רובוטים ניידים – תשפ"ד

תרגיל מסכם

I. תכנון מסלולים על סמך מידע מלא ומידע חסר

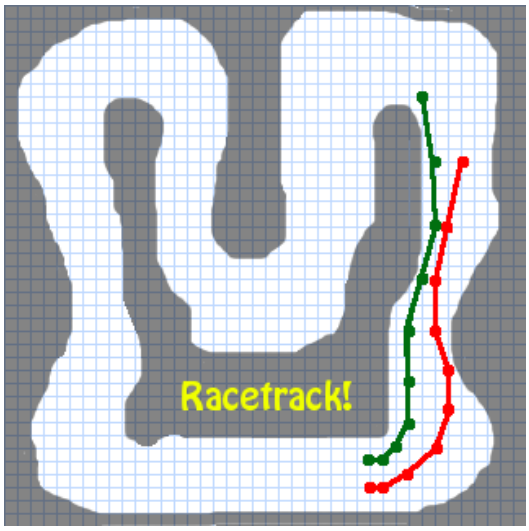
II. מציאת מיקום על פי מדידות מרחק

חלק I: תכנון מסלולים על סמך מידע מלא ומידע חסר מרוץ רובוטים במסלול סגור – חובה לכולם!

המוטיבציה הראשונית היא משחק עט ונייר לשני שחקנים או יותר בו מציירים מסלול מרוצים סגור ומעגלי. ראו למשל את זה

https://en.wikipedia.org/wiki/Racetrack_(game)

התנועה מותרת על פני המרחב החופשי. כל שחקן מיוצג על ידי נקודה בעלת מיקום התחלתי (בקו ההתחלה) ומהירות התחלתית 0 בשני הצירים. על פי תור, כל שחקן מחליט כיצד להתקדם. מותר לשנות את ערך המהירות (גודל הצעד) ביחידה אחת ובציר אחד לכל היותר. למשל, כדי להתחיל לנוע ממצב מנוחה (dx=0,dy=0) לכיוון ימין, יש להגדיל את dx ביחידה אחת (dx=1,dy=0). בצעד הבא אפשרויות התנועה הן: ללא שינוי מהירות (dx=1,dy=0), או אחד מהשינויים הללו (dx=0,dy=0), (dx=2,dy=0), (dx=1,dy=-1), (dx=1,dy=1). שחקן הנתקע בקיר חוזר לנקודת ההתחלה. הנה דוגמה למספר צעדים עבור שני שחקנים.



המנצח הוא זה שמגיע ראשון לקו הסיום (כלומר במספר צעדים מינימלי).

הנה גרסת online להנאת כולם

http://www.harmmade.com/vectorracer/#u-turn

התרגיל

התרגילים הקודמים התמקדו במשימות של עקיבה (שם יש מידע לוקאלי של חיישנים), אודומטריה ומיפוי (שימוש במידע לוקאלי ליצירת מידע גלובאלי), וניווט (שימוש במידע הגלובאלי). בתרגילים בניתם מערכת משולבת של חומרה ותוכנה – רובוט וקוד הרץ על גביו ועל מחשב חיצוני כדי לבצע את המשימות.

כעת נתמקד בתכנון מסלולים בתנאים שונים. בתרגיל זה תממשו ותבדקו אלגוריתמים לתכנון מסלולים בתנאי סימולציה.

הקלט:
א. מפה הממומשת על ידי מטריצה ובה שני ערכים אפשריים לתאים. ערכי קיר יסומנו ב 1, וערכי מרחב פנוי לתנועה ב 0.
ב. נקודת ההתחלה של הרובוט במפה: x0,y0.
ג. מיקום קו הסיום, שימומש כמערך ובו מיקומי התאים במטריצה השייכים לקו.

המשימה:

לתכנן ולממש מסלול תנועה כך שהרובוט יגיע במספר המינימלי של צעדים לסיום, ומבלי להתנגש בקירות.

לצורך **תכנון המסלול** נניח שהמפה כולה ידועה מראש. אתם תממשו שני אלגוריתמי תכנון מסלול.
לצורך **מעקב אחר המסלול** המתוכנן (מימוש התנועה עלפי המסלול), נניח כי המפה אינה ידועה. אתם תממשו שני אלגוריתמי בקרה למעקב אחר המסלול.

כדי להציג את הפתרון שלכם בצורה ויזואלית תשתמשו בקוד המחולק עם התרגיל. קוד זה מממש סביבת הרצה להצגת המפה והתקדמות הרובוט על פני המפה.

השלבים לפתרון (בסגול שלבים שאתם תבצעו, בחום שלבים שיבוצעו על ידי הקוד המחולק בתרגיל):
א. חישוב מסלול אופטימלי.
ב. תכנון ומימוש אלגוריתם בקרה לנסיעה במסלול משלב א.
ג. ביצוע בפועל של התנועה תוך איסוף מידע מחיישנים ושמירת המיקום, עד לסיום.
ד. דיווח על זמן ומסלול בפועל.

עליכם להשתמש בתשתית הקוד (MATLAB) המצורפת לתרגיל, ולהשלים את הפרטים החסרים על פי ההנחיות המפורטות. התוצר של התרגיל שלכם יהיה קוד מלא ובר הרצה שאותו תגישו. את כל ההסברים עליכם לצרף בתור תיעוד בקוד עצמו.

פרוט השלבים

א. חישוב מסלול אופטימלי
כאן יש להגדיר מה הכוונה באופטימלי. כזכור, משימת הרובוט היא להגיע ממקומו ההתחלתי לסיום במספר הצעדים הקטן ביותר ומבלי להתנגש בקירות, מכאן שמסלול אופטימלי צריך להיות מסלול שיאפשר את ביצוע המשימה. אבל בתרגיל לא ננסה לממש קריטריון אופטימליות כזה במלואו אלא נציע שתי שיטות לתכנון מסלול (קצר ביותר, ורחוק ביותר מן הקירות), ונבחן את ביצועי הרובוט בכל אחת מהן.

א1. שיטה ראשונה: חישוב מסלול קצר ביותר
לפניכם הנחיות מפורטות לביצוע שלב זה.

A. הצגת המפה (מספרי הסעיפים הכתובים באנגלית מופעים כך גם בקוד, בקובץ ex3_main.m)

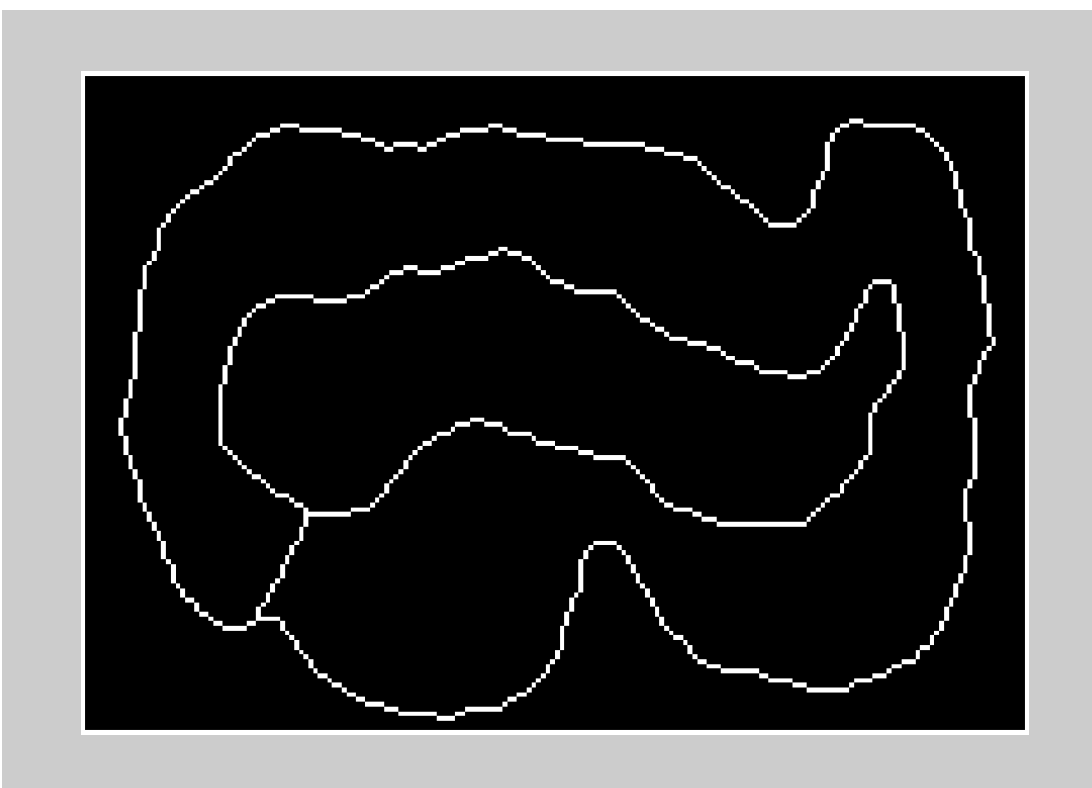
המפה מאוכסנת במטריצה map השמורה בקובץ 'map.mat' המצורף עם התרגיל.

הקוד הזה קורא ומציג את המפה:

```
%% constants for the speed of painting in slow mode
SLOW_1_FAST_0  = 1;   % 1 to view in slow mode, 0 to view in fast mode
PAINT_JUMP     = 200;
LINE_JUMP      = 10;

%% A. SHOW THE MAP

close all
figure(1); clf; colormap gray

temp_var = load('map.mat');
map      = temp_var.map;
[sy,sx]  = size(map);
clear temp_var

% show the map
imagesc(map); axis equal; axis tight; axis off; drawnow

set(gcf,'units','normalized','outerposition',[0 0 1 1]);

if SLOW_1_FAST_0, pause; end
```

ודאו שאתם מבינים את כל פרטי הקוד עד כה והריצו את הפונקציה הראשית (שתעצור אחרי הצגת המפה ותחכה להקשת מקש להמשך).

הנה המפה המוצגת:



שימו לב לפרטים:

- המפה מורכבת מפיקסלים של לבן ושחור בלבד (ההיקף האפור שייך לתצוגה של ה figure ונמצא מחוץ למפה). הפיקסלים הלבנים מתלכדים לכדי עקומות רציפות (ללא רווחים) ובעלות רוחב של פיקסל אחד בדיוק. עקומות אלה מייצגות קירות.

- בין הקירות המפותלים נמצא אזור הנסיעה העתידי. האזור תחום על ידי האי הפנימי ועל ידי ההיקף המפותל החיצוני לו.

- בחלק השמאלי התחתון של אזור הנסיעה מונח קיר קצר המחבר את האי וההיקף החיצוני. קיר זה נועד לחצוץ בין נקודת תחילת המרוץ (שתוגדר מימינו) לאזור סיום המרוץ (שיוגדר משמאלו).

- הקיר החיצוני ביותר נועד לאפשר צביעה של האזור החיצוני של המפה, כפי שיוצג בהמשך.

B. צביעה בהצפה של שני אזורים

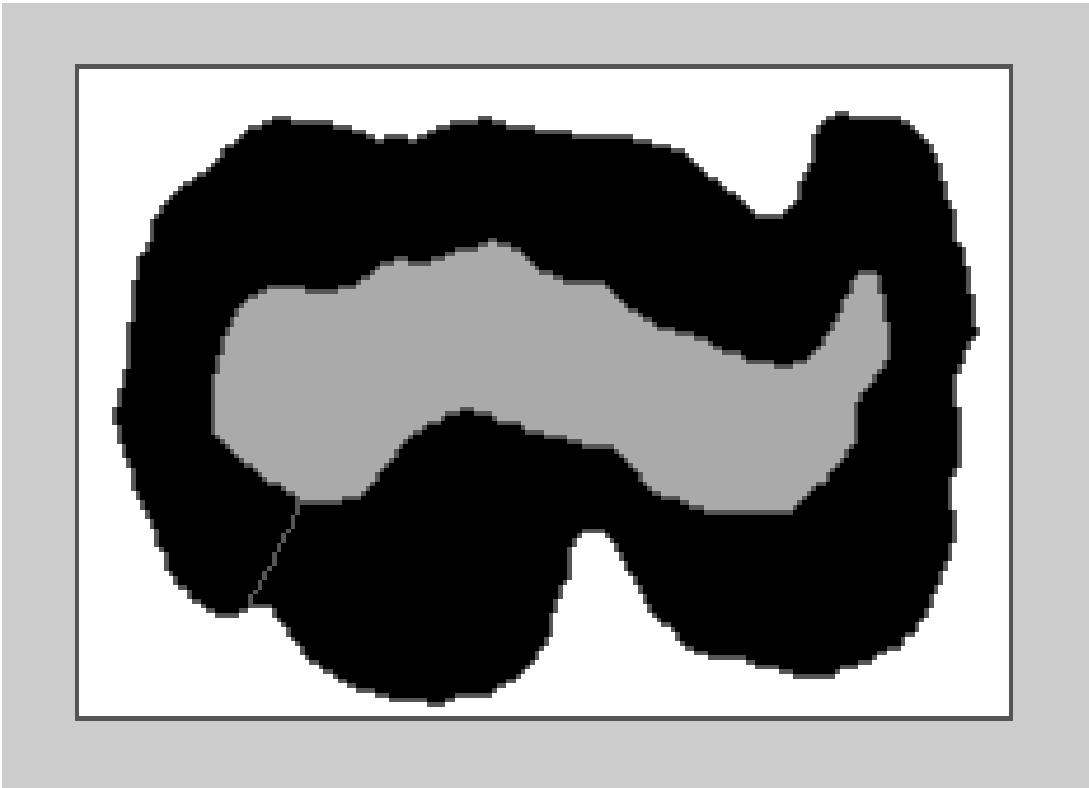
כעת נצבע בהצפה שני אזורים – האי הפנימי והאזור החיצוני. מטרת השלב הזה להכיר לכם את אפשרויות השימוש בפונקציית הצביעה בהצפה `floodfill_FIFO` (המוגדרת בקוד המצורף לתרגיל).
קוד זה מבצע את הצביעה:

```
%% B. paint two regions: the inner island, and the outer region
WALL_COLOR      = 1;
FREE_CELL_COLOR = 0;

% constants for the speed of painting in slow mode
jump            = PAINT_JUMP;        % x (jump) speed up in slow mode

% --- paint the island from the middle ----
% colors
ISLAND_COLOR    = WALL_COLOR + 1;
ColorToPaintOn = FREE_CELL_COLOR;    % paint on FREE_CELL_COLOR, stop on other colors
SeedColor       = ISLAND_COLOR;      % start painting using this color
dC              = 0;                 % do not change painting color while painting

m = floodfill_FIFO(double(map),round(sy/2),round(sx/2),...
   ColorToPaintOn,SeedColor,dC,SLOW_1_FAST_0,jump);

imagesc(m); axis equal; axis tight; axis off
if SLOW_1_FAST_0, pause; end

% --- paint the outer area from a point on the top left ----
OUTER_COLOR     = WALL_COLOR + 2;
SeedColor       = OUTER_COLOR;       % start painting with this color

m = floodfill_FIFO...
   (double(m),4,4,ColorToPaintOn,SeedColor,dC,SLOW_1_FAST_0,jump);

imagesc(m); axis equal; axis tight; axis off
if SLOW_1_FAST_0, pause; end

% change colors for next step
max_color = 2*(sx+sy); % should be maximal for the current track

% set all wall pixels to color max_color, so it will be the maximal value,
% thus not part of the possible path solution
m(m==WALL_COLOR) = max_color;

% set all island pixels to color max_color*0.9
m(m==ISLAND_COLOR) = round(max_color*0.9);

% set all outer area pixels to color max_color*0.9
m(m==OUTER_COLOR) = round(max_color*0.9);

% find the color of the track (should be FREE_CELL_COLOR == 0)
```

```
TRACK_COLOR = m(102,38);

% make sure all track pixels have FREE_CELL_COLOR (==0)
m(m==TRACK_COLOR) = FREE_CELL_COLOR;

imagesc(m); axis equal; axis tight; axis off
colormap jet

% save the map for later use
map1 = m;

if SLOW_1_FAST_0, pause; end
```

ודאו שאתם מבינים את הקוד, כולל מימוש הצביעה בהצפה, והפונקציות הממשות את מבנה הנתונים של התור.

הריצו את הקוד. הנה התוצאה המתקבלת לפני החלפת הצבעים של האזורים:



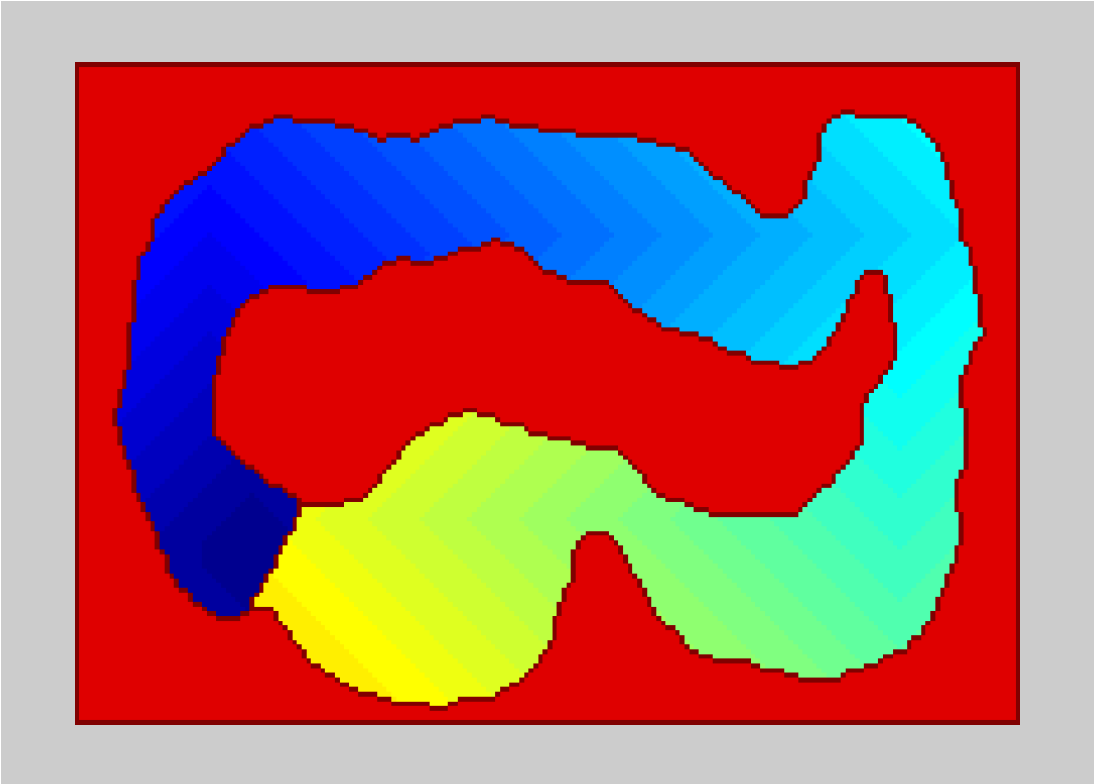
C. יצירת מפת צבעים כפונקציה של המרחק מן המטרה

הפונקציה `floodfill_FIFO` מאפשרת לשנות את הצבע במהלך הצביעה בגודל שינוי dC לכל קליפת צביעה. הקוד הזה מייצר מפת מרחק מקו המטרה אל כל השטח הפנוי של אזור המרוץ:

```
%% C. find shortest path, first stage
% --- paint the track area from the finish line ----

% the finish line (as points)
y_line = [102,103,104,105,106,107];
x_line = [ 38, 38, 38, 38, 38, 38];

ColorToPaintOn = FREE_CELL_COLOR;   % paint on FREE_CELL_COLOR, stop on other colors
SeedColor      = 1; % start painting with color 1
dC             = 1; % increase color with distance from the finish line

m = floodfill_FIFO...
   (double(m),y_line,x_line,ColorToPaintOn,SeedColor,dC,SLOW_1_FAST_0,jump);
```

```
imagesc(m); axis equal; axis tight; axis off;  colormap jet
if SLOW_1_FAST_0, pause; end
```

הריצו. התוצאה המתקבלת נראית כך:



D. מימוש פונקציה לתנועה במורד מפת הצבעים (להזכירכם – צבע זה מסמן משהו שאתם צריכים לעשות)

כפי שהסברנו בכיתה, מפת מרחקים זו מאפשרת כעת למצוא מסלולים קצרים ביותר מכל נקודת יעד אל המטרה.

ממשו פונקציה בשם gradient_descent, למציאת מסלול מנקודת יעד אל המטרה, על פי הממשק הבא

```
function [path_y, path_x, m] = gradient_descent(m, start_y, start_x, ...
      target_color, path_color, slow, jump, num_neighbors )
   % gradient_descent finds a continuous path on a color map, from a
   % starting point to a target color (value on the map).
   % this function assumes a that the color map represents increasing
   % distances from the target point(s), and that the starting point is
   % inside the painted region.
   %
   % input:
   %  m
   %    matrix, the color map.
   %
   %  start_y, start_x
   %    scalars, the starting point.
   %
   %  target_color
   %    scalar, the color (value) of the target point(s) in the map
   %
   %  path_color
   %    scalar, the color to paint the path on the map
   %
   %  slow,
   %    Boolean, if == 1, use presentation mode.
   %
   %  jump
   %    scalar, how many steps to jump in slow mode
   %
   %  num_neighbors
```

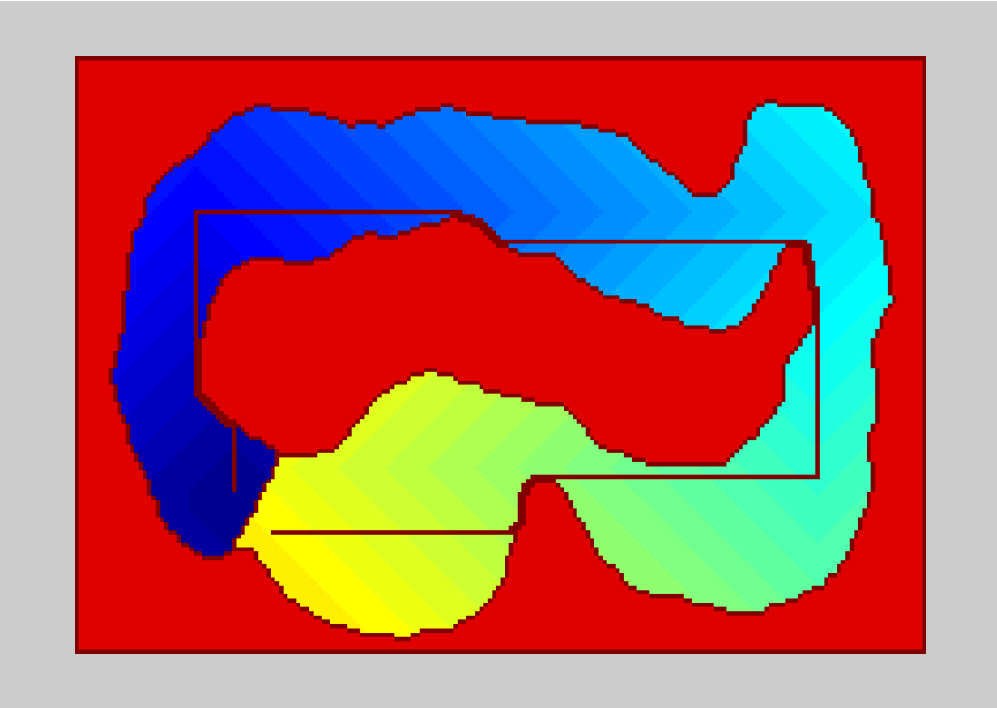
```
%    scalar, the number of neighbors (4 or 8) to check around a point in
%    the path creation process
%
% output:
%  path_y, path_x
%    1d arrays, the ordered positions of the found path.
%
%  m
%    matrix, the color map with the path painted on it.
```

על הפונקציה להתחיל מנקודת היעד (start_y, start_x) וממנה להתקדם במורד הערכים של המפה m (המוצגים כצבעים). בכל אחד מן המיקומים הפונקציה תשנה את ערך המפה במקום הנוכחי ל path_color, ותשמור את המיקום הנוכחי במערכים path_y, path_x.
מציאת המקום הבא ביחס למקום הנוכחי צריכה להתבסס על מציאת השכן בעל הערך הנמוך ביותר למיקום הנוכחי.
עליכם לממש חיפוש על פני 4 השכנים (משמאל, מימין, מעל ומתחת למיקום הנוכחי) ועל פני 8 שכנים (הכוללים גם את האלכסונים).

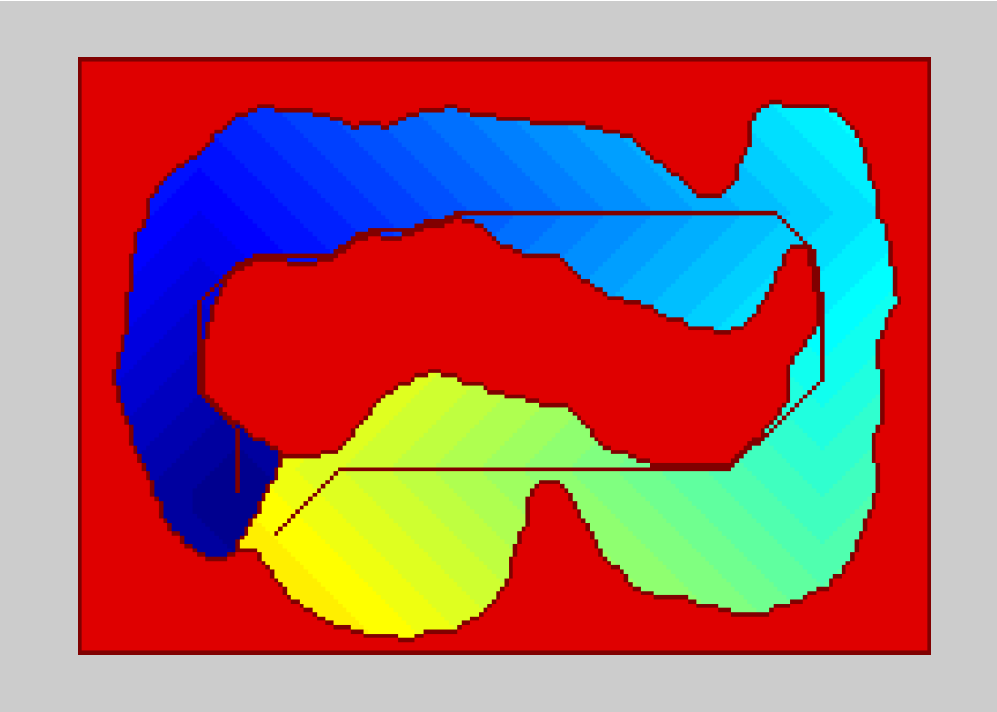
הפעילו את החיפוש מנקודת היעד (112,47), פעם עבור שכנות 4, ופעם עבור שכנות 8 כך:

```
%% D. find shortest path, second stage
% ----  gradient descent from start to finish line

% start point
start_y = 112;
start_x = 47;

PATH_COLOR   = max_color - 1;
target_color = SeedColor;

% 4 connected
num_neighbors = 4;

[path_y_4, path_x_4, m_4] = gradient_descent(m, start_y, start_x, ...
   target_color, PATH_COLOR, SLOW_1_FAST_0, LINE_JUMP, num_neighbors);

imagesc(m_4); axis equal; axis tight; axis off; drawnow
if SLOW_1_FAST_0, pause; end

% 8 connected
num_neighbors = 8;

[path_y_8, path_x_8, m_8] = gradient_descent(m, start_y, start_x, ...
   target_color, PATH_COLOR, SLOW_1_FAST_0, LINE_JUMP, num_neighbors);

imagesc(m_8); axis equal; axis tight; axis off; drawnow
if SLOW_1_FAST_0, pause; end
```

הנה תוצאות המתקבלות עבור חיפוש מבין 4 שכנים (התוצאות עשויות להיות שונות בהתאם לסדר המעבר על פני השכנים, כאשר לשנים או שלושה שכנים יש אותו ערך מינימלי):



והנה הנה תוצאות המתקבלות עבור חיפוש מבין 8 השכנים (גם כאן התוצאה תלויה בסדר המעבר על השכנים):



שימו לב שהפונקציה מחזירה גם את המפה ועליה מצויר המסלול (בתור המטריצה m) וגם את המסלול עצמו (בשני מערכים).

נחזור ונרשום את השלבים לפתרון המשימה כולה, כדי למקם את **שנעשה עד כה** בהקשר הכללי:

א. חישוב מסלול אופטימלי
א1. מסלול קצר ביותר (בוצע).
א2. מסלול רחוק ביותר מן הקירות.
ב. תכנון ומימוש אלגוריתם בקרה לנסיעה במסלול משלב א.
ג. ביצוע בפועל של התנועה תוך איסוף מידע מחיישנים ושמירת המיקום, עד לסיום (מבוצע על ידי הקוד המחולק).
ד. דיווח על זמן ומסלול בפועל (מבוצע על ידי הקוד המחולק).

א2. חישוב מסלול אופטימלי, שיטה שניה: חישוב מסלול רחוק ביותר מן הקירות

כדי למצוא את המסלול הרחוק ביותר מן הקירות יש לייצר צביעה מן ההיקפים של אזור הנסיעה: מן האי כלפי חוץ (בצבע אחד) ומן ההיקף החיצוני כלפי פנים (בצבע שני). כאשר שתי חזיתות הצביעה מתקדמות בו זמנית ובאותו קצב, הן נפגשות בקו האמצע של אזור הנסיעה, שהוא המסלול המרוחק ביותר מן הקירות.

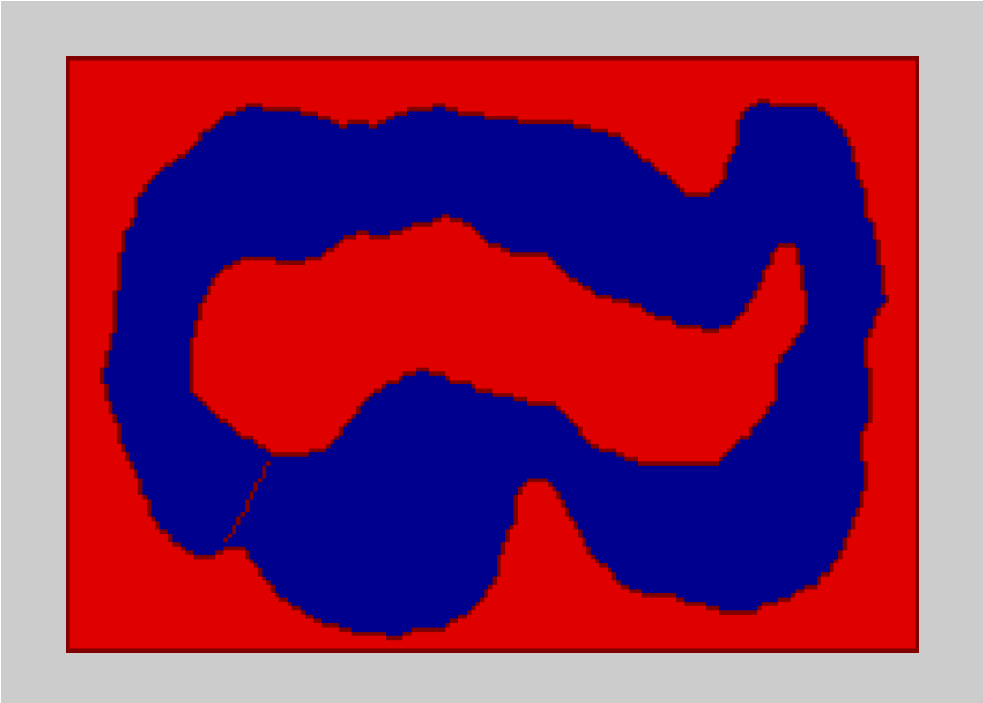
כדי להתחיל יש למצוא את מיקומי כל הפיקסלים השייכים לשני היקפים אלה, כמתואר בשלבים הבאים.

E. הכנת המפה לשלב הבא

השתמשו במפה המקורית אחרי שינויי הצבעים כפי שנשמרה במטריצה map1. נתקו את הקו המחבר את שני ההיקפים, כך:

```
%% E. use the map (prior to the find path stage) to find the path furthest from the walls

m1 = map1;

% a reminder -  the colors are:
% walls      ==  max_color
% island     ==  max_color*0.9
% outer area ==  max_color*0.9
% track      ==  FREE_CELL_COLOR == 0

% disconnect the walls from the connecting line:
m1(94,48)  = FREE_CELL_COLOR;
m1(95,48)  = FREE_CELL_COLOR;
m1(115,38) = FREE_CELL_COLOR;

imagesc(m1); axis equal; axis tight; axis off;  colormap jet
if SLOW_1_FAST_0, pause; end
```

הנה התוצאה



F. עקיבה אחר היקף שטח

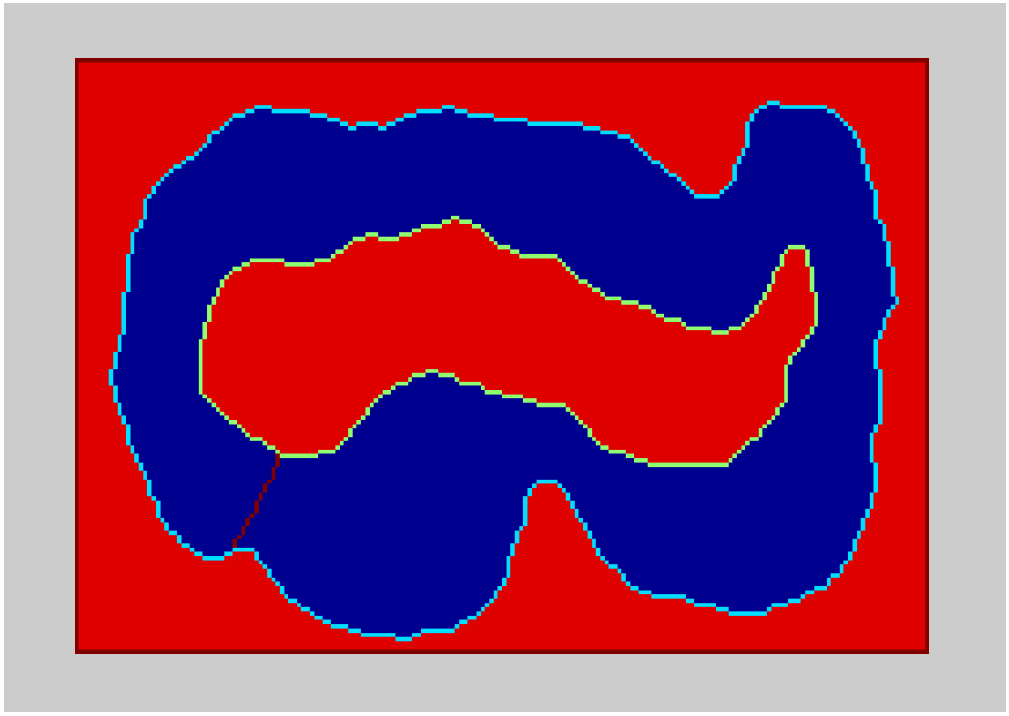
ממשו פונקציה בשם trace_line המקבלת את המפה ונקודת התחלה על קו, ומחזירה את הפיקסלים השייכים לקו, על פי הממשק הבא

```
function [line_y, line_x, m] = trace_line(m, initial_y,initial_x,...
      color_to_follow, path_color, slow, jump)
   % trace_line: follow the line, paint each point to a
   % different color, and store the points one by one until no more points
   % of the color COLOR_TO_FOLLOW exist in the current neighborhood.
   % assumes the curve to follow is composed of one-pixel width, continuous,
   % (no gaps) 8-connected curve.
   %
   % input:
   %  m
   %    matrix, the map with lines to trace.
   %
   %  initial_y,initial_x
   %    scalars, the starting point for the trace.
   %
   %  color_to_follow,
   %    scalar, the color of the line to trace
   %
   %  path_color,
   %    scalar, the color to paint the traced line
   %
   %  slow,
   %    Boolean, if == 1, use presentation mode.
   %
   %  jump
   %    scalar, how many steps to jump in slow mode
   %
   % output:
   %  line_y, line_x
   %    1d arrays, the ordered positions of the path to follow.
   %
   %  m
   %    matrix, the color map with the traced line painted on it.
```

הפעילו את הפונקציה פעמיים, לקבלת שני קווי ההיקף של אזור הנסיעה מנקודות התחלה ידועות, כך:

```
%% F. trace the outlines
% 1, find the pixels of the outer border of the island
% use trace_line procedure to trace the line, starting from a given
% initial point
initial_y = 94;
initial_x = 49;
COLOR_TO_FOLLOW    = max_color;
ISLAND_PATH_COLOR = round(max_color/2);

jump            = LINE_JUMP;          % x (jump) speed up for the slow mode

[island_line_y, island_line_x, m1] =...
   trace_line(m1, initial_y, initial_x,...
   COLOR_TO_FOLLOW, ISLAND_PATH_COLOR, SLOW_1_FAST_0, jump);

% 2, find the inner pixels of the border of the outer area
% use trace_line procedure to trace the line, starting from a given
% initial point
initial_y = 116;
initial_x = 38;
COLOR_TO_FOLLOW  = max_color;
OUTER_PATH_COLOR = round(max_color/3);

[outer_area_line_y, outer_area_line_x, m1] =...
   trace_line(m1, initial_y, initial_x,...
   COLOR_TO_FOLLOW, OUTER_PATH_COLOR,  SLOW_1_FAST_0, jump);

% 3, put back the removed points that disconnected the walls
m1(94,48)  = max_color;
m1(95,48)  = max_color;
m1(115,38) = max_color;

imagesc(m1); axis equal; axis tight; axis off
colormap jet

if SLOW_1_FAST_0, pause; end
```

התוצאה המתקבלת צריכה להראות כך:



G. צביעה בהצפה כלפי האזור הפנימי שבין שני ההיקפים

ממשו פונקציה בשם floodfill_FIFO_2colors המתבסס על floodfill_FIFO ומקבלת את נקודות ההתחלה של שני ההיקפים, את הצבע אליו יש לצבוע, את הצבעים בהם יש לצבוע כל חזית צביעה על פי הממשק הבא:

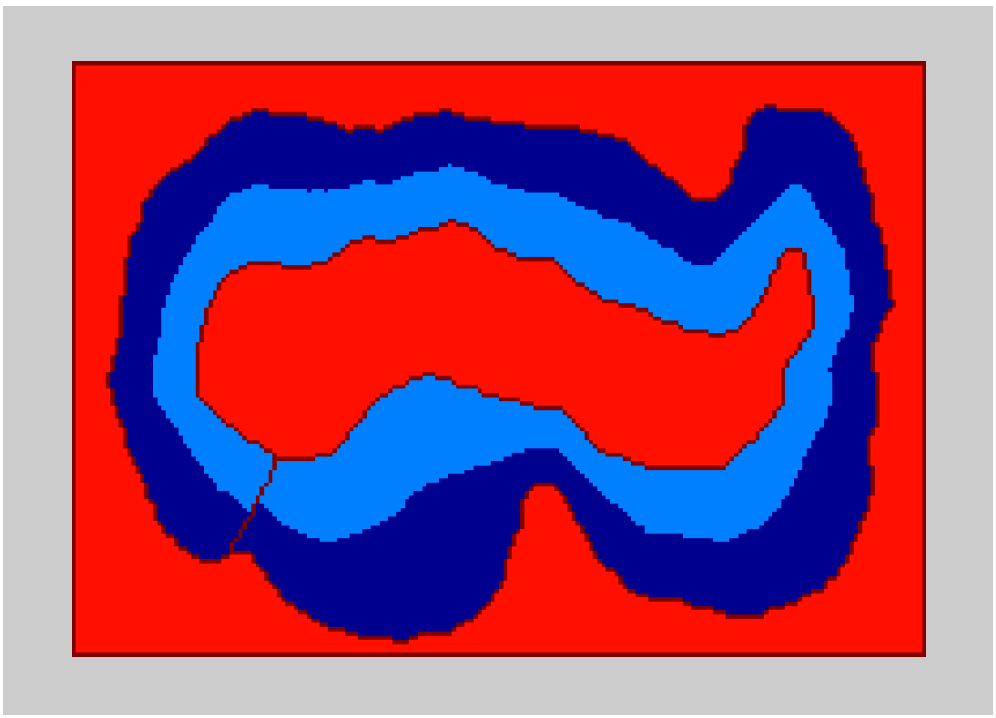
```
function m = floodfill_FIFO_2colors(m, y1, x1, y2, x2, ColorToPaintOn,...
      SeedColor1, SeedColor2, dC, slow, jump)
   % using a FIFO queue to implement flood fill (the breadth first search version).
   % paints an area in a matrix whose cells have the color ColorToPaintOn.
   % the painting starts from both from positions (y1,x1) and positions
   % (y2,x2). The flood progresses in two waves.
   % the first painting wave start with SeedColor1 and increases depending
   % on the distance from the (y1,x1) cells. Each step increases the color in
   % the amount dC.
   % the second painting wave start with SeedColor2 and increases depending
   % on the distance from the (y1,x1) cells. Each step increases the color in
   % the amount dC.
   % Using dC = 0 implements painting of all the pixels to SeedColor1 / SeedColor2.
   %
   % input:
   %  m
   %    matrix, the color map.
   %
   %  y1, x1, y2, x2
   %    1D arrays, the starting points 1 and 2 to paint from.
   %
   %  ColorToPaintOn
   %    scalar, only pixels with this color will be painted on.
   %
   %  SeedColor1, SeedColor1
   %    scalar2, the color to start painting with.
   %
   %  dC,
   %    scalar, the delta to add to the color while painting,
   %    use dC = 0 to paint all the pixels to SeedColor.
   %    use dC = 1 to paint in color values as distance from the
   %    starting points.
   %
   %  slow,
   %    Boolean, if == 1, use presentation mode.
   %
   %  jump
   %    scalar, how many steps to jump in slow mode.
   %
   % output:
   %  m
   %    matrix, the updated map after painting.
```

הפעילו את הצביעה מן ההיקפים פנימה כך:

```
    %% G. painting inwards
   % paint in ISLAND_PATH_COLOR outwards from the island into the free
   % cells, and at the same time, paint in OUTER_PATH_COLOR inwards from the outer area
   % into the free cells.

   jump           = PAINT_JUMP*3;                % x (jump) speed up for the slow mode
   dC             = 0;
   ColorToPaintOn = FREE_CELL_COLOR;

   % before flooding, the lines should be painted to FREE_CELL_COLOR,
   % so the floodfill algorithm will be able to start with them
   for i=1:numel(island_line_y),
      m1(island_line_y(i),island_line_x(i)) = FREE_CELL_COLOR;
   end
   for i=1:numel(outer_area_line_y),
```

```
    m1(outer_area_line_y(i),outer_area_line_x(i)) = FREE_CELL_COLOR;
end

imagesc(m1); axis equal; axis tight; axis off; drawnow

m2 = floodfill_FIFO_2colors(...
    m1, ...
    island_line_y,     island_line_x,     ...
    outer_area_line_y, outer_area_line_x, ...
    ColorToPaintOn,...
    ISLAND_PATH_COLOR,...
    OUTER_PATH_COLOR,...
    dC, SLOW_1_FAST_0, jump);

% and after flooding, paint the lines back to max_color
for i=1:numel(island_line_y),
    m2(island_line_y(i),island_line_x(i)) = max_color;
end
for i=1:numel(outer_area_line_y),
    m2(outer_area_line_y(i),outer_area_line_x(i)) = max_color;
end

imagesc(m2); axis equal; axis tight; axis off; drawnow
if SLOW_1_FAST_0, pause; end
```

התוצאה המתקבלת צריכה להראות כך:



H. מציאת הגבול בין הצבעים

כעת יש למצוא את כל הפיקסלים השייכים לגבול בין שני הצבעים, וזה יהיה המסלול המרוחק ביותר מן הקירות.

ממשו פונקציה בשם trace_border המקבלת את המפה, נקודת התחלה על קו ואת שני הצבעים, ומחזירה את הפיקסלים השייכים לקו, על פי הממשק הבא:

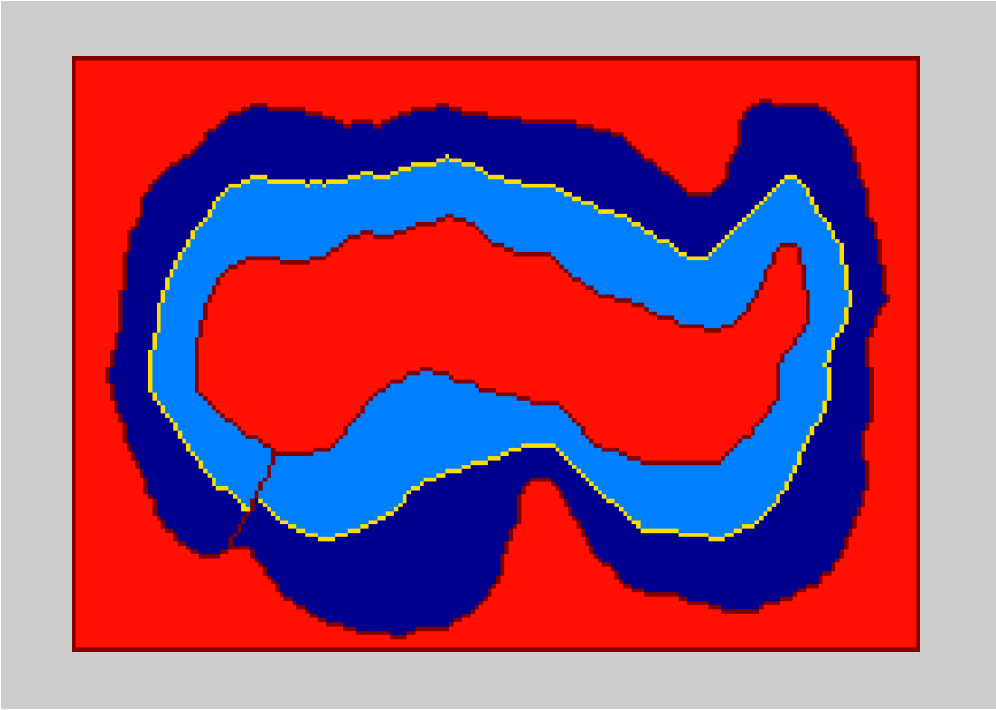
```
function [line_y, line_x, m] = trace_border(m, initial_y, initial_x,...
      initial_dy, initial_dx, color_1, color_2, border_path_color,  slow, jump)
   % trace_border finds the border line between two colors (color_1 and color_2),
   % and return this line as a set of positions. all border points are
   % points that are of one color and have neighbors of the other color.
   % the function starts from the initial point that must be the first point of
   % the border. the color at this location must be color_1 or color_2,
   % and it must have neighbors of both colors. the function then starts
```

```
% to trace the border in the initial direction (initial_dx, initial_dy),
% the direction to the next point along the border.
% the function continues to trace until no border points remain.
%
% input:
%  m
%    matrix, the map with the two colors.
%
%  initial_y,initial_x
%    scalars, the starting point for the trace.
%
%  initial_dy,initial_dx
%    scalars, the initial direction of the border
%    i.e. the increment from the first point of the border line to the
%    next point.
%
%  color_1, color_2,
%    scalars, the two colors. the border point are pixels in the map m
%    that are of one color and have neighbors of the other color.
%
%  border_path_color,
%    scalar, the color to paint the traced border line.
%
%  slow,
%    Boolean, if == 1, use presentation mode.
%
%  jump
%    scalar, how many steps to jump in slow mode
%
% output:
%  line_y, line_x
%    1d arrays, the ordered positions of the border between the two colors.
%
%  m
%    matrix, the color map with the traced border painted on it.
```

הפעילו את הפונקציה כך:

```
%% H. trace the border
% use trace_border to trace the border line between the two colors
% ISLAND_PATH_COLOR & OUTER_PATH_COLOR

initial_y  = 105;
initial_x  = 45;
initial_dy = 1;
initial_dx = 1;
jump       = LINE_JUMP;           % x (jump) speed up for the slow mode

COLOR_1           = ISLAND_PATH_COLOR;
COLOR_2           = OUTER_PATH_COLOR;
BORDER_PATH_COLOR = round(max_color/1.3);

[border_line_y, border_line_x, m3] =...
   trace_border(m2, initial_y, initial_x, initial_dy, initial_dx,...
   COLOR_1, COLOR_2, BORDER_PATH_COLOR,  SLOW_1_FAST_0, jump);

imagesc(m3); axis equal; axis tight; axis off; drawnow
if SLOW_1_FAST_0, pause; end
```

התוצאה המתקבלת צריכה להראות כך:



עד כה ביצעתם את השלבים של חישוב המסלול נחזור ונרשום את השלבים לפתרון המשימה כולה, כדי למקם את **שנעשה עד כה** בהקשר הכללי:

א. חישוב מסלול אופטימלי

א1. מסלול קצר ביותר.

א2. מסלול רחוק ביותר מן הקירות.

ב. תכנון ומימוש אלגוריתם בקרה לנסיעה במסלול משלב א.

ג. ביצוע בפועל של התנועה תוך איסוף מידע מחיישנים ושמירת המיקום, עד לסיום.

ד. דיווח על זמן ומסלול בפועל.

ב. תכנון ומימוש אלגוריתם בקרה לנסיעה במסלול משלב א.

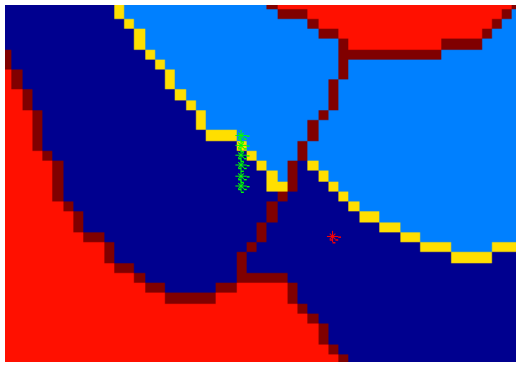
בשלבים הקודמים יצרתם שלושה מסלולים לבדיקה: מסלול קצר ביותר בשכנות 4, מסלול קצר ביותר בשכנות 8, ומסלול מרוחק ביותר מן הקירות.

כעת עליכם לממש שני אלגוריתמי בקרה. כל אלגוריתם בקרה יקבל מסלול, נקודות התחלה וסיום, ויוביל את הרובוט בצורה המהירה ביותר לאורך המסלול. **שימו לב שאלגוריתם הבקרה לא מקבל את המפה.**

בשני אלגוריתמי הבקרה הרובוט חייב להתחיל מנקודת ההתחלה, ולסיים באחת מהנקודות על קו הסיום, וזאת כאשר מהירותו אפס (כלומר לעצור בדיוק על קו הסיום).
התנועה עצמה ובדיקות ההתנגשויות ינוהלו על ידי הקוד המחולק עם התרגיל.

אלגוריתם הבקרה הראשון יניע את הרובוט אך ורק על המסלול, ללא סטיה ממנו. אלגוריתם זה יקבל את אחד משני המסלולים הראשונים (מסלול קצר ביותר בשכנות 4, מסלול קצר ביותר בשכנות 8), המתחילים בנקודת ההתחלה ומסתיימים בקו המטרה.

אלגוריתם הבקרה השני יניע את הרובוט על פי המסלול כאשר סטיות מן המסלול מותרות. אלגוריתם זה יקבל את המסלול השלישי (המסלול מרוחק ביותר מן הקירות). נשים לב שמסלול זה אינו מתחיל ומסתיים כדרוש (כפי שאפשר לראות כאן):



על כן אלגוריתם הבקרה השני יצטרך לנוע אל עבר המסלול בתחילת תנועת הרובוט, ולנוע אל המטרה, בסוף תנועת הרובוט. (באנלוגיה לעולם התעופה, בתחילה הרובוט צריך "להמריא" ולהגיע אל מסלול השיוט, אחר כך "לשייט" לאורכו, ולבסוף "לנחות" אל נקודת הסיום.)

שני אלגוריתמי הבקרה שולטים ברובוט באותו אופן - דרך שינוי המהירות בלבד, כלומר התאוצה. שינוי התאוצה מדמה שליטה במנועי רובוט המייצרים כוח (ומכאן תאוצה).

התאוצה המותרת היא בגודל 1 לכל כיוון. על כן וקטורי תאוצה חוקיים הם אחת מ 9 האפשרויות האלה:

[ay, ax] = [-1,-1], [-1, 0] , [-1, 1], [0, -1], [0,0], [0,1], [1,-1], [1,0], [1,1]

משוואות התנועה של הרובוט ובדיקות הממומשות בקוד המחולק

א. תנאי התחלה:

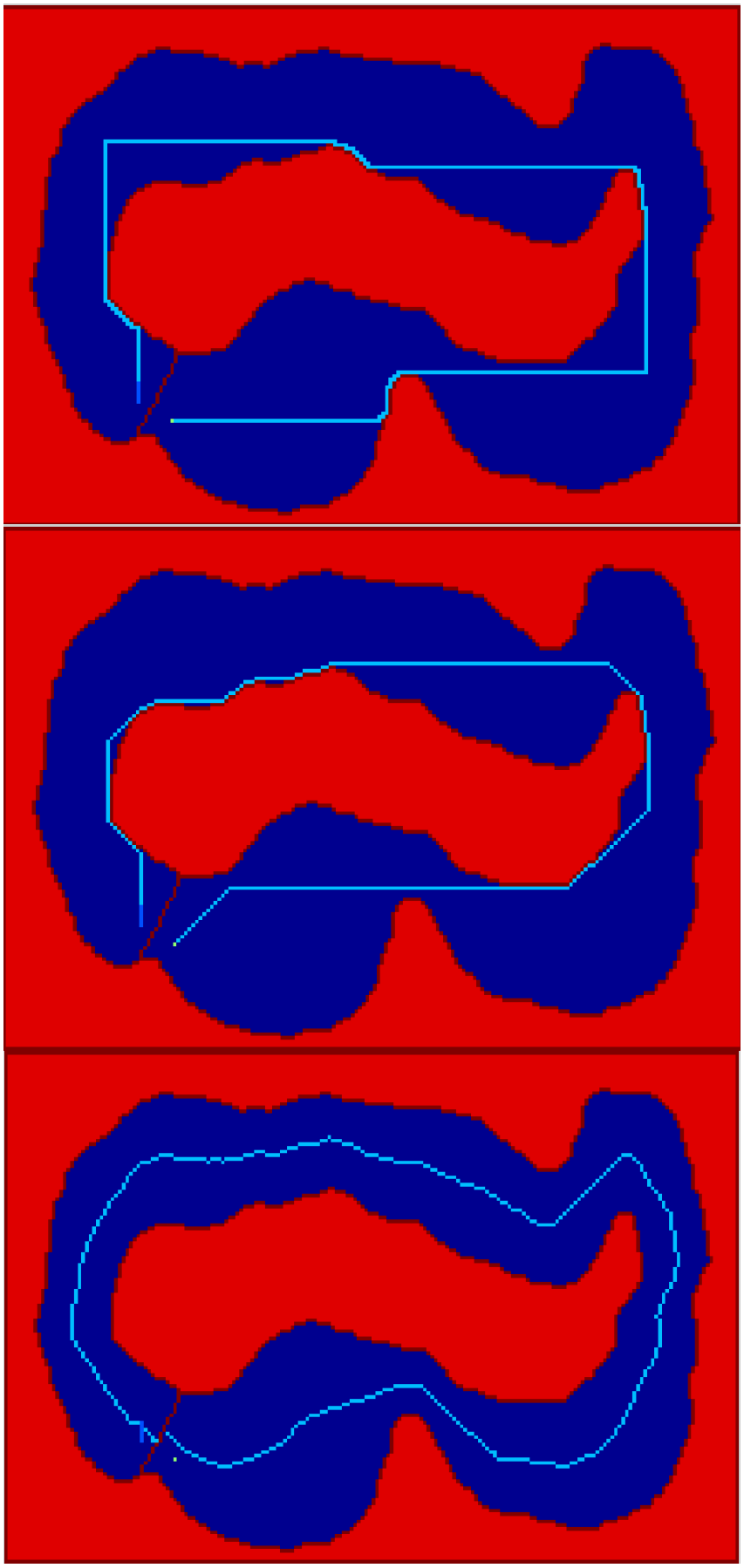
```
y(0)   = y_initial
x(0)   = x_initial

vy(0) = 0  % initial velocity in y direction
vx(0) = 0  % initial velocity in x direction

ay(0) = 0  % initial acceleration in y direction
ax(0) = 0  % initial acceleration in y direction
```

ב. התקדמות:

```
ay(t+1)  = from control algorithm
ax(t+1)  = from control algorithm

vy(t+1) = vy(t) + ay(t+1)
vx(t+1) = vx(t) + ax(t+1)

y(t+1) = y(t1) + vy(t+1)
x(t+1) = x(t1) + vx(t+1)
```

ג. בנוסף ממומשות בדיקות חוקיות התנועה: גודל התאוצה, התנגשויות, והגעה לקו הסיום.

הנה המפות המוגשות כקלט לשלב התנועה (כממומש בקוד המחולק עם התרגיל)



ו. מימוש אלגוריתמי בקרה והפעלתם

ממשו את אלגוריתמי הבקרה על פי הממשקים הבאים:

```
function [ay, ax, index_along_path] = control_algo1(...
      y, x, vy, vx, path_y, path_x, index_along_path, target_y, target_x)
   % control_algo1 finds the next desired location along the path, according
   % to the current location, the path, the already traversed path and the target.
   % this function assumes a continuous path from initial position to target.
   %
   % input:
   %  y, x
   %    scalars, the current position of the robot.
   %
   %  vy, vx
   %    scalars, the current velocity of the robot.
   %
   %  path_y, path_x
   %    1d arrays, the ordered positions of the path to follow.
   %
   %  index_along_path,
   %    scalar, the current index along the path where the robot is on
   %
   %  target_y, target_x,
   %    1d arrays, the positions of the target
   %
   % output:
   %  ay, ax,
   %    scalars, the accelerations for the next step.
   %
   %  index_along_path,
   %    scalar, the updated current index along the path.


function [ay, ax, index_along_path] = control_algo2(...
      y, x, vy, vx, path_y, path_x, index_along_path, target_y, target_x)
   % control_algo2 finds the next desired location along the path, according
   % to the current location, the path, the already traversed path and the target.
   % this function assumes a continuous path, but does not assume
   % the initial position or target are on the path.
   % It also assumes the path is furthest from the walls, and that the
   % distance to the walls is at least 4 (in diagonal).
   % control_algo2 finds the next desired location along the path, according
   % to the current location, the path, the already traversed path and the target.
   %
   % input:
   %  y, x
   %    scalars, the current position of the robot.
   %
   %  vy, vx
   %    scalars, the current velocity of the robot.
   %
   %  path_y, path_x
   %    1d arrays, the ordered positions of the path to follow.
   %
   %  index_along_path,
   %    scalar, the current index along the path where the robot is on
   %
   %  target_y, target_x,
   %    1d arrays, the positions of the target
   %
   % output:
   %  ay, ax,
   %    scalars, the accelerations for the next step.
   %
   %  index_along_path,
   %    scalar, the updated current index along the path.
```

הריצו את התוכנית מתחילתה ועד סופה.

התוכנית תציג את כל השלבים ותדווח על מספר הצעדים הנדרשים לפתרון עבור אלגוריתם הבקרה הראשון עם שני המסלולים הראשונים (שכנות 4 ושכנות 8) ועבור אלגוריתם הבקרה השני עם המסלול השלישי (המרוחק ביותר מן הקירות).

בונוסים:

א. ממשו והגישו כל רעיון שישפר את הביצועים (למשל שילוב של מסלול רחוק במידת מה מן הקירות ומסלול קצר ביותר).
הקריאה לתוספת תבוא בסוף הפונקציה ex3_main כך שהתוספת שלכם תורץ בסיום המשימה הסטנדרטית. אם נדרשות לכם פונקציות נוספות, מקמו אותן בסוף הקובץ ex3_main.

ב. בקוד נעשו כל מני פעולות הספציפיות למפה הזו, דוגמת מחיקת נקודות והחזרתן בהכנה לשלב הבא, או מתן ערך מפורש של מיקום (למשל בהתחלת מציאת הגבול בין שני הצבעים). ממשו קוד כללי יותר שאינו דורש התערבות ידנית וספציפית כזו.

הקבצים המצורפים לתרגיל (בתוך ספריה דחוסה ex3_mobile_robotics_code.zip):

ex3_main.m	- הפונקציה הראשית בה אתם צריכים להשלים פונקציות משנה.
floodfill_FIFO.m	- צביעה בהצפה בגל יחיד.
floodfill_FIFO_2colors.m	- צביעה בהצפה בשני גלים – אתם צריכים להשלים.
LQ...m	- חבילת קוד הכוללת 7 קבצים, למימוש תור ומחסנית.
map.mat	- המפה של המסלול.

קישורים מועילים להרחבת הידע בנושא תכנון תנועה

https://en.wikipedia.org/wiki/Racetrack_(game)

Efficient customisable dynamic motion planning for assistive robots in complex human environments
http://www.ict-acanto.eu/wp-content/uploads/2015/04/jaise_.pdf

Motion Planning for Car-like Robots
https://www.auto.tuwien.ac.at/bib/pdf_thesis/THESIS0006.pdf

Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching
https://e-archivo.uc3m.es/bitstream/handle/10016/18721/path_IJRA_2011.pdf;jsessionid=7A13C33773632E3688EB0D0AC7118F5C?sequence=3

A Guide to Heuristic-based Path Planning
https://www.cs.cmu.edu/~maxim/files/hsplanguide_icaps05ws.pdf

Red Blob Games
https://www.redblobgames.com/

Efficient Path Planning in Belief Space for Safe Navigation
http://www.ipb.uni-bonn.de/pdfs/schirmer17iros.pdf

Fast Marching Methods in Path Planning
https://jvgomez.github.io/files/pubs/fm2star.pdf

Robot Path Planning: Off-line and On-line Algorithms
http://cs.rkmvu.ac.in/~sghosh/public_html/nitrkl_igga/lec-rourkela-march2010.pdf

חלק II: מציאת מיקום על פי מדידות מרחק – רשות (בונוס) למגיש יחיד, חובה למגישים בצוות

בכחול שלבים לביצוע והגשה

1. מציאת מיקום של נקודות במרחב על פי סט של מדידות מרחק

נתונה רשת חיישנים המפוזרים במישור. כל חיישן יכול למדוד מרחק לכל חיישן אחר. בהינתן מטריצת המרחקים בין החיישנים מצא את הסידור המרחבי שלהם – כלומר קואורדינאטות שלהם ביחס למערכת צירים כלשהי.

נניח כי כולם נמצאים במקומות נבדלים, לכן מרחק של חיישן i מחיישן j ($i \neq j$) יהיה > 0.

1א. שרטטו פתרון לבעיה על פי השלבים הבאים (1,2 & 3). השרטוטים צריכים להתלוות לכל שלב ולהיעשות במדויק (שרטוט ממוחשב או סרגל ומחוגה).

1. נתחיל בפתרון גיאומטרי למקרה של שלשה חיישנים

א. נקבע את מיקום ראשית הצירים כמיקומו של חיישן 1. נסמן את חיישן 1 כנקודה.

ב. נשרטט קו אופקי בגודל d_{12}, הוא המרחק בין חיישן 1 לחיישן 2. נסמן בקצה הקו נקודה עבור חיישן 2. קו זה יהיה כיוון ציר ה X של מערכת הצירים.

ג. נשרטט מעגל ברדיוס d_{13} סביב נקודה 1. חיישן 3 יהיה על מעגל זה.

ד. נשרטט מעגל ברדיוס d_{23} סביב נקודה 2. חיישן 3 יהיה על מעגל זה.

ה. אם בין שני המעגלים נקודת השקה אזי מיקום חיישן 3 הוא בנקודת ההשקה (ואז החיישנים נמצאים על קו אחד).

ו. אם בין המעגלים יש שתי נקודות חיתוך, אזי חיישן 3 יהיה באחת משתיהן – אלה הם שני פתרונות מראה. נבחר בצורה שרירותית בפתרון העליון, וכך בעצם נקבע את כיוונו החיובי של ציר ה Y כפונה כלפי מעלה.

2. עבור יותר משלושה חיישנים: ראשית נבחר שלושה חיישנים ונפתור על פי 1. אחר כך, עבור כל חיישן נוסף j נחזור על סעיפים ג-ו בעזרת המרחקים d_{1j} , d_{2j}. כדי להכריע בדבר פתרון יחיד מבין השניים האפשריים (סעיף ו) נשתמש במרחק d_{3j}.

שרטטו את הפתרון המוצע כאן עבור חיישן רביעי.

3. למעשה בסעיף 2, אפשר להשתמש בכל שלשה מבין אלה **שכבר מצאנו להם מיקום** כדי למקם חיישן חדש.

מכאן ניתן להסיק כי כדי לפתור את בעיית המיקומים אין צורך במרחקים בין כל זוג חיישנים, אלא ניתן להסתפק במטריצת מרחקים דלילה בתנאים הבאים:

א. **כל חיישן מקושר לשלושה אחרים לפחות**. על כן בכל שורה או עמודה במטריצה צריכים להיות לפחות שלושה ערכים גדולים מ 0. (כמובן שבאלכסון יהיו אפסים, 0 שאינו על האלכסון יסמן חוסר במידע על המרחק).

ב. **חייבים להתחיל ביצירת משולש** ע"י מציאת שלושה חיישנים שלהם נתונים המרחקים ההדדיים.

ג. נשים לב כי אם שלושת המיקומים בעזרתם ננסה למקם חיישן חדש נמצאים על קו (collinear locations), אזי עדיין נותרת אי ודאות כשהקו מהווה ציר סימטריה לחיישן החדש. לכן עלינו לזהות מצב זה ולהתגבר עליו. הציעו דרך מפורשת לזיהוי מצב זה. הציעו פתרון. העזרו בשרטוט להצגת המצב ולהסבר הפתרון.

הערה: ניתן לתאר את החיישנים והמרחקים כגרף. הקודקודים יקבלו את מספר החיישנים ואילו הקשתות ייצגו את המרחקים בין החיישנים.

כשהתהליך שתיארנו מצליח, אזי הגרף הנבנה נקרא קשיח גלובלי (globally rigid).

1ב. כתבו פונקציית Matlab הפותרת את בעיית המיקומים בהינתן מטריצת מרחקים בין החיישנים, על פי ההצהרה

```
function P = CalcPositions(D)
   % CalcPositions calculates the relative positions 2D of points given a
   % distance matrix.
   % input:
   %       D    n x n    matrix.
   %            D(i,j) is the distance between point i to point j.
   %
   % output:
   %       P    n x 2    matrix.
   %            P(i,1:2) == (x,y)i, the position of point i
   %
   %            The function checks that D is legal:
   %            1. Dii == 0 for all i  (zero diagonal elements).
   %            2. Dij == Dji          (D is symmetric).
   %            3. Dij >=0 for all i,j (off diagonal zeros are allowed).
   %
   %            return P=[]  if D is illegal
   %
   %            return P=[0] if the calculation of positions is not possible
   %
   % example:
   %
   %   >> D = [0 1 1; 1 0 1.4142; 1 1.4142 0];
   %   >> P = CalcPositions(D);
   %
   %   P =
   %
   %        0     0
   %        1     0
   %        0     1

   % function code

end
```

1. על הפונקציה לבדוק את חוקיות המטריצה על פי הכללים והתנאים הבאים:
   1. המטריצה המתקבלת כקלט חייבת להיות סימטרית.
   2. כל האיברים במטריצה חייבים להיות 0 =<.
   3. באלכסון של המטריצה חייבים להיות אפסים - המייצגים את המרחק של כל חיישן מעצמו.
   4. 0 שאינו באלכסון הוא חוקי ומשמעותו מידע חסר על המרחק בין זוג חיישנים ולא מרחק אפס.
   5. ניתן להניח כי המרחקים הגדולים מ 0 נכונים (כלומר נמדדו כראוי על סט החיישנים).

   בכל מקרה של קלט לא חוקי יש להחזיר מטריצה ריקה ([]).

2. אם אין לפחות שלושה מרחקים עבור כל חיישן לא ניתן לחשב מיקומים ואזי יש להחזיר ערך 0.
3. יש למצוא שלשת חיישנים שנתונים מרחקיהם הדדיים ולהתחיל בהם. כיצד ניתן לעשות זאת ביעילות? אם אין שלשה כזו, יש להחזיר 0.
4. אם ניתן לחשב מיקומים, יש לחשבם על פי השיטה שהוצעה למעלה ולהחזיר מטריצה בה מיקומי החיישנים.
5. אם מגיעים למצב של הסתמכות על שלשה הנמצאת על קו ישר יש לנסות לפתור על פי הצעתכם מ 1א. אם לא ניתן לפתור יש להחזיר 0.

2. מציאת מסלולו של רובוט ומיקומם של עצמים במישור

רובוט נייד בעל חיישנים אודומטריים נע במישור במסלול מסוים. הרובוט יודע כמה התקדם בכל צעד אבל לא לאיזה כיוון. במהלך תנועתו מודד הרובוט מרחק אל מספר עצמים (ע"י קריאת האות שאותם עצמים משדרים). חיישן המרחק מדייק לחלוטין במדידת המרחק אך אינו מחזיר את הכיוון של העצמים.

נניח כמה הנחות מקלות:

- קיימים לפחות 3 עצמים.
- העצמים מרוחקים זה מזה לפחות מרחק d.
- מרחק הרובוט מן העצמים במהלך כל המסלול ניתן למדידה בדיוק מוחלט.
- הרובוט נע ועוצר חליפות והמדידות מתבצעות במהלך העצירות בלבד.
- המרחק בין עצירות עוקבות ידוע וקטן בהרבה מ d.
- קיימים לפחות שלושה מיקומים בהם עצר הרובוט ונערכו מדידות מרחק לעצמים.
- הרובוט מודד מרחק אל כל העצמים במהלך כל עצירה.

2א. <u>בעיית שיוך המדידות לעצמים</u>

קיימים n עצמים ו T צעדי זמן, על כן יש n*T מדידות.

בהינתן ערכי המדידות $m_{i,t}$, $i=1..n, t=1..T$, מצא את המיפוי בין המדידות $m_{i,t}$ לעצמים o_j, $j=1..n$.

הציעו שיטה מפורשת לפתרון בעיית השיוך. ניתן להשתמש במסלול ישר (אם כי כיוונו אינו ידוע).

האם יש מקרים בהם השיטה תכשל? הסבירו. אם עניתם כן הדגימו מקרה אחד לפחות בו השיטה כושלת.

2ב. <u>בעיית חישוב המסלול מן המדידות</u>

בהינתן המדידות ושיוך המדידות לעצמים, חשב את מיקום נקודות העצירה.

הציעו שיטה מפורשת לפתרון בעיית חישוב המסלול בהסתמך על הפתרון לשאלה 1.

2ג. האם עבור כל מסלול של הרובוט קיים פתרון יחיד (גרף קשיח)? חלקו את הסבר לשני שלבים:

1. האם תמיד נוכל לפתור את בעיית השיוך? הסבירו. אם לא הציגו דוגמה נגדית והסבירו.
2. בהינתן שיוך, האם כל מסלול יאפשר פתרון יחיד? הסבירו. אם לא הציגו דוגמה נגדית והסבירו.

עזרה כללית - הרצת קבצי MATLAB

בפניכם מגוון אפשרויות להריץ MATLAB:

1. הרצה ממחשבי מעבדה 4117 במכללה.
2. רכישת רישיון סטודנט ב 55 USD הכולל MATLAB, Simulink ועוד 10 חבילות
https://www.mathworks.com/products/matlab/student.html
3. שימוש ב MATLAB online (basic) הכפוף למגבלות מסוימות
https://www.mathworks.com/products/matlab-online/matlab-online-versions.html
4. הורדת גרסת ניסיון מלאה וחינמית ל 30 יום
https://www.mathworks.com/campaigns/products/trials.html
5. שימוש בתחליף חינמי ל MATLAB בשם Octave המסוגל להריץ קוד MATLAB בסיסי.
https://octave.org/
לא אמורה להיות בעיה עם הקוד המחולק בתרגיל זה, למעט אולי בדרכי ההצגה בצבע שיהיו שונות ממה שמודגם בתרגיל.
אם אתם בוחרים באופציה זו באחריותכם לבדוק את ההרצה ולדווח על שינויים או בעיות.
6. לא מומלץ אבל אפשרי -
להמיר את כל הקוד המחולק לפייתון או שפה אחרת הנוחה לכם. כאן באחריותכם לבדוק הכול, ואז לממש התרגיל.
7. אפשר לשלב אופציות:
להריץ על מחשבי המעבדה, להשוות להרצה על Octave ואז להחליט אם זה תחליף ראוי.

<u>הנחיות הגשה:</u>

1. הגשה בתיבת ההגשה של אתר הקורס עד יום רביעי 7.8.24 בשעת לילה.
2. **חלק I**:
יש להגיש שני קבצי MATLAB ex3_main.m , floodfill_FIFO_2colors.m הממומשים כנדרש בתרגיל.
יש לתעד את הקוד, ולהסביר רעיונות אלגוריתמיים והנחות בתוך הקוד.
3. **חלק II**:
יש להגיש קובץ Word או PDF ובו כל השרטוטים וההסברים המרכיבים את הפתרון.
יש לשמור על מספור הסעיפים כמו בתרגיל ולציין אותם במפורש במסמך הפתרון.
4. יש לרשום שם ות"ז של כל המגישים בראשית כל המסמכים.
5. מותר להשתמש במקורות חיצוניים (תיעוד, מאמרים, הדגמות, קוד). חובה לציין כל מקור.

בהצלחה!