پیمایش ماز با قانون دست راست/چپ در محیط گریدی و نمایش انیمیشنی مسیر

چکیده

در این پروژه، مسئلهٔ پیمایش ماز در یک محیط گریدی با استفاده از راهبرد واکنشی «دیواردنبال‌کن» (Wall-Following) بررسی شده است. ربات بدون داشتن نقشهٔ قبلی از محیط و تنها بر اساس اطلاعات محلی، با استفاده از قانون دست راست یا دست چپ، مسیر خود را از نقطهٔ شروع به سمت هدف پیدا می‌کند. الگوریتم پیشنهادی بر روی سه ماز مختلف آزمایش شده و نتایج شامل مسیر پیموده‌شده، تعداد گام‌ها و نرخ موفقیت به‌صورت عددی و گرافیکی گزارش شده است.

۱. مقدمه

پیمایش خودکار در محیط‌های ناشناخته یکی از مسائل پایه‌ای و مهم در روباتیک سیار است. در بسیاری از کاربردهای عملی، ربات به نقشهٔ کامل محیط دسترسی ندارد و باید تنها با اتکا به اطلاعات محلی و واکنش‌های ساده حرکت کند. یکی از ساده‌ترین و در عین حال شناخته‌شده‌ترین راهبردها در این زمینه، الگوریتم دیواردنبال‌کن مبتنی بر قانون دست راست یا دست چپ است.

در این راهبرد، ربات همواره یکی از دیوارهای اطراف خود را دنبال کرده و بر اساس اولویت‌های حرکتی مشخص، مسیر خود را تعیین می‌کند. این روش به‌ویژه برای محیط‌های «یک‌تکه» (Simply Connected) مناسب بوده و در بسیاری از موارد منجر به رسیدن به هدف می‌شود.

۲. هدف پروژه

هدف اصلی این پروژه عبارت است از:

- پیاده‌سازی الگوریتم دیواردنبال‌کن با قانون دست راست یا چپ؛
- بررسی عملکرد این الگوریتم در چند ماز با ساختارهای متفاوت؛
- نمایش مسیر حرکت ربات به‌صورت انیمیشنی؛
- گزارش نرخ موفقیت و تعداد گام‌های موردنیاز برای رسیدن به هدف.

۳. مدل‌سازی محیط

محیط به‌صورت یک ماتریس دودویی دوبعدی مدل شده است:

- مقدار **۱** نشان‌دهندهٔ دیوار
- مقدار ۰ نشان‌دهندهٔ فضای آزاد

هر ماز دارای دیوارهٔ بسته در اطراف خود است تا خروج ربات از محدودهٔ محیط جلوگیری شود. سه ماز مختلف با ابعاد و پیچیدگی متفاوت در نظر گرفته شده‌اند که نقاط شروع و هدف هر یک از آن‌ها از پیش مشخص شده است.

نمایش گرافیکی ماز با استفاده از دستور imagesc و نگاشت خاکستری انجام شده است؛ به‌طوری‌که سلول‌های دیوار و مسیر آزاد به‌صورت واضح از یکدیگر قابل تشخیص هستند.

۴. الگوریتم دیواردنبال‌کن (Wall-Following)

۱.۴ جهت‌های حرکتی

ربات تنها قادر به حرکت در چهار جهت اصلی است:

- شمال (N)
- شرق (E)
- جنوب (S)
- غرب (W)

جهت اولیهٔ حرکت ربات به سمت شرق در نظر گرفته شده است.

۲.۴ قانون دست راست / دست چپ

در هر گام، ربات جهت‌های حرکتی مجاز را بر اساس قانون انتخاب‌شده بررسی می‌کند:

- **قانون دست راست:**

  راست → مستقیم → چپ → عقب

- **قانون دست چپ:**

  چپ → مستقیم → راست → عقب

اولین جهت آزاد (بدون دیوار) انتخاب شده و ربات به آن سلول حرکت می‌کند.

۳.۴ جلوگیری از حلقه

برای جلوگیری از گیر افتادن ربات در مسیرهای تکراری، وضعیت هر سلول به‌همراه جهت قرارگیری ربات، یعنی ترکیب ردیف، ستون و جهت حرکت، ذخیره می‌شود. در صورت تکرار یک وضعیت مشابه، الگوریتم تشخیص می‌دهد که وارد حلقه شده و متوقف می‌شود.

۴.۴ معیار توقف

الگوریتم در یکی از حالات زیر متوقف می‌شود:

- رسیدن ربات به هدف؛
- عدم امکان حرکت؛
- عبور از حداکثر تعداد گام مجاز (maxSteps)؛
- ورود به یک حلقهٔ حرکتی.

۵. پیاده‌سازی و شبیه‌سازی

الگوریتم در محیط MATLAB پیاده‌سازی شده است. برای هر ماز:

- مسیر پیموده‌شده ذخیره می‌شود؛
- تعداد گام‌ها محاسبه می‌گردد؛
- وضعیت موفقیت یا شکست ثبت می‌شود؛
- در صورت فعال بودن انیمیشن، مسیر حرکت به‌صورت گام‌به‌گام نمایش داده می‌شود.

در نمایش گرافیکی:

- نقطهٔ شروع با **دایرهٔ سبز**
- هدف با **ستارهٔ سبز**
- مسیر طی‌شده با **خط آبی**
- موقعیت لحظه‌ای ربات با **دایرهٔ قرمز** نمایش داده شده است.

۱.۵ شرح بخش‌های اصلی کد

در ابتدای برنامه، پارامترهای اصلی شبیه‌سازی تعیین می‌شوند. متغیر handRule نوع قانون مورد استفاده را مشخص می‌کند که می‌تواند مقدار 'right' برای قانون دست راست یا مقدار 'left' برای قانون دست چپ داشته باشد. همچنین، متغیر maxSteps حداکثر تعداد گام‌های مجاز برای حرکت ربات را تعیین می‌کند و متغیر animateAllMazes برای فعال یا غیرفعال کردن نمایش انیمیشنی مسیر استفاده می‌شود.

```
handRule = 'right'; % 'right' or 'left'
maxSteps = 2000;
animateAllMazes = true;
```

در بخش بعدی، مازها به‌صورت ماتریس‌های دودویی تعریف شده‌اند. در این ماتریس‌ها مقدار ۱ نشان‌دهندهٔ دیوار و مقدار ۰ نشان‌دهندهٔ مسیر آزاد است. همچنین برای هر ماز، نقطهٔ شروع و نقطهٔ هدف به‌صورت مختصات سطری و ستونی مشخص شده است.

```
mazes{end+1} = [...];   % Binary matrix representation of the maze
starts{end+1} = [2, 2]; % Start cell coordinates [row, column]
goals{end+1} = [4, 10]; % Goal cell coordinates [row, column]
```

پس از تعریف مازها، برنامه با استفاده از یک حلقه، الگوریتم دیواردنبال‌کن را برای هر ماز اجرا می‌کند. در هر تکرار، تابع runWallFollow فراخوانی شده و خروجی آن شامل وضعیت رسیدن به هدف، تعداد گام‌ها و مسیر پیموده‌شده ذخیره می‌شود.

```
res = runWallFollow(M, start, goal, handRule, maxSteps);
results(m) = res;
```

تابع اصلی حل ماز، تابع runWallFollow است. در این تابع ابتدا جهت‌های حرکتی ربات تعریف می‌شوند:

```
dirs = [-1, 0; 0, 1; 1, 0; 0, -1]; % N,E,S,W
```

در این تعریف، چهار جهت حرکت به‌ترتیب شامل شمال، شرق، جنوب و غرب هستند. سپس بر اساس نوع قانون انتخاب‌شده، ترتیب بررسی جهت‌ها تعیین می‌شود. برای قانون دست راست، اولویت حرکت به‌صورت راست، مستقیم، چپ و عقب است؛ در حالی که برای قانون دست چپ، اولویت‌ها به‌صورت چپ، مستقیم، راست و عقب در نظر گرفته می‌شوند.

```
if strcmpi(handRule, 'right')
    orderFcn = @(h) [mod(h, 4) + 1, h, mod(h+2, 4) + 1, mod(h+1, 4) + 1]; % right,
straight, left, back
else
    orderFcn = @(h) [mod(h+2, 4) + 1, h, mod(h, 4) + 1, mod(h+1, 4) + 1]; % left,
straight, right, back
end
```

در هر گام، موقعیت فعلی ربات و جهت حرکت آن بررسی می‌شود. سپس جهت‌های ممکن بر اساس قانون انتخاب‌شده مرتب شده و اولین جهتی که آزاد باشد انتخاب می‌شود. اگر سلول بعدی در محدودهٔ ماز باشد و مقدار آن برابر ۰ باشد، ربات به آن خانه حرکت می‌کند.

```
for ci = 1:4
    d = candDirs(ci);
    np = pos + dirs(d, :);
    if inB(np) && M(np(1), np(2)) == 0
        pos = np;
        heading = d;
        path = [path; pos];
        moved = true;
        break;
    end
end
```

برای جلوگیری از تکرار بی‌نهایت، وضعیت‌های بازدیدشده در آرایهٔ visited ذخیره می‌شوند. هر وضعیت شامل موقعیت ربات و جهت حرکت آن است. اگر ربات دوباره به همان موقعیت با همان جهت برسد، الگوریتم تشخیص می‌دهد که وارد حلقه شده و متوقف می‌شود.

```
visited = false(size(M, 1), size(M, 2), 4);
```

در پایان تابع، اطلاعات نهایی شامل موفقیت یا عدم موفقیت، تعداد گام‌ها و مسیر پیموده‌شده در ساختار res ذخیره و به برنامهٔ اصلی بازگردانده می‌شود.

```
res.found = found;
res.steps = size(path, 1) - 1;
res.path = path;
```

همچنین برای نمایش صحیح ماز در محیط گرافیکی MATLAB، جهت محور عمودی با دستور زیر تنظیم شده است تا نمایش تصویر با ساختار ماتریسی ماز هماهنگ باشد:

```
set(gca, 'YDir', 'reverse');
```

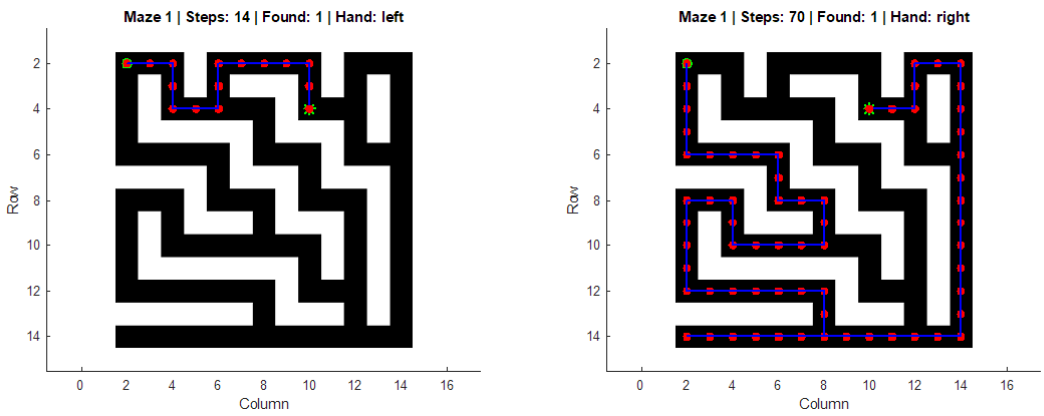
این تنظیم باعث می‌شود ردیف‌های ماتریس در نمایش گرافیکی به‌درستی از بالا به پایین نشان داده شوند و جهت‌های حرکتی ربات از نظر بصری با منطق الگوریتم سازگار باشند.

۶. نتایج شبیه‌سازی

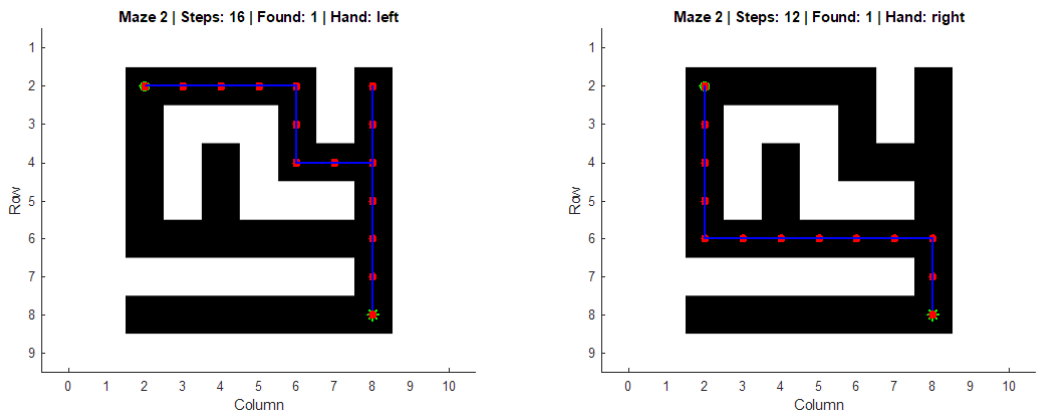
نتایج کمی حاصل از اجرای الگوریتم دیواردنبال‌کن گریدی بر روی مازهای مختلف در جدول ۱ ارائه شده است. همچنین، خروجی بصری و مسیر حرکت ربات در محیط‌های گریدی مختلف در شکل‌های (۱) تا (۳) نشان داده شده است. برای تهیهٔ جدول نتایج، برنامه یک‌بار با مقدار 'handRule = 'left و یک‌بار با مقدار 'handRule = 'right اجرا شده است.

جدول ۱ – خلاصه نتایج پیمایش مازها

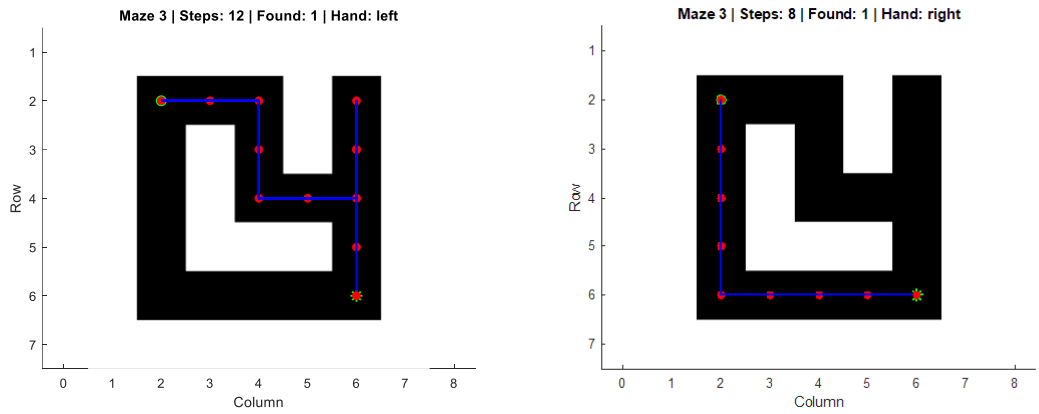
شماره ماز	موفقیت	تعداد گام قانون دست چپ	تعداد گام قانون دست راست
۱	✓	۱۴	۷۰
۲	✓	۱۶	۱۲
۳	✓	۱۲	۸



شکل ۱ – مسیر پیموده‌شده در ماز اول



شکل ۲ – مسیر پیموده‌شده در ماز دوم



شکل ۳ – مسیر پیموده‌شده در ماز سوم

۷. بحث و تحلیل

الگوریتم دیواردنبال‌کن به‌دلیل سادگی و ماهیت واکنشی خود، گزینه‌ای مناسب برای آموزش مفاهیم اولیهٔ ناوبری در روباتیک است. این روش نیاز به نقشهٔ سراسری ندارد و تنها با اطلاعات محلی تصمیم‌گیری

می‌کند. با این حال، تضمینی برای یافتن کوتاه‌ترین مسیر وجود ندارد و در برخی ساختارهای خاص ماز ممکن است به هدف نرسد.

۸. نتیجه‌گیری

در این پروژه، یک الگوریتم ساده و مؤثر برای پیمایش ماز پیاده‌سازی و ارزیابی شد. نتایج نشان می‌دهد که راهبرد دست راست/چپ در بسیاری از محیط‌های گریدی عملکرد قابل قبولی دارد و به‌ویژه برای کاربردهای آموزشی و شبیه‌سازی‌های مقدماتی در درس روباتیک بسیار مناسب است.

کلیدواژه‌ها

پیمایش ماز، دیواردنبال‌کن، قانون دست راست و چپ، ناوبری واکنشی، روباتیک سیار، MATLAB، شبیه‌سازی

پیوست الف – کد کامل MATLAB

```
clear;
clc;
close all;
%% Parameters
handRule = 'right'; % 'right' or 'left'
maxSteps = 2000;
animateAllMazes = true;
%% Define mazes, starts, goals
mazes = {};
starts = {};
goals = {};

mazes{end+1} = [; ...
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1; ...
    1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1; ...
    1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1; ...
    1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1; ...
    1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1; ...
    1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1; ...
    1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1; ...
    1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1; ...
    1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1; ...
    1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1; ...
    1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1; ...
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
starts{end+1} = [2, 2];
goals{end+1} = [4, 10];

mazes{end+1} = [; ...
    1, 1, 1, 1, 1, 1, 1, 1, 1; ...
    1, 0, 0, 0, 0, 0, 1, 0, 1; ...
    1, 0, 1, 1, 1, 0, 1, 0, 1; ...
    1, 0, 1, 0, 1, 0, 0, 0, 1; ...
```

```
    1, 0, 1, 0, 1, 1, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 0, 0, 1; ...
    1, 1, 1, 1, 1, 1, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 0, 0, 1; ...
    1, 1, 1, 1, 1, 1, 1, 1, 1];
starts{end+1} = [2, 2];
goals{end+1} = [8, 8];

mazes{end+1} = [; ...
    1, 1, 1, 1, 1, 1, 1; ...
    1, 0, 0, 0, 1, 0, 1; ...
    1, 0, 1, 0, 1, 0, 1; ...
    1, 0, 1, 0, 0, 0, 1; ...
    1, 0, 1, 1, 1, 0, 1; ...
    1, 0, 0, 0, 0, 0, 1; ...
    1, 1, 1, 1, 1, 1, 1];
starts{end+1} = [2, 2];
goals{end+1} = [6, 6];

nMazes = numel(mazes);
results = struct('found', false, 'steps', 0, 'path', []);
%% Run over mazes
success = 0;
totalSteps = 0;
for m = 1:nMazes
    M = mazes{m};
    start = starts{m};
    goal = goals{m};
    res = runWallFollow(M, start, goal, handRule, maxSteps);
    results(m) = res;
    if res.found
        success = success + 1;
        totalSteps = totalSteps + res.steps;
    end

    if animateAllMazes || m == 1
        figure('Color', 'w');
        hold on;
        axis equal;
        imagesc(M);
        colormap(gray);
        set(gca, 'YDir', 'reverse');
        plot(start(2), start(1), 'go', 'MarkerFaceColor', 'g', 'MarkerSize', 7,
'LineWidth', 1.2);
        plot(goal(2), goal(1), 'g*', 'MarkerSize', 9, 'LineWidth', 1.4);
        for i = 1:size(res.path, 1)
            plot(res.path(1:i, 2), res.path(1:i, 1), 'b-', 'LineWidth', 1.5);
            plot(res.path(i, 2), res.path(i, 1), 'ro', 'MarkerFaceColor', 'r',
'MarkerSize', 6);
            drawnow;
            pause(0.03);
        end
        title(sprintf('Maze %d | Steps: %d | Found: %d | Hand: %s', m, res.steps,
res.found, handRule));
        xlabel('Column');
        ylabel('Row');
    end
end
%% Report
```

```
fprintf('Hand rule: %s\n', handRule);
fprintf('Mazes tested: %d\n', nMazes);
fprintf('Successes: %d (%.1f%%)\n', success, 100*success/nMazes);
if success > 0
    fprintf('Mean steps (successful runs): %.1f\n', totalSteps/success);
end

function res = runWallFollow(M, start, goal, handRule, maxSteps)
dirs = [-1, 0; 0, 1; 1, 0; 0, -1]; % N,E,S,W
if strcmpi(handRule, 'right')
    orderFcn = @(h) [mod(h, 4) + 1, h, mod(h+2, 4) + 1, mod(h+1, 4) + 1]; % right,
straight, left, back
else
    orderFcn = @(h) [mod(h+2, 4) + 1, h, mod(h, 4) + 1, mod(h+1, 4) + 1]; % left,
straight, right, back
end
inB = @(p) p(1) >= 1 && p(1) <= size(M, 1) && p(2) >= 1 && p(2) <= size(M, 2);

pos = start;
heading = 2; % start heading East
path = pos;
visited = false(size(M, 1), size(M, 2), 4);
found = false;

for k = 1:maxSteps
    visited(pos(1), pos(2), heading) = true;
    candDirs = orderFcn(heading);
    moved = false;
    for ci = 1:4
        d = candDirs(ci);
        np = pos + dirs(d, :);
        if inB(np) && M(np(1), np(2)) == 0
            pos = np;
            heading = d;
            path = [path; pos];
            moved = true;
            break;
        end
    end
    if ~moved
        break;
    end
    if isequal(pos, goal)
        found = true;
        break;
    end
    if visited(pos(1), pos(2), heading)
        break;
    end
end
res.found = found;
res.steps = size(path, 1) - 1;
res.path = path;
end
```