I. Problem description

Genetic Algorithm based Path Selection for Unmanned Autonomous Vehicles (UAV)

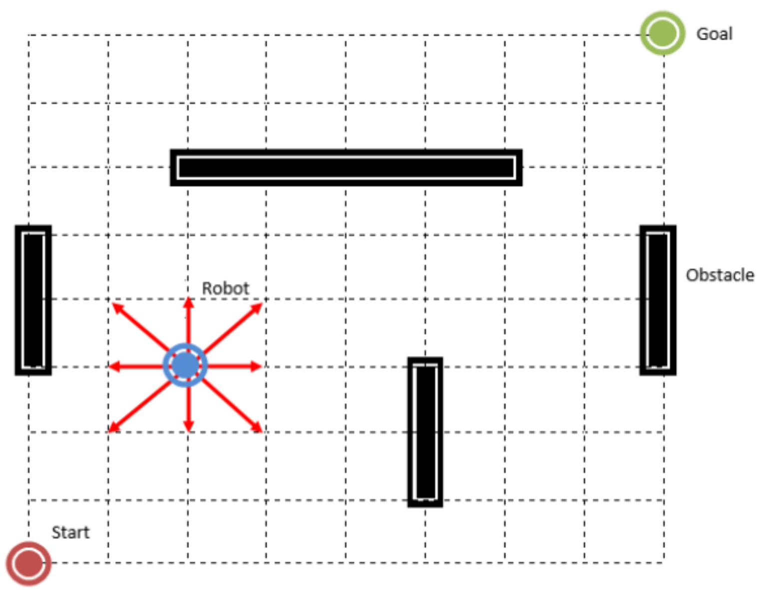
Motivation

Pursuit of a moving target by cooperative and autonomous UAVs pose an interesting topic of study with increasing adoption of drones for military and civilian applications. Further, recent interests shown by logistics companies like DHL, Amazon for small package delivery using UAVs demand for developing UAV capabilities for collision free and optimal path trajectory under clustered environments. The basics of solving such problems narrow down to selecting a shortest path from start to a goal (or target) in the presence of static or dynamic obstacles with a static or moving target involving single or multiple UAVs/ robots.

Specific problem statement:

With the above motivation, the current problem is framed by limiting the scope to two dimensional space first and keeping the goal and obstacles as static. Then, the problem is scaled up to three dimensions. The problem can be best described as per the indicative figure below:

Figure 1: General schematic representation of problem



Objective:

A robot has to select a shortest path to go from START to GOAL evading the obstacles.

Constraints:

There are 8 possible moves for the robot at a time as shown (if not at corner) and cannot go out of the square grid lines in a 2D environment. A robot cannot cross/jump over an obstacle. The robot will operate in constant speed (unit step). For three dimensional case, the possible steps increase to 26 other constraints still being the same.

II. Discussion on methodology:

There are possibilities for switching directions at every point of grid with uncertainty of obstacle in path and numerous non-minimal solutions available. Hence, it is best suited for solving using Genetic algorithm. GA offers flexibility to scale up further to three dimensional space/ multi robots/ dynamic obstacles/ dynamic targets.

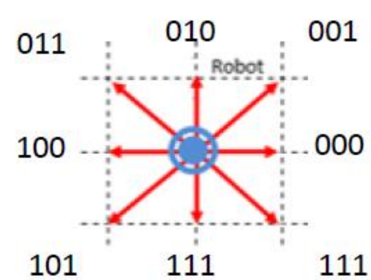
Problem set up and approach: Design variables coding:

As we can see the complete discrete nature of this problem, the design variables denote the step taken by the robot throughout the path travelled. Say for example, the first variable denotes the first step taken by the robot and the second variable denotes the second step and so on. It also gives the direction and distance travelled in that particular step as shown in table below.

Table 1:Decoding of design variable

x_i	Decoded	Distance	Direction
1	000	1	east
2	001	$\sqrt{2}$	northeast
3	010	1	north
4	011	$\sqrt{2}$	northwest
5	100	1	west
6	101	$\sqrt{2}$	southwest
7	110	1	south
8	111	$\sqrt{2}$	southeast

Figure 2: Decoding of design variable



Fitness function formulation:

Each cell in the grid is assigned a weighted penalty value (w_j) as 0 if no obstacle is present, 300 if obstacle is present and -0.9 if target is present. (In real time scenario, this can be obtained from GPS/ computer vision/another UAV).

The fitness function can be calculated by following formula:

$$f = \sum_{j=1}^{K} d_j (1 + w_j) \ldots (1)$$

where d_j = 1 if in vertical/ horizontal direction or $\sqrt{2}$ if it is in diagonal direction and K is the no. of grids travelled by robot. Here, let us restrict the value of K to 3n or 4n where 'n' is from dimension of square grid 'n*n' under study and assuming the optimal solution will be less than 3n/4n.

There are **two main considerations** in following the above approach:

1. If there are 30*3 bits long chromosome (30 – no. of steps with each step represented in 3 bits), the robot may reach the target in less than 1/3 of the steps in most cases and remaining part of chromosome may not converge at all.
2. Since there cannot be unique full length chromosome that specifies a shortest part solution, the convergence based on the bit string affinity does not work anymore.

For tackling above issues, the distance travelled after reaching the target is set to zero. In this way, the fitness function does not increase after reaching the target (since fitness function 'f' becomes zero when d_j becomes zero). As we see the minimum fitness function is a near approximation of the distance travelled in the shortest path, the convergence is set based on the consecutive generations having the same best fitness values. Here, the convergence criterion is set as 25 consecutive generations with best fitness values.

III. Discussion on results:

In the first part, the arena was limited to two dimensions and a 10*10 Grid was taken with different configurations of obstacles. In one such case, the following results were obtained.

Table 2 Results from solving a 2D environment as in figure 3b

	Population size*	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 1	420	0.001202	127	10.17107	11.07107	25
Run 2	420	0.001202	162	10.17107	11.07107	25
Run 3	420	0.001202	135	10.17107	11.07107	25

**as we could see, there are 35 design variables (35 steps) and cannot be printed. The figure shows the x* as plotted.*

GA population evolution:

Figure 3a: Generation 0

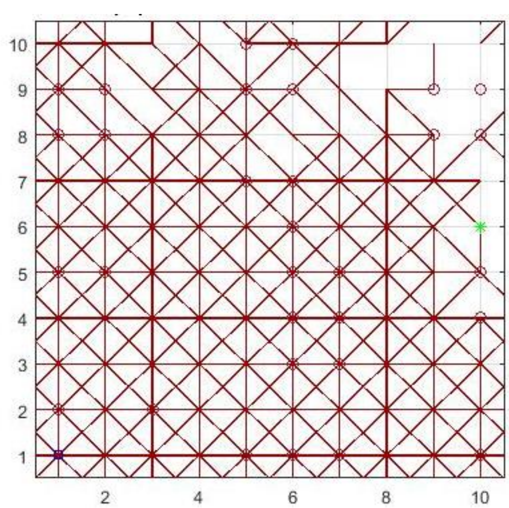
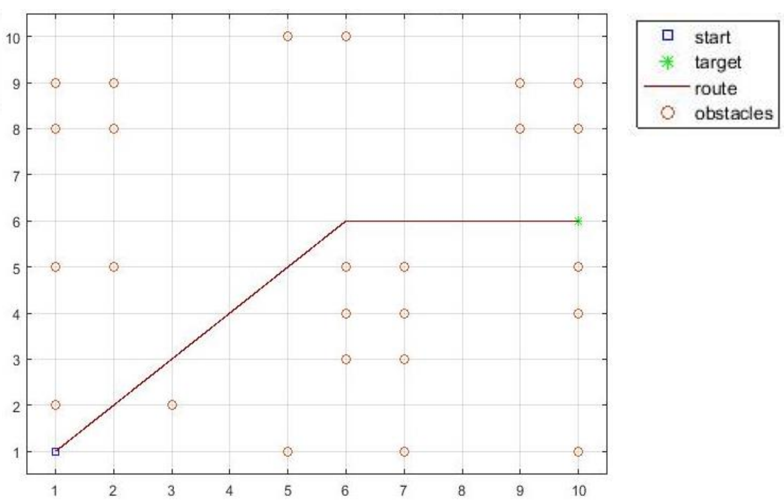


Figure 3b: Final Generation showing shortest path



As we could see, the GA has explored good enough at the start. There was no constraint violation (crossing obstacle) in the final answer and yet reached the target with shortest path. The obtained answer in this case could be intuitively acceptable as shortest.

Problem scaled up to three dimensions:

Besides the capability to handle discreteness of the problem, Genetic algorithm also offers scalability options without changing much of the code. In 3D, the number of discrete options for each step got increased to 26 as we could see in below figure with the robot being inside the centre of cube (refer fig. 4). The grid arena, visualization techniques had to be changed in order to scale the problem to 3 dimensions. The results are convincing for a number of trial cases as below (more cases attached in appendix). In this case, a 6*6*6 grid is chosen for study due to computational time limitations.

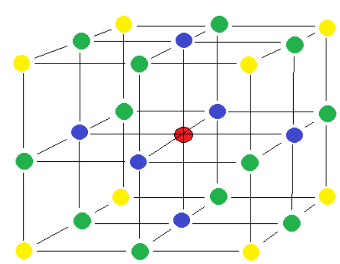
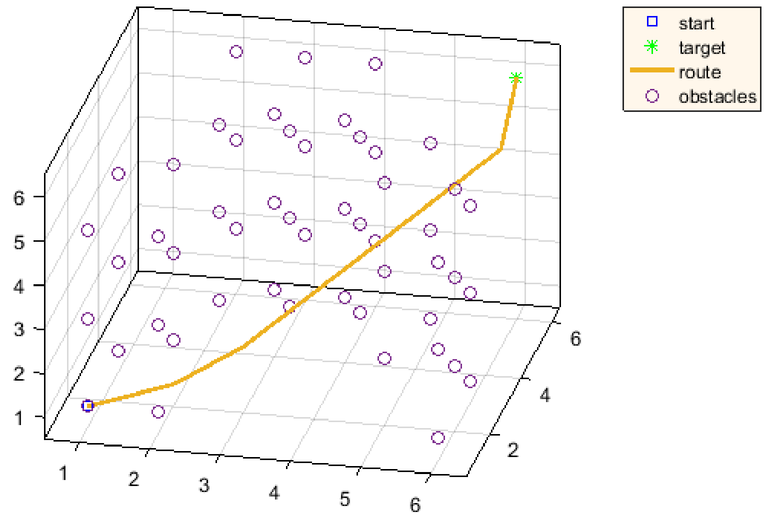


Figure 4: 3D representation showing possible movements. Points in Green, Yellow, Blue are at distances √2, √3,1 length units from robot respectively

Figure 5: Result showing shortest path converged in a 3D environment



In this case, the robot does not cross over any obstacles, reaches target in shortest path and could be intuitively agreed as the minimum.

Table 3: Results from solving a 3D environment as in figure 5

	Population size	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 1	800	0.000628	73	8.166001	9.438793	25
Run 2	800	0.000628	125	8.166001	9.438793	25
Run 3	800	0.000628	121	8.166001	9.438793	25

**as we could see, there are 40 design variables (40 steps) and cannot be printed. The figure shows the x* as plotted.*

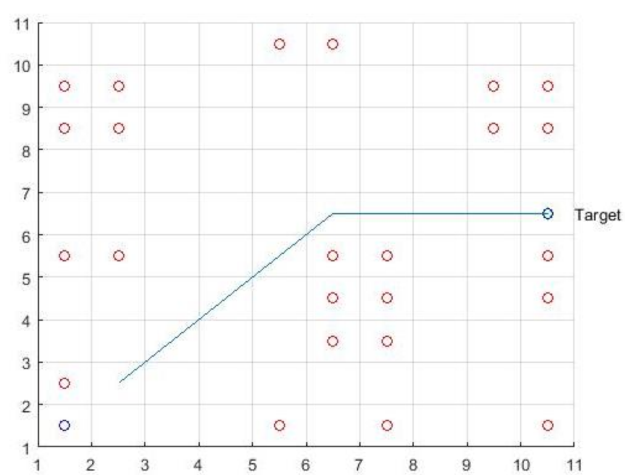
IV. A comparative study with A* algorithm for path planning

Although many path planning algorithms are available, the A* algorithm is said to be more widespread due to its performance and accuracy. It uses a heuristic based search on the grid with tendency to move towards the goal in shortest path avoiding obstacles.

A classic representation of the A* algorithm:
f'(n) = g(n) + h'(n)

where g(n) is the total distance it has taken to get from the starting position to the current location and h'(n) is the estimated distance from the current position to the goal destination/state.

Figure 6: Result from A algorithm on same 2D environment as figure 3b*



A heuristic function is used to create an estimate on the distance to goal from current position. This algorithm works in a way such that it does not search the **complete feasible space** for finding the global minimum. With relatively less number of functional evaluations, the algorithm is able to converge to the minimum value as we could see in the test case. Please note, the algorithm was available readily (like GA550.m) and it has been only customized to a sample problem for study. Comparatively results are shown in figure 6.

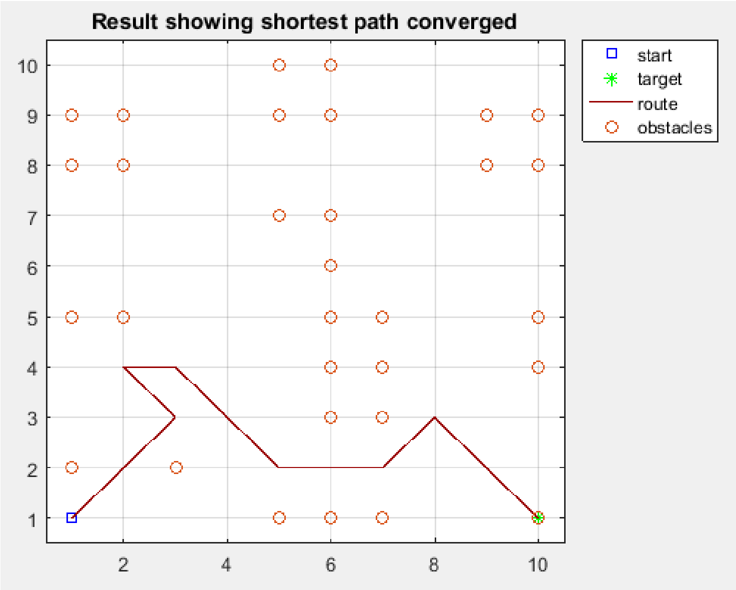
V. Important Conclusions

1. The major part of project dealt with formulating the best fitness function, finding the right weights/penalty, creating the look up table to get weights and importantly, on visualization of results. Once the 2D grid code was working, it was easier to scale up to 3D environment. The computational time for a dynamically changing environment was considerably high and could not be attempted given the time frame of this project. But provisions are in place to scale up immediately. This **factor of scalability** is the **biggest advantage** of the Genetic algorithm besides handling the **discreteness** of the problem.
2. When the environment became increasingly complex, there was a trend observed where the exploration was limited to first few generations due to the penalty for crossing obstacles. It resulted in situation where the GA was not able to converge to global solution. This was tackled to certain extent by using **elitism** where the **best individuals from previous generations** were preserved and passed on to succeeding generations. The population size was customized so that elitism did not limit the exploration in design space. We could see improvements in quality of results in certain cases with reduced number of generations. This was an important learning out of this project.
3. Clearly, the uniform crossover and fixed chromosome length with tournament selection had limitations on solving this problem on a complex environment. For example, in certain configurations of clustered obstacles, the algorithm converged during 1 out of 3 runs with other two solutions very near to the global minima. Thus, a need for exploring other options within GA was felt. **Variable chromosomal length, one point/ two point cross-over** methods have to be explored further in future to reduce the computational cost and improving quality of results. The literature on these GA options show promising trend to explore further but the same could not be experimented given the short timeframe of this project.
4. In the current problem set up, there was no heuristics involved in the selection of fittest individuals. After understanding the working of A*star algorithm, it was felt that a **heuristic** factor (intuitive judgement) could be included while formulating the fitness function which may result in faster convergence of Genetic algorithm compared to current set up. In order words, the formulation of fitness function was the key for successful implementation of Genetic algorithm. Although the number of iterations, functional evaluations, set up etc. could not be directly compared as "Genetic algorithm vs A* algorithm", the computational time comparison on Matlab showed considerably **lesser time taken by A* algorithm** for same problem. This is in line with literature and motivates to learn more problem specific-tailored algorithms.

In all cases solved by GA, we could see no constraint violation, acceptable answer leading to global minima as verified with other methods/intuition. To conclude, GA is more **robust, easy to implement and reliable tool** for an optimizer when other methods failed.

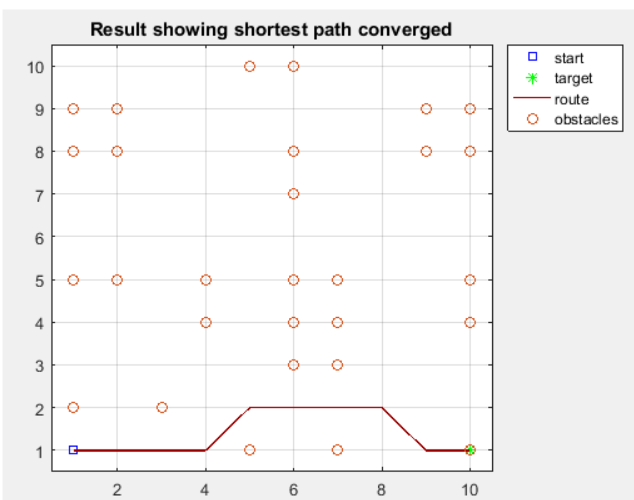
Appendix: Table of contents

Different 2D case results included for showing the robustness/ testing of algorithm:
Case 1: with default GA options and without elitism, the converged answer was not global minima



Result table	Population size	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 1	420	0.001202	255	9.384062	10.65685	25
Run 2	420	0.001202	258	13.04092	14.31371	25

Case 2: With elitism switched on; no. of elite individuals=50; population size=1000; crossover probability =0.5. Bit string affinity: switched off. Maximum generations set to 1000 and rest default.

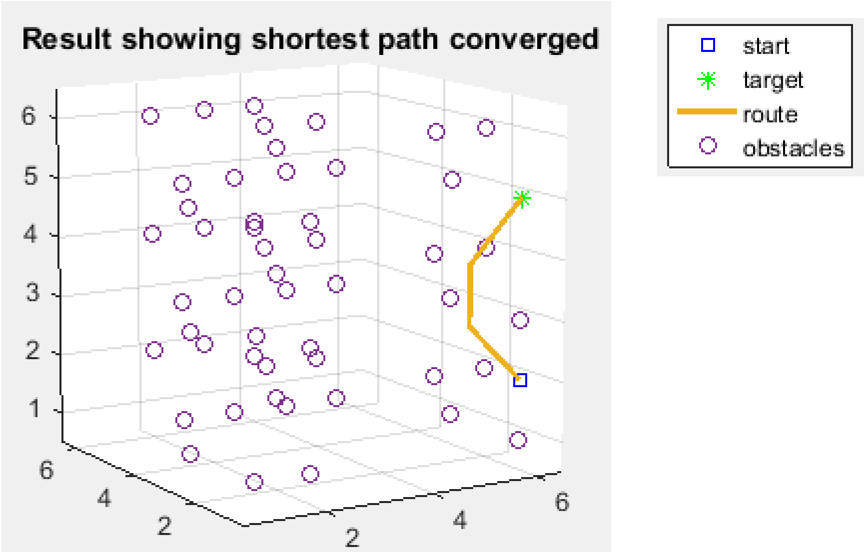


Result table	Population size	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 2	1000	0.000505	56	9.756854	10.65685	25
Run 3	1000	0.000505	34	8.928427	9.828427*	25

**best possible value as confirmed seeing from figure*

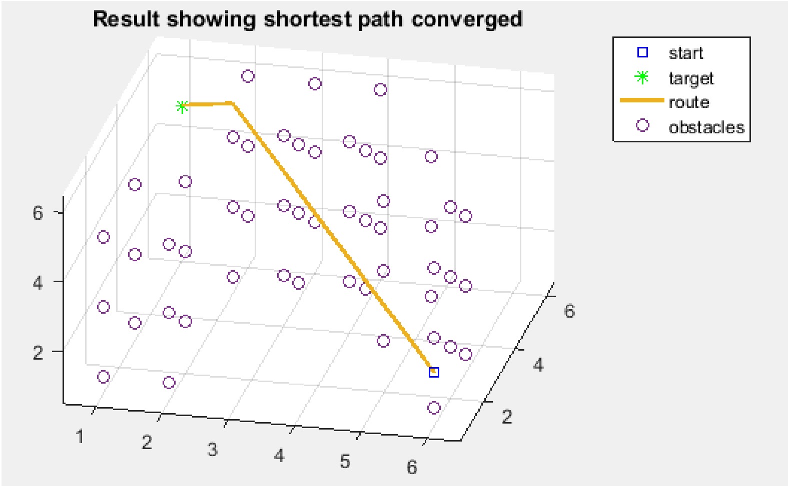
Different 3D case results included for showing the robustness/ testing of algorithm:

case 1:



Result table	Population size	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 1	800	0.000628	201	2.555635	3.828427	25
Run 2	800	0.000628	201	2.555635	3.828427	25
Run 3	800	0.000628	201	2.555635	3.828427	25

case 2:



Result table	Population size	Mutation probability	# generations	Fitness function	Shortest path distance	Convergence consecutive generations with best fitness value
	no units	no units	no units	no units	length units	
Run 1	800	0.000628	117	7.101408	8.660254	25
Run 2	800	0.000628	201	7.101408	8.660254	25
Run 3	800	0.000628	115	7.101408	8.660254	25

callGAproj.m	To call the GA550proj.m algorithm to solve the functional file GAfuncproj.m which has the fitness function

```
% this file provides input variables to call the genetic algorithm
% for solving the path planning problem
% upper and lower bounds, and number of bits chosen path planning problem
% Modified on 12/09/16 by Kumaraguru Sivasankaran.
% AAE550 Project
close all;
clear all;
clc;
options = goptions([]);

options(2)=0;        %   OPTIONS(2)-Termination bit string affinity value
(Default: 0.90; set to zero to turn off)
options(4)=25;       %   OPTIONS(4)-Termination number of consecutive
options(14)=1000;    %   OPTIONS(14)-Maximum number of generations, always
used as safeguard %   (Default: 200).
table=zeros(2,2);
start_cood=[1 1];   %specifying start co-ordinate of robot
target_cood=[10 1]; %specifying target co-ordinate of robot
n_min=1;            %controlling the no. of runs
n_max=3;
x0=[];
for n=n_min:n_max
    vlb = linspace(1,1,35); %Lower bound of each gene - all variables
    vub = linspace(8,8,35); %Upper bound of each gene - all variables
    bits =linspace(3,3,35); %number of bits describing each gene - all
variables
    %calling the GA550 algorithm
    [xbest,fbest,stats,nfit,fgen,lgen,lfit,pop_size,Pm,generation]=
GA550proj('GAfuncproj',x0,options,vlb,vub,bits,start_cood,target_cood);
    %evaluating the best values
    [f,w,d,dd] = GAfuncproj(xbest,start_cood,target_cood);
    %rest of the bits after reaching target is set to zero
    %this is helped to understand the plot better
    length_w=length(w);
    for i=1:length_w
        if w(i)==-0.9
            xbest(i+1:length_w)=0;
            break;
        end
    end
    %required results saved in table for reference
    table(n+1-chrom_leng_min,1)=pop_size;
    table(n+1-chrom_leng_min,2)=Pm;
    table(n+1-chrom_leng_min,3)=generation;
    table(n+1-chrom_leng_min,4)=fbest;
    table(n+1-chrom_leng_min,5)=dd;
    length_x=length(xbest);
for i=1:length(xbest)
    table(n+1-chrom_leng_min,i+5)=xbest(i);
end
end
%function called for visualization of results
m=visualize(xbest,start_cood,target_cood);
```

GAfuncproj.m	Function file which evaluates the fitness function (1) by calling the grid_weight.m function

```
function [f,w,d,dd] = GAfuncproj(x,start_cood,target_cood)
%objective function: GA based path planning
%AAE 550 project - GA based path planning
%Kumaraguru Sivasankaran 12/09/2016

%initializing variables
    length_x=length(x);
    f=0;
    dd=0;
    d=zeros(length_x,1);
    ww=0;
    w=zeros(length_x,1);

%finding the weight of each point in grid
%the step is reset as zero once target achieved
for i=1:length_x
    xx=x(1:i);
    ww=grid_weight(xx,start_cood,target_cood);
    w(i)=ww;
    if ww==-0.9
        x(i+1:length_x)=0;
    end
end
 %to find the distance travelled at each step
 %distance set to zero after target reached
for i=1:length_x
    if (x(i)==1|| x(i)==3|| x(i)==5|| x(i)==7)
        d(i)=1;
    elseif (x(i)==2|| x(i)==4|| x(i)==6|| x(i)==8)
        d(i)=sqrt(2);
    elseif x(i)==0
        d(i)=0;
    else
        d(i)=500;   %for extra variables, higher penalty is set
    end
end
%fitness function formulation as weighted distance
for i=1:length_x
    f = f+(d(i))*(1+w(i));
    dd=dd+d(i);
end
```

grid_weight.m	Function file which returns the weight of grid at each point. This function file helps to identify obstacles in the path

```
function w=grid_weight(x,start_cood,target_cood)
%grid weight evaluation function
%AAE 550 project - GA based path planning
%Kumaraguru Sivasankaran 12/09/2016
    length_x=length(x);
    %Grid matrix with the weight of each point in grid
    Grid=300*[  0 0 0 0 1 0 1 0 0 1 ;...
            1 0 1 0 0 0 0 0 0 0 ;...
            0 0 0 0 0 1 1 0 0 0 ;...
            0 0 0 1 0 1 1 0 0 1 ;...
            1 1 0 1 0 1 1 0 0 1 ;...
            0 0 0 0 0 0 0 0 0 0 ;...
            0 0 0 0 0 1 0 0 0 0 ;...
            1 1 0 0 0 1 0 0 1 1 ;...
            1 1 0 0 0 0 0 0 1 1 ;...
            0 0 0 0 1 1 0 0 0 0 ];
     %user defined target copied on to grid
     Grid(target_cood(2),target_cood(1))=-0.9;
     size_Grid=size(Grid);
     %control loop to find the position on grid
     cood=start_cood;
     move=[0 0];
 for i=1:length_x
    if x(i)==0
     move=[0 0];
    elseif x(i)==1
     move=[1 0];
    elseif x(i)==2
     move=[1 1];
    elseif x(i)==3
     move=[0 1];
    elseif x(i)==4
     move=[-1 1];
    elseif x(i)==5
     move=[-1 0];
    elseif x(i)==6
     move=[-1 -1];
    elseif x(i)==7
     move=[0 -1];
    elseif x(i)==8
     move=[1 -1];
    end
 cood=cood+move;
 end
 m=cood(1);
 n=cood(2);
 %penalty to prevent undesirable conditions  %final grid weight returned
 if m>size_Grid(2)||n>size_Grid(1)||m<=0||n<=0
    w=300;
 else
    w=Grid(n,m);
 end
end
```

visualize.m: Function file which helps to visualize the result by decoding and plotting in the graph

```
function f=visualize(x,start_cood,target_cood)
%function to visualize the results
%AAE 550 project - GA based path planning
%Kumaraguru Sivasankaran 12/09/2016
    format short;
    cood=start_cood;
    move=[0 0];
    length_x=length(x);
    xplot(1)=cood(1);
    yplot(1)=cood(2);
%control loop for identifying the position of point in grid
for i=1:length_x
    if x(i)==0
     move=[0 0];
    elseif x(i)==1
     move=[1 0];
    elseif x(i)==2
     move=[1 1];
    elseif x(i)==3
     move=[0 1];
    elseif x(i)==4
     move=[-1 1];
    elseif x(i)==5
     move=[-1 0];
    elseif x(i)==6
     move=[-1 -1];
    elseif x(i)==7
     move=[0 -1];
    elseif x(i)==8
     move=[1 -1];
    end
    cood=cood+move;
    xplot(i+1)=cood(1);
    yplot(i+1)=cood(2);
end
 %to plot the start coordinate
     plot(start_cood(1),start_cood(2),'bs','DisplayName','start')
     hold on
%to plot the target coordinate
     plot(target_cood(1),target_cood(2),'g*','DisplayName','target')
     hold on
%to plot the route/path taken
     plot(xplot,yplot,'LineWidth',1,'Color',[.6 0 0],'DisplayName','route')
     hold on
 %two dimensional grid 10*10 dimensions
 Grid=300*[  0 0 0 0 1 0 1 0 0 1 ;...
            1 0 1 0 0 0 0 0 0 0 ;...
            0 0 0 0 0 1 1 0 0 0 ;...
            0 0 0 1 0 1 1 0 0 1 ;...
            1 1 0 1 0 1 1 0 0 1 ;...
            0 0 0 0 0 0 0 0 0 0 ;...
            0 0 0 0 0 1 0 0 0 0 ;...
            1 1 0 0 0 1 0 0 1 1 ;...
            1 1 0 0 0 0 0 0 1 1 ;...
```

```
            0 0 0 0 1 1 0 0 0 0 ];
       size_grid=size(Grid);

 %control loop to identify the presence of obstacles
       k=1;
 for i=1:size_grid(1)
     for j=1:size_grid(2)
         if Grid(i,j)>1
             table(k,1)=j;
             table(k,2)=i;
             k=k+1;
         end
     end
 end
 %to plot the obstacles
    plot(table(:,1),table(:,2),'o','DisplayName','obstacles');
 %axis control, title, legend settings,grid control
    axis([0.5 10.5 0.5 10.5]);
    title('Result showing shortest path converged')
    grid on
    f=1;
    legend('show','Location','northeastoutside')
end
```

GA550proj.m	GA algorithm file same as included in AAE550 class

```
function [xopt,fopt,stats,nfit,fgen,lgen,lfit,pop_size,Pm,generation] =
GA550proj(fun, ...
    x0,options,vlb,vub,bits,P1,P2,P3,P4,P5,P6,P7P,P8,P9,P10)
%GA550 minimizes a fitness function using a simple genetic algorithm.
%
%   X=GA550('FUN',X0,OPTIONS,VLB,VUB) uses a simple
%       genetic algorithm to find a minimum of the fitness function
%       FUN.  FUN can be a user-defined M-file: FUN.M, or it can be a
%   string containing the function itself.  The user may define all
%       or part of an initial population X0. Any undefined individuals
%   will be randomly generated between the lower and upper bounds
%   (VLB and VUB).  If X0 is an empty matrix, the entire initial
%   population will be randomly generated.  Use OPTIONS to specify
%   flags, tolerances, and input parameters.  Type HELP GOPTIONS
%       for more information and default values.
%
%   X=GA550('FUN',X0,OPTIONS,VLB,VUB,BITS) allows the user to
%   define the number of BITS used to code non-binary parameters
%   as binary strings.  Note: length(BITS) must equal length(VLB)
%   and length(VUB).  If BITS is not specified, as in the previous
%   call, the algorithm assumes that the fitness function is
%   operating on a binary population.
%
%   X=GA550('FUN',X0,OPTIONS,VLB,VUB,BITS,P1,P2,...) allows up
%   to ten arguments, P1,P2,... to be passed directly to FUN.
%   F=FUN(X,P1,P2,...). If P1,P2,... are not defined, F=FUN(X).
%
%   [X,FOPT,STATS,NFIT,FGEN,LGEN,LFIT]=GA550(<ARGS>)
%          X       - design variables of best ever individual
%          FOPT    - fitness value of best ever individual
%          STATS   - [min mean max stopping_criterion] fitness values
%                    for each generation
%          NFIT  - number of fitness function evalations
%          FGEN    - first generation population
%          LGEN    - last generation population
%          LFIT    - last generation fitness
%
%       The algorithm implemented here is based on the book: Genetic
%       Algorithms in Search, Optimization, and Machine Learning,
%       David E. Goldberg, Addison-Wiley Publishing Company, Inc.,
%       1989.
%
%   Originally created on 1/10/93 by Andrew Potvin, Mathworks, Inc.
%   Modified on 2/3/96 by Joel Grasmeyer.
%   Modified on 11/12/02 by Bill Crossley.
%   Modified on 7/20/04 by Bill Crossley.

% Make best_feas global for stopping criteria (4/13/96)
global best_feas
global gen
global fit_hist
% Load input arguments and check for errors
if nargin<4,
    error('No population bounds given.')
elseif (size(vlb,1)~=1) | (size(vub,1)~=1),
```

```
    % Remark: this will change if algorithm accomodates matrix variables
    error('VLB and VUB must be row vectors')
elseif (size(vlb,2)~=size(vub,2)),
    error('VLB and VUB must have the same number of columns.')
elseif (size(vub,2)~=size(x0,2)) & (size(x0,1)>0),
    error('X0 must all have the same number of columns as VLB and VUB.')
elseif any(vlb>vub),
    error('Some lower bounds greater than upper bounds')
else
    x0_row = size(x0,1);
    for i=1:x0_row,
        if any(x0(x0_row,:)<vlb) | any(x0(x0_row,:)>vub),
            error('Some initial population not within bounds.')
        end % if initial pop not within bounds
    end % for initial pop
end % if nargin<4

if nargin<6,
    bits = [];
elseif (size(bits,1)~=1) | (size(bits,2)~=size(vlb,2)),
    % Remark: this will change if algorithm accomodates matrix variables
    error('BITS must have one row and length(VLB) columns')
elseif any(bits~=round(bits)) | any(bits<1),
    error('BITS must be a vector of integers >0')
end % if nargin<6

% Form string to call for function evaluation
if ~( any(fun<48) | any(fun>122) | any((fun>90) & (fun<97)) | ...
        any((fun>57) & (fun<65)) ),
    % Only alphanumeric characters implies that 'fun' is a separate m-file
    evalstr = [fun '(x'];
    for i=1:nargin-6,
        evalstr = [evalstr,',P',int2str(i)];
    end
else
    % Non-alphanumeric characters implies that the function is contained
    % within the single quotes
    evalstr = ['(',fun];
end

% Determine all options
% Remark: add another options index for type of termination criterion
if size(options,1)>1,
    error('OPTIONS must be a row vector')
else
    % Use default options for those that were not passed in
    options = goptions(options);
end
PRINTING = options(1);
BSA = options(2);
fit_tol = options(3);
nsame = options(4)-1;
elite = options(5);

% Since operators are tournament selection and uniform crossover and
% default coding is Gray / binary, set crossover rate to 0.50 and use
```

```
% population size and mutation rate based on Williams, E. A., and Crossley,
% W. A., "Empirically-derived population size and mutation rate guidelines
% for a genetic algorithm with uniform crossover," Soft Computing in
% Engineering Design and Manufacturing, 1998.  If user has entered values
% for these options, then user input values are used.
if options(11) == 0,
    pop_size = sum(bits) * 4;
else
    pop_size = options(11);
end
if options(12) == 0,
    Pc = 0.5;
else
    Pc = options(12);
end
if options(13) == 0,
    Pm = (sum(bits) + 1) / (2 * pop_size * sum(bits));
else
    Pm = options(13);
end
max_gen = options(14);
% Ensure valid options: e.q. Pc,Pm,pop_size,max_gen>0, Pc,Pm<1
if any([Pc Pm pop_size max_gen]<0) | any([Pc Pm]>1),
    error('Some Pc,Pm,pop_size,max_gen<0 or Pc,Pm>1')
end

% Encode fitness (cost) function if necessary
ENCODED = any(any(([vlb; vub; x0]~=0) & ([vlb; vub; x0]~=1))) |  ....
    ~isempty(bits);
if ENCODED,
    [fgen,lchrom] = encode(x0,vlb,vub,bits);
else
    fgen = x0;
    lchrom = size(vlb,2);
end

% Display warning if initial population size is odd
if rem(pop_size,2)==1,
    disp('Warning: Population size should be even.  Adding 1 to population.')
    pop_size = pop_size +1;
end

% Form random initial population if not enough supplied by user
if size(fgen,1)<pop_size,
    fgen = [fgen; (rand(pop_size-size(fgen,1),lchrom)<0.5)];
end
xopt = vlb;
nfit = 0;
new_gen = fgen;
isame = 0;
bitlocavg = mean(fgen,1);  % initial bit string affinity
BSA_pop = 2 * mean(abs(bitlocavg - 0.5));
fopt = Inf;
stats = [];

% Header display
```

```
if PRINTING>=1,
    if ENCODED,
        disp('Variable coding as binary chromosomes successful.')
        disp('')
        fgen = decode(fgen,vlb,vub,bits);
    end
    disp('                   Fitness statistics')
    if nsame > 0
        disp('Generation Minimum      Mean          Maximum       isame')
    elseif BSA > 0
        disp('Generation Minimum      Mean          Maximum       BSA')
    else
        disp('Generation Minimum      Mean          Maximum       not used')
    end
end

% Set up main loop
STOP_FLAG = 0;
for generation = 1:max_gen+1,
    old_gen = new_gen;

    % Decode binary strings if necessary
    if ENCODED,
        x_pop = decode(old_gen,vlb,vub,bits);
    else
        x_pop = old_gen;
    end

    % Get fitness of each string in population
    for i = 1:pop_size,
        x = x_pop(i,:);
        fitness(i) = eval([evalstr,')']);
        nfit = nfit + 1;
    end

    % Store minimum fitness value from previous generation (except for
    % initial generation)
    if generation > 1,
        min_fit_prev = min_fit;
        min_gen_prev = min_gen;
        min_x_prev = min_x;
    end

    % identify worst (maximum) fitness individual in current generation
    [max_fit,max_index] = max(fitness);

    % impose elitism - currently only one individual; this replaces worst
    % individual of current generation with best of previous generation
    if (generation > 1 & elite > 0),
        old_gen(max_index,:) = min_gen_prev;
        x_pop(max_index,:) = min_x_prev;
        fitness(max_index) = min_fit_prev;
    end

    % identify best (minimum) fitness individual in current generation and
```

```
    % store bit string and x values
    [min_fit,min_index] = min(fitness);
    min_gen = old_gen(min_index,:);
    min_x = x_pop(min_index,:);

    % Store best fitness and x values
    if min_fit < fopt,
        fopt = min_fit;
        xopt = min_x;
    end

    % Compute values for isame or BSA_pop stopping criteria
    if nsame > 0
        if generation > 1
            if min_fit_prev == min_fit
                isame = isame + 1;
            else
                isame = 0;
            end
        end
    elseif BSA > 0
        bitlocavg = mean(old_gen,1);
        BSA_pop = 2 * mean(abs(bitlocavg - 0.5));
    end


    % Calculate generation statistics
    if nsame > 0
        stats = [stats; generation-1,min(fitness),mean(fitness), ...
            max(fitness), isame];
    elseif BSA > 0
        stats = [stats; generation-1,min(fitness),mean(fitness), ...
            max(fitness), BSA_pop];
    else
        stats = [stats; generation-1,min(fitness),mean(fitness), ...
            max(fitness), 0];
    end

    % Display if necessary
    if PRINTING>=1,
        disp([sprintf('%5.0f %12.6g %12.6g %12.6g %12.6g',
stats(generation,1), ...
                stats(generation,2),stats(generation,3),
stats(generation,4),...
                stats(generation,5))]);
    end

    % Check for termination
    % The default termination criterion is bit string affinity.  Also
    % available are fitness tolerance across five generations and number of
    % consecutive generations with same best fitness.  These can be used
    % concurrently.
    if fit_tol>0,    % if fit_tol > 0, then fitness tolerance criterion used
        if generation>5,
            % Check for normalized difference in fitness minimums
            if stats(generation,1) ~= 0,
```

```
                if abs(stats(generation-5,1)-stats(generation,1))/ ...
                        stats(generation,1) < fit_tol
                    if PRINTING >= 1
                        fprintf('\n')
                        disp('GA converged based on difference in fitness 
minimums.')
                    end
                    lfit = fitness;
                    if ENCODED,
                        lgen = x_pop;
                    else
                        lgen = old_gen;
                    end
                    return
                end
            else
                if abs(stats(generation-5,1)-stats(generation,1)) < fit_tol
                    if PRINTING >= 1
                        fprintf('\n')
                        disp('GA converged based on difference in fitness 
minimums.')
                    end
                    lfit = fitness;
                    if ENCODED,
                        lgen = x_pop;
                    else
                        lgen = old_gen;
                    end
                    return
                end
            end
        end
    elseif nsame > 0,     % consecutive minimum fitness value criterion
            if isame == nsame
                if PRINTING >= 1
                    fprintf('\n')
                    disp('GA stopped based on consecutive minimum fitness 
values.')
                end
                lfit = fitness;
                if ENCODED,
                    lgen = x_pop;
                else
                    lgen = old_gen;
                end
                return
            end
    elseif BSA > 0,   % bit string affinity criterion
        if BSA_pop >= BSA,
            if PRINTING >=1
                fprintf('\n')
                disp('GA stopped based on bit string affinity value.')
            end
            lfit = fitness;
            if ENCODED,
                lgen = x_pop;
            else
```

```
                lgen = old_gen;
            end
            return
        end
    end

    % Tournament selection
    new_gen = tourney(old_gen,fitness);

    % Crossover
    new_gen = uniformx(new_gen,Pc);

    % Mutation
    new_gen = mutate(new_gen,Pm);

    % Always save last generation.  This allows user to cancel and
    % restart with x0 = lgen
    if ENCODED,
        lgen = x_pop;
    else
        lgen = old_gen;
    end


end % for max_gen

% Maximum number of generations reached without termination
lfit = fitness;
if PRINTING>=1,
    fprintf('\n')
    disp('Maximum number of generations reached without termination')
    disp('criterion met.  Either increase maximum generations')
    disp('or ease termination criterion.')
end


% end genetic

function [gen,lchrom,coarse,nround] = encode(x,vlb,vub,bits)
%ENCODE Converts from variable to binary representation.
%   [GEN,LCHROM,COARSE,nround] = ENCODE(X,VLB,VUB,BITS)
%       encodes non-binary variables of X to binary.  The variables
%       in the i'th column of X will be encoded by BITS(i) bits.  VLB
%       and VUB are the lower and upper bounds on X.  GEN is the binary
%       representation of these X.  LCHROM=SUM(BITS) is the length of
%       the binary chromosome.  COARSE(i) is the coarseness of the
%       i'th variable as determined by the variable ranges and
%       BITS(i).  ROUND contains the absolute indices of the
%       X which where rounded due to finite BIT length.
%
%   Copyright (c) 1993 by the MathWorks, Inc.
%   Andrew Potvin 1-10-93.

% Remark: what about handling case where length(bits)~=length(vlb)?
```

```
lchrom = sum(bits);
coarse = (vub-vlb)./((2.^bits)-1);
[x_row,x_col] = size(x);

gen = [];
if ~isempty(x),
   temp = (x-ones(x_row,1)*vlb)./ ...
          (ones(x_row,1)*coarse);
   b10 = round(temp);
   % Since temp and b10 should contain integers 1e-4 is close enough
   nround = find(b10-temp>1e-4);
   gen = b10to2(b10,bits);
end

% end encode


function [x,coarse] = decode(gen,vlb,vub,bits)
%DECODE Converts from binary Gray code to variable representation.
%    [X,COARSE] = DECODE(GEN,VLB,VUB,BITS) converts the binary
%        population GEN to variable representation.  Each individual
%        of GEN should have SUM(BITS).  Each individual binary string
%        encodes LENGTH(VLB)=LENGTH(VUB)=LENGTH(BITS) variables.
%        COARSE is the coarseness of the binary mapping and is also
%        of length LENGTH(VUB).
%
%  this *.m file created by combining "decode.m" from the MathWorks, Inc.
%  originally created by Andrew Potvin in 1993, with "GDECODE.FOR" written
%  by William A. Crossley in 1996.
%
%   William A. Crossley, Assoc. Prof. School of Aero. & Astro.
%  Purdue University, 2001
%
%  gen is an array [population size , string length], each row is one
individual's chromosome
%  vlb is a row vector [number of parameters], each entry is the lower bound
for a variable
%  vub is a row vector [number of parameters], each entry is the upper bound
for a variable
%  bits is a row vector [number of parameters], each entry is number of bits
used for a variable
%

no_para = length(bits); % extract number of parameters using number of rows
in bits vector
npop = size(gen,1);      % extract population size using number of rows in gen
array
x = zeros(npop, no_para);  % sets up x as an array [population size, number
of parameters]
coarse = zeros(1,no_para); % sets up coarse as a row vector [number of
parameters]

for J = 1:no_para,  % extract the resolution of the parameters
```

```
    coarse(J) = (vub(J)-vlb(J))/(2^bits(J)-1);  % resolution of parameter J
end

for K = 1:npop,      % outer loop through each individual (there may be a more
efficient way to operate on the
                  % gen array) BC 10/10/01
    sbit = 1;        % initialize starting bit location for a parameter
    ebit = 0;        % initialize ending bit location

   for J = 1:no_para,    % loop through each parameter in the problem
    ebit = bits(J) + ebit;  % pick the end bit for parameter J
        accum = 0.0;                % initialize the running sum for
parameter J
      ADD = 1;                      % add / subtract flag for Gray code; add
if(ADD), subtract otherwise
      for I = sbit:ebit,            % loop through each bit in parameter J
         pbit = I + 1 - sbit;       % pbit determines value to be added or
subtracted for Gray code
         if (gen(K,I))                  % if "1" is at current location
            if (ADD)                        % add if appropriate
               accum = accum + (2.0^(bits(J)-pbit+1) - 1.0);
               ADD = 0;                 % next time subtract
            else
               accum = accum - (2.0^(bits(J)-pbit+1) - 1.0);
               ADD = 1;                 % next time add
            end
         end
      end                               % end of I loop through each bit
      x(K,J) = accum * coarse(J) + vlb(J);          % decoded parameter J for
individual K
      sbit = ebit + 1;                                      % next parameter
starting bit location
   end                          % end of J loop through each parameter
end                         % end of K loop through each individual

%end gdecode


function [new_gen,mutated] = mutate(old_gen,Pm)
%MUTATE Changes a gene of the OLD_GEN with probability Pm.
%   [NEW_GEN,MUTATED] = MUTATE(OLD_GEN,Pm) performs random
%       mutation on the population OLD_POP.  Each gene of each
%       individual of the population can mutate independently
%       with probability Pm.  Genes are assumed possess boolean
%       alleles.  MUTATED contains the indices of the mutated genes.
%
%   Copyright (c) 1993 by the MathWorks, Inc.
%   Andrew Potvin 1-10-93.

mutated = find(rand(size(old_gen))<Pm);
new_gen = old_gen;
new_gen(mutated) = 1-old_gen(mutated);

% end mutate
```

```
function [new_gen,nselected] = tourney(old_gen,fitness)
%TOURNEY Creates NEW_GEN from OLD_GEN, based on tournament selection.
%    [NEW_GEN,NSELECTED] = TOURNEY(OLD_GEN,FITNESS) selects
%        individuals from OLD_GEN by competing consecutive individuals
%    after random shuffling.  NEW_GEN will have the same number of
%    individuals as OLD_GEN.
%        NSELECTED contains the number of copies of each individual
%    that survived.  This vector corresponds to the original order
%    of OLD_GEN.
%
%    Created on 1/21/96 by Joel Grasmeyer

% Initialize nselected vector and indices of old_gen
new_gen = [];
nselected = zeros(size(old_gen,1),1);
i_old_gen = 1:size(old_gen,1);

% Perform two "tournaments" to generate size(old_gen,1) new individuals
for j = 1:2,

  % Shuffle the old generation and the corresponding fitness values
  [old_gen,i_shuffled] = shuffle(old_gen);
  fitness = fitness(i_shuffled);
  i_old_gen = i_old_gen(i_shuffled);

  % Keep the best of each pair of individuals
  index = 1:2:(size(old_gen,1)-1);
  [min_fit,i_min] = min([fitness(index);fitness(index+1)]);
  selected = i_min + [0:2:size(old_gen,1)-2];
  new_gen = [new_gen; old_gen(selected,:)];

  % Increment counters in nselected for each individual that survived
  temp = zeros(size(old_gen,1),1);
  temp(i_old_gen(selected)) = ones(length(selected),1);
  nselected = nselected + temp;

end

% end tourney


function [new_gen,index] = shuffle(old_gen)
%SHUFFLE Randomly reorders OLD_GEN into NEW_GEN.
%    [NEW_GEN,INDEX] = MATE(OLD_GEN) performs random reordering
%        on the indices of OLD_GEN to create NEW_GEN.
%    INDEX is a vector containing the shuffled row indices of OLD_GEN.
%
%    Created on 1/21/96 by Joel Grasmeyer

[junk,index] = sort(rand(size(old_gen,1),1));
new_gen = old_gen(index,:);

% end shuffle
```

```
function [new_gen,sites] = uniformx(old_gen,Pc)
%UNIFORMX Creates a NEW_GEN from OLD_GEN using uniform crossover.
%      [NEW_GEN,SITES] = UNIFORMX(OLD_GEN,Pc) performs uniform crossover
%          on consecutive pairs of OLD_GEN with probability Pc.
%      SITES shows which bits experienced crossover.  1 indicates
%      allele exchange, 0 indicates no allele exchange.  SITES has
%      size(old_gen,1)/2 rows.
%
%      Created 1/20/96 by Joel Grasmeyer

new_gen = old_gen;
sites = rand(size(old_gen,1)/2,size(old_gen,2)) < Pc;
for i = 1:size(sites,1),
  new_gen([2*i-1 2*i],find(sites(i,:))) = old_gen([2*i
2*i-1],find(sites(i,:)));
end

% end uniformx
```

gooptions.m	To set different options for the genetic algorithm GA550proj.m based on user inputs

```
function OPTIONS=goptions(parain);
%GOPTIONS Default parameters used by the genetic algorithm GENETIC.
%
% Note that since the original version was written, the Matlab Optimization
% Toolbox now uses "optimset" to set generic optimization parameters, so
% this format is somewhat outdated.
%
% The genetic algorithm parameters used for this implementation are:
%
%   OPTIONS(1)-Display flag:  0 = none, 1 = some, 2 = all  (Default: 1).
%   OPTIONS(2)-Termination bit string affinity value (Default: 0.90; set to
zero to turn off)
%   OPTIONS(3)-Termination tolerance for fitness (Default: 0; not normally
used).
%   OPTIONS(4)-Termination number of consecutive generations with same best
%   fitness (Default: 0; to use, set number, be sure OPTIONS(2) and
OPTIONS(3) = 0).
%   OPTIONS(5)-Number of elite individuals (Default: 0; no elitism).
%   OPTIONS(6)-
%   OPTIONS(7)-
%   OPTIONS(8)-
%   OPTIONS(9)-
%   OPTIONS(10)-
% Genetic Algorithm-specific inputs
%   OPTIONS(11)-Population size (fixed)
%   OPTIONS(12)-Probability of crossover
%   OPTIONS(13)-Probability of mutation
%   OPTIONS(14)-Maximum number of generations, always used as safeguard
%   (Default: 200).
%
%
% Explanation of defaults:
%   The default algorithm displays statistical information for each
%   generation by setting OPTIONS(1) = 1.  Plots are produced when
%   OPTIONS(1) = 2.
%   The OPTIONS(2) flag is originally set for termination criterion based
%   on X; here it is used if bit string affinity is selected.
%   The default fitness function termination tolerance,
%   OPTIONS(3), is set to 0, which terminates the optimization when 5
%   consecutive best generation fitness values are the same.  A positive
%   value terminates the optimization when the normalized difference
%   between the previous fitness and current generation fitness is less
%   than the tolerance.  See the code for details.
%  OPTIONS(4) has a default value of 5; this means if the best fitness
%   value in the population is unchanged for 5 consecutive generations
%  the GA is terminated.
%       The default algorithm uses a fixed population size, OPTIONS(11),
%       and no generational overlap.  The default population size is 30.
%   Three genetic operations:  selection, crossover, and mutation are
%   used for procreation.
%   The default selection scheme is tournament selection.
%       Crossover occurs with probability Pc=OPTIONS(12).  The default
%   crossover scheme is uniform crossover with Pc = 0.5.
```

```
%   Each allele of the offspring mutates independently with probability
%       Pm=OPTIONS(13); here the default is 0.01.
%       The default number of maximum generations, OPTIONS(14) is 200.
%
%   Last modified by Bill Crossley 07/20/04

% The following lines have been commented out by Steven Lamberson.
% They have been changed to what is seen below them. (06/30/06).
% This change was made in order to fix the following problems:
%   1 - code changed user supplied options(1)=0 to options(1)=1
%   2 - code changed user supplied options(2)=0 to options(2)=0.9

%if nargin<1; parain = []; end
%sizep=length(parain);
%OPTIONS=zeros(1,14);
%OPTIONS(1:sizep)=parain(1:sizep);
%default_options=[1,0.9,0,0,0,0,0,0,0,0,0,0,0,200];
%OPTIONS=OPTIONS+(OPTIONS==0).*default_options

if nargin<1; parain = []; end
sizep=length(parain);
OPTIONS=zeros(1,14)-1;
OPTIONS(1:sizep)=parain(1:sizep);
default_options=[1,0.9,0,0,0,0,0,0,0,0,0,0,0,200];
for i = 1:length(OPTIONS)
    if OPTIONS(i) == -1
        OPTIONS(i) = default_options(i);
    end
end
```

callGAproj3D.m	To call the GA550proj3D.m algorithm to solve the functional file GAfuncproj3D.m which has the fitness function

```
% this file provides input variables to the genetic algorithm
% upper and lower bounds, and number of bits chosen path planning problem
% Modified on 12/09/16 by Kumaraguru Sivasankaran.
% AAE550 Project
close all;
clear all;
clc;
options = goptions([]);
start_cood=[1 1 1];               %specifying start co-ordinate of robot
target_cood=[6 6 6];              %specifying target co-ordinate of robot
options(2)=0;                     %termination criteria of bit string affinity
set to zero
options(4)=50;                    %termination criteria of consecutive
generations with best fitness value
table=zeros(2,2);
n_min=1;                          %controlling the no. of runs
n_max=3;
for n=n_min:n_max
    vlb = linspace(1,1,40);       %Lower bound of each gene - all variables
    vub = linspace(32,32,40);     %Upper bound of each gene - all variables
    bits =linspace(5,5,40);       %number of bits describing each gene - all
variables
    %calling the GA550 algorithm
    [xbest,fbest,stats,nfit,fgen,lgen,lfit,pop_size,Pm,generation]=
GA550proj3D('GAfuncproj3D',[ ],options,vlb,vub,bits,start_cood,target_cood);
    %evaluating the best values
    [f,dd,w,d] = GAfuncproj3D(xbest,start_cood,target_cood);
    length_d=length(d);
    %rest of the bits after reaching target is set to zero
    %this is helped to understand the plot better
    for i=1:length_d
        if d(i)==0
            xbest(i:length_d)=0;
            break;
        end
    end
    %required results saved in table for reference
    table(n+1-chrom_leng_min,1)=pop_size;
    table(n+1-chrom_leng_min,2)=Pm;
    table(n+1-chrom_leng_min,3)=generation;
    table(n+1-chrom_leng_min,4)=fbest;
    table(n+1-chrom_leng_min,5)=dd;
    length_x=length(xbest);
    for i=1:length(xbest)
    table(n+1-chrom_leng_min,i+5)=xbest(i);
end
end
%function called for visualization of results
m=visualize3D(xbest,start_cood,target_cood);
```

GAfuncproj3D.m	Function file which evaluates the fitness function (1) by calling the grid_weight3D.m function

```
function [f,dd,w,d] = GAfuncproj3D(x,start_cood,target_cood)
%objective function: GA based path planning
%AAE 550 project - GA based path planning
%Kumaraguru Sivasankaran 12/09/2016

%initializing variables
    length_x=length(x);
    f=0;
    dd=0;
    d=zeros(length_x,1);
    ww=0;
    w=zeros(length_x,1);

 %finding the weight of each point in grid
 %the step is reset as zero once target achieved
for i=1:length_x
    xx=x(1:i);
    ww=grid_weight3D(xx,start_cood,target_cood);
    w(i)=ww;
    if ww==-0.9
        x(i+1:length_x)=0;
    end
end

 %to find the distance travelled at each step
 %distance set to zero after target reached
for i=1:length_x
    if (x(i)==1|| x(i)==3|| x(i)==5|| x(i)==7|| x(i)==17||x(i)==26)
        d(i)=1;
    elseif (x(i)==2|| x(i)==4|| x(i)==6|| 
x(i)==8||x(i)==9||x(i)==13||x(i)==11||x(i)==15)
        d(i)=sqrt(2);
    elseif (x(i)==18|| x(i)==24|| x(i)==20|| x(i)==22)
        d(i)=sqrt(2);
    elseif (x(i)==10|| x(i)==12|| x(i)==14|| x(i)==16||x(i)==19|| x(i)==21|| 
x(i)==23|| x(i)==25)
        d(i)=sqrt(3);
    elseif x(i)==0
        d(i)=0;
    else
        d(i)=1000;  %for extra variables, higher penalty is set
    end
end

%fitness function formulation as weighted distance
for i=1:length_x
    f = f+d(i)*(1+w(i));
    dd=dd+d(i);
end
```

grid_weight3D.m	Function file which returns the weight of grid at each point. This function file helps to identify obstacles in the path

```
function [w,cood]=grid_weight3D(x,start_cood,target_cood)
%grid weight evaluation function
%AAE 550 project - GA based path planning
%Kumaraguru Sivasankaran 12/09/2016
 length_x=length(x);
 %Grid showing the weight of each point in grid
 Grid(:,:,1)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,2)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];
  Grid(:,:,3)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,4)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];
  Grid(:,:,5)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,6)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];

  %user defined target copied on to grid
 Grid(target_cood(1),target_cood(2),target_cood(3))=-0.9;
 size_Grid=size(Grid);
 cood=start_cood;

 %control loop to find the position on grid
 move=[0 0 0];
 for i=1:length_x
    if x(i)==0
     move=[0 0 0];
```

```
    elseif x(i)==1
     move=[1 0 0];
    elseif x(i)==2
     move=[1 1 0];
    elseif x(i)==3
     move=[0 1 0];
    elseif x(i)==4
     move=[-1 1 0];
    elseif x(i)==5
     move=[-1 0 0];
    elseif x(i)==6
     move=[-1 -1 0];
    elseif x(i)==7
     move=[0 -1 0];
    elseif x(i)==8
     move=[1 -1 0];
     elseif x(i)==9
     move=[1 0 1];
    elseif x(i)==10
     move=[1 1 1];
    elseif x(i)==11
     move=[0 1 1];
    elseif x(i)==12
     move=[-1 1 1];
    elseif x(i)==13
     move=[-1 0 1];
    elseif x(i)==14
     move=[-1 -1 1];
    elseif x(i)==15
     move=[0 -1 1];
    elseif x(i)==16
     move=[1 -1 1];
    elseif x(i)==17
     move=[0 0 1];
     elseif x(i)==18
     move=[1 0 -1];
    elseif x(i)==19
     move=[1 1 -1];
    elseif x(i)==20
     move=[0 1 -1];
    elseif x(i)==21
     move=[-1 1 -1];
    elseif x(i)==22
     move=[-1 0 -1];
    elseif x(i)==23
     move=[-1 -1 -1];
    elseif x(i)==24
     move=[0 -1 -1];
    elseif x(i)==25
     move=[1 -1 -1];
    elseif x(i)==26
     move=[0 0 -1];
    end
 cood=cood+move;
 end
 m=cood(1);
 n=cood(2);
```

```
 l=cood(3);

 %penalty to prevent undesirable conditions
 %final grid weight returned
 if m>size_Grid(2)||n>size_Grid(1)||m<=0||n<=0||l<=0||l>size_Grid(3)
    w=3000;
 else
    w=Grid(n,m,l);
 end
end
```

visualize3D.m	Function file which helps to visualize the result by decoding and plotting in the graph

```
function f=visualize3D(x,start_cood,target_cood)
    %function to visualize the results
    %AAE 550 project - GA based path planning
    %Kumaraguru Sivasankaran 12/09/2016
    format short;
    %three dimensional grid 6*6*6 dimensions
  Grid(:,:,1)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,2)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];
  Grid(:,:,3)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,4)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];
  Grid(:,:,5)=300*[1 1 0 0 0 1;...
      0 0 0 0 0 0;...
      1 0 0 0 0 1;...
      0 0 0 0 0 0;...
      0 1 1 1 1 0;...
      0 0 0 0 0 0];
  Grid(:,:,6)=300*[0 0 0 0 0 0;...
      0 1 0 0 1 1;...
      0 0 0 0 0 0;...
      0 0 1 1 0 0;...
      0 0 0 0 0 0;...
      0 1 1 1 0 0];
   size_grid=size(Grid);
    cood=start_cood;
    length_x=length(x);
    %target grid value reset using user input
    xplot(1)=cood(1);
    yplot(1)=cood(2);
    zplot(1)=cood(3);
    Grid(target_cood(1),target_cood(2),target_cood(3))=-0.9;
    table1=zeros(3,3);
    %control loop for identifying the position of point in grid
    move=[0 0 0];
```

```
for i=1:length_x
    if x(i)==0
     move=[0 0 0];
    elseif x(i)==1
     move=[1 0 0];
    elseif x(i)==2
     move=[1 1 0];
    elseif x(i)==3
     move=[0 1 0];
    elseif x(i)==4
     move=[-1 1 0];
    elseif x(i)==5
     move=[-1 0 0];
    elseif x(i)==6
     move=[-1 -1 0];
    elseif x(i)==7
     move=[0 -1 0];
    elseif x(i)==8
     move=[1 -1 0];
     elseif x(i)==9
     move=[1 0 1];
    elseif x(i)==10
     move=[1 1 1];
    elseif x(i)==11
     move=[0 1 1];
    elseif x(i)==12
     move=[-1 1 1];
    elseif x(i)==13
     move=[-1 0 1];
    elseif x(i)==14
     move=[-1 -1 1];
    elseif x(i)==15
     move=[0 -1 1];
    elseif x(i)==16
     move=[1 -1 1];
    elseif x(i)==17
     move=[0 0 1];
     elseif x(i)==18
     move=[1 0 -1];
    elseif x(i)==19
     move=[1 1 -1];
    elseif x(i)==20
     move=[0 1 -1];
    elseif x(i)==21
     move=[-1 1 -1];
    elseif x(i)==22
     move=[-1 0 -1];
    elseif x(i)==23
     move=[-1 -1 -1];
    elseif x(i)==24
     move=[0 -1 -1];
    elseif x(i)==25
     move=[1 -1 -1];
    elseif x(i)==26
     move=[0 0 -1];
    end
    cood=cood+move;
```

```
    xplot(i+1)=cood(1);
    yplot(i+1)=cood(2);
    zplot(i+1)=cood(3);
 end
    m=cood(1);
    n=cood(2);
    l=cood(3);

    %to plot the start coordinate

scatter3(start_cood(1),start_cood(2),start_cood(3),'bs','DisplayName','start'
)
    hold on
    %to plot the target coordinate

scatter3(target_cood(2),target_cood(1),target_cood(3),'g*','DisplayName','tar
get')
    hold on
    %to plot the route/path taken
    plot3(xplot,yplot,zplot,'LineWidth',2,'DisplayName','route')
    hold on
    %control loop to identify the presence of obstacles
    m=1;
 for k=1:size_grid(3)
     for i=1:size_grid(1)
       for j=1:size_grid(2)
         if Grid(i,j,k)>1
             table1(m,1)=j;
             table1(m,2)=i;
             table1(m,3)=k;
             m=m+1;
         end
       end
     end
 end
    %to plot the obstacles

scatter3(table1(:,1),table1(:,2),table1(:,3),'o','DisplayName','obstacles');
    %axis control, title, legend settings,grid
    axis([0.5 6.5 0.5 6.5 0.5 6.5]);
    title('Result showing shortest path converged')
    grid on
    f=1;
    legend('show','Location','northeastoutside')
    hold on
    table1;
end
```