expandido para ser simulado em um cenário que represente melhor a realidade de um vôo de um VANT.

2 FUNDAMENTOS TEÓRICOS

Um algoritmo evolutivo é uma heurística de busca inspirada na teoria da evolução de Charles Darwin em A Origem das Espécies (DARWIN, 1859). É uma técnica utilizada para achar soluções aproximadas em problemas que envolvem busca em um espaço muito grande de soluções. Este algoritmo reflete o processo de seleção natural, onde os indivíduos mais aptos são selecionados para reprodução, a fim de produzir descendentes da próxima geração. Com essa abordagem, é possível encontrar uma solução de boa qualidade ou ótima para o problema de traçar rotas de Drones com a existência de obstáculos.

2.1 Algoritmos Evolutivos

Segundo GOLDBERG (1989), um algoritmo evolutivo difere dos métodos tradicionais de busca, pois a partir de um conjunto de parâmetros, opera sobre uma população de indivíduos simultaneamente. Suas regras de escolha são baseadas em Probabilidade e não Determinismo. O processo de seleção natural se inicia com a seleção de indivíduos, dada uma população, que sejam mais aptos para sobreviver, em um determinado ambiente. Esses indivíduos selecionados vão se relacionar entre si e produzir descendentes que herdam características dos pais e irão compor a próxima geração. Se forem escolhidas boas características dos geradores, seus filhos poderão se tornar mais aptos e assim, ter maiores chances de sobreviver. O processo de reprodução itera até que seja atingido um determinado objetivo e a geração de indivíduos mais aptos seja encontrada.

Apresentados por HOLLAND (1975) os algoritmos genéticos são uma área de pesquisa dentro da Computação Evolutiva podendo ser considerados seis fatores ou fases para iteração do algoritmo que serão abordadas a seguir:

- População Inicial
- Função de Aptidão
- Seleção
- Cruzamento
- Mutação
- Predação

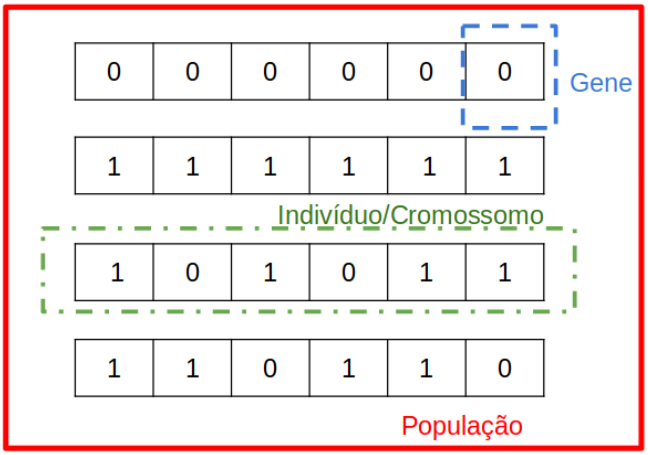
2.1.1 População Inicial

O processo se inicia com um determinado número de indivíduos, cada qual representando uma solução do problema. Esse conjunto de indivíduos é chamado de População Inicial e é definido aleatoriamente dentro do espaço de busca.

Cada indivíduo é caracterizado por um conjunto de parâmetros, nomeados como genes, conforme indicado na Figura 2. Usualmente, esses genes são representados como Bits, mas podem assumir outras formas como números inteiros ou *strings*.

O conjunto desses atributos gera então o Indivíduo em si, podendo ser nomeado em Algoritmos Evolutivos como Cromossomos.

Figura 2 – Representação de Genes, Cromossomos e População



Fonte: Elaborado pelo autor

2.1.2 Função de Aptidão e Seleção

A fim de seguir os passos de Darwin, deve haver um meio de selecionar quais indivíduos passam seus genes para frente, ou seja, quais se demonstram mais aptos a sobreviver de acordo com o ambiente oferecido.

A cada geração, a seleção é feita por uma Função de Aptidão, ou Função *Fitness*, que dará uma determinada pontuação a cada indivíduo, determinando o quanto esse é apto a sobreviver em comparação com os outros, sendo possível, portanto, encontrar o melhor de cada geração. Essa função deve ser bem definida, para que haja o bom funcionamento do algoritmo evolutivo.

2.1.3 Cruzamento

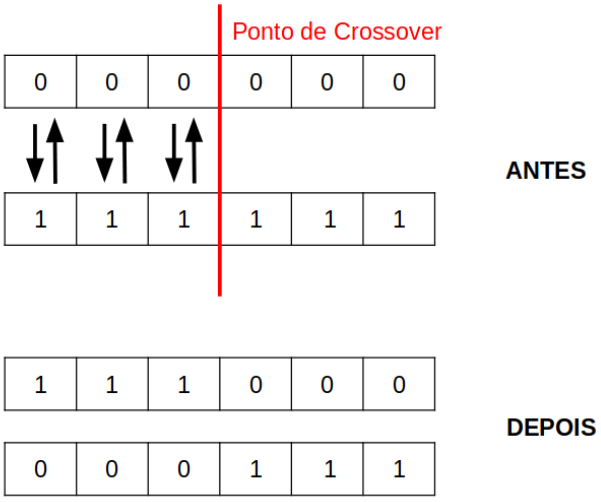
O Cruzamento, ou *Crossover*, é a parte significante do Algoritmo Evolutivo, pois ele é responsável pela reprodução, ou seja, criação de novos indivíduos.

Para cada indivíduo novo a ser criado, são escolhidos seus pais, para que suas características sejam transmitidas a ele. Uma vez que o intuito é criar indivíduos com maiores chances de sobrevivência, a escolha dos pais pode ser feita em uma combinação do mais apto daquela geração com os demais.

Assim como a Função de Aptidão, o *Crossover* não possui uma forma padrão para ser feito. Sendo assim, um exemplo típico de sua realização seria a troca de metade dos bits de cada indivíduo, gerando, portanto, novos indivíduos combinados. A Figura 3 demonstra um cruzamento de único ponto, porém existe também a uniforme, de dois ou n pontos.

Deve-se sempre lembrar que é de extrema importância guardar o indivíduo mais apto de cada geração e não modificá-lo até que se encontre alguém melhor.

Figura 3 – Cruzamento entre dois indivíduos

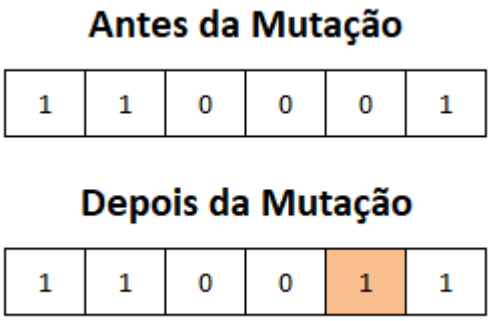


Fonte: Elaborado pelo autor

2.1.4 Mutação

A mutação em um algoritmo evolutivo é responsável por alterar aleatoriamente alguma característica ou gene de um novo indivíduo em dada geração. Ela não possui momento certo para ocorrer e pode ocorrer e um ou mais indivíduos. Na Figura 4, a mutação pode ser observada no penúltimo Gene de um indivíduo, onde sua característica era representada pelo Bit 0 e é modificada pelo Bit 1. Ela é de extrema importância, pois é responsável por manter a diversidade dentro da população, ou seja, aumenta o espaço de busca de soluções e previne qualquer convergência prematura, ou seja, que o sistema perca a capacidade de gerar diversidade.

Figura 4 – Processo de Mutação em um indivíduo



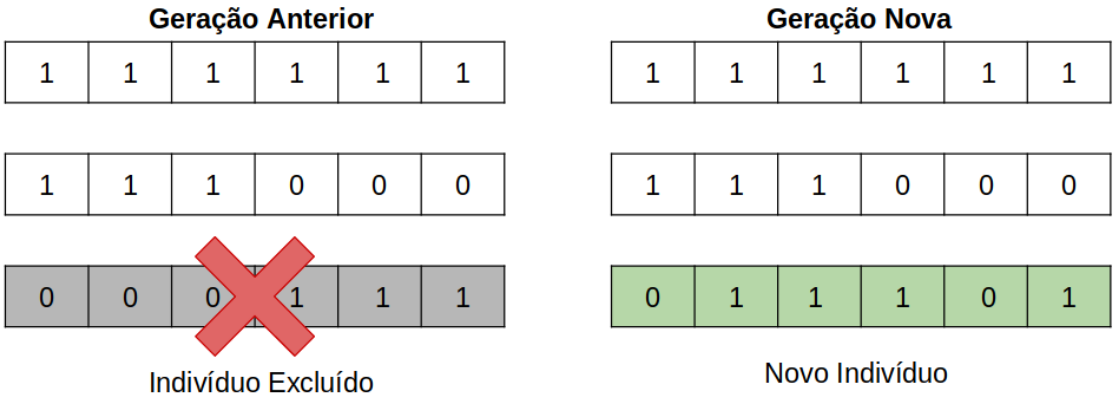
Fonte: Elaborado pelo autor

2.1.5 Predação

O processo de predação tem como objetivo remover completamente indivíduos de uma população e substituí-los por outros completamente novos, a fim de aumentar o espaço de busca de soluções, da forma como a mutação atua. A Figura 5 demonstra um exemplo de predação, onde o terceiro indivíduo é excluído e um novo é gerado em seu lugar.

Essa escolha para exclusão pode ser feita dado um determinado fator negativo atingido por tal indivíduo ou pode ser feita aleatoriamente. É de extrema importância que não se descarte o indivíduo mais apto, pois até aquele momento ele representa a melhor solução para o problema.

Figura 5 – Indivíduo sendo excluído de uma geração para a próxima



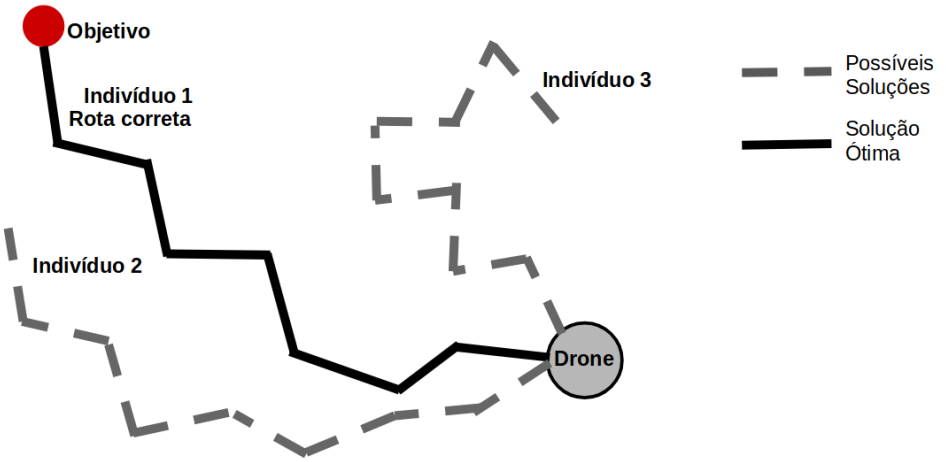
Fonte: Elaborado pelo autor

2.2 Roteamento de um VANT

Este trabalho considera o formato de rota de um Drone/VANT como uma sequência de direções tomadas (dado um número fixo de direções). As Figuras 6 e 7 ilustram o cenário com a ausência e presença de obstáculos, sendo o segundo o foco deste trabalho.

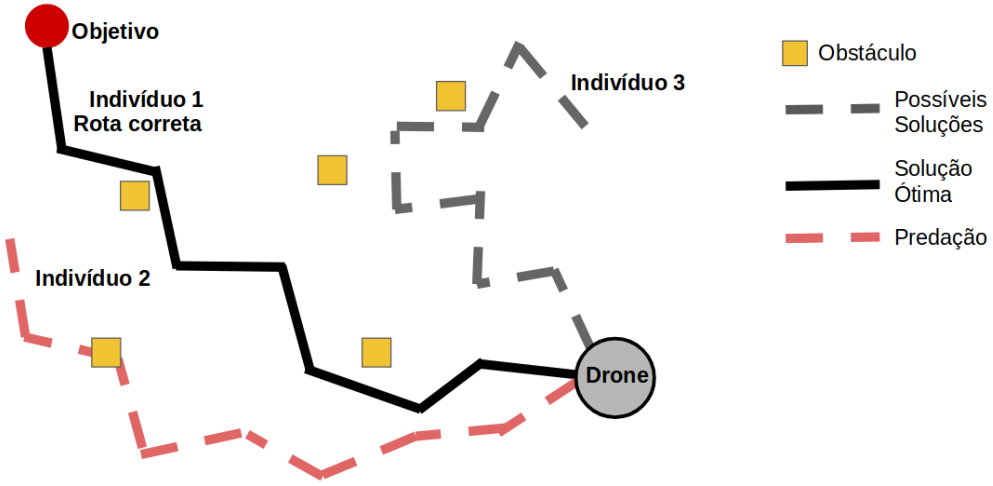
Em ambas, o indivíduo com melhor rota selecionada pelo Algoritmo Evolutivo é destacado. O objetivo de ligar o algoritmo evolutivo do braço robótico ao roteamento do VANT se dá pela simplificação do espaço amostral de soluções, visto que há movimentos limitados e em linha reta para que o objeto se desloque.

Figura 6 – Antes dos Obstáculos



Fonte: Elaborado pelo autor

Figura 7 – Depois dos Obstáculos

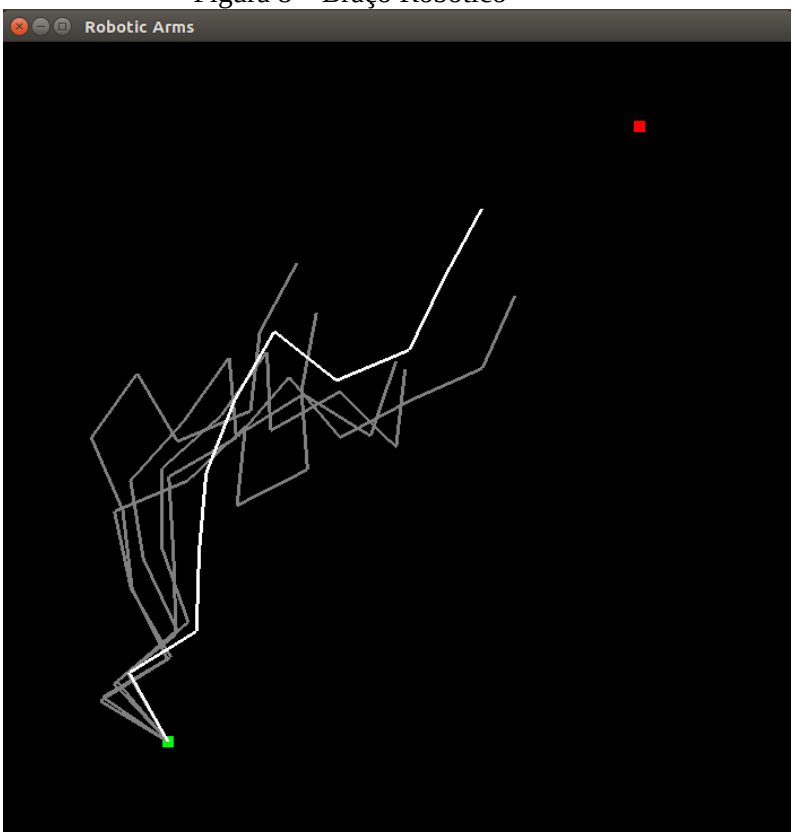


Fonte: Elaborado pelo autor

3 DESENVOLVIMENTO

Este capítulo descreve o desenvolvimento do trabalho e mostra como o algoritmo evolutivo pode auxiliar a resolver o problema de roteamento de um VANT, em uma versão 2D. Trata-se, portanto, de um caso especial em que o Drone permanece em uma altura fixa. Descrevem-se aqui as mudanças feitas em um projeto já existente de Algoritmos Evolutivos, que tinha por finalidade simular o funcionamento de um braço robótico, mostrado na Figura 8. As possíveis soluções são traçadas em cinza, enquanto a melhor solução é destacada em branco. O código criado pelo ex-aluno de graduação do ICMC Luiz Miyazaki está disponível em: <https://github.com/simoesusp/SSC0713_Sistemas-Evolutivos-Aplicados-a-Robotica/blob/master/ProjetosAlunos.zip>. Utilizou-se dessa mesma abordagem para tratar os braços como formatos de rotas e inserir obstáculos a serem desviados pelo Drone.

Figura 8 – Braço Robótico

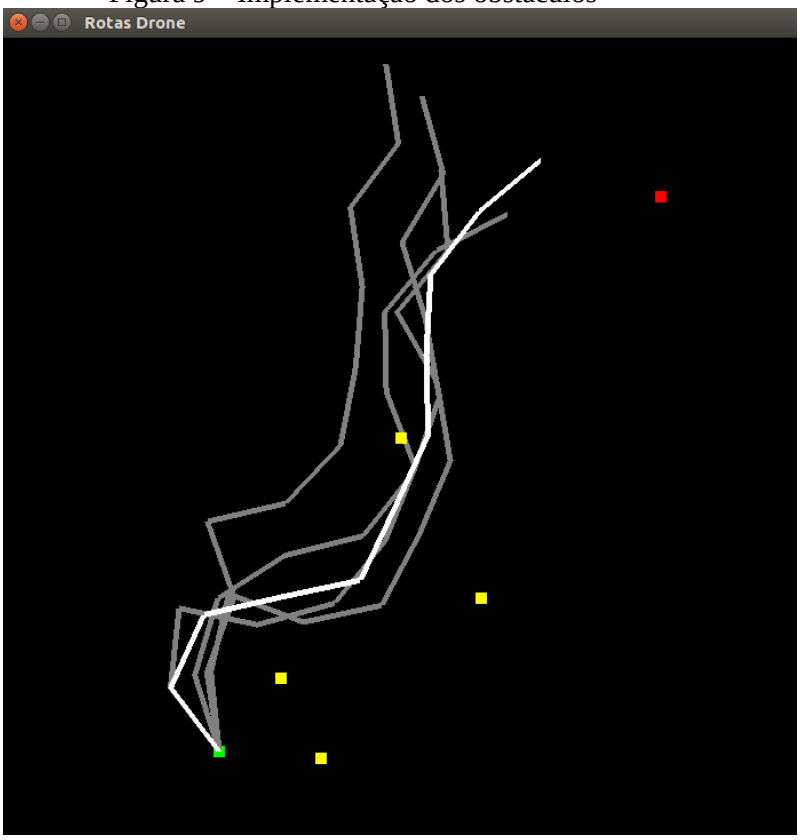


Fonte: Elaborado pelo autor

3.1 Algoritmo Evolutivo

O algoritmo evolutivo será aqui usado como um meio de se projetar o trajeto de um VANT, conforme mostrado na Figura 9, dado um ponto de partida da rota (em verde) e um ponto de chegada/objetivo (em vermelho), para posteriormente o inserir no cenário real através das soluções encontradas. Para isso, serão inseridos obstáculos (em amarelo) no código do Braço Robótico já apresentado. Neste capítulo, serão discutidos como cada uma das fases de um algoritmo evolutivo usual, apresentadas no capítulo 2, são aplicadas no problema deste trabalho.

Figura 9 – Implementação dos obstáculos

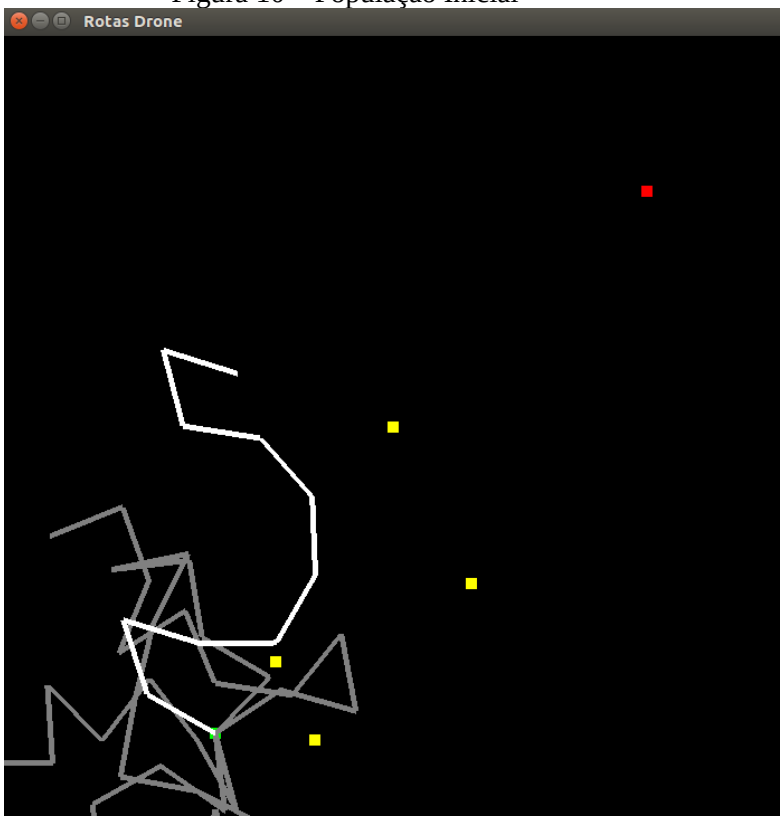


Fonte: Elaborado pelo autor

3.1.1 População Inicial

A população inicial é a representação de possíveis rotas a serem seguidas para que se atinja o objetivo. Como dito no início deste capítulo, essa é uma modificação de um problema em que se tem um braço robótico com N dobras e deseja-se levar seu extremo para uma determinada coordenada.

Figura 10 – População Inicial



Fonte: Elaborado pelo autor

Neste trabalho, as dobras feitas pelo braço robótico, ou seja, suas angulações serão tratadas como possíveis caminhos a serem seguidos pelo Drone. A população é gerada de forma aleatória, como observado na Figura 10, com um tamanho especificado previamente no código, nesse caso sendo uma população com cinco indivíduos iniciais, representados por ângulos como números reais e não no formato binário. Os parâmetros iniciais são explicados mais adiante neste capítulo, na seção de Implementação.

3.1.2 Função de Aptidão

A partir da população criada, é necessário definir uma função que pontue cada indivíduo, a fim de se decidir se irá ou não passar seus genes adiante e determinar qual indivíduo é o mais apto de cada geração.

Neste trabalho, será utilizada uma função aptidão, ou *Fitness*, que considera:

1. A distância do último ponto da rota até o objetivo (*goal*)
2. A soma das distâncias de cada ponto da rota até os obstáculos
3. Penalidade

$$Fitness(w) = \sqrt{(X - Xgoal)^2 + (Y - Ygoal)^2} + \frac{5000}{\sum Distancia\_Obstáculos} + P \quad (3.1)$$

em que :

w = representa uma rota ou indivíduo da população
X, Y = coordenadas dos pontos finais de cada rota
Distancia_Obstáculos = distância de cada ponto da rota para os obstáculos
P = função de penalidade de acordo com as proximidades dos obstáculos

O primeiro termo é utilizado, para que a distância entre o final da rota e o objetivo seja avaliada. Como se pretende obter um menor *Fitness* possível, o segundo e o terceiro termo são acrescentados para aumentar o valor de *Fitness* das rotas que passarem próximas a algum obstáculo. A função de penalidade adiciona uma alta punição caso a distancia até algum obstáculo for muito pequena, sendo definida pela seguinte condição:

- if (menor distância obstáculo < 5,0)
- P = 3000
- else
- P = 0

3.1.3 Seleção do mais Apto

Com a população inicial estabelecida e a função de aptidão, pode-se então julgar os indivíduos, a fim de escolher aqueles que vão fazer parte do processo de reprodução, ou seja, passar seus genes para a próxima geração.

A seleção é feita com a comparação entre o *fitness* do indivíduo mais apto até aquele momento, com cada um dos outros indivíduos.

```
best = 0
for ( i = 1 : tamPop )
    if ( fitness(i) < fitness(best))
        best = i
    end
end
```

A Figura 11 mostra esse processo acontecendo, ao trocar o melhor individuo de índice 4 da geração 59 pelo de índice 0 da geração 60, quando este recebe um *Fitness* menor que o melhor da geração anterior.

Figura 11 – Seleção do mais Apto

```
-----Número de Gerações até aqui: 59------
Individual: 0: Fitness: 3023.51
Individual: 1: Fitness: 3030.39
Individual: 2: Fitness: 3047.90
Individual: 3: Fitness: 46.37
Individual: 4: Fitness: 32.44
Best index: 4

-----Número de Gerações até aqui: 60------
Individual: 0: Fitness: 25.95
Individual: 1: Fitness: 3026.72
Individual: 2: Fitness: 3039.39
Individual: 3: Fitness: 42.92
Individual: 4: Fitness: 32.44
Best index: 0
```

Fonte: Elaborado pelo autor

3.1.4 Cruzamento

Assim que o indivíduo mais apto é selecionado, suas características devem ser passadas para os demais. Nesse trabalho será utilizada a seguinte função de *crossover*:

$$Angulo\ novo(i) = \frac{Angulo\ Antigo(i) + Angulo\ Melhor\ Indivíduo(i)}{2} \qquad (3.2)$$

Nessa função faz-se uma combinação do vetor de ângulos, que determinam as mudanças de direção em uma rota, do mais apto com cada um dos outros indivíduos. O cruzamento não altera nenhum valor dentre os que pertencem ao mais apto, até que em alguma geração o mesmo deixe de ocupar esse posto.

3.1.5 Mutação

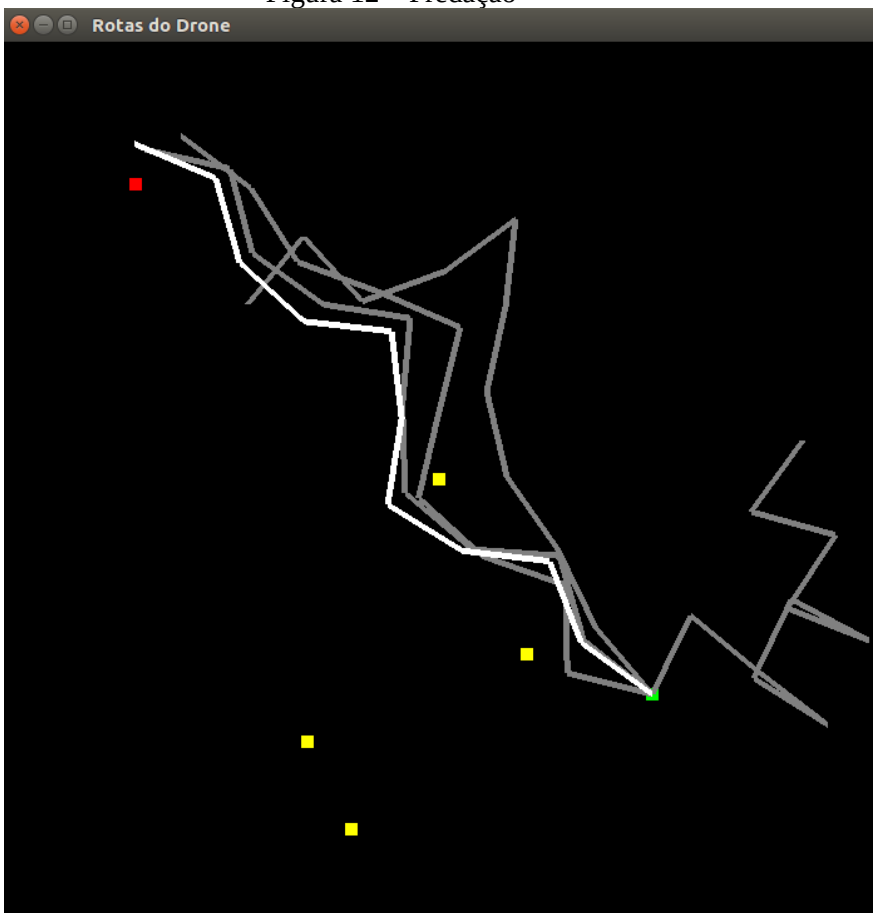
Neste trabalho, a mutação utilizada foi uma pequena variação nos ângulos entre cada segmento da rota. Para incrementar ou decrementar a taxa de mutação dos valores dos ângulos, é utilizada uma função que traz aleatoriamente um numero e se for par, incrementa, caso contrário, decrementa.

- if (rand() % 2)
- Angulo Novo = Angulo Antigo * (1 + taxa de mutação);
-
- else
- Angulo Novo = Angulo Antigo * (1 – taxa de mutação);

3.1.6 Predação

Para eliminar os indivíduos (rotas) que colidirem com os obstáculos, ou passarem muito próximos, é possível ativar a predação no código. Quando a predação está ativa, os indivíduos eliminados são substituídos por outros novos. A Figura 12 mostra um indivíduo que foi eliminado sendo criado novamente com valores aleatórios de ângulos, da mesma forma que a população inicial.

Figura 12 – Predação

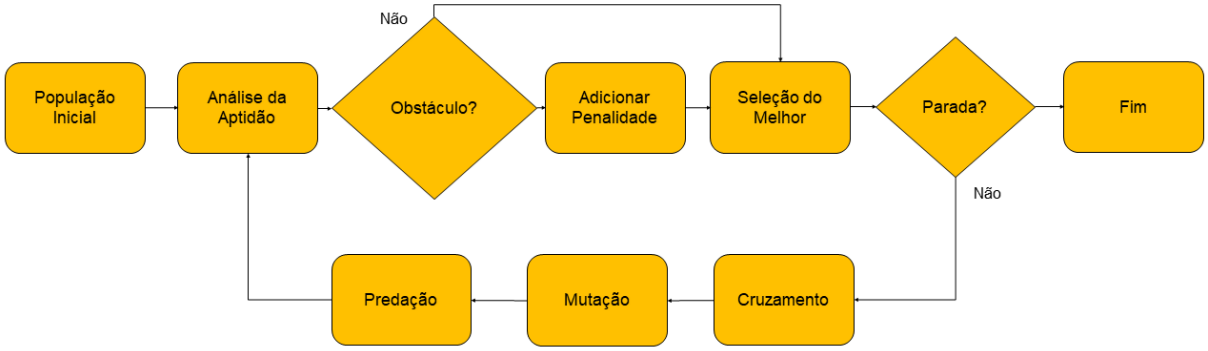


Fonte: Elaborado pelo autor

3.2 Implementação

O algoritmo foi implementado utilizando a linguagem C, com o auxílio da biblioteca OpenGL para visualização de sua execução. A ideia primária da implementação foi aproveitar o sistema já existente do braço robótico implementado com algoritmo evolutivo, apresentado no início desta seção, e inserir os obstáculos no mesmo, modificando os elementos da Função de Aptidão. O sistema implementado é representado abaixo pela Figura 13.

Figura 13 – Sistema de Roteamento Implementado com um Algoritmo Evolutivo



Fonte: Elaborado pelo autor

Para inicializar o sistema, foram utilizados os dados apresentados na Tabela 1. Foi escolhido um número pequeno de população para uma melhor observação visual dos resultados. A quantidade de braços e seus tamanhos correspondem à melhor visualização no tamanho da tela aberta pela biblioteca OpenGL.

Tabela 1 – Parâmetros de Entrada do Algoritmo Implementado

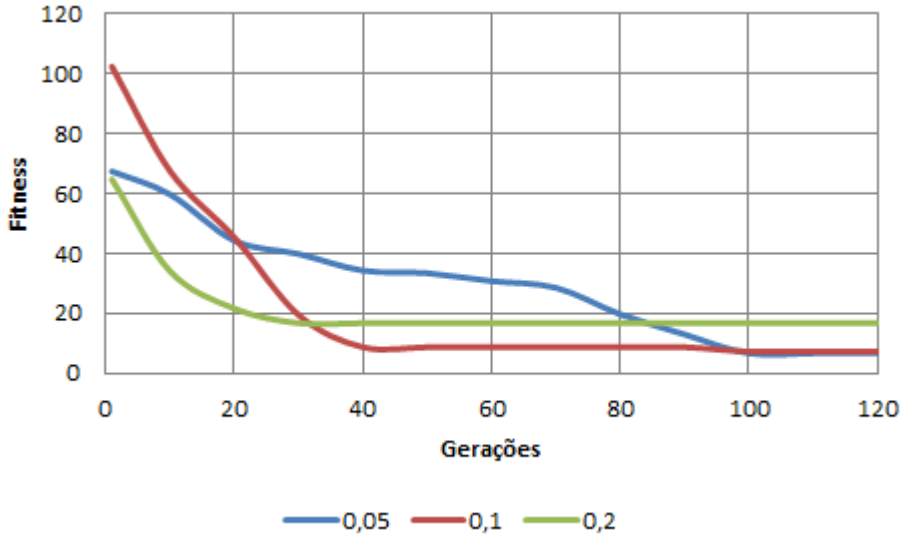
Variável	**Valor**
Tamanho da População	5
Número de Braços	10
Tamanho dos Braços	10
Taxa de mutação inicial	0,1
Gerações	Até que o usuário esteja satisfeito

Para a definição da taxa de mutação, foram realizados testes utilizando os valores 0,05, 0,1 e 0,2. A Tabela 2 mostra os resultados do *Fitness* para 120 gerações comparando os três valores. Utilizando uma taxa de mutação 0,05, observa-se que apenas na geração 100 o *Fitness* converge para um valor abaixo de 10,0, indicando que a variação nos ângulos é pequena demais para gerar uma boa diversidade para o resultado convergir rapidamente para a solução ótima. Com uma taxa de mutação de 0,2, pode-se observar que apesar do resultado convergir rapidamente com apenas 30 gerações, ele não consegue obter um *Fitness* tão bom quanto os outros valores, mostrando que a variação é alta demais, levando os novos indivíduos para longe do esperado. Sendo assim, a melhor opção foi utilizar a taxa de mutação de 0,1, onde em apenas 40 gerações já atingiu um *Fitness* menor que 10,0. Esses resultados são visivelmente melhor apresentados na Figura 14, que demonstra um gráfico de Fitness x Gerações para os três valores de taxa de mutação testados.

Tabela 2 – *Fitness* por Taxa de Mutação

Número de Gerações	Fitness (0,05)	Fitness (0,1)	Fitness (0,2)
1	67,43	102,42	64,74
10	59,77	67,62	34,13
20	44,33	45,43	21,67
30	39,86	19,53	16,77
40	34,31	8,72	16,77
50	33,42	8,72	16,77
60	30,81	8,72	16,77
70	28,58	8,72	16,77
80	19,73	8,72	16,77
90	12,99	8,72	16,77
100	6,67	7,26	16,77
110	6,67	7,26	16,77
120	6,67	7,26	16,77

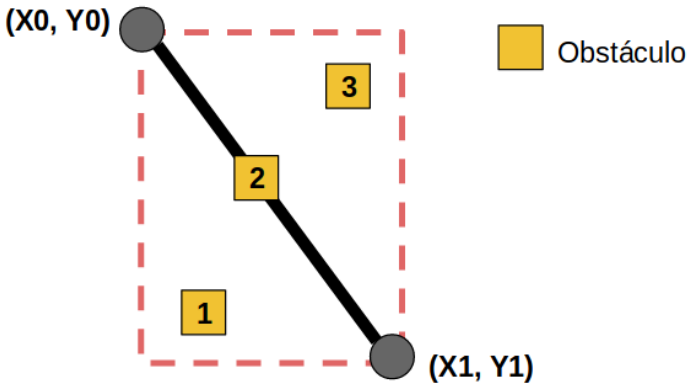
Figura 14 – Comparação entre as taxas de mutações



O maior problema em se detectar obstáculos cruzando uma rota, dentre vários segmentos pertencentes a essa, ocorre por tratar-se de pontos flutuantes. Portanto, detectar o ponto passando fielmente pelo segmento demonstrou-se uma garantia fraca para aumentar seu *fitness* e consequentemente determinar que um indivíduo seja menos apto.

Inicialmente tentou-se usar a abordagem de intervalos entre os pontos pertencentes a cada segmento. Assim como mostra a Figura 15, o obstáculo seria detectado se estivesse presente dentro do intervalo de coordenadas.

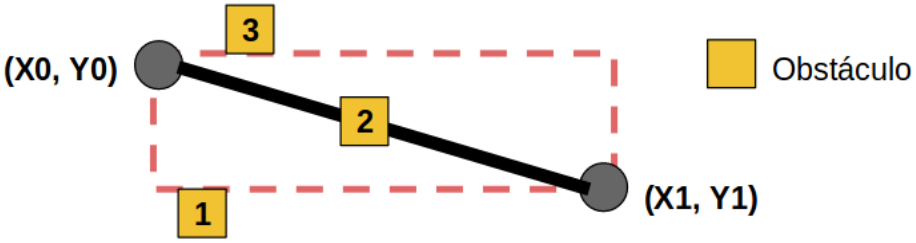
Figura 15 – Primeiro teste para detecção de colisões



Fonte: Elaborado pelo autor

Entretanto, esse método se demonstrou falho, uma vez que dependendo da inclinação do segmento, o espaço que contém os intervalos de X e Y se torna menor, e consequentemente com menores chances de incluir obstáculos que estejam muito próximo do segmento. Esse tipo de falha é ilustrado na Figura 16.

Figura 16 – Falha encontrada no teste de colisões



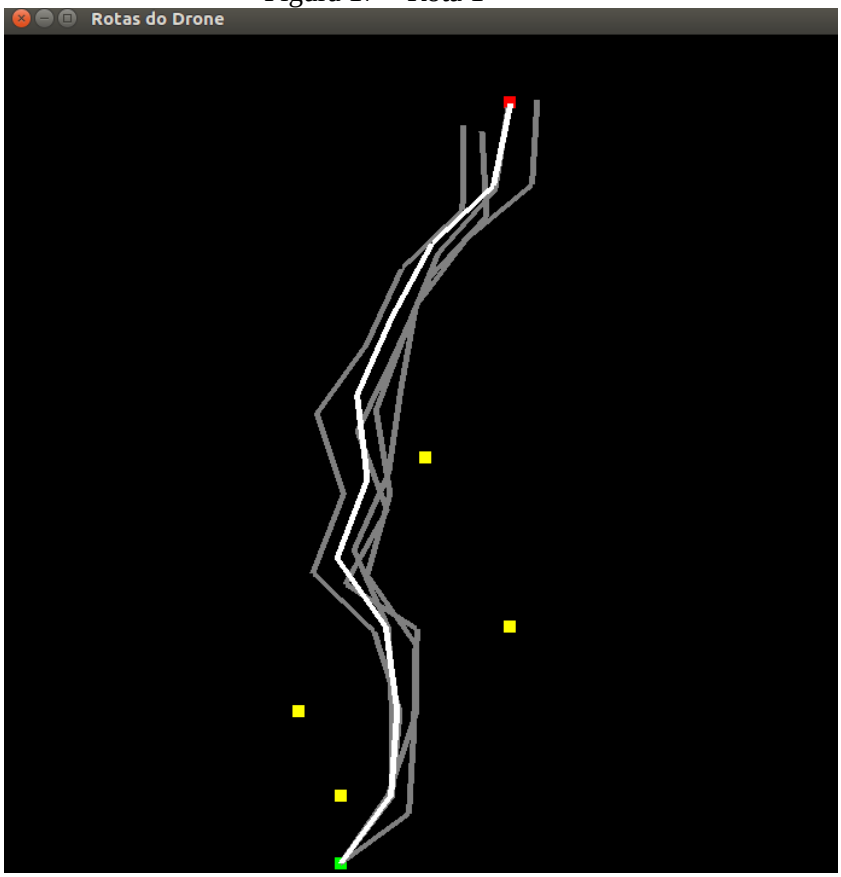
Fonte: Elaborado pelo autor

Decidiu-se então utilizar a abordagem de distância entre dois pontos, apresentada na seção 3.1.2 deste trabalho, determinando um limite para que essa distância seja considerada um perigo de colisão com o obstáculo.

3.3 Análise de Resultados

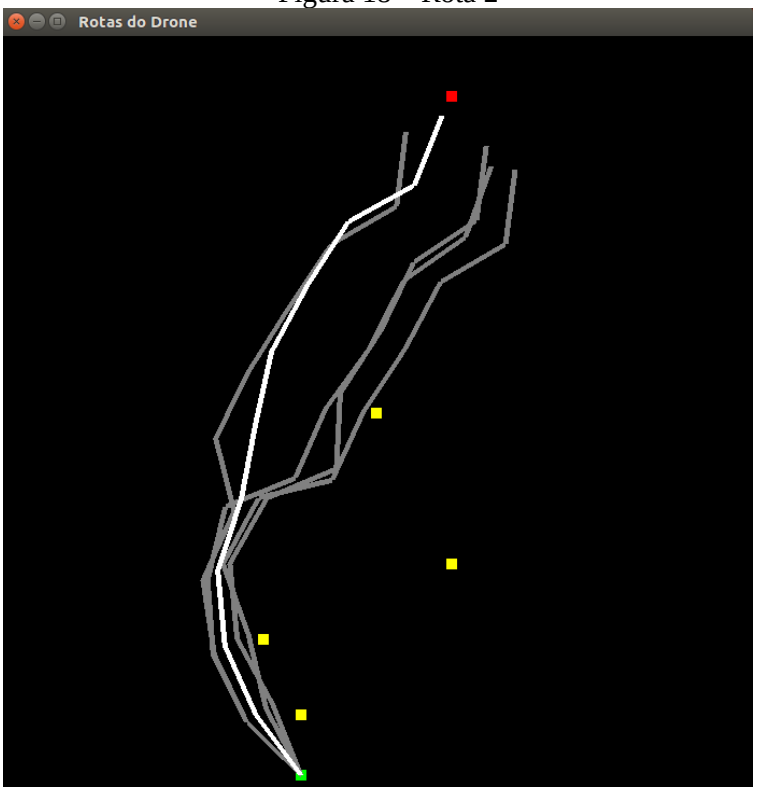
Os resultados de alguns dos testes feitos para fins de verificar o funcionamento do algoritmo demonstraram semelhanças quanto às direções traçadas para as rotas do Drone. As Figuras 17, 18 e 19 mostram três simulações onde foram traçadas rotas distintas para um mesmo cenário.

Figura 17 – Rota 1



Fonte: Elaborado pelo autor

Figura 18 – Rota 2



Fonte: Elaborado pelo autor

Figura 19 – Rota 3

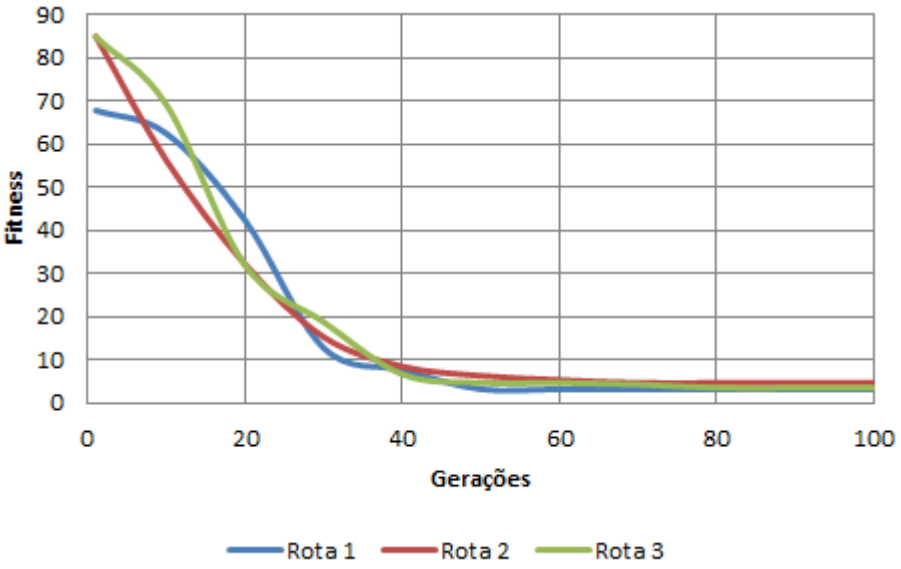
Fonte: Elaborado pelo autor

Os valores dos *Fitness* por geração das três simulações são apresentados na Tabela 3. Pode-se observar, através da Figura 20, que apesar das rotas serem distintas ambas convergem para um valor de *Fitness* abaixo de 5,0, mostrando que a eficácia do algoritmo é satisfatória para encontrar a melhor rota do Drone até um objetivo desviando dos obstáculos.

Tabela 3 – Fitness por rota

Número de Gerações	Rota 1	Rota 2	Rota 3
1	67,84	85,04	84,94
10	62,50	56,22	69,17
20	42,36	32,19	31,97
30	12,80	15,32	18,80
40	7,66	8,50	6,69
50	3,23	6,34	4,72
60	3,23	5,34	4,72
70	3,23	4,68	4,33
80	3,23	4,68	3,51
90	3,23	4,68	3,51
100	3,23	4,68	3,51

Figura 20 – *Fitness* por rota



Observa-se também que todas as rotas convergiram rapidamente para um valor de *Fitness* abaixo de 10,0 com menos de 40 gerações, provando ser uma ótima solução para encontrar rotas para os Drones dada uma emergência onde se tenha pouco tempo para planejar sua rota.

4 CONSIDERAÇÕES FINAIS

4.1 Conclusão

Esse trabalho consistiu em transformar a solução de um braço robótico, que busca aproximar sua extremidade de um ponto declarado como objetivo, através de um algoritmo evolutivo. Essa modificação foi feita assemelhando a estrutura do braço robótico com uma rota de um VANT, indo de uma origem ao seu ponto objetivo, sendo cada uma de suas "dobras" uma angulação que corresponde à mudança de direção na rota do Drone. O projeto com algoritmos permitiu a simulação sem que houvesse qualquer tipo de problema relacionado à segurança.

Adicionou-se obstáculos a fim de dificultar a rota e observar os resultados do algoritmo evolutivo em um cenário mais próximo da realidade. Os resultados demonstraram a comparação entre os tipos de abordagem para desviar dos obstáculos, e assim, obter um caminho ideal até o objetivo, que fosse o mais curto e não colidisse com algum dos obstáculos pré-determinados. A análise do *Fitness* mostrou que foi possível encontrar uma solução ótima com a utilização do algoritmo com poucas iterações.

Conclui-se, portanto, que a boa escolha de funções e parâmetros em um algoritmo evolutivo é essencial para o seu bom funcionamento. O algoritmo apresentado demonstrou-se didático, uma vez que o espaço 2D é muito limitado para, de fato, definir o roteamento de um VANT, sendo necessário expandi-lo para simular um cenário real.

Todos os arquivos deste trabalho estarão disponíveis em: < https://github.com/ simoesusp/Hexacoptero/tree/master/Lucas>.

4.2 Estudos Futuros

O algoritmo apresentado nesse trabalho se limita a um ambiente com duas dimensões, com altura fixa e obstáculos similares, ou seja, sua aplicação diária é minimamente válida quando se é necessário considerar todos os parâmetros que envolvem o vôo de um VANT. Portando um próximo passo para incremento do algoritmo aqui tratado seria transformar o ambiente para três dimensões, para se tornar mais similar a um cenário real. O código do braço robótico em 3D análogo ao utilizado nos capítulos anteriores, porém, sem interface gráfica, é apresentado nos anexos desse trabalho.

REFERÊNCIAS

ANDRE, D. **Evolution of map making: learning, planning, and memory using genetic programming.** Proceedings of the First IEEE Conference on Evolutionary Computation. Vol I. IEEE Press, 1994.

ARANTES, J. S.. **Planejamento de rota para VANTs em caso de situação crítica: uma abordagem baseada em segurança.** 2016.

ARANTES, M. S**. Hybrid qualitative state plan problem e o planejamento de missão com VANTs.** 2017.

ARANTES, M. S.; ARANTES, J. S; TOLEDO, C. F. M.; WILLIAMS, B. C. A hybrid multi-population genetic algorithm for uav path planning. In: **Genetic and Evolutionary Computation Conference.** [S.l.: s.n.], 2016.

BAKER, J. E. **An analysis of the the effects of selection in genetic algorithms.** PhD thesis, Graduate Schol of Vanderbilt University. Nashiville, Tennessee, 1989.

BANZHAF, W. et al. **Genetic Programming: na introduction.** [S.l.]: Morgan Kaufmann, 1998.

BARRETO, J. **Inteligência Artificial. No Limiar do Século XXI.** [S.l.: s.n.],1997.

BUSETTI, F. **Genetic Algorithms Overview.** 2001.

BUSINESS WEEK. **Texaco closes the loop.** New York: McGraw-Hill, 1959.

DEB, K.; KALYANMOY, D. **Multi-Objective Optimization Using Evolutionary Algorithms.** New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN 047187339X.

FIGUEIRA, N.; TRINDADE, O.; MATTEI, A. L. P.; NERIS, L. Mission oriented sensor arrays– an approach towards uas usability improvement in practical applications. In: **5th European Conference for Aeronautics and Space Sciences (EUCASS).** [S.l.: s.n.], 2013.

Goldberg, D. E. (1989), **Genetic Algorithms in Search, Optimization, and Machine Learning.** Addison-Wesley Publishing Company, INC. USA.

HOLLAND, J. H. **Adaptation in natural and artificial systems.** MIT Press Cambridge, MA, USA: [s.n.], 1975.

ICAO, I. C. A. O. **Doc 9859: Safety Management Manual (SMM)**. Edition 3. [S.l.], 2013. 251p.

MIYAZAKI, L. **RobotArms**, 2017. Disponível em: <https://github.com/simoesusp/SSC0713_Sistemas-Evolutivos-Aplicados-a-Robotica/blob/master/ProjetosAlunos.zip>.

RYAN, C.; COLLINS, J. J. & O'NEIL, M. **Grammatical evolution: evolving programs for an arbitrary language.** Proceedings of the First Workshop on Genetic Programming, 1998 Vol. LNCS 1391. Springer-Verlag, 1998.

SRINIVAS, N.; DEB, K. **Multiobjective optimization using nondominated sorting in genetic algorithms**. Technical reports, Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India, 1993.

STANZANI, E. **Ajuste evolutivo de um controlador de um voo de um quadricóptero.** 2018.

WILLIS, M. J.; HIDEN, H. G.; MARENBACH , P.; MCKAY, B. & MONTAGE, G. A. **Genetic programming: an introduction and survey of applications**. The Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications 1997 (GALESIA 97). IEEE Press, 1997

ZONKGER, D.; PUNCH, B. & RAND, B. **Lil-gp 1.01 User's manual** Genetic Algorithms Research and Applications Group (GARAGe), Department of Computer Science, Michigan State University, 1996.

Apêndices

APÊNDICE A – CÓDIGO PRINCIPAL

```
#include "rota_drone.h"
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int NUM_GERACOES = 0;
double origin[2] = {80.0, 20.0}; // define a origem
double goal[2] = {40.0, 30.0};   // define o objetivo
OBSTACULOS* obstaculos;
double mutation_rate = 0.1;
Angles *pop;
int best;

void Draw(void) {
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glPointSize(10.0f);
        glLineWidth(5.0f);
        glBegin(GL_POINTS);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(origin[0], origin[1]);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(goal[0], goal[1]);
        glEnd();

        // desenha todos os braços
        double x, y, prevx = origin[0], prevy = origin[1];
        int i, j;
```

```
for (j = 0; j < POP; j++) {
glBegin(GL_LINE_STRIP);
glColor3f(0.5f, 0.5f, 0.5f);
prevx = origin[0];
prevy = origin[1];
glVertex2f(prevx, prevy);

for (i = 0; i < ARMS; i++) {
x = prevx + cos(pop[j].val[i]) * ARM_SIZE;
y = prevy + sin(pop[j].val[i]) * ARM_SIZE;
prevx = x;
prevy = y;
glVertex2f(x, y);
}
glEnd();
}
for(i = 0; i < NUM_OBSTACULOS; i++){
glBegin(GL_POINTS);
glColor3f(1.0f, 1.0f, 0.0f);
glVertex2f(obstaculos[i].cord[0], obstaculos[i].cord[1]);
glEnd();
}

// desenha melhor rota
glBegin(GL_LINE_STRIP);
glColor3f(1.0f, 1.0f, 1.0f);
prevx = origin[0];
prevy = origin[1];
glVertex2f(prevx, prevy);
for (i = 0; i < ARMS; i++) {
x = prevx + cos(pop[best].val[i]) * ARM_SIZE;
y = prevy + sin(pop[best].val[i]) * ARM_SIZE;
prevx = x;
prevy = y;
```

```
            glVertex2f(x, y);
            }
            glEnd();
            glFlush();
}

void Mouse(int button, int state, int x, int y) {
            if (button == GLUT_LEFT_BUTTON) { // botão esquerdo
            goal[0] = x * 1.0 / 7;
            goal[1] = (700 - y) * 1.0 / 7;
            printf("Mouse x: %d, y: %d\n", x, y);
            glutPostRedisplay();
            }
            if (button == GLUT_RIGHT_BUTTON) { // botão direito
            origin[0] = x * 1.0 / 7;
            origin[1] = (700 - y) * 1.0 / 7;
            printf("Mouse x: %d, y: %d\n", x, y);
            glutPostRedisplay();
            }
}

void Keyboard(unsigned char key, int x, int y) {
            if (key == 13) {  // ENTER
            NUM_GERACOES++;
            crossover(pop, best, mutation_rate);
            best = selection(pop, origin, goal, obstaculos);
            show_result(pop, origin, goal, obstaculos);
            printf("-----Número de Gerações até aqui: %d------\n", NUM_GERACOES);
            glutPostRedisplay();
            }

            if (key == 'r') { // reset
            destroy(pop);
            printf("\n---------------------------------------------\n");
```

```
        printf("\nOld population destroyed and a new initialized!\n\n");
        pop = init_pop();
        best = selection(pop, origin, goal, obstaculos);
        show_result(pop, origin, goal, obstaculos);
        mutation_rate = 0.1;
        glutPostRedisplay();
        }

        if (key == 'p') { // aumenta taxa de mutação
        mutation_rate *= 10.0;
        }

        if (key == 'o') { // diminui taxa de mutação
        mutation_rate /= 10.0;
        }

        if (key == 27) { // ESC
        exit(0);
        }
}

// explica os comandos
void instruction() {
        printf("\nr: restart\n");
        printf("ENTER: new generation\n");
        printf("p: increase mutation rate\n");
        printf("o: decrease mutation rate\n");
        printf("LMB: change goal\n");
        printf("RMB: change origin\n");
        printf("ESC: exit\n\n");
        printf("------------------------\n\n");
}

int main(int argc, char *argv[]) {
```

```
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(700, 700);
        glutCreateWindow("Robotic Arms");
        glutDisplayFunc(Draw);
        glutMouseFunc(Mouse);
        glutKeyboardFunc(Keyboard);
        gluOrtho2D(0, 100, 0, 100);
        instruction();
        pop = init_pop();
        obstaculos = init_obstaculos();
        best = selection(pop, origin, goal, obstaculos);
        glutMainLoop();
        destroy(pop);

        return 0;
}
```

APÊNDICE B – FUNÇÕES DE ALGORITMO EVOLUTIVO

```
#include "rota_drone.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#define MAX 0x3f3f3f;
#define PREDACAO 0 //define predação ligada ou desligada

// passo 1
Angles *init_pop() {
        Angles *pop = (Angles *)malloc(POP * sizeof(Angles));
        if (pop == NULL) {
        printf("ERROR! pop is null\n");
        exit(0);
        }

        int i, j;
        int pi20000 = PI * 20000;
        srand(time(NULL));
        for (i = 0; i < POP; i++) {
        //cada "indivíduo" (rota) possui um número de ARMS e ARMS - 1 dobras
        pop[i].val = (double *)malloc(ARMS * sizeof(double));
        for (int j = 0; j < ARMS; j++) {
        pop[i].val[j] = rand() % pi20000 * 1.0 / 10000;
        }
        }

        return pop;
}
```

```
OBSTACULOS* init_obstaculos(){
        OBSTACULOS* obstaculos = (OBSTACULOS*) malloc(NUM_OBSTACULOS * sizeof(OBSTACULOS));
        int i, j;
        int coordenadas[NUM_OBSTACULOS * 2 + 1] = {40.0, 10.0, 50.0, 50.0, 60.0, 30.0, 35.0, 20.0, 50.0, 50.0};
        for(i = 0, j = 0; i < (NUM_OBSTACULOS * 2); i+=2, j++){
        obstaculos[j].cord[0] = coordenadas[i];
        obstaculos[j].cord[1] = coordenadas[i+1];
        }
        return obstaculos;
}

// passo 2 - Avalia, quanto menor melhor
double fitness(double origin[2], double goal[2], double *angles, OBSTACULOS *obstaculos) {

    double x, y, prevx = origin[0], prevy = origin[1], distance_obs = 0.0, closest_distance = MAX;
    double penalidade = 0.0, distance = 0.0;
    int pi20000 = PI * 20000;
    int i;
    for (i = 0; i < ARMS; i++){
       x = prevx + cos(angles[i])*ARM_SIZE;
       y = prevy + sin(angles[i])*ARM_SIZE;
       prevx = x;
       prevy = y;

       //calculando as distancias até os obstaculos
       for(int j = 0; j < NUM_OBSTACULOS; j++){
          distance = sqrt((x - obstaculos[j].cord[0])*(x - obstaculos[j].cord[0]) + (y - obstaculos[j].cord[1])*(y - obstaculos[j].cord[1]));
          if(distance < closest_distance){
```

```
                closest_distance = distance;
            }
            distance_obs += distance;
        }

    }
    //se a distancia até algum dos obstaculos for muito pequena adiciona-se uma penalidade alta
    if(closest_distance < 5.0){
        penalidade = 3000.0;
        if(PREDACAO){
            //na predação, cria-se um novo indivíduo e retorna-se seu fitness
                for (int j = 0; j < ARMS; j++){
                angles[j] = rand()%pi20000*1.0/10000;
            }
            return fitness(origin, goal, angles, obstaculos);
        }
    }
    //retorna a soma da distancia até o objetivo, a soma das distancias até o obstáculo e a penalidade de intersecção com algum obstaculo
    return sqrt((x - goal[0])*(x - goal[0]) + (y - goal[1])*(y - goal[1])) + (5000.0/ distance_obs) + penalidade;
}

// passo 3 - retorna o indice do melhor
int selection(Angles *pop, double origin[2], double goal[2], OBSTACULOS *obstaculos) {
        int best = 0;
        for (int i = 1; i < POP; ++i) {
        if (fitness(origin, goal, pop[i].val, obstaculos) < fitness(origin, goal, pop[best].val, obstaculos)){
        best = i;
        }
        }
```

```
        return best;
    }

    // passos 4, 5, 6 - crossover, mutação e nova população
    void crossover(Angles *pop, int best, double mutation_rate) {
        int i, j;
        for (i = 0; i < POP; i++) {
        if (i == best)
        continue;
        for (j = 0; j < ARMS; j++) { // crossover
        pop[i].val[j] += pop[best].val[j];
        pop[i].val[j] /= 2;
        if (rand() % 2) // mutação
            pop[i].val[j] *= (1 + mutation_rate);
        else
            pop[i].val[j] *= (1 - mutation_rate);
        }
        }
    }

    // mostrar resultados
    void show_result(Angles *pop, double origin[2], double goal[2], OBSTACULOS
*obstaculos) {
        int i, j;
        for (i = 0; i < POP; ++i) {
        printf("Individual: %d: ", i);
        for (j = 0; j < ARMS; j++) {
        printf("%.4lf  ", pop[i].val[j]);
        }
        printf("Fitness: %.2lf\n", fitness(origin, goal, pop[i].val, obstaculos));
        }
        printf("Best index: %d\n\n", selection(pop, origin, goal, obstaculos));
    }
```

```
void destroy(Angles *pop) {
        int i;
        for (i = 0; i < POP; i++) {
        free(pop[i].val);
        }
        free(pop);
}
```

APÊNDICE C – CABEÇALHOS DAS FUNÇÕES

```
#ifndef ROTA_DRONE_H
#define ROTA_DRONE_H
#define POP 5   // number of individuals
#define ARMS 10 // number of arms
#define PI 3.14159265359
#define ARM_SIZE 10.0
#define NUM_OBSTACULOS 5

typedef struct angles {
        double *val;
} Angles;

typedef struct OBSTACULOS{
   double cord[2];
}OBSTACULOS;

Angles *init_pop();
OBSTACULOS *init_obstaculos();
double fitness(double origin[2], double goal[2], double *angles, OBSTACULOS *obstaculos);
int selection(Angles *pop, double origin[2], double goal[2], OBSTACULOS *obstaculos);
void crossover(Angles *pop, int best, double mutation_rate);
void show_result(Angles *pop, double origin[2], double goal[2], OBSTACULOS *obstaculos);
void destroy(Angles *pop);
#endif
```

Anexos

ANEXO A – BRAÇO ROBÓTICO 3D

```
// Código principal
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include "robot_arms.h"
 float origin[3] = {80.0, 20.0, 0.0}; //origin
float goal[3] = {40.0, 30.0, 10.0}; //goal
float mutation_rate = 0.1;
Angles *pop;
int best;
 /*
void Draw(void) {
        //draw all arms
   float x, y, prevx = origin[0], prevy = origin[1];
        int i, j;
        for ( j = 0; j < POP; j++){
        glBegin(GL_LINE_STRIP); //draw lines in sequence
        glColor3f(0.5f, 0.5f, 0.5f); //define gray
        prevx = origin[0];
        prevy = origin[1];
        glVertex2f(prevx, prevy);
        for ( i = 0; i < ARMS; i++){
                x = prevx + cos(pop[j].val[i])*ARM_SIZE;
                        y = prevy + sin(pop[j].val[i])*ARM_SIZE;
                        prevx = x;
                        prevy = y;
```

```
                glVertex2f(x, y);
            }
            glEnd();
            }
            //draw best arms
            glBegin(GL_LINE_STRIP);
            glColor3f(1.0f, 1.0f, 1.0f);
            //float x, y, prevx = origin[0], prevy = origin[1];
            //int i;
            prevx = origin[0];
            prevy = origin[1];
            glVertex2f(prevx, prevy);
            for (i = 0; i < ARMS; i++){
                x = prevx + cos(pop[best].val[i])*ARM_SIZE;
                y = prevy + sin(pop[best].val[i])*ARM_SIZE;
                prevx = x;
                prevy = y;
                glVertex2f(x, y);
            }
}*/
 void Keyboard(char key){
            //ENTER
   if (key == '\n' || key == '\0'){
        crossover(pop, best, mutation_rate);
        best = selection(pop, origin, goal);
        show_result(pop, origin, goal);
            }
            //reset
            if(key == 'r'){
```

```
            destroy(pop);
            printf("\n-----------------------------------------------\n");
            printf("\nOld population destroyed and a new initialized!\n\n");
            pop = init_pop();
            best = selection(pop, origin, goal);
        show_result(pop, origin, goal);
        mutation_rate = 0.1;
          }
          //increase mutation rate
          if (key == 'p'){
           mutation_rate *= 10.0;
          }
          //decrease mutation rate
          if (key == 'o'){
           mutation_rate /= 10.0;
          }
}
 //just explain the commands
void instruction(){
   printf("\nr: restart\n");
   printf("ENTER: new generation\n");
   printf("p: increase mutation rate\n");
   printf("o: decrease mutation rate\n");
   printf("e: exit\n\n");
   printf("------------------------\n\n");
}

int main(int argc, char *argv[]){
          char key = '\0';
```

```
                //instruction();
                pop = init_pop();
                best = selection(pop, origin, goal);
                while (key != 'e'){
                //show_result(pop, origin, goal);
                instruction();
                key = getchar();
                Keyboard(key);
                }
            destroy(pop);

            return 0;
        }
// Funções de algoritmo evolutivo
        #include <stdio.h>
        #include <stdlib.h>
        #include <time.h>
        #include <string.h>
        #include <math.h>
        #include "robot_arms.h"
         //step 1
        //declara um array de individuos populados
        Angles *init_pop(){
            Angles *pop = (Angles *)malloc(POP * sizeof(Angles));
            if(pop == NULL){
                printf("ERROR! pop is null\n");
                exit(0);
            }
            int i;
```

```
    int pi20000 = 2*PI*100; //2PI = 360 graus
    srand(time(NULL));
  //popula individuo com numeros aleatorios
    for (i = 0; i < POP; i++){
        pop[i].val = (float *)malloc(ARMS * sizeof(float));
        pop[i].z_plan = (float *)malloc(ARMS * sizeof(float));
        for (int j = 0; j < ARMS; j++){
            pop[i].val[j] = rand()%pi20000*1.0/100; //criar um numero randomico entre 0 e 2*PI
            pop[i].z_plan[j] = ((rand()%pi20000*1.0/2)/100)-(PI/2); //criar um numero randomico entre 0 e 2*PI
        }
    }
     return pop;
  }

  //step 2
  //evalute, in this case the less the better
  float fitness(float origin[3], float goal[3], float *angles, float *z_plan){
    float x, y, z, prevx = origin[0], prevy = origin[1], prevz = origin[2];
    int i;
    for (i = 0; i < ARMS; i++){
        x = prevx + cos(angles[i])*ARM_SIZE;
        y = prevy + sin(angles[i])*ARM_SIZE;
        z = prevz + sin(z_plan[i])*ARM_SIZE;
        prevx = x;
        prevy = y;
        prevz = z;
    }
    return sqrt((x - goal[0])*(x - goal[0]) + (y - goal[1])*(y - goal[1]) + (z - goal[2])*(z - goal[2]));
```

```
}
//step 3
//return the index of the best
int selection(Angles *pop, float origin[3], float goal[3]){
   int best = 0;
   for (int i = 1; i < POP; ++i){
        if(fitness(origin, goal, pop[i].val, pop[i].z_plan) < fitness(origin, goal, pop[best].val, pop[best].z_plan))
             best = i;
   }
   return best;
}
//step 4, 5, 6
//crossover, mutation and new population
void crossover(Angles *pop, int best, float mutation_rate){
   int i, j;
   for (i = 0; i < POP; i++){
        if (i == best) continue;
        for (j = 0; j < ARMS; j++){ //crossover
             pop[i].val[j] += pop[best].val[j];
             pop[i].val[j] /= 2;
             pop[i].val[j] += pop[best].z_plan[j];
             pop[i].val[j] /= 2;
             if (rand()%2) pop[i].val[j] *= (1+mutation_rate); //mutation
             else pop[i].val[j] *= (1-mutation_rate);
             if (rand()%2) pop[i].z_plan[j] *= (1+mutation_rate);
             else pop[i].z_plan[j] *= (1-mutation_rate);
        }
   }
}
```

```
//show results
void show_result(Angles *pop, float origin[3], float goal[3]){
    float x, y, z, prevx = origin[0], prevy = origin[1], prevz = origin[2];
    int i, j;
    for (i = 0; i < POP; ++i){
            prevx = origin[0];
            prevy = origin[1];
            prevz = origin[2];
            for (j = 0; j < ARMS; j++){
           x = prevx + cos(pop[i].val[j])*ARM_SIZE;
           y = prevy + sin(pop[i].val[j])*ARM_SIZE;
           z = prevz + sin(pop[i].z_plan[i])*ARM_SIZE;
           prevx = x;
           prevy = y;
           prevz = z;
            }
            printf("Fitness[%d]: %.2f\n", i, fitness(origin, goal, pop[i].val, pop[i].z_plan));
    }
    printf("Best index: %d\n\n", selection(pop, origin, goal));
}
//free memory
void destroy(Angles *pop){
    int i;
    for (i = 0; i < POP; i++){
            free(pop[i].val);
            free(pop[i].z_plan);
    }
    free(pop);
}
```

```
// Cabeçalhos das funções
        #ifndef ROBOT_ARMS_H
        #define ROBOT_ARMS_H
        #define POP 10 //number of individuals
        #define ARMS 10 //number of arms
        #define PI 3.14159265359
        #define ARM_SIZE 10.0

        typedef struct angles{
           float *val;
           float *z_plan;
        }Angles;

        Angles *init_pop();
        float fitness(float origin[3], float goal[3], float *angles, float *z_plan);
        int selection(Angles *pop, float origin[3], float goal[3]);
        void crossover(Angles *pop, int best, float mutation_rate);
        void show_result(Angles *pop, float origin[3], float goal[3]);
        void destroy(Angles *pop);
        #endif.
```