Problem 1

The code for my MATLAB program is attached. Below are plots of the unsafe region for $T=-5$, at the various different relative headings.

![Plot 1: $T=-5, \psi_1 = 1.571$](image1)

![Plot 2: $T=-5, \psi_1 = 0.000$](image2)
As $T \to -\infty$, all of the reachable (unsafe) sets continue to grow. This is because regardless of the relative heading, no matter what the evader (controller) attempts to do (speed up or slow down) the pursuer (disturbance) can always instantly match her speed. Thus, given an initial state at $T$, all states that are propagated backward (at the same speeds) must also be unsafe; thus the reachable set should not shrink as $T \to -\infty$. Additionally, since the pursuer can always travel faster (or slower) than the evader, there will always be new unsafe states as $T$ decreases, and so in fact the unsafe sets will grow as $T \to -\infty$. 
```matlab
psi_r_array = [pi/2 0 -pi/4 -pi/2];
for i=1:4
    air3D('medium', 5, i);
    xlabel('x_r');
    ylabel('y_r');
    title(['T = -5, \{it\}psi_r = ' num2str(psi_r_array(i), '%1.3f')]);
    saves(gcf, ['HW3_Probl_Fig' num2str(i, '%1d') '.png']);
end

function [ data, g, data0 ] = air3D(accuracy, tMax, psi_index)
% Adapted heavily (or modified slightly, depending on how you look at it)
% from Ian Mitchell's example in the level set toolbox.
% Original example Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).

%-------------------------------------------------------------------------------
% Make sure we can see the kernel m-files.
addpath(genpath('c:/Program Files/MATLAB/R2008b/ToolboxLS-1.1/Kernel'));

%-------------------------------------------------------------------------------
% Integration parameters.
plotSteps = 9; % How many intermediate plots to produce?
t0 = 0; % Start time.
singleStep = 0; % Plot at each timestep (overrides tPlot).

% Period at which intermediate plots should be produced.
tPlot = (tMax - t0) / (plotSteps - 1);

% How close (relative) do we need to get to tMax to be considered finished?
small = 100 * eps;

% What kind of dissipation?
dissType = 'global';

%-------------------------------------------------------------------------------
% What level set should we view?
level = 0;

% visualize the 3D reachable set.
displayType = 'contour';

% Pause after each plot?
pauseAfterPlot = 0;

% Delete previous plot before showing next?
deleteLastPlot = 1;

% Visualize the angular dimension a little bigger.
aspectRatio = [ 1 1 0.4 ];

% Plot in separate subplots (set deleteLastPlot = 0 in this case)?
useSubplots = 0;
```

% Approximately how many grid cells?
% (Slightly different grid cell counts will be chosen for each dimension.)
Nx = 51;

% Create the grid.
g.dim = 2;
mins = [ -6 -26; -11 -6; -6 -6; -6 -6];
maxs = [+16 +16; +11 +6; +13 +20; +16 +20];
g.min = mins.psi_index,);
g.max = maxs.psi_index,);
g.bdry = [ @addGhostExtrapolate; @addGhostExtrapolate ];
% Roughly equal dx in x and y (so different N).
g.N = [ Nx; ceil(Nx * (g.max(2) - g.min(2)) / (g.max(1) - g.min(1))) ];
% Need to trim max bound in \psi (since the BC are periodic in this
% dimension).
g = processGrid(g);

% Create constants
v1_min = 2;
v1_max = 4;
v2_min = 1;
v2_max = 5;
targetRadius = 5;
psi_r_array = [pi/2 0 -pi/4 -pi/2];
psi_r = psi_r_array(psi_index);

% Create initial conditions (cylinder centered on origin).
data = shapeSphere(g, [ 0; 0], targetRadius);
data0 = data;

% Set up spatial approximation scheme.
schemeFunc = @termLaxFriedrichs;
schemeData.hamFunc = @air3DHamFunc;
schemeData.partialFunc = @air3DPartialFunc;
schemeData.grid = g;

% The Hamiltonian and partial functions need problem parameters.
schemeData.psi_r = psi_r;
schemeData.v1_min = v1_min;
schemeData.v2_min = v2_min;
schemeData.v1_max = v1_max;
schemeData.v2_max = v2_max;

% Choose degree of dissipation.
switch(dissType)
case 'global'
schemeData.dissFunc = @artificialDissipationGLF;
case 'local'
schemeData.dissFunc = @artificialDissipationLLF;
case 'local'
    schemeData.dissFunc = @artificialDissipationLLL;
    otherwise
    error('Unknown dissipation function \%s', dissFunc);
end

if(nargin < 1)
    accuracy = 'medium';
end

% Set up time approximation scheme.
integratorOptions = odeCFLset('factorCFL', 0.75, 'stats', 'on');

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
    case 'low'
        schemeData.derivFunc = @upwindFirstFirst;
        integratorFunc = @odeCFL1;
    case 'medium'
        schemeData.derivFunc = @upwindFirstENO2;
        integratorFunc = @odeCFL2;
    case 'high'
        schemeData.derivFunc = @upwindFirstENO3;
        integratorFunc = @odeCFL3;
    case 'veryHigh'
        schemeData.derivFunc = @upwindFirstWENO5;
        integratorFunc = @odeCFL3;
    otherwise
        error('Unknown accuracy level \%s', accuracy);
end

if(singleStep)
    integratorOptions = odeCFLset(integratorOptions, 'singlestep', 'on');
end

% Restrict the Hamiltonian so that reachable set only grows.
% The Lax-Friedrichs approximation scheme MUST already be completely set up.
innerFunc = schemeFunc;
innerData = schemeData;
clear schemeFunc schemeData;

% Wrap the true Hamiltonian inside the term approximation restriction routine.
schemeFunc = @termRestrictUpdate;
schemeData.innerFunc = innerFunc;
schemeData.innerData = innerData;
schemeData.positive = 0;

% Initialize Display
f = figure;

% Set up subplot parameters if necessary.
if(useSubplots)
rows = ceil(sqrt(plotSteps));
cols = ceil(plotSteps / rows);
plotNum = 1;
subplot(rows, cols, plotNum);
end

h = visualizeLevelSet(g, data, displayType, level, [ 't = ' num2str(t0) ]);%
camlight right; camlight left;
hold on;
axis(g.axis);
daspect(aspectRatio);
drawnow;

%-------------------------------------------------------------------------------
% Loop until tMax (subject to a little roundoff).
tNow = t0;
startTime = cputime;
while(tMax - tNow > small * tMax)

    % Reshape data array into column vector for ode solver call.
y0 = data(:,);

    % How far to step?
tSpan = [ tNow, min(tMax, tNow + tPlot) ];

    % Take a timestep.
    [ t y ] = feval(integratorFunc, schemeFunc, tSpan, y0,...
                  integratorOptions, schemeData);
    tNow = t(end);

    % Get back the correctly shaped data array
    data = reshape(y, g.shape);

    if(pauseAfterPlot)
        % Wait for last plot to be digested.
        pause;
    end

    % Get correct figure, and remember its current view.
    figure(f);
    figureView = view;

    % Delete last visualization if necessary.
    if(deleteLastPlot)
        delete(h);
    end

    % Move to next subplot if necessary.
    if(useSubplots)
        plotNum = plotNum + 1;
        subplot(rows, cols, plotNum);
    end

end
Problem 2

The code for my MATLAB program is attached. Below are plots of the safe and unsafe regions for various different turning radii. (The unsafe set is shaded. For my program, I increased the turning radius by increasing the linear velocity.) As we can see, the size of the unsafe set decreases as the turning radius increases.
airspeeds = [2 4 8 16];
for i=1:length(airspeeds)
    airMode('medium', airspeeds(i));
    xlabel('x_r');
    ylabel('y_r');
    title(['Safe and Unsafe Sets for Turning Radius = ', num2str(airspeeds(i), '%.1f')]);
    saveas(gcf, ['HW3_Prob2_Fig' num2str(i, '%d') '.png']);
end

function [ reach, g, avoid, data0 ] = airMode(accuracy, airspeed)
% Adapted heavily (or modified slightly, depending on how you look at it)
% from Ian Mitchell's example in the level set toolbox.
% Original example Copyright 2004 Ian M. Mitchell (mitchell@cs.ubc.ca).
%--------------------------------------------------------------------------
addpath(genpath('c:/Program Files/MATLAB/R2008b/ToolboxLS-1.1/Kernel'));
%--------------------------------------------------------------------------
% Problem Parameters.
targetRadius = 5;
velocityA = airspeed;
velocityB = airspeed;
psi = 2 * pi / 3;
omega = 1;
%--------------------------------------------------------------------------
% Integration parameters.
tMaxStraight = 20; % End time for straight segments.
tMaxCurved = pi / omega; % End time for curved segments.
% Period at which intermediate plots should be produced.
tPlot = 10;
fig = figure;
%--------------------------------------------------------------------------
% Default accuracy.
if(margin < 1)
    accuracy = 'medium';
end
%--------------------------------------------------------------------------
% Create the grid.
g.dim = 2;
g.min = -25;
g.max = +25;
g.bdry = @addGhostExtrapolate;
g.N = 101;
g = processGrid(g);
%--------------------------------------------------------------------------
% Create initial conditions (circle centered on origin).
data0 = shapeSphere(g, [ 0; 0 ], targetRadius);
%--------------------------------------------------------------------------
% Create the flow fields for the two types of motion.
\% Multiply by -1 to get forward PDE.
straight = [ \{-\text{velocityA} + \text{velocityB} \times \cos(\psi)\}; ... \\
\{-\text{velocityB} \times \sin(\psi)\} ];
curved = [ \{-\text{velocityA} + \text{velocityB} \times \cos(\psi) + \omega \times g.xs(2)\}; ... \\
\{-\text{velocityB} \times \sin(\psi) - \omega \times g.xs(1)\} ];

\% Mode 3 reachable set is a simple reach computation.
\% Anywhere in this set in mode 3 is unsafe.
[ mode3, stepTime ] = findReachSet(g, data0, straight, accuracy, ... 
\quad tMaxStraight, tPlot, fig, 1, []);

\% Next is Mode 2 reachable set without relation to mode 3.
\% Anywhere in this set in mode 2 is unsafe.
[ mode2, stepTime ] = findReachSet(g, data0, curved, accuracy, ... 
\quad tMaxCurved, tPlot, fig, 1, []);

\% Now we need the set of states that is taken by mode 2 into
\% an unsafe state in mode 3.
\% Initial condition for this set are taken from mode 3
\% (taking rotational reset map between modes 2 and 3 into account).
initial = rot90(mode3, \{-1\});

[ switch2, stepTime ] = findReachSet(g, initial, curved, accuracy, ... 
\quad tMaxCurved, tPlot, fig, 0, []);

\% Finally, we get back to mode 1, which has a controlled switch and hence
\% an escape set.
\% Escape set is complement of
\% everything unsafe in mode 2 union everything mode 2 will take to unsafe in
\% mode 3. (don't forget the reset map between modes 1 and 2.)
avoid = min(rot90(mode2, \{-1\}), rot90(switch2, \{-1\}));

[ reach, stepTime ] = findReachSet(g, data0, straight, accuracy, ... 
\quad tMaxStraight, tPlot, fig, 1, avoid);
close;

\% Now we can summarize the results in mode 1.
figure;
level = [ \{0 0\};
contourf(g.xs(1), g.xs(2), \{-reach, level\});
hold on;
contour(g.xs(1), g.xs(2), avoid, level, 'r-');
contour(g.xs(1), g.xs(2), mode3, level, 'b--');
axis equal;
axis(g.axis);
fprintf('Total execution time %5g seconds', executionTime);

\% Clip back the axis bounds slightly so that the distortion caused
\% by the artificial boundary doesn't show.
clip = [ +5, -5, +5, -5 ];
axis(g.axis + clip);

function [ final, executionTime ] = findReachSet(g, initial, velocity, ...
   accuracy, tMax, tPlot, fig, growOnly, avoid)

% What level set should we view?
level = 0;
% Visualize the 2D reachable set.
displayType = 'contour';
% Pause after each plot?
pauseAfterPlot = 0;
% Delete previous plot before showing next?
deleteLastPlot = 1;
% How close (relative) do we need to get to tMax to be considered finished?
small = 100 * eps;
t0 = 0; % Start time.

% Set up spatial approximation scheme.
schemeFunc = @termConvection;
schemeData.grid = g;
schemeData.velocity = velocity;

% Set up time approximation scheme.
integratorOptions = odeCFLset('factorCFL', 0.75, 'stats', 'on');

% Choose approximations at appropriate level of accuracy.
switch(accuracy)
   case 'low'
      schemeData.derivFunc = @upwindFirstFirst;
      integratorFunc = @odeCFL1;
   case 'medium'
      schemeData.derivFunc = @upwindFirstENO2;
      integratorFunc = @odeCFL2;
   case 'high'
      schemeData.derivFunc = @upwindFirstENO3;
      integratorFunc = @odeCFL3;
   case 'veryHigh'
      schemeData.derivFunc = @upwindFirstWENO5;
      integratorFunc = @odeCFL3;
   otherwise
      error('Unknown accuracy level %s', accuracy);
end

% Restrict the Hamiltonian so that reachable set only grows.
if(growOnly)
   innerFunc = schemeFunc;
   innerData = schemeData;
   clear schemeFunc schemeData;
% Wrap the true Hamiltonian inside the term approximation restriction.
schemeFunc = @termRestrictUpdate;
schemeData.innerFunc = innerFunc;
schemeData.innerData = innerData;
schemeData.positive = 0;
end

%----------------------------------------------------------
if(isempty(avoid))
  data = initial;
else
  % Ensure that the initial data satisfies the avoid set.
  data = max(initial, avoid);
  % Set up data required for masking by the avoid set.
  % Mask will be compared to vector form of data array used by integrator.
  schemeData.maskData = avoid();
  schemeData.maskFunc = @max;
  % Let the integrator know what function to call.
  integratorOptions = odeCFLset(integratorOptions, ...
    'postTimestep', @postTimestepMask);
end

%----------------------------------------------------------
% Initialize Display
figure(fig);

h = visualizeLevelSet(g, data, displayType, level, [ 't = ' num2str(t0) ]); 

%----------------------------------------------------------
% Loop until tMax (subject to a little roundoff).
tNow = t0;
startTime = cputime;
while(tMax - tNow > small * tMax)
  % Reshape data array into column vector for ode solver call.
  y0 = data(:);
  % How far to step?
  tSpan = [ tNow, min(tMax, tNow + tPlot) ];
  % Take a timestep.
  [ t y ] = feval(integratorFunc, schemeFunc, tSpan, y0, ...
    integratorOptions, schemeData);
  tNow = t(end);
  % Get back the correctly shaped data array
  data = reshape(y, g.shape);
  if(pauseAfterPlot)
    % Wait for last plot to be digested.
    pause;
  end
  % Get correct figure, and remember its current view.
  figure(fig);
  % Delete last visualization if necessary.
  if(deleteLastPlot)
    delete(h);
  end
  % Create new visualization.
  h = visualizeLevelSet(g, data, displayType, level, [ 't = ' num2str(tNow) ]); 
end
final = data;