We build from the ground up a computer simulation of a search of missing aircraft.

We are able to parse past 50 years of aircraft accident data, extracting the root cause of the accidents and their typical response (glide, free fall etc.). By parsing these data, we are able to construct distributions of probable crash radius with a relatively high confidence. We run our algorithms on three different types of aircrafts, G280 (small), B737-900ER (medium), and Airbus 380 (large).

We assumes three types of search agents, UAV, helicopter, and marine surface vessels with variable detector statistics.

We attempted to use an algorithm based on statistical local search to optimize the search plan for each agent [4].

Contents

1 Problem Restatement

Concerns over the disappearance of flight MH370 have rekindled interests on how to devise an optimal search plan to maximize our chance at finding the debris of an aircraft lost in a vast open body of water. Given the last known state of the aircraft, we wish to construct a probability distribution of the aircraft debris to and then a search plan that can assist the search-and-rescue teams in allocation of their efforts. Since we fear that the plane has been crashed, we assume no signals can be received from the lost plane, and we ignore the possibility that foul-play or navigation error could be the only cause for lost of contact with the aircraft.

The specific factors that should be taken into account in obtaining such optimal search plan include the variation in the type of crashed airplane and that of the search agents. Furthermore, we must first determine what constitutes the optimality of a search plan. Once the objective function is determined, the planning problem now turns into one of the optimization and we must determine an efficient and robust way to compute the search plan.

2 Acronyms and Terms

- **SAR**. Search-And-Rescue.
- **MTOW**. Maximum Take Off Weight.
- **USCG**. United States Coast Guards.
- **SAROPS**. Search and Rescue Optimal Planning System.
- **IMTS**. Informational Moving Target Search.
- **Location Density**. The probability density function of a target across a 2D domain.
- **Search Density**. The probability density function of search effort of a single search agent distributed across a 2D domain.

- **Search Effort**. A real number representing the total effort a single search agent applies. It is also the cost function that we try to optimize against. It is derived from the agent's velocity and effective sensor area.
- **(Detector) Range Law**. The probability density function of detection for a search agent on its observable area.

3 Assumptions and their Justifications

About the Search Domain and the Missing Aircraft

- **The search domain Ω is a 500km by 300km rectangle of unobstructed ocean.** This rectangle would cover the whole uncertainty range based on the last known state of the lost aircraft for an interval of 15 minutes to 1 hour based on INMARSAT's "Log-on Interrogation" old and newly recommended standards in light of the MH370 accident[**citation**].
- **The missing aircraft is assumed to be still in this domain at $t = 0$.** Although escaping the domain at a later time is allowed.
- **The SAR targets remain clustered.** This means that the targets always stay in the same cell on the discretized grid. This assumption is valid as long as the search is initiated close to the incident time and no serve weather condition causes disturbances.
- **There are buoyant indicator of target location at all time.** Since no underwater search is performed, we assume that either our SAR objective (e.g. survivors, life rafts, parts of crashed debris) remain buoyant throughout our search planning, or our sensor can detect signs of the objectives
- **The local trajectory of concern is straight**. In addition to the obvious smoothness arguments, it is always possible to apply a conformal transform on the entire search domain to obtain a solution based on a curved trajectory.

- **The trajectory from incident to crash is in a straight line.** Even in the worse case of gliding due to single engine failure, the average time from initial to of roughly 10 minutes based on our analysis. And during this time any banking maneuver is unlikely to cause significant deviation of the aircraft location density in terms of our discretized cells.

- **Hijacking or on-board navigation system only problems are not the cause of the incident**. This means that we assume the aircraft is crashed and assumes only the dynamics of the ocean and wind in our search domain. Although hijacking incidents account for nearly 20% of all accidents in past 50 years **Citation!**, the search agents in consideration (e.g. low flying aircrafts and surface vessel) are largely useless in finding a rouge cruising plane. Also see Section 1.

- **The missing aircraft can be accurately modeled as either G280, Boeing 737-900ER, or Airbus 380.** These three types of aircraft are well-known representatives of small private/business jets, medium range commercial flights, and large international flights. Cruise speed and other aircraft form factors (e.g. Lift to Drag ratio) are derived based on this assumption.

- **The crash radius of the aircraft is only a function of the cause of the incident and the type of the aircraft.** In addition, we assume that the historical distribution of the cause of the accidents is a reasonable prior for the current incident at hand, and it is it is invariant with respect to the type of the aircraft[1].

- **The crash radius calculations assumes constant air density and constant cruise altitude regarless of the types of aircraft involved.** This is not going to be a significant source of error in comparison to other assumptions.

[1]We do not have an aircraft aficionado at hand to sift through and separate the accident records based on size

- **Aircraft is operating at MTOW**. Assuming this is why we absolutely need an optimal search plan to find the cargos and passengers ASAP.

About Debris Drifting

- **The local drift direction and speed can be accurately modeled as constant within each cell.** Operationally, the resolution of the cells can be adapted to keep the assumption while also accommodating the actual drift data.
- **Debris drifting outside the search domain do not come back.** This assumption only introduces a small error on the probability of escaping the search domain.

About the Search Agents

- **All agents are commanded and controlled by the central planner at each update interval.** No command and control overhead is assumed for the sake of simplicity.
- **Agents arrive at the boundary of the search domain at $t = 0$.** Search agents are assumed to have arrived on the edge of the search domain at $t = 0$.
- **Unlimited bandwidth between search agent communications**. This is necessary from a planning perspective as to ignore the less than pertinent issues with sensor fusion and coordination. Moreover, this factor is more than likely fixed by the hardware.
- **Search agents are assumed to be either helicopters, UAVs, and surface vessels, all equipped with a bi-variate Gaussian/definite range law detection probability function [6] [7].** Moreover, all types of agents are assumed to h Although the problem only mentions "search planes," Marine SAR vessels are very commonly used.

- **Search agents admit a detector range law of a bivariate Gaussian as a function of their type, altitude, and speed.** Most sensors used for marine SAR (e.g. magnetometer or camera) behaves roughly according to this distribution. Furthermore, the exact sensor range law varies greatly[2] and we can always use actual measured data in operations. And finally, the detection statistics also follows Koopman's Random search formula [3].

- **The agents all have null probability of false alarm.** According to [2] and [4], this assumption is a standard one employed by the SAR planning industry. It is an reasonable assumption in that usually false alarms for SAR can be resolved quickly and locally (i.e. reexamining the targets are usually an easy task).

- **The search agents have zero knowledge of local ocean/wind drift information.** Although modern planning softwares used by professional SAR entities (e.g. SAROPS by U.S. Coast Guards) all have existing database to accommodate real-time ocean drifting [1], these data are often not precise, not to mention useless on our fictitious geography.

- **Operational range and refueling problems are neglected.** In operation, search agents can request refueling vehicles etc. and the time is not quite relevant.

- **Agents move in either the horizontal or vertical direction.** Although this is clearly not so realistic operationally, we can always approximate the real search trajectories better by increasing grid resolution.

- **Search agent trajectories are known and executed exactly.** This assumption is alliveated by the fact that we do not assume a definite range law.

[2]We were not able to find good data on this subject

4 Literature Review

The problem of finding the optimal search strategy of a target in a fixed region like an open body of water, known in the literature as the "Search and Screening Problem," had been extensively researched and analyzed by generations of researchers in the field of operational research. First aroused as a naval problem, Koopman [3] established the foundation on how to approach this problem. In his seminal papers, many reasonable assumptions as well as useful probability functions and formulas (e.g. the random-search formula) were devised and are still in use today. As the study for this problem matures, Stone analyzed the overall development of this field in his 1983 article [2], in which he explains the common assumptions made in practice and problems facing the continual development. He is also partially responsible to the construction of CASP (Computer Assisted Search Planning) algorithm that the USCG used to conduct ocean searching activities. He then went on to contribute constructing SAROPS, which will be compared with our algorithm in Section 6.4.

In [4], Kagan and Ben-Gal exposited more recent progress on this old problem. Specifically, many group-testing and Statistical Local Search (SLS) algorithms are explained in detail with proofs. And our optimal search plan algorithm for this paper is derived from their IMTS methods.

5 Criteria for Optimal Solution

In real life, the only success criterion is whether we can find the target and how long it takes us to do so. However, from the standpoint of search plan optimization, what we want is **to maximize the likelihood of detection given a fixed search effort**. In symbols, given $k(t)$ the search effort as our cost function, we want to maximize the Lagrangian $p(\Omega, t)\phi(k(t)) - \lambda k(t)$. Here Ω is our search domain as usual, p the location density of our target, ϕ the probability of detection, λ the Lagrange multiplier. From our assumptions, the Koopman Random-Search formula gives that $\phi(k(t)) = 1 - e^{-k(t)}$.

6 Debris Density Distribution and Search Planning

6.1 Crash Radius Distribution

Our first step is to model **the probable crash radius of the aircraft** in question. To do this, we noted the total number and root cause of commercial aircraft accidents since 1957[5]. From the cruise velocity and cruising altitude of the approximate size of the aircraft as well as the mode of response in reaction to the cause of the accident, we can categorize the data into 4 scenarios: **free fall, impact, gliding, and others**. See Figure 1.

For example, when facing an fire emergency on board, we assume that the airplane would descend in free fall with its initial cruise velocity since it is unlikely that the pilots can initiate significant correction maneuver due to chaos on board. Similarly for impact or collision (e.g. with birds), we assume a reduced initial velocity for the free fall. When only a single engine encounters failure, the aircraft is likely to be able to glide for a longer distance and hope for a smoother impact with the surface of the ocean. And such glide distance is obtained based on the lift to drag ratio of the aircraft frame. The others category includes all the other accidents such as airframe/structural problems, except that hijacking and navigation error only incidents were excluded per assumptions. See Section 3.

Our model can of course improve with a more detailed analysis of previous accidents. However, since we are exaggerating the probable crash radius into a single distribution, we can *always* substitute a better prior into our optimization algorithm to obtain a more realistic optimal search plan.

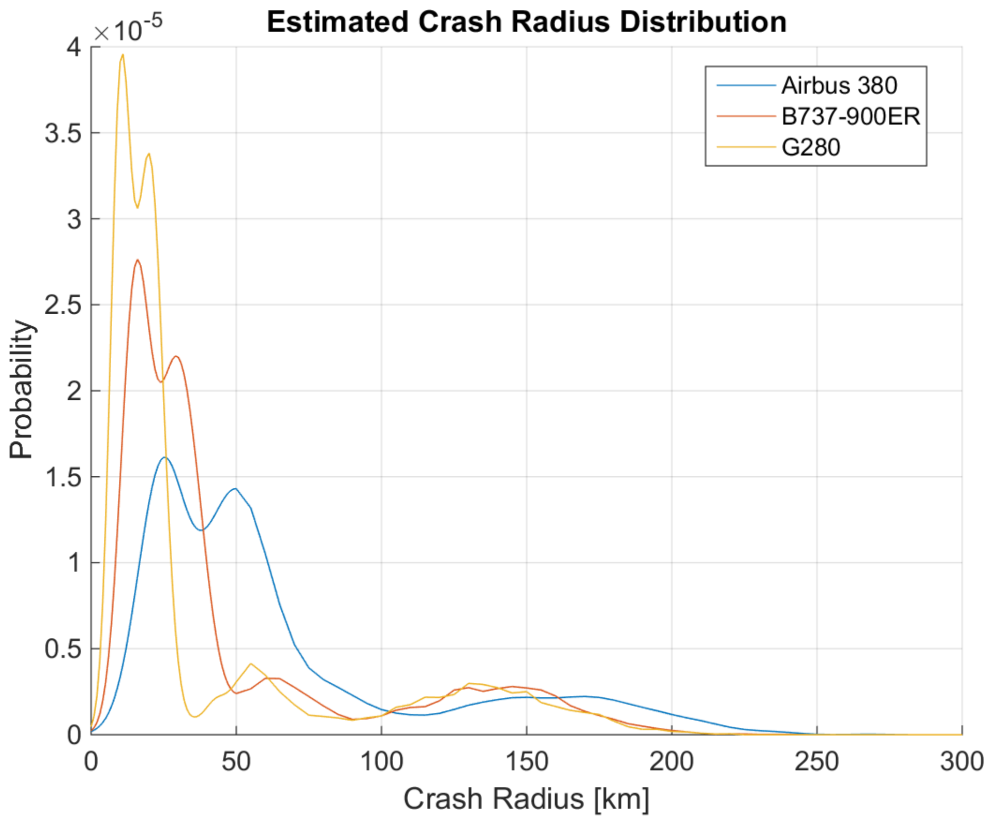


Figure 1: Estimated Crash Radius Distribution of three types of missing aircraft. From left to right, the peaks represents the impact, free fall, others, and glide categories.

6.2 Location Density in Search Domain

Once we obtained the distribution of crash distances, we can calculate the location density at each point away from the intended trajectory (the horizontal line at the bottom in Figure 2) by (Notations explained on next page)

$$P[crash\,at\,*] = \int P[crash\,radius = \tilde{R}(\tilde{r})] \cdot P[Deviation = \theta(\tilde{R}(\tilde{r}))]\,d\tilde{r}$$

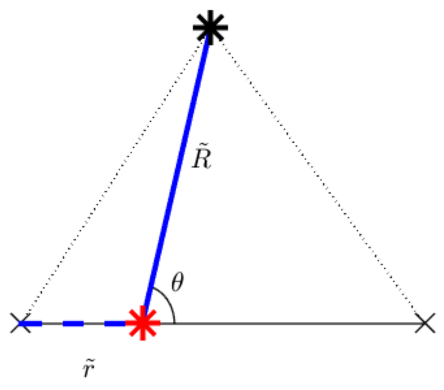


Figure 2: Illustration of the accident-site geometry. The left $\times$ represents the last known location, the right $\times$ first checkpoint where the aircraft was not found. The incident site is marked by the red $\ast$, while the crash site is marked by the black $\ast$.

Finally, the probability that the aircraft crashed within one cell is obtained through the double integral.

6.3 Search Agents and the Search Effort

The model for each of the three types of the agent is reduced to two parameters, agent speed and 1σ detector range. From the range we can find the area such that we achieve a desired cumulative probability of detection (say 95%) basing on the bi-variate Gaussian detector range law.

Next the search effort can be calculated as $k(t) = |A|t = 2rV$, where r is the radius of the previously mentioned observable area, V the speed of the agent, and t the time steps. Using Koopman's random search formula (which is justified basing on our assumptions in [2]), we get the probability of detection to be $\phi(t) = 1 - e^{2rV}$.

6.4 Comparison to U.S.C.G. SAROPS

In comparison to the direct Monte Carlo / particle filter approach employed by the U.S. Coast Guards' SAROPS system, our search planning algorithm

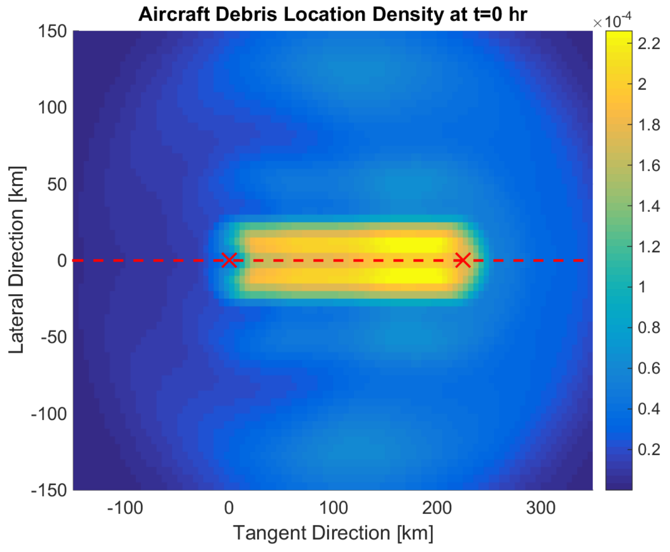
- **Do not employ multiple scenarios.** SAROPS is capable of separate different scenarios of accident into independent draws. However, according to our assumptions and the problem statement, we are warranted to ignore unlikely cases (e.g. hijacking) See 3
- **Do not use a particle filter.** SAROPS uses a particle filter to represent the probability density functions. Although more fashionable and advanced, it requires more computing power and is roughly equivalent to our representation given a fine enough grid resolution.

7 Comparison to a Parallel Search Plan

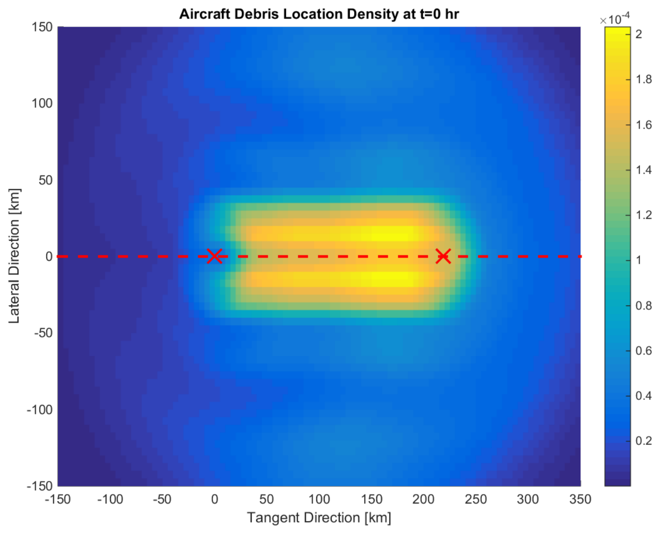
Naively, one can also devise the classic parallel search plan where the search agents travel in parallel lines that each agent's observed area overlaps. Such a plan is analyzed in fair amount of detail by Stone in [2]. In our crude simulation, we assume that the trajectory of the search agents are known and executed exactly

8 Experimental Setup

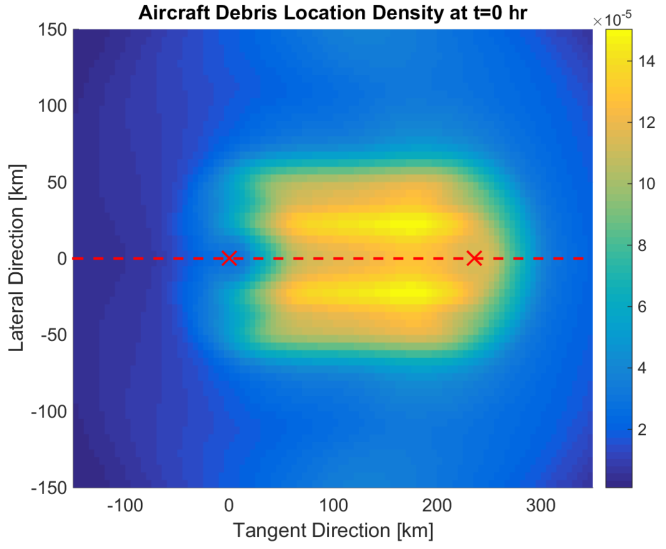
By changing the type of lost plane, we can notice the difference in prior distribution as demonstrated in Figure 3 below.



(a)



(b)



(c)

Figure 3: Prior Distributions. From top to bottom, (a) G280; (b) Boeing 737-900ER; (c) Airbus 380.

9 Results

......

We did not have enough time to reach desirable results.

10 Sensitivity to Parameters

By refining our search domain grid, we receive a varying level of discretization error as exemplified by the probability of aircraft escape given a uniform search plan. See Figure 4 below.

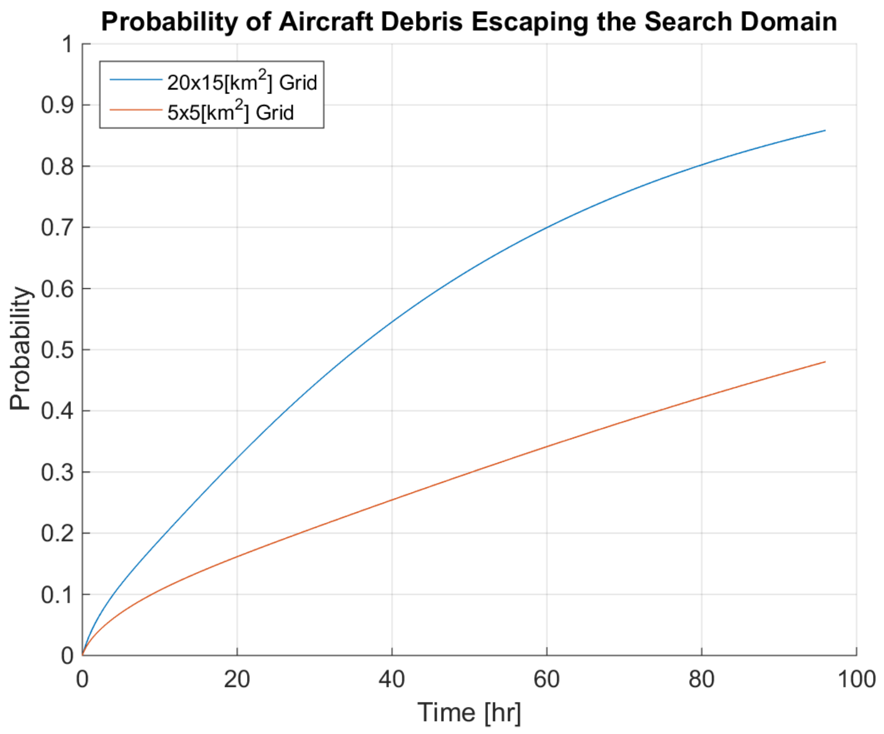


Figure 4: Decrease in grid cell resolution causes higher numerical leakage

In Figure 4, it is evident that reduction in cell resolution causes the Markov drifting simulation leakage.

11 Strengths and Weaknesses

Strengths:

- **Intuitive to understand**.
- **Requires no input data for the ocean drift**. Saves lots of measurement and preparation time.
- **Can be transformed into desired terrain with ease**. Conformal mapping.

Weaknesses:

- **Require a fine grid resolution for realistic results**. Excessive discretization can cause unwanted error in total escape probability. See Section 10 and Figure 4.
- **very simplified drift modeling**. Description.

12 Conclusion

- **Need a more accessible database that can model airplane crash radius and ocean drifts.**. Why the data says so.
- **Make Sure to form a team of search agents that would not give up the search half way**. From our personal experience, we conclude that one person contributing is $\lll$ two person $\lll$ three person, etc.

References

[1] Kratzke, T.M.; Stone, L.D.; Frost, J.R. "Search and Rescue Optimal Planning System" *Information Fusion (FUSION), 2010 13th Conference on* **DOI:10.1109/ICIF.2010.5712114** (2010) 1-8

[2] Stone, L. D. "The Process of Search Planning: Current Approaches and Continuing Problems." *Operations Research* **31(2)** (1983): 207-233.

[3] Koopman, B.O. "Search and Screening." *Operation Evaluation Research Group Report* **56** (1946) Center for Naval Analysis; See also: "The Theory of Search, I-III" *Operations Research* (1956) **4** 324-246; (1956) **4** 503-531; (1957) **5** 613-626.

[4] Kagan, E.;Ben-Gal, I. "Probabilistic Search for Tracking Targets: Theory and Modern Applications." *WILEY* (2013) 19-313.

[5] `http://Aviation-safety.net`

[6] U.S. Coast Guard Acquisition Directorate. "Aviation Fact Sheet." `http://www.uscg.mil/acquisITION/programs/pdf/air.pdf`

[7] Damen Shipyards, "NH 1816: a New Type of Search and Rescue boat for KNRM" *Maritime by Holland* (2014) **63** 41-43 `http://products.damen.com/~/media/Products/Images/Clusters%20groups/High%20Speed%20Crafts/Search%20and%20Rescue%20Vessel%201906/Documents/Maritime_by_Holland_2014_Magazine.ashx`

Appendix: Computer Code

main.m :

```
clc; clear; close all;
% constants
gE = 9.81; %m/s2
%% plane specs (B737-900ER; G280; A380)
% Cruise speed (m/s)
B737.Vc = 243;
G280.Vc = 250;
A380.Vc = 262;
% crash distance
% fire, collision, glide, other
A380.R = [24 50 160 80]*1e3; %m
B737.R = [15 30 140 63]*1e3; %m
G280.R = [10 20 135 56]*1e3; %m

%% Assumptions/parameters

% target aircraft make
ACcase = 3;
if ACcase == 1
    AC = B737; acname = 'B737-900ER';
elseif ACcase == 2
    AC = A380; acname = 'Airbus380';
else
    AC = G280; acname = 'G280';
end
% remaining fuel ratio at incident
Fr = .5;
% communication interval/distance
interval = 15*60; %s
rint = AC.Vc*interval; %m
% continuous probability (rtil) riemann sum resolution
Nrtil = 50;
% grid resolution
GRIDcase = 1;
if GRIDcase == 1
    N1grid = 100;
    N2grid = 60;
else
    N1grid = 25;
    N2grid = 20;
end
% # of 1D quadrature points (use even number for symm)
Nquad = 2;
% search domain bounds [-x1 +x1 -x2 +x2]
bdry = [-150 350 -150 150]*1e3; %m
% reversing probability param
s = .05; q = 1/pi-2*s;
```

```
% average debris drift speed
dV = 10; %m/s

%% incident to crash range pdf

% fire, collision, glide, other
Ncrash = [1406, 1901, 901, 498];

% stdev of the statistics (guess)
Rstdev = [.2 .2 .2 .2];

Rhist = [];
for i=1:numel(Ncrash)
    Rhist = [Rhist; randn(Ncrash(i),1)*Rstdev(i)*AC.R(i) + AC.R(i)];
end

% smoothed profile + visuialization
[PR,R]=ksdensity(Rhist,[0:1e3:50e3 55e3:5e3:300e3 310e3:10e3:500e3]);
save(['../data/' acname '_CrashRadius.mat'],'R','PR');

%% continuous probability at x=(x1,x2)
    rtil = linspace(0,rint,Nrtil);
    Rtil = @(x) sqrt(sum((repmat(x,1,Nrtil)-[rtil;zeros(size(rtil))]).^2));
    theta = @(x) abs(atan2(x(2),x(1)-rtil));
    ptheta = @(x) q + theta(x)*(s-q)/pi;
    pdfx = @(x) interp1(R,PR,Rtil(x)).*ptheta(x);
    %% discritized probability at cell (eu,ev)

    % pre-compute gauss-legendre points and weights
    [u,wu] = gaussquad(Nquad);
    [v,wv] = gaussquad(Nquad);
    Nu = nodefun(u);
    Nv = nodefun(v);

    % total domain area
    Agrid = (bdry(2)-bdry(1))*(bdry(4)-bdry(3)); %m2
    % grid pt construction
    S = Surface(N1grid,N2grid,bdry);
    P = zeros(S.numelements);

    %% loops
    % note the vertical symmetry!!!
    wv = wv*2;
    for ev=1:S.numelements(2)/2
        for eu=1:S.numelements(1)
            for l=1:Nquad/2
                for k=1:Nquad
                    % pullback quadrature pts coordinate
                    [x,y] = S.coords(eu,ev,Nu(:,k),Nv(:,l));
                    Ptemp = sum(pdfx([x;y]))*rint/Nrtil;
                    P(eu,ev) = P(eu,ev) + wu(k)*wv(l)*Ptemp;
                end
            end
        end
        P(:,S.numelements(2)-ev+1) = P(:,ev);
    end
    %% renormalize and save data
    P = P / sum(P(:));
    save(num2str([ACcase GRIDcase],'../data/prior%d%d.mat'),'P','S');

    %% crash probability distribution graph

    Splot = Surface(N1grid,N2grid,bdry/1e3);
    figure(); hold all; grid on;
    plottwoform(Splot,P,3); colorbar;
    xlabel('Tangent Direction [km]'); ylabel('Lateral Direction [km]');
    title('Aircraft Debris Location Density at t=0 hr');
    hold all;
    traj = plot3([-1e6 0 rint 1e6]/1e3, [0 0 0 0], [1 1 1 1],'rx--');
    set(traj,'linewidth',2,'markersize',15)
    saveas(gcf,num2str(GRIDcase,['../figures/' acname '_PriorDistribution%d.png']));

%% drift/diffusion simulation
    Tsim = 96*3600; %s

    PP = P;
    % update interval that is appropriate
    % i.e. allow only single cell diffusion given grid resolution
    if GRIDcase == 1
        dt = .1*60*60; %s
        Nsim = Tsim/dt; %steps
    else
        dt = .4*60*60; %s
        Nsim = Tsim/dt; %steps
    end
    [Pmove,~] = driftP(S,dt,dV);

    % propogation steps
    tVec = (0:dt:Tsim)/3600; %hr
    % Probability of escape at t
    qt = zeros(Nsim+1,1);

    for t = 1:Nsim
        [PP,qt(t+1)] = next(S,PP,Pmove);
    end
    save(num2str([ACcase GRIDcase],'../data/nosearchEsc%d%d.mat'),'tVec','qt');
    %% Location density if no search initiates
    figure(); hold all; grid on;
    plottwoform(Splot,PP,3); colorbar;
    xlabel('Tangent Direction [km]'); ylabel('Lateral Direction [km]');
    title(num2str(Tsim/3600, 'Aircraft Debris Location Density at t=%d hr'));
    saveas(gcf,num2str(GRIDcase,['../figures/' acname '_NoSearchDistribution%d.png']));
    %% Graph of escape probability over time
    figure(); hold all; grid on;
    plot(tVec,qt,'k-');
    xlabel('Time [hr]'); ylabel('Probability');
    title('Probability of Aircraft Debris Escaping the Search Domain');
    saveas(gcf,num2str(GRIDcase,['../figures/' acname '_NoSearchEscape%d.png']));
```

```
%% Search Agent Data

% assume 95% detection range as effective area
ncdf = @(sig,Rd) normcdf(Rd,0,sig)-normcdf(-Rd,0,sig);
cdf2sig = @(Rd) fminsearch(@(sig) abs(ncdf(sig,Rd)-.95),Rd/2);

% Marine Vessel (Damen SAR vessel 1816/1906 range= 600km+)
MV.Vs = 15.5; %m/s
MV.Rdetect = 1e3; %m
MV.FA = 0;
MV.sig = cdf2sig(MV.Rdetect); %m

% UAV (Hermes)
UAV.Vs = 49; %m/s
UAV.Rdetect = 5e3; %m
UAV.alt = 5e3; %m
UAV.FA = .05;
UAV.sig = cdf2sig(UAV.Rdetect); %m

% Helicoptor (USCG Dolphin MH65C range= 280km+)
heli.Vs = 82; %m/s
heli.Rdetect = 5e3; %m
heli.alt = 5.5e3; %m
heli.FA = .05;
heli.sig = cdf2sig(heli.Rdetect); %m

%% search agent initial states
% number of agents on one side
Nagent = 100;

% Initial position

% Ps0 = [S.xnodes(S.getglobalboundarynodes) ...
%     S.ynodes(S.getglobalboundarynodes)];

% Detection Probability are assumed to be normal
A =  mvncdf(x1r,x2r,xs,sig);
%% Distributed Search Plan (edge first and chase the highest cell)
    Tsim = 96*3600; %s

    PP = P;
    % update interval that is appropriate
    % i.e. allow only single cell diffusion given grid resolution
    if GRIDcase == 1
        dt = .1*60*60; %s
        Nsim = Tsim/dt; %steps
    else
        dt = .4*60*60; %s
        Nsim = Tsim/dt; %steps
    end
    [Pmove,Ncell] = driftP(S,dt,dV);

    % propogation steps
    % Probability of escape at t
    qt = zeros(Nsim+1,1);

    for t = 1:Nsim
        [PP,qt(t+1)] = next(S,PP,Pmove);
    end
    %% Location density
    figure(); hold all; grid on;
    plottwoform(Splot,PP,3); colorbar;
    xlabel('Tangent Direction [km]'); ylabel('Lateral Direction [km]');
    title(num2str(Tsim/3600, 'Aircraft Debris Location Density at t=%d hr'));
    saveas(gcf,['../figures/' acname '_NoSearchDistribution.png']);
    %% Graph of escape probability over time
    tVec = (0:dt:Tsim)/3600; %hr
    figure(); hold all; grid on;
    plot(tVec,qt,'k-');
    xlabel('Time [hr]'); ylabel('Probability');
    title('Probability of Aircraft Debris Escaping the Search Domain');
    saveas(gcf,['../figures/' acname '_NoSearchEscape.png']);
%% Distributed Search Plan (center first)


%% concentrated "single" agent Search Plan (steepest descent)
```

drift.m :

```
function [Pmove,Ncell] = driftP(S,dt,dV)

%% assumes zero knowledge of local drift direction and speed.

% average debris drift speed
ds = dV*dt; %m
% standard deviation (GUESS!)
sigma_s = .5*ds; %m
% grid size
dx1 = mean(diff(unique(S.xnodes))); %m
dx2 = mean(diff(unique(S.ynodes))); %m
% nextcell distance (approximate by diam)
dd = sqrt(dx1^2+dx2^2)/2;

% range of motion in cells (1sigma)
Nmove = ceil((ds+sigma_s)/dd);
Pmove = zeros(Nmove,1);
Pmove(1) = normcdf(dd,ds,sigma_s);
for i=2:Nmove-1
    Pmove(i) = normcdf(dd*i,ds,sigma_s) - Pmove(i-1);
end
Pmove(end) = 1 - sum(Pmove(1:end-1));
Ncell = 8*(1:Nmove)-8;
Ncell(1) = 1;
Pmove = Pmove(:)./Ncell(:);
```

```
end
```

next.m :

```
function [P,q] = next(S,P,Pmove)

Nmove = numel(Pmove);
Ppadded = zeros(S.dim+Nmove-1);
% Ppadded(Nmove:(end-Nmove+1),Nmove:(end-Nmove+1)) = P;

for i = 1:size(P,1)
    for j = 1:size(P,2)
        Pcell = P(i,j);
%          for prop = 1:Nmove
            Ppadded(i+[0 2],j+(0:2)) = Ppadded(i+[0 2],j+(0:2)) + Pmove(2)*Pcell;
            Ppadded(i+1,j+[0 2]) = Ppadded(i+1,j+[0 2]) + Pmove(2)*Pcell;
            Ppadded(i+1,j+1) = Ppadded(i+1,j+1) + Pmove(1)*Pcell;
%          end
    end
end

P = Ppadded(Nmove:(end-Nmove+1),Nmove:(end-Nmove+1));
q = 1 - sum(P(:));
end
```

Surface.m :

```
classdef Surface
    %SURFACE Summary of this class goes here
    %   Detailed explanation goes here

    % x-,y- coordinates of nodes
    properties
        xnodes
        ynodes
    end

    % number of mesh-primitives
    properties
        dim
        numelements
        numnodes
        numedges
        numfaces
    end

    methods

        % constructor
        function S = Surface(varargin)
        switch nargin
            case 0
                n = 1; m = 1;
                [S.ynodes,S.xnodes] = meshgrid(linspace(0,1,n+1),linspace(0,1,m+1));

            case 2
                n = varargin{1};
                m = varargin{2};
                [X,Y] = meshgrid(linspace(0,1,n+1),linspace(0,1,m+1));
                S.xnodes = X'; S.ynodes = Y';

            case 3
                n = varargin{1};
                m = varargin{2};
                % boundary distance [-x1 +x1 -x2 +x2]
                bdry = varargin{3};
                [X,Y] = meshgrid(linspace(bdry(1),bdry(2),n+1),...
                                 linspace(bdry(3),bdry(4),m+1));
                S.xnodes = X'; S.ynodes = Y';
            case 4
                n = varargin{1};
                m = varargin{2};
                S.xnodes = varargin{3};
                S.ynodes = varargin{4};

            otherwise
                error('To many input arguments');
        end

        % set dependent values
        S.dim = [n+1,m+1];
        S.numelements = [n,m];
        S.numnodes    = prod([n+1,m+1]);
        S.numedges(2) = prod([n,m+1]);
        S.numedges(1) = prod([n+1,m]);
        S.numfaces    = prod([n,m]);
    end

    % compute jacobian 2 by 2 matrix
    function J = jacobian(S,eu,ev,Nu,Nv,dNu,dNv)
        % input: Surface S, element (eu,ev) and
        % basis functions Nu, Nv and derivatives dNu,dNv

        % active nodes in element (eu,ev)
        X = S.xnodes(eu:eu+1,ev:ev+1);
        Y = S.ynodes(eu:eu+1,ev:ev+1);

        % compute Jacobian
        J = [dNu' * X * Nv, Nu' * X * dNv;
             dNu' * Y * Nv, Nu' * Y * dNv];
    end

    % compute coordinates of mapping
```

```
function [x,y] = coords(S,eu,ev,Nu,Nv)
    % input: Surface S, element (eu,ev) and
    % basis functions Nu, Nv

    % active nodes in element (eu,ev)
    X = S.xnodes(eu:eu+1,ev:ev+1);
    Y = S.ynodes(eu:eu+1,ev:ev+1);

    % compute Jacobian
    x = Nu' * X * Nv;
    y = Nu' * Y * Nv;
end

% get global nodenumbers on the boundary
function gnn = getglobalboundarynodes(S)
    n = S.dim(1); m = S.dim(2);

    temp = reshape([1:S.numnodes],n,m);
    gnn = setdiff(temp,temp(2:end-1,2:end-1));
end

% get global edgenumbers on the boundary
function gnn = getglobalboundaryedges(S)
    n = S.dim(1); m = S.dim(2);

    temp = reshape([1:S.numedges(1)],n,m-1);
    gnn{1} = setdiff(temp,temp(2:end-1,:));

    temp = reshape([1:S.numedges(2)],n-1,m) + S.numedges(1);
    gnn{2} = setdiff(temp,temp(:,2:end-1));
    gnn = [gnn{1}(:);gnn{2}(:)];
end

% get global nodenumbers for element (eu,ev)
function gnn = getglobalnodenumber(S,eu,ev)
    % global numbering in subs
    [nnu,nnv] = meshgrid(eu:eu+1,ev:ev+1);

    % global numbering in linear indexing
    gnn = sub2ind(S.dim, nnu', nnv');
end

% get global edgenumber for element (eu,ev) in z-dir
function gnn = getglobaledgenumber(S,eu,ev,z)

    % direction 1
    if z==2
        [nnu,nnv] = meshgrid(eu,[ev:ev+1]');        % global numbering in subs-forma
        gnn = sub2ind([S.dim(1)-1,S.dim(2)], nnu', nnv') + S.numedges(1); % global

    % direction 2
    elseif z==1
        [nnu,nnv] = meshgrid(eu:eu+1,ev);        % global numbering in subs-format
        gnn = sub2ind([S.dim(1),S.dim(2)-1], nnu', nnv');     % global numbering in
```

```
            else
                error('Wrong input arguments');
            end
        end

        % get global facenumbers for element (eu,ev)
        function gnn = getglobalfacenumber(S,eu,ev)
            % global numbering in linear indexing
            gnn = sub2ind(S.dim-1, eu, ev);
        end

        % plot surface
        function plot(S)

            % Create figure
            figure1 = figure;

            % Create axes
            axes1 = axes('Parent',figure1,'PlotBoxAspectRatio',[434 342.3 684.6],...
            'DataAspectRatio',[1 1 1]);
            grid(axes1,'on');
            hold(axes1,'on');

            % Create surf
            surf(S.xnodes,S.ynodes,zeros(S.dim),'Parent',axes1,'FaceLighting','none',...
                'EdgeLighting','flat',...
                'MarkerSize',20,...
                'Marker','.',...
                'LineWidth',2,...
                'FaceColor','none');
            hold off;
        end


    end



end
```