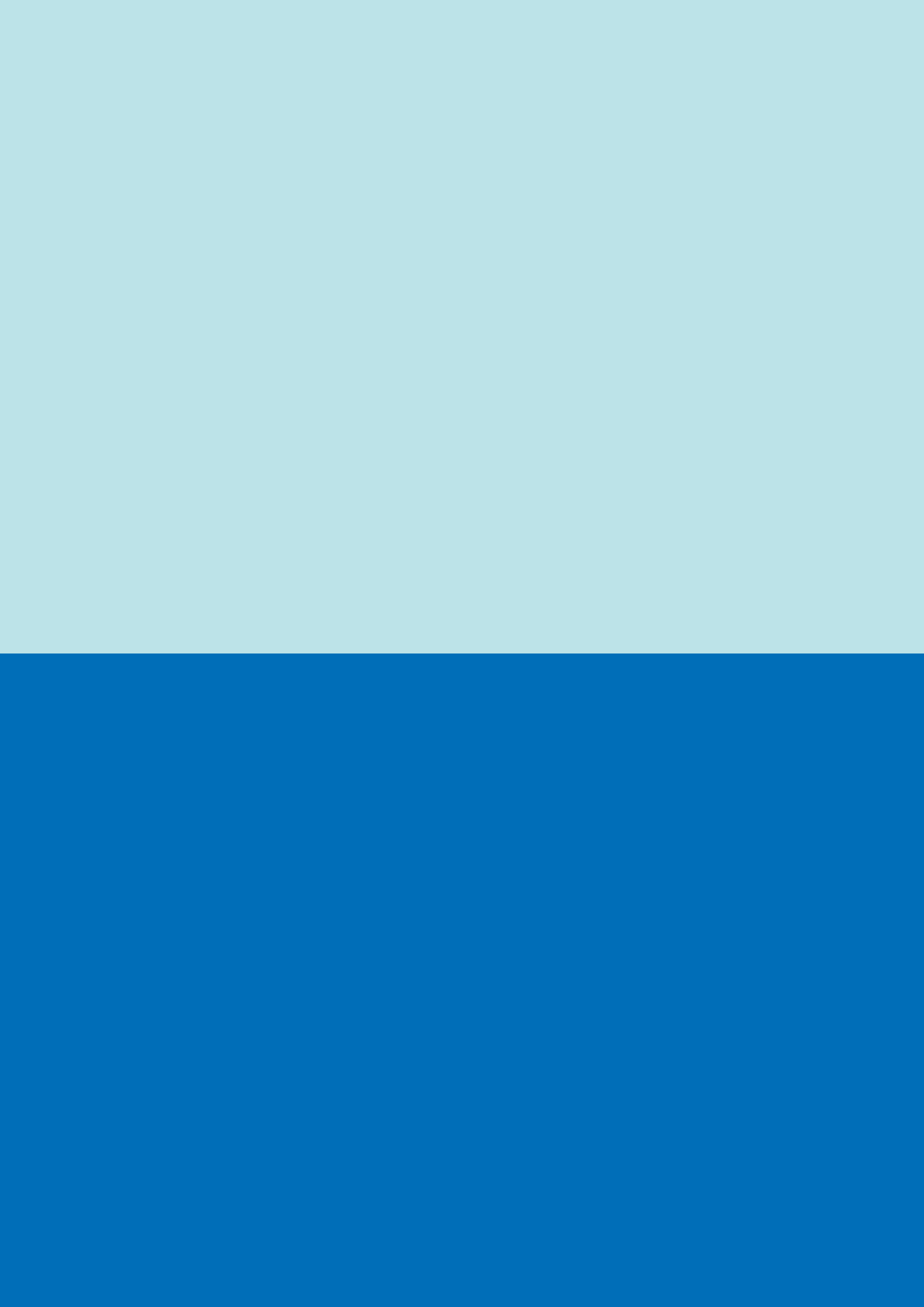


UAS with Matlab/Simulink

**Elaborato di Unmanned
Aircraft Systems**

Antonio Carotenuto



Indice

1	Exercise 1: Pose Estimation	6
1.1	Procedure	6
1.2	Results	10
2	Exercise 2: Autopilot	12
3	Exercise 3: Path Planning 3D	16
4	Exercise 4: Tracking for collision avoidance applications	21
4.1	Procedure	21
4.2	Results	28

Elenco delle figure

1.1	North position	10
1.2	East position	10
1.3	Down position	10
1.4	Heading	11
1.5	Pitch	11
1.6	Roll	11
2.1	Simulink model, some Block parameters: 'Toworkspace' are implemented to save the results	12
2.2	Different values for e_{max} are considered: 0.1, 0.005, 0.5	13
2.3	Different values for ζ are considered: 0.9, 0.1, 0.001	13
2.4	Effect of change of K_p on vertical dinamyc	13
2.5	Effect of change of k_p on airspeed	14
2.6	Effect of change of ζ on vertical dinamyc	14
2.7	Effect of change of ζ on airspeed	15
3.1	Change the map	16
3.2	Change Va and departure and destination	17
3.3	Inject the wind	17
3.4	Changed map	17
3.5	Changed start aand goal	18
3.6	Planned and simulated trajectory	18
3.7	Smoothed trajectory: shorter and lower number of unnecessary turns	19
3.8	Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased	19
3.9	Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased (smoothing operation effect	20
3.10	Planned and simulated trajectory with wind: the aircraft has some problrn in following the trajectory	20
4.1	True data	28
4.2	Measurements of the radar	28
4.3	Comparison between true range, azimuth and elevation and measured one	28
4.4	comparison true and EKF-estimated data	29
4.5	comparison true and EKF-estimated X_n for two different value of q_z	29

4.6	standard deviations $\sigma_x, \sigma_y, \sigma_z$	29
-----	--	----

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 Matlab 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

```
1 clc;
2 clear;
3 close all;
4
5 vid = VideoReader('Agnano_Multiscale_Vertiport.avi');
6 numFrames = vid.NumFrames;
7 for i = 1:numFrames
8     frames = readFrame(vid);
9     if i > 1 % Not write the 1st frame because it is a repetition of the
10         2nd one
11         imwrite(frames, ['ImagesUAS/Agnano_UAS_' int2str(i-1) '.bmp']);
12     end
13 end
14 % Camera Parameters
```

```

15 focalLength = [1109, 1109];
16 principalPoint = [808, 640];
17 imageSize = [1280, 1616];
18
19 intrinsics = cameraIntrinsics(focalLength, principalPoint, imageSize);
20
21 % Initializations
22 estimatePosition = zeros(400, 3);
23 estimateYaw = zeros(400, 1);
24 estimatePitch = zeros(400, 1);
25 estimateRoll = zeros(400, 1);
26
27 %%
28 % Corner position of the Marker in NED
29 Marker_NED36h11 = [
30     -5,  5, -29;
31     5,  5, -29;
32     5, -5, -29;
33    -5, -5, -29
34 ];
35
36 Marker_NEDcircle21h7 = [
37     6.75, -3.85, -29;
38    10.25, -3.85, -29;
39    10.25, -7.35, -29;
40     6.75, -7.35, -29
41 ];
42
43 Marker_NED25h9 = [
44     8,  0, -29;
45    10,  0, -29;
46    10, -2, -29;
47     8, -2, -29
48 ];
49
50
51
52 %%
53 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% From CRF To BODY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
54 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55
56 RcameraToBody = angle2dcm(deg2rad(90), deg2rad(70-90), deg2rad(0));
57
58 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
59 %%
60
61
62
63
64
65 % Import true data and initializations
66 trajectory = load('UAS_trajectory_24_25.mat');
67 xyzNED(:,1) = trajectory.N;
68 xyzNED(:,2) = trajectory.E;
69 xyzNED(:,3) = trajectory.D;
70 realYaw = trajectory.Yaw;

```

```

71 realPitch = trajectory.Pitch;
72 realRoll = trajectory.Roll;
73
74 error = zeros(400, 6);
75 j = 1;
76
77 %%
78 for i = 1:400
79     try
80         image = imread(['ImagesUAS/Agnano_UAS_' int2str(i) '.bmp']);
81
82         tagFamily = ["tag36h11", "tagCircle21h7", "tag25h9"];
83
84         [id_36h11, loc_36h11, detectedFamily_36h11] = readAprilTag(image,
85             tagFamily(1));
86         [id_Circle21h7, loc_Circle21h7, detectedFamily_Circle21h7] =
87             readAprilTag(image, tagFamily(2));
88         [id_25h9, loc_25h9, detectedFamily_25h9] = readAprilTag(image,
89             tagFamily(3));
90
91         if ~isempty(loc_36h11) && ~isempty(loc_Circle21h7) % Se entrambi i tag 36
92             h11 e 25h9 sono presenti
93             [worldOrientation, worldLocation] = estimateWorldCameraPose([
94                 loc_36h11(:,:); loc_Circle21h7(:,:)], ...
95                 [Marker_NED36h11; Marker_NEDcircle21h7], intrinsics);
96             estimatePosition(i, :) = worldLocation;
97
98             %% HERE
99             [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(
100                 RcameraToBody * worldOrientation, 'ZYX', 'Robust');
101
102         elseif ~isempty(loc_36h11) % Se presente solo il tag 36h11
103             [worldOrientation, worldLocation] = estimateWorldCameraPose(loc_36h11
104                 (:,:), Marker_NED36h11, intrinsics);
105             estimatePosition(i, :) = worldLocation;
106
107             %% HERE
108             [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(
109                 RcameraToBody * worldOrientation, 'ZYX', 'Robust');
110
111         elseif ~isempty(loc_Circle21h7) % Se presente solo il tag 25h9
112             [worldOrientation, worldLocation] = estimateWorldCameraPose(
113                 loc_Circle21h7(:,:), Marker_NEDcircle21h7, intrinsics);
114             estimatePosition(i, :) = worldLocation;
115
116             %% HERE
117             [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(
118                 RcameraToBody * worldOrientation, 'ZYX', 'Robust');
119
120         elseif ~isempty(loc_25h9) % Se nessuno dei tag precedenti presente, si

```



```

    usa il tag 16h5
117 [worldOrientation, worldLocation] = estimateWorldCameraPose(loc_25h9
    (:,:), Marker_NED25h9, intrinsics);
118 estimatePosition(i, :) = worldLocation;
119
120 %% HERE
121 [estimateYaw(i), estimatePitch(i), estimateRoll(i)] = dcm2angle(
    RcameraToBody * worldOrientation', 'ZYX', 'Robust');
122
123
124 end
125
126 catch
127     disp(['errore alla ', num2str(i), ' iterazione']);
128 end
129
130 %%
131 if (estimatePosition(i,1) ~= 0) && (estimatePosition(i,2) ~= 0) && (
    estimatePosition(i,3) ~= 0)
132     error(i,1) = xyzNED(i,1) - estimatePosition(i,1);
133     error(i,2) = xyzNED(i,2) - estimatePosition(i,2);
134     error(i,3) = xyzNED(i,3) - estimatePosition(i,3);
135     error(i,4) = realPitch(i) - estimatePitch(i);
136     error(i,5) = realRoll(i) - estimateRoll(i);
137     error(i,6) = realYaw(i) - estimateYaw(i);
138
139     j = j + 1;
140 end
141 end
142 estimatePosition(:,1) = xyzNED(:,1) - error(:,1);
143 estimatePosition(:,2) = xyzNED(:,2) - error(:,2);
144 estimatePosition(:,3) = xyzNED(:,3) - error(:,3);
145
146 estimatePitch = realPitch - error(:,4);
147 estimateRoll = realRoll - error(:,5);
148 estimateYaw = realYaw - error(:,6);

```

1.2 Results

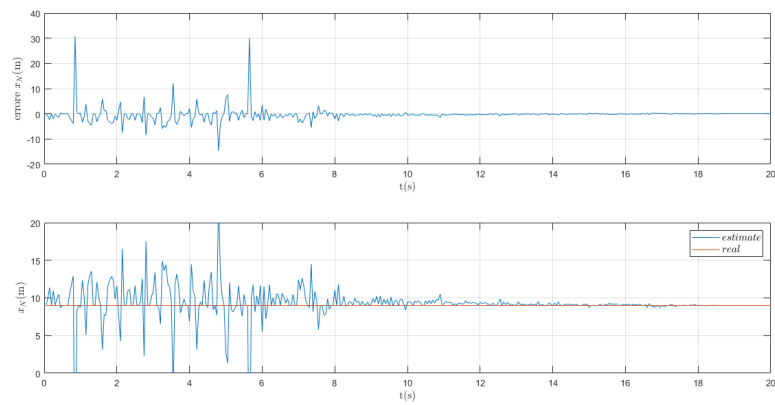


Figure 1.1 North position

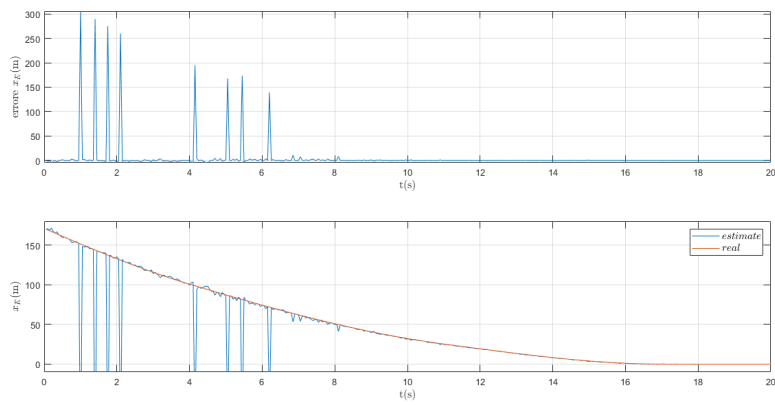


Figure 1.2 East position

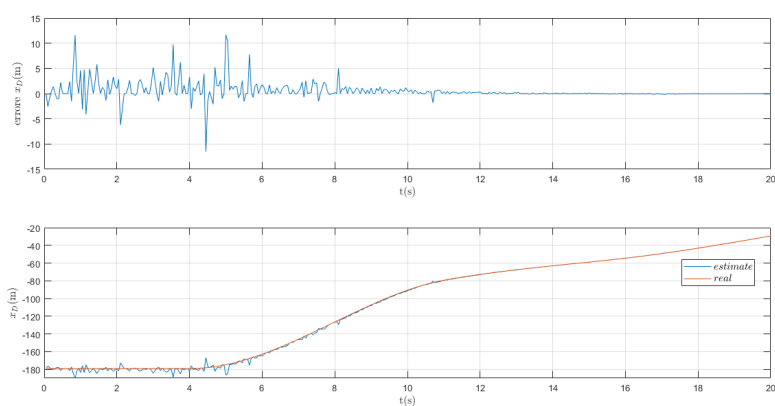


Figure 1.3 Down position

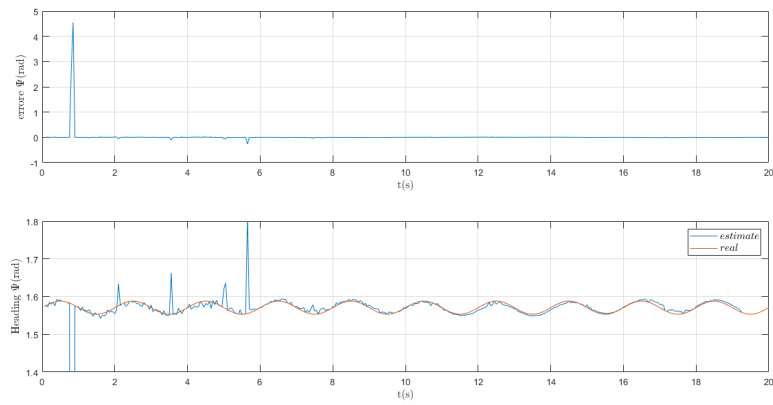


Figura 1.4 Heading

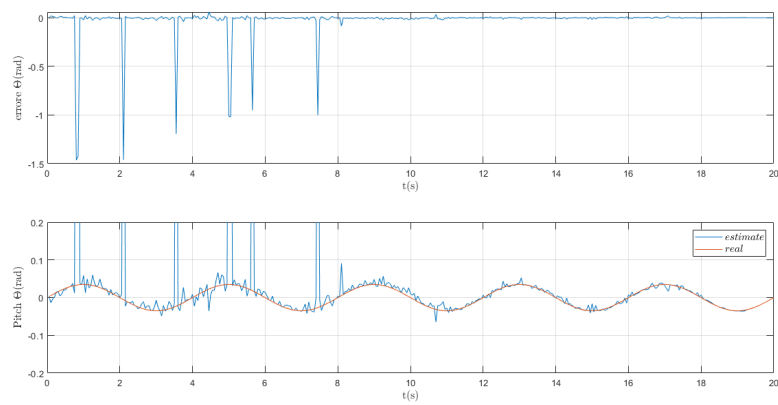


Figura 1.5 Pitch

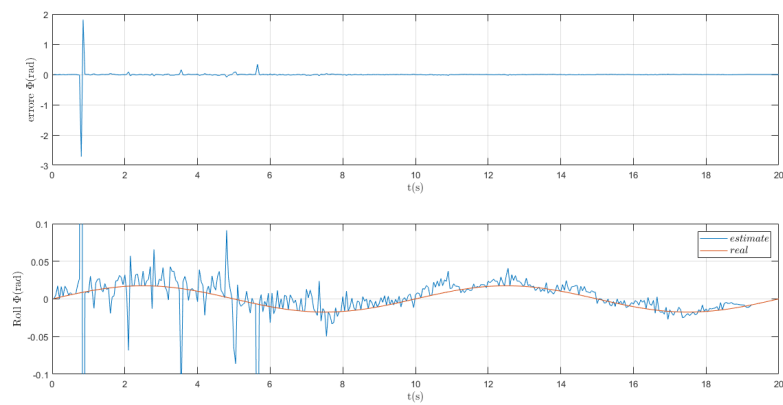


Figura 1.6 Roll

Exercise 2: Autopilot

The objective of this exercise is to change some parameters 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 Damping 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.

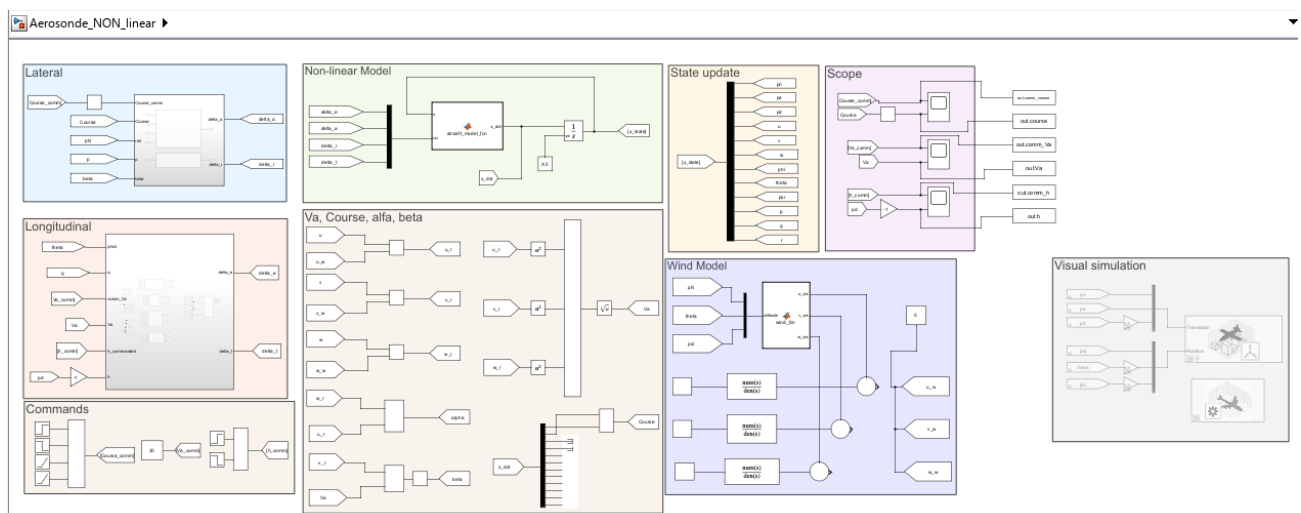


Figura 2.1 Simulink model, some Block parameters: 'Toworkspace' are implemented to save the results

```

%% 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

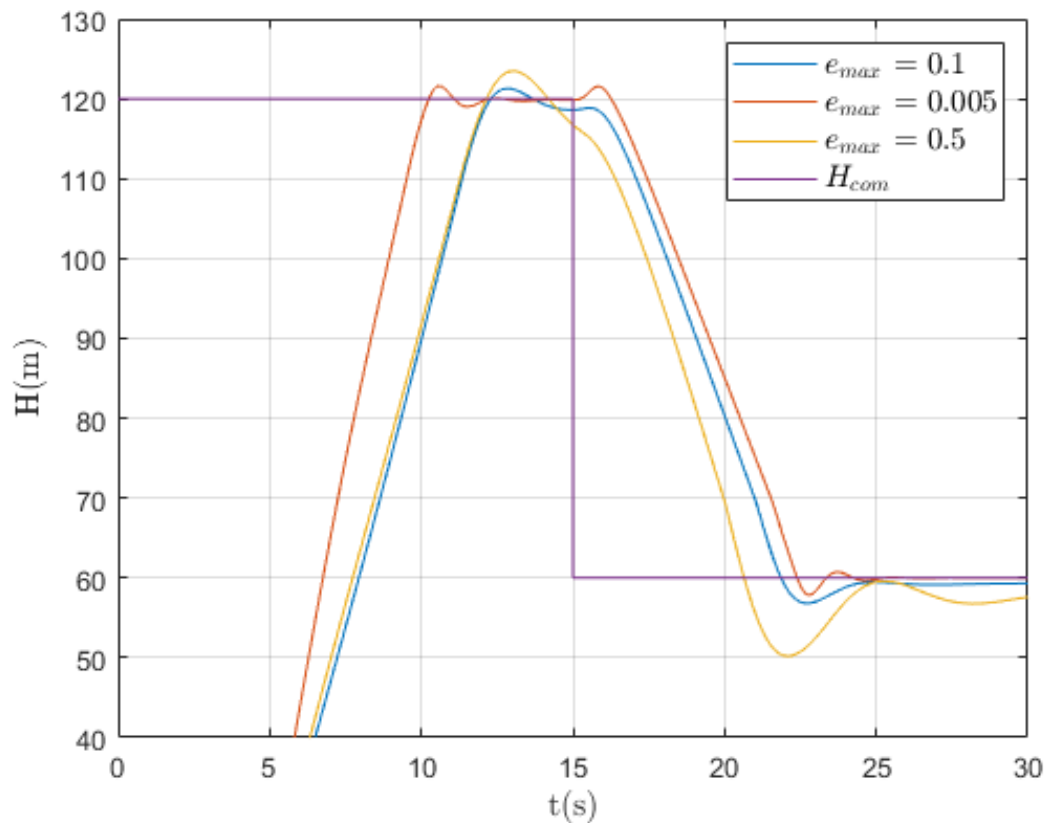


Figura 2.4 Effect of change of K_p on vertical dynamics

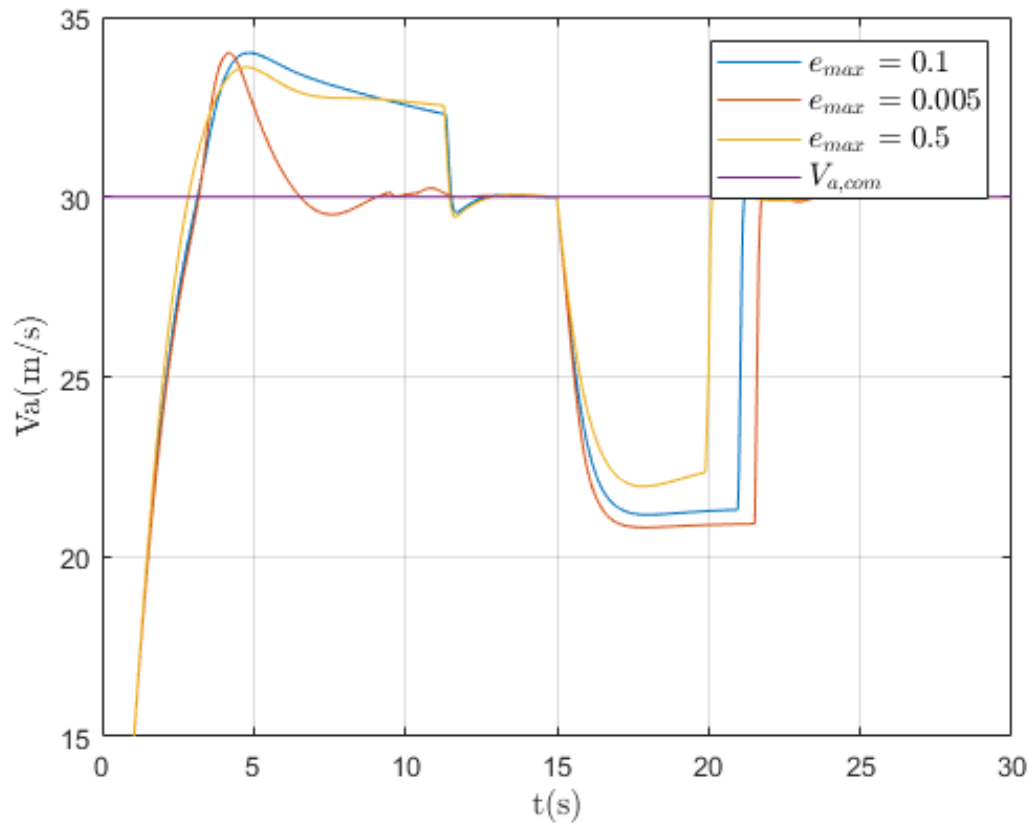


Figure 2.5 Effect of change of k_p on airspeed

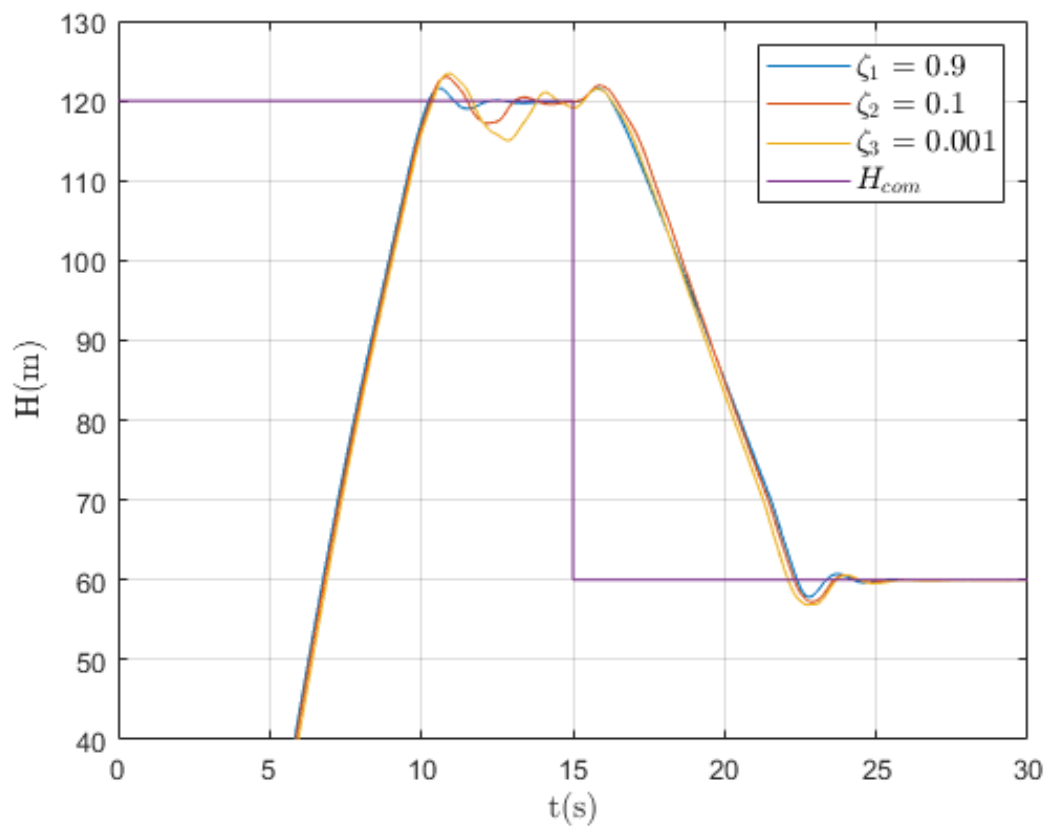


Figure 2.6 Effect of change of ζ on vertical dynamics

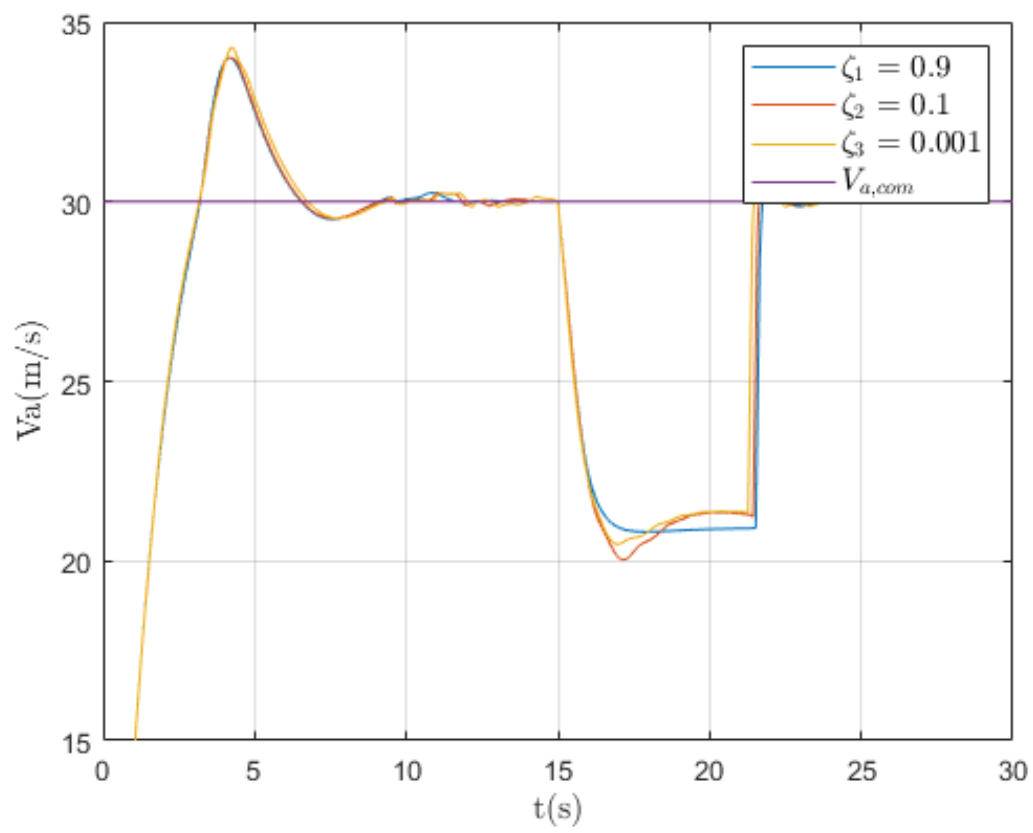


Figura 2.7 Effect of change of ζ on airspeed

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 maneuverability 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

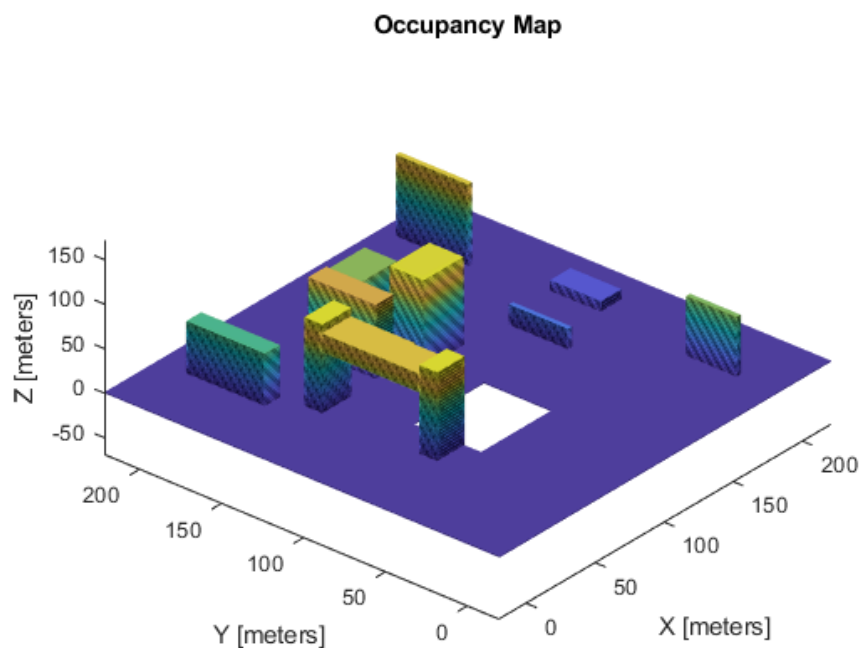


Figura 3.4 Changed map

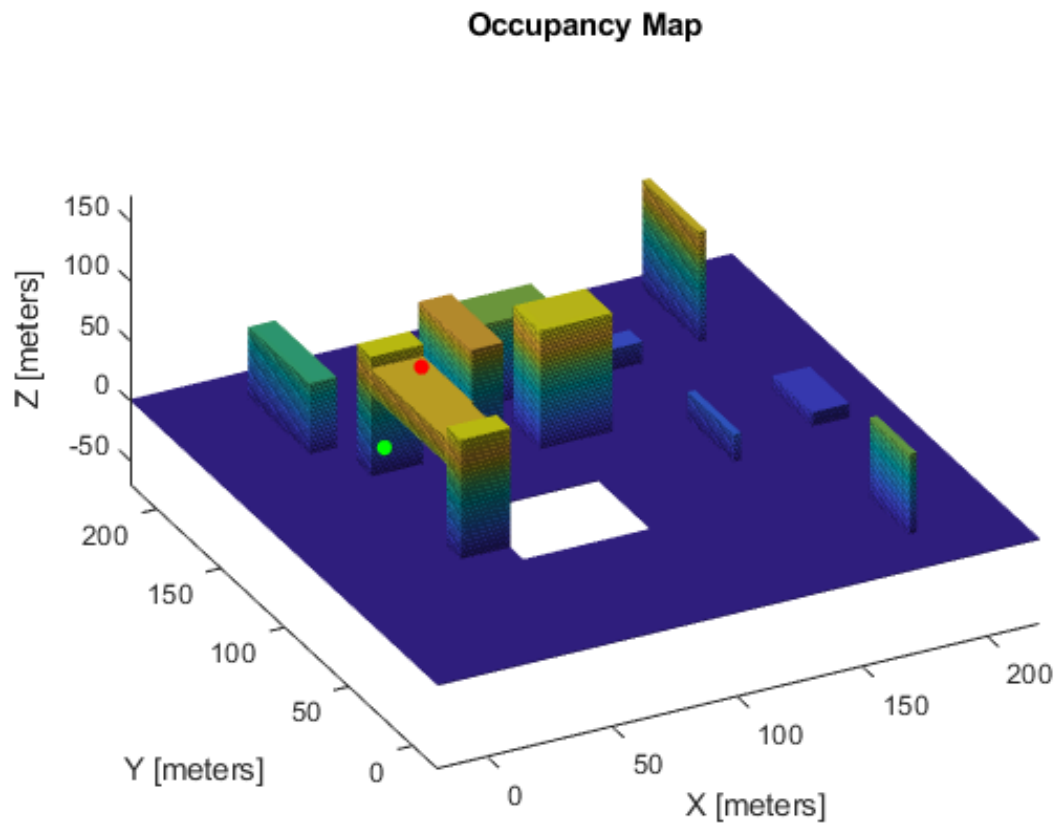


Figure 3.5 Changed start and goal

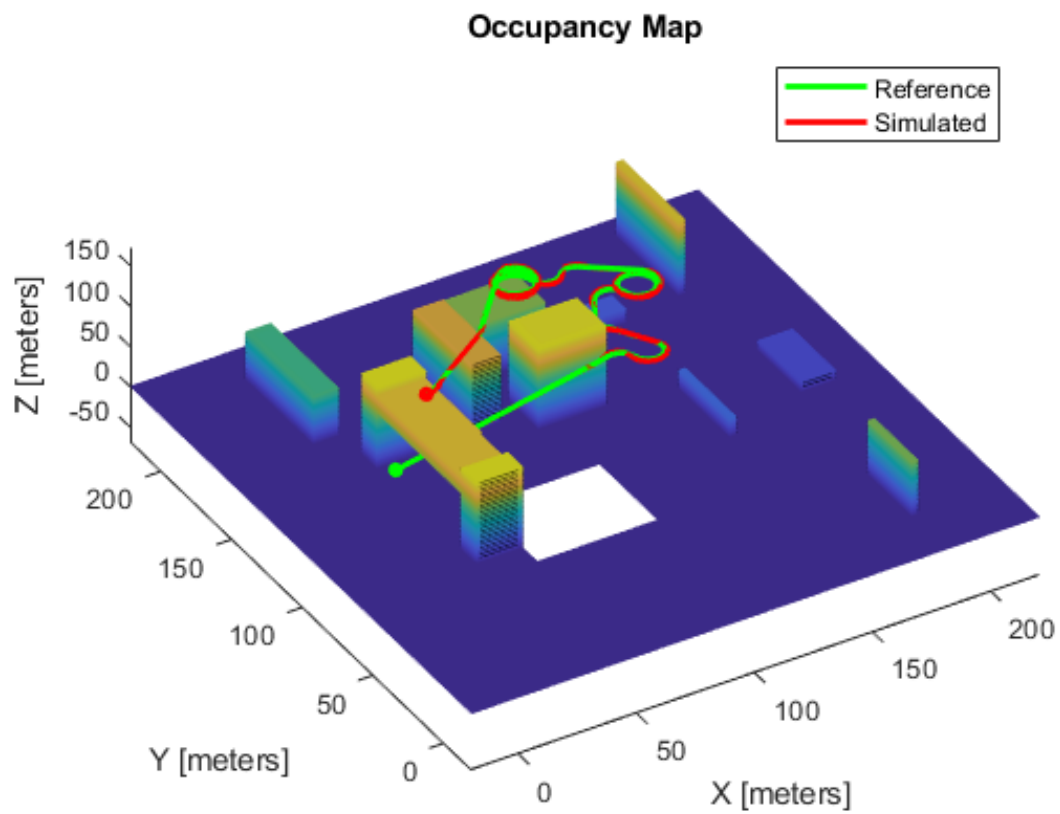


Figure 3.6 Planned and simulated trajectory

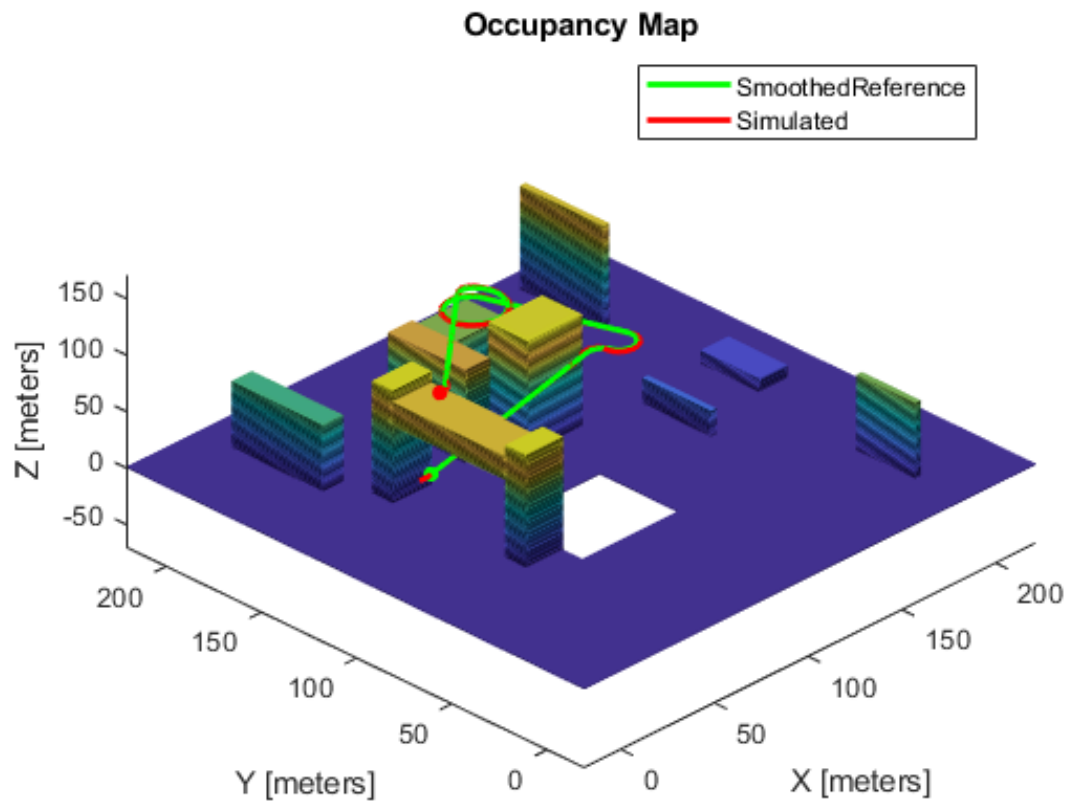


Figure 3.7 Smoothed trajectory: shorter and lower number of unnecessary turns

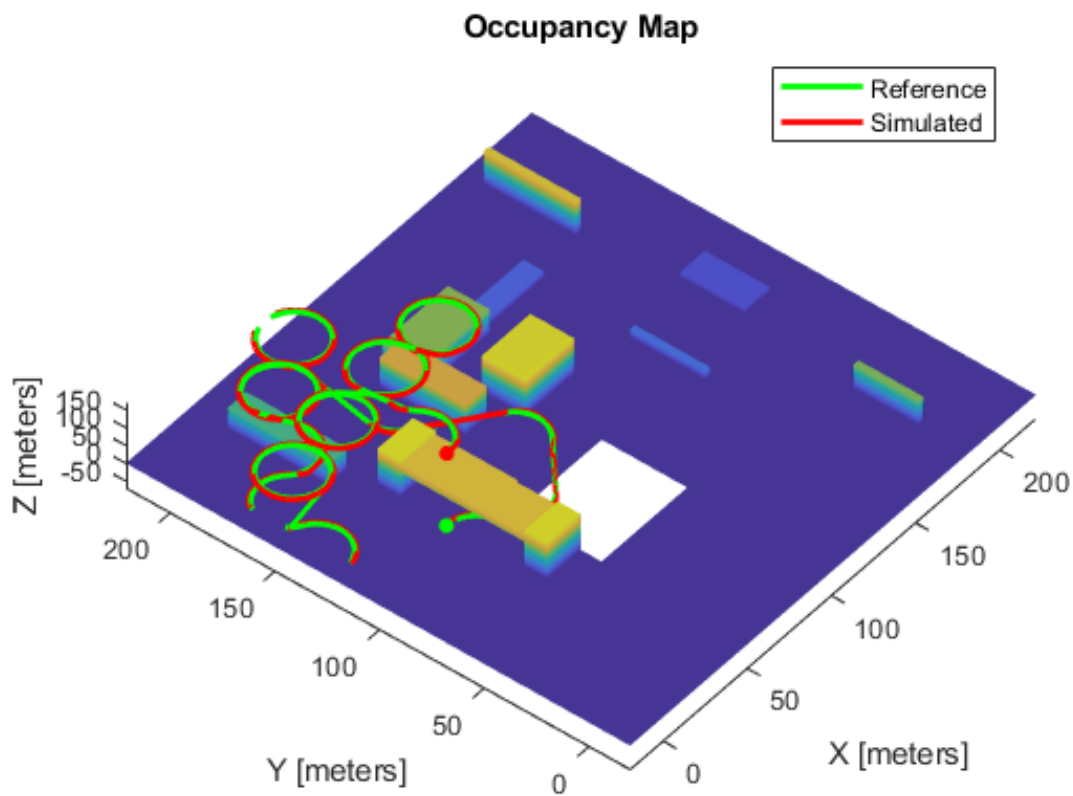


Figure 3.8 Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased

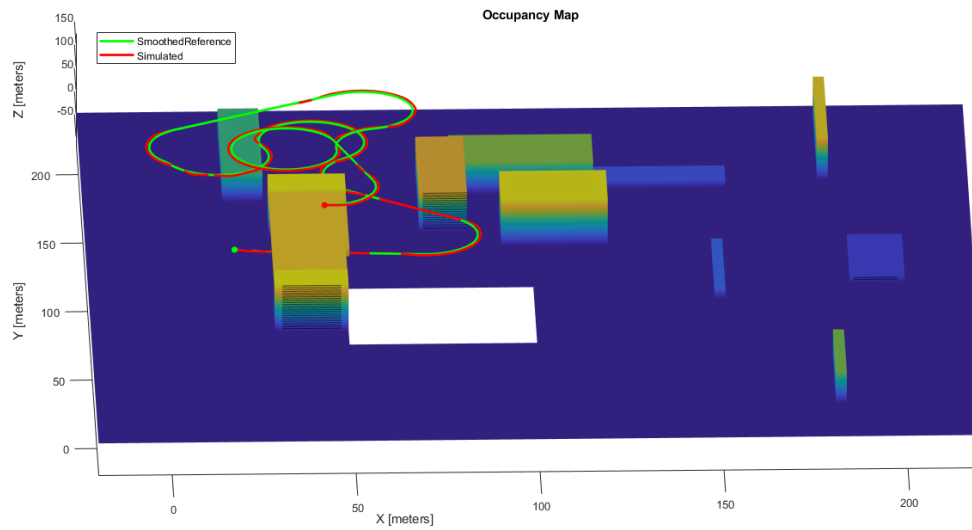


Figura 3.9 Planned and simulated trajectory with higher airspeed: the minimum turn radius has increased (smoothing operation effect)

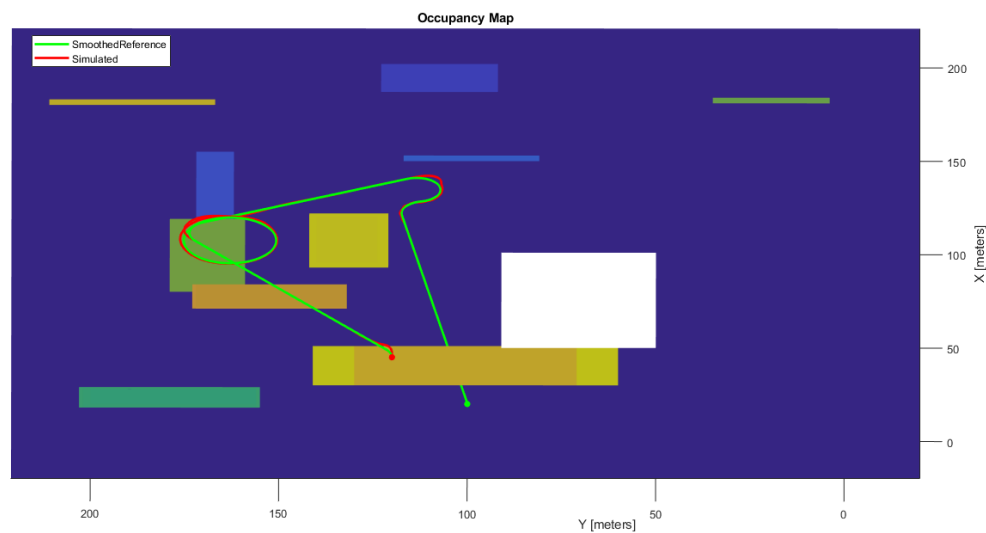


Figura 3.10 Planned and simulated trajectory with wind: the aircraft has some problem in following the trajectory

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

```
1 clc;
2 clear;
3 close all;
4
5 load('SimulationData_2024_periodicalV_v2.mat')
6
7 %% Grafica Dati Truth and Measured
8 figure(1)
```

```

9 % Truth
10 subplot(3,1,1);
11 plot(Truth.Time,Truth.EgoPos(:,1),Truth.Time,Truth.TargetPos(:,1))
12 hold on
13 grid on
14 axis([0 25 -400 400])
15 xlabel('t(s)', 'Interpreter', 'latex', 'FontSize', 12);
16 ylabel('$x_{East}(m)$', 'Interpreter', 'latex', 'FontSize', 12')
17 lgd = legend('$estimate$', '$real$');
18 lgd.Interpreter = 'latex';
19 lgd.FontSize = 11;
20
21 subplot(3,1,2);
22 plot(Truth.Time,Truth.EgoPos(:,2),Truth.Time,Truth.TargetPos(:,2))
23 hold on
24 grid on
25 axis([0 25 30 70])
26 xlabel('t(s)', 'Interpreter', 'latex', 'FontSize', 12);
27 ylabel('$x_{North}(m)$', 'Interpreter', 'latex', 'FontSize', 12')
28 lgd = legend('$estimate$', '$real$');
29 lgd.Interpreter = 'latex';
30 lgd.FontSize = 11;
31
32 subplot(3,1,3);
33 plot(Truth.Time,Truth.EgoPos(:,3),Truth.Time,Truth.TargetPos(:,3))
34 hold on
35 grid on
36 axis([0 25 20 50])
37 xlabel('t(s)', 'Interpreter', 'latex', 'FontSize', 12);
38 ylabel('$x_{up}(m)$', 'Interpreter', 'latex', 'FontSize', 12')
39 lgd = legend('$estimate$', '$real$');
40 lgd.Interpreter = 'latex';
41 lgd.FontSize = 11;
42
43 %Measures
44
45 figure(2)
46 subplot(3,1,1);
47 plot(Radar.Time,Radar.Range, 'Xb', 'MarkerFaceColor', 'auto', 'MarkerSize', 3)
48 hold on
49 grid on
50 %axis([0 25 -400 400])
51 xlabel('t(s)', 'Interpreter', 'latex', 'FontSize', 12);
52 ylabel('$Range(m)$', 'Interpreter', 'latex', 'FontSize', 12')
53
54
55 subplot(3,1,2);
56 plot(Radar.Time,Radar.Az, 'Xb', 'MarkerFaceColor', 'r', 'MarkerSize', 3)
57 hold on
58 grid on
59 %axis([0 25 30 70])
60 xlabel('t(s)', 'Interpreter', 'latex', 'FontSize', 12);
61 ylabel('$Azimuth(deg)$', 'Interpreter', 'latex', 'FontSize', 12')
62
63
64 subplot(3,1,3);

```

```

65 plot(Radar.Time,Radar.El,'Xb','MarkerFaceColor','auto','MarkerSize',3)
66 hold on
67 grid on
68 %axis([0 25 20 50])
69 xlabel('t(s)','Interpreter','latex','FontSize',12);
70 ylabel('$Elevation(deg)$','Interpreter','latex','FontSize',12')
71
72
73
74
75
76 %% by considering relative position of target with respect to ownship
77 Delta_x = Truth.TargetPos(:,1) - Truth.EgoPos(:,1);
78 Delta_y = Truth.TargetPos(:,2) - Truth.EgoPos(:,2);
79 Delta_z = Truth.TargetPos(:,3) - Truth.EgoPos(:,3);
80
81 R_truth = sqrt(Delta_x.^2 + Delta_y.^2 + Delta_z.^2);
82 Az_truth = atan2(Delta_y, Delta_x);
83 Az_truth_deg = rad2deg(Az_truth);
84 El_truth = asin(Delta_z ./ R_truth);
85 El_truth_deg = rad2deg(El_truth);
86
87
88
89
90
91
92 %%
93
94 % 2) Conversion from SRF to ENU for the radar data.
95 % 2.1) Computing cartesian measures in SRF
96 El_radar_SRF = convang(Radar.El, 'deg', 'rad');
97 Az_radar_SRF = convang(Radar.Az, 'deg', 'rad');
98 x_SRF = Radar.Range .* cos(El_radar_SRF) .* cos(Az_radar_SRF);
99 y_SRF = Radar.Range .* cos(El_radar_SRF) .* sin(Az_radar_SRF);
100 z_SRF = Radar.Range .* sin(El_radar_SRF);
101
102 % 2.2) Transforming cartesian measures from SRF to BRF accounting for
    sensor
103 % mounting orientation and location.
104 x_BRF = x_SRF + Radar.MountingLocation(1);
105 y_BRF = y_SRF + Radar.MountingLocation(2);
106 z_BRF = z_SRF + Radar.MountingLocation(3);
107
108 % 2.3) cartesian measure from BRF to ENU
109 x_ENU = x_BRF;
110 y_ENU = y_BRF;
111 z_ENU = z_BRF;
112
113 % 2.4) cartesian component to polar ones
114 R_radar = sqrt(x_ENU.^2 + y_ENU.^2 + z_ENU.^2);
115 Az_radar = atan2(y_ENU, x_ENU);
116 Az_radar_deg = rad2deg(Az_radar);
117 El_radar = asin(z_ENU ./ R_radar);
118 El_radar_deg = rad2deg(El_radar);
119

```

```

120 %%
121
122 % Errors
123 error_R = R_radar - R_truth(1:end-1);
124
125 error_AZ = Az_radar - Az_truth(1:end-1);
126 error_AZ_deg = rad2deg(error_AZ);
127
128 error_El = El_radar - El_truth(1:end-1);
129 error_El_deg = rad2deg(error_El);
130
131
132 %% Plot compairison measure and truth and errors
133
134
135 figure(3)
136 % Truth
137 subplot(3,1,1);
138 plot(Truth.Time,R_truth,'b');
139 hold on
140 grid on
141 plot(Radar.Time,R_radar,'Xr','MarkerFaceColor','auto','MarkerSize',5);
142 %axis([0 25 -400 400])
143 xlabel('t(s)','Interpreter','latex','FontSize',12);
144 ylabel('$Range$(m)','Interpreter','latex','FontSize',12')
145 lgd = legend('$Truth$','$Radar$');
146 lgd.Interpreter = 'latex';
147 lgd.FontSize = 11;
148
149 subplot(3,1,2);
150 plot(Truth.Time,Az_truth_deg,'b');
151 hold on
152 grid on
153 plot(Radar.Time,Az_radar_deg,'Xr','MarkerFaceColor','auto','MarkerSize'
    ,5);
154 %axis([0 25 -400 400])
155 xlabel('t(s)','Interpreter','latex','FontSize',12);
156 ylabel('$Azimuth$(deg)','Interpreter','latex','FontSize',12')
157 lgd = legend('$Truth$','$Radar$');
158 lgd.Interpreter = 'latex';
159 lgd.FontSize = 11;
160
161 subplot(3,1,3);
162 plot(Truth.Time,El_truth_deg,'b');
163 hold on
164 grid on
165 plot(Radar.Time,El_radar_deg,'Xr','MarkerFaceColor','auto','MarkerSize'
    ,5);
166 %axis([0 25 -400 400])
167 xlabel('t(s)','Interpreter','latex','FontSize',12);
168 ylabel('$Elevetion$(deg)','Interpreter','latex','FontSize',12')
169 lgd = legend('$Truth$','$Radar$');
170 lgd.Interpreter = 'latex';
171 lgd.FontSize = 11;
172
173 %%

```



```

174 figure(4)
175 subplot(3,1,1);
176 plot(Radar.Time,error_R,'Xb','MarkerFaceColor','auto','MarkerSize',3)
177 hold on
178 grid on
179 %axis([0 25 -400 400])
180 xlabel('t(s)','Interpreter','latex','FontSize',12);
181 ylabel('$Range\ error(m)$','Interpreter','latex','FontSize',12')
182
183
184
185 subplot(3,1,2);
186 plot(Radar.Time,error_AZ_deg,'Xb','MarkerFaceColor','r','MarkerSize',3)
187 hold on
188 grid on
189 %axis([0 25 30 70])
190 xlabel('t(s)','Interpreter','latex','FontSize',12);
191 ylabel('$Azimuth\ error(deg)$','Interpreter','latex','FontSize',12')
192
193
194 subplot(3,1,3);
195 plot(Radar.Time,error_El_deg,'Xb','MarkerFaceColor','auto','MarkerSize'
    ,3)
196 hold on
197 grid on
198 %axis([0 25 20 50])
199 xlabel('t(s)','Interpreter','latex','FontSize',12);
200 ylabel('$Elevation\ error(deg)$','Interpreter','latex','FontSize',12')
201
202
203 % Variance
204
205 error_R_new = error_R(~isnan(error_R))';
206 var_R = var(error_R_new);
207 dev_st_R = std(error_R_new);
208
209 error_Az_new = error_AZ(~isnan(error_AZ))';
210 var_Az = var(error_Az_new);
211 dev_st_Az = std(error_Az_new);
212
213 error_El_new = error_El(~isnan(error_El))';
214 var_El = var(error_El_new);
215 dev_st_El = std(error_El_new);
216
217
218 %% End accuracy estimation
219
220
221 %% Tracking Problem
222
223
224 % Matrices
225 % State Transition Matrix
226
227 T = 0.01; %s , filter sampling time
228

```

```

229 % Costant velocity Model
230 F = [ 1 T
231       0 1];
232
233 Zero = zeros(2,2);
234
235 Phi = [F Zero Zero
236         Zero F Zero
237         Zero Zero F];
238
239 % Process noise matrix Da capire ???
240 Qi = [T^3/3 T^2/2
241        T^2/2 T];
242
243 qx = 1; % scale factor
244 Qx = qx*Qi;
245
246 qy = 1000; % scale factor
247 Qy = qy*Qi;
248
249 qz = 1; % scale factor
250 Qz = qz*Qi;
251
252
253 Q = [Qx Zero Zero
254       Zero Qy Zero
255       Zero Zero Qz];
256
257
258 % Measurement covariance matrix
259 R = [var_R  0      0;
260       0      var_Az 0;
261       0      0      var_El];
262
263
264 % Identity matrix
265 I = eye(6);
266
267 % State/Covariance Initializations
268 % state zero x0 and Covariance zero can be initialized by using first
    radar measure
269 % initial velocity is assumed zero
270 sigmas_vel = [0; 0; 0];
271
272 idx = find(~isnan(R_radar), 1, 'first');
273 meas = [R_radar(idx); Az_radar(idx); El_radar(idx)];
274
275 % Covariance matrix prediction
276 P = RadEKFStateCov(R, sigmas_vel, meas);
277
278 % x=[x, x_dot, y, y_dot, z, z_dot] ^ENU
279
280 x0 = zeros(6,1);
281 x0(1) = R_radar(idx)*cos(Az_radar(idx))*cos(El_radar(idx));
282 x0(2) = 0;
283 x0(3) = R_radar(idx)*sin(Az_radar(idx))*cos(El_radar(idx));

```

```

284 x0(4) = 0;
285 x0(5) = R_radar(idx)*sin(El_radar(idx));
286 x0(6) = 0;
287
288 % It's x_k^ENU
289 x_k = x0; % state
290
291 z = [R_radar; Az_radar; El_radar]; % measure
292
293 %%
294
295 P_k= P;
296 P_plot = nan(6,6,2500);
297 x_plot = nan(2500,6);
298 for i=idx+1:2500
299
300 % State and prediction
301 x_kp = Phi*x_k;
302 P_kp = Phi*P_k*Phi'+Q;
303
304
305 % If there is a measurement available
306 if ~isnan(R_radar(i))
307
308 % Jacobian of measurement with respect to state
309 H = RadEKFH(x_k(1),x_k(3),x_k(5),length(x_k));
310
311 % Kalman gain
312 K_kp = (P_kp*H')*(H*P_kp*H' + R)^-1;
313
314 % Covariance correction
315 P_kpp = (I-K_kp*H)*P_kp;
316
317 % Correct Measurement to do the State correction
318 z_kp = [R_radar(i); Az_radar(i); El_radar(i)];
319
320 z_pred = [sqrt(x_kp(1).^2 + x_kp(3).^2 + x_kp(5).^2); ...
321           atan2(x_kp(3), x_kp(1)) ;...
322           asin(x_kp(5)./sqrt(x_kp(1).^2+x_kp(3).^2+x_kp(5).^2))];
323
324 % Correct State
325 x_kpp = x_kp + K_kp*(z_kp-z_pred);
326 x_k = x_kpp;
327 P_k = P_kpp;
328 else
329 x_k = x_kp;
330 P_k= P_kp;
331 end
332 x_plot(i ,: ) = x_k;
333 P_plot( :, :, i) = P_k;
334 end

```

4.2 Results

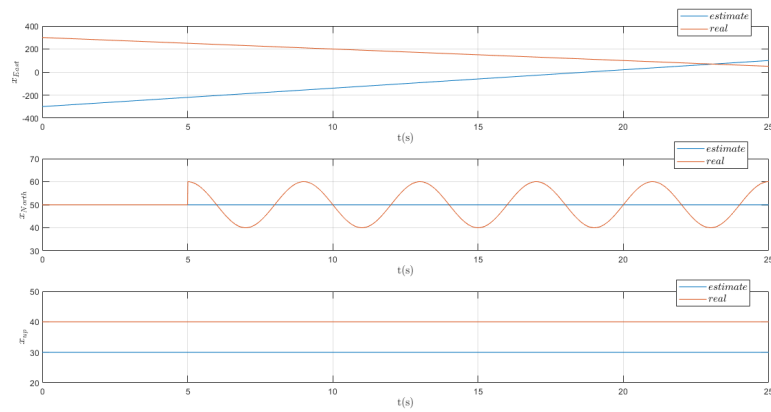


Figura 4.1 True data

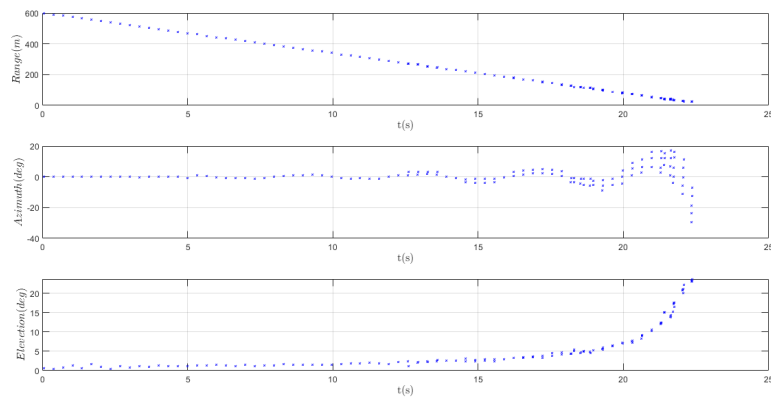


Figura 4.2 Measurements of the radar

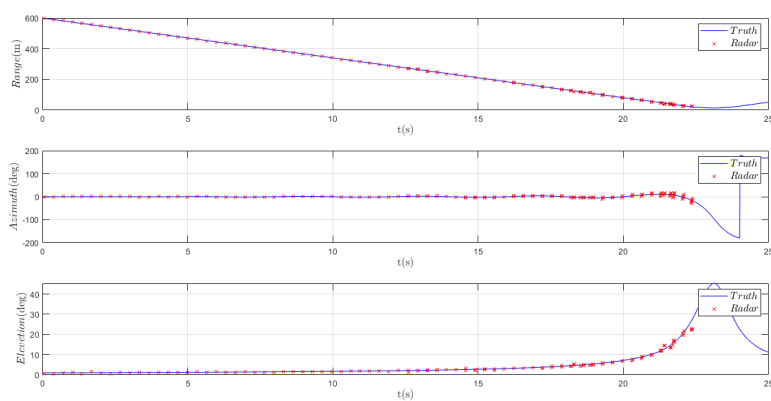


Figure 4.3 Comparison between true range, azimuth and elevation and measured one

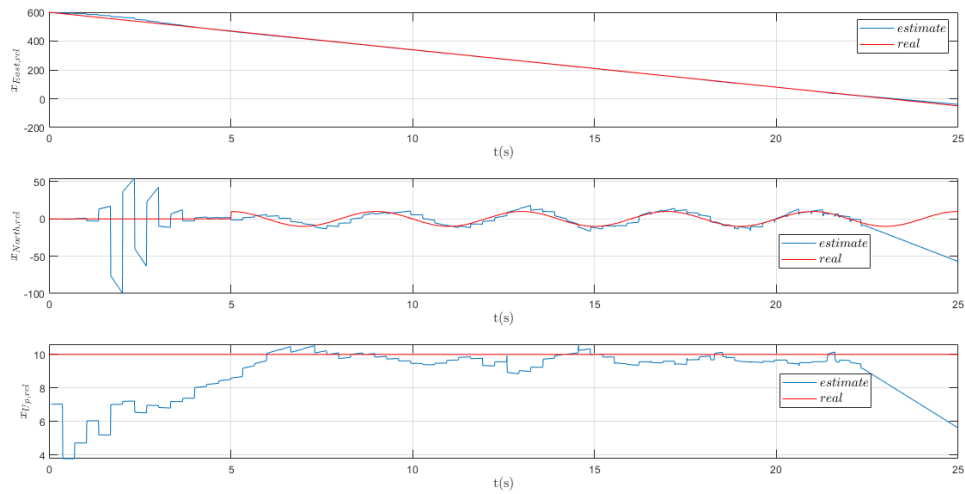


Figure 4.4 comparison true and EKF-estimated data

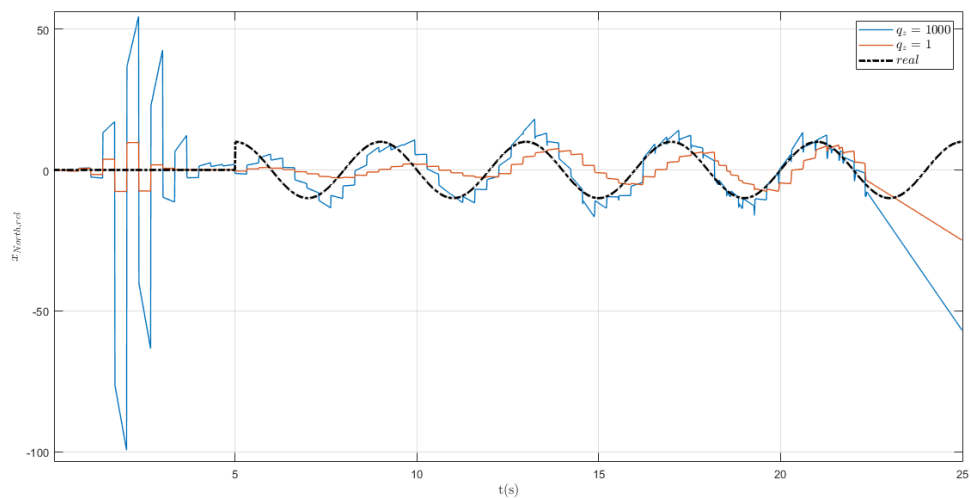


Figure 4.5 comparison true and EKF-estimated X_n for two different value of q_z

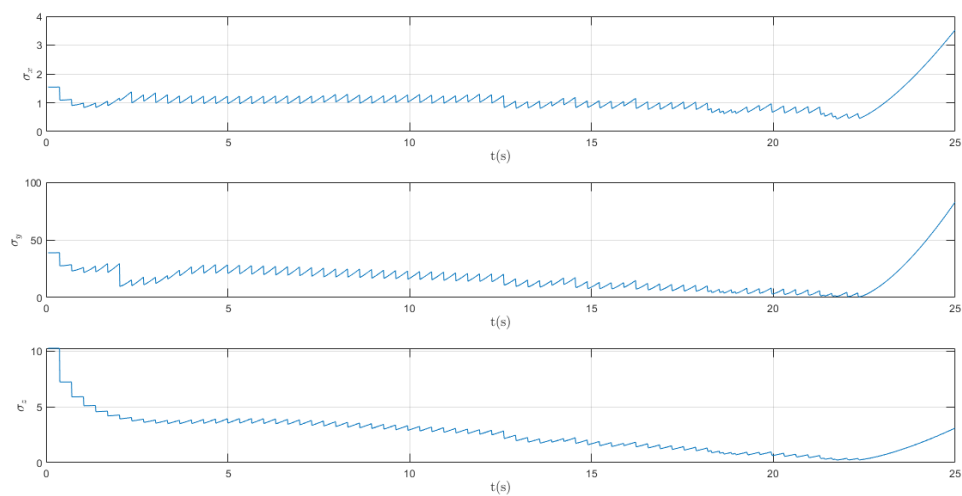


Figure 4.6 standard deviations σ_x , σ_y , σ_z