Tracking for collision avoidance applications

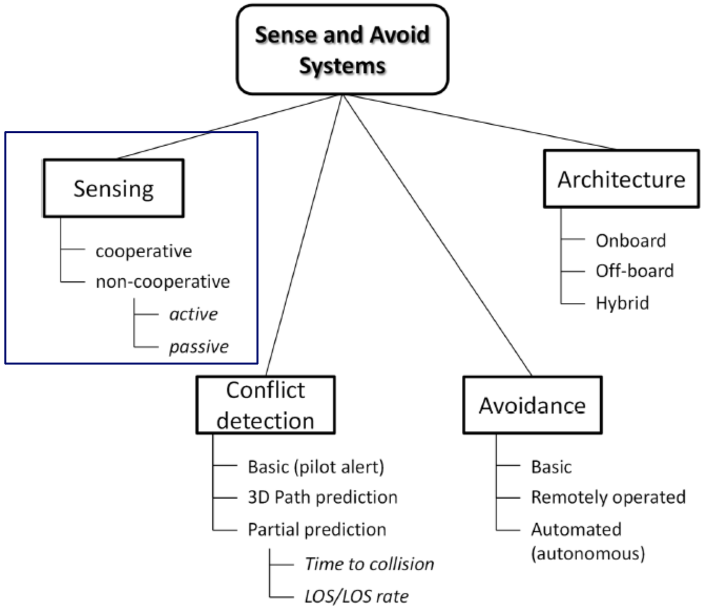
Federica Vitiello
federica.vitiello@unina.it

UNMANNED AIRCRAFT SYSTEMS – A.A 2024/2025

Concepts recap

SENSE AND AVOID

Sensing the environment around the aircraft and **avoiding** any possible collision with other aircrafts/obstacles (moving or fixed).



Focus on the **SENSING** task

DETECT

Using sensors to retrieve information on the environment surrounding the aircraft and detect any possible intruder.

- **Simulated** UAV scenario (MATLAB)

TRACK

Using retrieved detections to estimate tracks of potential intruders.

- **Kalman Filter** design on simulation data (MATLAB)

Requirements

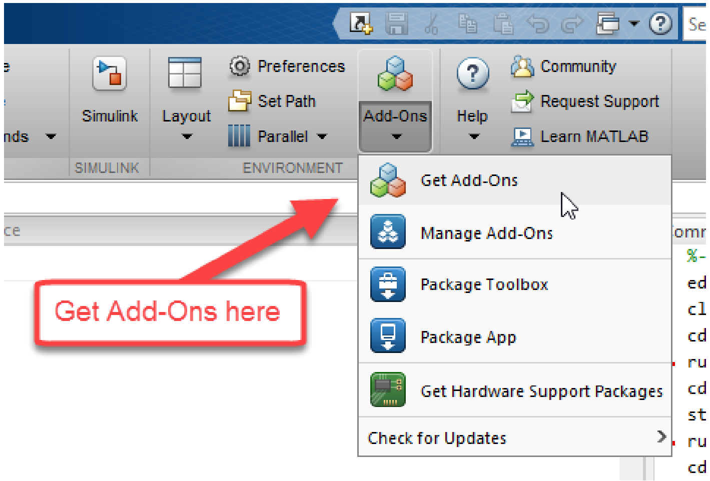
Steps to develop today's applications

Install MATLAB software

- New **R2024a** version suggested

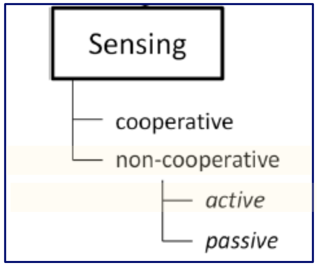
Install needed add-ons

- UAV Toolbox
- Radar Toolbox
- Sensor Fusion and Tracking toolbox
- Mapping Toolbox



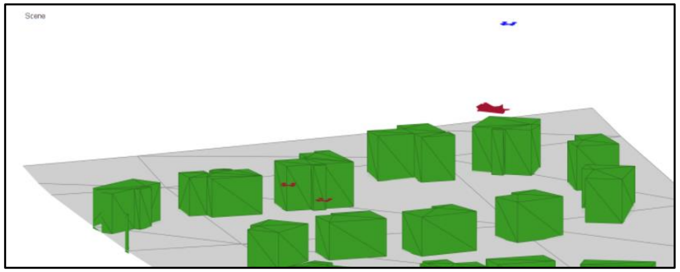
Overview

D
E
T
E
C
T



SIMULATION

The detection task will be covered by a non-cooperative, active **RADAR** sensor mounted onboard an ownship UAV (**ego**), detecting an intruder UAV (**target**).



Simulation goals:

- Generate data for tracking exercise
- Gain confidence with the UAV toolbox

T
R
A
C
K

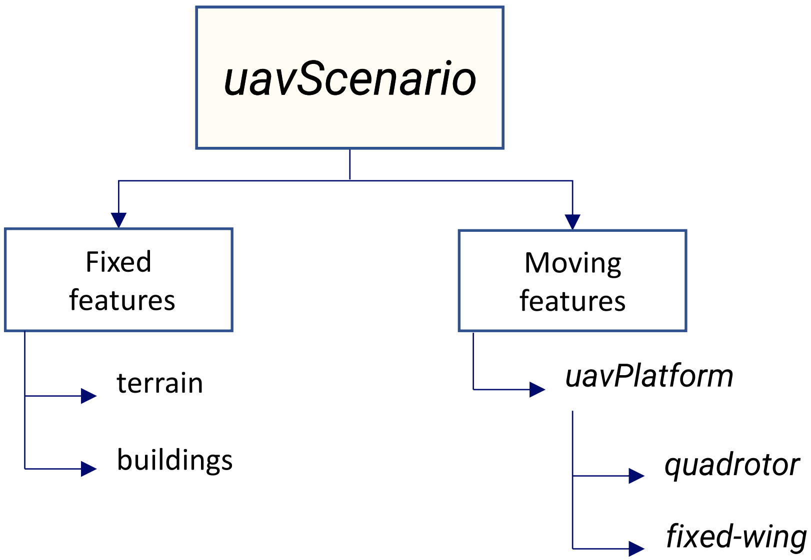
YOUR EXERCISE

On data retrieved with simulation:

- Investigate accuracy of radar-retrieved data with respect to truth
- Generate a tracker based on the **Nearly-Constant Velocity Extended Kalman Filter** model

Simulation – uavScenario

Simulation scenario definition – scene object



object

uavScenario

- **UpdateRate**
  Frequency with which the scene is updated (Hz)
- **ReferenceLocation**
  scenario origin in geodetic coordinates
  [latitude (°), longitude (°),altitude$_{aboveWGS84}$ (m)]
- **StopTime**
  Time at which simulation is stopped (s)

MATLAB function

uavScenario → Leading to the definition of a generic scene object with ENU or NED reference (inertial) frames on which features can be added.

Simulation – fixed features

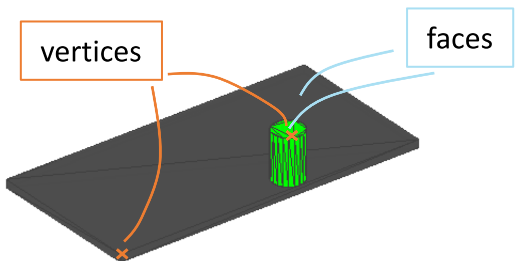
Simulation scenario definition – fixed features

uavScenario

- Fixed features
  - terrain
  - buildings
- Moving features
  - *uavPlatform*
    - *quadrotor*
    - *fixed-wing*

Fixed features

Terrain and buildings can be generated or uploaded. In both cases they are represented by **meshes** in the scene.

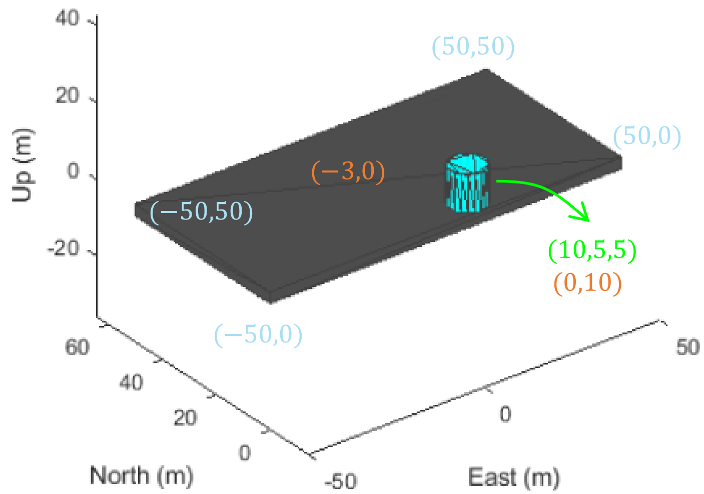


MATLAB function

addMesh → Used for defining both terrain and buildings meshes

Simulation – fixed features

Simulation scenario definition – fixed features



Terrain is a polygon defined by:

- Extension in North-East plane with corner points (x_E, x_N)
- Extension in Up direction with lower and upper limits $(x_{U,min}, x_{U,max})$

Buildings are cylinders defined by:

- Center corrdinates in North-East plane with radius in meters $(x_{C,E}, x_{C,N}, r)$
- Extension in Up direction with lower and upper limits $(x_{U,min}, x_{U,max})$

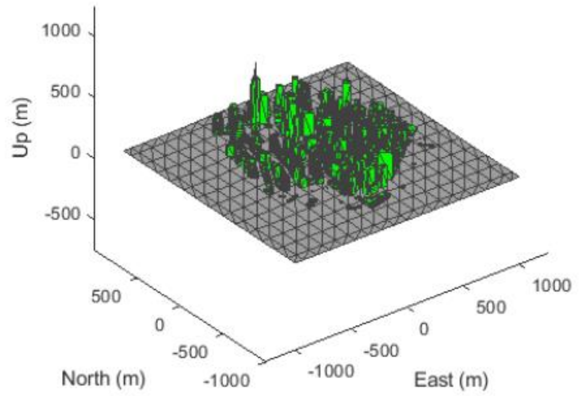
Fixed features

Terrain and buildings can be generated or uploaded. In both cases they are represented by **meshes** in the scene.

- Generation of terrain and buildings can be achieved by using simple extruded polygons placed in the scene.

Simulation – fixed features

Simulation scenario definition – fixed features



Terrain is defined by:
- Considering a reference location in terms of geodetic coordinates of the scene origin (latitude(°),longitude(°),altitude$_{aboveWGS84}$(m)).
- Defining upper and lower bounds for terrain extension in North-East plane.

Buildings are added by
- Uploading .osm file exported from Open Street Map.
- Defining upper and lower bounds for buildings extension in North-East plane.

Fixed features

Terrain and buildings can be generated or uploaded. In both cases they are represented by **meshes** in the scene.

- Generation of terrain and buildings can be achieved by using simple extruded polygons placed in the scene.

- Uploading of terrain data can be achieved by exploiting available Matlab Digital Terrain model dataset (gmted2010).

  Uploading of buildings data can be achieved by using Open Street Map data (uploading .osm file)

Simulation –buildings .osm file

Steps to download .osm file

1. Open https://www.openstreetmap.org/,
2. Navigate/search the location of interest,
3. Export the .osm file of that location.

TAKE NOTE OF THESE VALUES!

Latitude (λ) and longitude (ϕ) minimum and maximum values of the location of interest. These are used for defining:

- Scene reference location center

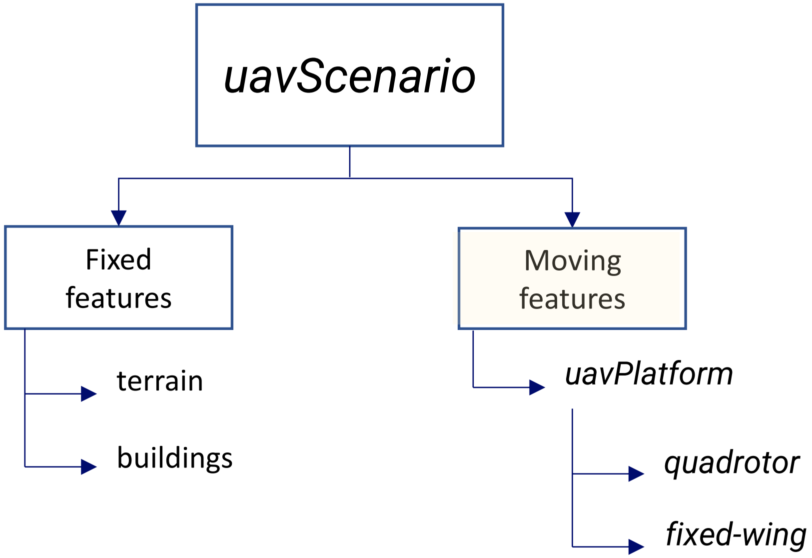
$$\lambda_c = \frac{\lambda_{min} + \lambda_{max}}{2}, \phi_c = \frac{\phi_{min} + \phi_{max}}{2}$$

  Altitude above WGS84 ellipsoid can be computed with MATLAB functions starting from the knowledge of (λ_c, ϕ_c).

- Scene upper and lower bounds in North-East plane which can be computed by considering the $(\lambda_{min}, \phi_{min})$ and $(\lambda_{max}, \phi_{max})$ points.

Simulation – moving features

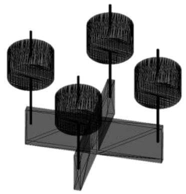
Simulation scenario definition – moving features



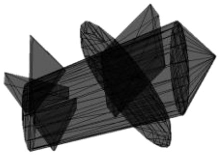
Moving features

UAV platforms added to the scenario are also represented by meshes.

quadrotor mesh



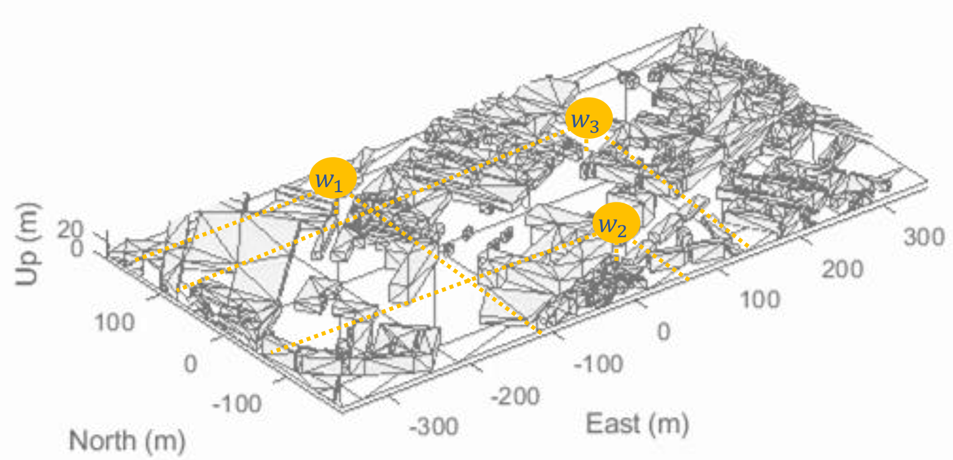
fixed-wing mesh



Their motion on the scenario is defined in terms of **waypoint trajectories** with respect to one of the scene intertial reference frames (ENU or NED).

Simulation – moving features

Simulation scenario definition – moving features trajectories



If the trajectory reference frame is of type ENU and we want the trajectory to be made of three waypoints, then we need to define:

Waypoints	Times of arrival
$w_1 = [x_{E1}, x_{N1}, x_{U1}]$	t_1
$w_2 = [x_{E2}, x_{N2}, x_{U2}]$	t_2
$w_3 = [x_{E3}, x_{N3}, x_{U3}]$	t_3

Orientation can be either:

- Defined by a set of three quaternions or rotation matrices,
- Not specified

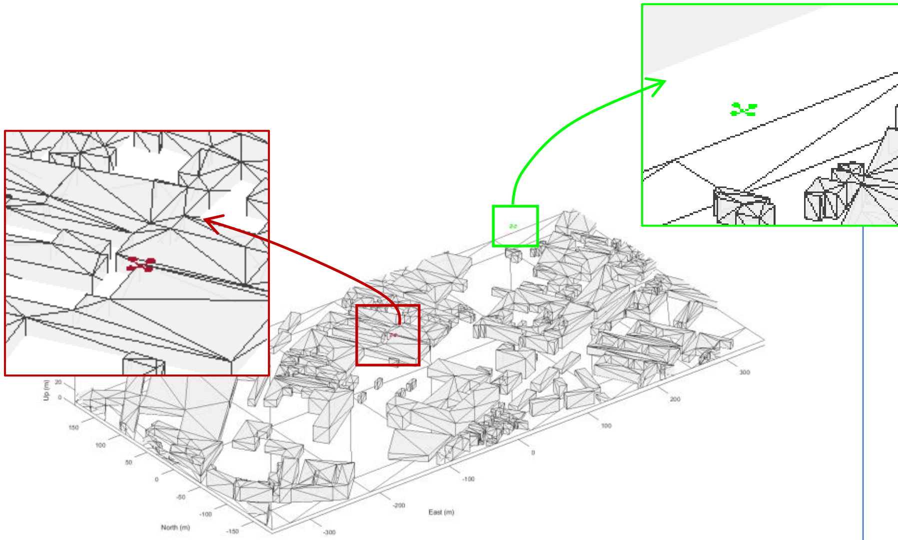
Moving features

Waypoint trajectories are defined based on:

- **Waypoints**
  Points in the scene through which platform must pass,
- **Times of arrival**
  Times at which the plafrom passes through each waypoint,
- **Orientation**
  Platform orientation (attitude of body frame with respect to local frame) at each waypoint passage

Simulation – moving features

Simulation scenario definition – moving features trajectories



Basic idea:
simulating an encounter between egoUAV and intruder

Moving features

In our simulation we will define two quadrotor platforms moving with respect to a ENU reference frame.

egoUAV

Equipped with onboard radar.

targetUAV

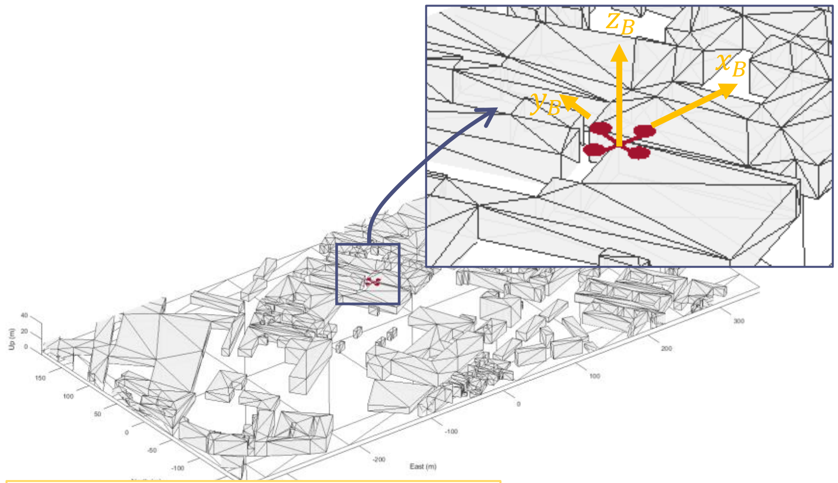
Intruder, target of tracking application.

MATLAB functions

- waypointTrajectory
- uavPlatform
- updateMesh

Simulation – moving features

Simulation scenario definition – moving features notes on orientation



Moving features

In our simulation the orientation is defined based on:

- 'X' notation for the body reference frame, with the x-axis (forward) in the direction of motion,
- Forward(x)-Left(y)-Up(z) body reference frame,
- Pitch and roll angles autonomously defined as to have the body z-axis in the direction of the net acceleration (Autobank property).

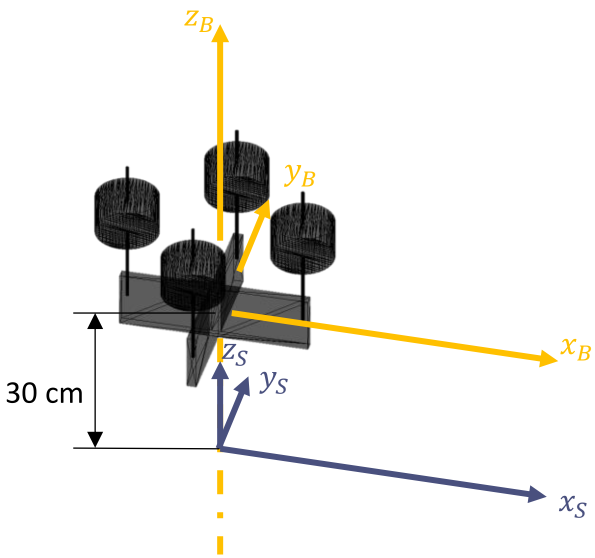
MATLAB functions

read

At each scenario time update retrieve platform motion information:

- Position in scene (x_E, x_N, x_U), m
- Velocity in scene (v_E, v_N, v_U), m/s
- Acceleration in scene (a_E, a_N, a_U), m/s^2
- Orientation quaternion (q_w, q_x, q_y, q_z)
- Angular velocity $(\omega_E, \omega_N, \omega_U)$, rad/s

Simulation – Radar sensor



In our simulation the radar is:

- Mounted 30 cm below the origin of body frame,
  Mounting location in BRF: (0, 0, -0.3), meters

- Oriented as the egoUAV body frame,
  Mounting angles: (0, 0, 0), degrees

radar sensor

A radar sensor can be simulated to be rigidly attached to the egoUAV platform by defining:

- Mounting parameters
  Location and orientation with respect to the platform body frame

- Operating properties
  Field of view, resolutions, operating frequency, output measures format

The rotation matrix from sensor to body reference frame is identical

$$R_B^S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Simulation – Radar sensor

Echoflight MESA radar – Electronically scanning metamaterial radar

ECHODYNE

https://www.echodyne.com/defense/uav-radar/

Field of view	120° azimuth x 80° elevation
Angular resolution	2° azimuth x 6° elevation
Frequency	K-band. 24.45 - 24.65 GHz (multichannel)
Size	20.3 cm x 16.3 cm x 4 cm
Data output	R/Vmaps: 40 MB/s Detections: 1 MB/s Measurements: 1 MB/s Tracks: 25 kB/s

radar sensor

A radar sensor can be simulated to be rigidly attached to the egoUAV platform by defining:

- Mounting parameters
  Location and orientation with respect to the platform body frame
- Operating properties
  Field of view, resolutions, operating frequency, output measures format

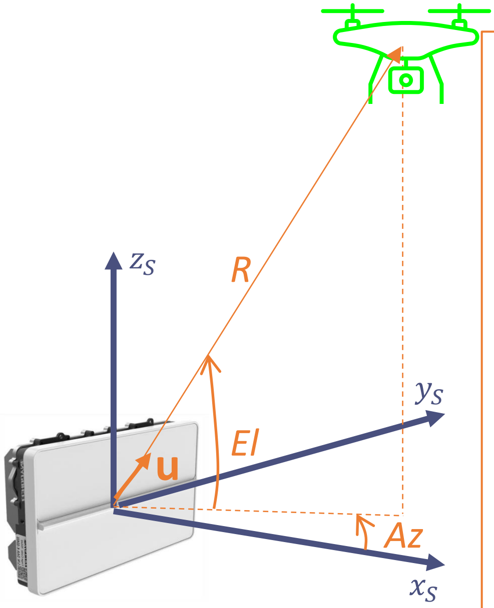
MATLAB functions

- radarDataGenerator
- helperRadarAdaptor

RadarDataGenerator initializes radar sensor with user-defined properties.
helperRadarAdaptor function is needed to simulate the mounting of the radar sensor onboard the UAV platform.

Simulation – Radar sensor output

Echoflight MESA radar – simulated output measures format



Spherical target coordinates in sensor frame:

- Range (R, m), distance between sensor and target.
- Azimuth (Az, °), angle between boresight (x_s) and **u** projection on x_s, y_s plane.
- Elevation (El, °), angle between **u** and its projection on x_s, y_s plane.

radar sensor

A radar sensor can be simulated to be rigidly attached to the egoUAV platform by defining:

- Mounting parameters
  Location and orientation with respect to the platform body frame
- Operating properties
  Field of view, resolutions, operating frequency, **output measures format**

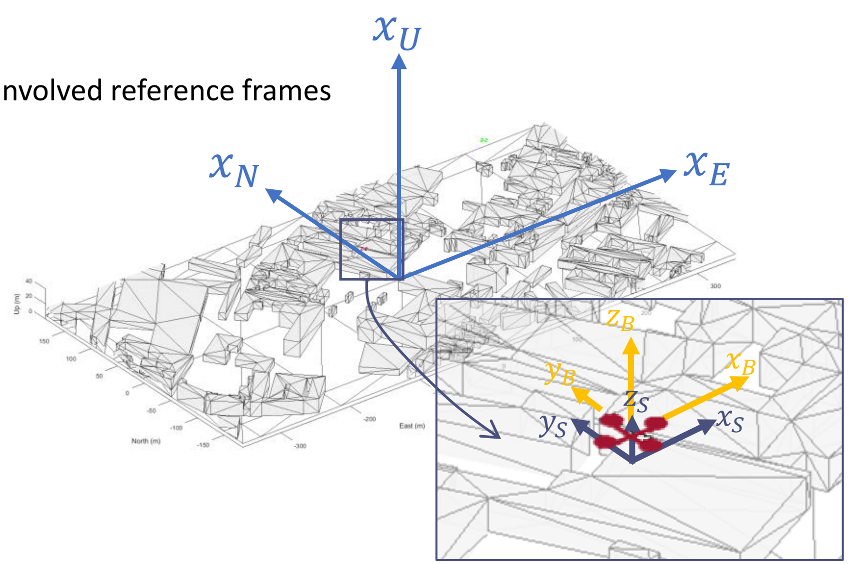
MATLAB functions

GetRadarDet → Function used to overcome MATLAB toolbox limits and achieve spherical detections expressed in sensor reference frame (emulating truth).

Simulation setup

Parameters for simulation setup

Involved reference frames



Simulation overview

- **E-N-U** oriented scenario
- Duration **25 s**
- Update frequency **100 Hz**
- Two quadrotors (ego VS target) during encounter

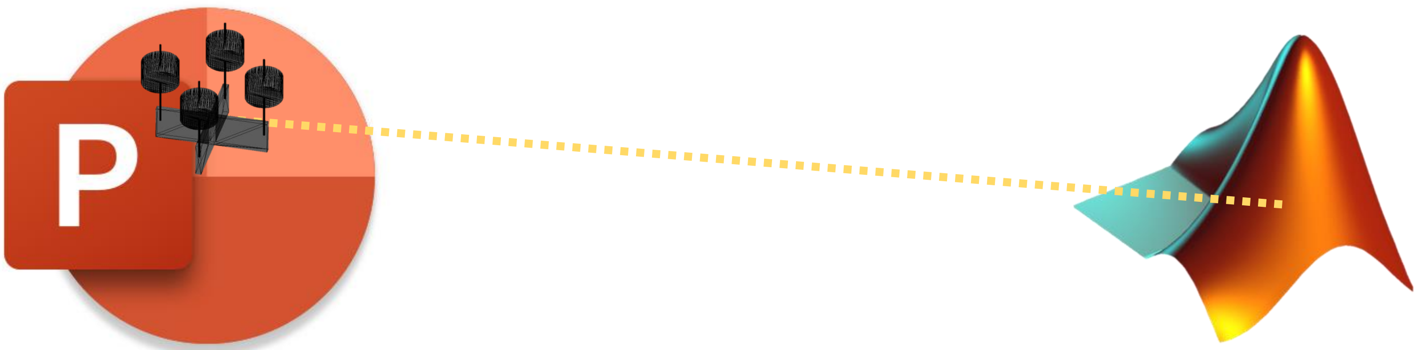
Simulation Sensors

- INS sensor onboard each quadrotor
  - retrieve truth during flight at **100 Hz**
- Radar sensor
  - retrieve target R,Az,El at **100 Hz**
  - monitoring a fixed FOV of [66°,30°]

MATLAB functions

setup
advance
updatsensors

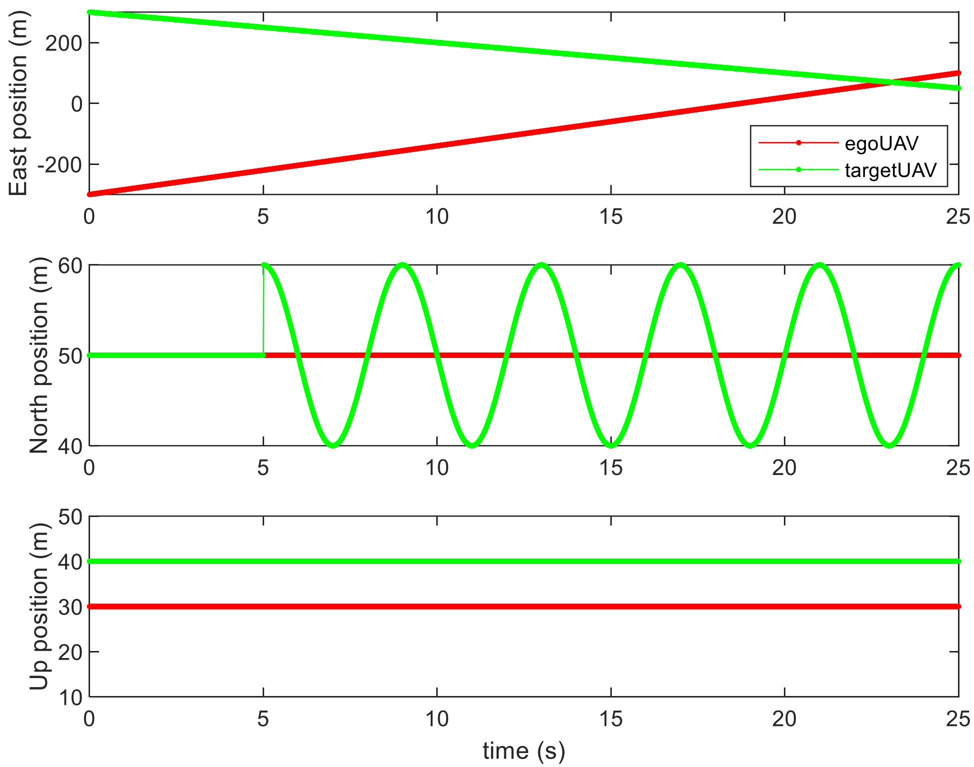
Simulation

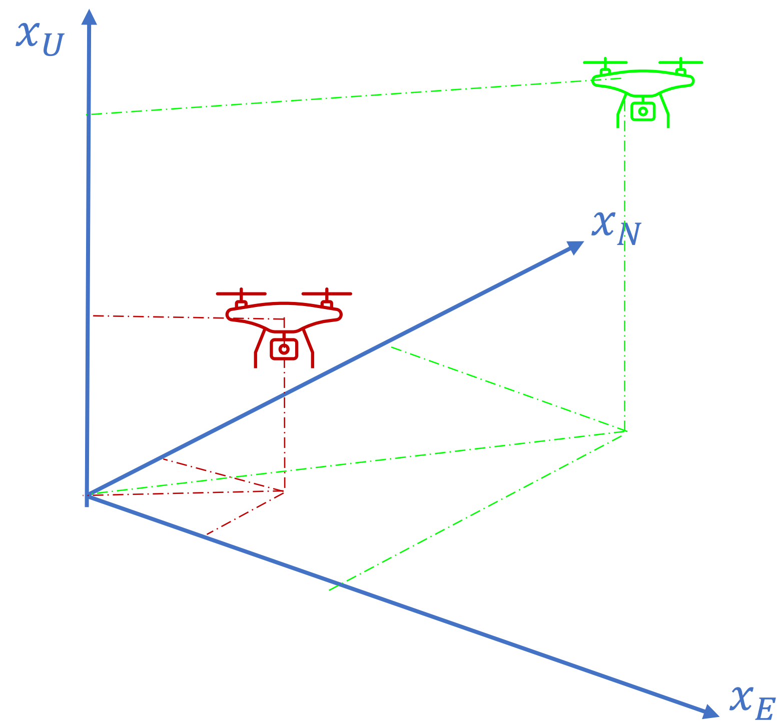


Simulation output - truth

Truth data is retrieved in terms of absolute position of target and ownship platforms in scene (**ENU**) reference frame with a frequency of **100 Hz .**

Data cover the whole encounter duration.

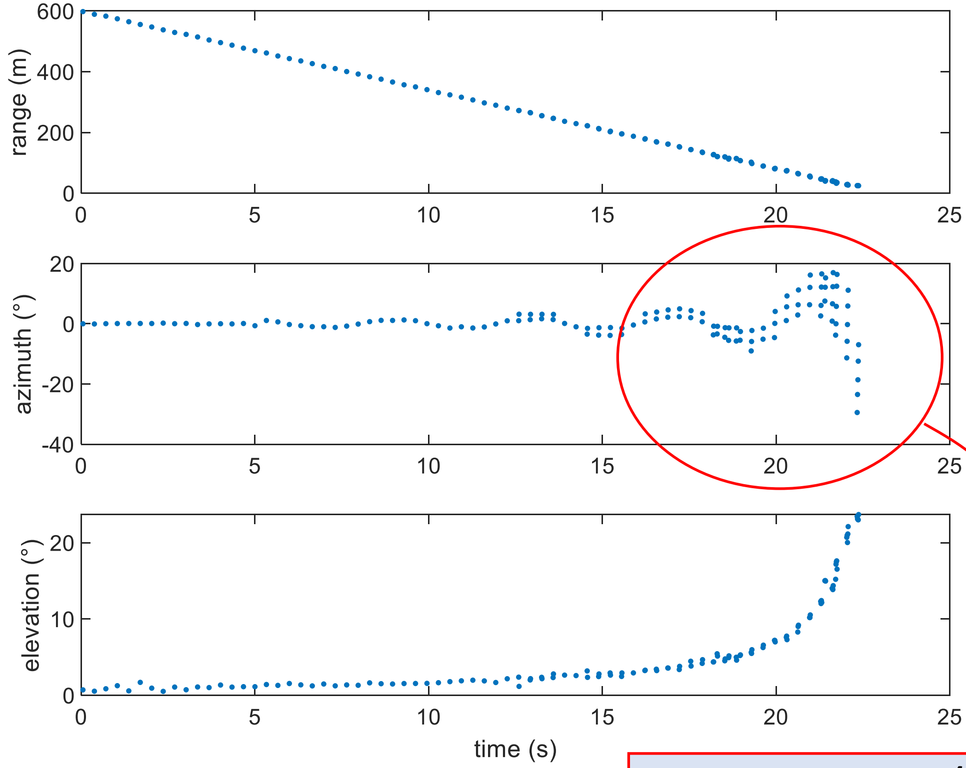


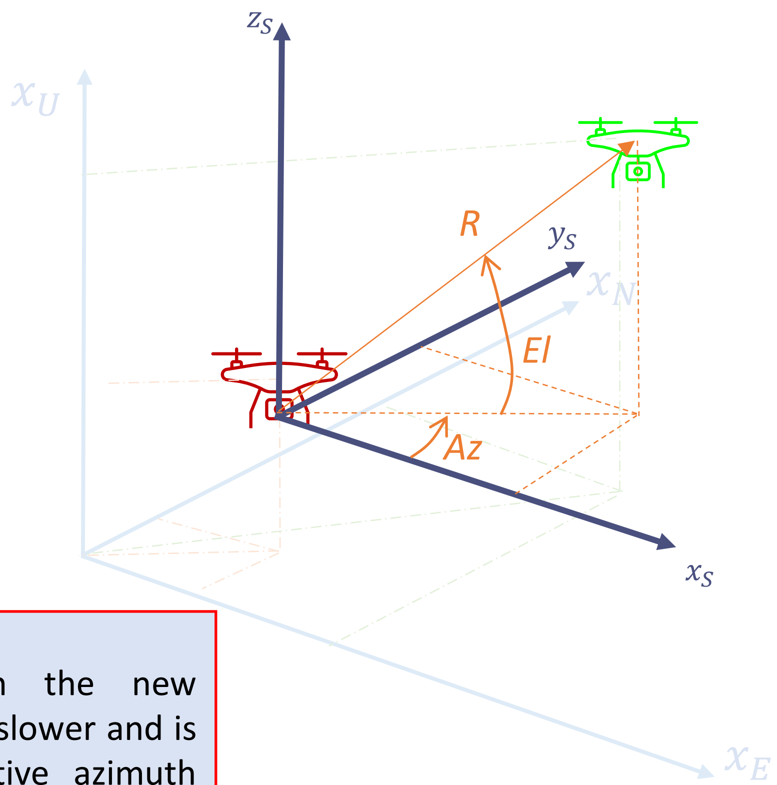


Simulation output - radar

Radar-retrieved data consists of range, azimuth and elevation of target platform with respect to ownship in **sensor reference frame** at a frequency of **100 Hz**

Data only cover times during which target platform is inside the radar FOV (up to about 22 s).





FYI (new)

This happens because in the new simulation the target moves slower and is detected between consecutive azimuth cells by the radar.

Radar accuracy

How has the radar performed during the encounter?

Sensor accuracy can be estimated by **comparing measures with truth**

$\boxed{[R, Az, El]^{SRF}}$

$\boxed{\begin{array}{l} [x_{tgt}, y_{tgt}, z_{tgt}]^{ENU} \\ [x_{ego}, y_{ego}, z_{ego}]^{ENU} \end{array}}$

To make comaprison possible we have to:

- Transform absolute cartesian truth in $[R_{truth}, Az_{truth}, El_{truth}]$
- Transform measured $[R, Az, El]$ from SRF to ENU

Radar accuracy

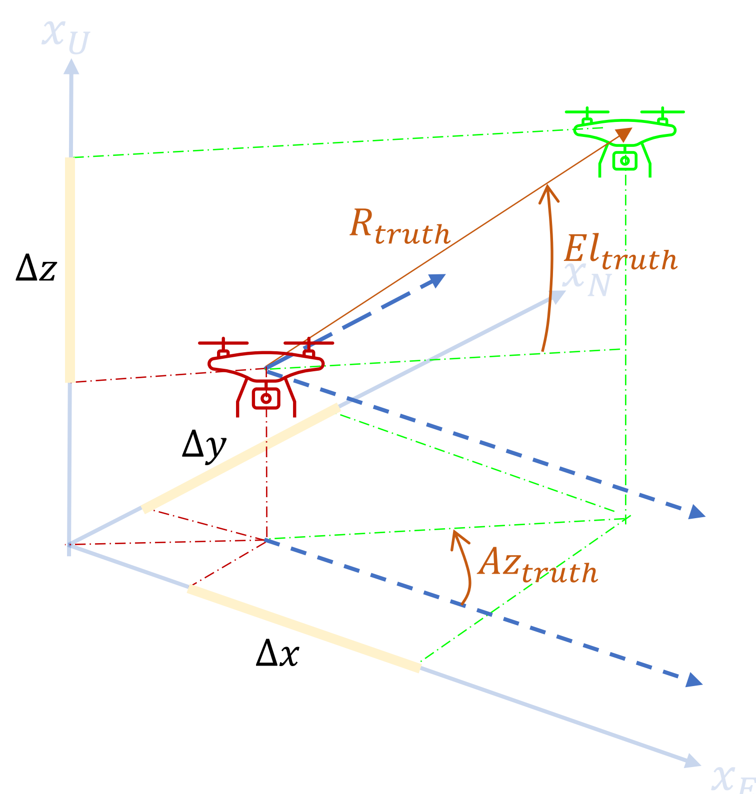
How has the radar performed during the encounter?

Sensor accuracy can be estimated by **comparing measures with truth**

$[R, Az, El]^{SRF}$

$[x_{tgt}, y_{tgt}, z_{tgt}]^{ENU}$

$[x_{ego}, y_{ego}, z_{ego}]^{ENU}$



Must be transformed in range, azimuth, elevation by considering relative position of target with respect to ownship

$$\Delta x = x_{tgt} - x_{ego},$$
$$\Delta y = y_{tgt} - y_{ego},$$
$$\Delta z = z_{tgt} - z_{ego},$$

$$R_{truth} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$
$$Az_{truth} = atan2\left(\frac{\Delta y}{\Delta x}\right)$$
$$El_{truth} = \operatorname{asin}\left(\frac{\Delta z}{R}\right)$$

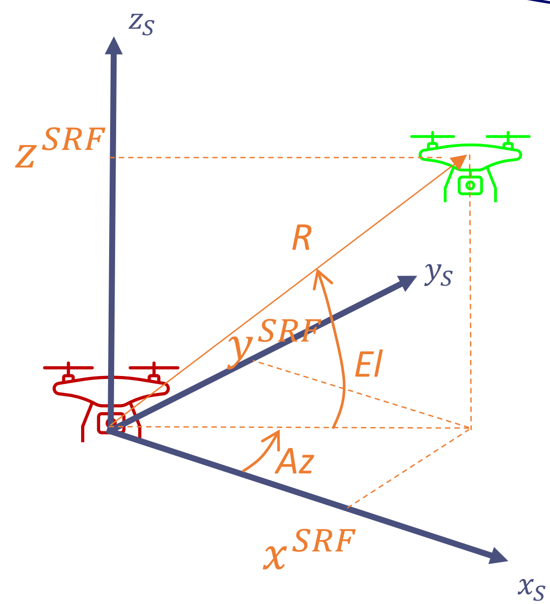
Radar accuracy

How has the radar performed during the encounter?

Sensor accuracy can be estimated by **comparing measures with truth**

$$[R, Az, El]^{SRF}$$

$$\left[x_{tgt}, y_{tgt}, z_{tgt}\right]^{ENU}$$
$$\left[x_{ego}, y_{ego}, z_{ego}\right]^{ENU}$$



Must be transformed in ENU by:

- Computing cartesian measures in SRF

$$x^{SRF} = R\cos(el)\cos(az)$$
$$y^{SRF} = R\cos(el)\sin(az)$$
$$z^{SRF} = R\sin(el)$$

Radar accuracy

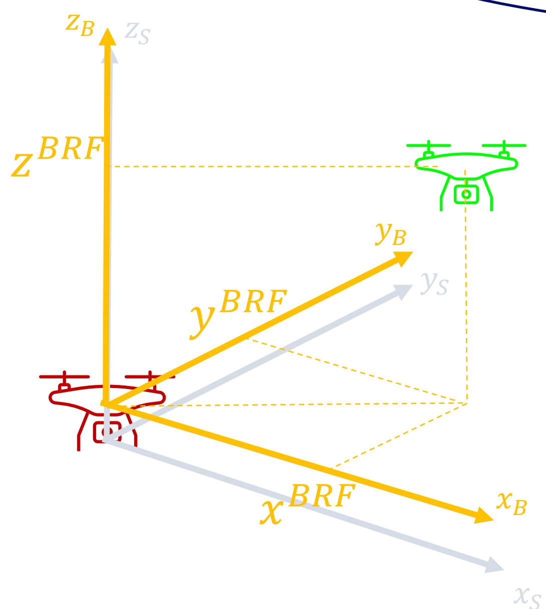
How has the radar performed during the encounter?

Sensor accuracy can be estimated by **comparing measures with truth**

$[R, Az, El]^{SRF}$

$[x_{tgt}, y_{tgt}, z_{tgt}]^{ENU}$

$[x_{ego}, y_{ego}, z_{ego}]^{ENU}$



Must be transformed in ENU by:

- Computing cartesian measures in SRF
- Transforming cartesian measures from SRF to BRF accounting for sensor mounting orientation and location

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{BRF} = R_S^B \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{SRF} - \begin{bmatrix} x_M \\ y_M \\ z_M \end{bmatrix}$$

Radar accuracy

How has the radar performed during the encounter?

Sensor accuracy can be estimated by **comparing measures with truth**

$\boxed{[R, Az, El]^{SRF}}$

$$\boxed{\begin{matrix}[x_{tgt}, y_{tgt}, z_{tgt}]^{ENU} \\ [x_{ego}, y_{ego}, z_{ego}]^{ENU}\end{matrix}}$$

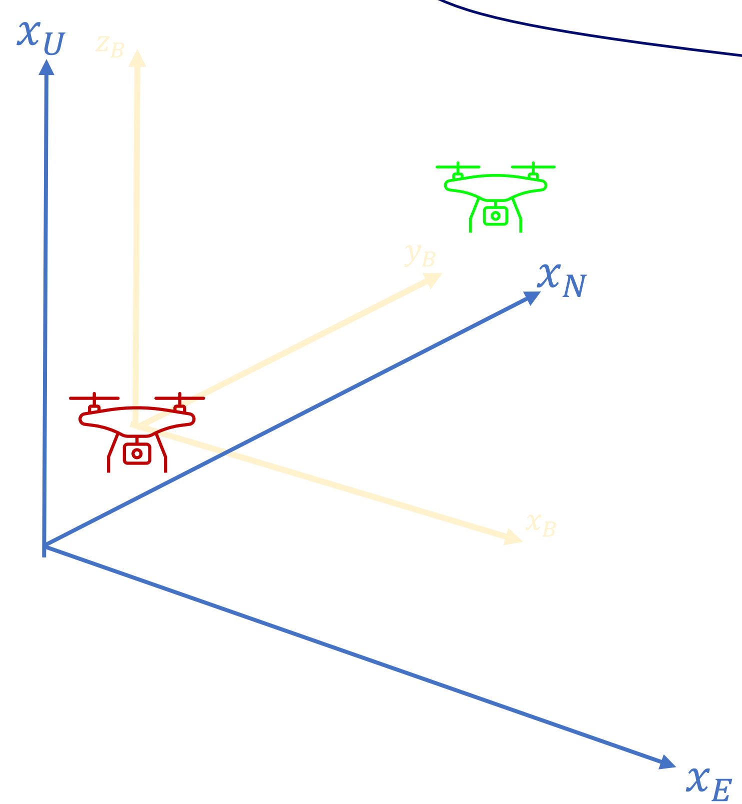
Must be transformed in ENU by:

- Computing cartesian measures in SRF
- Transforming cartesian measures from SRF to BRF accounting for sensor mounting orientation and location
- Transforming cartesian measures from BRF to ENU by accounting for ownship attitude

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{ENU} = R_B^E \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{BRF}$$

These can be transformed in

$[R, Az, El]^{ENU}$



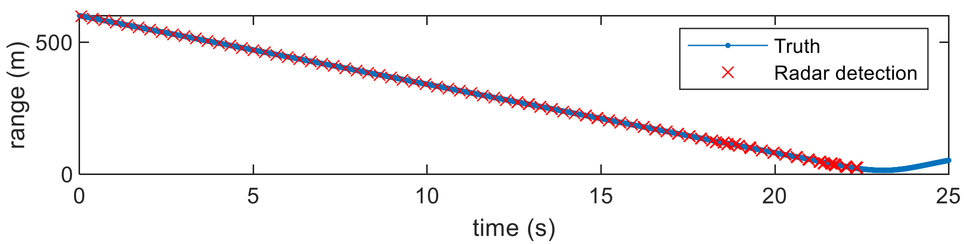
Radar accuracy – exercise

Inspect simulation data.

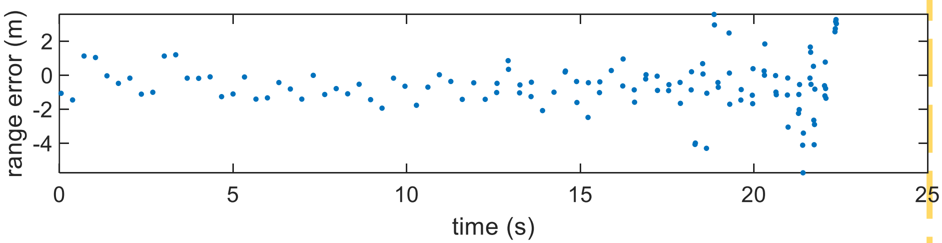
- Plot truth and measured quantities with respect to time

Write a MATLAB code to retrieve radar **accuracy** in range (σ_R), azimtuh (σ_{az}) and elevation (σ_{el}).

- Truth data must be expressed in terms of range, azimuth, elevation.
- Radar data retrieved in SRF must be transformed in ENU.
- Range, azimuth and elevation errors (measured-truth) must be computed accounting for different acquisition frequency.
- Error standard deviation must be computed.



Truth VS radar-retrieved range (ENU) in time



Range error (ENU) in time

Tracking

Estimate track of the target UAV by exploiting radar-retrieved measures using **Constant Velocity Extended Kalman filters** (**EKF**).

state, $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU}$ measures, $\mathbf{z} = [R, Az, El]^{ENU}$

MODEL

Assumption: target moves with constant velocity

State update model – Representing state transition (**PREDICTION**) from time *k* to time *k+1* exploiting (linear) dynamic model.

Continuous form

noise

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, t) + w_k$$

LINEAR

Measurement model – Representing the relationship between observation (measure) and state which will be used for state update **(FILTERING).**

Continuous form

noise

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k, t) + v_k$$

NON-LINEAR

$$R = \sqrt{x^2 + y^2 + z^2} \longrightarrow h_1$$

$$Az = atan2\left(\frac{y}{x}\right) \longrightarrow h_2$$

$$El = \text{asin}\left(\frac{z}{R}\right) \longrightarrow h_3$$

Tracking

MODEL

$$\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU} \qquad \mathbf{z} = [R, Az, El]^{ENU}$$

State/covariance prediction

$$\hat{\boldsymbol{x}}_{k+1} = \Phi\hat{\boldsymbol{x}}_k$$
$$\hat{P}_{k+1} = \Phi\hat{P}_k\Phi^T + Q$$

$$\Phi = \begin{bmatrix} F & 0 & 0 \\ 0 & F & 0 \\ 0 & 0 & F \end{bmatrix} \qquad F = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}$$

State transition matrix

T: filter sampling time

$$Q = \begin{bmatrix} Q_x & 0 & 0 \\ 0 & Q_y & 0 \\ 0 & 0 & Q_z \end{bmatrix} Q_i = q_i \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix}$$

Process noise matrix

q: scale factor

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{x\dot{x}} & \sigma_{xy} & \sigma_{x\dot{y}} & \sigma_{xz} & \sigma_{x\dot{z}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}y} & \sigma_{\dot{x}\dot{y}} & \sigma_{\dot{x}z} & \sigma_{\dot{x}\dot{z}} \\ \sigma_{yx} & \sigma_{y\dot{x}} & \sigma_y^2 & \sigma_{y\dot{y}} & \sigma_{yz} & \sigma_{y\dot{z}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}\dot{x}} & \sigma_{\dot{y}y} & \sigma_{\dot{y}}^2 & \sigma_{\dot{y}z} & \sigma_{\dot{y}\dot{z}} \\ \sigma_{zx} & \sigma_{z\dot{x}} & \sigma_{zy} & \sigma_{z\dot{y}} & \sigma_z^2 & \sigma_{z\dot{z}} \\ \sigma_{\dot{z}x} & \sigma_{\dot{z}\dot{x}} & \sigma_{\dot{z}y} & \sigma_{\dot{z}\dot{y}} & \sigma_{\dot{z}z} & \sigma_{\dot{z}}^2 \end{bmatrix}$$

State covariance matrix

Must be initialized

Tuning parameters

MODEL

$\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU}$ $\mathbf{z} = [R, Az, El]^{ENU}$

State filtering (correction)

$$K_{k+1} = \hat{P}_{k+1} H \left[H \hat{P}_{k+1} H^T + R \right]^{-1}$$
$$P_{k+1} = (I - K_{k+1} H) \hat{P}_{k+1}$$
$$\boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1} + K_{k+1}(\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1})$$

- $$H = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_6} \\ \vdots & \vdots & \vdots \\ \frac{\partial h_3}{\partial x_1} & \cdots & \frac{\partial h_3}{\partial x_6} \end{bmatrix}$$

Jacobian of measurement with respect to state

- $$R = \begin{bmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_{az}^2 & 0 \\ 0 & 0 & \sigma_{el}^2 \end{bmatrix}$$

Measurement covariance matrix

With estimated measurement accuracy from previous exercise

Tracking

MODEL

$$\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU} \qquad \mathbf{z} = [R, Az, El]^{ENU}$$

State/covariance initialization

State can be intitialized by using first radar measure

$$\boldsymbol{x_0} = [R_1 cos(az_1)\cos(el_1)\,, 0, R_1 sin(az_1)\cos(el_1)\,, 0, R_1 sin(el_1), 0]$$

State covariance **position elements** can be initialized by computing:

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{x\dot{x}} & \sigma_{xy} & \sigma_{x\dot{y}} & \sigma_{xz} & \sigma_{x\dot{z}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}y} & \sigma_{\dot{x}\dot{y}} & \sigma_{\dot{x}z} & \sigma_{\dot{x}\dot{z}} \\ \sigma_{yx} & \sigma_{y\dot{x}} & \sigma_y^2 & \sigma_{y\dot{y}} & \sigma_{yz} & \sigma_{y\dot{z}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}\dot{x}} & \sigma_{yy} & \sigma_{\dot{y}}^2 & \sigma_{\dot{y}z} & \sigma_{\dot{y}\dot{z}} \\ \sigma_{zx} & \sigma_{z\dot{x}} & \sigma_{zy} & \sigma_{z\dot{y}} & \sigma_z^2 & \sigma_{z\dot{z}} \\ \sigma_{\dot{z}x} & \sigma_{\dot{z}\dot{x}} & \sigma_{\dot{z}y} & \sigma_{\dot{z}\dot{y}} & \sigma_{\dot{z}z} & \sigma_{\dot{z}}^2 \end{bmatrix} \longrightarrow JRJ^T \qquad J = \begin{bmatrix} \frac{\partial x}{\partial R} & \frac{\partial x}{\partial Az} & \frac{\partial x}{\partial El} \\ \frac{\partial y}{\partial R} & \frac{\partial y}{\partial Az} & \frac{\partial y}{\partial El} \\ \frac{\partial z}{\partial R} & \frac{\partial z}{\partial Az} & \frac{\partial y}{\partial El} \end{bmatrix}$$

Jacobian of state with respect to measure

R → Measurement covariance matrix

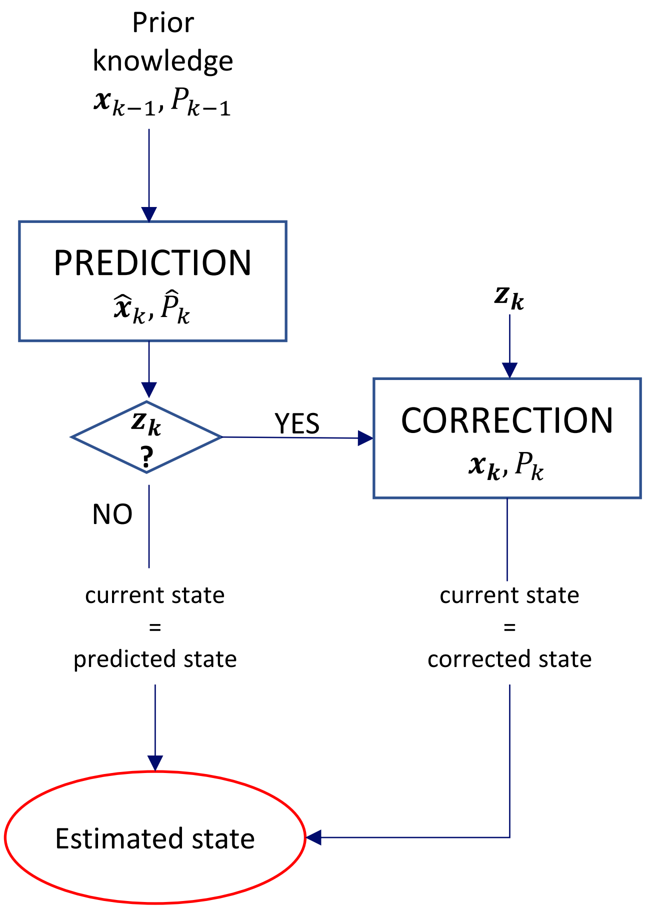
$$\sigma_x^2 = [\cos(el)\cos(az)]^2\sigma_R^2 + [\mathrm{R}\cos(el)\sin(az)]^2\sigma_{az}^2 + [\mathrm{R}\cos(az)\sin(el)]^2\sigma_{el}^2$$
$$\sigma_{xy} = \sin(az)\cos(az)\cos(el)^2\,\sigma_R^2 - R^2\sin(az)\cos(az)\cos(el)^2\,\sigma_{az}^2 + R^2\sin(az)\cos(az)\sin(el)^2\,\sigma_{el}^2$$
$$\sigma_{xz} = -\cos(az)\cos(el)\sin(el)\,\sigma_R^2 + R^2\cos(az)\cos(el)\sin(el)\sigma_{el}^2$$
$$\sigma_y^2 = [\sin(\mathrm{az})\cos(el)]^2\sigma_R^2 + [\mathrm{R}\cos(az)\cos(el)]^2\sigma_{az}^2 + [\mathrm{R}sin(az)\sin(el)]^2\sigma_{el}^2$$
$$\sigma_{yz} = -\sin(az)\cos(el)\sin(el)\,\sigma_R^2 + R^2\sin(az)\cos(el)\sin(el)\,\sigma_{el}^2$$
$$\sigma_z^2 = \sin(el)^2\sigma_R^2 + [Rcos(el)]^2\sigma_{el}^2$$

Tracking

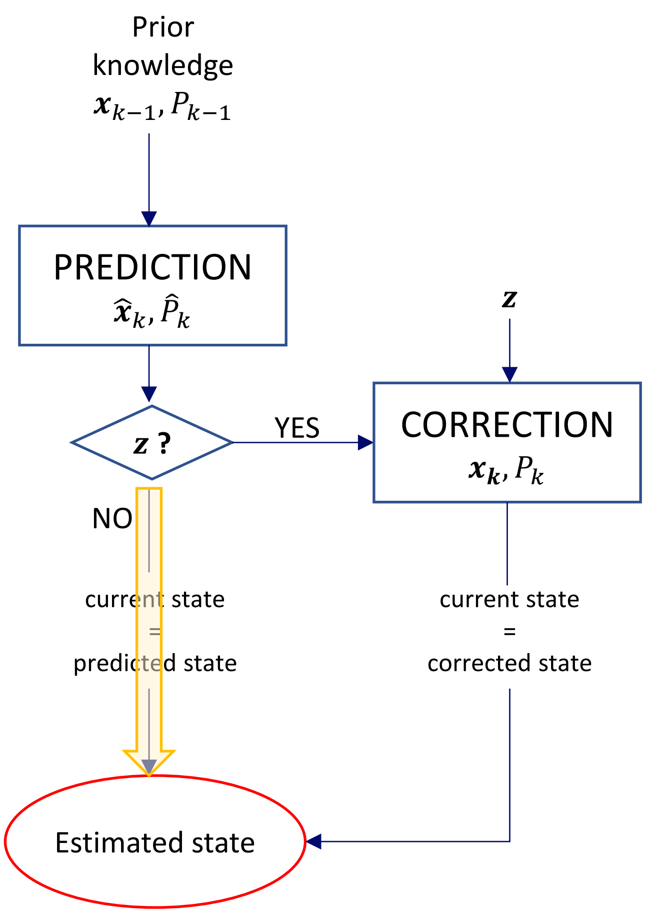
Kalman filter flow chart

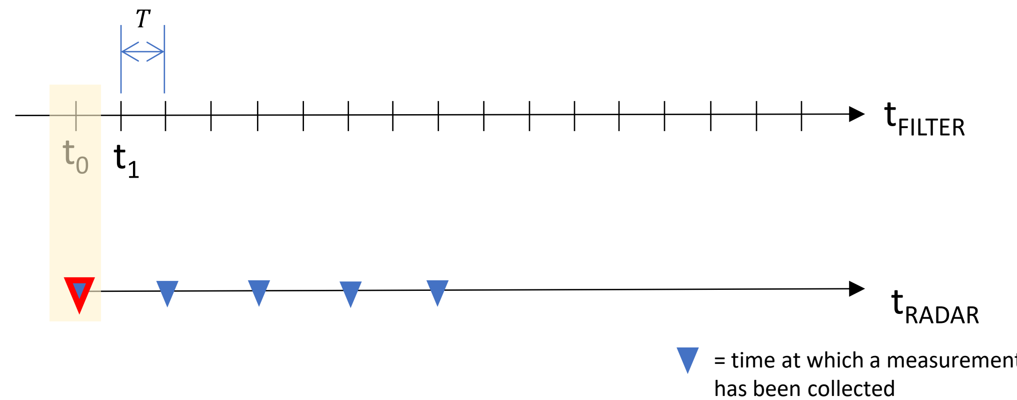
state, $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU}$

measure, $\mathbf{z} = [R, Az, El]^{ENU}$



Kalman filter flow chart state, $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU}$ measure, $\mathbf{z} = [R, Az, El]^{ENU}$





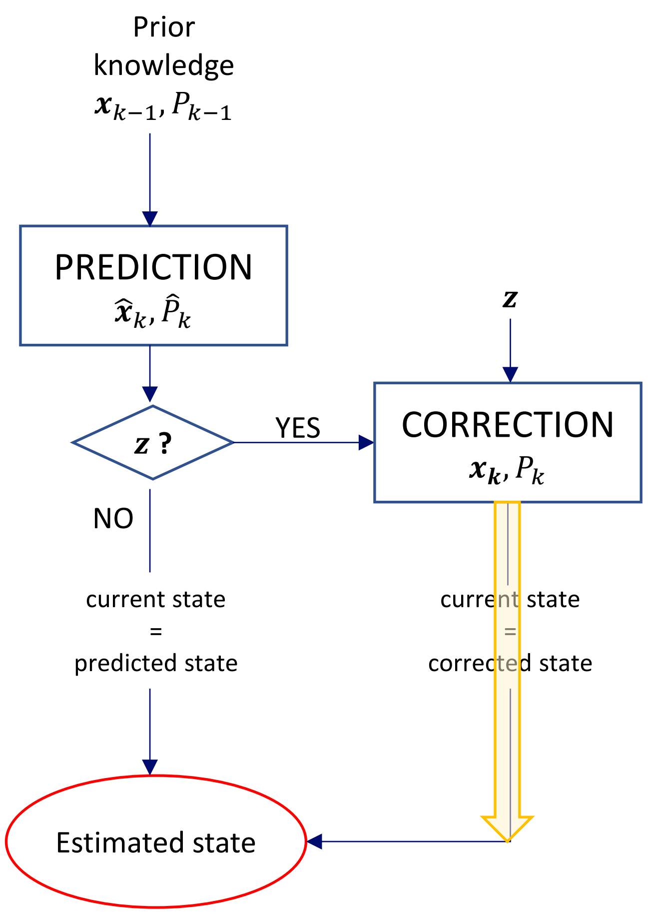
- Filter is initialized at first available radar measure.
- $\boldsymbol{x}_0, P_0$ → prediction → $\hat{\boldsymbol{x}}_1, \hat{P}_1$
- No available measure at t_1, the estimated state is

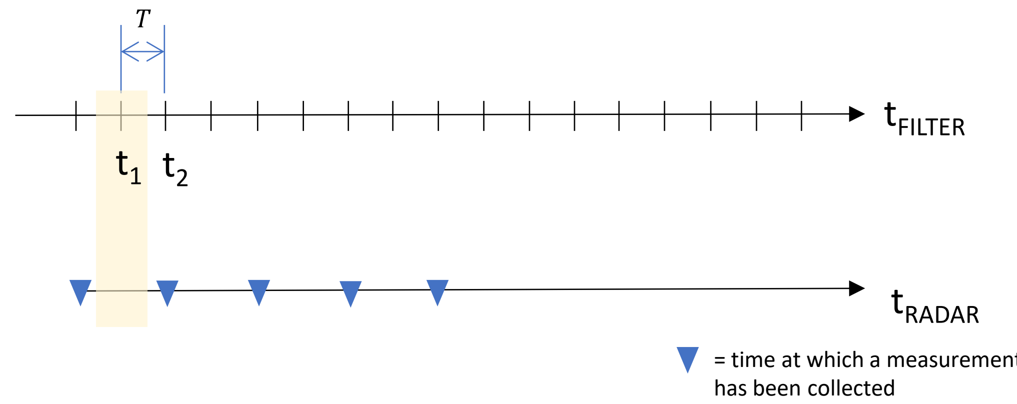
$$\hat{\boldsymbol{x}}_1, \hat{P}_1$$

Tracking

Kalman filter flow chart

state, $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}, z, \dot{z}]^{ENU}$ measure, $\mathbf{z} = [R, Az, El]^{ENU}$





- $\hat{\boldsymbol{x}}_1, \hat{P}_1 \rightarrow$ prediction $\rightarrow \hat{\boldsymbol{x}}_2, \hat{P}_2$

- Available radar measure at t_2

  $\hat{\boldsymbol{x}}_2, \hat{P}_2 \rightarrow$ correction $\rightarrow \boldsymbol{x}_2, P_2$

  Estimated state

Tracking exercise

Develop **EKF** tracking filter to estimate target state during encounter.

- Radar data should be expressed as $\mathbf{z} = [R, Az, El]^{ENU}$
- Set filter sampling time equal to simulation sampling time
- Initialize filter ($\boldsymbol{x_0}, P_0$) using first available radar measure
- Take into account the different frequency between filter time and radar measures time
  - How can you check for radar measure at each filter time step?
- Inspect filter output on state and state covariance
- Evaluate tracking performance by comparing truth and filter-estimated data.