**BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY**



**Course No :** EEE-212

**Course Title :** Numerical Technique Laboratory

**Project Title: MAZE SOLVING WITH IMAGE PROCESSING USING MATLAB**

**Submitted by:**

**Name Student ID**

1. Rizwan Ahamed 201606074
2. Oishy Saha 201606075
3. Jarin Tasnim 201606090
4. Abir Kallayan Abedin 201606091

**Abstract:**

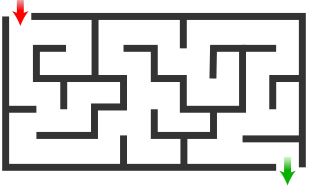
Mazes are a simplified form of navigation.  Developments used in efficient maze solving have helped to solve other navigation problems. There are many algorithms that can solve a maze very efficiently and quickly. But in our project, we did not use any of those algorithms, rather we solved mazes by image processing and using mathematical morphology. We chose perfect mazes on which we experimented our method. And we successfully solved any perfect maze of any dimension.

**Introduction:**

A maze is a path or collection of paths, typically from an entrance to a goal. The word is used to refer both to branching [tour puzzles](https://en.wikipedia.org/wiki/Tour_puzzle) through which the solver must find a route, and to simpler non-branching ("unicursal") patterns that lead unambiguously through a convoluted layout to a goal.(The term “labyrinth” is generally synonymous with “maze”, but can also connote specifically a unicursal pattern.)The pathways and walls in a maze are typically fixed, but puzzles in which the walls and paths can change are also categorized as mazes or tour puzzles.

**Maze Construction:**

Mazes have been built with walls and rooms, with hedges, turf, corn stalks, hay bales, books, paving stones of contrasting colors or designs, and bricks, or in fields of crops such as corn or, indeed, *maize.* Maize mazes can be very large; they are usually only kept for one growing season, so they can be different every year, and are promoted as seasonal tourist attractions. Indoors, mirror mazes are another form of maze, in which many of the apparent pathways are imaginary routes seen though multiple reflections in mirrors. Another type of maze consists of a set of rooms linked by doors (so a passageway is just another room in this definition).Players enter at one spot, and exit at another, or the idea may be to reach a certain spot in the maze. Mazes can also be printed on drawn on paper to be followed by a *pencil* or fingertip. Mazes can be built with snow.

  
 **Fig 1**:A small maze with one entrance and one exit

**Generating maze:**

Maze generation is the act of designing the layout of passages and walls within a maze. There are many different approaches to generating mazes, with various maze generation algorithms for building them, either by hand or automatically by computer.

There are two main mechanisms used to generate mazes. In “carving passages”, one marks out the network of available routes. In building a maze by “adding walls”, one lays out a set of obstructions within an open area. Most mazes drawn on paper are done by drawing the walls, with the spaces in between the marketing composing the passages.

**Solving mazes:**

Maze solving is the act of finding a route through the maze from the start to finish. Some maze solving methods are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas others are designed to be used by a person or computer program that can see the whole maze at once.

The mathematician *Leonard Euler* was one of the first to analyze plane mazes mathematically, and in doing so made the first significant contributions to the branch of mathematics known as *topology*.

Mazes containing no loops are known as “standard”, or “perfect” mazes ,and are equivalent to a *tree* in graph theory. Thus many maze solving algorithms are closely related to *graph theory*. Intuitively, if one pulled and stretched out the paths in the mazes in proper way, the result could be made to resemble a tree.

**Maze Solving Algorithms:**

1. **Wall follower:**

The wall follower, the best-known rule for traversing mazes, is also known as either the left-hand rule or the right-hand rule. If the maze is [simply connected](https://en.wikipedia.org/wiki/Simply_connected_space), that is, all its walls are connected together or to the maze's outer boundary, then by keeping one hand in contact with one wall of the maze the solver is guaranteed not to get lost and will reach a different exit if there is one; otherwise, he or she will return to the entrance having traversed every corridor next to that connected section of walls at least once.

Another perspective into why wall following works is topological. If the walls are connected, then they may be deformed into a loop or circle. Then wall following reduces to walking around a circle from start to finish. To further this idea, notice that by grouping together connected components of the maze walls, the boundaries between these are precisely the solutions, even if there is more than one solution (see figures on the right).

If the maze is not simply connected (i.e. if the start or endpoints are in the center of the structure surrounded by passage loops, or the pathways cross over and under each other and such parts of the solution path are surrounded by passage loops), this method will not reach the goal.

1. **Breadth-first Search:**

Breadth-first search (BFS) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for traversing or searching [tree](https://en.wikipedia.org/wiki/Tree_data_structure) or [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) data structures. It starts at the [tree root](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology) (or some arbitrary node of a graph, sometimes referred to as a 'search key’) and explores the neighbor nodes first, before moving to the next level neighbors. The [Breadth First Search](http://en.wikipedia.org/wiki/Breadth-first_search) algorithm is a common way to solve node-based path executions. Given a graph of nodes, BFS will basically collect all the possible paths that can be traveled, and visit them until the destination node is reached.

**3 . Dead End Filling:**

Dead-end filling is an algorithm for solving mazes that fills all dead ends, leaving only the correct ways unfilled. It can be used for solving mazes on paper or with a computer program, but it is not useful to a person inside an unknown maze since this method looks at the entire maze at once. The method is to 1) find all of the dead-ends in the maze, and then 2) "fill in" the path from each dead-end until the first junction is met. Note that some passages won't become parts of dead end passages until other dead ends are removed first.

Dead-end filling cannot accidentally "cut off" the start from the finish since each step of the process preserves the topology of the maze. Furthermore, the process won't stop "too soon" since the end result cannot contain any dead-ends. Thus if dead-end filling is done on a perfect maze (maze with no loops), then only the solution will remain. If it is done on a partially braid maze (maze with some loops), then every possible solution will remain but nothing more.

1. **Maze Routing Algorithm:**

The Maze-Routing algorithm [[10]](https://en.wikipedia.org/wiki/Maze_solving_algorithm#cite_note-10) is a low overhead method to find the way between any two locations of the maze. The algorithm is initially proposed for Chip Multiprocessors (CMPs) domain and guarantees to work for any grid-based maze. In addition to finding paths between two location of the grid (maze), the algorithm can detect when there is no path between the source and destination. Also, the algorithm is to be used by an inside traveler with no prior knowledge of the maze with fixed memory complexity regardless of the maze size; requiring 4 variables in total for finding the path and detecting the unreachable locations. Nevertheless, the algorithm is not to find the shortest-path.

1. **Shortest Path Algorithm:**

When a maze has multiple solutions, the solver may want to find the shortest path from start to finish. There are several algorithms to find shortest paths, most of them coming from [graph theory](https://en.wikipedia.org/wiki/Graph_theory). One possible algorithm finds the shortest path by implementing a [breadth-first search](https://en.wikipedia.org/wiki/Breadth-first_search), while another, the [A\* algorithm](https://en.wikipedia.org/wiki/A*_algorithm), uses a [heuristic](https://en.wikipedia.org/wiki/Heuristic) technique. The breadth-first search algorithm uses a [queue](https://en.wikipedia.org/wiki/Queue_(data_structure)) to visit cells in increasing distance order from the start until the finish is reached. Each visited cell needs to keep track of its distance from the start or which adjacent cell nearer to the start caused it to be added to the queue. When the finish location is found, follow the path of cells backwards to the start, which is the shortest path. The breadth-first search in its simplest form has its limitations, like finding the shortest path in weighted graphs.

**Image Processing:**

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

* Importing the image with optical scanner or by digital photography.
* Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
* Output is the last stage in which result can be altered image or report that is based on image analysis.

**Mathematical Morphology:**

Morphology is the study of shape. Mathematical morphology mostly deals with the mathematical theory of describing shapes using sets. In image processing, mathematical morphology is used to investigate the interaction between an image and a certain chosen structuring element using the basic operations of erosion and dilation. Mathematical morphology stands somewhat apart from traditional linear image processing, since the basic operations of morphology are non-linear in nature, and thus make use of a totally different type of algebra than the linear algebra. Mathematical Morphology is most commonly applied to [digital images](https://en.wikipedia.org/wiki/Digital_image), but it can be employed as well on [graphs](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)), [surface meshes](https://en.wikipedia.org/wiki/Polygon_mesh), [solids](https://en.wikipedia.org/wiki/Solid_geometry), and many other spatial structures. The basic morphological operators are [erosion](https://en.wikipedia.org/wiki/Erosion_(morphology)), [dilation](https://en.wikipedia.org/wiki/Dilation_(morphology)), [opening](https://en.wikipedia.org/wiki/Opening_(morphology)) and [closing](https://en.wikipedia.org/wiki/Closing_(morphology)). In this project, we have used dilation and erosion to get the required path for the solution of our given maze.

**Methodology:**

No searching or optimization methods (such as A\* or dynamic programming) are used in our project. We only used mathematical morphology and standard image processing methods. This method is only applicable for perfect maze. A perfect maze is defined as a maze which has one and only one path from any point in the maze to any other point. This means that the maze has no inaccessible sections, no circular paths, and no open areas.

First we read the image of the maze that we are intending to solve.

The Image Processing Toolbox™ software defines four basic types of images. These image types determine the way MATLAB® interprets data matrix elements as pixel intensity values. These four types are:

1. **Binary Image:** Logical array containing only 0s and 1s, interpreted as black and white, respectively.
2. **Grayscale Image:** Array of class uint8, uint16, int16, single, or double whose pixel values specify intensity values.

For single or double arrays, values range from [0, 1]. For uint8, values range from [0,255]. For uint16, values range from [0, 65535]. For int16, values range from [-32768, 32767]

1. **Indexed Image:** Array of class logical, uint8, uint16, single, or double whose pixel values are direct indices into a colormap. The colormap is an m-by-3 array of class double.
2. **RGB Image:** *m*-by-*n*-by-3 array of class uint8, uint16, single, or double whose pixel values specify intensity values. For single or double arrays, values range from [0, 1]. For uint8, values range from [0, 255]. For uint16, values range from [0, 65535].

If the maze image is grayscale image or indexed image then the number of colorband is one and we already have a monochrome image . But if the maze image is RGB image then number of colorband is 3 and we have to convert it to a monochrome image .For this we define the 1st array as red plane ,2nd array as green plane and 3rd plane as blue plane. Then we find the standard daviation of each plane and take the plane with highest standard daviation to get a better contrast. Now we have a monochrome image that we can use to solve the maze.

**Monochrome Image:**

****

**Fig 2:** An original monochrome image

After having the monochrome image we scale the image from 0 to 255 and define a thresold value. We convert pixels having value less than the thresold value to white and pixels having value greater than thresold value to black. As a result ,we get a binary image where the walls are white instead of balck.

****

**Fig 3:** A binary image with the walls of white colour

Having the binary image, we label image to find the discrete and separate walls. Here we use the function ‘bwlabel’ that finds the connected objects. After labeling the binary image we convert it to rgb image using colormap. For perfect maze ,we get two connected objects i.e two walls each with a different color. We can select either of the walls.



**Fig 4:** Segmentation to distingush between connected-wall and separated wall

****

**Fig 5:** Selection of the connected wall

Then we perform dilation on the wall we have selected. For dilation, we first determine a suitable dilation amount. Dilation adds pixels to the boundary of an object in an image.



**Fig 6:**Dilation process of one wall



**Fig 7:** Closing the holes of the dilated walls

After dilation we fill the image using ‘imfill’ function to get rid of holes if there are any. Then we eroded the image by the same amount as dilation. Erosion removes pixels on object boundaries.



**Fig 8:** Erosion process

The difference between the filled image and eroded image is the solution .For this we can either substract eroded image from filled image or we can set the eroded part of filled image to zero. Thus we get the final solution.



**Fig 9:** Obtained path got by subtraction process between dilation (+closing the holes) image and erosion image

Now we put this solution in green on top of original image. For this we first set green plane of the solution to 255 and other two planes to zero. Then we catenate this solution with the original image.



**Fig 10:** Path solution of the maze

**Application:**

There are some certain applications of maze solving. The prime application lies in navigation purposes .Developments in efficient maze solving have helped other navigation problems. Also maze solving is a quite interesting game. Maze game is one of the favorite ‘brain and puzzle’ game in the world. It is played by may be more than a billion people in the world, with all ages , with various level. We can solve the maze easily for a low level of difficulty, but it will takes a long time to solve for high level of difficulty.  If you take too long to solve the maze and you are so curious about the solution, we can solve the maze with image processing in very short time.

**Limitations:**

* Our algorithm is applicable for those mazes which have two walls only, that means one starting and one end way.
* We used image processing instead of any search algorithms, so the elapsed time of our code is greater than the other methods.

**Conclusion:**

The main aim of the project is to solve a maze with image processing .The image processing algorithm was based on image pre-processing , image-enhancement ,image segmentation ,morphological operation and image post processing . In the pre-processing section, the image was prepared to be solved. Image segmentation step is also performed quite satisfactorily throughout the appropriate algorithms. We mainly executed the path by dilation and erosion of the wall which is in the morphological operation. In the last post-processing process the solution path has been obtained with the initial image in the single image. Each step of the maze solving process has been performed with the help of built-in functions and after performing every steps each segment has been double checked to avoid any unexpected errors and erroneous output.