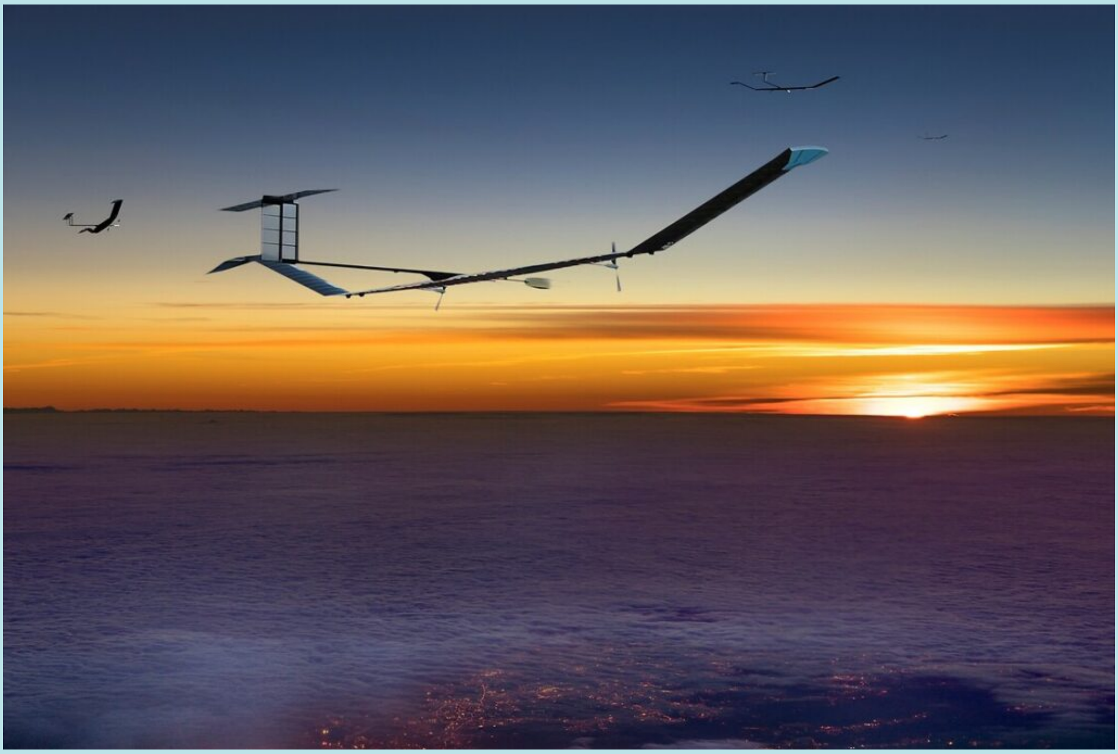

UAS with Matlab/Simulink

Elaborato di Unmanned Aircraft Systems

Antonio Carotenuto

A.A 2024-2025

Indice

Elenco delle figure

Capitolo 1

Exercise 1: Pose Estimation

1.1 Procedure

The objective of this exercise is to estimate the pose of the UAV during its landing by using the images captured by the drone's camera and leveraging the knowledge of the position of the AprilTags. In the following Matlab code, this procedure is implemented:

- Import video and trajectory of the UAV (the Mathlab function 'VideoReader' is used)
- Postprocess in MATLAB the video to obtain image frames ('readFrame', 'imwrite');
- Detect the AprilTag of interest, identify and find the '2D pixel position' of its corners('readAprilTag').
- Solve the PNP problem ('estimateWorldCameraPose')
- Plot the errors of the pose estimates along the landing trajectory.

Listing 1.1

```
clc;
clear;
close all;

vid = VideoReader('Agnano_Multiscale_Vertiport.avi');
numFrames = vid.NumFrames;
for i = 1:numFrames
    frames = readFrame(vid);
    if i > 1 % Not write the 1st frame because it is a repetition of the
   2nd one
        imwrite(frames, ['ImagesUAS/Agnano_UAS_' int2str(i-1) '.bmp']);
    end
end

% Camera Parameters
```

```
focalLength = [1109, 1109];
principalPoint = [808, 640];
imageSize = [1280, 1616];

intrinsics = cameraIntrinsics(focalLength, principalPoint, imageSize);

% Initializations
estimatePosition = zeros(400, 3);
estimateYaw = zeros(400, 1);
estimatePitch = zeros(400, 1);
estimateRoll = zeros(400, 1);

%%
% Corner position of the Marker in NED
Marker_NED36h11 = [
    -5,  5, -29;
     5,  5, -29;
     5, -5, -29;
    -5, -5, -29
];

Marker_NEDcircle21h7 = [
     6.75, -3.85, -29;
    10.25, -3.85, -29;
    10.25, -7.35, -29;
     6.75, -7.35, -29
];

Marker_NED25h9 = [
     8,  0, -29;
    10,  0, -29;
    10, -2, -29;
     8, -2, -29
];



%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% From CRF To BODY %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RcameraToBody = angle2dcm(deg2rad(90), deg2rad(70-90), deg2rad(0));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%





% Import true data and initializations
trajectory = load('UAS_trajectory_24_25.mat');
xyzNED(:,1) = trajectory.N;
xyzNED(:,2) = trajectory.E;
xyzNED(:,3) = trajectory.D;
realYaw = trajectory.Yaw;
```

```
realPitch = trajectory.Pitch;
realRoll = trajectory.Roll;

error = zeros(400, 6);
j = 1;

%%
for i = 1:400
    try
        image = imread(['ImagesUAS/Agnano_UAS_' int2str(i) '.bmp']);

        tagFamily = ["tag36h11", "tagCircle21h7", "tag25h9"];

        [id_36h11, loc_36h11, detectedFamily_36h11] = readAprilTag(image, tagFamily(1));
        [id_Circle21h7, loc_Circle21h7, detectedFamily_Circle21h7] = readAprilTag(image, tagFamily(2));
        [id_25h9, loc_25h9, detectedFamily_25h9] = readAprilTag(image, tagFamily(3));



if ~isempty(loc_36h11) && ~isempty(loc_Circle21h7) % Se entrambi i tag 36h11 e 25h9 sono presenti
    [worldOrientation, worldLocation] = estimateWorldCameraPose([loc_36h11(:,:); loc_Circle21h7(:,:)], ...
        [Marker_NED36h11; Marker_NEDcircle21h7], intrinsics);
    estimatePosition(i, :) = worldLocation;

    %% HERE
    [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(RcameraToBody * worldOrientation', 'ZYX', 'Robust');


elseif ~isempty(loc_36h11) % Se   presente solo il tag 36h11
    [worldOrientation, worldLocation] = estimateWorldCameraPose(loc_36h11(:,:), Marker_NED36h11, intrinsics);
    estimatePosition(i, :) = worldLocation;

     %% HERE
    [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(RcameraToBody * worldOrientation', 'ZYX', 'Robust');


elseif ~isempty(loc_Circle21h7) % Se   presente solo il tag 25h9
    [worldOrientation, worldLocation] = estimateWorldCameraPose(loc_Circle21h7(:,:), Marker_NEDcircle21h7, intrinsics);
    estimatePosition(i, :) = worldLocation;

    %% HERE
    [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(RcameraToBody * worldOrientation', 'ZYX', 'Robust');



elseif ~isempty(loc_25h9) % Se nessuno dei tag precedenti   presente, si
```

```
    usa il tag 16h5
    [worldOrientation, worldLocation] = estimateWorldCameraPose(loc_25h9
   (:,:), Marker_NED25h9, intrinsics);
    estimatePosition(i, :) = worldLocation;

    %% HERE
    [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(
   RcameraToBody * worldOrientation', 'ZYX', 'Robust');


end

 catch
    disp(['errore alla ', num2str(i), ' iterazione']);
end

%%
if (estimatePosition(i,1) ~= 0) %&& (estimatePosition(i,2) ~= 0) && (
   estimatePosition(i,3) ~= 0)
    error(i,1) = xyzNED(i,1) - estimatePosition(i,1);
    error(i,2) = xyzNED(i,2) - estimatePosition(i,2);
    error(i,3) = xyzNED(i,3) - estimatePosition(i,3);
    error(i,4) = realPitch(i) - estimatePitch(i);
    error(i,5) = realRoll(i) - estimateRoll(i);
    error(i,6) = realYaw(i) - estimateYaw(i);

    j = j + 1;
end
end
estimatePosition(:,1) = xyzNED(:,1) - error(:,1);
estimatePosition(:,2) = xyzNED(:,2) - error(:,2);
estimatePosition(:,3) = xyzNED(:,3) - error(:,3);

estimatePitch = realPitch - error(:,4);
estimateRoll = realRoll - error(:,5);
estimateYaw = realYaw - error(:,6);
```

1.2 Results

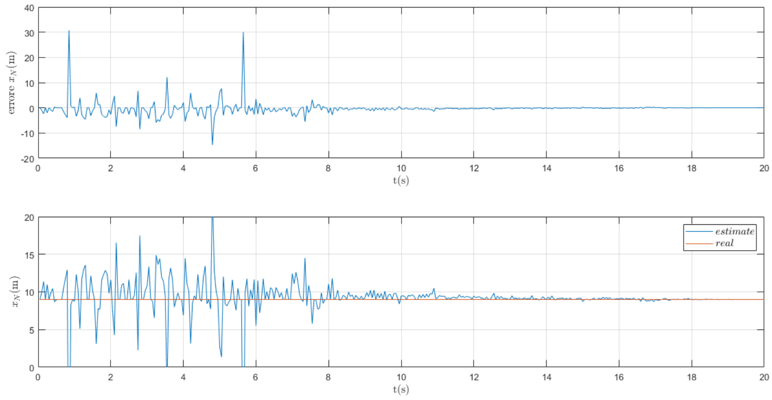

Figura 1.1 North position

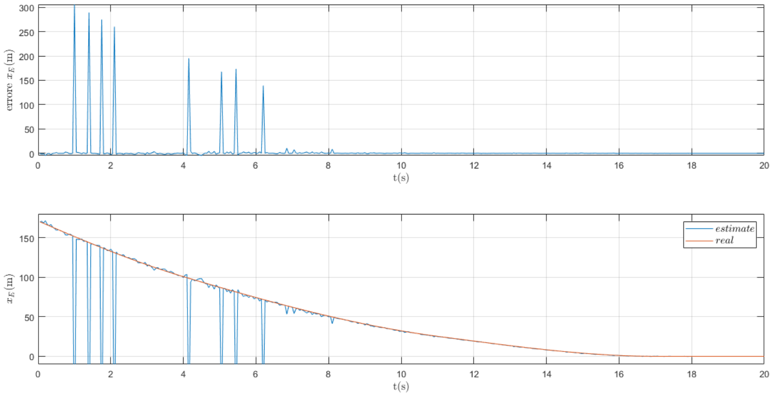

Figura 1.2 East position

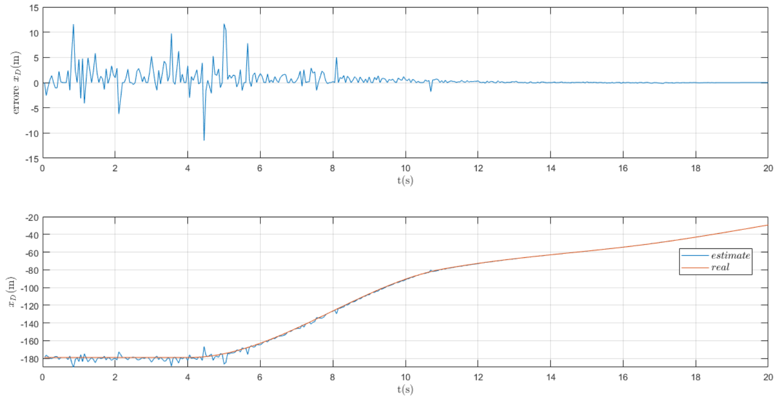

Figura 1.3 Down position

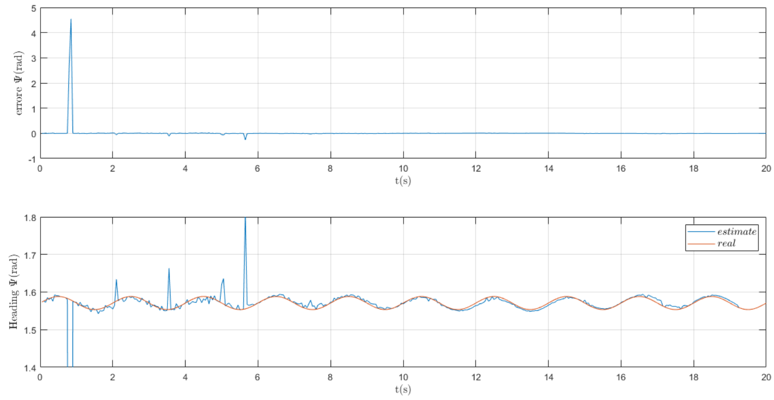

Figura 1.4 Heading

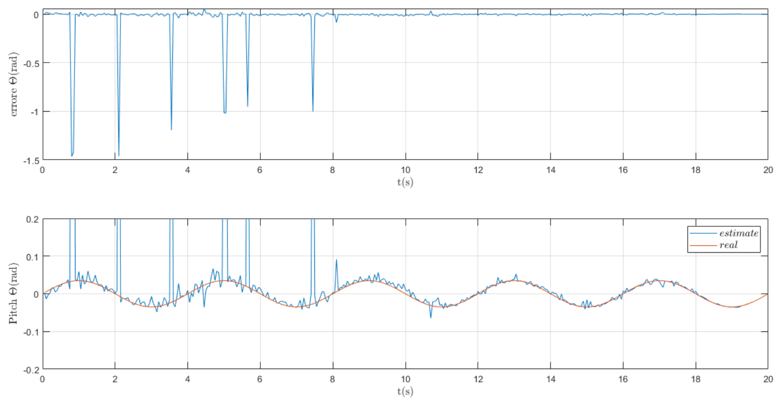

Figura 1.5 Pitch

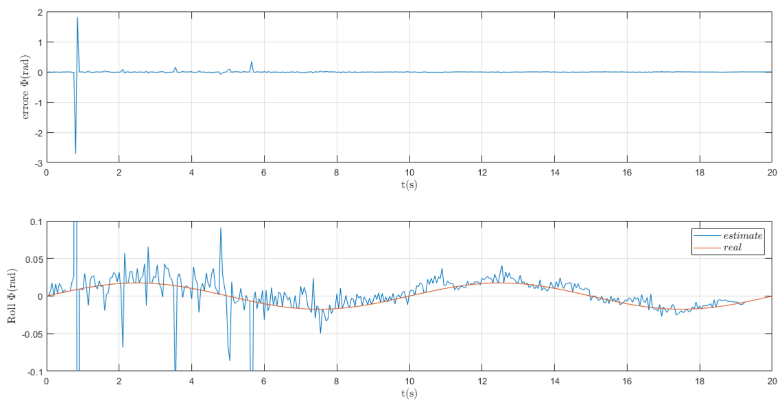

Figura 1.6 Roll

Capitolo 2

Exercise 2: Autopilot

The objective of this exercise is to change some parametres in the autopilot model build by Milone, Donnarumma e Norcaro with Matlab and Simulink. The model is used to study the change of the behaviour of the UAV due to different value for the Proportional gain K_p and Dumping ζ in the Pitch Loop. We can see that the higher is the proportional gain the more responsive is the system and the higher is the dumping the smaller are the oscillation.

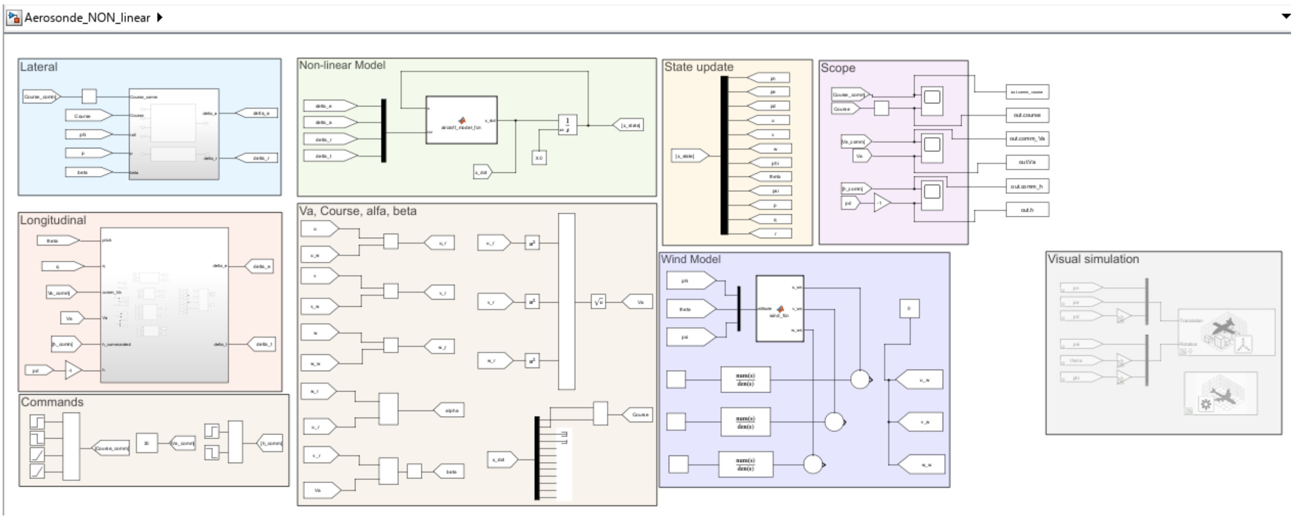

Figura 2.1 Simulink model, some Block parameters:'Toworkspace' are implemented to save the results

```
%% cambio e_max %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_e_max = 0.43;

%e1
%e_teta_max = 0.1; % kp

%e2
e_teta_max = 0.005;

%e3
%e_teta_max = 0.5;


kp_pitch = delta_e_max / e_teta_max * sign(a_teta3);
wn_pitch = sqrt(a_teta2+kp_pitch*a_teta3);
```

Figura 2.2 Different values for e_{max} are considered: 0.1, 0.005, 0.5

```
%% cambio zita considero e2
%zita 1
%zita_pitch = 0.9; % kd

%zita 2
%zita_pitch = 0.1;

%zita 3
zita_pitch = 0.001;

kd_pitch = (2*zita_pitch * wn_pitch -a_teta1)/a_teta3;
```

Figura 2.3 Different values for ζ are considered: 0.9, 0.1, 0.001

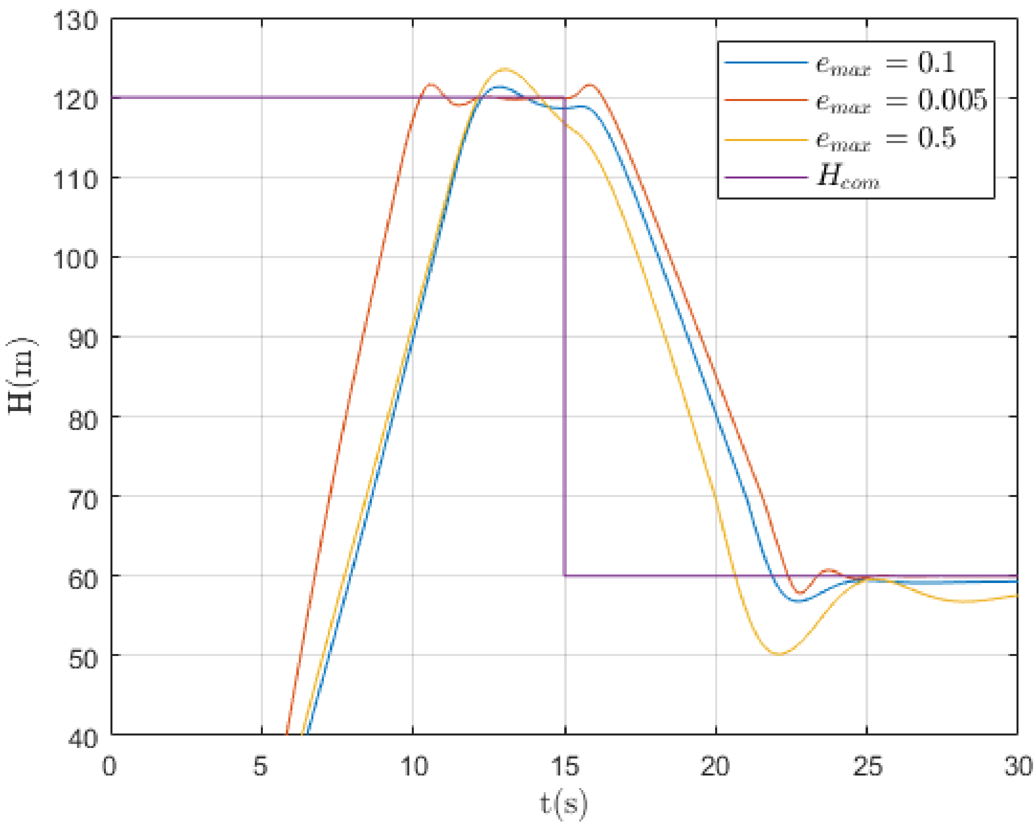

Figura 2.4 Effect of change of K_p on vertical dinamyc

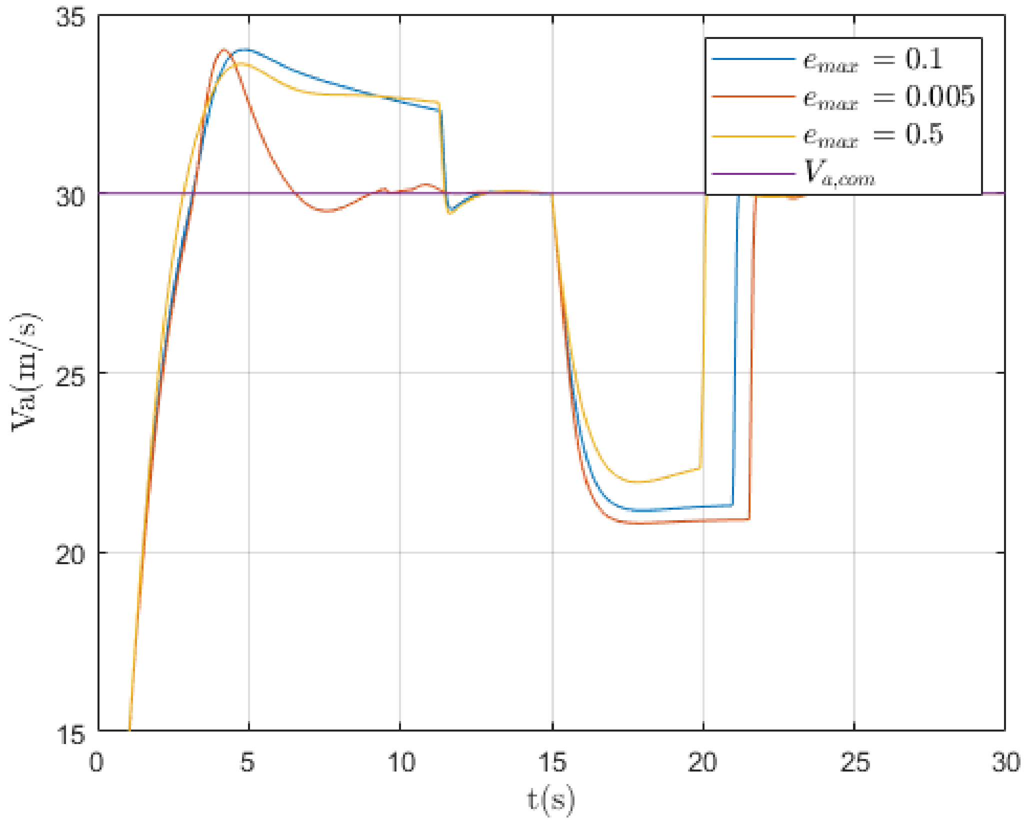

Figura 2.5 Effect of change of k_p on airspeed

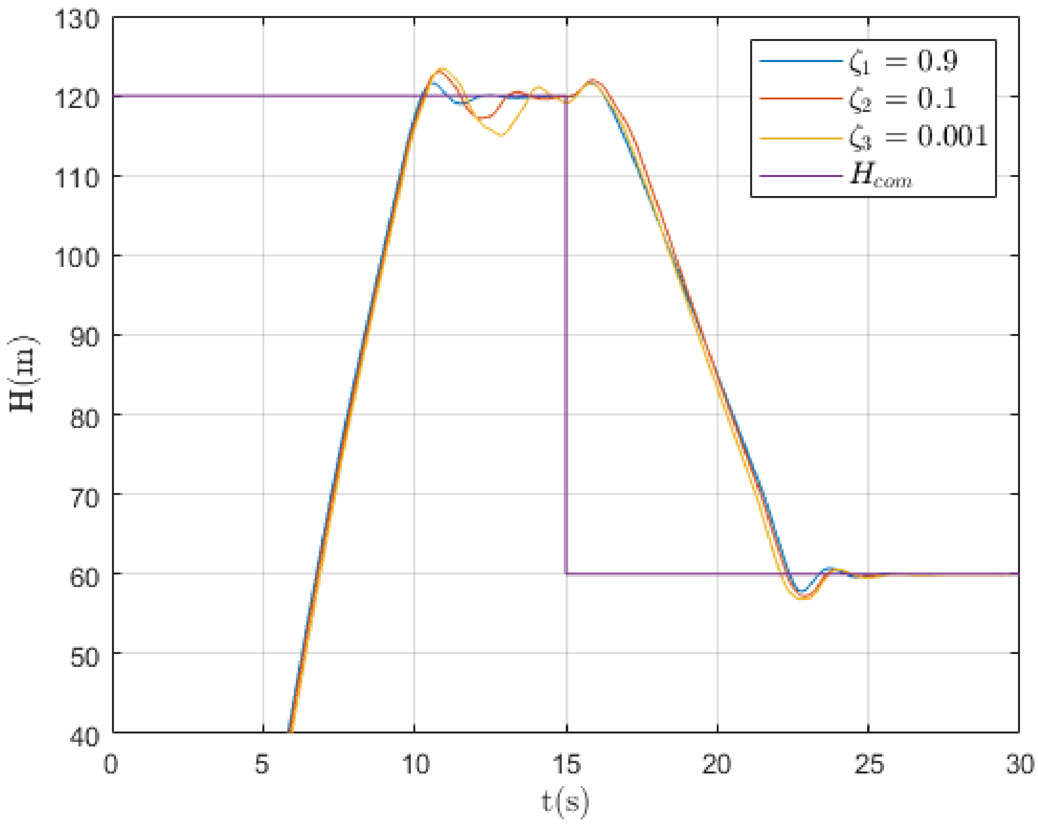

Figura 2.6 Effect of change of ζ on vertical dinamyc

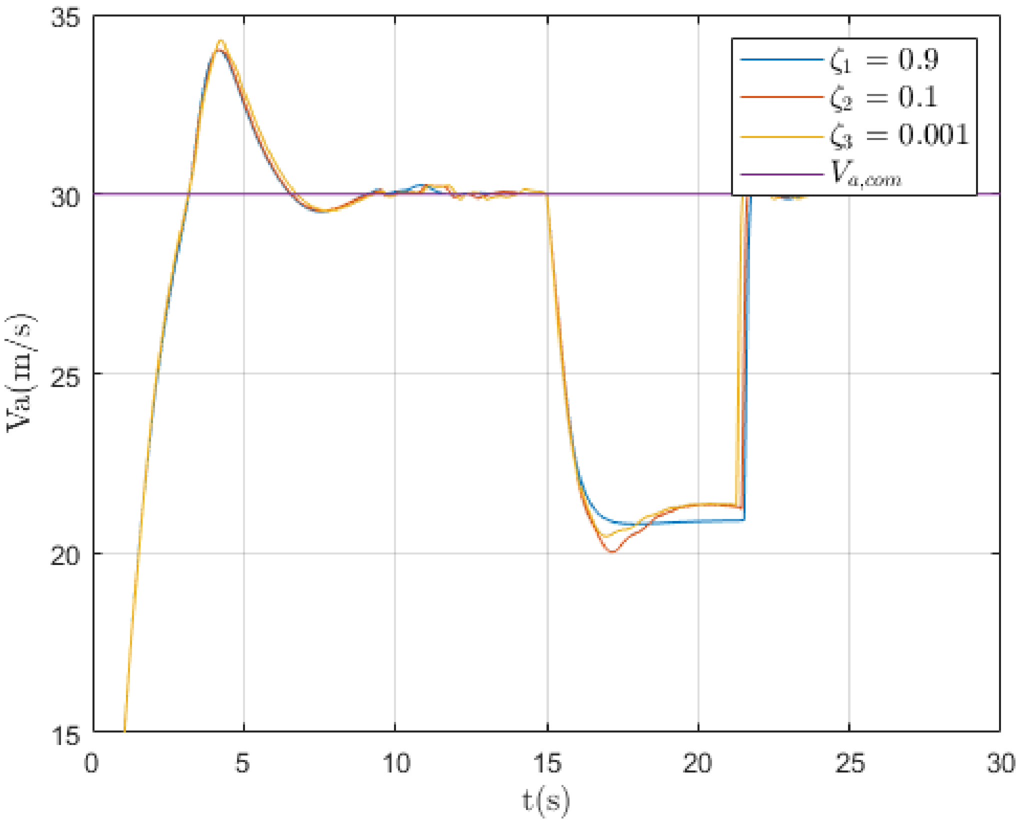

Figura 2.7 Effect of change of ζ on airspeed

Capitolo 3

Exercise 3: Path Planning 3D

The objective of this exercise is to plan a 3D path modifying the Matlab code 'RRTwith-Dubins_UAS2024.m' provided by Prof. G.Fasano. The code is modified in order to:

- change the map: added some buildings and removed other one;
- change the start configuration and goal configuration;
- change an aircraft manuverability parametre: the constant airspeed;
- inject wind;

```
% change map

[xBuilding5,yBuilding5,zBuilding5] = meshgrid(50:100,50:90,0:100);

xyzBuildings2 = [xBuilding5(:) yBuilding5(:) zBuilding5(:)];

obs2 = 0;
updateOccupancy(omap,xyzBuildings2,obs2)


 [xBuilding1,yBuilding1,zBuilding1] = meshgrid(30:50,70:130,70:90);
 [xBuilding2,yBuilding2,zBuilding2] = meshgrid(30:50,60:70,0:100);
 [xBuilding3,yBuilding3,zBuilding3] = meshgrid(30:50,130:140,0:100);
% [xBuilding4,yBuilding4,zBuilding4] = meshgrid(70:80,35:45,0:150);
%
 xyzBuildings = [xBuilding1(:) yBuilding1(:) zBuilding1(:); %%...
                 xBuilding2(:) yBuilding2(:) zBuilding2(:);...
                 xBuilding3(:) yBuilding3(:) zBuilding3(:)];...
%                 xBuilding4(:) yBuilding4(:) zBuilding4(:)];

obs = 0.65;
updateOccupancy(omap,xyzBuildings,obs)
```

Figura 3.1 Change the map

```
startPose = [45 120 95 0];
goalPose = [20 100 60 pi];
figure("Name","StartAndGoal")
hMap = show(omap);
hold on
scatter3(hMap,startPose(1),startPose(2),startPose(3),30,"red","filled")
scatter3(hMap,goalPose(1),goalPose(2),goalPose(3),30,"green","filled")
hold off
view([-31 63])

%pause

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %cambio Va 6 o 9

ss = ExampleHelperUAVStateSpace("MaxRollAngle",pi/6,...    % pi/6
                                "AirSpeed",9,...   %%%%%%  6
                                "FlightPathAngleLimit",[-0.1 0.1],...% 0.1
                                "Bounds",[-20 200; -20 220; 10 200; -pi pi]);
```

Figura 3.2 Change Va and departure and destination

```
function [dydt]=exampleHelperUAVDerivatives(y,wpFollowerObj,LookAheadDist,model,e,PHeadingAngle,
%exampleHelperUAVDerivatives Compute the derivative of states of the controlled UAV
%   Copyright 2019 The MathWorks, Inc.
[lookAheadPoint,desiredHeading] = wpFollowerObj([y(1) ;y(2) ;y(3); y(5)],LookAheadDist);

%NED to NEH frame conversion
desiredHeight=-lookAheadPoint(3);
RollAngle=exampleHelperHeadingControl(y,desiredHeading,e,PHeadingAngle,rollAngleLimit);
% Create control signal
u = control(model);
u.RollAngle=RollAngle;
u.Height=desiredHeight;
u.AirSpeed=airSpeed;

%% Change here
% random wind
e.WindNorth=1.5;
e.WindEast=-2;
e.WindUp=0;

%convert to NEH frame
yNEH=y;
yNEH(3)=-y(3);
dydtNEH = derivative(model,yNEH,u,e);
%convert from NEH to NED frame back
dydt=dydtNEH;
dydt(3)=-dydtNEH(3);
end
```

Figura 3.3 Inject the wind

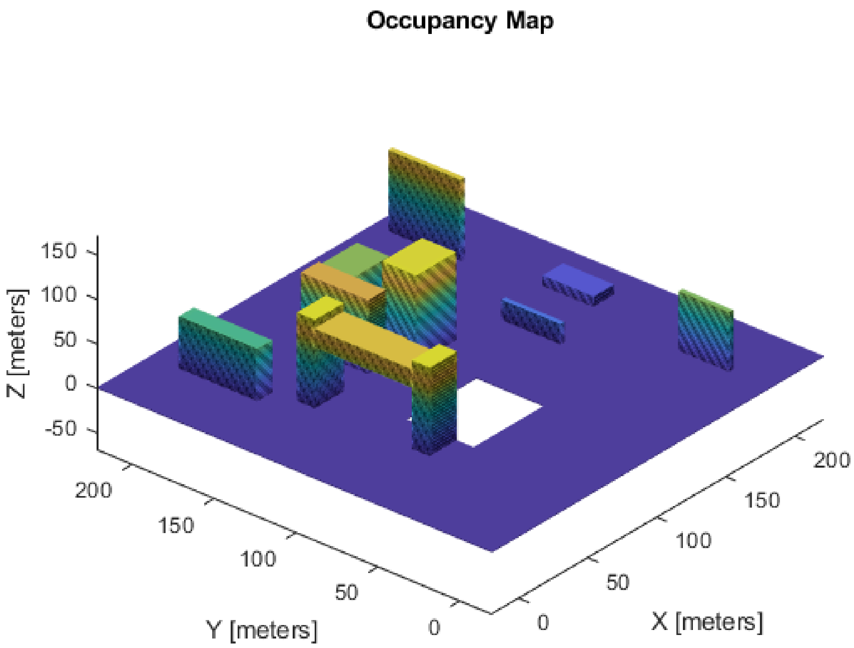

Figura 3.4 Changed map

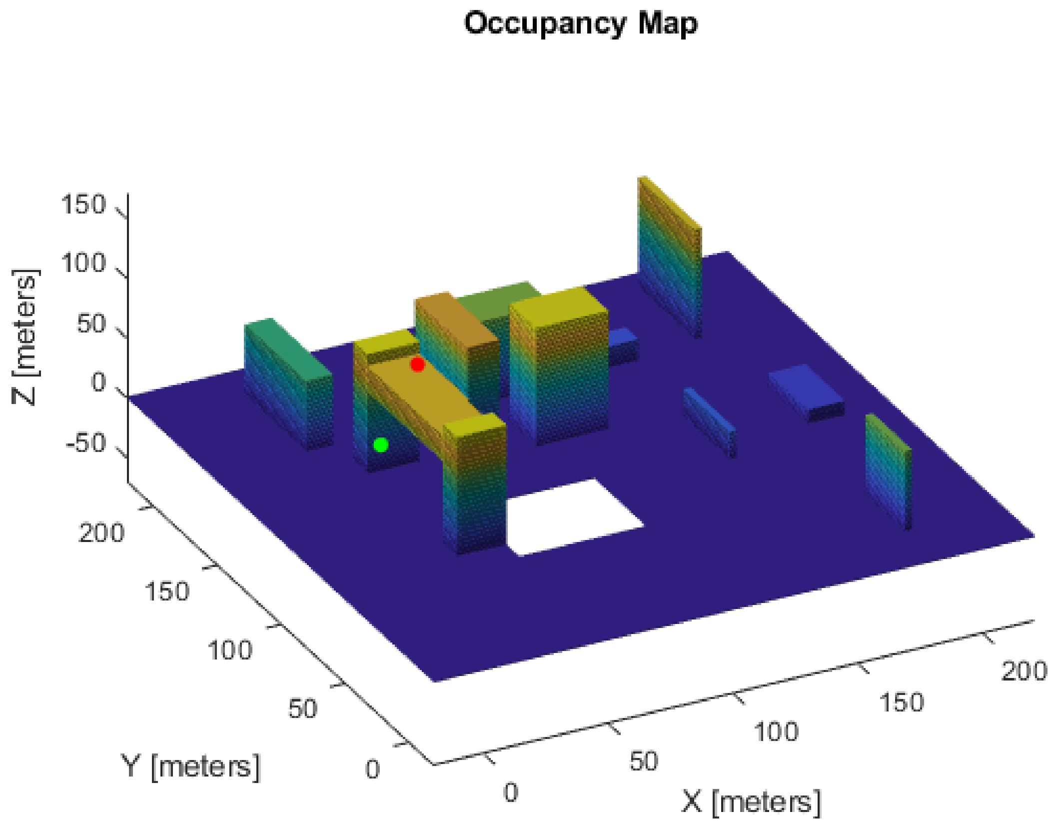

Figura 3.5 Changed start aand goal

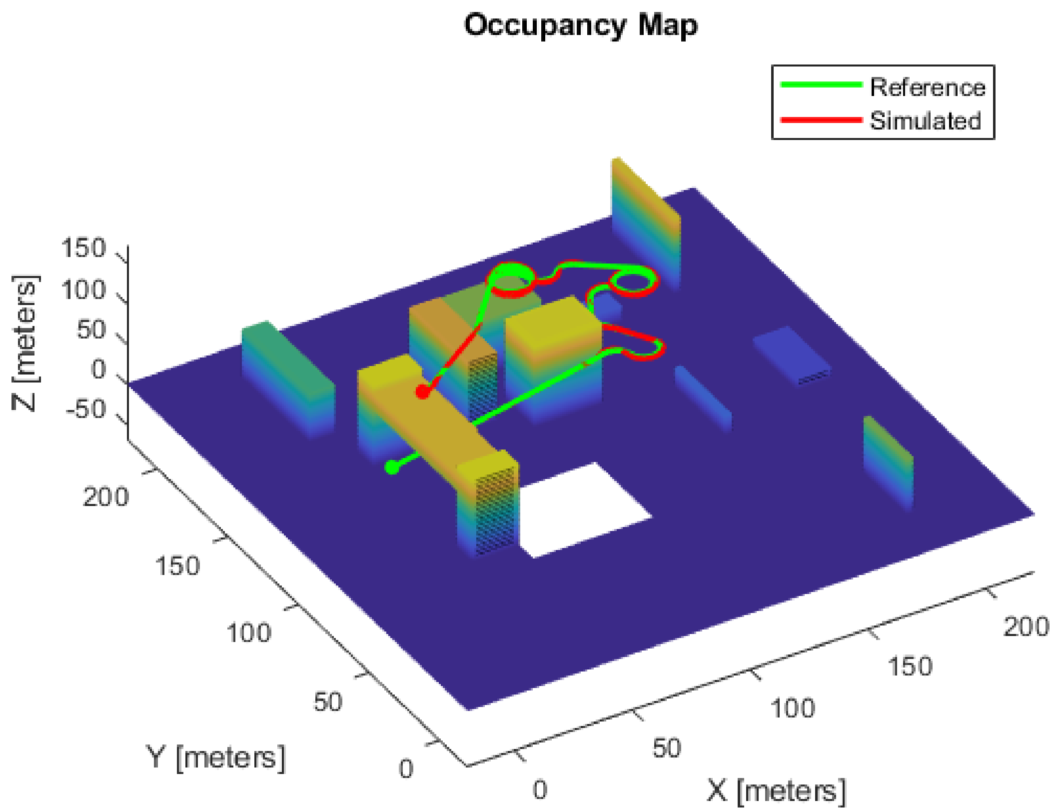

Figura 3.6 Planned and simulated trajectory

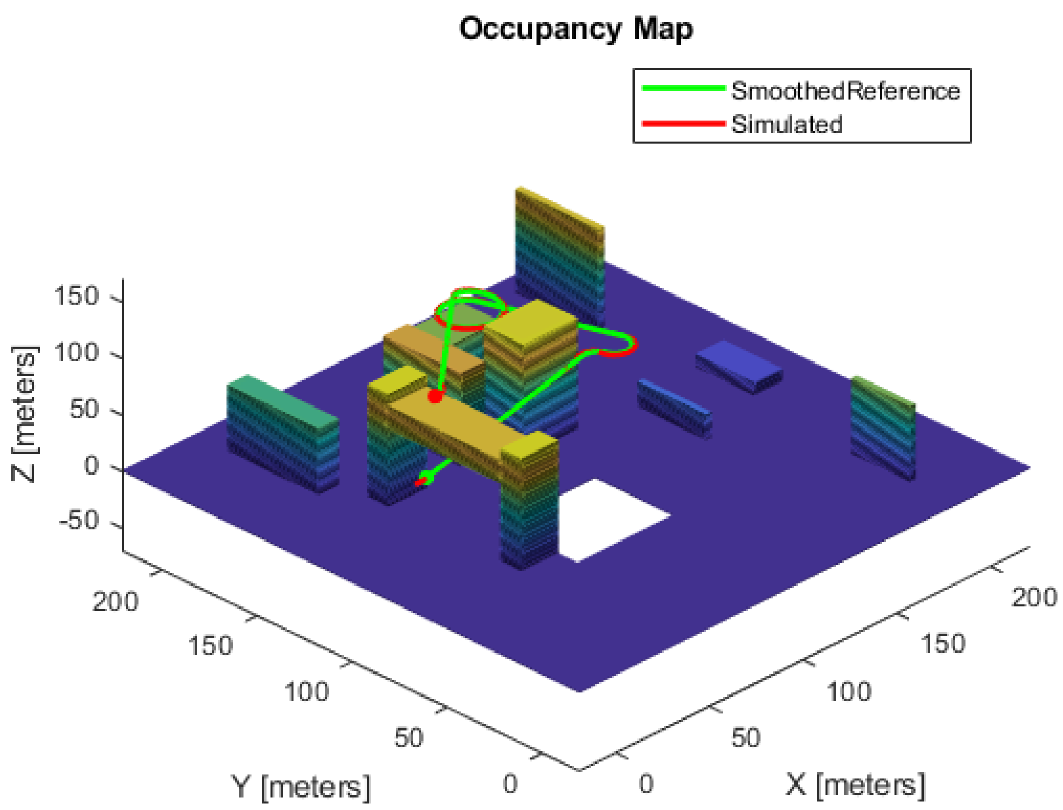

Figura 3.7 Smoothed trajectory: shorter and lower number of unnecessary turns

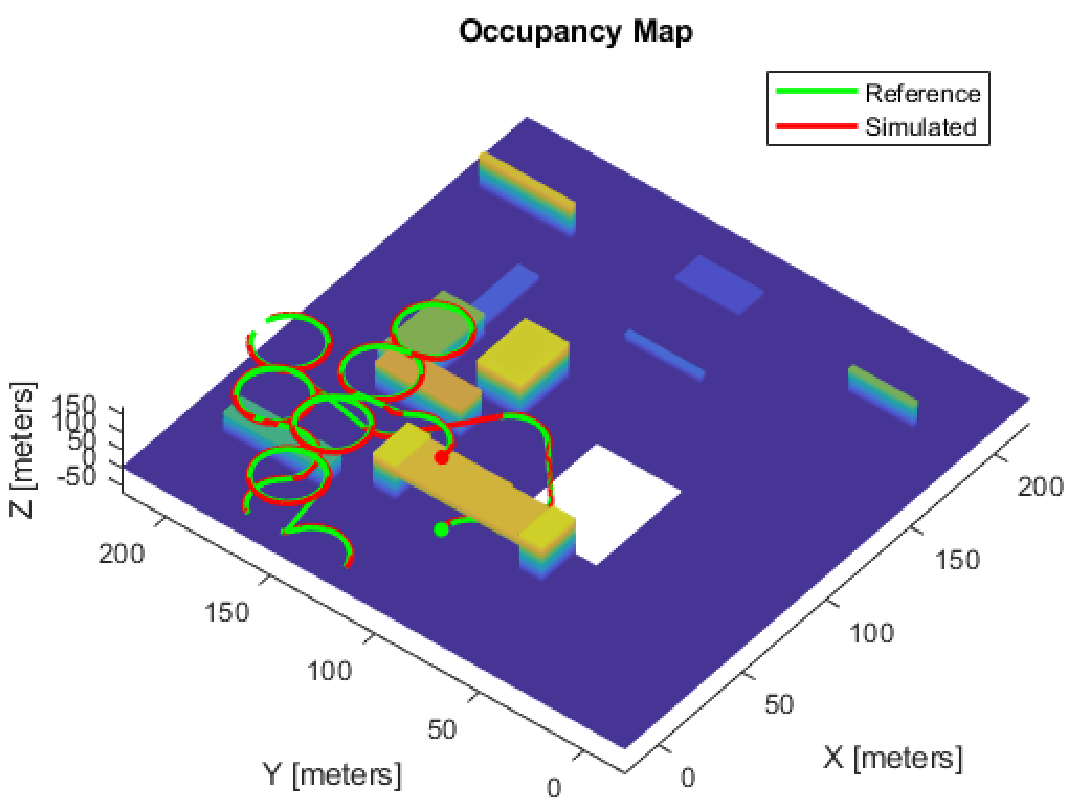

Figura 3.8 Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased

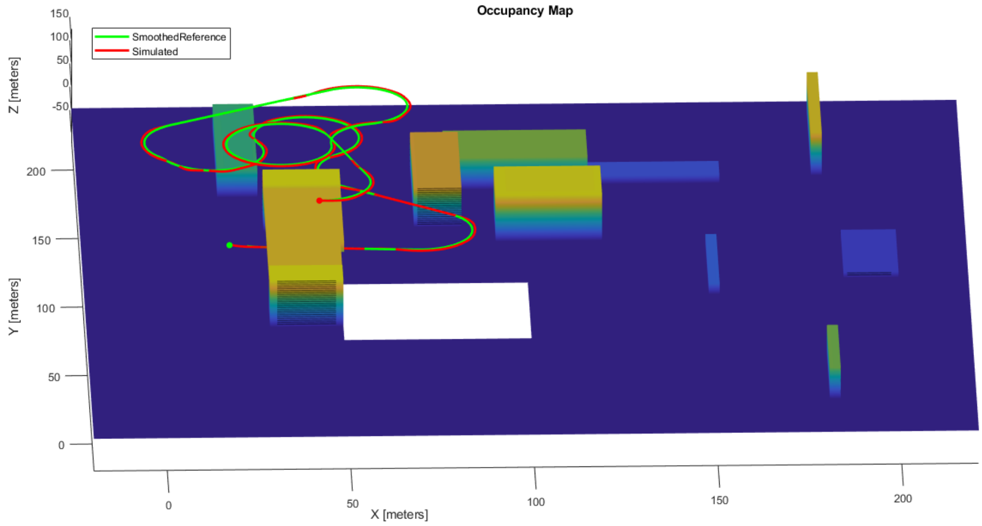

Figura 3.9 Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased (smoothing operation effect

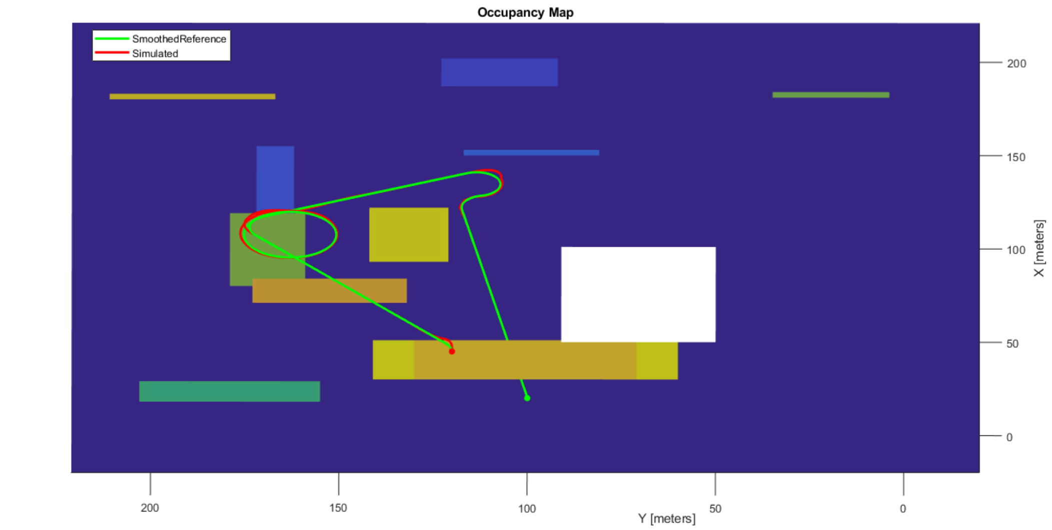

Figura 3.10 Planned and simulated trajectory with wind: the aircraft has some problrm in following the trajectory

Capitolo 4

Exercise 4: Tracking for collision avoidance applications

4.1 Procedure

The objective of this exercise is to develop a tracking Extended Kalman Filtre (EKF) to estimate the target position relative to the UAV (EGO) during the encounter. The state correction in the EKF is performed using the data retrieved by a radar installed on the UAV. In the following Matlab code, this procedure is implemented:

- Import and ispect the simulation data consisting of: true target and UAV position; range, azimuth and elevetion measured by the radar;
- Compute Variance and standard deviation of the Radar errors to quantify the Radar Accuracy: ;
- Tune the EKF: define the state transition matrix, set the Process noice matrix, compute Measurement covariance matrix, initialize State and Covariance matrix.
- develop the EKF: prediction based on a costant velocity model, correction based on radar measurements (P and H computed using 'RadEKFStateCov' and 'RadEKFH' matlab functions provided by Prof. Fasano);
- compare truth and estimated data.

Listing 4.1

```
clc;
clear;
close all;

load('SimulationData_2024_periodicalV_v2.mat')

%% Grafica Dati Truth and Measured
figure(1)
```

```
% Truth
subplot(3,1,1);
plot(Truth.Time,Truth.EgoPos(:,1),Truth.Time,Truth.TargetPos(:,1))
hold on
grid on
axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$x_{East}$(m)','Interpreter','latex','FontSize',12')
lgd = legend('$estimate$','$real$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

subplot(3,1,2);
plot(Truth.Time,Truth.EgoPos(:,2),Truth.Time,Truth.TargetPos(:,2))
hold on
grid on
axis([0 25 30 70])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$x_{North}$(m)','Interpreter','latex','FontSize',12')
lgd = legend('$estimate$','$real$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

subplot(3,1,3);
plot(Truth.Time,Truth.EgoPos(:,3),Truth.Time,Truth.TargetPos(:,3))
hold on
grid on
axis([0 25 20 50])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$x_{up}$(m)','Interpreter','latex','FontSize',12')
lgd = legend('$estimate$','$real$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

%Measures

figure(2)
subplot(3,1,1);
plot(Radar.Time,Radar.Range,'Xb','MarkerFaceColor','auto','MarkerSize',3)
hold on
grid on
%axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Range(m)$','Interpreter','latex','FontSize',12')


subplot(3,1,2);
plot(Radar.Time,Radar.Az,'Xb','MarkerFaceColor','r','MarkerSize',3)
hold on
grid on
%axis([0 25 30 70])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Azimuth(deg)$','Interpreter','latex','FontSize',12')


subplot(3,1,3);
```

```
plot(Radar.Time,Radar.El,'Xb','MarkerFaceColor','auto','MarkerSize',3)
hold on
grid on
%axis([0 25 20 50])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Elevetion(deg)$','Interpreter','latex','FontSize',12')





%% by considering relative position of target with respect to ownship
Delta_x = Truth.TargetPos(:,1) - Truth.EgoPos(:,1);
Delta_y = Truth.TargetPos(:,2) - Truth.EgoPos(:,2);
Delta_z = Truth.TargetPos(:,3) - Truth.EgoPos(:,3);

R_truth = sqrt(Delta_x.^2 + Delta_y.^2 + Delta_z.^2);
Az_truth = atan2(Delta_y, Delta_x);
Az_truth_deg = rad2deg(Az_truth);
El_truth = asin(Delta_z ./ R_truth);
El_truth_deg = rad2deg(El_truth);






%%

% 2) Conversion from SRF to ENU for the radar data.
% 2.1) Computing cartesian measures in SRF
El_radar_SRF = convang(Radar.El, 'deg', 'rad');
Az_radar_SRF = convang(Radar.Az, 'deg', 'rad');
x_SRF = Radar.Range .* cos(El_radar_SRF) .* cos(Az_radar_SRF);
y_SRF = Radar.Range .* cos(El_radar_SRF) .* sin(Az_radar_SRF);
z_SRF = Radar.Range .* sin(El_radar_SRF);

% 2.2) Transforming cartesian measures from SRF to BRF accounting for
    sensor
% mounting orientation and location.
x_BRF = x_SRF + Radar.MountingLocation(1);
y_BRF = y_SRF + Radar.MountingLocation(2);
z_BRF = z_SRF + Radar.MountingLocation(3);

% 2.3) cartesian measure from BRF to ENU
x_ENU = x_BRF;
y_ENU = y_BRF;
z_ENU = z_BRF;

% 2.4) cartesian component to polar ones
R_radar = sqrt(x_ENU.^2 + y_ENU.^2 + z_ENU.^2);
Az_radar = atan2(y_ENU, x_ENU);
Az_radar_deg = rad2deg(Az_radar);
El_radar = asin(z_ENU ./ R_radar);
El_radar_deg = rad2deg(El_radar);
```

```
%%

% Errors
error_R = R_radar - R_truth(1:end-1);

error_AZ = Az_radar - Az_truth(1:end-1);
error_AZ_deg = rad2deg(error_AZ);

error_El = El_radar - El_truth(1:end-1);
error_El_deg = rad2deg(error_El);


%% Plot compairison measure and truth and errors


figure(3)
% Truth
subplot(3,1,1);
plot(Truth.Time,R_truth,'b');
hold on
grid on
plot(Radar.Time,R_radar,'Xr','MarkerFaceColor','auto','MarkerSize',5);
%axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Range$(m)','Interpreter','latex','FontSize',12')
lgd = legend('$Truth$','$Radar$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

subplot(3,1,2);
plot(Truth.Time,Az_truth_deg,'b');
hold on
grid on
plot(Radar.Time,Az_radar_deg,'Xr','MarkerFaceColor','auto','MarkerSize'
    ,5);
%axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Azimuth$(deg)','Interpreter','latex','FontSize',12')
lgd = legend('$Truth$','$Radar$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

subplot(3,1,3);
plot(Truth.Time,El_truth_deg,'b');
hold on
grid on
plot(Radar.Time,El_radar_deg,'Xr','MarkerFaceColor','auto','MarkerSize'
    ,5);
%axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Elevetion$(deg)','Interpreter','latex','FontSize',12')
lgd = legend('$Truth$','$Radar$');
lgd.Interpreter = 'latex';
lgd.FontSize = 11;

%%
```

```

figure(4)
subplot(3,1,1);
plot(Radar.Time,error_R,'Xb','MarkerFaceColor','auto','MarkerSize',3)
hold on
grid on
%axis([0 25 -400 400])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Range error(m)$','Interpreter','latex','FontSize',12')


subplot(3,1,2);
plot(Radar.Time,error_AZ_deg ,'Xb','MarkerFaceColor','r','MarkerSize',3)
hold on
grid on
%axis([0 25 30 70])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Azimuth error(deg)$','Interpreter','latex','FontSize',12')


subplot(3,1,3);
plot(Radar.Time,error_El_deg,'Xb','MarkerFaceColor','auto','MarkerSize'
    ,3)
hold on
grid on
%axis([0 25 20 50])
xlabel('t(s)','Interpreter','latex','FontSize',12);
ylabel('$Elevetion error(deg)$','Interpreter','latex','FontSize',12')


% Variance

error_R_new = error_R(~isnan(error_R))';
var_R = var(error_R_new);
dev_st_R = std(error_R_new);

error_Az_new = error_AZ(~isnan(error_AZ))';
var_Az = var(error_Az_new);
dev_st_Az = std(error_Az_new);

error_El_new = error_El(~isnan(error_El))';
var_El = var(error_El_new);
dev_st_El = std(error_El_new);


%% End accuracy estimation


%% Tracking Problem


% Matrices
% State Transition Matrix

T = 0.01; %s , filter sampling time

```

```
% Costant velocity Model
F = [ 1 T
    0 1];

Zero = zeros(2,2);

Phi = [F Zero Zero
      Zero F Zero
      Zero Zero F];

% Process noise matrix Da capire ????
Qi = [T^3/3 T^2/2
      T^2/2 T];

qx = 1; % scale factor
Qx = qx*Qi;

qy = 1000; % scale factor
Qy = qy*Qi;

qz = 1; % scale factor
Qz = qz*Qi;


Q = [Qx Zero Zero
     Zero Qy Zero
     Zero Zero Qz];


% Measurement covariance matrix
R = [var_R  0     0;
     0    var_Az  0;
     0    0   var_El];


% Ientity matrix
I = eye(6);

% State/Covariance Initializations
% state zero x0 and Covariance zero can be initialized by using first
   radar measure
% initial velocity is assumed zero
sigmas_vel = [0; 0; 0];

idx = find(~isnan(R_radar), 1,'first');
meas = [R_radar(idx); Az_radar(idx); El_radar(idx)];

% Covariance matrix prediction
P = RadEKFStateCov(R, sigmas_vel,meas);

% x=[x, x_dot, y, y_dot, z, z _dot] ^ENU

x0 = zeros(6,1);
x0(1) = R_radar(idx)*cos(Az_radar(idx))*cos(El_radar(idx));
x0(2) = 0;
x0(3) = R_radar(idx)*sin(Az_radar(idx))*cos(El_radar(idx));
```

```
x0(4) = 0;
x0(5) = R_radar(idx)*sin(El_radar(idx));
x0(6) = 0;

% It's x_k^ENU
x_k = x0; % state

z = [R_radar; Az_radar; El_radar]; % measure

%%

P_k= P;
P_plot = nan(6,6,2500);
x_plot = nan(2500,6);
for i=idx+1:2500

% State and prediction
x_kp = Phi*x_k;
P_kp = Phi*P_k*Phi'+Q;


% If there is a maeasurement available
if ~isnan(R_radar(i))

% Jacobian of measurement with respect to state
H = RadEKFH(x_k(1),x_k(3),x_k(5),length(x_k));

% Kalman gain
K_kp = (P_kp*H')*(H*P_kp*H'+ R)^-1;

% Covariance correction
P_kpp = (I-K_kp*H)*P_kp;

% Correct Measurement to do the State correction
z_kp = [R_radar(i); Az_radar(i); El_radar(i)];

z_pred = [sqrt(x_kp(1).^2 + x_kp(3).^2 + x_kp(5).^2); ...
          atan2(x_kp(3),  x_kp(1)) ;...
          asin(x_kp(5)./sqrt(x_kp(1).^2+x_kp(3).^2+x_kp(5).^2))];

% Correct State
x_kpp = x_kp + K_kp*(z_kp-z_pred);
x_k = x_kpp;
P_k = P_kpp;
else
x_k = x_kp;
P_k= P_kp;
end
x_plot(i ,: ) = x_k;
P_plot( :,:, i) = P_k;
end
```

4.2 Results

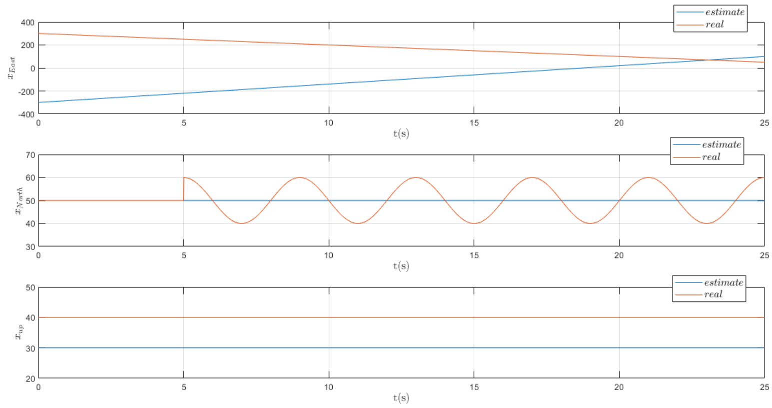

Figura 4.1 True data

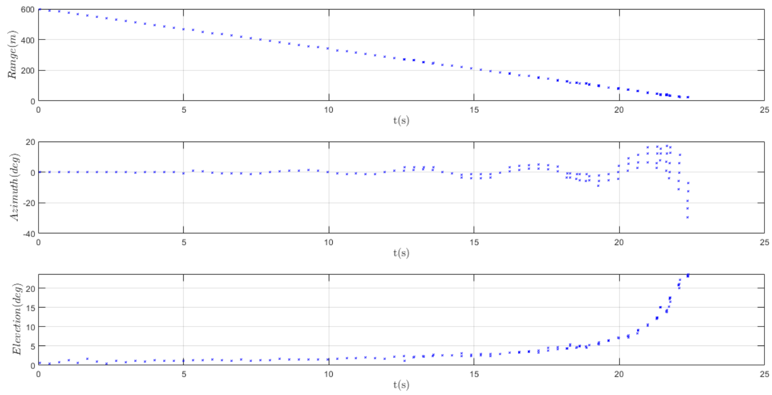

Figura 4.2 Measurements of the radar

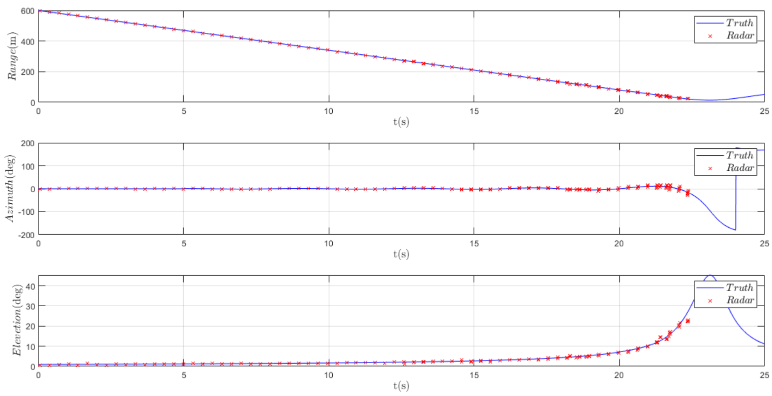

Figura 4.3 Comparison between true range, azimuth and elevation and measured one

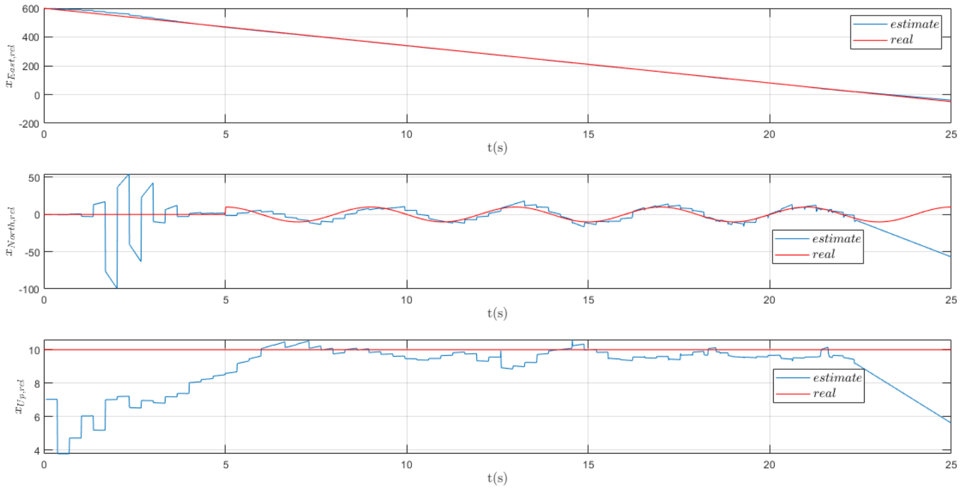

Figura 4.4 comparison true and EKF-estimated data

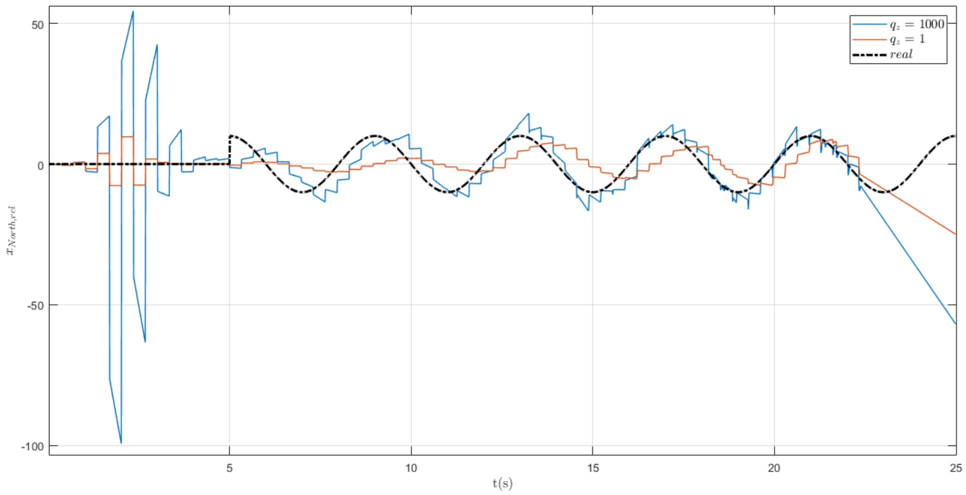

Figura 4.5 comparison true and EKF-estimated X_n for two different value of q_z

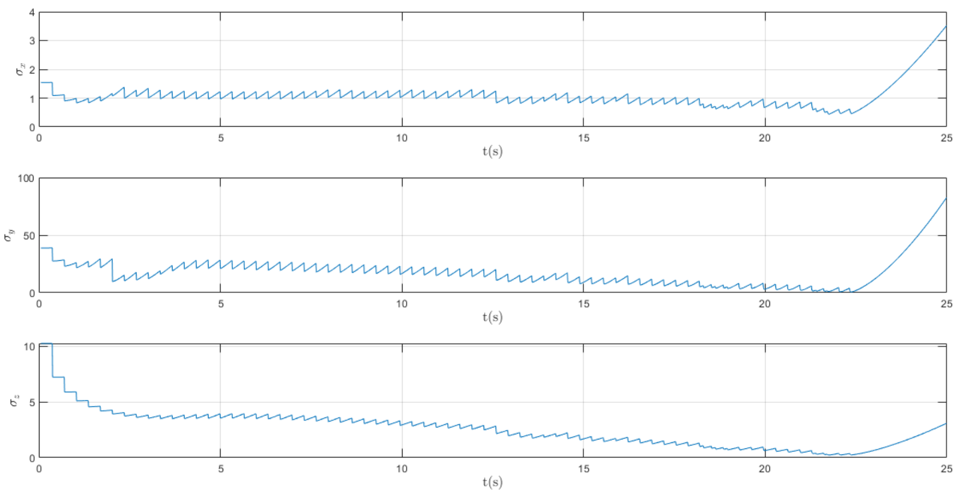

Figura 4.6 standard deviations σ_x, σ_y, σ_z