# SCC361 AI Course Work#2

## Introduction:

## Marking Scheme

20% of the mark for the SCC361 module is based on the Course Work#2.

## Submission deadline

**The deadline for submission of your CW is 6PM, Friday, 15 December, 2023** (end of week 10) and should be submitted electronically on Moodle.

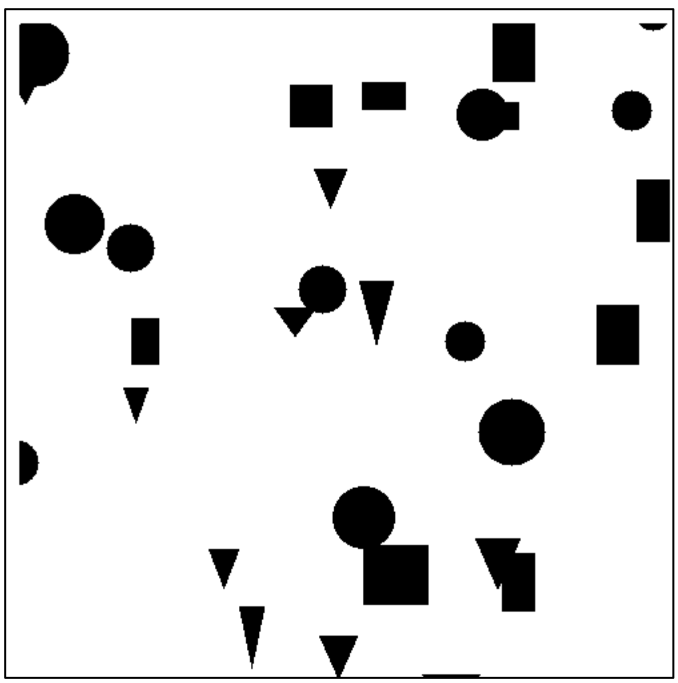
The implementation should be done in **MATLAB 2020a or later versions**. **The code should be run by pressing MATLAB RUN bottom (without requiring any changes)**. Save your code in a directory named "CW2_lastname_firstname" (substitute lastname and firstname with your actual names). Submit the file on Moodle.

In CW#2, your task is to apply a Genetic Algorithm (GA) to address a designated problem, and subsequently present the outcomes. This implementation should be carried out using MATLAB 2020a or a more recent version. The foundational concepts of GA were introduced and discussed during the week 5 and 6 lectures. Additionally, two lab sessions in week 6 and week 7 were dedicated to familiarizing you with GA. However, it's essential to note that this coursework involves applying GA to a distinct problem, different from those addressed in the lab sessions.

## Problem Definition

The task at hand involves implementing and enhancing a robot motion planning algorithm using Genetic Algorithms (GAs).

Assume that you have a robot with an overhead camera. The first step in motion planning is to create a representation of the robot's operating environment. This involve constructing a map that includes information about obstacles, terrain, and other relevant features. For the purposes of this CW, we simplify the process by assuming the existence of such a map, already provided as input, with dimensions of 500 pixels by 500 pixels. An illustrative example of such a map is depicted in the figure below.



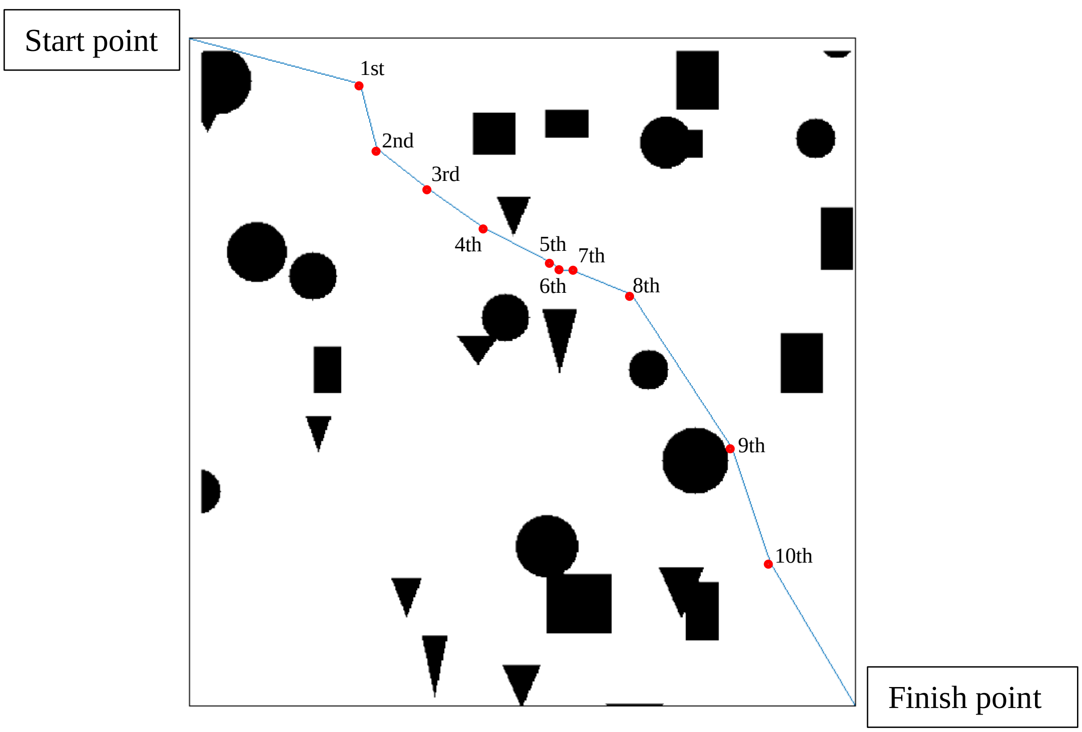
The next step is Localization. Localization in robot planning refers to the process of determining the robot's precise position and orientation within its operating environment. It is a crucial aspect of robotics that enables a robot to understand and update its location relative to a given coordinate system or map. Effective localization is essential for accurate navigation, as the robot needs to know where it is in order to plan and execute its movements successfully. For the purposes of this CW, it is presumed that both the starting point and the destination of the robot are provided as follows:

start = [1 1];

finish = [500 500];

The primary objective of the CW is thus limited to path planning for the robot.

Let a path be characterized by a fixed number of points (denoted as "noOfPointsInSolution") in the robotic map. As shown in the figure below for noOfPointsInSolution=10, the path is constructed by commencing from the start point [1 1] and connecting it to the first point through a straight line. Subsequently, each point is connected to the next in sequence by straight lines until the final point is linked to the finish point [500 500].




To keep things simple, we're treating the robot as a tiny point (or a pixel) instead of its actual size.

## Problem Solving using Genetic Algorithms

In formulating the problem as an optimization challenge for resolution by a Genetic Algorithm (GA), two key elements are essential: defining an objective function and specifying the variables of that function along with their bounds. The objective function, in this context, is the length of the path, where a shorter path is considered more favourable. A penalty should be imposed if any part of the path intersects with an obstacle, with the penalty proportional to the path length within the obstacle.

The optimization variables are the coordinates (x, y) of each of the fixed number of points along the path. The variable bounds are determined such that each point resides within the map. Specifically, the lower bound is set to 1, and the upper bound corresponds to the length or width of the map for the x and y axes (i.e., 500). Each of these points, arranged one after another, constitutes the genetic individual utilized in the optimization process.

Each point in the path marks a point of turn. The total number of points "noOfPointsInSolution" should be considered as a parameter and should be equal to the maximum number of turns a robot is expected to make in the robot map.

## How to Read and Generate Random Maps

In order to implement your code, you need to first read a map as a binary image as follows:

```
map=im2bw(imread('random_map.bmp'));
```

where 'random_map.bmp' is the name of an example map. A MATLAB script named "*Generate_Random_Map.m*" is provided to generate different (simple to complex) binary maps.

## Implement Genetic Algorithm

Your main task involves the implementation, using MATLAB, of a genetic algorithm designed to discover an optimal path through an evolutionary process. It is important to note that the built-in *ga* function in MATLAB is not permissible for use in this task. Instead, you are required to develop an approach akin to what was covered in the lab sessions during week 6 and week 7.

Your algorithm should make use of appropriate crossover and mutation operators. Numerous design decisions, including the implementation details of the algorithm and the selection of parameter values, will be required.

**Your code should output an image displaying the final best path using the provided code below:**

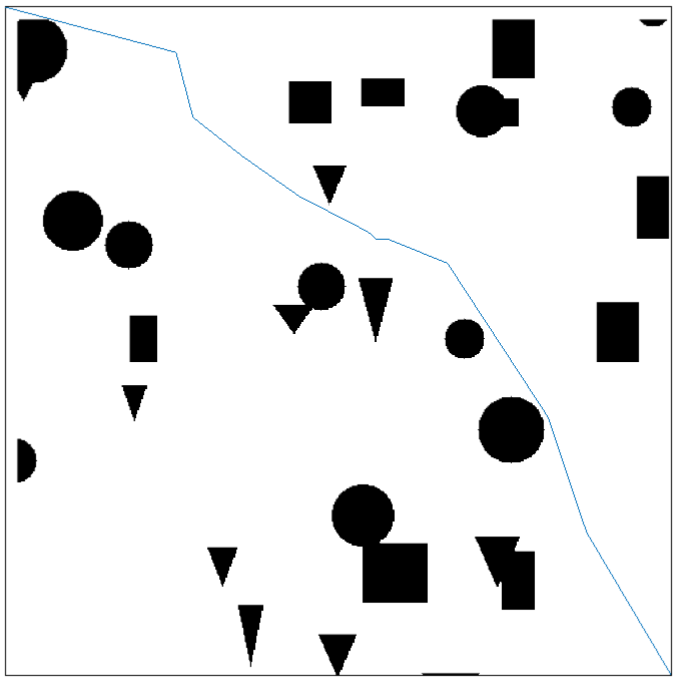
```
path = [start; [solution(1:2:end)'*size(map,1) solution(2:2:end)'*size(map,2)]; finish];

clf;

imshow(map);
```

```
rectangle('position',[1 1 size(map)-1],'edgecolor','k');
line(path(:,2),path(:,1));
```

solution is a row vector of size 2×noOfPointsInSolution. One possible output is shown below.



**Your code should include functionality to display both the execution time of the Genetic Algorithm (GA) and the total Euclidean distance of the optimal path.**

**Important Notes:**

1- You need to implement THREE selection methods including Roulette wheel selection (RWS), Tournament selection and Rank-based Selection. The code for RWS is available in Week 6's lab material on Moodle. Conduct some research to gain an understanding of how Rank-based selection operates, and subsequently, incorporate this knowledge into the implementation. Note that for the Tournament selection method, you need to consider its variants/parameters as variables.

2- You need to implement TWO appropriate cross-over operators. Note that k-point cross-over is counted as one method, and thus, changing k to 1, 2, 3 doesn't count as three cross-over operators.

3- You need to implement TWO appropriate mutation operators.

4- Your algorithm should provide optimal or near optimal solutions for any combination of selection/cross-over/mutation.

5- Upon initiation, the code should prompt the user to specify the types of selection, crossover, and mutation to be employed. For both crossover and mutation operators, the user should input a binary digit (0 or 1) to signify a specific type for each operator. For example, when selecting the crossover method, entering 0 signifies opting for crossover method 1, while entering 1 indicates the adoption of crossover method 2. Regarding selection, the user should input 0, 1, or 2 to indicate the chosen method.

6- The code should output an image displaying the final best path similar to the one shown in the figure above.

7- The code should be efficient and produces the required outputs. Try to use minimum number of loops in your code.

8- The maximum number of iterations/generations and the size of population should be as small as possible.

## Marks allocation for this course work:

**Structure of the code** **10**

The code should be well-structured and easy to understand. You should write your code in a single MATLAB script file named '*main.m*' in which you initialize your hyper-parameters and define your required functions and call the functions. The code should be run by pressing MATLAB **RUN** bottom without requiring any changes.

**Correctness of results** **50**

The code is expected to produce a solution that is close to optimal. The examiner will execute your code 10 times to assess how frequently the code generates an optimal or nearly optimal solution. The expectation is to achieve optimal results in 80% of the runs.

**Enough comments to understand the code** **10**

As you are not asked to submit a report for this CW, the code should be well-commented and provides enough details on methods used/implemented. I expect to see an informative comment next to each line of code.

**Time complexity of the algorithm (in all settings)** **30**

The code should minimize the use of loops and prioritize time efficiency. Utilize matrix operations in MATLAB to leverage computational advantages. The maximum number of iterations/generations and the population size should be minimized for optimal performance.

| | |
|---|---|
| **Total:** | **100** |