Table of Contents

Clean

```
close all; clear; clc;
```

Get Trim and Initial Conditions

```
% Define parameters for specifying control law
SLC = 2;
FEED = 1;

% Set flags
CONTROL_FLAG = SLC; % <========= Set to control law to use (SLC or FEED)

% Aircraft parameter file
ttwistor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine trim state and control inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V_trim = 18;
h_trim = 1805;
gamma_trim = 0;
trim_definition = [V_trim; gamma_trim; h_trim];

% STUDENTS REPLACE THESE TWO FUNCTIONS WITH YOUR VERSIONS FROM HW3/4
% [trim_variables, fval] = CalculateTrimVariables(trim_definition,
aircraft_parameters);
% [aircraft_state_trim, control_input_trim] =
TrimConditionFromDefinitionAndVariables(trim_variables, trim_definition);
```

```
[aircraft_state_trim, control_input_trim, trim_variables] =
TrimCalculator(trim_definition, aircraft_parameters);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine control gains
% See 'RunGVF.m for how gain files were created
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (CONTROL_FLAG==FEED)
    gains_file = 'ttwistor_gains_feed';
    fprintf(1, '\n==================================== \nAUTOPILOT: SLC with
Feedforward\n \n')
else
    gains_file = 'ttwistor_gains_slc';
    fprintf(1, '\n ==================================== \nAUTOPILOT: Simple
SLC\n \n')
end

load(gains_file)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set input commands for autopilot.
%
% Note, STUDENTS may need to change these while tuning the autopilot.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Course testing
% h_c           = h_trim;  % commanded altitude (m)
% h_dot_c       = 0;  % commanded altitude rate (m)
% chi_c         = 40*pi/180;  % commanded course (rad)
% chi_dot_ff    = 0;  % commanded course rate (rad)
% Va_c          = V_trim;  % commanded airspeed (m/s)

% Height testing
% h_c           = h_trim + 50;  % commanded altitude (m)
% h_dot_c       = 0;  % commanded altitude rate (m)
% chi_c         = 0;  % commanded course (rad)
% chi_dot_ff    = 0;  % commanded course rate (rad)
% Va_c          = V_trim;  % commanded airspeed (m/s)

% Velocity testing
h_c           = h_trim;  % commanded altitude (m)
h_dot_c       = 0;  % commanded altitude rate (m)
chi_c         = 0;  % commanded course (rad)
chi_dot_ff    = 0;  % commanded course rate (rad)
Va_c          = V_trim;  % commanded airspeed (m/s)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set aircraft and simulation initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aircraft_state0 = aircraft_state_trim;
control_input0 = control_input_trim;
```

```
wind_inertial = [0;0;0];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LINE FOLLOWING PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define line
%line_vec = [1;1;0.3];
line_vec = [-1;-1;0.3];
line.q = line_vec ./ norm(line_vec);
line.r = [0; 0; h_trim];

% Define start xy position of the models
% start_pos = [-1500; 1000];
%start_pos = [-1500; 1000];
%start_pos = [2000; -1000];
start_pos = [-1500; 0];
%start_pos = [0; 0];

% Define guidance algorithm parameters
algo_params.vel_d = V_trim;
algo_params.lookahead = 200;
```

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than the value of the step size tolerance and constraints are satisfied to within the value of the constraint tolerance.

```
 ====================================
AUTOPILOT: Simple SLC
```

Determine Guidance Model Parameters

```
% Course angle
wnx = sqrt(9.81*control_gain_struct.Ki_course / (V_trim));
dampx = control_gain_struct.Kp_course * 9.81 / (2*wnx*V_trim);

params.bx = wnx^2;
params.bx_dot = 2*wnx*dampx;

% Tune parameters to match the curves
params.bx = params.bx * 90;
params.bx_dot = params.bx_dot * 4;

% Height hold
wnh = sqrt(control_gain_struct.Kpitch_DC * V_trim *
control_gain_struct.Ki_height);
damph = control_gain_struct.Kpitch_DC * V_trim *
control_gain_struct.Kp_height / (2*wnh);
```

```
params.ah_dot = 1;
params.bh = wnh^2;
params.bh_dot = 2*damph*wnh;

% Airspeed hold
params.bva = 0.2;
```

Simulate Autopilot and Guidance Kinematic Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Ts = .1;
Tfinal = 300;
control_gain_struct.Ts=Ts;
```

iterate at control sample time

```
n_ind = Tfinal/Ts;

% Autopilot initial conditions
aircraft_array(:,1) = aircraft_state0;
control_array(:,1) = control_input0;
time_iter(1) = 0;

% Kinematic model initial conditions
guid_init = [0; 0; h_trim; 0; 0; 0; V_trim];
model_state = guid_init;
time_model(1) = 0;

% Set nonzero initial E and N for kinematic and
model_state(1) = start_pos(1);
aircraft_array(1,1) = start_pos(1);
model_state(2) = start_pos(2);
aircraft_array(2,1) = start_pos(2);

% Simulate
for i=1:n_ind
    %%%%%%%%%%%%%%% AUTOPILOT
    TSPAN = Ts*[i-1 i];

    wind_array(:,i) = wind_inertial;

    wind_body = TransformFromInertialToBody(wind_inertial,
aircraft_array(4:6,i));
    air_rel_vel_body = aircraft_array(7:9,i) - wind_body;
    wind_angles(:,i) =
AirRelativeVelocityVectorToWindAngles(air_rel_vel_body);
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Guidance level commands
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % control_objectives(1) = h_c;
    % control_objectives(2) = h_dot_c;
    % control_objectives(3) = chi_c;
    % control_objectives(4) = chi_dot_ff;
    % control_objectives(5) = Va_c;

    % Get desired velocity
    line_guid_state = aircraft_array(1:3,i);
    line_guid_state(3) = -line_guid_state(3);
    [vel_d, next_waypoint] = lineTrackingGuidance(Ts*(i-1), line_guid_state,
line, algo_params);
    control_objectives = getControlObjectives(vel_d, next_waypoint);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Autopilot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Autopilot output
    [control_out, x_c_out] = SimpleSLCAutopilot(Ts*(i-1),
aircraft_array(:,i), wind_angles(:,i), control_objectives,
control_gain_struct);

    control_array(:,i) = control_out;
    x_command(:,i) = x_c_out;
    x_command(5,i) = trim_variables(1);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Aircraft dynamics
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [TOUT2,YOUT2] = ode45(@(t,y)
AircraftEOM(t,y,control_array(:,i),wind_inertial,aircraft_parameters),TSPAN,a
ircraft_array(:,i),[]);


    aircraft_array(:,i+1) = YOUT2(end,:)';
    time_iter(i+1) = TOUT2(end);
    wind_array(:,i+1) = wind_inertial;
    control_array(:,i+1) = control_array(:,i);
    x_command(:,i+1) = x_command(:,i);


    %%%%%%%%%%%%%%% KINEMATIC MODEL
    % Overall Guidance model simulation
    wind_inertial = [0;0;0];
    line_track_state = [model_state(1,i); model_state(2,i);
model_state(3,i)];
    [vel_d, waypoint] = lineTrackingGuidance(Ts*(i-1), line_track_state,
line, algo_params);
    xc = getControlObjectives(vel_d, waypoint);
    guidFunc = @(t, x)guidanceEOM(t, x, xc, wind_inertial, params);
```

```
    % Simulate with ode
    [TOUT_model, YOUT_model] = ode45(guidFunc, TSPAN, model_state(:,i));

    % Update the model state
    model_state(:,i+1) = YOUT_model(end,:)';
    time_model(i+1) = TOUT_model(end);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%PlotSimulationWithCommands(time_iter,aircraft_array,control_array,
wind_array, x_command, 'b')
```

Nonlinear guidance simulation

```
% Course angle
xc = [chi_c; chi_dot_ff];
courseFunc = @(t, x)courseEOM(t, x, xc, params);

% Simulate with ode
tspan = [0 Tfinal];
[TOUT_course, YOUT_course] = ode45(courseFunc, tspan, [0;0]);


% Height
xc = [h_c; h_dot_c];
heightFunc = @(t, x)heightEOM(t, x, xc, params);

% Simulate with ode
height_init = [h_trim; 0];
tspan = [0 Tfinal];
[TOUT_height, YOUT_height] = ode45(heightFunc, tspan, height_init);


% Airspeed
xc = [Va_c];
velFunc = @(t, x)velocityEOM(t, x, xc, params);

% Simulate with ode
vel_init = [V_trim];
tspan = [0 Tfinal];
[TOUT_vel, YOUT_vel] = ode45(velFunc, tspan, vel_init);


% Overall Guidance model simulation
wind_inertial = [0;0;0];
xc = [chi_c; chi_dot_ff; h_c; h_dot_c; Va_c];
guidFunc = @(t,x)guidanceEOM(t, x, xc, wind_inertial, params);
```

```
% Simulate with ode
guid_init = [0; 0; 0; 0; h_trim; 0; V_trim];
tspan = [0 Tfinal];
[TOUT_guid, YOUT_guid] = ode45(guidFunc, tspan, guid_init);
```

Guidance Algorithm

```
% First order guidance algorithm simulation
% Simulate with ode
algoFunc = @(t, state)lineTrackingGuidance(t, state, line, algo_params);
algo_init = [start_pos(1); start_pos(2); h_trim];
tspan = [0 300];
[TOUT_alg, YOUT_alg] = ode45(algoFunc, tspan, algo_init);


% Test guidance algorithm with the kinematic guidance model
```

Plot Guidance Model Variables To Get Kinematic Model

```
ind_f = length(aircraft_array(1,:));
```

Replicate the plot from PlotStateVariables

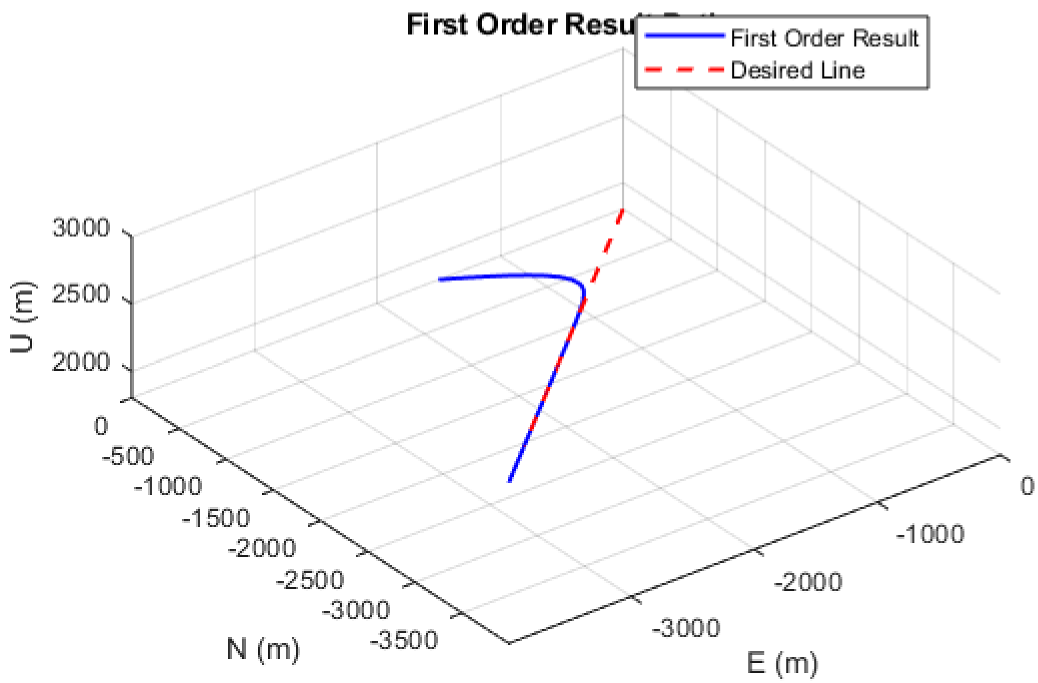
```
for i=1:ind_f
    wind_inertial = wind_array(1:3,i);
    wind_body = TransformFromInertialToBody(wind_inertial,
aircraft_array(4:6,i));
    air_rel_body = aircraft_array(7:9,i) - wind_body;
    wind_angles = AirRelativeVelocityVectorToWindAngles(air_rel_body);
    [flight_angles] = FlightPathAnglesFromState(aircraft_array(1:12,i));

    % Get the air relative velocity over time
    Va(i) = wind_angles(1);

    % Get the course angle over time
    chi(i) = 180/pi*flight_angles(2);
end

% Course angle plot
% figure();
% plot(time_iter, chi, 'linewidth', 2, 'color', 'r');
% hold on;
% plot(TOUT_course, 180/pi .* YOUT_course(:,1), 'linewidth', 2, 'color',
'b');
% yline(chi_c*180/pi, '--', 'color', 'g')
%
% xlabel('Time (s)');
% ylabel('\chi_c (deg)')
% title('Course Angle Response')
```

```
% legend('Autopilot', 'Guidance Model')
%
% % Height plot
% figure();
% plot(time_iter, -aircraft_array(3,:), 'linewidth', 2, 'color', 'r');
% hold on;
% plot(TOUT_height,YOUT_height(:,1), 'linewidth', 2, 'color', 'b');
% yline(h_c, '--', 'color', 'g')
%
% xlabel('Time (s)');
% ylabel('h_c (m)')
% title('Height Response')
% legend('Autopilot', 'Guidance Model')
%
% % Velocity plot
% figure();
% plot(time_iter, Va, 'linewidth', 2, 'color', 'r');
% hold on;
% plot(TOUT_vel,YOUT_vel(:,1), 'linewidth', 2, 'color', 'b');
% yline(Va_c, '--', 'color', 'g')
%
% xlabel('Time (s)');
% ylabel('Va_c (m/s)')
% title('Velocity Response')
% legend('Autopilot', 'Guidance Model')
```

Plot Guidance Algorithm Results

```
% Create line plot
line_length = 5000;
point1 = line.r;
point2 = line.r + line_length * line.q;

% Plot the desired line and first order result
figure();
plot(YOUT_alg(:,1), YOUT_alg(:,2), 'b-', 'LineWidth', 1.5, 'DisplayName',
'First Order Result');
hold on;
plot([point1(1), point2(1)], [point1(2), point2(2)], 'r--', 'LineWidth',
1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
title('First Order Result Path');
axis equal;
grid on;
legend('show', 'Location', 'best');


% First order result in 3D
figure();
plot3(YOUT_alg(:,1), YOUT_alg(:,2), YOUT_alg(:,3), 'b-', 'LineWidth', 1.5,
'DisplayName', 'First Order Result');
```

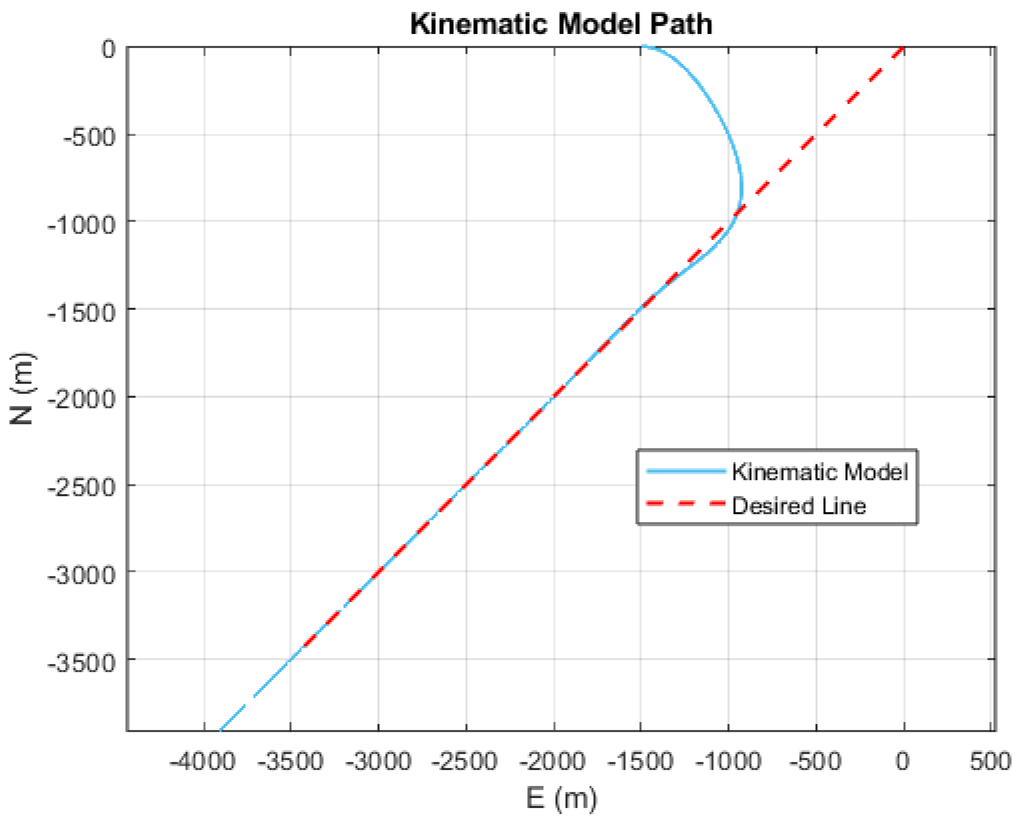
```
hold on;
plot3([point1(1), point2(1)], [point1(2), point2(2)], [point1(3),
point2(3)], ...
      'r--', 'LineWidth', 1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
zlabel('U (m)');
title('First Order Result Path');
grid on;
axis equal;
legend('show', 'Location', 'best');
view(3);
```

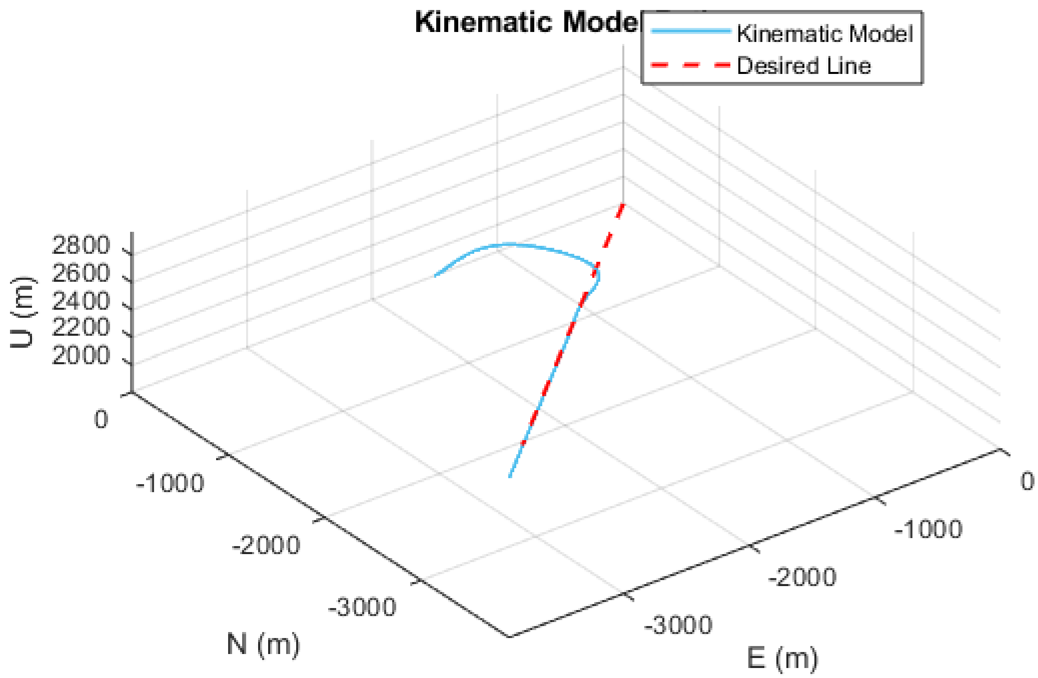


Plot Guidance Model with Guidance Algorithm

```
% Plot the desired line and first-order result
figure();
plot(model_state(1,:), model_state(2,:), '-','color', [0.3010 0.7450
0.9330], 'LineWidth', 1.5, 'DisplayName', 'Kinematic Model');
hold on;
plot([point1(1), point2(1)], [point1(2), point2(2)], 'r--', 'LineWidth',
1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
title('Kinematic Model Path');
axis equal;
grid on;
legend('show', 'Location', 'best');

% 3D
figure();
plot3(model_state(1,:), model_state(2,:), model_state(3,:), '-', 'color',
[0.3010 0.7450 0.9330], 'LineWidth', 1.5, 'DisplayName', 'Kinematic Model');
hold on;
plot3([point1(1), point2(1)], [point1(2), point2(2)], [point1(3),
point2(3)], ...
      'r--', 'LineWidth', 1.5, 'DisplayName', 'Desired Line');
```

```
xlabel('E (m)');
ylabel('N (m)');
zlabel('U (m)');
title('Kinematic Model Path');
grid on;
axis equal;
legend('show', 'Location', 'best');
view(3);
```

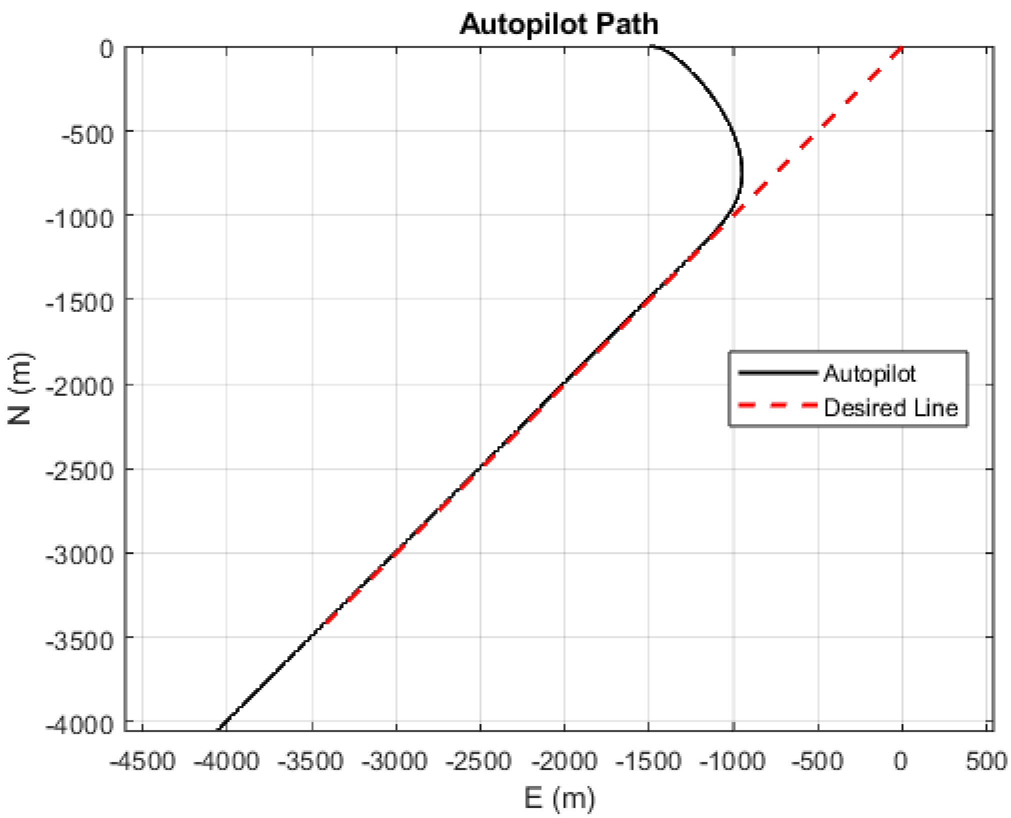


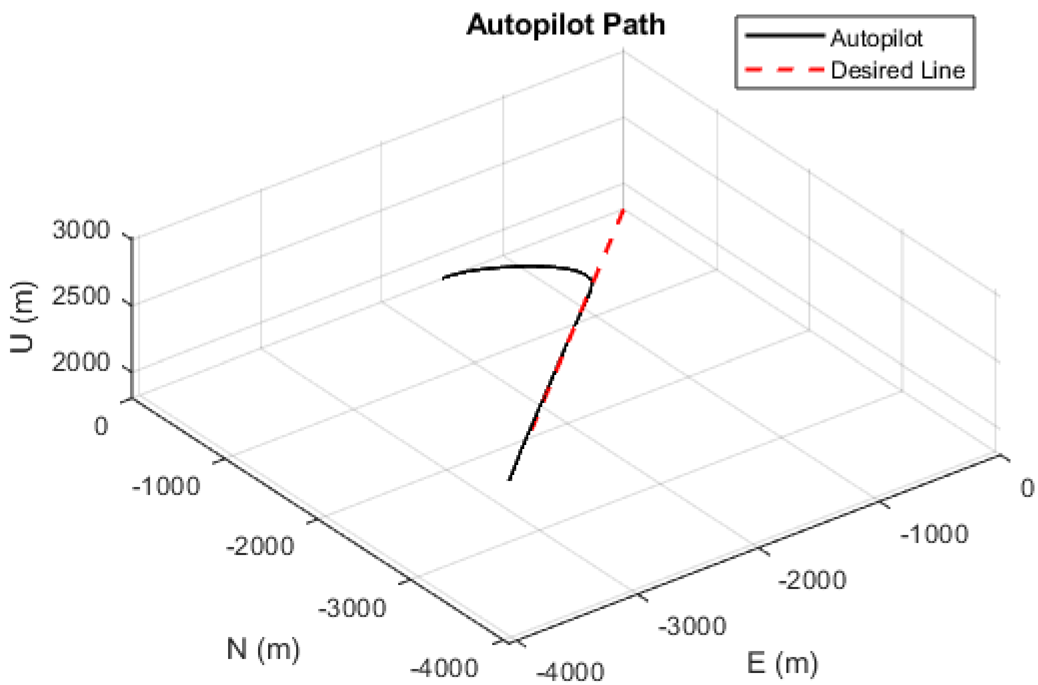


Plot Autopilot with Guidance Algorithm

```
% Plot the desired line and first-order result
figure();
plot(aircraft_array(1,:), aircraft_array(2,:), 'k-', 'LineWidth', 1.5,
'DisplayName', 'Autopilot');
hold on;
plot([point1(1), point2(1)], [point1(2), point2(2)], 'r--', 'LineWidth',
1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
title('Autopilot Path');
axis equal;
grid on;
legend('show', 'Location', 'best');

% 3D
figure();
plot3(aircraft_array(1,:), aircraft_array(2,:), -aircraft_array(3,:), 'k-',
'LineWidth', 1.5, 'DisplayName', 'Autopilot');
hold on;
plot3([point1(1), point2(1)], [point1(2), point2(2)], [point1(3),
point2(3)], ...
      'r--', 'LineWidth', 1.5, 'DisplayName', 'Desired Line');
```

```
xlabel('E (m)');
ylabel('N (m)');
zlabel('U (m)');
title('Autopilot Path');
grid on;
axis equal;
legend('show', 'Location', 'best');
view(3);
```

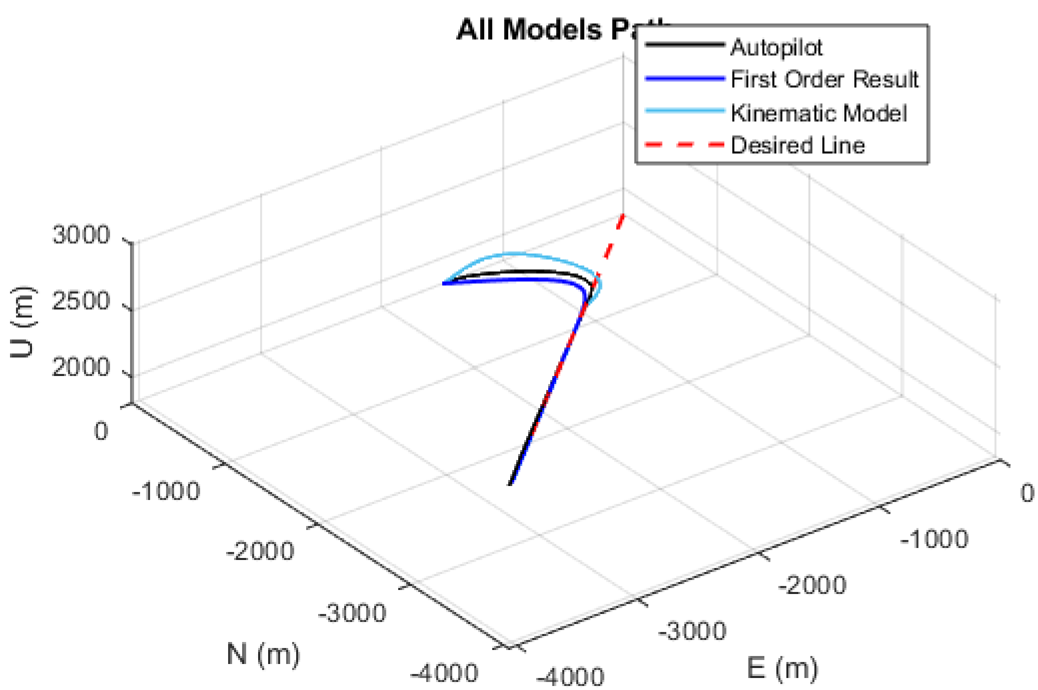


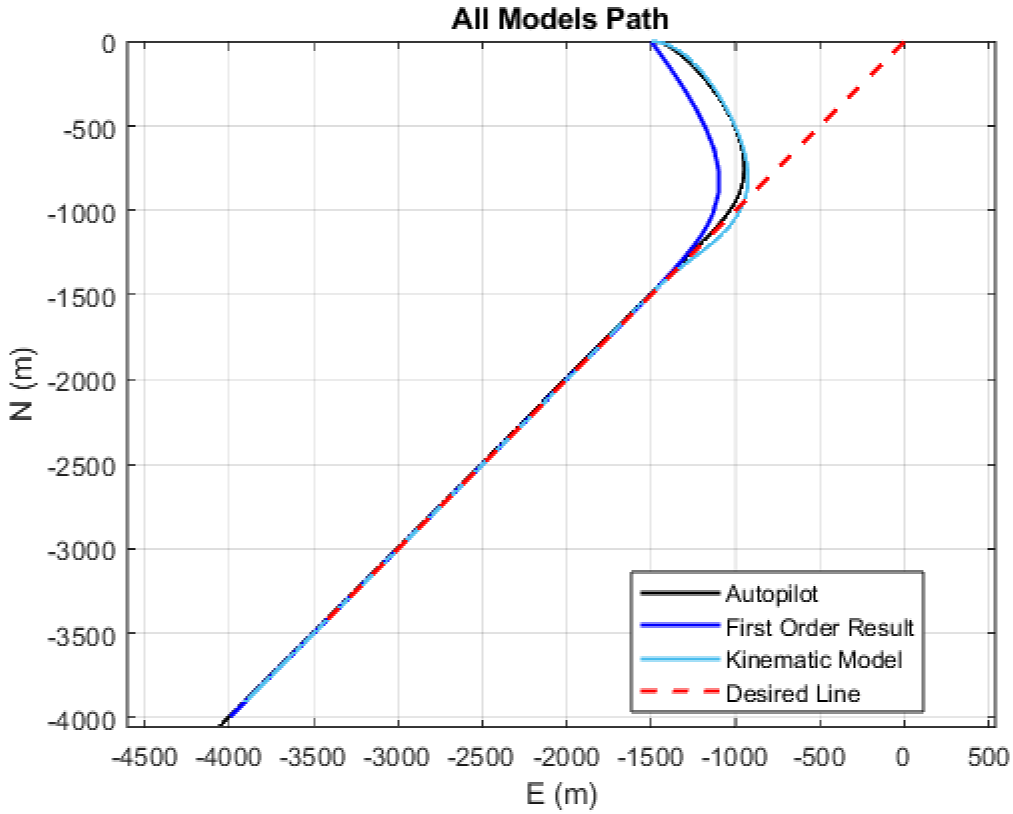


3D plot all together

```
% 3D
figure();
plot3(aircraft_array(1,:), aircraft_array(2,:), -aircraft_array(3,:), 'k-',
'LineWidth', 1.5, 'DisplayName', 'Autopilot');
hold on;
plot3(YOUT_alg(:,1), YOUT_alg(:,2), YOUT_alg(:,3), 'b-', 'LineWidth', 1.5,
'DisplayName', 'First Order Result');
plot3(model_state(1,:), model_state(2,:), model_state(3,:), '-', 'color',
[0.3010 0.7450 0.9330], 'LineWidth', 1.5, 'DisplayName', 'Kinematic Model');
plot3([point1(1), point2(1)], [point1(2), point2(2)], [point1(3),
point2(3)], ...
      'r--', 'LineWidth', 1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
zlabel('U (m)');
title('All Models Path');
grid on;
axis equal;
legend('show', 'Location', 'best');
view(3);

% 2D
```

```
figure();
plot(aircraft_array(1,:), aircraft_array(2,:), 'k-', 'LineWidth', 1.5,
'DisplayName', 'Autopilot');
hold on;
plot(YOUT_alg(:,1), YOUT_alg(:,2), 'b-', 'LineWidth', 1.5, 'DisplayName',
'First Order Result');
plot(model_state(1,:), model_state(2,:), '-','color', [0.3010 0.7450
0.9330], 'LineWidth', 1.5, 'DisplayName', 'Kinematic Model');
plot([point1(1), point2(1)], [point1(2), point2(2)], 'r--', 'LineWidth',
1.5, 'DisplayName', 'Desired Line');

xlabel('E (m)');
ylabel('N (m)');
title('All Models Path');
axis equal;
grid on;
legend('show', 'Location', 'best');
```





Problem 5 Plots

```
% Plot the airspeed, course angle, and height on the same plots
% Get the course angle for the first order system
chi_first_order = zeros(length(YOUT_alg(:,1)), 1);
Va_first_order = zeros(length(YOUT_alg(:,1)), 1);
for i = 1:length(chi_first_order)
    [vel_d, waypoint] = lineTrackingGuidance(0, YOUT_alg(i,:)', line,
algo_params);
    control_objectives_first = getControlObjectives(vel_d, waypoint);
    chi_first_order(i) = control_objectives_first(3);
end

% Course angle plot
figure();
plot(time_iter, chi, 'linewidth', 2, 'color', 'k');
hold on;
plot(time_iter, 180/pi .* model_state(5,:), 'linewidth', 2, 'color', [0.3010
0.7450 0.9330])
plot(TOUT_alg, 180/pi .* chi_first_order, 'linewidth', 2, 'color', 'b');

grid on
xlabel('Time (s)');
ylabel('\chi (deg)')
title('Course Angle Response')
```

```
legend('Autopilot', 'Kinematic Model', 'First Order Model')

% Height plot
figure();
plot(time_iter, -aircraft_array(3,:), 'linewidth', 2, 'color', 'k');
hold on;
grid on
plot(time_iter, model_state(3,:), 'linewidth', 2, 'color', [0.3010 0.7450
0.9330]);
plot(TOUT_alg, YOUT_alg(:,3), 'linewidth', 2, 'color', 'b');

xlabel('Time (s)');
ylabel('h (m)')
title('Height Response')
legend('Autopilot', 'Kinematic Model', 'First Order Model')

% % Velocity plot
figure();
plot(time_iter, Va, 'linewidth', 2, 'color', 'k');
hold on;
grid on
plot(time_iter, model_state(7,:), 'linewidth', 2, 'color', [0.3010 0.7450
0.9330]);
yline(18, 'linewidth', 2, 'color', 'b')

xlabel('Time (s)');
ylabel('Va (m/s)')
title('Velocity Response')
legend('Autopilot', 'Kinematic Model', 'First Order Model')
```

EOM Functions

```
% Course angle EOM of nonlinear guidance model
function dxdt = courseEOM(t, x, xc, params)
    % Current state
    chi = x(1);
    chi_dot = x(2);

    % Desired state
    chi_c = xc(1);
    chi_c_dot = xc(2);

    % ROC
    dxdt = [chi_dot; params.bx_dot*(chi_c_dot - chi_dot) + params.bx*(chi_c
- chi)];
end

% Height EOM of nonlinear guidance model
function dxdt = heightEOM(t, x, xc, params)
    % Current state
    h = x(1);
    h_dot = x(2);
```

```
    % Desired state
    h_c = xc(1);
    h_c_dot = xc(2);

    % ROC
    dxdt = [h_dot; -params.ah_dot * h_c_dot - params.bh_dot * h_dot +
params.bh*(h_c - h)];
end

% Velocity EOM for guidance model
function dvdt = velocityEOM(t, x, xc, params)
    % Current state
    Va = x(1);

    % Desired state
    Va_c = xc(1);

    % ROC
    dvdt = params.bva * (Va_c - Va);
end
```

External Functions

```
function [vel_vector, next_waypoint] = lineTrackingGuidance(t, state, line,
params)
```

```
%LINETRACKINGGUIDANCE Calculates a desired velocity vector given a line
%definition and a current position in 3D space
% Inputs:
%   line: struct with fields line.q unit vector of line and line.r point
%   state: [x; y; z] position
%   params: parameters for gain values

% Get state values
x = state(1);
y = state(2);
z = state(3);

% Line parameters
r = line.r;
q = line.q;
```

Longitudinal tracking

Find point along the line nearest to the current location

```
p = [x; y; z];
pr = p - r;

% Project onto q
q_proj = dot(pr, q) / dot(q, q) .* q;

% Nearest point
```

```
nearest_point = r + q_proj;

% Determine next waypoint to track
next_waypoint = nearest_point + (params.lookahead .* q);

% Get vector from current position to new lookahead position
dir_vector = (next_waypoint - p) ./ norm(next_waypoint - p);

% Use desired airspeed to get desired velocity vector
vel_vector = dir_vector .* params.vel_d;

end


function control_objectives = getControlObjectives(vel_d, waypoint)
%GET This function takes as input a desired velocity vector and outputs the
%desired course angle, airspeed, and climb rate
% Inputs:
%   vel_d = desired velocity vector in intertial coordinates
%   waypoint = next desired waypoint for the path following
% Outputs:
%   control_objectives = [h_c; h_dot_c; chi_c; chi_dot_ff; Va_c]

% Desired course angle
chi_c = atan2(vel_d(2), vel_d(1));
chi_dot_ff = 0;

% Desired height
h_dot_c = vel_d(3);
h_dot_c = 0;

% Make desired height the position propagation after 1 second in the
% veritcal direction
h_c = waypoint(3);

% Desired airspeed
Va_c = norm(vel_d);

control_objectives = [h_c; h_dot_c; chi_c; chi_dot_ff; Va_c];
end


% Full guidance model EOM
function dxdt = guidanceEOM(t, x, xc, wind_inertial, params)
% Calculates the rate of change of the kinematic model 1a variables to be
% used in ode45
% Inputs:
%   t = time
%   x = state [Pn; Pe; h; hdot; chi; chi_dot; Va]
%   xc = control objectives [h_c; h_c_dot; chi_c; chi_c_dot; Va_c]
%   wind_inertial = inertial wind vector in inertial coordinates
%   params = struct with tuning constants for the model
% Outputs:
%   dxdt = rate of change of the variables of the state
```

```
% Get the full state
Pn = x(1);
Pe = x(2);
h = x(3);
h_dot = x(4);
chi = x(5);
chi_dot = x(6);
Va = x(7);

% Desired states
h_c = xc(1);
h_c_dot = xc(2);
chi_c = xc(3);
chi_c_dot = xc(4);
Va_c = xc(5);

% Calculate psi
psi = chi - asin(1/Va * wind_inertial(1:2)' * [-sin(chi); cos(chi)]);

% ROC
dpndt = Va*cos(psi) + wind_inertial(1);
dpedt = Va*sin(psi) + wind_inertial(2);
dhdt = [h_dot; -params.ah_dot * h_c_dot - params.bh_dot * h_dot +
params.bh*(h_c - h)];
dchidt = [chi_dot; params.bx_dot*(chi_c_dot - chi_dot) + params.bx*(chi_c -
chi)];
dvdt = params.bva * (Va_c - Va);

% ROC of the state
dxdt = [dpndt; dpedt; dhdt; dchidt; dvdt];

end
```

```
Altitude mode: Climb
Altitude mode: Altitude Hold
Altitude mode: Climb
```

Published with MATLAB® R2023b