SUMMER INTERNSHIP REPORT

ON

UAV FLIGHT MISSION PLANNING

(Using A* algorithm interfacing in STK and MATLAB)

Submitted by:

Sammit Jain (2014B4A3909G)

3rd Year MSc. Mathematics and B.E. EEE

Anirudh Srinivas (2015B4A3407G)

2nd Year MSc. Mathematics and B.E. EEE

Prudhvi Rampey (2015A3PS127G)

2nd Year B.E. Electronics

Under the guidance

Of

Ranjana N (Sc. 'G')

Sumant (Sc. 'G')

Institute of Systems Studies & Analysis (ISSA)

Defence Research and Development Organization (DRDO)

Metcafe house, Delhi -110054

CANDIDATE'S DECLARATION

We, hereby declare that the work which is being presented in this project report entitled "UAV Flight Mission Planning" submitted, as a part of curriculum is an authentic record of our original work carried out under the guidance of N Ranjana, Sc. 'G' and Sumant,Sc.'G' ISSA, DRDO, Metcalfe House, Delhi-110054.

We are highly obliged to Defence Research and Development Organization, Delhi.

DATE :

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

N Ranjana , Sc. 'G'

ISSA DRDO

Metcalfe House

Delhi-110054

Abstract

Route planning has been an important topic of research in many areas including robotics and navigation. This is because it needs to provide the agent with a collision-free and optimised route through the workspace. Here, we address route planning in the context of *Unmanned Aerial Vehicles* as path planning is used in the realm of flight navigation to avoid potential weather hazards, radar detection, or sudden threats.

An Unmanned Aerial Vehicle (UAV) is one of the most popular and useful unmanned systems. It is a vital element for both civilian and military areas. The major fields in which UAVs are deployed are weather forecasting, surveillance, cargo distribution, traffic and environmental monitoring. UAVs possess portability and agility which give them a high success rate in military and civilian missions, as compared to manned aircrafts. UAVs will be playing a more significant role in the aviation industry in the future.

UAV FLIGHT MISSION PLANNING presents an application that allows global 3D maps, weather conditions and intelligence data to be used in developing navigation solutions while optimising for fuel and leg times.

This application includes *Systems Toolkit* (visual software tool) to simulate and depict mission plans in a realistic environment. It also includes MATLAB scripts to automate the computations associated with mission planning. MATLAB was used because it allows us to easily carry out and *visualize* computations. The A* algorithm is used optimize the mission route by generating a stealthy path through a set of enemy radar sites of known location. It also provides an intuitive way to trade-off stealth versus path length.

Table of contents

Introduction

Literature Survey

The path planning problem is still an active area of research. UAV mission planning requires generating a plan that is large enough to account for the overall mission performance. Also, optimization of the planned path according to a set of factors requires more computation time.

Genetic Algorithms are a possible solution for finding an appropriate path for a UAV. However, they are relatively slower when it comes to simple path planning in a Grid system and do not necessarily generate the most optimal path (might not converge at global minimum). Artificial Neural Networks again face the same problem. Although they are faster than Genetic algorithms, they cannot always produce the most optimum solution.

A better solution would be to use a Fast-pass A* algorithm that is guaranteed to give us the most optimal path. The A* star algorithm is designed to effectively navigate graphs and grid systems. It is built on a flexible *heuristics* model that allows us to easily add more factors and increase complexity of the mission plan, while keeping computation time to a minimum.

Problem statement

The trajectory of a UAV has to be optimised to take into account fuel constraints, positioning of enemy radars, weather conditions and other constraints. The purpose of this project is to provide a **working model** of optimised flight route planning, in the wake of enemy dangers like radars.

Scope

All simulations are carried out by means of software suite by Analytical Graphics, Inc (AGI) called Systems Toolkit (STK). STK runs simulations with real-world parameters, therefore this allows us to plan routes in highly realistic scenarios.

In order to account for limited *fuel supply* the route plan is optimised to find the shortest path, while navigating enemy radars.

Tools

Systems toolkit

- Systems Tool Kit is a physics-based software package from Analytical Graphics, Inc. that allows engineers and scientists to perform complex analyses of ground, sea, air, and space assets, and share results in one integrated solution.
- STK provides the choice of positioning enemy radars anywhere on its virtual globe of Earth.
- It also allows us to take snapshots of radar and flight positions and send those images for processing later.
- Flight path data can be captured and flight paths can be traced, simulated and their realistic videos recorded.

MATLAB

- **MATLAB** (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language.
- The project employs MATLAB for image processing as matrix computations can be easily coded.
- In our case MATLAB is also used for computing the shortest path (using an *A-Star* library) on the processed image and generating the requisite STK Connect commands for path plotting in STK. More shall be discussed about this in the Implementation section.

Theory

A* Algorithm

A* is an informed search algorithm, meaning that it solves problems by searching among all possible paths to the goal, for the one that incurs the smallest cost (least distance travelled, shortest time, etc.).

We use this algorithm for UAV pathfinding. It is an extension of the Dijkstra's algorithm (also another pathfinding algorithm). A* achieves better performance by using *heuristics* to guide its search. It works on a graph of nodes or grid of points and plots an efficiently directed path between multiple points.

The geodetic locations of enemy radars are provided to the A* algorithm. The desired geodetic location of the UAV's start point and stop point are also fed to A* algorithm, which allows it to calculate the optimal route.

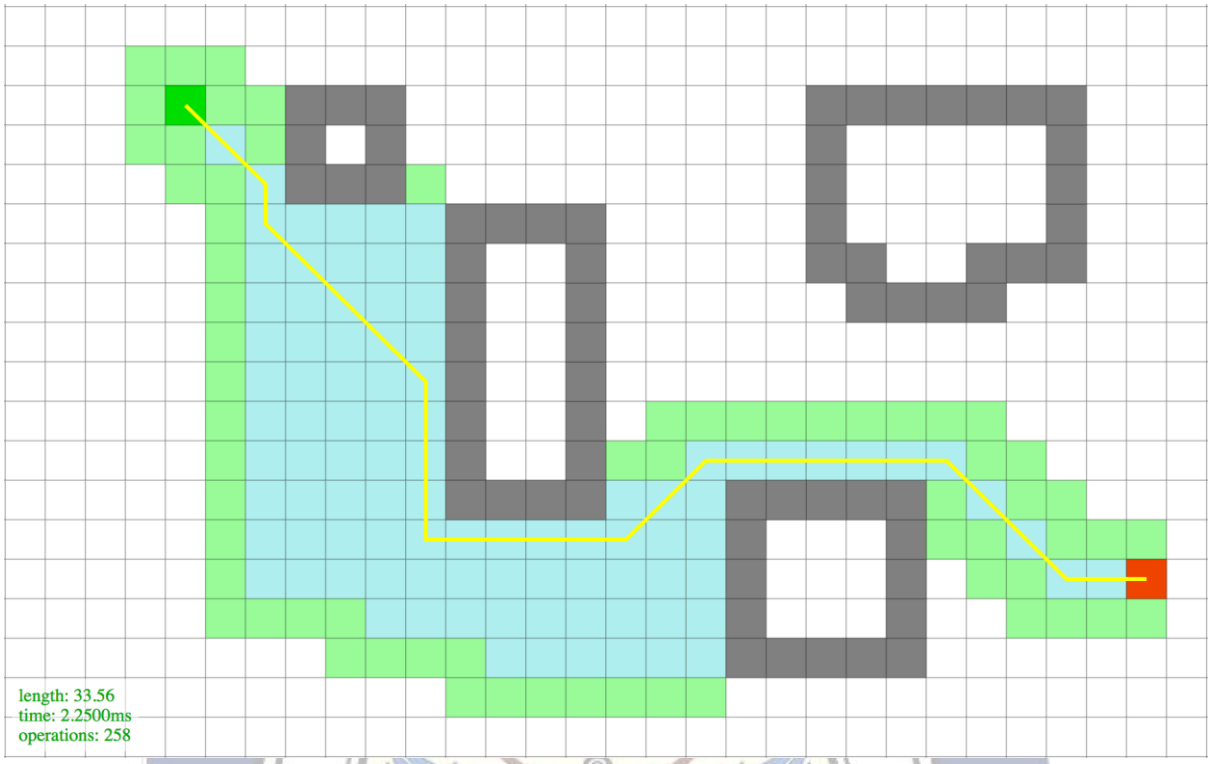
Heuristics

A* star consists of a heuristic function that gives it an estimate of the minimum cost from any vertex 'n' to the goal node. Intuitively, this mean that heuristics can be used to *control* A*'s behavior.

Since we are operating on a grid map where geodetic coordinates form grid points, we have used **Diagonal distance** as our A* heuristic.

Keeping diagonal movement in mind, this heuristic allows us to find the optimal path with the least computing time. It keeps constant track of the destination target and penalises the algorithm when it strays too far away from the destination node. This is useful to take care of the limited **fuel** supply on-board the UAV.

Example



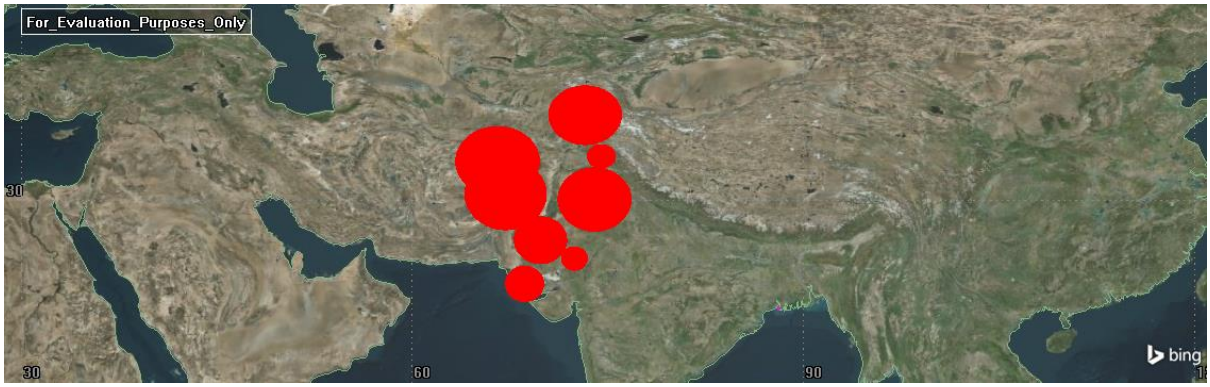
The above figure gives an example of A* pathfinding on a grid. The green box is the start point while the red box is the destination. The grey walls can be taken to be enemy radar boundaries. The algorithm avoids grey walls while following the shortest possible route.

Implementation

Generating snap

First, an STK scenario is set up using radars or area targets to model the situation. In this particular the area targets were chosen along the border, modeled as circular targets of radii 100km, 200km or 300km. To prepare the targets for preprocessing, they're filled with a distinctive color so that image segmentation can be used to isolate the targets.

Preprocessing



The generated snap is loaded into a MATLAB script, where it is pre-processed before any calculations.

Our areas of interest are the circles filled with red, as they correspond to enemy radar based areas. Therefore, we will extract the red circles from the snap.

First, we use the *impixel* MATLAB function to returns the RGB values of pixels in the specified snap. *impixel* displays the image specified and waits for you to select the pixels in the image using the mouse.

We click anywhere in the red enclosed area to extract its colour. *Impixel* then returns the red pixel values at the selected point. These pixel values are then used to specify colour ranges for each of the colour spectrums - Red, Green and Blue.

Since we chose a purely red pixel, the Red spectrum range will have max a value of 255 while the other spectrum ranges are centered about 0.

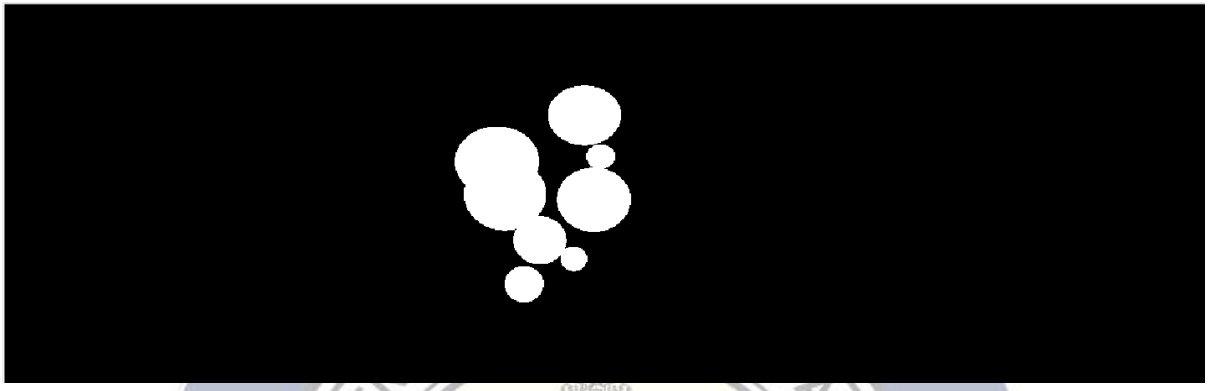
```
r_max = MAXC(1) + 1; %256

r_min = MINC(1) - 1; %254

g_max = MAXC(2) + 1; %1

g_min = MINC(2) - 1; %0

b_max = MAXC(3) + 1; %1

b_min = MINC(3) - 1; %0
```

Through a combination of Boolean and matrix algebra, a filter is created that lets in only pixel values that are present in the created colour ranges. The comparison filter outputs boolean integer values.

Therefore, all pixels that pass the filter have a value of 1 (True) while those do not have a value of 0 (False), thus creating a binary image.

```
OutImg = TestImg_R<r_max & TestImg_R>r_min & TestImg_G<g_max & TestImg_G>g_min & TestImg_B<b_max & TestImg_B>b_min;
```

The snap is stripped of any non-red pixel values and is converted to a grayscale image.



The grayscale image is inverted for the sake of convenience in further calculations. The inversion process looks like this.

```
OutImg = OutImg~=1;
```

The preprocessed snap is then saved to disk for further computations.

Map generation

The A* MATLAB library we use follows a particular convention for its occupancy grid. Any grid point with the value 1 represents an obstacle that cannot be navigated while 0 is a free path.

The circular areas represent our avoidable paths so they are to be designated with a value of 1 while the remaining areas are assigned with 0. The preprocessed snap is loaded into the Matlab script and now represents a **Map** that can be computed.

Start points are created.

```
%Start Positions

StartX=476;

StartY=106;
```

The Astar library accepts a matrix with the destination points specified in it.

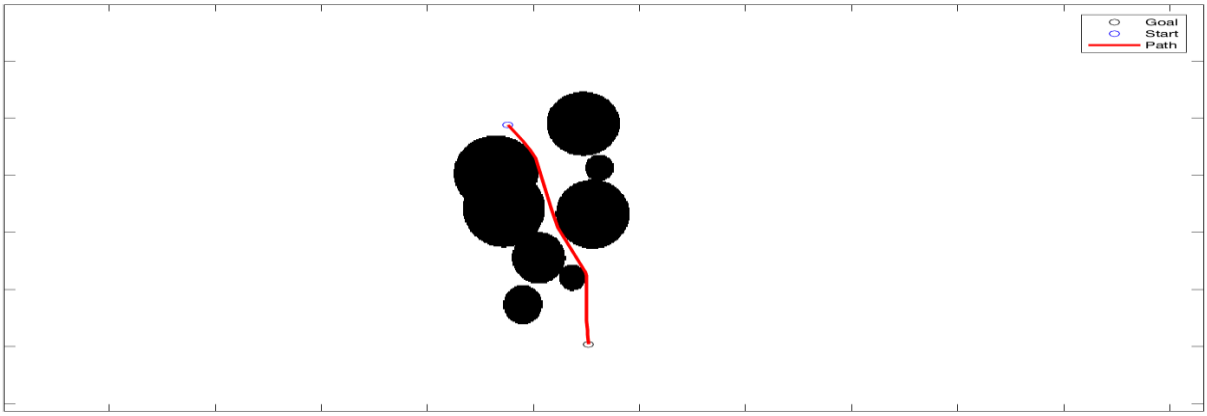
```
%Generating goal nodes, which is represented by a matrix. Several goals can be specified, in which case the pathfinder will find the closest goal.

%A cell with the value 1 represent a goal cell

GoalRegister=int8(zeros(size(rad_map2)));

GoalRegister(298,552)=1;
```

The PathFinder algorithm is then run on the Map.

```
OptimalPath=ASTARPATH(StartX,StartY,MAP,GoalRegister,Connecting_Distance)
```

OptimalPath consists of a matrix of grid point values. The grid point values are plotted on the generated map.

```
plot(OptimalPath(:,2),OptimalPath(:,1),'r','LineWidth',2);
```



Translation to STK Coordinates

The top left and bottom right coordinates are obtained from the STK interface. These are used as reference points to translate the Grid point values to geodetic coordinates in STK.

```
top_left = [42.64595 36.08668]; %Got this from STK cursor
bottom_right = [19.18550 110.76360]; %Got this from STK cursor
```

The top-left and bottom right Lat-Long coordinates are used to suitably scale and translate the grid points.

```
converted_points = [bottom_right(1)+(size(rad_map2,1)-OptimalPath(:,1)).*(top_left(1)-
bottom_right(1))/size(rad_map2,1) top_left(2)+(bottom_right(2)-top_left(2)).*OptimalPath(:,2)/size(rad_map2,2)];
```

The *HTML Connect* module is a library of string commands for STK that are easy to read, understand, and build. It allows custom applications to send command instructions to STK. The following algorithm writes STK Connect commands into a file on disk.

```
formatSpec = 'AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel %4.4f %4.4f 11000 257.22222\n';
for i = 1:size(converted_points,1)
   fprintf(STK_instr,formatSpec,converted_points(i,1),converted_points(i,2));
end
```

The text file is populated with looped waypoint generation commands. These commands provide path waypoints to the UAV STK object MyAircraft_Con.

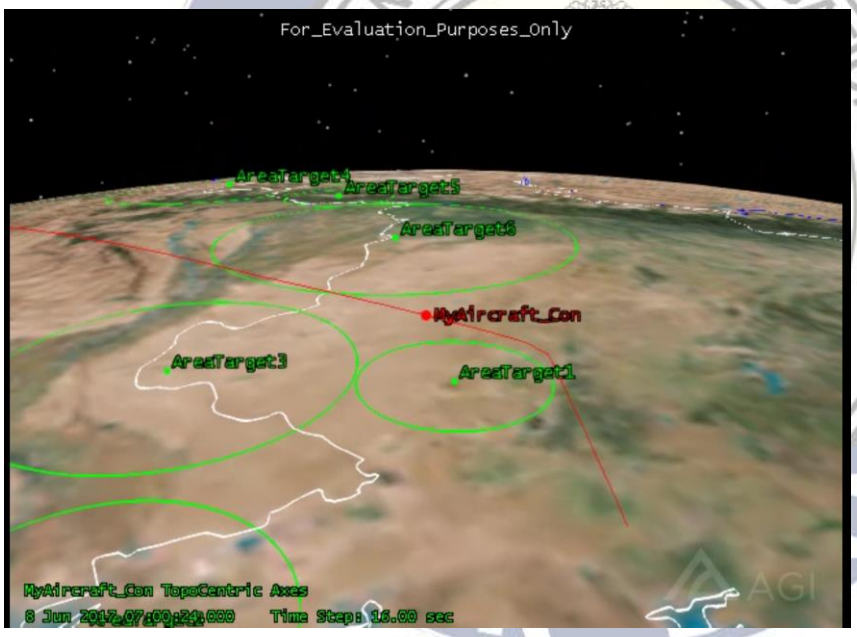
STK_instr.txt - Notepad

File Edit Format View Help

```
AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 23.0077 72.5338 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 23.3372 72.6658 11000
257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 23.8644 72.8639 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 24.1939 72.9960
11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 24.7211 73.1940 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 25.0506
73.3261 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 25.3801 73.4582 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel
25.5778 73.5242 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 25.9073 73.4582 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con
DetTimeAccFromVel 26.4345 73.1280 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 26.8299 72.8639 11000 257.22222AddWaypoint
*/Aircraft/MyAircraft_Con DetTimeAccFromVel 27.2253 72.5998 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 27.6207 72.3357 11000 257.22222
AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 28.0161 72.0716 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 28.4115 71.8075 11000
257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 28.6092 71.6754 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 29.1364 71.3453
11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 29.5977 71.0812 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel 30.0590
70.8171 11000 257.22222AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel
```

Simulation

Now, with the calculated waypoints in place. The UAV trajectory is simulated with the help of STK. Its Movie Timeline Tool allows us to capture a video of the entire simulation.

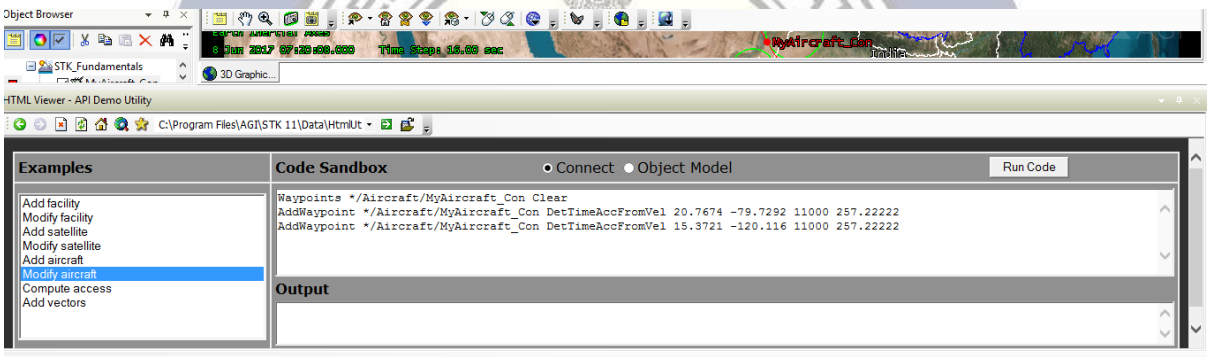


Here is a screenshot of the simulation in action.

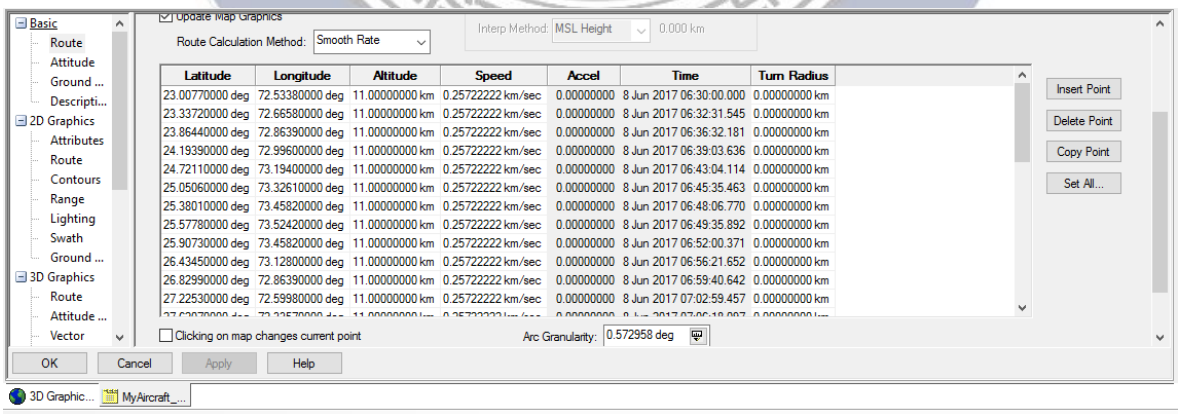
Results and discussion

After following the afore-proposed plan of action, we arrive at the following results

The generated HTML connect commands that are fed to STK are visible in the following image:



After adding the waypoints, the Aircraft is initialized as can be seen from the properties window:



The optimal path trajectory appears on the STK 3D window in red as follows:

After appropriately simulating the scenario, we see the following instances:

Therefore, a cost-optimal trajectory for the UAV was obtained using a fast pass algorithm (A*) and tested (and simulated) in STK. To assume a 3D scenario, scenarios with deceasing radii were simulated, and the best path was chosen. The use of A* algorithm in this case saved a lot of computational cost, by only checking neighboring pixels as opposed to all pixels in Djikstra's algorithm.

Hence, a working tool for generating optimized paths for UAVs was assembled using MATLAB and STK.

Future scope of work

Our current implementation of the A* library only uses **Diagonal distance** as a heuristic. The current model can be tweaked to incorporate many more factors.

- A **mobility** heuristic that provides a mobility cost based on the obstacles encountered. For eg, meteorologically difficult areas can be a given a much higher mobility cost compared to clear weather conditions.
- Any 3D or depth information can be incorporated into the heuristics, to make up for the **two dimensionality** of the grid map. For eg, low-flying UAVs can map mountainous terrain to a range of mobility costs.
- A **proximity** heuristic can be used to track swarm conditions in UAVs. UAVs can keep track of each other and ensure backup services while assigning costs to the proximal distances amongst themselves. This allows us to use **multiple** UAVs for route planning.
- **Aircraft Mission Modeller**, an STK module provides an enhanced method for modelling aircraft. With AMM, the aircraft's route is modelled by a sequence of curves parameterized by well-known performance characteristics of aircraft, including cruise airspeed, climb rate, roll rate, and bank angle. The trajectory generated can be used to simulate ultra-realistic conditions. The path along the trajectory can be infused with a **favourable path** heuristic to serve as a guideline for mission route planning.

Researchers think that the computational cost of using A* for highly-detailed and intensive 2D route planning would be prohibitive. However, these new heuristics substantially speed up the A * algorithm so that the run times are quite reasonable for large and complicated 2D grids. We also have the added advantage of A* giving us the most **optimal** path, which is quite essential when globally planning beforehand for static environments.

Conclusion

We believe we were successful in creating a client-side UAV mission planning tool that minimizes the distance heuristic. The tool works on the relatively computationally inexpensive “A star” algorithm, and it’s implemented in MATLAB.

The simulation was done in STK with satisfactory results.

Bibliography

Websites

- https://qiao.github.io/PathFinding.js/visual/
- http://help.agi.com/stk/index.htm#training/manuals.htm

Research papers

- Allen Z.J., et al, *Modelling of Unmanned Aerial Vehicle Deliveries in Populated Urban Areas for Risk Management*
- Nayak C.G., et al, *Optimized UAV Flight Mission Planning Using STK & A* Algorithm*
- Tsai Yi-Ju, et al, *Development of Flight Path Planning for Multirotor Aerial Vehicles*

Appendix

The following MATLAB code was written and used in this project.

```
clear

rad_map =
imread('C:\Users\Sammit\Downloads\DRDO_Confidential\processed.jpg');
rad_map = rad_map(:,:,1);
rad_map2 = int8(rad_map<127);
%%% Generating a MAP
%1 represents an object that the path cannot penetrate, zero is a free path

MAP = rad_map2;

%Start Positions
StartX=476;
StartY=106;

%Generating goal nodes, which is represented by a matrix. Several goals can
be speciefied, in which case the pathfinder will find the closest goal.
%a cell with the value 1 represent a goal cell
GoalRegister=int8(zeros(size(rad_map2)));
GoalRegister(298,552)=1;

%Number of Neighboors one wants to investigate from each cell. A larger
%number of nodes means that the path can be alligned in more directions.
%Connecting_Distance=1-> Path can  be alligned along 8 different direction.
%Connecting_Distance=2-> Path can be alligned along 16 different direction.
%Connecting_Distance=3-> Path can be alligned along 32 different direction.
%Connecting_Distance=4-> Path can be alligned along 56 different direction.
%ETC......

Connecting_Distance=8; %Avoid to high values Connecting_Distances for
reasonable runtimes.

% Running PathFinder
OptimalPath=ASTARPATH(StartX,StartY,MAP,GoalRegister,Connecting_Distance)
% End.

if size(OptimalPath,2)>1
figure(10)
imagesc((MAP))
    colormap(flipud(gray));
```

```
hold on
plot(OptimalPath(1,2),OptimalPath(1,1),'o','color','k')
plot(OptimalPath(end,2),OptimalPath(end,1),'o','color','b')
plot(OptimalPath(:,2),OptimalPath(:,1),'r','LineWidth',2)
legend('Goal','Start','Path')

else
     pause(1);
 h=msgbox('Sorry, No path exists to the Target!','warn');
 uiwait(h,5);
 end




showNeighboors=0; %Set to 1 if you want to visualize how the possible
directions of path. The code
%below are purley for illustrating purposes.
if showNeighboors==1
%



%2
NeigboorCheck=[0 1 0 1 0;1 1 1 1 1;0 1 0 1 0;1 1 1 1 1;0 1 0 1 0]; %Heading
has 16 possible allignments
[row col]=find(NeigboorCheck==1);
Neighboors=[row col]-(2+1);
figure(2)

for p=1:size(Neighboors,1)
  i=Neighboors(p,1);
       j=Neighboors(p,2);

     plot([0 i],[0 j],'k')
 hold on
 axis equal

grid on
title('Connecting distance=2')
end

%3
NeigboorCheck=[0 1 1 0 1 1 0;1 0 1 0 1 0 1;1 1 1 1 1 1 1;0 0 1 0 1 0 0;1 1
1 1 1 1 1;1 0 1 0 1 0 1;0 1 1 0 1 1 0]; %Heading has 32 possible
allignments
figure(3)
[row col]=find(NeigboorCheck==1);
Neighboors=[row col]-(3+1);

for p=1:size(Neighboors,1)
  i=Neighboors(p,1);
       j=Neighboors(p,2);

     plot([0 i],[0 j],'k')
 hold on
 axis equal
 grid on
```

```
title('Connecting distance=3')

end

%4
NeigboorCheck=[0 1 1 1 0 1 1 1 0;1 0 1 1 0 1 1 0 1;1 1 0 1 0 1 0 1 1;1 1 1
1 1 1 1 1 1;0 0 0 1 0 1 0 0 0;1 1 1 1 1 1 1 1 1;1 1 0 1 0 1 0 1 1 ;1 0 1 1
0 1 1 0 1 ;0 1 1 1 0 1 1 1 0];  %Heading has 56 possible allignments
figure(4)
[row col]=find(NeigboorCheck==1);
Neighboors=[row col]-(4+1);

for p=1:size(Neighboors,1)
  i=Neighboors(p,1);
       j=Neighboors(p,2);

     plot([0 i],[0 j],'k')
 hold on
 axis equal
grid on
title('Connecting distance=4')

end
%1
NeigboorCheck=[1 1 1;1 0 1;1 1 1];
figure(1)
[row col]=find(NeigboorCheck==1);
Neighboors=[row col]-(1+1);

for p=1:size(Neighboors,1)
  i=Neighboors(p,1);
       j=Neighboors(p,2);

     plot([0 i],[0 j],'k')
 hold on
 axis equal
grid on
title('Connecting distance=1')

end
end

%% Conversion algorithm
top_left = [42.64595 36.08668]; %Got this from STK cursor
bottom_right = [19.18550 110.76360]; %Got this from STK cursor

converted_points = [bottom_right(1)+(size(rad_map2,1)-
OptimalPath(:,1)).*(top_left(1)-bottom_right(1))/size(rad_map2,1)
top_left(2)+(bottom_right(2)-
top_left(2)).*OptimalPath(:,2)/size(rad_map2,2)];
%%
STK_instr = fopen('STK_instr.txt','a');

formatSpec = 'AddWaypoint */Aircraft/MyAircraft_Con DetTimeAccFromVel %4.4f
%4.4f 11000 257.22222\n';
for i = 1:size(converted_points,1)

fprintf(STK_instr,formatSpec,converted_points(i,1),converted_points(i,2));
end
```

```
fclose(STK_instr);
```

The MATLAB code for the A* algorithm is as follows:

```
function
OptimalPath=ASTARPATH(StartX,StartY,MAP,GoalRegister,Connecting_Distance)

%nNeighboor=3;
% Preallocation of Matrices
[Height,Width]=size(MAP); %Height and width of matrix
GScore=zeros(Height,Width);          %Matrix keeping track of G-scores
FScore=single(inf(Height,Width));    %Matrix keeping track of F-scores
(only open list)
Hn=single(zeros(Height,Width));      %Heuristic matrix
OpenMAT=int8(zeros(Height,Width));   %Matrix keeping of open grid cells
ClosedMAT=int8(zeros(Height,Width)); %Matrix keeping track of closed grid
cells
ClosedMAT(MAP==1)=1;                 %Adding object-cells to closed matrix
ParentX=int16(zeros(Height,Width));  %Matrix keeping track of X position
of parent
ParentY=int16(zeros(Height,Width));  %Matrix keeping track of Y position
of parent


%%% Setting up matrices representing neighboors to be investigated
NeighboorCheck=ones(2*Connecting_Distance+1);
Dummy=2*Connecting_Distance+2;
Mid=Connecting_Distance+1;
for i=1:Connecting_Distance-1
NeighboorCheck(i,i)=0;
NeighboorCheck(Dummy-i,i)=0;
NeighboorCheck(i,Dummy-i)=0;
NeighboorCheck(Dummy-i,Dummy-i)=0;
NeighboorCheck(Mid,i)=0;
NeighboorCheck(Mid,Dummy-i)=0;
NeighboorCheck(i,Mid)=0;
NeighboorCheck(Dummy-i,Mid)=0;
end
NeighboorCheck(Mid,Mid)=0;

[row, col]=find(NeighboorCheck==1);
Neighboors=[row col]-(Connecting_Distance+1);
N_Neighboors=size(col,1);
%%% End of setting up matrices representing neighboors to be investigated


%%%%%%%%% Creating Heuristic-matrix based on distance to nearest  goal node
[col, row]=find(GoalRegister==1);
RegisteredGoals=[row col];
Nodesfound=size(RegisteredGoals,1);

for k=1:size(GoalRegister,1)
    for j=1:size(GoalRegister,2)
```

```
        if MAP(k,j)==0
            Mat=RegisteredGoals-(repmat([j k],(Nodesfound),1));
            Distance=(min(sqrt(sum(abs(Mat).^2,2))));
            Hn(k,j)=Distance;
        end
    end
end
%End of creating Heuristic-matrix.

%Note: If Hn values is set to zero the method will reduce to the Dijkstras
method.

%Initializign start node with FValue and opening first node.
FScore(StartY,StartX)=Hn(StartY,StartX);
OpenMAT(StartY,StartX)=1;




while 1==1 %Code will break when path found or when no path exist
    MINopenFSCORE=min(min(FScore));
    if MINopenFSCORE==inf;
    %Failuere!
    OptimalPath=[inf];
    RECONSTRUCTPATH=0;
     break
    end
    [CurrentY,CurrentX]=find(FScore==MINopenFSCORE);
    CurrentY=CurrentY(1);
    CurrentX=CurrentX(1);

    if GoalRegister(CurrentY,CurrentX)==1
    %GOAL!!
        RECONSTRUCTPATH=1;
        break
    end

  %Remobing node from OpenList to ClosedList
    OpenMAT(CurrentY,CurrentX)=0;
    FScore(CurrentY,CurrentX)=inf;
    ClosedMAT(CurrentY,CurrentX)=1;
    for p=1:N_Neighboors
        i=Neighboors(p,1); %Y
        j=Neighboors(p,2); %X
        if CurrentY+i<1||CurrentY+i>Height||CurrentX+j<1||CurrentX+j>Width
            continue
        end
        Flag=1;
        if(ClosedMAT(CurrentY+i,CurrentX+j)==0) %Neiboor is open;
            if (abs(i)>1||abs(j)>1);
                % Need to check that the path does not pass an object
                JumpCells=2*max(abs(i),abs(j))-1;
                for K=1:JumpCells;
                    YPOS=round(K*i/JumpCells);
                    XPOS=round(K*j/JumpCells);

                    if (MAP(CurrentY+YPOS,CurrentX+XPOS)==1)
                        Flag=0;
                    end
```

```
                end
            end
             %End of  checking that the path does not pass an object

            if Flag==1;
                tentative_gScore = GScore(CurrentY,CurrentX) +
sqrt(i^2+j^2);
                if OpenMAT(CurrentY+i,CurrentX+j)==0
                    OpenMAT(CurrentY+i,CurrentX+j)=1;
                elseif tentative_gScore >= GScore(CurrentY+i,CurrentX+j)
                    continue
                end
                ParentX(CurrentY+i,CurrentX+j)=CurrentX;
                ParentY(CurrentY+i,CurrentX+j)=CurrentY;
                GScore(CurrentY+i,CurrentX+j)=tentative_gScore;
                FScore(CurrentY+i,CurrentX+j)=
tentative_gScore+Hn(CurrentY+i,CurrentX+j);
            end
        end
    end
end

k=2;
if RECONSTRUCTPATH
    OptimalPath(1,:)=[CurrentY CurrentX];
    while RECONSTRUCTPATH
        CurrentXDummy=ParentX(CurrentY,CurrentX);
        CurrentY=ParentY(CurrentY,CurrentX);
        CurrentX=CurrentXDummy;
        OptimalPath(k,:)=[CurrentY CurrentX];
        k=k+1;
        if (((CurrentX== StartX)) &&(CurrentY==StartY))
            break
        end
    end
end


end
```