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

```
function [tFull, xFull, uFull, cmdFull] = UAVFlyWaypointSequence(x0_orig, wpSet, data, Rmin, hDotMax)
    % Trevor Burgoyne
    % 9 Dec 2021
    %
    % Usage: Simulates a UAV flight that starts at the initial state and
    % flies to a sequence of waypoints.
    % Function Call: [tFull, xFull, uFull, cmdFull] = UAVFlyWaypointSequence(x0_orig, wpSet, p)
    %
    % INPUTS:
    %
    % x0_orig     (1,7)   x = [V;gamma;psi;x;y;h;Tbar]
    %
    %     V     true airspeed (m/s)
    %     gamma  air relative flight path angle (rad)
    %     psi   air relative flight heading angle (rad)
    %     x     East position (m)
    %     y     North position (m)
    %     h     altitude (m)
    %     Tbar  normalized excess thrust
    %
    % wpSet       (3,N)     matrix of N waypoints, in order
    % Rmin        (1,1)     minimun turn radius of UAV
    % hDotMax     (1,1)     maximum altitude rate of change
    %
    % OUTPUTS:
    % tFull       (1,M)     time vector
    % xFull       (7,M)     states across time, in form x = [V;gamma;psi;x;y;h;Tbar]
    % uFull       (3,M)     controls across time
    % cmdFull     (5,M)     commands across time

    x0 = x0_orig; % For first waypoint, we start at x0_orig
    n = size(wpSet,2); % number of waypoints
    tFull = []; xFull = []; uFull = []; cmdFull = []; % Initialize arrays to store all the flight data

    % Loop through the waypoints and navigate from point to point
    for i=1:n

        % Get current waypoint
        wp = wpSet(:,i); % column vector with [x; y; h;]

        % set the flight parameters for this segment
        p = struct();
        p.wp = wp;
        p.Rmin = Rmin;
        p.hDotMax = hDotMax;
        p.dT = .001; % sec
        p.duration = 120; % sec

        % Stopping function
        stop = @(t,x) stopSim(t,x,wp,p.duration);
        p.stopSim = stop;

        % disp(i)
        % disp('delta_h')
        % disp(wp(3)-x0(6))

        % Navigate to waypoint from current x0
        [tSeg, xSeg, uSeg, cmdSeg] = UAVFlyToWaypoint(x0, data, p);
```

```
        % Append path details to the total flight data arrays
        xFull = [xFull, xSeg];
        uFull = [uFull, uSeg];
        cmdFull = [cmdFull, cmdSeg];

        % Time vector need to be offset based on last waypoint's final timestamp
        if (i > 1)
            tSeg = tSeg + tFull(end);
        end
        tFull = [tFull, tSeg];

        % Update x0 to be the final state just calculated (last column of xSeg)
        x0 = xSeg(:, end);

    end

end
```

Published with MATLAB® R2020a

```
function [tSeg, xSeg, uSeg, cmdSeg] = UAVFlyToWaypoint(x0, data, p)
    % Final Project: Group 4
    % Trevor Burgoyne
    % 9 Dec 2021
    % UAVFlyToWaypoint
    % Usage: [tSeg, xSeg, uSeg, cmdSeg] = UAVFlyToWaypoint(x0, wp, p)
    %
    % Fly a point mass aircraft model from an initial state to a waypoint
    %
    %   State:     x = [V;gamma;psi;x;y;h;Tbar]
    %   -------------------------------------
    %     V     true airspeed
    %     gamma  air relative flight path angle
    %     psi   air relative flight heading angle
    %     x     East position
    %     y     North position
    %     h     altitude
    %     Tbar  normalized excess thrust
    %
    %   Control:   u = [Lbar;phi;Tcbar]
    %   -------------------------------
    %     Lbar    normalized excess lift
    %     phi     bank angle
    %     Tcbar   normalized excess thrust command
    %
    %   Command:   cmd = [v;psi;h;x;y]
    %   -------------------------------
    %     v       velocity command (true airspeed, m/s)
    %     psi     heading command (rad)
    %     x       eastward position (m)
    %     y       northward position (m)
    %     h       altitude command (m)
    %
    %-------------------------------------------------------------------------
    %   Form:
    %   [tSeg,xSeg,uSeg,cmdSeg] = UAVFlyToWaypoint( x0, wp, data );
    %-------------------------------------------------------------------------
    %
    %   ------
    %   Inputs
    %   ------
    %   x0      (7,1)   Initial state vector
    %   data            Feedback control parameters. Data structure with fields:
    %                       g     Gravitational acceleration
    %                       Kh    Altitude control gains
    %                       KL    Lateral control gains
    %                       Ks    Longitudinal control gains
    %   p       (.)    Flight parameters. Data structure with fields:
    %                       wp        (3,1)   Target waypoint position [x;y;h] (m)
    %                       Rmin      (1,1)   Minimum turn radius (m)
    %                       hDotMax   (1,1)   Maximum climb rate (abs val) (m/s)
    %                       dT        (1,1)   Time step (s)
    %                       duration  (1,1)   Max simulation duration (s)
    %                       stopSim = @(t,x)  Anonymous function. Sim terminates
    %                                         when this evaluates to true.
    %
    %   -------
    %   Outputs
    %   -------
```

```
    %   tSeg    (1,N)     Time vector for this segment. Equivalent to "t" input.
    %   xSeg    (7,N)     State vector across time for this segment.
    %   uSeg    (3,N)     Control vector across time for this segment.
    %   cmdSeg  (3,N)     Commands (v,h,psi) across time for this segment.
    %
    %------------------------------------------------------------------------



    % Call UAV Steering using the waypoint to generate steering functions
    % [tSeg, vCmdFun, hCmdFun, psiCmdFun] = UAVSteering(x0, wp, p);
    % [cmd, cmdDot] = UAVGuidance( t, x, p );

    % Define time interval (TODO: Decide on limits)
    tSeg = 0:p.dT:p.duration; % sec

    % Call UAVSim on the steering functions to simulate the UAV
    [tSeg, xSeg, uSeg, cmdSeg] = UAVSim(tSeg, x0, data, p);

end
```

Published with MATLAB® R2020a

Contents


```
function UAVFlyThrough( time, states, fig )
```

```
% Animated fly-through of the obstacle course along the UAV trajectory.
%
% Call PlotUAVObstacleCourse or ScoreUAVObstacleCourse first to generate
% the plot and get the figure handle.
%
% USAGE:
%   UAVFlyThrough( time, states, fig );
%
% INPUTS:
%   time    (1,N)     Time vector
%   states  (7,N)     State history over time. [v;gamma;psi;h;x;y;Tbar]
%   fig     (1,1)     Figure handle to the figure showing the UAV obstacle
%                     course.
% OUTPUTS:
%   None
%
```

Input Checking

```
if nargin<3
  fig = gcf;
end
figure(fig);
```

Run Function

```
xd = states(4,:);
yd = states(5,:);
zd = states(6,:);

camproj perspective
camva(25)

hlight = camlight('headlight');

fprintf(1,'Press a key to begin the flythrough...\n');
pause()

nn = 50;
i=1;
g=plot3(xd(i:i+nn),yd(i:i+nn),zd(i:i+nn),'y','linewidth',3);
for i=1:length(xd)-nn
   g.XData = xd(i:i+nn);
   g.YData = yd(i:i+nn);
   g.ZData = zd(i:i+nn);
   campos([xd(i),yd(i),zd(i)])
   camtarget([xd(i+nn/5),yd(i+nn/5),zd(i+nn/5)])
```

```
    camlight(hlight,'headlight')
    drawnow

end
```

Published with MATLAB® R2020a

```
function [tSeg,xSeg,uSeg,cmdSeg] = UAVSim( t, x0, data, p )
% Usage: Simulate a point mass aircraft model from an initial state w/ steering.
%   State:     x = [V;gama;psi;x;y;h;Tbar]
%   -------------------------------------
%     V       true airspeed
%     gamma   air relative flight path angle
%     psi     air relative flight heading angle
%     x       East position
%     y       North position
%     h       altitude
%     Tbar    normalized excess thrust
%
%   Control:   u = [Lbar;phi;Tcbar]
%   -------------------------------
%     Lbar    normalized excess lift
%     phi     bank angle
%     Tcbar   normalized excess thrust command
%
%   Command:   cmd = [v;psi;h;x;y]
%   ------------------------------
%     v       velocity command (true airspeed, m/s)
%     psi     heading command (rad)
%     h       altitude command (m)
%     x       eastward position (m)
%     y       northward position (m)
%
%-------------------------------------------------------------------------
%   Form:
%   [tSeg,xSeg,uSeg,cmdSeg] = UAVSim( time, x0, data, p );
%-------------------------------------------------------------------------
%
%   ------
%   INPUTS
%   ------
%   t       (1,N)     Time vector
%   x0      (7,1)     Initial state vector
%   data              Data structure with fields:
%                       g     Gravitational acceleration
%                       Kh    Altitude control gains
%                       KL    Lateral control gains
%                       Ks    Longitudinal control gains
%   p       (.)       Flight parameters. This function uses the following
%                     fields:
%                       wp        (3,1)   Target waypoint position (x,y,h)
%                       Rmin      (1,1)   Minimum turn radius (m)
%                       hDotMax   (1,1)   Maximum climb rate (abs val)
%                       dT        (1,1)   Time step
%                       stopSim = @(t,x)  Anonymous function. Sim terminates
%                                         when this evaluates to true.
%
%   -------
%   OUTPUTS
%   -------
%   tSeg    (1,M)     Time vector for this segment. Equivalent to "t" input.
%   xSeg    (7,M)     State vector across time for this segment.
%   uSeg    (3,M)     Control vector across time for this segment.
%   cmdSeg  (3,M)     Commands (v,h,psi) across time for this segment.
%
% ** Note: M>=N. The simulation may terminate before it reaches the end of
```

```
% the original input time vector.
%
%-------------------------------------------------------------------------
  nt = length(t);
  ns = size(x0,1);

  if nargin < 4
    p = [];
  end

  if ~isfield(p,'stopSim')
    p.stopSim = @(t,x) 0;
  end

  if( nargin < 3 )
    data.a   = zeros(3,1);
    data.W   = data.a;
    data.g   = 9.81;
    data.tau = 5;
    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
  end

  tSeg = t;
  xSeg = zeros(ns,nt);
  uSeg = zeros(3,nt);
  cmdSeg = zeros(5,nt);

  dT = diff(t);

  % initial state
  xSeg(:,1) = x0;

  xtmp = x0;
  cmdDot = zeros(5,1);
  for k=1:nt-1

    % compute commands
    %=================
    % fprintf('iter %d\n',n)
%     disp(k)
%     disp('state vec')
%     disp(xtmp)
    [cmd,cmdDot] = UAVGuidance(t(k), xtmp, p );

    cmdSeg(:,k) = cmd;
%     disp('post guidance')
%
%     disp(xSeg(:,k))
%     disp(cmdSeg(:,k))
%     disp(cmdDot)

    % compute controls
    %=================
    u = UAVControl( xSeg(:,k), cmdSeg(:,k), cmdDot, data );
%     disp('post control')
%     disp(u)
%     uSeg(:,k) = u;
```

```
    % integrate state with controls
    %==============================
    rhs = @(t,x) UAVRHS(x,u,data.g,data.tau);
    xx = ODENumIntRK4(rhs,[0 dT(k)],xtmp')';
    xtmp = xx(2,:); % Get only the new part of xtmp

    % if (mod(k, 1000) == 0) % Print every 1000 iterations
    %   fprintf('iter %d\n',k)
    %   disp('state')
    %   disp(xtmp)
    %   disp('cmd')
    %   disp(cmd)
    %   disp('cmdDot')
    %   disp(cmdDot)
    %   disp('u')
    %   disp(u)
    % end

    % store state data
    %=================
    xSeg(:,k+1) = xtmp;

    % terminate if we are close enough to the target waypoint
    %========================================================
    if p.stopSim(t(k),xtmp)
      % disp('stop activated at:')
      % disp(t(k))
      break
    end

  end

  k = k+1;
  uSeg(:,k) = UAVControl( xSeg(:,k), cmdSeg(:,k), cmdDot, data );

  % Trim any excess columns
  tSeg = tSeg(1:k);
  xSeg = xSeg(:,1:k);
  uSeg = uSeg(:,1:k);
  cmdSeg = cmdSeg(:,1:k);

end
```

Published with MATLAB® R2020a

Contents


```
function u = UAVControl(x0,stateCmd,stateCmdDot, data)
```

```
% Compute Lift (Lbar), bank angle (phi) and Thrust (Tbar) required for
% a commanded state.
% INPUTS:
% x0            (6,1)           %
% State:     x = [V;gamma;psi;x;y;h;Tbar]
%   -------------------------------------
%     V     true airspeed (m/s)
%     gamma  air relative flight path angle (rad)
%     psi   air relative flight heading angle (rad)
%     x     East position (m)
%     y     North position (m)
%     h     altitude (m)
%     Tbar  normalized excess thrust
% stateCmd      (5,1)   Commanded velocity, heading, altitude, and horizontal
%                       position.  [v;psi;h;x;y]
%
% stateCmdDot   (3,1)   Commanded rate of change of velocity, heading, altitude.
%                       [vDot;psiDot;hDot]
%
%
% data              Data structure with fields:
%                       g     Gravitational acceleration (1,1)
%                       Kh    Altitude control gains     (1,2)
%                       KL    Lateral control gains      (1,2)
%                       Ks    Longitudinal control gains (1,2)
% OUTPUTS:
% u             (1,3)   Command vector [Lbar, phi, Tbar]
%                       Lbar     Normalized Lift required, (1,1)
%                       phi      Bank angle (x = East, y = North), (rad), (1,1)
%                       Tbar     Normalized Thrust required, (1,1)
```

Demo

```
if nargin == 0
    disp('Demo Mode')
    x0 = [ 1;  1; 0; 0; 0; 0;1];
    stateCmd = [ 2;  2; 4; 3; 1];
    stateCmdDot = [ 1;  1; pi/4];
    K = [ 1; 1; 1; 1; 1; 1];
    data.g = 9.81; % (m/s)
    data.Kh = [1,1];
    data.KL = [1,1];
    data.Ks = [1,1];
end
```

Input Checking

```
% check state vector is complete
if length(x0) ~= 7 || ~isreal(x0)
    error('Error inital state not complete or is not real')
end
% check stateCmd vector is complete and real
if length(stateCmd) < 5 || ~isreal(stateCmd)
    error('stateCmd must have 5 real elements')
end
% check stateCmdDot is complete and real
if length(stateCmdDot) < 3 || ~isreal(stateCmdDot)
    length(stateCmdDot)
    ~isreal(stateCmdDot)
    error('stateCmdDot must have 3 real elements')
end
% check data is a structure
if ~isstruct(data)
    error('data must be a struct')
end
```

Constants

```
% gravitational acceleration on Earth
g = data.g; % (m/s/s)

% uncertainites
nH = 0; % altitude uncertainty, (m)
nHDot = 0; % altitude derivative uncertainty, (m/s)
nV = 0; % speed uncertainty, (m/s)
nPsi = 0; % air-relative heading uncertainty, (rad)
nZeta = 0; % along-track position uncertainty (m)
nEta = 0; % cross track position uncertainty (m)

% define gains
Kh1 = data.Kh(1);
Kh2 = data.Kh(2);
KL1 = data.KL(1);
KL2 = data.KL(2);
KN1 = data.Ks(1);
KN2 = data.Ks(2);
```

Compute function

```
% pull apart x0 vector
v = x0(1,1);
gamma = x0(2,1);
psi = x0(3,1);
xe = x0(4,1);
yn = x0(5,1);
h = x0(6,1);

% define derivatives necessary for computing Lbar, phi, Tcbar
hDot = v*sin(gamma);
xeDot = v*cos(gamma)*sin(psi);
ynDot = v*cos(gamma)*cos(psi);
```

```
% pull apart stateCmd
vCmd = stateCmd(1,1);
psiCmd = stateCmd(2,1);
hCmd = stateCmd(3,1);
xeCmd = stateCmd(4,1);
ynCmd = stateCmd(5,1);

% pull apart stateCmdDot
vCmdDot = stateCmdDot(1,1);
psiCmdDot = stateCmdDot(2,1);
hCmdDot = stateCmdDot(3,1);

% compute ground speed
vGround = sqrt(xeDot^2 + ynDot^2);

% compute zeta and eta
values = [sin(psiCmd) cos(psiCmd); cos(psiCmd) -1*sin(psiCmd)] * [ xe - xeCmd; yn - ynCmd ];
zeta = values(1); eta = values(2);
```

Compute phi, Lbar and Tbar

```
% phi = bank angle, +- pi/2
% Lbar = normalized excess thrust
% Tbar = normalized excess thrust

% make sure phi is real
X = (vCmd/g)*psiCmdDot - ((KL1*vCmd)/g)*(psi - psiCmd + nPsi) - (KL2/g)*(eta + nEta);

if X > sin(pi/2)
    X = sin(pi/2);
end
if X < -sin(pi/2)
    X = -sin(pi/2);
end

phi = asin(X);
Lbar = 1/cos(phi)*(1-Kh1/g*(hDot - hCmdDot + nHDot) - Kh2/g*(h - hCmd + nH));
Tbar = sin(gamma) + vCmdDot/g - KN1/g*(vGround - vCmd + nV) - KN2/g*(zeta + nZeta);

phiMax = pi/2; LbarMax = 10; TbarMax = 1;

if phi > phiMax
    phi = phiMax;
elseif phi < -phiMax
    phi = -phiMax;
end

if Lbar > LbarMax
    Lbar = LbarMax;
elseif Lbar < -LbarMax
    Lbar = -LbarMax;
end

if Tbar > TbarMax
    Tbar = TbarMax;
elseif Tbar < -TbarMax
    Tbar = -TbarMax;
end
```

```
u = [Lbar, phi, Tbar];

if ~isreal(u)
    u
    error('Imaginary control vector')
end
```

Published with MATLAB® R2020a

Contents


```
function bool = stopSim(t,x,wp,tStop)
```

```
% Trevor Burgoyne
% 15 Dec 2021
%
% Usage: bool = stopSim(t,x,wp)
% Return 'true' if simulation has reached the waypoint
% or has timed out (30 sec).
%
% Inputs:
% t = current timestamp
% x = state vector
% wp = current waypoint
%
% Outputs:
% bool = returns 1 if simulation should stop; else 0
```

Demo

```
    if nargin <= 0
        x = [0; 0; 0; 2; 3; 4; 0];
        t = 10;
        wp = [2.5; 3; 4];
        tStop = 30;
    end
```

Stop function

Current position

```
    xe = x(4);
    yn = x(5);
    h = x(6);

    % Define waypoint position [x,y,h]
    xe_wp = wp(1);
    yn_wp = wp(2);
    h_wp = wp(3);

    % Tolerance
    tol = 3; % m

    dist = sqrt((xe-xe_wp)^2 + (yn-yn_wp)^2 + (h-h_wp)^2);

    if (dist <= tol)
        disp('hit waypoint!')
        bool = 1;
    else if (t > tStop)
        disp('took too long, timed out!')
```

```
        bool = 1;
    else
        bool = 0;
    end

end
```

Published with MATLAB® R2020a

Contents


```
function [rhs] = UAVRHS(x,u,g,tau)
```

```
% Given state vector "x", command vector "u", and constants "g" and "tau",
% compute the Right Hand Side state derivative
%
% Usage:
% UAV_RHS(x, u, g, tau)
%
% INPUTS:
% x - (7,1) element state vector
%     V       true airspeed (m/s)
%     gamma   air relative flight path angle (radians)
%     psi     air relative flight heading angle (radians)
%     x       East position (m)
%     y       North position (m)
%     h       altitude (m)
%     Tbar    normalized excess thrust (N)
%
% u - (3,1) element command vector
%     v       velocity command (true airspeed, m/s)
%     psi     heading command (rad)
%     h       altitude command (m)
%
% g - gravity (m/s^2)
% tau - engine thrust response time (s)
%
% OUTPUTS:
% rhs - (7,1) right hand side output of the changes in the state vector

% initalize rhs
rhs = [0;0;0;0;0;0;0];
```

Input checking

```
if g <= 0
    error("G must not be negative")
end
if tau < 0
    error("Engine thrust response time must be greater than zero")
end
if length(x) ~= 7
    error("State vector must contain 7 elements")
end
if length(u) ~= 3
    error("Control vector must contain 3 elements")
end
```

Demo

```
if nargin == 0
    x = [5;0;0;6;10;15;2];
    u = [2;5;3];
    g = 9.81;
    tau = 2;
end
```

Dynamic Model

```
% current state
V = x(1);                   % true airspeed
gamma = x(2);               % air-relative flight path angle
psi = x(3);                 % air-relative heading
x_e = x(4);                 % East position
y_n = x(5);                 % North position
h = x(6);                   % altitude
T = x(7);                   % normalized excess thrust

% current controls
L = u(1);                   % velocity command
phi = u(2);                 % heading command
T_c = u(3);                 % altitude command

% State limits
vMax = 30; % max speed (m/s)
hMax = 39; % near top of course (m)
hMin = 1;  % near bottom of course of course (m)

% rhs calculations

vDot = T*g - g*sin(gamma);                    % vDot
if (abs(V) < vMax)
    rhs(1) = vDot;
elseif (sign(V) == sign(vDot))
    % Case when V is beyond maximum: only allow vDot that reduces abs(V)
    % When V and vDot have same sign, this will INCREASE abs(V)

    % Prevent accelerating beyond vMax
    rhs(1) = 0;
else
    rhs(1) = vDot; % vDot is decreasing abs(V)
end

rhs(2) = 1/V * (g*L*cos(phi) - g *cos(gamma)); % gammaDot
rhs(3) = 1/(V*cos(gamma))*(g*L*sin(phi));      % psiDot
rhs(4) = V*cos(gamma)*sin(psi);                % xDot
rhs(5) = V*cos(gamma)*cos(psi);                % yDot

hDot = V*sin(gamma);                           % hDot
if (h > hMin && h < hMax)
    rhs(6) = hDot;
elseif (h < hMin && hDot > 0)
    % Allow hDot only if it INCREASES h
    rhs(6) = hDot;
elseif (h > hMax && hDot < 0)
    % Allow hDot only if it DECREASES h
    rhs(6) = hDot;
else
    % Prevent leaving bounds of course
```

```
    rhs(6) = 0;
end
rhs(7) = 1/tau*(T_c-T);                         % TbarDot
```

```
end
```

Published with MATLAB® R2020a

Contents


```
function [cmd,cmdDot] = UAVGuidance( t, x, p )
```

Generate velocity, heading and altitude commands for a UAV

```
State:     x = [V;gama;psi;x;y;h;Tbar]
-------------------------------------
  V     true airspeed
  gama  air relative flight path angle
  psi   air relative flight heading angle
  x     East position
  y     North position
  h     altitude
  Tbar  normalized excess thrust

Command:   cmd = [v;psi;h;x;y]
------------------------------
  v       velocity command (true airspeed, m/s)
  psi     heading command (rad)
  h       altitude command (m)
  x       eastward position (m)
  y       northward position (m)
```

```
%------------------------------------------------------------------------
%   Form:
%   [cmd,cmdDot,p] = UAVGuidance( t, x, p );
%------------------------------------------------------------------------
%
%   ------
%   Inputs
%   ------
%   t   (1,1)     Current time
%   x   (7,1)     Current state vector
%   p   (.)       Flight parameters. This function uses the following
%                 fields:
%                   wp        (3,1)   Target waypoint position (x,y,h)
%                   Rmin      (1,1)   Minimum turn radius (m)
%                   hDotMax   (1,1)   Maximum climb rate (abs val)
%                   duration  (1,1)   Max duration to simulate (sec)
%                   dT        (1,1)   Time step
%                   stopSim = @(t,x)  Anonymous function. Sim terminates
%                                         when this evaluates to true.
%
%   -------
%   Outputs
%   -------
%   cmd     (5,1)     Commanded velocity, heading, altitude, and horizontal
%                       position.  [v;psi;h;x;y]
%   cmdDot  (3,1)     Commanded rate of change of velocity, heading, altitude.
%                       [vDot;psiDot;hDot]
%
%------------------------------------------------------------------------

% If no guidance parameters are provided, just keep flying along current
```

```
% trajectory
if isempty(p)
  cmd = x([1 3 6 4 5]); % current state values for: [v, psi, h, x, y]
  cmdDot = zeros(3,1);
  return
end

% Turning -- velocity and heading command
[vDotCmd,psiDotCmd] = UAVAutoTurn( x, p.wp, p.Rmin, p.dT );
vCmd = x(1)+vDotCmd*p.dT;
psiCmd = x(3)+psiDotCmd*p.dT;

% Follow the turn -- lateral position commands
xCmd = x(4) + vCmd*sin(psiCmd);
yCmd = x(5) + vCmd*cos(psiCmd);

% Climbing -- altitude commands
dh = p.wp(3) - x(6);
if abs(dh/p.dT)>p.hDotMax
  hDotCmd = p.hDotMax*sign(dh);
else
  hDotCmd = dh/p.dT;
end
hCmd = p.wp(3);


% Stack the commands into vectors
cmd = [vCmd;psiCmd;hCmd;xCmd;yCmd];
cmdDot = [vDotCmd;psiDotCmd;hDotCmd];
```

```
function [vDot,psiDot] = UAVAutoTurn( state, wp, Rmin, dT )
% getting varibales from input
x0 = state(4);
y0 = state(5);

xT = wp(1);
yT = wp(2);

v0 = state(1);
gamma0 = state(2);
psi0 = state(3);

% center of turning circle: potentially two sides
xC1 = x0+Rmin*cos(psi0);
yC1 = y0-Rmin*sin(psi0);

xC2 = x0+Rmin*cos(psi0+pi);
yC2 = y0-Rmin*sin(psi0+pi);

% choose the side closer to the target as long as it is > R
d1 = norm([xT-xC1;yT-yC1]);
d2 = norm([xT-xC2;yT-yC2]);

if d1<Rmin || d2<Rmin
  % fly straight for now
  psiDot = 0;
else

  % desired heading
```

```
  psiT = atan2(xT-x0,yT-y0);

  % est time to destination (assuming correct heading)
  xDot = v0*cos(gamma0)*sin(psi0);
  yDot = v0*cos(gamma0)*cos(psi0);

  tDest = sqrt((xT-x0)^2 + (yT-y0)^2) / sqrt(xDot^2 + yDot^2);

  if abs(psiT-psi0)>pi
    1;
  end

  while ( (psiT-psi0) < -pi )
    psiT = psiT+2*pi;
  end
  while ( (psiT-psi0) > pi )
    psiT = psiT-2*pi;
  end

  % psiDot = (psiT-psi0)/dT;
  % Turn slightly slower than literally dT
  psiDot = (psiT-psi0)/(.1*tDest);

  if abs(psiDot)>v0/Rmin
    psiDot = sign(psiDot)*v0/Rmin;
  end

end

vDot = 0;
```

Published with MATLAB® R2020a