Министерство образования и науки РФ
Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

КУРСОВАЯ РАБОТА НА ТЕМУ
«Планирование палета БПЛА с использованием оптимизационного алгоритма AQUILA»
по дисциплине «Промышленное программирование»

Выполнила:
студентка гр. 3540201/20301 _____ Климова О. А.
подпись, дата

Проверил:
к.т.н. _____ Моторин Д. С.
подпись, дата

Санкт-Петербург
2022

Содержание

Введение

Беспилотные летательные аппараты (БПЛА) благодаря доступности мобильности и стоимости широко используются в таких областях, как безопасность, наблюдение, спасение, сельское хозяйство и метеорология [1].

Большинство миссий БПЛА требуют планирование пути – определение оптимальной траектории без столкновений между исходным и целевым узлами с учетом различных ограничений, связанных с БПЛА и окружающей средой.

Планирование траектории движения БПЛА – это сложная задача, для решения которой широко используются алгоритмы роевого интеллекта (Swarm Intelligence algorithms) [2], которые предлагают высокую эффективность в отношении их сходимости и качества сгенерированного пути.

Aquila Optimization (AO) — метаэвристика роевого интеллекта, которая имитирует методы охоты орла на разные виды добычи. Алгоритм Aquila обеспечивает эффективную глобальную разведывательную способность.

В данной работы будет продемонстрирован оптимизационный алгоритм построения пути БПЛА на базе метаэвристики роевого интеллекта Aquila Optimization [3].

1. Постановка задачи

Целью данной работы является создание алгоритма Aquila для решения многокритериальной задачи оптимизации, направленной на построение оптимального пути с точки зрения длины траектории (Path) и избегания препятствий (Obs). Таким образом, общая целевая функция [4] данной задачи включает в себя две целевых функции:

$$C_{Cost} = w_1 C_{Path} + w_2 C_{Obs}, \quad (1)$$

где w_1, w_2 – вклады целевых функций.

Каждая целевая функция описывает поведение определенного параметра:

$$C_{Path} = \sum_{\mathrm{i}=0}^{n} \|\overrightarrow{L_{\mathrm{i}} L_{\mathrm{i}+1}}\|, \quad (2)$$

где $\overrightarrow{L_{\mathrm{i}} L_{\mathrm{i}+1}}$ – вектора одинаковой длины, составляющие путь.

$$C_{Obs} = \sum_{i=0}^{n} \sum_{j=0}^{m-1} O_j\ (\overrightarrow{L_i L_{i+1}}), \quad (2)$$

где $O_j(\overrightarrow{L_{\mathrm{i}} L_{\mathrm{i}+1}}) = \begin{cases} 0,\ if\ D > R_j + D_{\mathrm{s}} \\ \infty,\ if\ D < R_j + U_{\mathrm{s}}\,. \\ D_{\mathrm{s}} - D,\ otherwise \end{cases}$

Пространство для полета представляет из себя карту местности, координаты начала и конца пути, а также центры и радиусы R_j статических препятствий (рис. 1), которые необходимо огибать во время полета на безопасном расстоянии $D_{\mathrm{s}} - D$ (рис. 2).

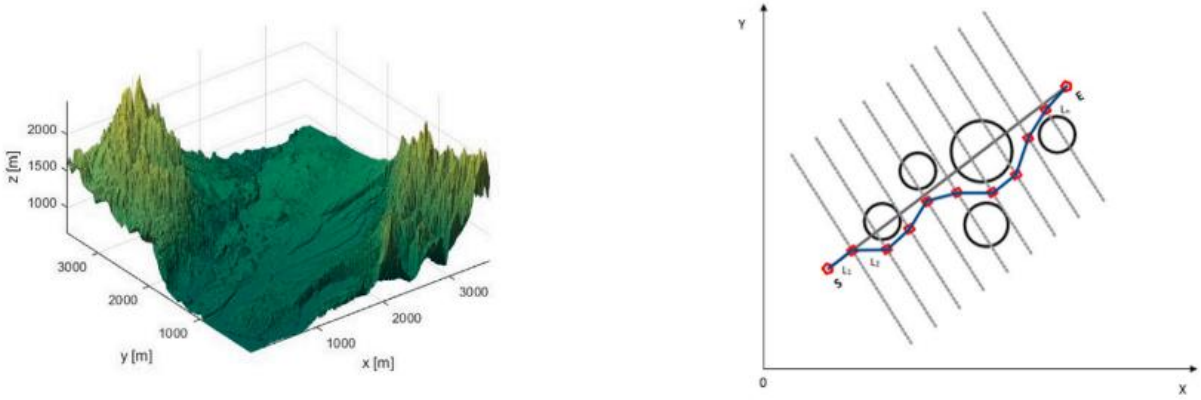

Рисунок 1 – Карта местности и траектория огибания препятствий

Рисунок 2 – Безопасная дистанция для огибания препятствий

Алгоритм Aquila основывается на четырех стратегиях [3], которые при случайном использовании в сочетании друг с другом позволяют получать решение оптимизационной задачи.

1. Expanded exploration (расширенная разведка)

При этой стратегии Aquila совершает высокий взлет с вертикальным наклоном. Математически это представлено следующим образом:

$$X_i^{(t+1)} = X_{best}^t \times \left(1 - \frac{1}{T}\right) + (X_m^t - X_{best}^t \times \text{rand}), \quad (3)$$

где $X_i^{(t+1)}$ – позиция i-го индивидуума на t+1 итерации;

X_{best}^t – наилучшая позиция полученная на t-й итерации;

$rand$ – случайное число в диапазоне от 0 до 1;

T – максимальное число итераций;

X_m^t – среднее значение текущей позиции на t-й итерации.

2. Narrowed exploration (суженная разведка)

При данной стратегии Aquila ловит добычу, изменяя траекторию полета**,** короткой атакой по скольжению, что выражается формулой:

$$X_i^{(t+1)} = X_{best}^t \times \text{Levy}(Dim) + X_r^t + (y - x) \times \text{rand}), \quad (4)$$

где $Levy(Dim)$ – полет Леви в пространственном измерении Dim;

X_r^t – случайное решение;

x и y – описывают спиральную форму движения Aquila.

3. Expanded exploitation (расширенная эксплуатация)

При данной стратегии Aquila выполняет низкий полет с медленной атакой на спуске. Это сформулировано в уравнении:

$$X_i^{(t+1)} = (X_{best}^t - X_m^t) \times a \ - rand + ((UB - LB) \times rand + LB) \times \delta, \quad (5)$$

где α, δ – эксплуатационные параметры;

UB, LB – верхняя и нижняя границы.

4. Narrowed exploitation (суженная эксплуатация)

Эта стратегия называется хождением и захватом добычи. Он используется для охоты на крупную добычу. Это выражается следующим образом:

$$X_i^{(t+1)} = QF \times X_{best}^t - (G_1 \times X(t) \times rand) - G_1 \times Levy(Dim) + rand \times G_1, \quad (6)$$

где G_1 , G_2 – движение Aquila и наклон полета;

QF – функция качества.

2. Описание программного обеспечения

Для решения поставленной задачи будет использоваться пакет прикладных программ для технических вычислений MATLAB.

Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования [5].

Программы, написанные на MATLAB, бывают двух типов — функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции сохраняются в виде текстовых файлов и компилируются в машинный код динамически.

Данное программное обеспечение наилучшим образом подходит для решения поставленной задачи, так как оно предоставляет широкие возможности при работе с матрицами [6] и при построении графиков.

3. Программная реализация

В первую очередь была написана функция Aquila_Map(m_size, m_prop), принимающая пару чисел как размер карты и значения карты в ее углах. Данная функция позволяет фрактально генерировать карту местности (рис. 3).

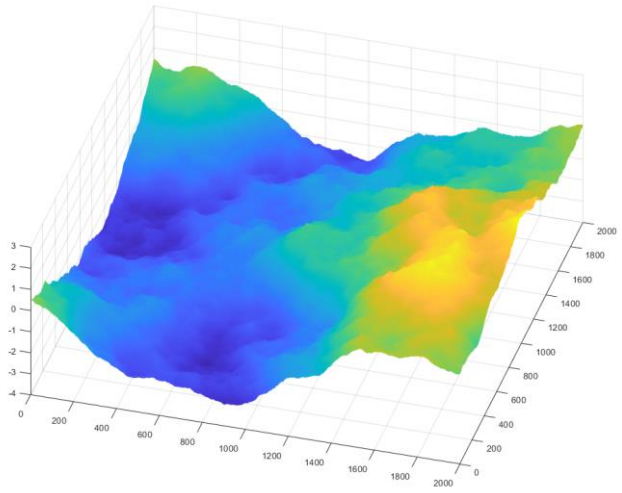

Рисунок 3 – Генерация карты местности с помощью функции Aquila_Map(m_size, m_prop)

Далее случайным образом генерировались точки начала и конца пути, а также точка, являющаяся центром препятствия. Точки начала и конца пути на графике обозначаются черными звездами, а центры препятствий – синими. Препятствие строится в виде цилиндра указанного радиуса с центром в полученной точке (рис. 4).

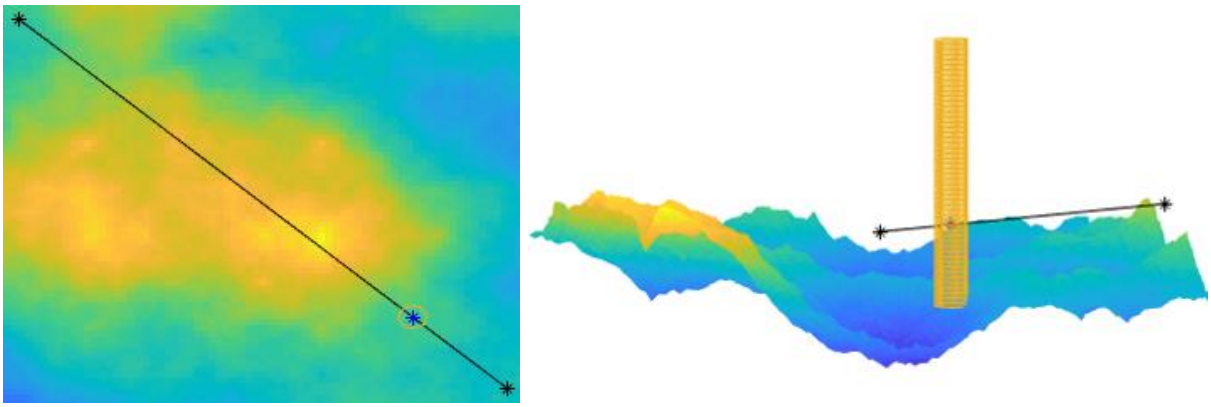

Рисунок 4 – Построение прямого пути и статического препятствия

Затем полученная траектория разбивалась на n+1 отрезок (рис. 5), после чего вызывалась функция Aquila_Get_F(), позволяющая формировать целевую функцию, включающую ограничение на огибание препятствий и длину траектории.

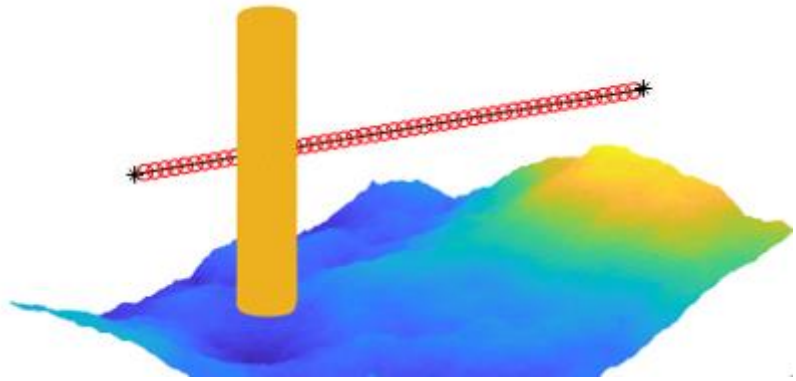

Рисунок 5 – Разделение траектории на n+1 отрезок

Сам алгоритм Aquila реализуется в функции Aquila_AO(), которая вызывается после всего препроцессинга, включающего формирование карты, препятствий и разделение пути на составляющие. В результате работы программы формируется новая траектория (розовые звезды), позволяющая оптимальным образом огибать препятствие (рис. 6).

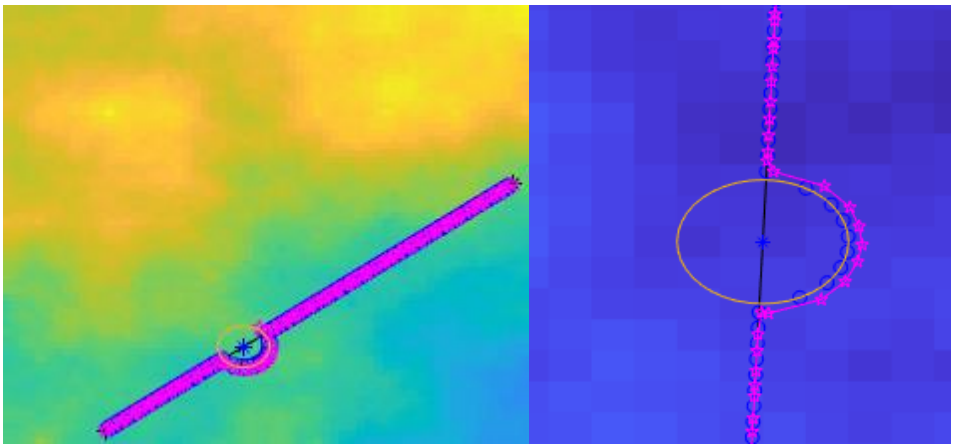

Рисунок 6 – Огибание препятствий

Код программной реализации представлен в Приложении 1.

5. Анализ работоспособности решения

Для проверки корректной работы программной реализации проводились эксперименты, описанные в статье, рассматривающей оптимизацию полета БПЛА.

Всего проводилось три эксперимента, входными параметрами которых являлись координаты точек начала и конца пути, точек-центров препятствий, а также радиусы препятствий (рис. 7).

Cases	Start point	End point	Obstacles Center	Obstacles Radius
1	(1000,1000,600)	(3000,2000,700)	[1400,1300]	100
			[2500,2000]	150
			[2200,1200]	100
			[1700,2000]	200
			[2000,1600]	75
2	(500,800,600)	(2500,2500,700)	[1500,1300]	100
			[2500,2000]	150
			[2200,1200]	100
			[1700,2000]	200
			[2000,1600]	75
			[1000,1500]	100
			[800,1000]	80
			[2200,2300]	75
3	(500,800,600)	(2500,2500,700)	[1500,1300]	100
			[2500,2000]	150
			[2200,1200]	100
			[1700,2000]	200
			[2000,1600]	150
			[1000,1500]	100
			[800,1000]	80
			[2200,2300]	75
			[1400,1600]	80
			[2150,1900]	75

Рисунок 7 – Параметры экспериментов

Для реализации первого эксперимента были заданы пять указанных препятствий, для второго восемь, а для третьего десять.

Масштаб реализуемой карты в 10 раз меньше, описанной в статье, так как большая размерность требует высоких технических мощностей.

В результате первого эксперимента была получена первая траектория (рис. 8, 9).

В результате второго эксперимента была получена вторая траектория (рис. 10, 11).

В результате третьего эксперимента была получена третья траектория (рис. 12, 13).

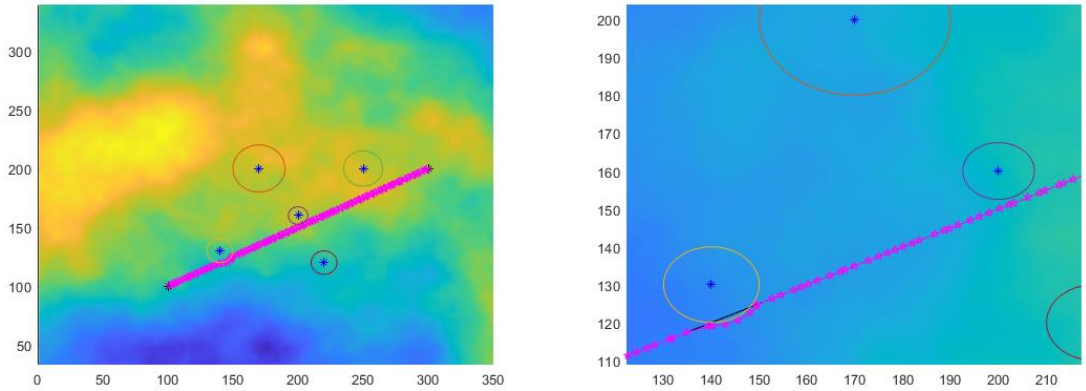

Рисунок 8 – Траектория при пяти препятствиях в 2D

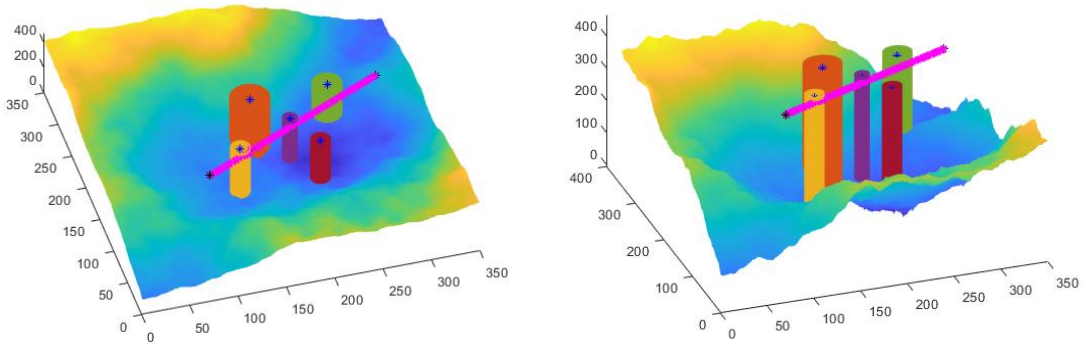

Рисунок 9 – Траектория при пяти препятствиях в 3D

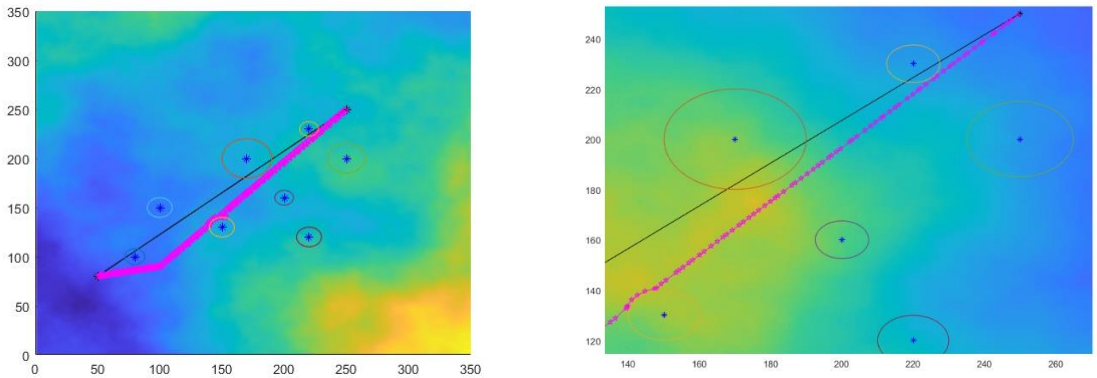

Рисунок 10 – Траектория при восьми препятствиях в 2D

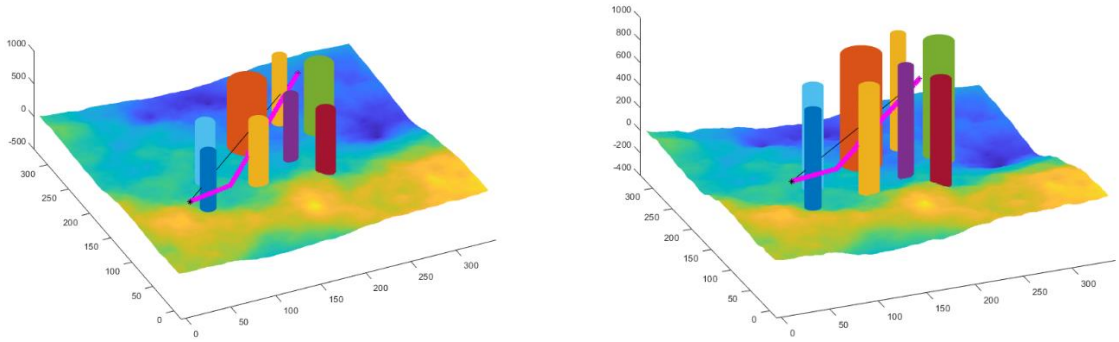

Рисунок 11 – Траектория при восьми препятствиях в 3D

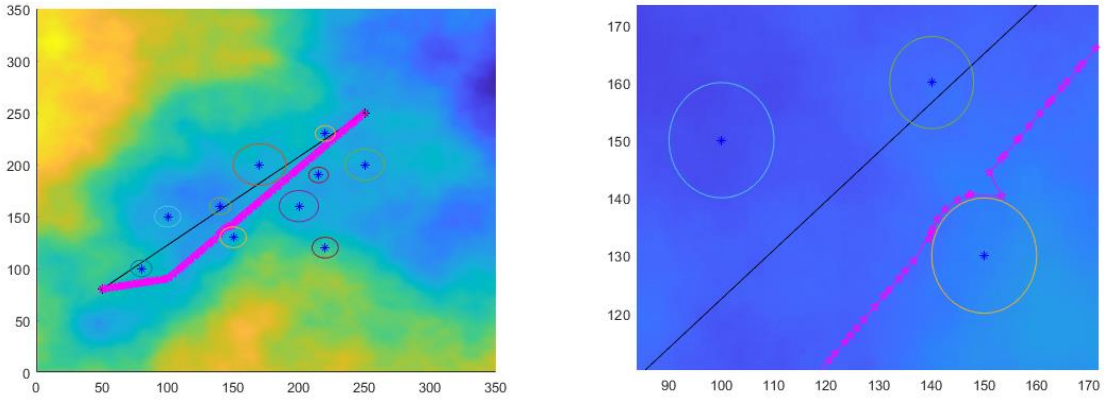

Рисунок 12 – Траектория при десяти препятствиях в 2D

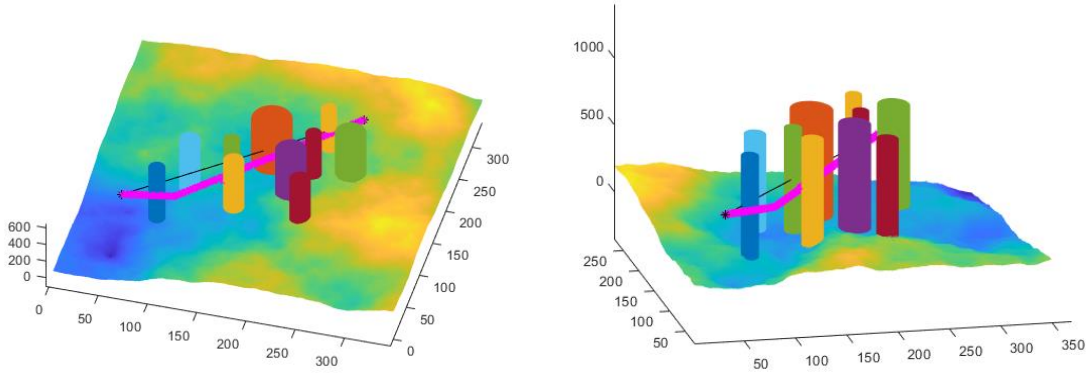

Рисунок 13 – Траектория при десяти препятствиях в 3D

После построения траекторий с различным числом и видом препятствий, полученные графики сравнивались с графиками, представленными в статье (рис. 14).

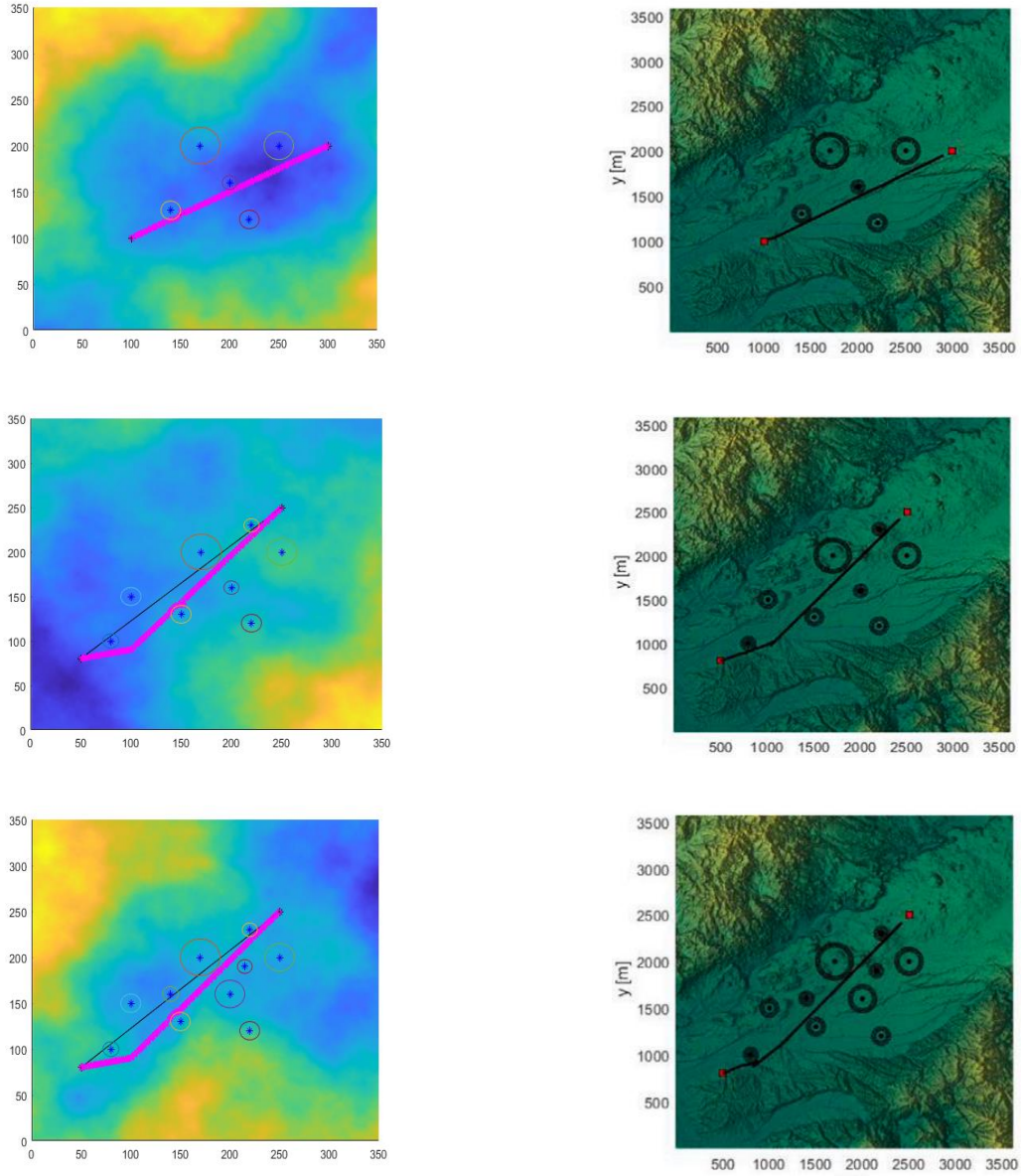

Рисунок 14 – Сравнение результатов

Представленные выше графики похожи, что может свидетельствовать о корректности работы программы.

Выводы

В ходе проделанной работы была решена многокритериальная задача планирования пути БПЛА с учетом ограничений на длину пути, а также количество и размер препятствий.

Для создания программной реализации использовался высокоуровневый интерпретируемый язык MATLAB, который позволил не только решить задачу планирования пути, но и визуально представлять карту местности, местонахождение препятствий и саму траекторию.

Для решения оптимизационной задачи использовался алгоритм Aquila, который в своей основе содержит методы (четыре стратегии) охоты орла на разные виды добычи.

Корректность работы программы проверялась путем проведения трех экспериментов (с пятью, восемью и десятью препятствиями). Полученные результаты сравнивались с результатами экспериментов, имеющими такие же характеристики, из научной статьи. При сравнении графиков был сделан вывод о их схожести и соответственно корректности работы программы.

Список литературы

1. Amylia Ait-Saadi, Yassine Meraihi, Assia Soukane, Amar Ramdane-Cherif, Asma Benmessaoud Gabis. A novel hybrid Chaotic Aquila Optimization algorithm with Simulated Annealing for Unmanned Aerial Vehicles path planning // Computers and Electrical Engineering – [2022]. – URL: https://doi.org/10.1016/j.compeleceng.2022.108461. – (дата обращения: 23.05.2023).

2. Роевой интеллект: Википедия. Свободная энциклопедия. – URL: https://ru.wikipedia.org/wiki/Роевой_интеллект – (дата обращения 23.05.2023).

3. Laith Abualigaha, Dalia Yousrib, Mohamed Abd Elazizc, Ahmed A. Eweesd, Mohammed A. A. Al-qanesse, Amir H. Gandomif. Aquila Optimizer: A novel meta-heuristic optimization Algorithm // Computers & Industrial Engineering – [2021]. – URL: https://www.sciencedirect.com/science/article/abs/pii/S0360835221001546. – (дата обращения: 23.05.2023)

4. Рассказова М.Н. Прикладные задачи математического программирования: учебное пособие / М.Н. Рассказова, Л.С. Рыженко. – Омск.: Издательство ОмГТУ, 2011. – 68 с.

5. MATLAB: Википедия. Свободная энциклопедия. – URL: https://ru.wikipedia.org/wiki/MATLAB_ – (дата обращения 22.05.2023).

6. Беллман Р. Введение в теорию матриц / Р. Беллман. – М.: Наука, 1970. – 368 с.

Приложение 1

Aquila.m

```
clc;
clear;
close all;

figure(1)
hold on;


%генерация коньена
map1 = Aquala_Map([350 350], [0 350 350 0 350]);

max_h = max(map1(:));
min_h = min(map1(:));
hh = max_h+abs(min_h);
z = [hh hh];

%координаты пути
x = [100 300];
y = [100 200];
surf(map1,'edgecolor','none')
plot3(x,y,z,'k-*');

%препятствия
k = 5;
%координаты центров
xc = [140 250 220 170 200];
yc = [130 200 120 200 160];
zc = [hh hh hh hh hh hh];
%радиусы
r = [10 15 10 20 7.5];

%построение препятствий
for i = 1:k
plot3(xc(i),yc(i),zc(i),'b-*');
g = map1(xc(i),yc(i));
t = g:pi/100:max_h+abs(min_h)*pi;
xt1 = r(i)*sin(5*t) + xc(i);
yt1 = r(i)*cos(5*t) + yc(i);
plot3(xt1,yt1,t)
end

% Разбиение пути на n+1 отрезок n точками
n = 100;
Li_x = zeros(1,n);
Li_y = zeros(1,n);
Li_z = zeros(1,n);


for i = 1:n
```

```
lamb = i/(n+1-i);
Li_x(i) = (x(1) + lamb*x(2))/(1+lamb);
Li_y(i) = (y(1) + lamb*y(2))/(1+lamb);
Li_z(i) = (z(1) + lamb*z(2))/(1+lamb);

%plot3(Li_x(i),Li_y(i),Li_z(i),'r-o');
end

%Применение Aquala optimization algorithm
Solution_no=20;  % N особей в алгоритме Aquala
M_Iter=1000;     % Количество итераций: T в алгоритме Aquala

Best_P_x = zeros(1,n+1);
Best_P_y = zeros(1,n+1);
Best_P_z = zeros(1,n+1);

[LB,UB,Dim,F_obj]= Aquala_Get_F(xc(1), yc(1), r(1), x(1), y(1), Li_x(1), Li_y(1));
[~,Best_P,~] = Aquila_AO(Solution_no,M_Iter,LB,UB,Dim,F_obj);
Best_P_x(1) = Best_P(1);
Best_P_y(1) = Best_P(2);
Best_P_z(1) = z(1);

for i = 1:n-1
[LB,UB,Dim,F_obj]= Aquala_Get_F(xc(1), yc(1), r(1), Li_x(i), Li_y(i), Li_x(i+1), Li_y(i+1));
[Best_FF(i),Best_P,conv] = Aquila_AO(Solution_no,M_Iter,LB,UB,Dim,F_obj);
Best_P_x(i+1) = Best_P(1);
Best_P_y(i+1) = Best_P(2);
Best_P_z(i+1) = Li_z(i);
end

[LB,UB,Dim,F_obj]= Aquala_Get_F(xc(1), yc(1), r(1), Li_x(n), Li_y(n), x(2), y(2));
[Best_FF(n+1),Best_P,conv] = Aquila_AO(Solution_no,M_Iter,LB,UB,Dim,F_obj);
Best_P_x(n+1) = Best_P(1);
Best_P_y(n+1) = Best_P(2);
Best_P_z(n+1) = z(2);

plot3(Best_P_x,Best_P_y,Best_P_z,'m-p')

display('The optimal path is built');

Best_Path = 0;
for i = 1:n
Best_Path = Best_Path + (sqrt((Best_P_x(i+1) - Best_P_x(i)).^2 + (Best_P_y(i+1) -
Best_P_y(i)).^2 + (Best_P_z(i+1) - Best_P_z(i)).^2));
end

display(['The length of path found by AO is : ', num2str(Best_Path)]);
display(['The best optimal value of the objective funciton found by AO is : ',
num2str(sum(Best_FF))]);
```

Aquila_Map.m

```
%Генерация карты
function MAP = Aquala_Map(m_size, m_prop)
% Формирование основы карты
n=ceil(log2(max(max(m_size(1), m_size(2)))-1));
MAP = inf(2^n+1, 2^n+1);
% Коэффициент разброса вероятности
h = m_prop(5);

% Значения в углах карты
if ismember(Inf, m_prop(1:4)) || ismember(-Inf, m_prop(1:4))
   % Если среди начальных значений есть невозможные (+/- бесконечность)
   % то генерируем случайные значения
   MAP(1,1) = rand;
   MAP(size(MAP,1),1) = rand;
   MAP(1,size(MAP,2)) = rand;
   MAP(size(MAP,1),size(MAP,2)) = rand;
else
   % В противном случае берем заданные значения
   MAP(1,1) = m_prop(1);
   MAP(size(MAP,1),1) = m_prop(2);
   MAP(1,size(MAP,2)) = m_prop(3);
   MAP(size(MAP,1),size(MAP,2)) = m_prop(4);
end

% Генерация остальных значений карты
for i = 1 : n
   % ширина (глубина) шага вычисления значений функции (шаг квадратов)
   m = 2^(n-i);
   for j = 1:2:2^i
      for k = 1:2:2^i
         if MAP(j*m+1, k*m+1) == inf
            MAP(j*m+1, k*m+1) = (MAP((j+1)*m+1, (k+1)*m+1) + MAP((j-1)*m+1, (k-1)*m+1)
+ MAP((j+1)*m+1, (k-1)*m+1) + MAP((j-1)*m+1, (k+1)*m+1))/4 + sign(rand-0.5)*rand*h;
         end
      end
   end

   % Шаг ромбов
   for j = 0 : 2^i
      % Проверка наличия смещения на текущей заполняемой строке
      % (наличие первного вычисленного значения)
      if mod(j,2)==0
         sk=1;
      else
         sk=0;
      end
      for k = sk : 2 : 2^i
         if MAP(j*m+1, k*m+1) == inf
            s = [];
            p = 0;
```

```
            % Проверка границ карты
            if (j+1)*m+1 <= 2^n+1
               s = [s, MAP((j+1)*m+1, (k)*m+1)];
               p = p + 1;
            end
            if (j-1)*m+1 >= 1
               s = [s, MAP((j-1)*m+1, (k)*m+1)];
               p = p + 1;
            end
            if (k+1)*m+1 <= 2^n+1
               s = [s, MAP((j)*m+1, (k+1)*m+1)];
               p = p + 1;
            end
            if (k-1)*m+1 >= 1
               s = [s, MAP((j)*m+1, (k-1)*m+1)];
               p = p + 1;
            end
            MAP(j*m+1, k*m+1) = sum(s)/p + sign(rand-0.5)*rand*h;
         end
      end
   end
   % Снижение влияния разброса
   h = h/2;
end

% % Обрезка карты по размеру
if size(MAP,1) ~= m_size(1) && size(MAP,2) ~= m_size(2)
MAP = MAP(1+round((size(MAP,1)-m_size(1))/2):m_size(1)+round((size(MAP,1)-
m_size(1))/2), 1+round((size(MAP,2)-m_size(2))/2):m_size(2)+round((size(MAP,2)-
m_size(2))/2));
end

end
```

Aquila_Get_F.m

```
function [LB,UB,Dim,F_obj] = Aquala_Get_F(xc, yc, r, Li_x, Li_y, Linext_x, Linext_y)
F_obj = @F1;
LB=-350;
UB=350;
Dim = 2;

% F1
function o = F1(x)

%центр препятствия
c_pr = [xc yc];
R = r;
Ds = 0.9; %значение данного параметра в статье 0.906
Us = 0.3; %значение данного параметра в статье 0.302

%расстояние от точки до прямой
A = (x(2)-Li_y)/(x(1)-Li_x);
B = -1;
C = Li_y-Li_x*(x(2)-Li_y)/(x(1)-Li_x);
D = abs(A*c_pr(1)+B*c_pr(2)+C)/sqrt(A^2+B^2);

% параметр для целевой функции
if D > R + Ds
o_pr = 0;
elseif D < R + Us
o_pr = inf;
else
o_pr = Ds - D;
end

% целевая функция состоящая из минимизации расстояния и возможности огибания препятствий
if sqrt((c_pr(1)-x(1))^2+(c_pr(2)-x(2))^2)> R + Ds
   o= sqrt((Li_x - x(1)).^2 + (Li_y - x(2)).^2) + sqrt((Linext_x - x(1)).^2 + (Linext_y - x(2)).^2);
else
   o= sqrt((Li_x - x(1)).^2 + (Li_y - x(2)).^2) + sqrt((Linext_x - x(1)).^2 + (Linext_y - x(2)).^2) +
o_pr;
end
end

end
```

Aquila_AO.m

```
%Aquala optimization algotithm

function [Best_FF,Best_P,conv]= Aquila_AO(N,T,LB,UB,Dim,F_obj)
Best_P=zeros(1,Dim);
Best_FF=inf;

X=Aquila_Initialization(N,Dim,UB,LB);
Xnew=X;
Ffun=zeros(1,size(X,1));
Ffun_new=zeros(1,size(Xnew,1));

t=1;

alpha=0.1;
delta=0.1;

while t<T+1
   for i=1:size(X,1)
      F_UB=X(i,:)>UB;
      F_LB=X(i,:)<LB;
      X(i,:)=(X(i,:).*(~(F_UB+F_LB)))+UB.*F_UB+LB.*F_LB;
      Ffun(1,i)=F_obj(X(i,:));
      if Ffun(1,i)<Best_FF
         Best_FF=Ffun(1,i);
         Best_P=X(i,:);
      end
   end

   G2=2*rand()-1; % Eq. (16)
   G1=2*(1-(t/T));  % Eq. (17)
   to = 1:Dim;
   u = .0265;
   r0 = 10;
   r = r0 +u*to;
   omega = .005;
   phi0 = 3*pi/2;
   phi = -omega*to+phi0;
   x = r .* sin(phi);  % Eq. (9)
   y = r .* cos(phi); % Eq. (10)
   QF=t^((2*rand()-1)/(1-T)^2); % Eq. (15)
      %-------------------------------------------------------------------------------
   for i=1:size(X,1)
      %-------------------------------------------------------------------------------
      if t<=(2/3)*T
         if rand <0.5
            Xnew(i,:)=Best_P(1,:)*(1-t/T)+(mean(X(i,:))-Best_P(1,:))*rand(); % Eq. (3) and Eq. (4)
            Ffun_new(1,i)=F_obj(Xnew(i,:));
            if Ffun_new(1,i)<Ffun(1,i)
               X(i,:)=Xnew(i,:);
               Ffun(1,i)=Ffun_new(1,i);
```

```
            end
        else
            %-------------------------------------------------------------------------------
            Xnew(i,:)=Best_P(1,:).*Levy(Dim)+X((floor(N*rand()+1)),:)+(y-x)*rand;      % Eq. (5)
            Ffun_new(1,i)=F_obj(Xnew(i,:));
            if Ffun_new(1,i)<Ffun(1,i)
                X(i,:)=Xnew(i,:);
                Ffun(1,i)=Ffun_new(1,i);
            end
        end
        %-------------------------------------------------------------------------------
    else
        if rand<0.5
            Xnew(i,:)=(Best_P(1,:)-mean(X))*alpha-rand+((UB-LB)*rand+LB)*delta;   % Eq. (13)
            Ffun_new(1,i)=F_obj(Xnew(i,:));
            if Ffun_new(1,i)<Ffun(1,i)
                X(i,:)=Xnew(i,:);
                Ffun(1,i)=Ffun_new(1,i);
            end
        else
            %-------------------------------------------------------------------------------
            Xnew(i,:)=QF*Best_P(1,:)-(G2*X(i,:)*rand)-G1.*Levy(Dim)+rand*G2; % Eq. (14)
            Ffun_new(1,i)=F_obj(Xnew(i,:));
            if Ffun_new(1,i)<Ffun(1,i)
                X(i,:)=Xnew(i,:);
                Ffun(1,i)=Ffun_new(1,i);
            end
        end
    end
    %-------------------------------------------------------------------------------
%     if mod(t,100)==0
%         display(['At iteration ', num2str(t), ' the best solution fitness is ', num2str(Best_FF)]);
%     end
    conv(t)=Best_FF;
    t=t+1;
end

end
function o=Levy(d)
beta=1.5;
sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);
u=randn(1,d)*sigma;v=randn(1,d);step=u./abs(v).^(1/beta);
o=step;
end
```