**COMPUTER PROGRAMMING - I**



**Semester Project Report**

**Submitted by:**

Name: **Muhammad Rafay**  Registration No: **2021-MC-39**

**Submitted to:**

**Shujat Ali**

**Department of Mechatronics & Control Engineering**

**University of Engineering & Technology, Lahore.**

December-2022

# INTRODUCTION:

## Computer programming is essentially the technique to communicate with the computer in the twenty-first century when computers form the basis of all technology on Earth. It has developed into a crucial skill that is used by more people than just computer scientists and engineers, but also other specialists. Robotics is one of the numerous fields in which programming is used. A smart robot's routine activities include path finding.

## Autonomous robot navigation:

An important task for the robot is autonomous navigation where the robot travels between a starting point and a target point without the need for human.

* **Uncertain Environment:**

There are certain obstacles that may exist within the navigation area. In order to reach the target point we have to maneuver these obstacles, and this is called as uncertain environment.

* **World Space:**

The world space refers to the physical space in which robots and the obstacles may exist and the free space is the subset of the world space that is not occupied by obstacles table

* **Feasible Path:**

A path between the starting and the target point that avoids collision with obstacles is said to be feasible and this path lies within three space.

## robot path planning:

We divide the area into a grid so that the robot can only travel in specific areas to simplify our challenge. Our plan is to construct chromosomes, each of which is made up of collection of numbers that correspond to specific grid positions. Then, using a few functions, we'll create a path linking those spots and assess the way's suitability based on the obstacles the robot must navigate, the number of turns it must make, and the length of the path it must travel to reach the goal.

## genetic algorithm:

A genetic algorithm is a heuristic search method used in artificial intelligence and computing. It is used for finding optimized solutions to search problems based on the theory of natural selection and evolutionary biology. Genetic algorithms are excellent for searching through large and complex data sets. They are considered capable of finding reasonable solutions to complex issues as they are highly capable of solving unconstrained and constrained optimization issues.

## selection of path:

We have two orientations column-wise and row-wise in which we can arrange the chromosomes on the grid there can only be one number for each column and in column-wise each single digit corresponds to a certain row number in its respective column. For the latter, the same reasoning is used. The robot advances in its column to match the row number of its subsequent column when using the column-first strategy. Like this, when using a row-first strategy, the robot begins by moving within its current row until it matches the column number of its subsequent row.

## assumptions:

* The robot only moves in the specified coordinates, and it will not move beyond the grid size.
* The starting point of the robot is (0, 0) whereas the ending point is (gridsize-1, gridsize-1) where gridsize is the size of the grid.

# autonomous robot navigation code:

## concepts used:

* Functional Programming
* Conditionals
* Loops
* Array Manipulation
* File Handling

## libraries:

* #include<stdio.h>
* #include<stdlib.h>
* #include<time.h>

## macros defined:

* #define Grid
* #define n\_Population
* #define Maximum\_iter
* #define inner\_size (grid – 2)
* #define w\_l , #define w\_t, #define w\_f
* Const char emptySpace = ‘ ‘, const char obstacle = ‘X’
* Const char start = ‘0’, const char goal = ‘0’, const char rStep = ‘.’

## global variables:

* Char sGrid[Grid][Grid]
* int random\_population[n\_Population][ inner\_size]
* int changed\_population[n\_Population][Grid]
* int turns\_population[n\_Population]
* int length\_population[n\_Population]
* int infeasible\_population[n\_Population]
* int Sol = 0
* int dir\_bit[n\_Population]
* int or\_bit[n\_Population]
* int minimum[3], maximum[3]
* float fit\_val[n\_Population]
* int iteration = 0

## user defined functions:

* Void RandomPopulation()
* Void turns\_storing(int changed\_population[n\_Population][Grid])
* Void path\_inf\_storing(int changed\_population[n\_Population][Grid])
* Void Changed Population(int random\_population[n\_Population][ inner\_size], int changed\_population n\_Population][Grid])
* Void Parents(int random\_population[n\_Population][ inner\_size], float fit\_val[n\_Population])
* Void Mutation(int random\_population[n\_Population][ inner\_size])
* Void CrossOver(int random\_population[n\_Population][ inner\_size])
* Void Display\_function(int grid\_Define[Grid][Grid], int dir\_bit[n\_Population], int or\_bit[n\_Population])
* Void MinimumMaximum()
* Void Fitness()
* Int Solution(float fit\_val[n\_Population]

# functions description:

## Void Randompopulation():

This function establishes random population based on size of grid. Each chromosome has a size two less than the grid. Apart from population, this function has a orientation and direction bit which is assigned to every chromosome. These bits are stored in 1D array where each chromosome corresponds to a chromosome in the population.

## Void Turns\_storing(int changed\_population [n\_Population][grid]):

In this function, turns of random population are stored by comparing first element index with 2nd . If they are not equal to 0 then turns will be added and stored in a array.

## Void path\_inf\_storing(int changed\_population [n\_Population][grid]):

This function calculates the path in 4 directions:

* Column wise and Column First (Direction bit = 0 & Orientation bit = 0)
* Column wise and row first (Direction bit = 1 & Orientation bit = 0)
* Row wise and column First (Direction bit = 0 & Orientation bit = 1)
* Row wise and Row first (Direction bit = 1 & Orientation bit = 1)

Path coordinates are calculated, and then infeasible steps are also calculated by comparing the condition of grid and decision bit as 1 and increment it.

## Void changedpopulation (int random\_population[n\_Population][size], int changed\_population [n\_Population][grid]):

In this function, the copy of random population is generated in which 1st column is assigned the value 0 and last value is assigned as Grid – 1.

## Void minimummaximum():

All the found turns, path length, and infeasible steps are then compared to their minimum and maximum array and assigned to it as a value.

## Void Fitness():

This function then find the actual fitness of turns, path length and infeasible steps by applying the formula which is:

A picture containing schematic

Description automatically generated

Actual fitness is calculated as:

Text, letter

Description automatically generated

## Void parents(int random\_population[n\_Population][size], float fit[n\_Population]):

This function uses basic bubble sort algorithm to rearrange the chromosomes based on their fitness. The chromosome having highest fitness come to the start of the array whereas the chromosome having fitness zero or less will come at the end of the array. Chromosomes are also bubble sorted on the basis of their Direction bit.

## Void crossover(int random\_population n\_Population][size]):

After sorting the chromosomes, the other half chromosomes of the array are  
replaced by the daughter chromosomes of the first half. For example, the first half of first  
chromosome is combined by the second half of the second chromosome whereas the second  
half of the first chromosome is combined by the first half of the second chromosome. In this  
way we get the two new offspring.

## Void mutation(int random\_population[n\_Population][size]):

Using the rand() function, the function modifies a gene's value on a chromosome at random. Additionally, it modifies a few of the direction and orientation bits.

## Void Display\_function(int grid\_Define[grid][grid], int dir\_bit[n\_Population], int or\_bit[n\_Population]):

This display function displays the final Direction, Orientation, Turns, Path, Fitness and Board of the whole Code.

## Int solution(float fit\_val[n\_Population]:

This function uses the infeasible steps and fitness of the population to determine if  
it represents the correct path or not. If the chromosome has zero infeasible  
steps and its fitness is greater than/equal to 300, the function returns the index of the population. If not, the function returns zero.

# COde testing:

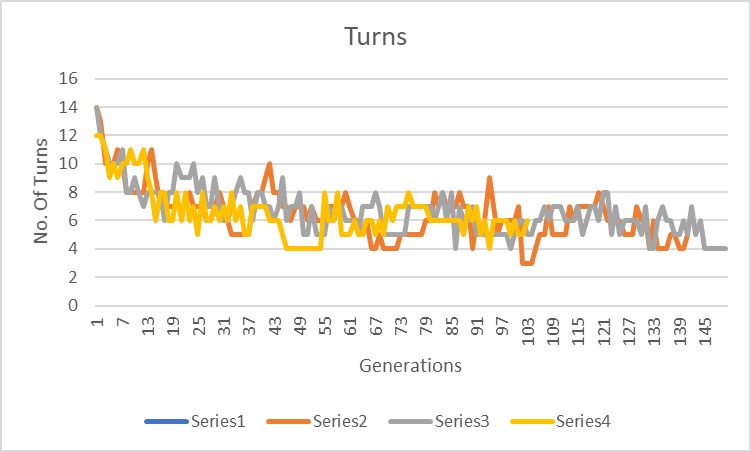
This program is tested on the grid size ranging from 8x8 to 20x20 with different Populations.

## Example of 16x16 grid:

int grid [16][16]={{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1},{0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1},{0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1},{0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1},{0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},{0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0},{0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},{1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},{1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0},{1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0},{0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},{0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0}};

# Graphs:

## Between Generations & Turns:



## between generations & path length:

Chart

Description automatically generated

## between generations & Infeasible steps:

Timeline

Description automatically generated with medium confidence

## between generations & fitness:

Graphical user interface

Description automatically generated with medium confidence

# output display:

