Contents


```
function [d, tFull, xFull, cmdFull] = PlotUAVObstacleCourse( courseDataFile )
```

```
% Plot the obstacle course for UAVs to fly through.
%
% USAGE:
%   PlotUAVObstacleCourse( courseDataFile )
%
% INPUTS:
%   courseDataFile  (:)   String name of the .mat file that has all of the
%                         UAV obstacle course info.
%
% OUTPUTS:
%   none
%
```

Load the mat-file

```
% initialize variables
cuboid = struct(); distToOtherTargets = []; hoopIR = []; hoopOR = []; hoopPsi = []; hoopTheta = [];
hoopX = []; hoopY = []; hoopZ = []; insideCuboid = 0; nCuboids = 0; nDone = 0; nHoops = 0; nTargets = 0; targetPos = [];
targetScore = []; thisPos = []; xLim = []; xS = 0; yLim = []; yS = 0; zLim = []; zS = 0;

if nargin<1
  courseDataFile = 'UAVCourseData_1';
end
if ~isstr(courseDataFile)
  error('Provide the NAME (as a string in single quotes) of the .mat file with the UAV course data.')
  end
load(courseDataFile);
```

Draw all of the hoops and cuboids.

```
d.fig=figure('name','UAV Obstacle Course','Position',[10 400 1400 1000],...
  'Color','k');
d.ax=axes('parent',d.fig,'xcolor','w','ycolor','w','zcolor','w','color','k');

% bounding region
[v,f] = Cuboid( xLim(1),yLim(1),zLim(1), ...
  diff(xLim), diff(yLim), diff(zLim) );
d.hb = patch('faces',f,'Vertices',v,'facecolor','none','edgecolor','c');
grid on, hold on, axis equal
d.ax.XLim=xLim+[-1 1]*diff(xLim)/20;
d.ax.YLim=yLim+[-1 1]*diff(yLim)/20;
d.ax.ZLim=zLim+[-1 1]*diff(zLim)/20;
view(-20,20), rotate3d on
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')

% hoops
for j=1:nHoops
  [v,f] = torus(40,30,hoopOR(j),'R',hoopIR(j));
  m1 = RotMat(pi/2,1);
  m2 = RotMat(hoopPsi(j),3);
  m3 = RotMat(hoopTheta(j),2);
  v = (m3*m2*m1*v')'+[hoopX(j),hoopY(j),hoopZ(j)];
```

```
  d.hh(j) = patch(d.ax,'faces',f,'Vertices',v,'facecolor',rand(1,3),'edgecolor','none',...
    'SpecularColorReflectance',.7);
end

% cuboids
for j=1:nCuboids
  cx = cuboid(j).pos(1);
  cy = cuboid(j).pos(2);
  cz = cuboid(j).pos(3);
  L = cuboid(j).dims(1);
  W = cuboid(j).dims(2);
  H = cuboid(j).dims(3);
  [v,f] = Cuboid(cx,cy,cz,L,W,H);
  m1 = RotMat(cuboid(j).phi,3);
  m2 = RotMat(cuboid(j).theta,2);
  pos = cuboid(j).pos;
  v = (m2*m1*(v'-pos))+ pos;
  d.hc(j) = patch(d.ax,'faces',f,'Vertices',v','facecolor',rand(1,3),'edgecolor',[.1 .1 .1],...
    'SpecularColorReflectance',.9);
end

% targets
% for j=1:nTargets
% wpOrder = [2 3 5]; % Order of waypoints
% wpOrder = [2 3 5];
[~,wpOrder] = sort(targetPos(3,:),'descend'); % Fly to points in descending order by altitude
testWps = targetPos(:,wpOrder);
for j=1:size(testWps,2)
  % d.hTgt(j) = plot3(d.ax,targetPos(1,j),targetPos(2,j),targetPos(3,j),'y.','markersize',25);
  d.hTgt(j) = plot3(d.ax,testWps(1,j),testWps(2,j),testWps(3,j),'y.','markersize',25);
end

light
light('position',[xS yS zS])
lighting phong
material metal

% Initialize state vector
V = 5;      % true airspeed (m/s)
gamma = 0;  % air relative flight path angle (rad)
psi = 0;    % air relative flight heading angle (rad)
x = 50;     % east position (m)
y = 200;    % north position (m)
h = 35;     % altitude (m)
Tbar = 0;   % normalized excess thrust
% State:    x = [V;gamma;psi;x;y;h;Tbar]
x0_orig = [V; gamma; psi; x; y; h; Tbar];

% data: Data structure with fields:
data = struct();
data.g = 9.81;                % Gravitational acceleration (m/s^2)
wn = 0.1;
zeta = 0.6;
data.Kh  = [2*wn*zeta, wn^2]; % altitude control gains
data.KL  = [.1, .005];        % lateral control gains
data.Ks  = [.1, .001];        % longitudinal control gains
% data.KL  = [.1, 0];        % lateral control gains
% data.Ks  = [.1, 0];        % longitudinal control gains
data.tau = 0.005;             % Engine response time (s)

% UAV Parameters
Rmin = 0.1;   % minimum turn radius (m)
hDotMax = 10; % maximum climb rate (m/s)

% Waypoints
% wpSet = targetPos;
wpSet = testWps;
```

```
% Run Simulation
[tFull, xFull, uFull, cmdFull] = UAVFlyWaypointSequence(x0_orig, wpSet, data, Rmin, hDotMax);
plot3(xFull(4,:), xFull(5,:), xFull(6,:))


function [V,F,Q] = torus(n,m,r,varargin)
  % TORUS Construct a triangle mesh of a unit torus.
  %
  % [V,F] = torus(n,m,r)
  % [V,F] = torus(n,m,r,'ParameterName',ParameterValue, ...)
  %
  % Inputs:
  %   n  number of vertices around inner ring
  %   m  number of vertices around outer ring
  %   r  radius of the inner ring
  %   Optional:
  %     'R'  followed by outer ring radius {1}
  % Outputs:
  %   V  #V by 3 list of mesh vertex positions
  %   F  #F by 3 list of triangle mesh indices
  %
  % Example:
  %   % Roughly even shaped triangles
  %   n = 40;
  %   r = 0.4;
  %   [V,F] = torus(n,round(r*n),r);
  R = 1;
  params_to_variables = containers.Map( ...
    {'R'},{'R'});
  v = 1;
  while v <= numel(varargin)
    param_name = varargin{v};
    if isKey(params_to_variables,param_name)
      assert(v+1<=numel(varargin));
      v = v+1;
      % Trick: use feval on anonymous function to use assignin to this workspace
      feval(@()assignin('caller',params_to_variables(param_name),varargin{v}));
    else
      error('Unsupported parameter: %s',varargin{v});
    end
    v=v+1;
  end

  [V,F] = create_regular_grid(n,m,true,true);

  V = V*2*pi;
  th = V(:,2);
  phi = V(:,1);
  V = [cos(phi).*(R+r*cos(th)) sin(phi).*(R+r*cos(th)) r*sin(th)];
  Q = [F(1:2:end-1,[1 2]) F(2:2:end,[2 3])];

end


function [UV,F, res, edge_norms] = ...
  create_regular_grid(xRes, yRes, xWrap, yWrap, near, far)
% Creates list of triangle vertex indices for a rectangular domain,
% optionally wrapping around in X/Y direction.
%
% Usage:
%   [UV,F,res,edge_norms] = create_regular_grid(xRes, yRes, xWrap, yWrap)
%
% Input:
%   xRes, yRes: number of points in X/Y direction
%   wrapX, wrapY: wrap around in X/Y direction
%   near, far: near and far should be fractions of one which control the
%              pinching of the domain at the center and sides
%
```

```
% Output:
%   F : mesh connectivity (triangles)
%   UV: UV coordinates in interval [0,1]x[0,1]
%   res: mesh resolution
%
% Example:
%  % Create and m by n cylinder
%  m = 10; n = 20;
%  [V,F] = create_regular_grid(m,n,1,0);
%  V = [sin(2*pi*V(:,1)) cos(2*pi*V(:,1)) (n-1)*2*pi/(m-1)*V(:,2)];
%  tsurf(F,V); axis equal;
%
%  % Quads:
%  Q = [F(1:2:end-1,[1 2]) F(2:2:end,[2 3])];
%

if (nargin<2) yRes=xRes; end
if (nargin<3) xWrap=0; end
if (nargin<4) yWrap=0; end
if (nargin<5) overlap=0; end

%res = [yRes, xRes];
res_wrap = [yRes+yWrap, xRes+xWrap];

%xSpace = linspace(0,1,xRes+xWrap); if (xWrap) xSpace = xSpace(1:end-1); end
%ySpace = linspace(0,1,yRes+yWrap); if (yWrap) ySpace = ySpace(1:end-1); end
xSpace = linspace(0,1,xRes+xWrap);
ySpace = linspace(0,1,yRes+yWrap);

[X, Y] = meshgrid(xSpace, ySpace);
UV_wrap = [X(:), Y(:)];

% Must perform pinch before edge_norms are taken
if(exist('near') & exist('far'))
  if(near>0 & far>0)
  t = ( ...
      UV_wrap(:,1).*(UV_wrap(:,1)<0.5)+ ...
      (1-UV_wrap(:,1)).*(UV_wrap(:,1)>=0.5) ...
    )/0.5;
  t = 1-sin(t*pi/2+pi/2);
  UV_wrap(:,2) = ...
    far/2 + ...
    near*(UV_wrap(:,2)-0.5).*(1-t) + ...
    far*(UV_wrap(:,2)-0.5).*t;
  else
    %error('Pinch must be between 0 and 1');
  end
end


idx_wrap = reshape(1:prod(res_wrap), res_wrap);

v1_wrap = idx_wrap(1:end-1, 1:end-1); v1_wrap=v1_wrap(:)';
v2_wrap = idx_wrap(1:end-1, 2:end  ); v2_wrap=v2_wrap(:)';
v3_wrap = idx_wrap(2:end  , 1:end-1); v3_wrap=v3_wrap(:)';
v4_wrap = idx_wrap(2:end  , 2:end  ); v4_wrap=v4_wrap(:)';

F_wrap = [v1_wrap;v2_wrap;v3_wrap; v2_wrap;v4_wrap;v3_wrap];
F_wrap = reshape(F_wrap, [3, 2*length(v1_wrap)])';

% old way
% edges = [F_wrap(:,1) F_wrap(:,2); F_wrap(:,2) F_wrap(:,3); F_wrap(:,3) F_wrap(:,1)];
% edge_norms = sqrt(sum((UV_wrap(edges(:,1),:)-UV_wrap(edges(:,2),:)).^2,2));
% edge_norms = reshape(edge_norms,size(F_wrap,1),3);

% edges numbered same as opposite vertices
edge_norms = [ ...
  sqrt(sum((UV_wrap(F_wrap(:,2),:)-UV_wrap(F_wrap(:,3),:)).^2,2)) ...
```

```
  sqrt(sum((UV_wrap(F_wrap(:,3),:)-UV_wrap(F_wrap(:,1),:)).^2,2)) ...
  sqrt(sum((UV_wrap(F_wrap(:,1),:)-UV_wrap(F_wrap(:,2),:)).^2,2)) ...
  ];

% correct indices
res = [yRes,xRes];
idx = reshape(1:prod(res),res);
if (xWrap) idx = [idx, idx(:,1)]; end
if (yWrap) idx = [idx; idx(1,:)]; end
idx_flat = idx(:);

% this might not be neccessary, could just rebuild UV like before
UV = reshape(UV_wrap,[size(idx_wrap),2]);
UV = UV(1:end-yWrap,1:end-xWrap,:);
UV = reshape(UV,xRes*yRes,2);

F = [idx_flat(F_wrap(:,1)),idx_flat(F_wrap(:,2)),idx_flat(F_wrap(:,3))];
end
```

```
end
```

Published with MATLAB® R2020a