Documentatie

Task 3: Structuri de date utilizate pentru reprezentarea mișcărilor robotului

1. Reprezentarea Cinematicii Robotului

Un braț robotic poate fi modelat folosind structurile:

- **Vectori și matrici** – pentru pozițiile și unghiurile articulațiilor.
- **Structuri** – pentru stocarea parametrilor robotului.
- **Grafuri** – pentru planificarea mișcărilor.

a) Vectori și Matrici:

- Vectorul pozițiilor finale ale robotului:

```
P = [x1, y1; x2, y2; ...; xn, yn]; % Coordonatele punctelor traiectoriei
```

- Vectorul unghiurilor articulațiilor:

```
Theta = [theta1, theta2, ..., thetan];
```

- Matrici pentru transformările omogene (utilizate în cinematică directă):

```
T = [cos(theta) -sin(theta) x;
    sin(theta)  cos(theta) y;
    0        0        1];
```

b) Structuri pentru stocarea parametrilor robotului

```
robot.L1 = 5; % Lungimea primei legături
robot.L2 = 3; % Lungimea celei de-a doua legături
robot.theta = [0, 0]; % Unghiurile inițiale
```

c) Grafuri pentru planificarea mișcărilor

Reprezentarea unei traiectorii ca graf:

- **Noduri:** Puncte din spațiul de lucru al robotului.

- **Arce:** Conexiuni între puncte, reprezentând mișcările posibile.
  Se poate folosi digraph() pentru a reprezenta rețeaua de mișcare a robotului.

Cod Matlab:

```
clc; clear; close all;


% === Definirea structurii pentru mișcările robotului ===
robot_movement = struct('x', [], 'y', [], 'theta1', [], 'theta2', []);


% === Parametri braț robotic ===
L1 = 5; % Lungimea primului segment
L2 = 3; % Lungimea celui de-al doilea segment


% === Definirea punctelor traiectoriei ===
x_traj = linspace(2, 6, 10);
y_traj = linspace(3, 5, 10);


% === Calculul unghiurilor folosind cinematica inversă ===
for i = 1:length(x_traj)
    xe = x_traj(i);
    ye = y_traj(i);


    % Cinematica inversă
    c2 = (xe^2 + ye^2 - L1^2 - L2^2) / (2 * L1 * L2);
    if abs(c2) > 1
        continue; % Evităm soluții imposibile
```

```
    end
    s2 = sqrt(1 - c2^2);
    theta2 = atan2(s2, c2);
    k1 = L1 + L2 * cos(theta2);
    k2 = L2 * sin(theta2);
    theta1 = atan2(ye, xe) - atan2(k2, k1);

    % Stocăm valorile în structura de date
    robot_movement(i).x = xe;
    robot_movement(i).y = ye;
    robot_movement(i).theta1 = theta1;
    robot_movement(i).theta2 = theta2;
end

% === Afișare structuri de date stocate ===
disp('Mișcările robotului stocate:');
disp(robot_movement);

% === Funcție de animație ===
animeaza_brat(robot_movement, L1, L2, 'Traiectorie Robot');

% === Funcție pentru animația brațului robotic ===
function animeaza_brat(movement_data, L1, L2, titlu)
    figure; axis([-10 10 -10 10]); hold on; grid on;
    xlabel('X'); ylabel('Y'); title(titlu);
```

```
    h1 = plot([0, 0], [0, 0], 'ro-', 'LineWidth', 3);
    h2 = plot([0, 0], [0, 0], 'bo-', 'LineWidth', 3);
    joint1 = plot(0, 0, 'ko', 'MarkerFaceColor', 'g');
    joint2 = plot(0, 0, 'ko', 'MarkerFaceColor', 'b');

    for i = 1:length(movement_data)
        x1 = L1 * cos(movement_data(i).theta1);
        y1 = L1 * sin(movement_data(i).theta1);
        x2 = x1 + L2 * cos(movement_data(i).theta1 + movement_data(i).theta2);
        y2 = y1 + L2 * sin(movement_data(i).theta1 + movement_data(i).theta2);

        set(h1, 'XData', [0 x1], 'YData', [0 y1]);
        set(h2, 'XData', [x1 x2], 'YData', [y1 y2]);
        set(joint1, 'XData', x1, 'YData', y1);
        set(joint2, 'XData', x2, 'YData', y2);

        pause(0.1);
    end
    hold off;
end
```

Task 4: Algoritmi exacți și heuristici pentru optimizarea mișcărilor

Optimizarea mișcărilor robotului se poate face utilizând algoritmi exacți care garantează soluția optima și algoritmi euristici care găsesc soluții aproximative într-un timp rezonabil.

1. Algoritmi exacți

- caută soluția optimă prin metode matematice riguroase:

a) Programare Neliniară (Nonlinear Programming - NLP)

- Se minimizează o funcție obiectiv, cum ar fi consumul de energie sau variația unghiurilor.
- se poate folosi fminunc sau fmincon.

```
options = optimoptions('fminunc', 'Algorithm', 'quasi-newton');
sol = fminunc(@cost_variatie_unghiuri, x0, options);
```

- Se aplică constrângeri legate de limitele articulațiilor și coliziuni.

b) Metoda de Planificare Rapidă a Mișcării

- Construiește un arbore de căutare care explorează rapid spațiul de lucru.
- Evită obstacolele și găsește drumuri eficiente.

2. Algoritmi euristici

- utilizați când problema este prea complexă pentru a fi rezolvată exact.

a) Algoritmi Genetici (GA - Genetic Algorithms)

- Simulează selecția naturală pentru a găsi cea mai bună traiectorie.
- se poate folosi ga din Optimization Toolbox:

```
options = optimoptions('ga', 'PopulationSize', 50, 'MaxGenerations', 100);
sol = ga(@cost_variatie_unghiuri, numel(x0), [], [], [], [], lb, ub, [], options);
```

b) Optimizare cu Roi de Particule

- Simulează un grup de particule care explorează spațiul pentru a găsi o soluție optimă.
- Se utilizează metoda particleswarm.

c) Optimizare prin Algoritmul de Furnici (ACO - Ant Colony Optimization)

- Inspirat de comportamentul furnicilor care găsesc cel mai scurt drum.
- utilizat pentru optimizarea traseului unui robot mobil.

Deci pentru reprezentarea mișcărilor, se utilizează vectori, matrici, structuri și grafuri, iar pentru optimizare, metodele exacte (programare neliniară, RRT) oferă soluții precise, dar pot fi lente, în timp ce metodele euristice (GA, PSO, ACO) sunt mai rapide, dar aproximative.

Cod MATLAB:

```
clc; clear; close all;

% === Definirea parametrilor brațului robotic ===
L1 = 5; L2 = 3;

% === Generarea unei traiectorii inițiale ===
x_init = linspace(2, 6, 10);
y_init = linspace(3, 5, 10);
x0 = [x_init; y_init];
x0 = x0(:); % Transformăm în vector coloană

% === Funcția cost: Minimizarea variației unghiurilor ===
fun = @(v) cost_variatie_unghiuri(v, L1, L2);

%% **1. Optimizare Exactă: Metoda Quasi-Newton (fminunc)**
options = optimoptions('fminunc', 'Algorithm', 'quasi-newton', 'Display', 'iter');
[x_opt_exact, cost_exact] = fminunc(fun, x0, options);
```

```
%% **2. Optimizare Euristică: Algoritm Genetic (GA)**
options_ga = optimoptions('ga', 'Display', 'iter', 'PopulationSize', 50, 'MaxGenerations', 100);
[x_opt_heuristic, cost_heuristic] = ga(fun, length(x0), [], [], [], [], [], [], [], options_ga);

% === Reconstruim traiectoriile optimizate ===
x_opt1 = reshape(x_opt_exact, 2, []);
x_opt2 = reshape(x_opt_heuristic, 2, []);

% === Vizualizare traiectorii ===
figure; hold on; grid on;
plot(x_init, y_init, 'k--o', 'LineWidth', 1.5, 'DisplayName', 'Traiectorie inițială');
plot(x_opt1(1, :), x_opt1(2, :), 'r-o', 'LineWidth', 2, 'DisplayName', 'Optimizare Exactă (Quasi-Newton)');
plot(x_opt2(1, :), x_opt2(2, :), 'b-o', 'LineWidth', 2, 'DisplayName', 'Optimizare Euristică (GA)');
xlabel('X'); ylabel('Y'); legend; title('Compararea Optimizărilor');
hold off;

%% === Funcția cost pentru minimizarea variațiilor de unghiuri ===
function J = cost_variatie_unghiuri(v, L1, L2)
    p = reshape(v, 2, []); x = p(1, :); y = p(2, :);
    theta = zeros(1, length(x));
    for i = 1:length(x)
        xe = x(i); ye = y(i);
        c2 = (xe^2 + ye^2 - L1^2 - L2^2) / (2 * L1 * L2);
        if abs(c2) > 1
```

```
            J = inf; return;
        end
        s2 = sqrt(1 - c2^2);
        theta2 = atan2(s2, c2);
        k1 = L1 + L2 * cos(theta2);
        k2 = L2 * sin(theta2);
        theta1 = atan2(ye, xe) - atan2(k2, k1);
        theta(i) = theta1;
    end
    J = sum(diff(theta).^2);
end
```