MATLAB EXPO 2019

Sensor Fusion and Tracking for Autonomous Systems

Marc Willerton
Senior Application Engineer
MathWorks

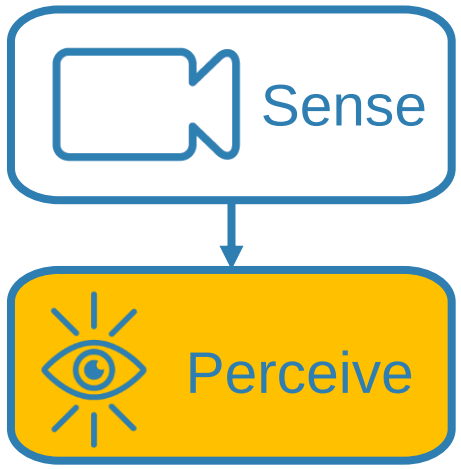
Abstract

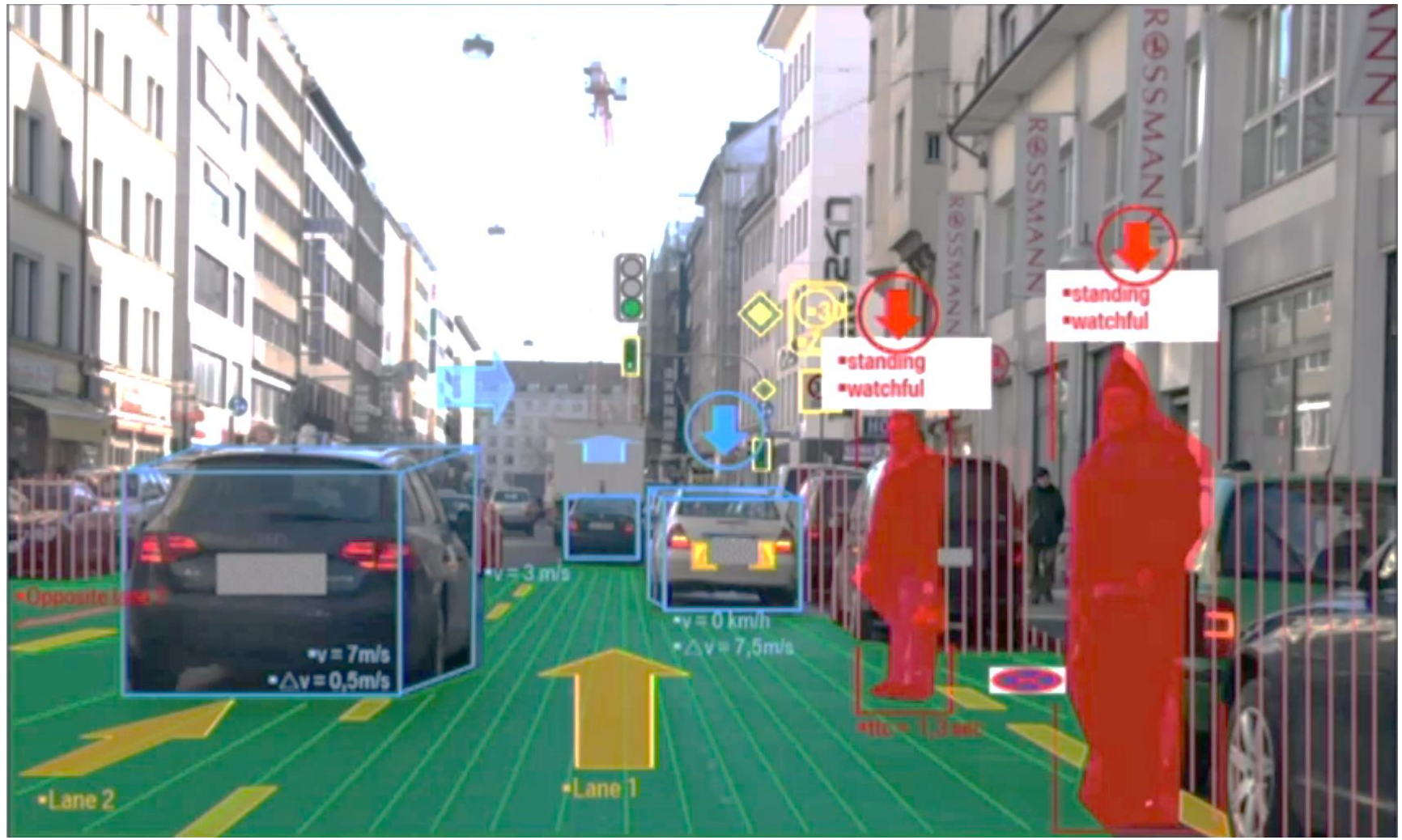
- There is an exponential growth in the development of increasingly autonomous systems. These systems range from road vehicles that meet the various NHTSA levels of autonomy, through consumer quadcopters capable of autonomous flight and remote piloting, package delivery drones, flying taxis, and robots for disaster relief and space exploration. Work on autonomous systems spans industries and includes academia as well as government agencies.
- In this talk, you will learn to design, simulate, and analyze systems that fuse data from multiple sensors to maintain position, orientation, and situational awareness. By fusing multiple sensors data, you ensure a better result than would otherwise be possible by looking at the output of individual sensors.
- Several autonomous system examples are explored to show you how to:
  - Define trajectories and create multiplatform scenarios
  - Simulate measurements from inertial and GPS sensors
  - Generate object detections with radar, EO/IR, sonar, and RWR sensor models
  - Design multi-object trackers as well as fusion and localization algorithms
  - Evaluate system accuracy and performance on real and synthetic data

Capabilities of an Autonomous System

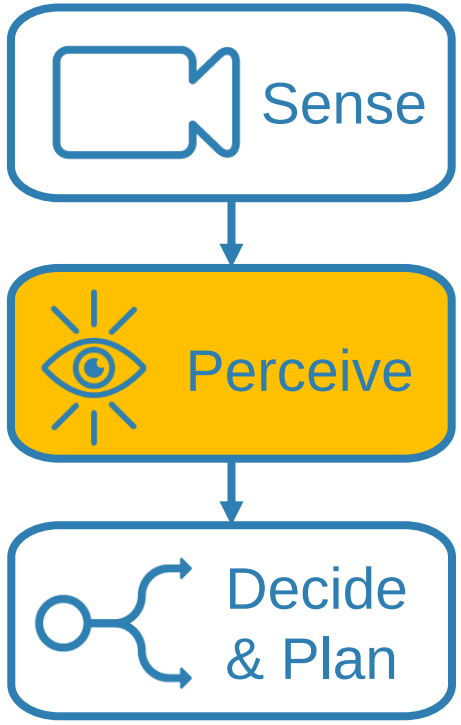
Sense

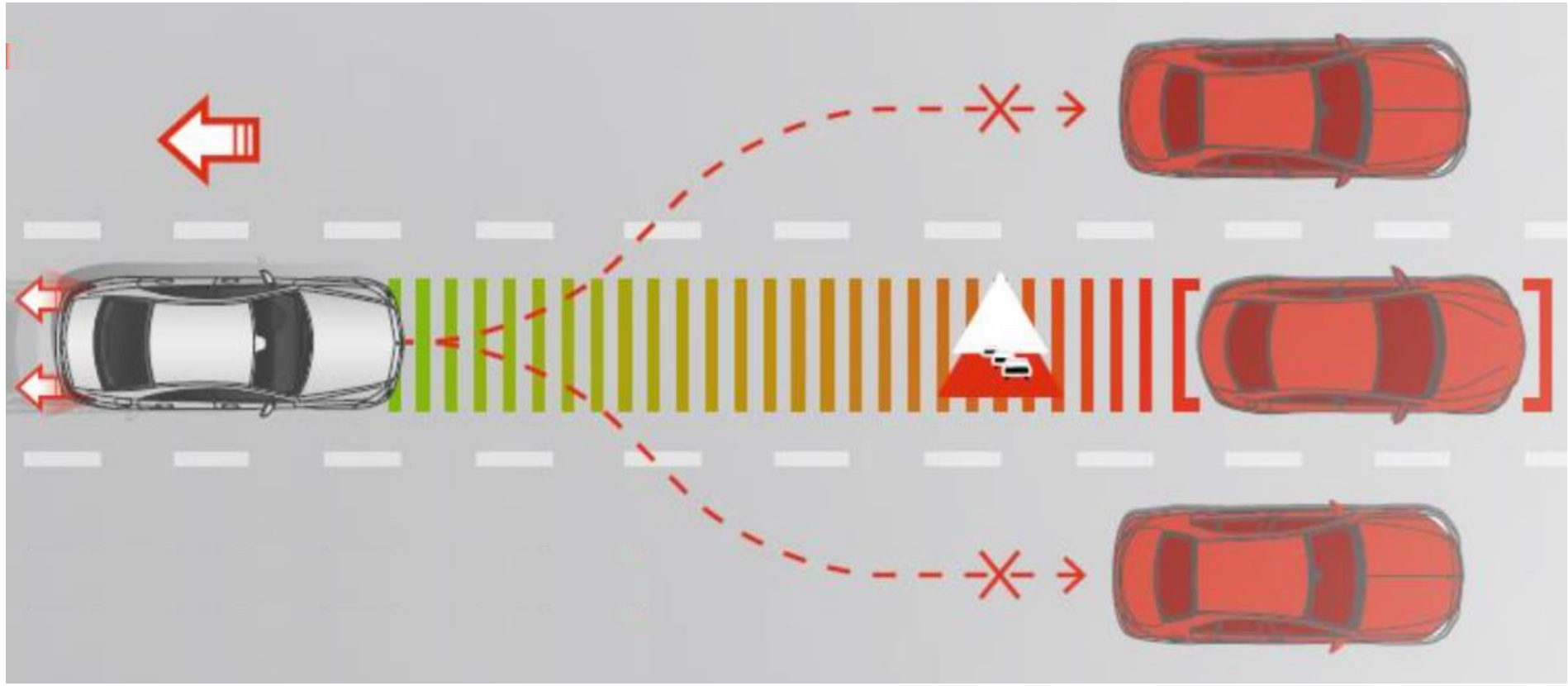
Capabilities of an Autonomous System



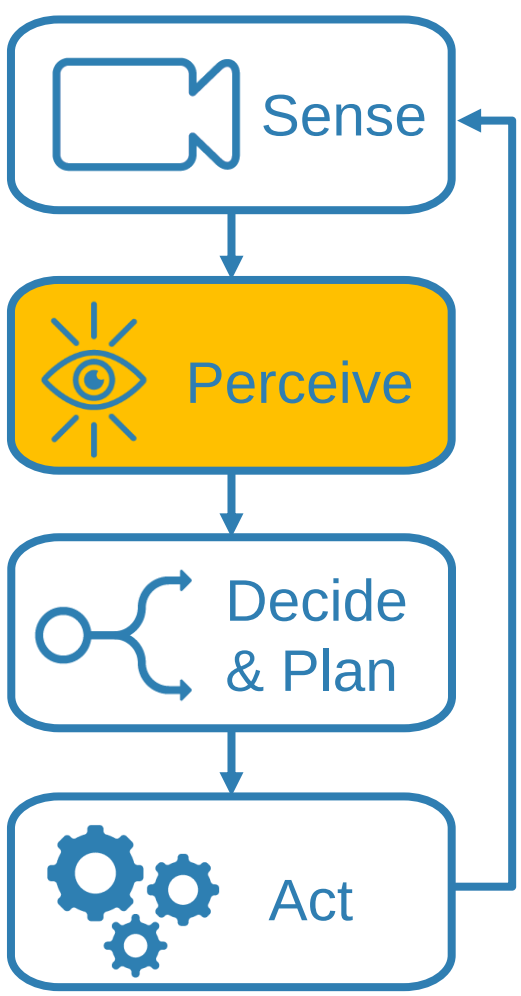


Capabilities of an Autonomous System





Capabilities of an Autonomous System



Agenda

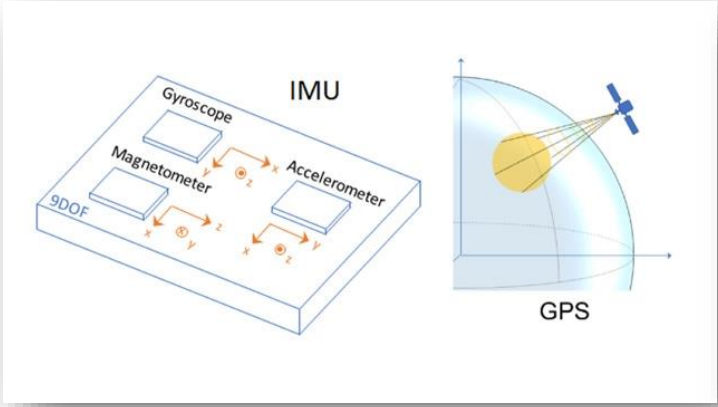
- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Agenda

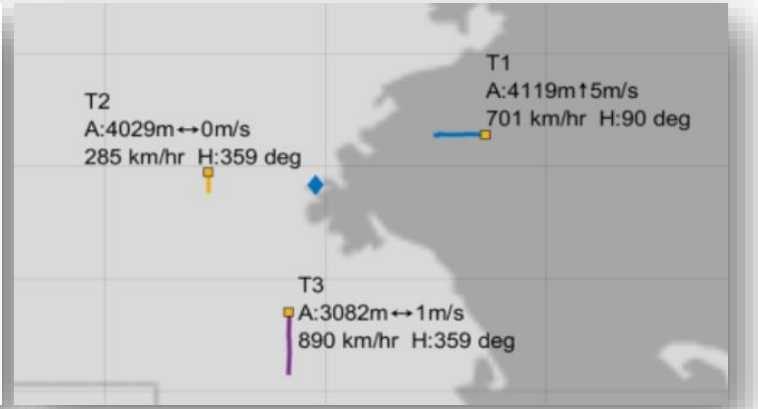
- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Sensor fusion and tracking is…

Self- awareness



Situational awareness



Accelerometer, Magnetometer, Gyro, GPS…

Radar, Camera, IR, Sonar, Lidar, …

Signal and Image Processing

Sensor Fusion and Tracking

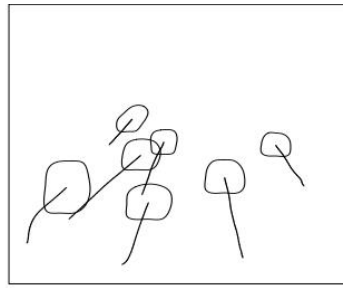
Control

Sensor Fusion and Tracking Toolbox NEW PRODUCT

Timeline of Technology Advances

Multi-object tracking

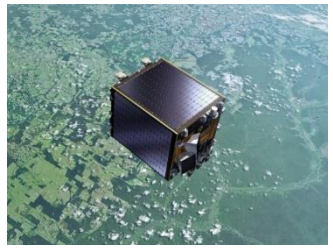
Air Traffic Control



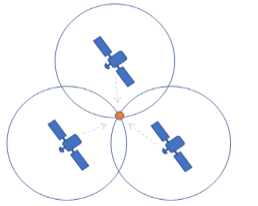
Computer Vision for Transportation

Multi-sensor Fusion for Autonomous Systems

Localization



Military



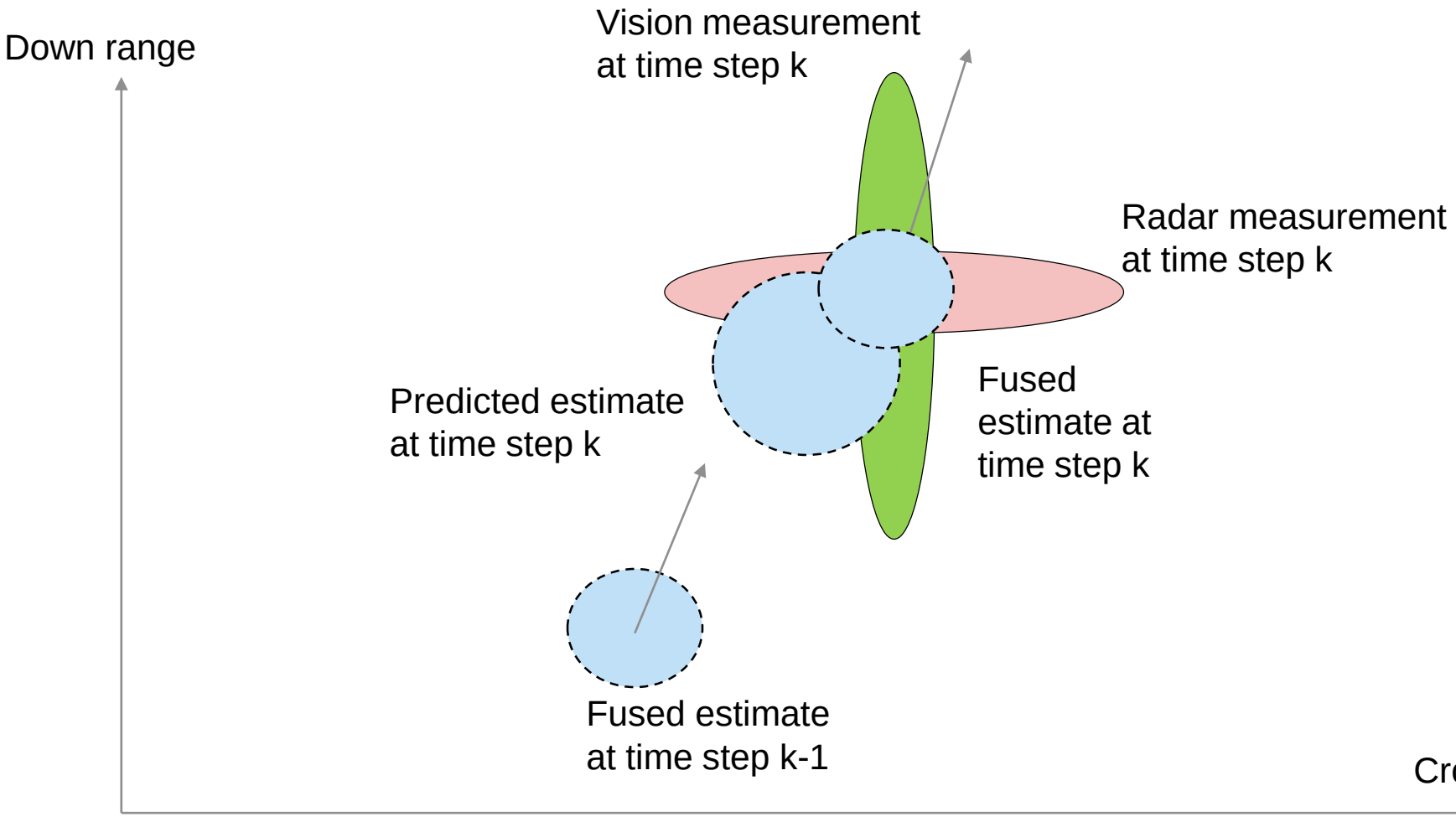
Commercial

Ubiquitous

Today

Timeline

Fusion Combines the Strengths of Each Sensor

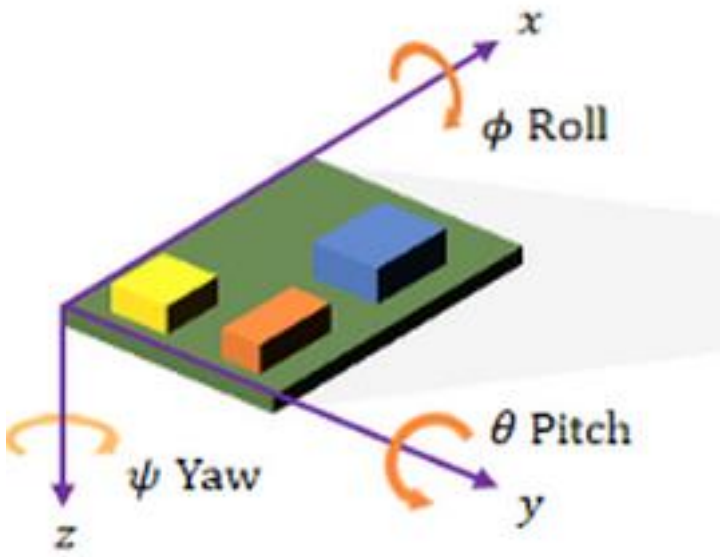


Legend

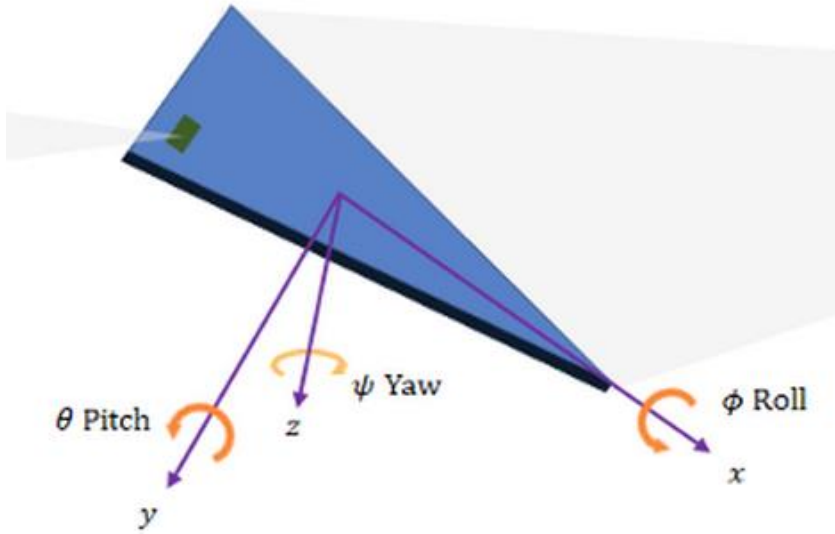
- Vision Measurement
- Radar measurement
- Track (fused estimate)

Ellipse represents uncertainty

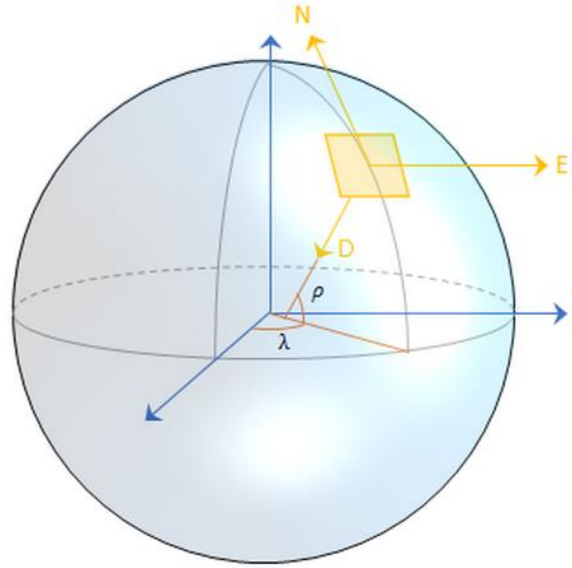
What is Localization?



Inertial Sensor



Attitude



Position

Agenda

- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Sensor Models for Sensor Fusion and Tracking

- **Altimeter** — altimeterSensor
- **GPS** — gpsSensor
- **IMU** — imuSensor
- **Infrared** — irSensor, irSignature
- **INS** — insSensor
- **Radar** — monostaticRadarSensor, radarEmitter, radarSensor, rcsSignature
- **Sonar** — sonarEmission, sonarEmitter, sonarSensor, tsSignature

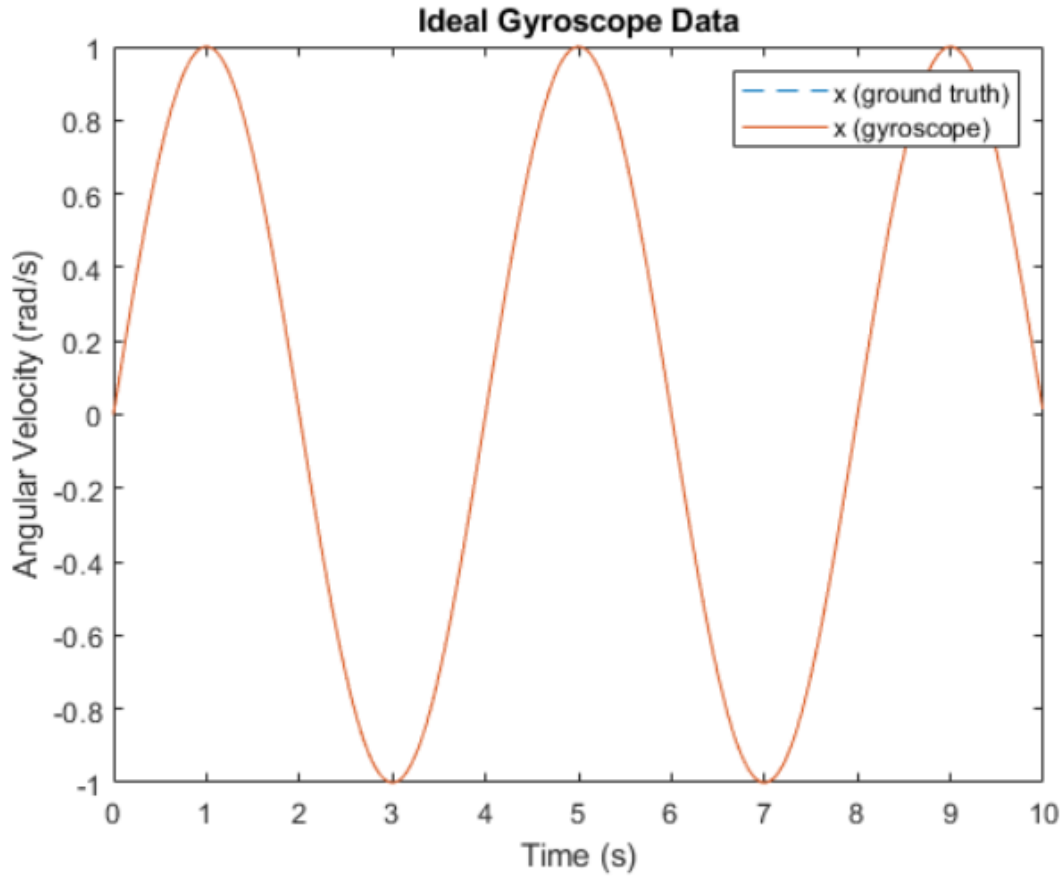
```
>> imu = imuSensor

imu =

  imuSensor with properties:

          IMUType: 'accel-gyro'
       SampleRate: 100
      Temperature: 25
    Accelerometer: [1×1 accelparams]
        Gyroscope: [1×1 gyroparams]
     RandomStream: 'Global stream'
```

```
>> imu.Accelerometer

ans =

  accelparams with properties:

    MeasurementRange: Inf          m/s²
          Resolution: 0            (m/s²)/LSB
        ConstantBias: [0 0 0]      m/s²
    AxesMisalignment: [0 0 0]      %

       NoiseDensity: [0 0 0]       (m/s²)/√Hz
    BiasInstability: [0 0 0]       m/s²
         RandomWalk: [0 0 0]       (m/s²)*√Hz

          TemperatureBias: [0 0 0]       (m/s²)/°C
    TemperatureScaleFactor: [0 0 0]      %/°C
```

```
>> imu.Gyroscope

ans =

  gyroparams with properties:

    MeasurementRange: Inf          rad/s
          Resolution: 0            (rad/s)/LSB
        ConstantBias: [0 0 0]      rad/s
    AxesMisalignment: [0 0 0]      %

       NoiseDensity: [0 0 0]       (rad/s)/√Hz
    BiasInstability: [0 0 0]       rad/s
         RandomWalk: [0 0 0]       (rad/s)*√Hz

          TemperatureBias: [0 0 0]       (rad/s)/°C
    TemperatureScaleFactor: [0 0 0]      %/°C
          AccelerationBias: [0 0 0]      (rad/s)/(m/s²)
```

Exploring gyro model in Sensor Fusion and Tracking Toolbox

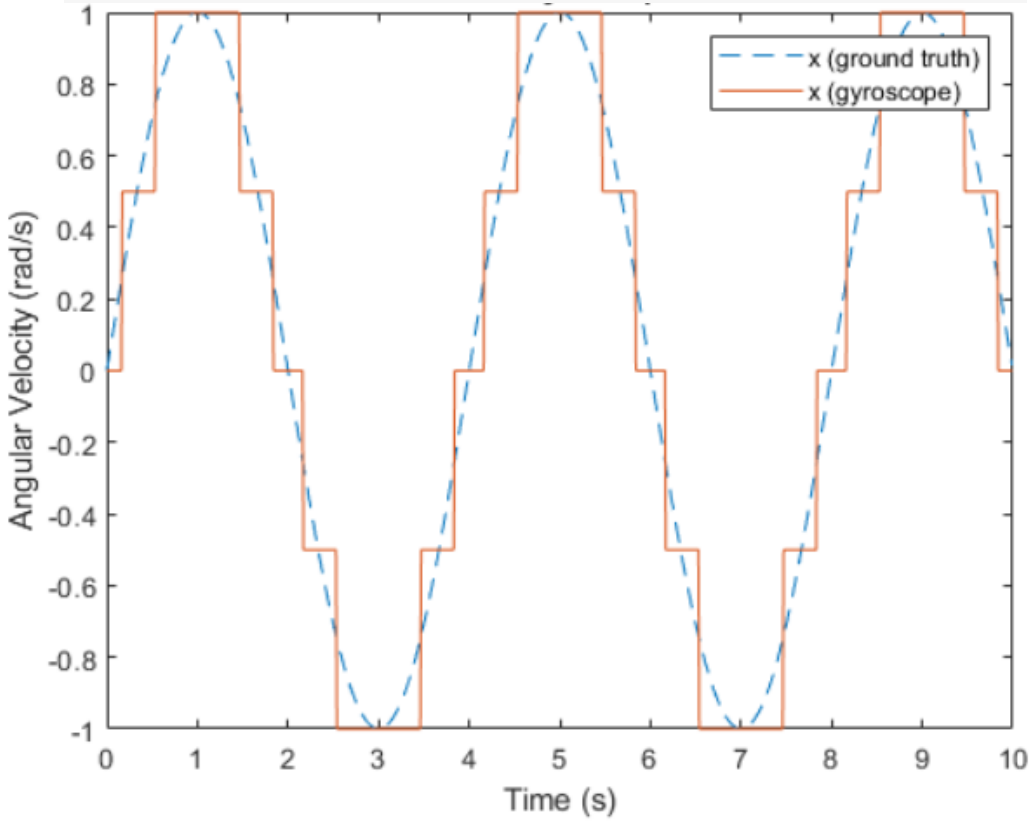
```
params = gyroparams

% Generate N samples at a sampling rate of Fs with a sinusoidal frequency
% of Fc.
N = 1000;
Fs = 100;
Fc = 0.25;

t = (0:(1/Fs):((N-1)/Fs)).';
acc = zeros(N, 3);
angvel = zeros(N, 3);
angvel(:,1) = sin(2*pi*Fc*t);

imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
[~, gyroData] = imu(acc, angvel);

figure
plot(t, angvel(:,1), '--', t, gyroData(:,1))
xlabel('Time (s)')
ylabel('Angular Velocity (rad/s)')
title('Ideal Gyroscope Data')
legend('x (ground truth)', 'x (gyroscope)')
```



Exploring gyro model in Sensor Fusion and Tracking Toolbox

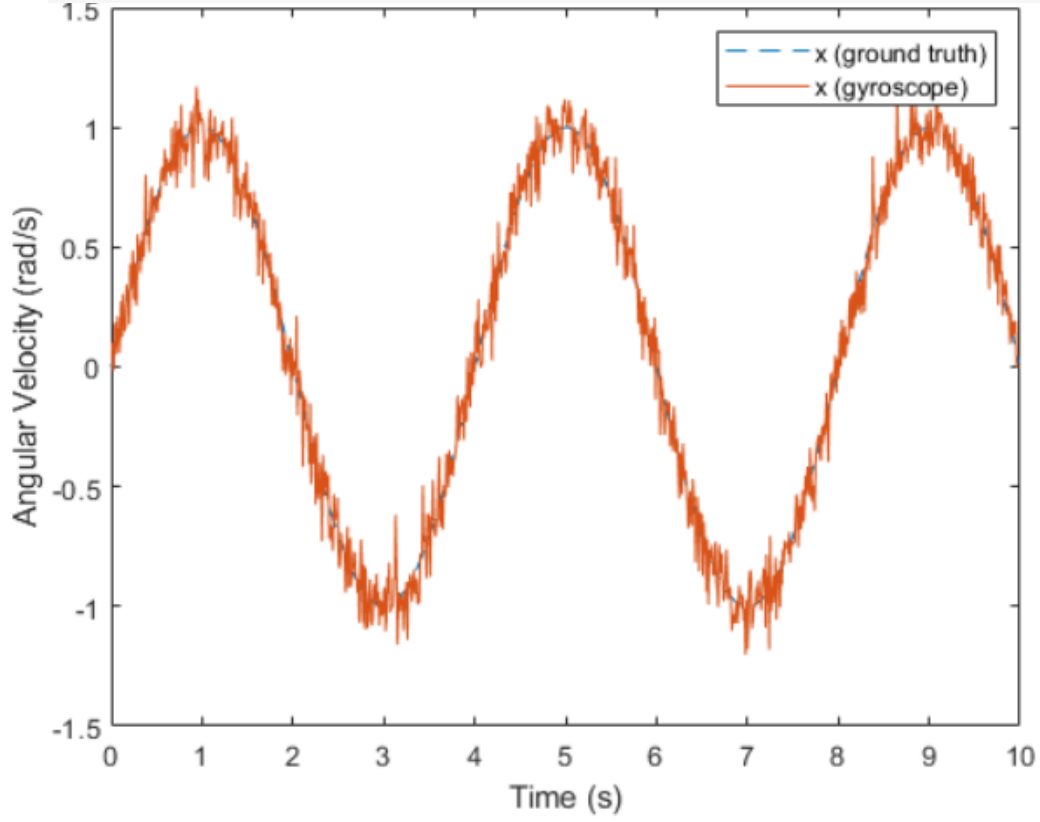
Imposing ADC Quantisation Effects

```
imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
imu.Gyroscope.Resolution = 0.5; % (rad/s)/LSB

[~, gyroData] = imu(acc, angvel);
```



Imposing White Noise

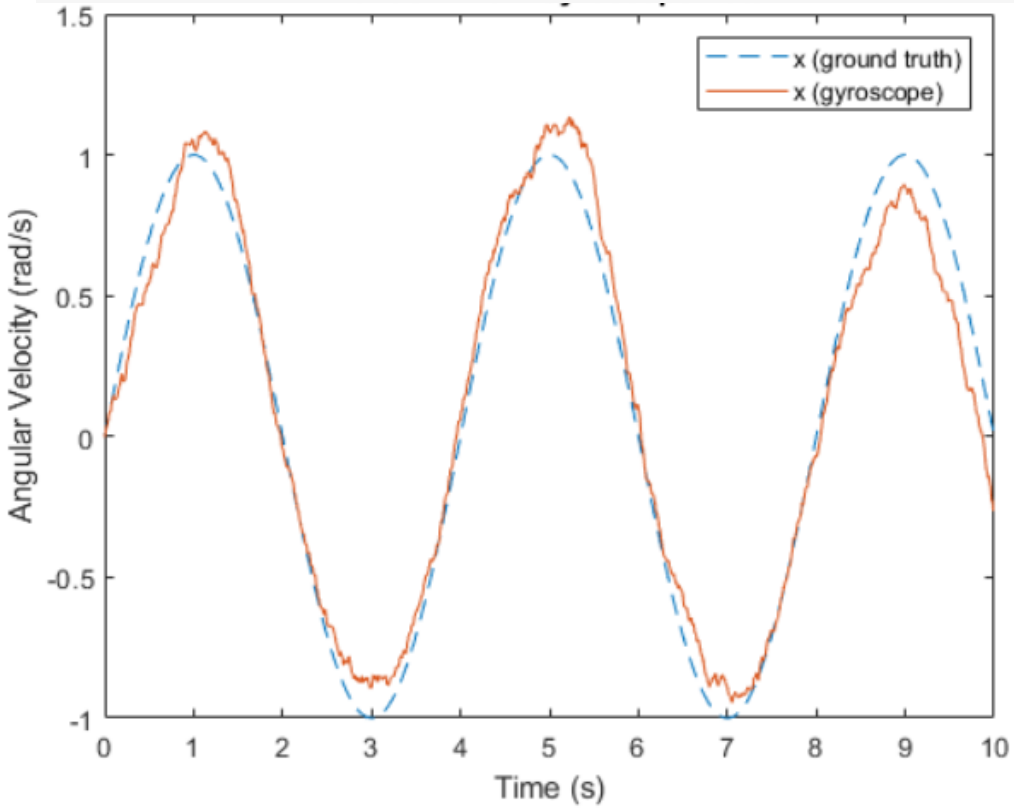
```
imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
imu.Gyroscope.NoiseDensity = 1.25e-2; % (rad/s)/sqrt(Hz)

[~, gyroData] = imu(acc, angvel);
```



Exploring gyro model in Sensor Fusion and Tracking Toolbox

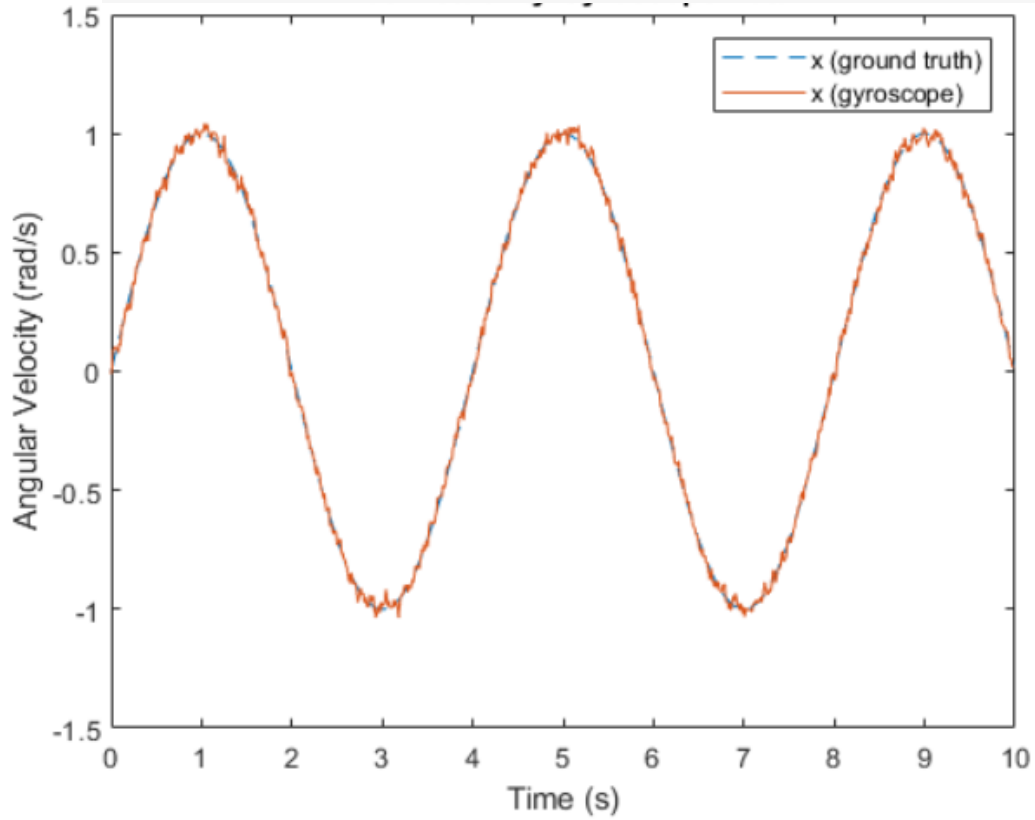
Imposing Brown Noise

```
imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
imu.Gyroscope.RandomWalk = 9.1e-2; % (rad/s)*sqrt(Hz)

[~, gyroData] = imu(acc, angvel);
```



Imposing Pink Noise

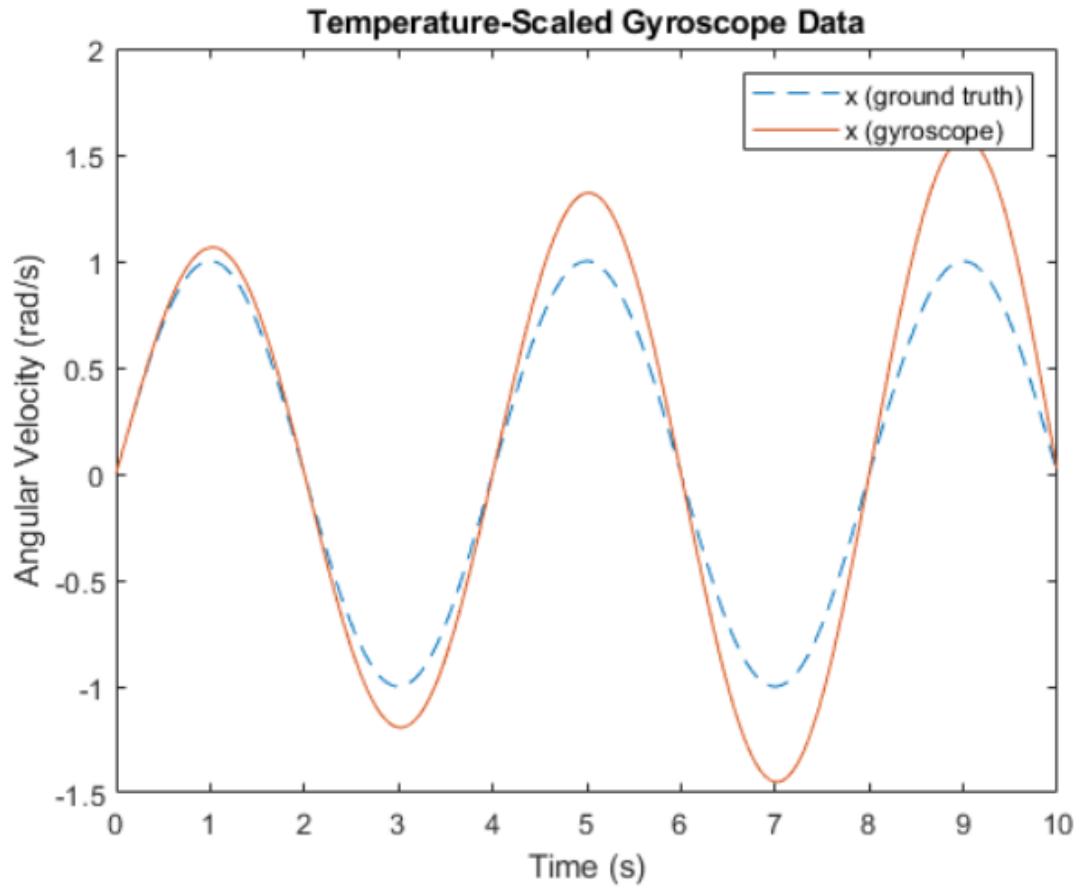
```
imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
imu.Gyroscope.BiasInstability = 2.0e-2; % rad/s

[~, gyroData] = imu(acc, angvel);
```



Exploring gyro model in Sensor Fusion and Tracking Toolbox

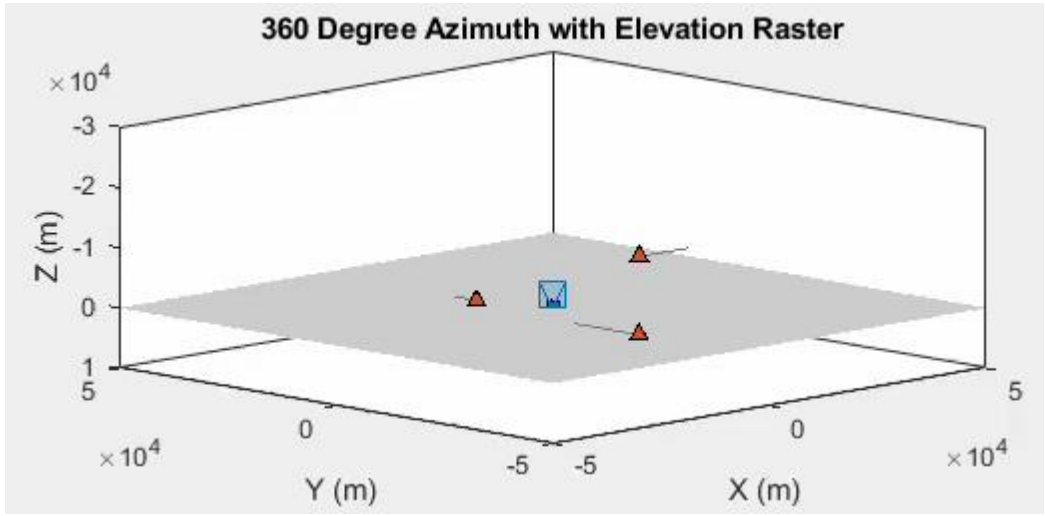
Imposing Temperature Scaled Bias

```
imu = imuSensor('SampleRate', Fs, 'Gyroscope', params);
imu.Gyroscope.TemperatureScaleFactor = 3.2; % %/(degrees C)

standardTemperature = 25; % degrees C
temperatureSlope = 2; % (degrees C)/s

temperature = temperatureSlope*t + standardTemperature;

gyroData = zeros(N, 3);
for i = 1:N
    imu.Temperature = temperature(i);
    [~, gyroData(i,:)] = imu(acc(i,:), angvel(i,:));
end
```

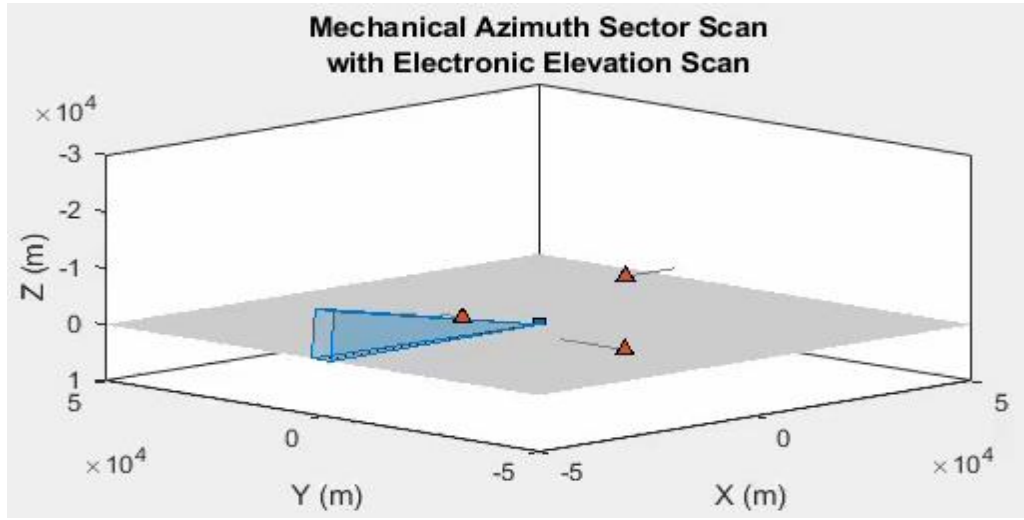


For more information, see example.

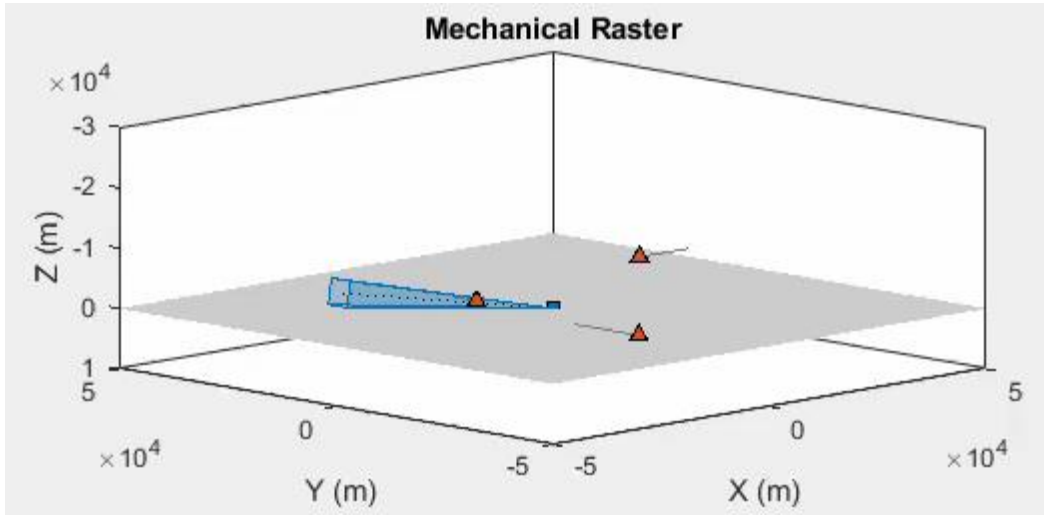
Levels of Fidelity: Radar Detections vs. I/Q Samples Simulation



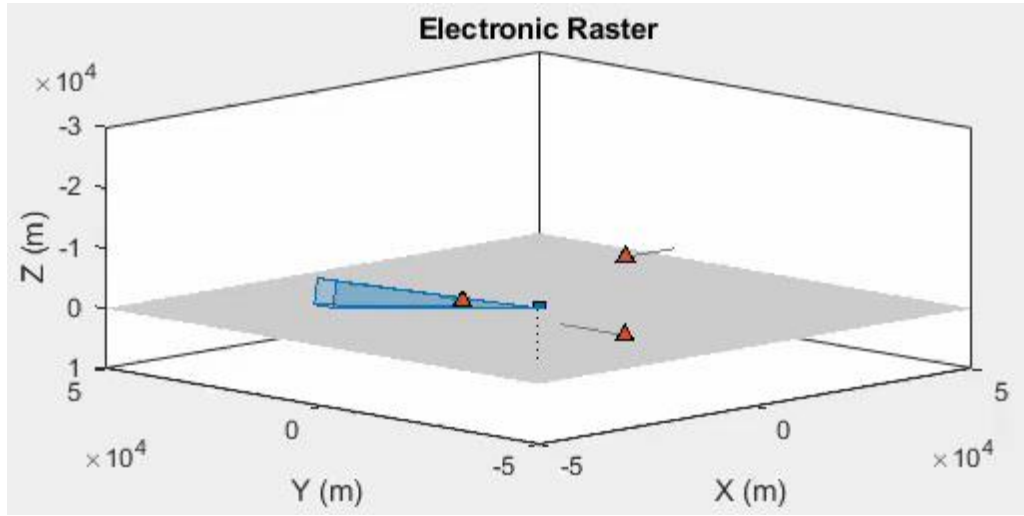

```
radar = monostaticRadarSensor('Rotator')
```




```
radar = monostaticRadarSensor('Sector')
```




```
radar = monostaticRadarSensor('Mechanical Raster')
```




```
radar = monostaticRadarSensor('Electronic Raster')
```

Generating Radar Detections in MATLAB

Sensor ID

```
radar = monostaticRadarSensor(1, 'UpdateRate',10, ... % In Hz
    'MountingLocation', [0 0 -30], 'FieldOfView', [1.5;45],...
    'MechanicalScanLimits', [-60 60; 0 0], 'HasElevation', true)
```

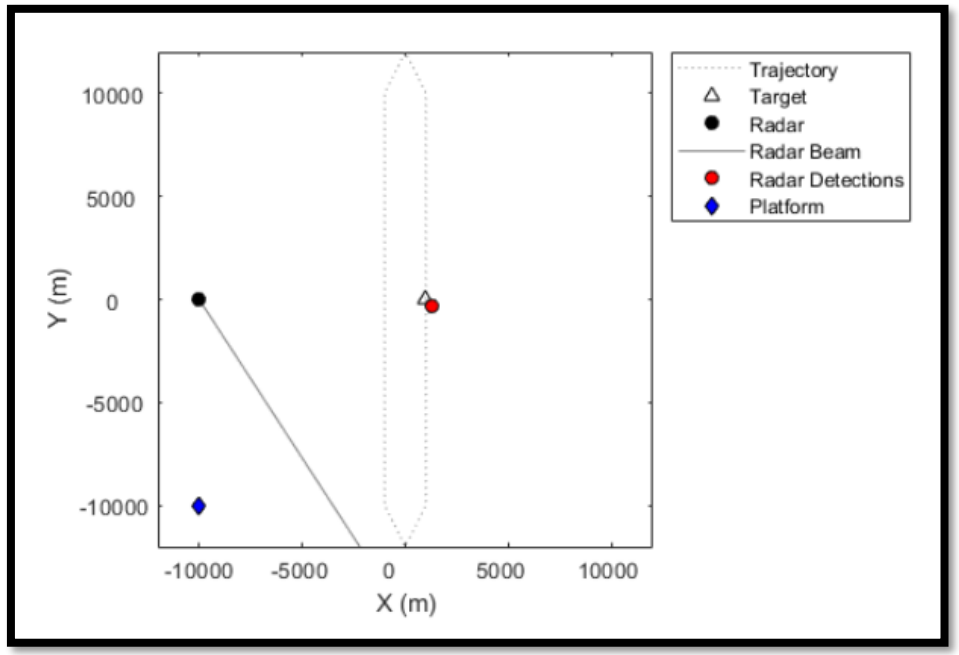
```
% Simulate and plot the radar detections
[detections, numDets] = radar(tgtPose, scene.SimulationTime);
```

Detections (time, measurement, etc…)

Target positions

Simulation time

```
radar =

  monostaticRadarSensor with properties:

                SensorIndex: 1
                 UpdateRate: 10
                   ScanMode: 'Mechanical'

            MountingLocation: [0 0 -30]
              MountingAngles: [0 0 0]

          MaxUnambiguousRange: 100000
    MaxUnambiguousRadialSpeed: 200
                  FieldOfView: [2x1 double]
       MaxMechanicalScanRate: [2x1 double]
        MechanicalScanLimits: [2x2 double]
             MechanicalAngle: [2x1 double]
                   LookAngle: [2x1 double]

        DetectionProbability: 0.9000
              FalseAlarmRate: 1.0000e-06

  Use get to show all properties
```



For more information, see example here

MATLAB EXPO 2019

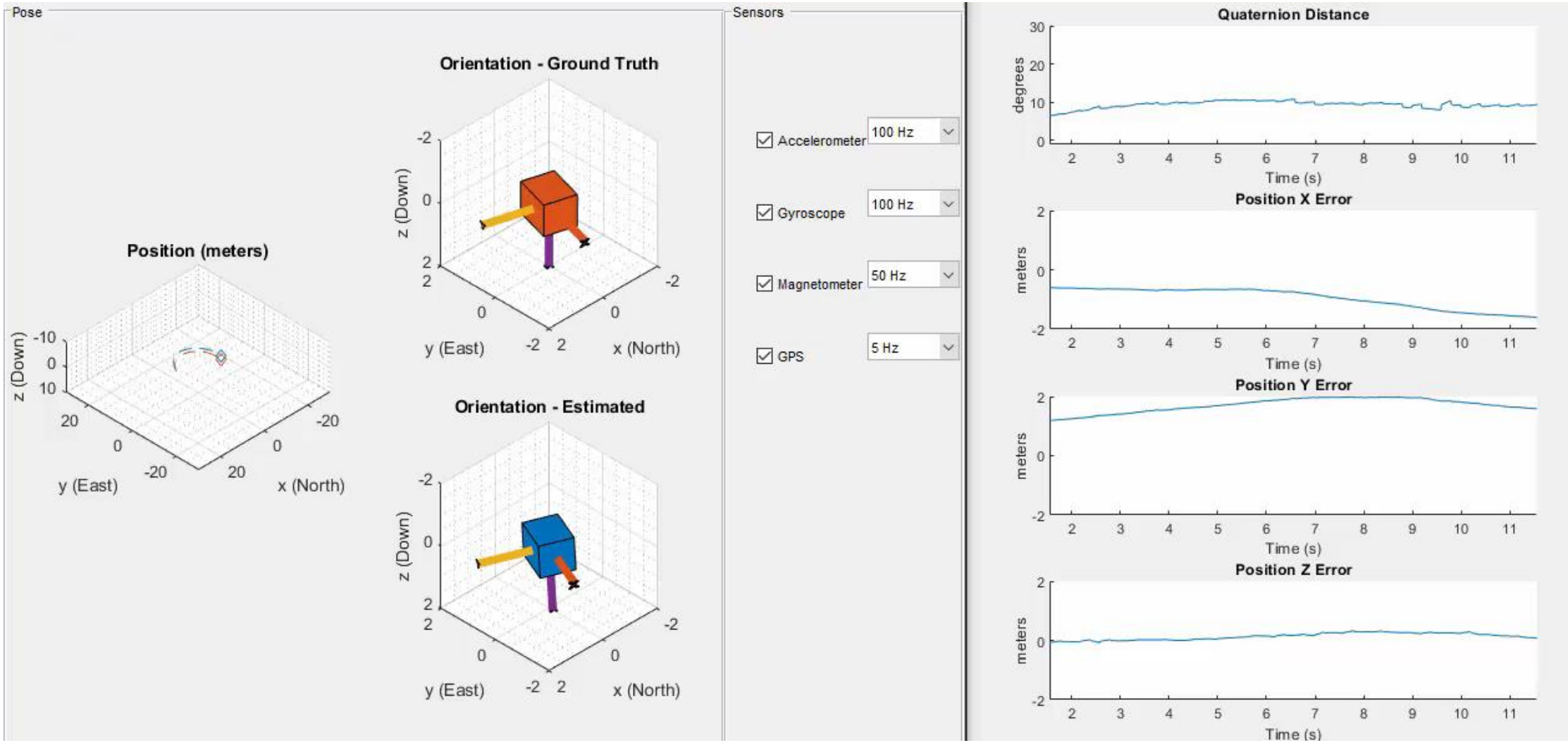
TBC – we will see how to build this as a scenario later…

Fusing Sensor Data Improves Localization

Ground truth vs. Estimate

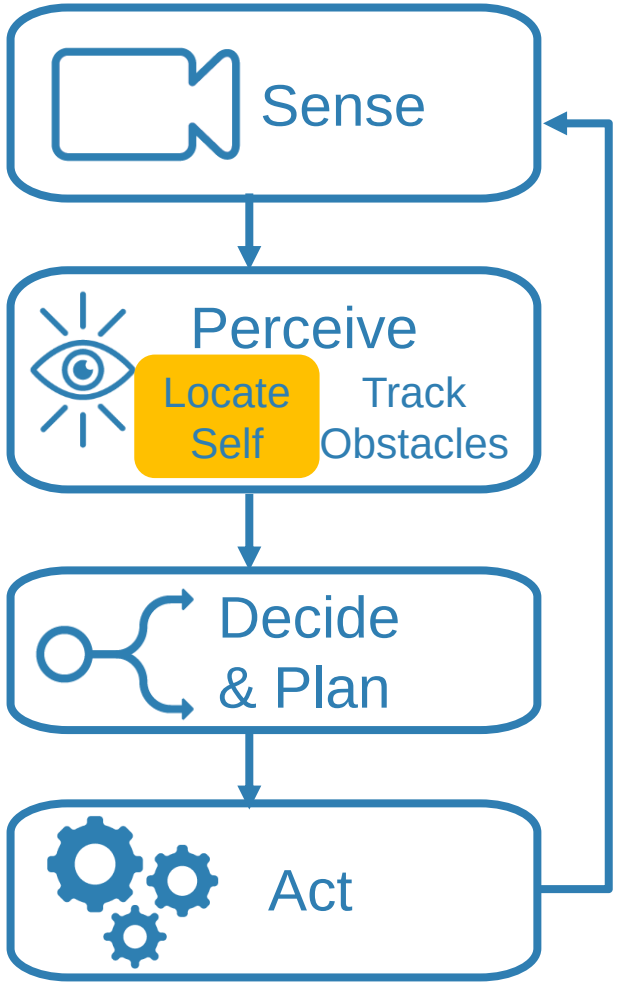
Sensors

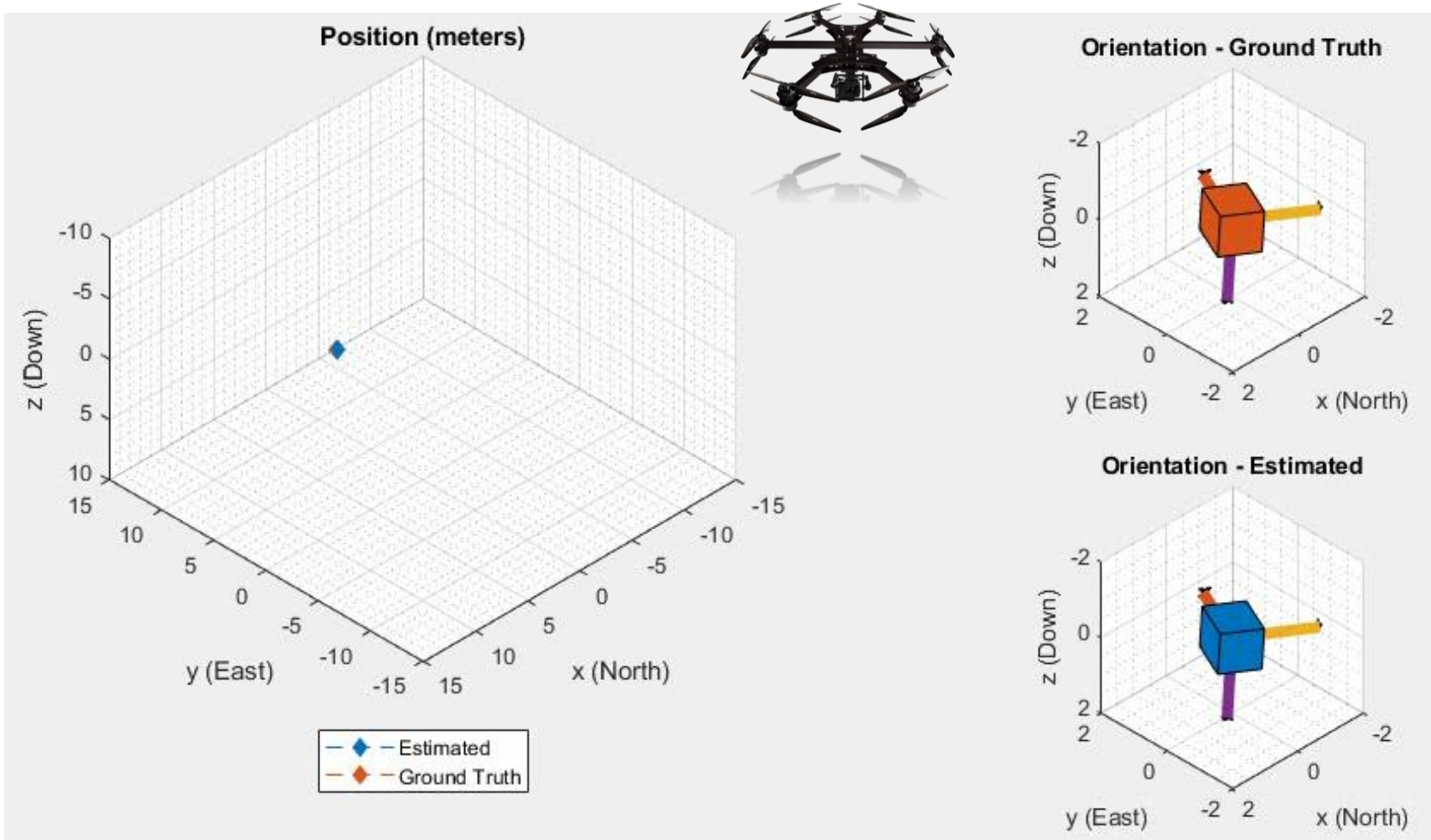
Error Measurements



Example here

Fuse IMU & GPS for Self-Localization of a UAV

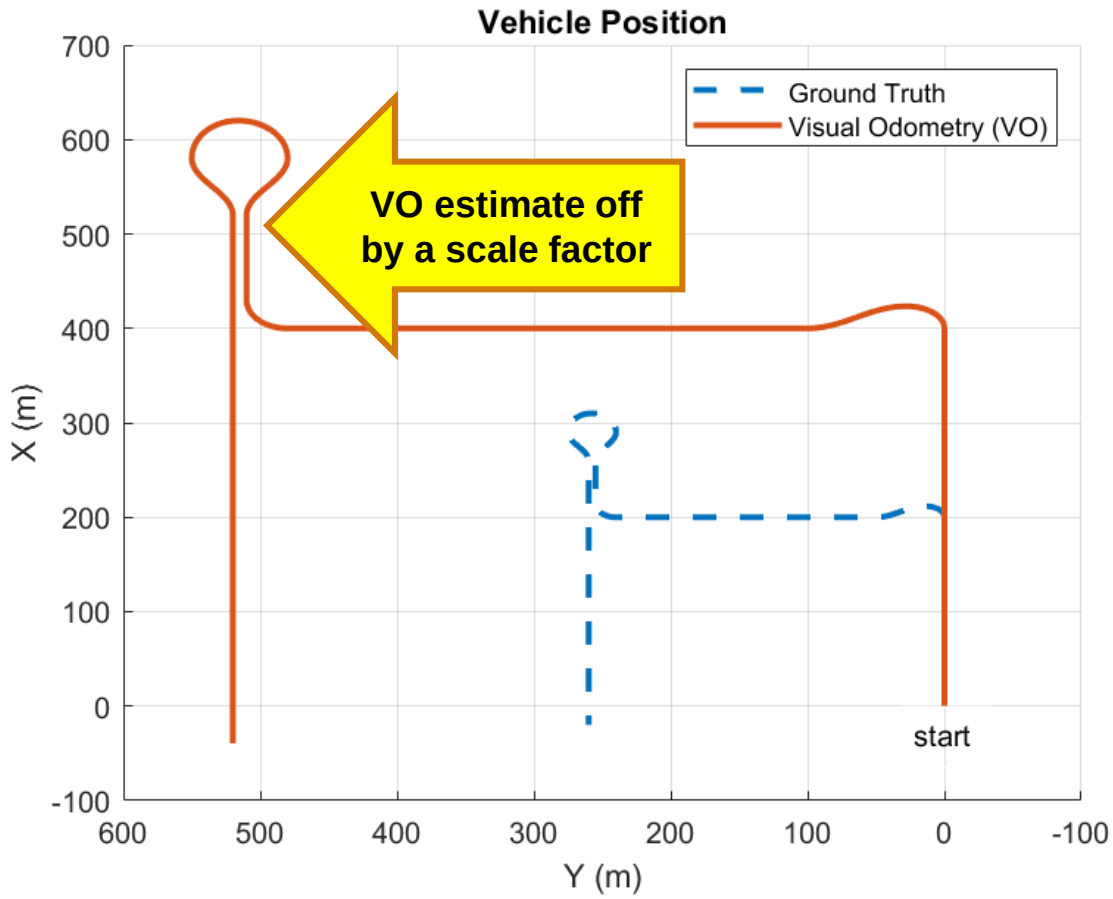


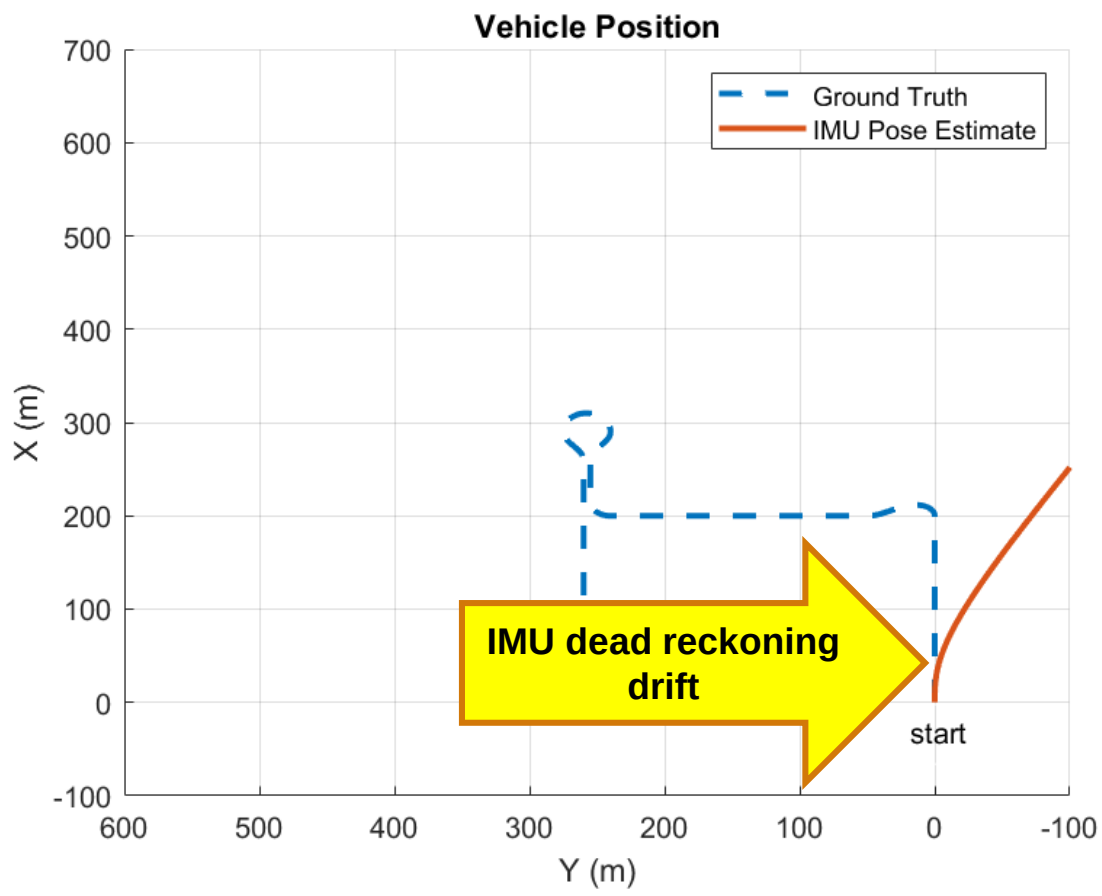


MATLAB EXPO 2019

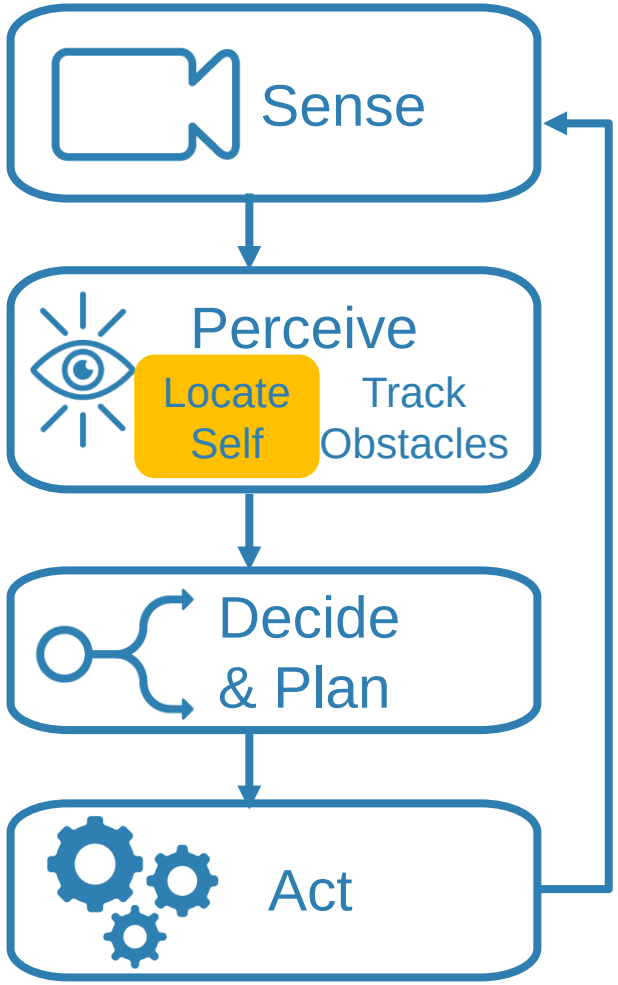
Example here

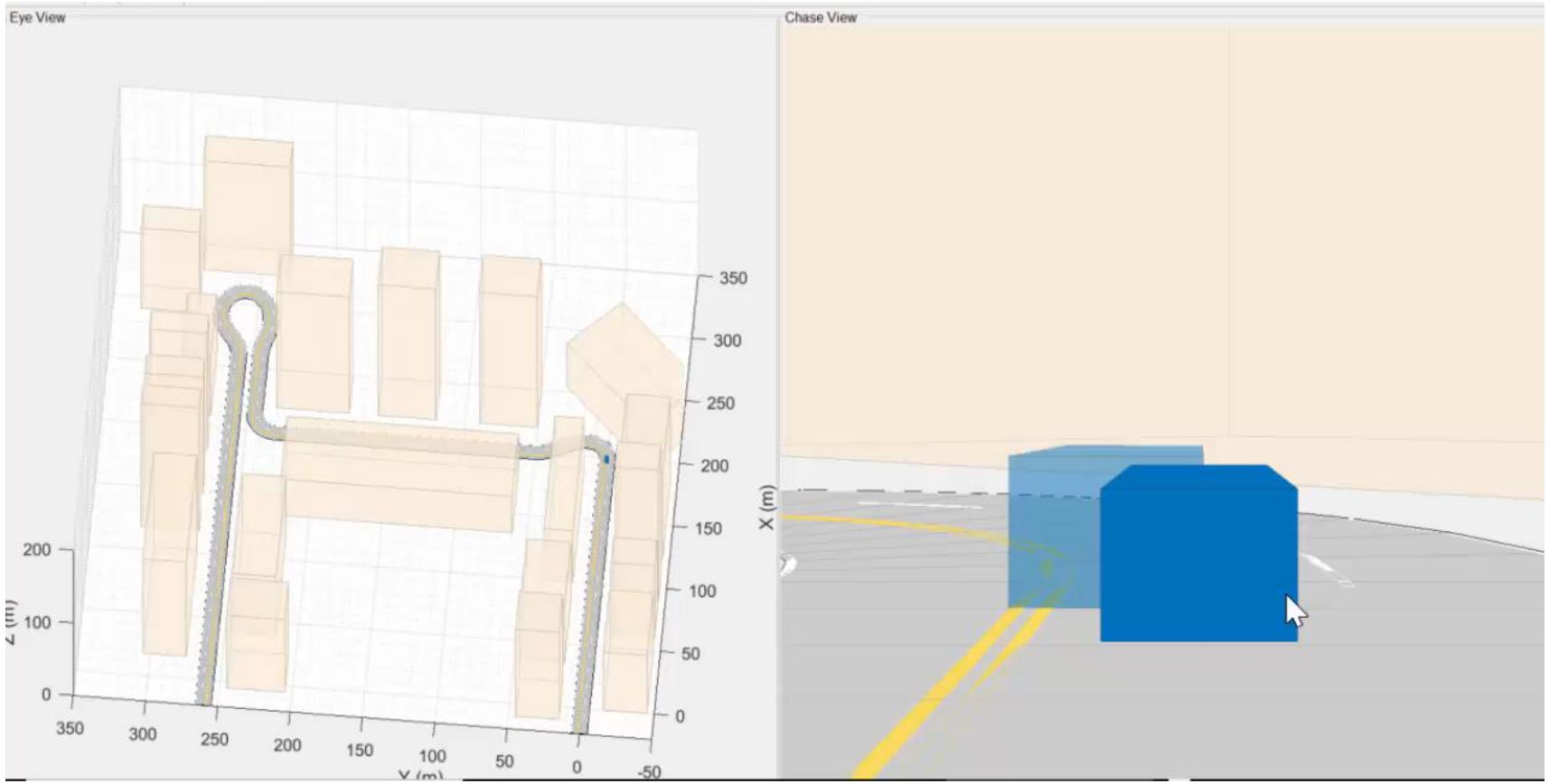
Fuse IMU & Odometry for Self-Localization in GPS-Denied Areas



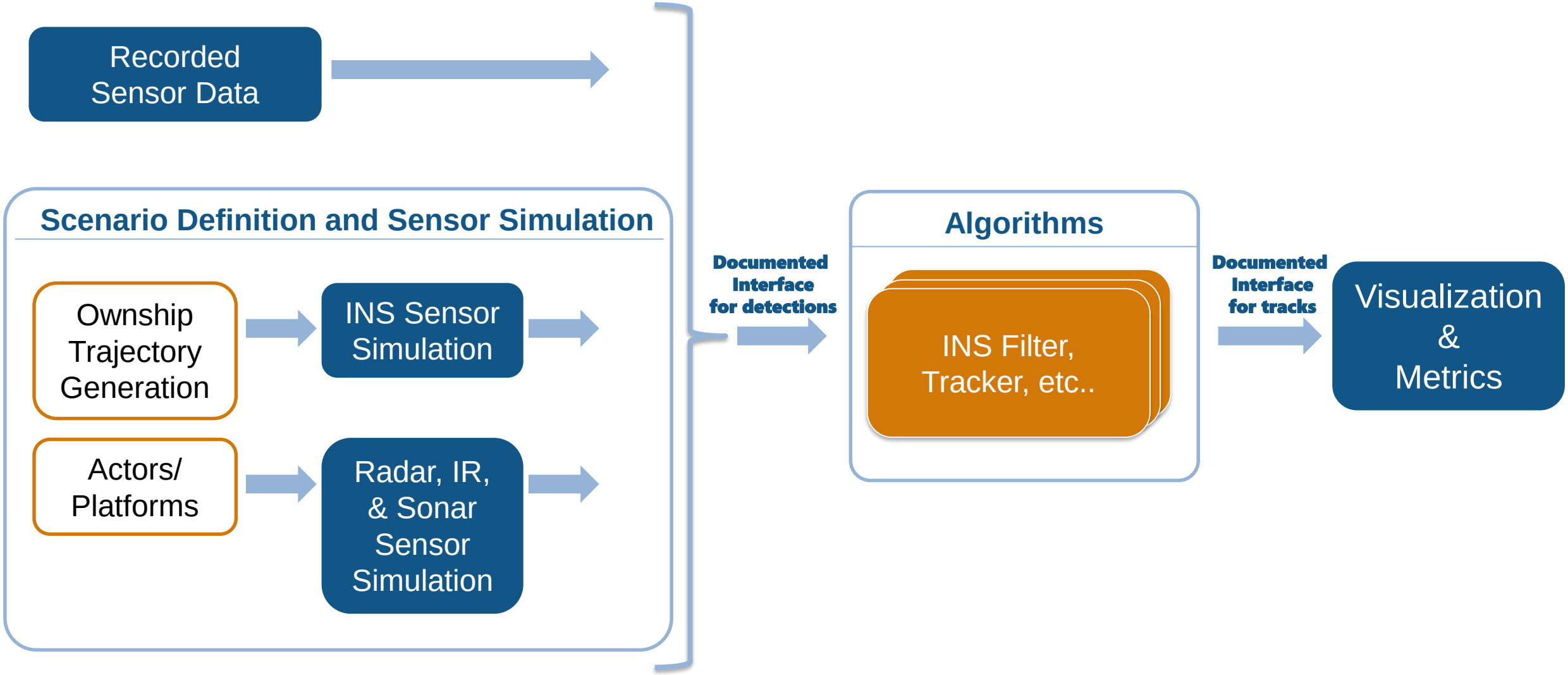


Fuse IMU & Odometry for Self-Localization in GPS-Denied Areas





Flexible Workflows Ease Adoption: Wholesale or Piecemeal

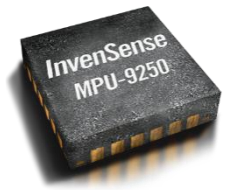


Stream Data to MATLAB from IMUs Connected to Arduino

MathWorks R2019a

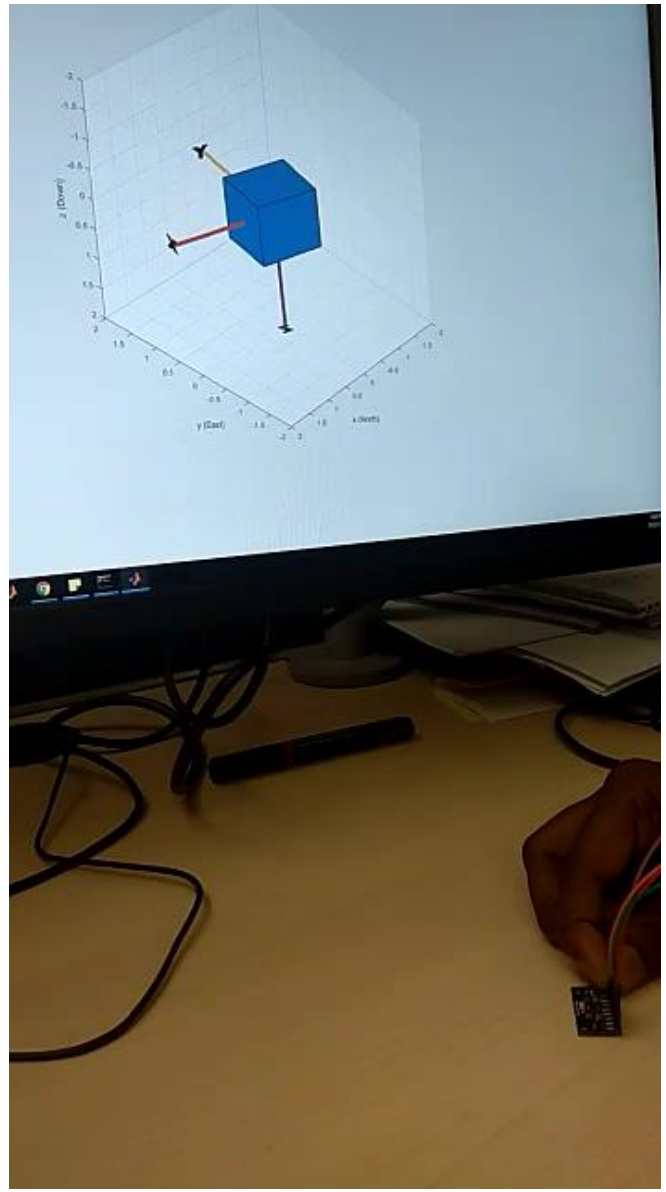
MEMS Devices

- 9-axis (Gyro + Accelerometer + Compass)



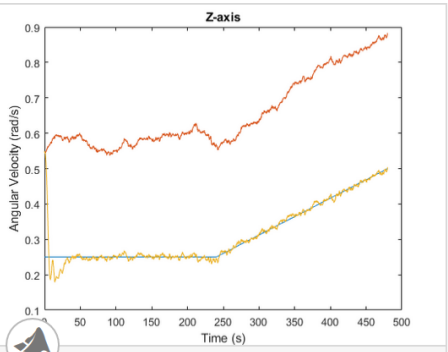
- 6 axis (Gyro + Accelerometer)

- Up to 200 Hz sampling rate



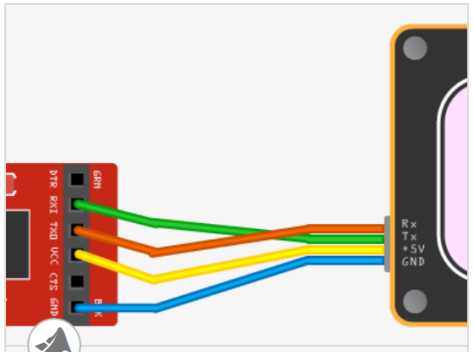
New Hardware and Multisensor Positioning Examples

R2019b



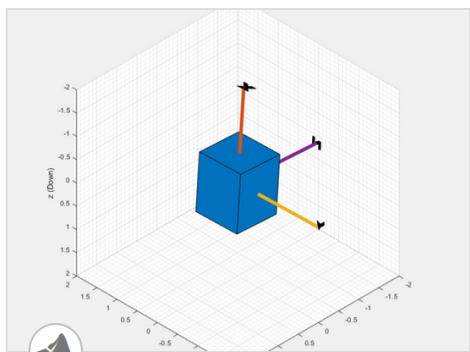
Remove Bias from Angular Velocity Measurement

Remove gyroscope bias from an IMU using imufilter.



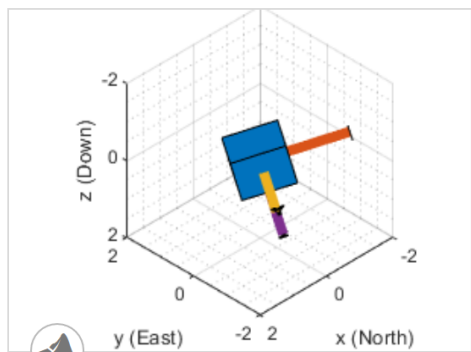
Read and Parse NMEA Data Directly From GPS Receiver

Read the data from a GPS receiver connected to a computer, and parse the National Marine Electronics Association(NMEA) data.



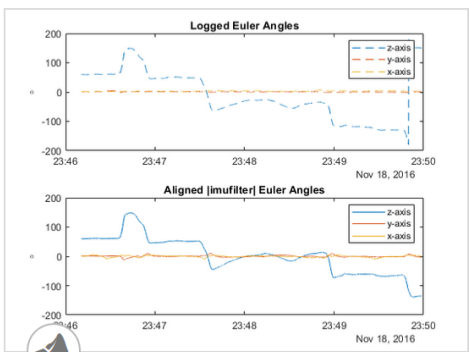
Estimating Orientation Using Inertial Sensor Fusion and MPU-9250

Get data from an InvenSense MPU-9250 IMU sensor and to use the 6-axis and 9-axis fusion algorithms in the sensor data to



Estimate Orientation Through Inertial Sensor Fusion

Use 6-axis and 9-axis fusion algorithms to compute orientation. There are several algorithms to compute orientation from inertial



Logged Sensor Data Alignment for Orientation Estimation

Align and preprocess logged sensor data. This allows the fusion filters to perform orientation estimation as expected. The logged data was

Agenda

- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Building a Simulation Scenario

```
scene = trackingScenario('UpdateRate', 10)
```

```
scene =

  trackingScenario with properties:

       UpdateRate: 10
         StopTime: Inf
   SimulationTime: 0
        IsRunning: 1
        Platforms: {}
```

```
plat = platform(scene)
```

```
plat =

  Platform with properties:

      PlatformID: 1
         ClassID: 0
      Dimensions: [1x1 struct]
      Trajectory: [1x1 kinematicTrajectory]
   PoseEstimator: [1x1 insSensor]
        Emitters: {}
         Sensors: {}
      Signatures: {[1x1 rcsSignature]  [1x1 irSignature]  [1x1 tsSignature]}
```

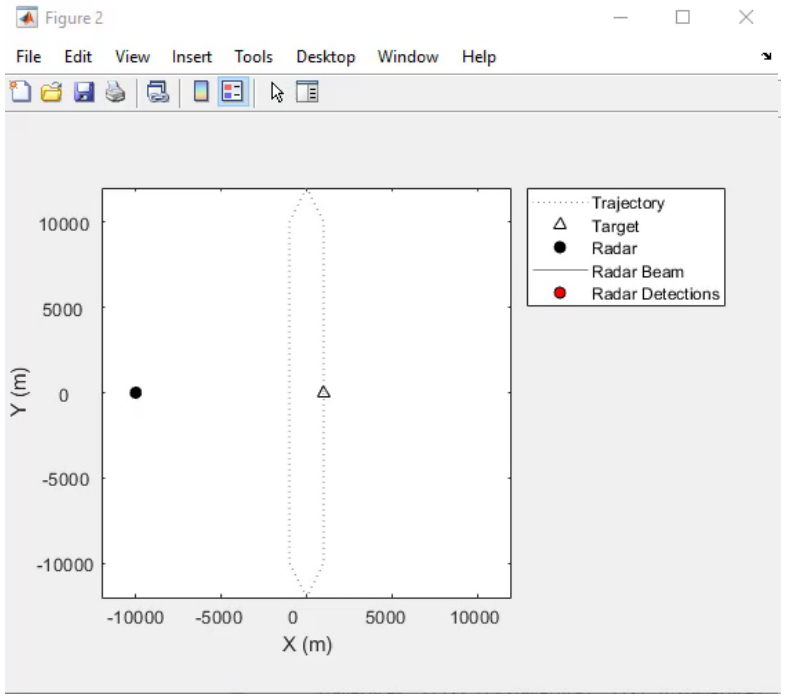
```
h = -3; % Altitude above ground, in Kilometers
waypoints = 1e3*   [  1    1    0   -1    -1   0     1    1
    0   10   12   10   -10  -12  -10    0
    h    h    h    h     h    h     h    h]';
course =           [ 90   90  180   270  270    0   90   90]';
timeOfArrival = 60*[  0   1.5  1.8  2.1  5.1  5.4  5.7  7.2]';
traj = waypointTrajectory('Waypoints', waypoints, 'TimeOfArrival', timeOfArrival);
plat.Trajectory = traj;
```

```
tp = theaterPlot('XLimits', [-12 12]*1e3, 'Ylimits', [-12 12]*1e3, 'ZLimits', [1.1*h 0]*1e3);
pPlotter = trajectoryPlotter(tp, 'DisplayName', 'Trajectory');
plotTrajectory(pPlotter, {waypoints})
aPlotter = platformPlotter(tp, 'DisplayName', 'Target');
```

```
while advance(scene)
    p = pose(plat,'true');
    plotPlatform(aPlotter, p.Position);
end
```

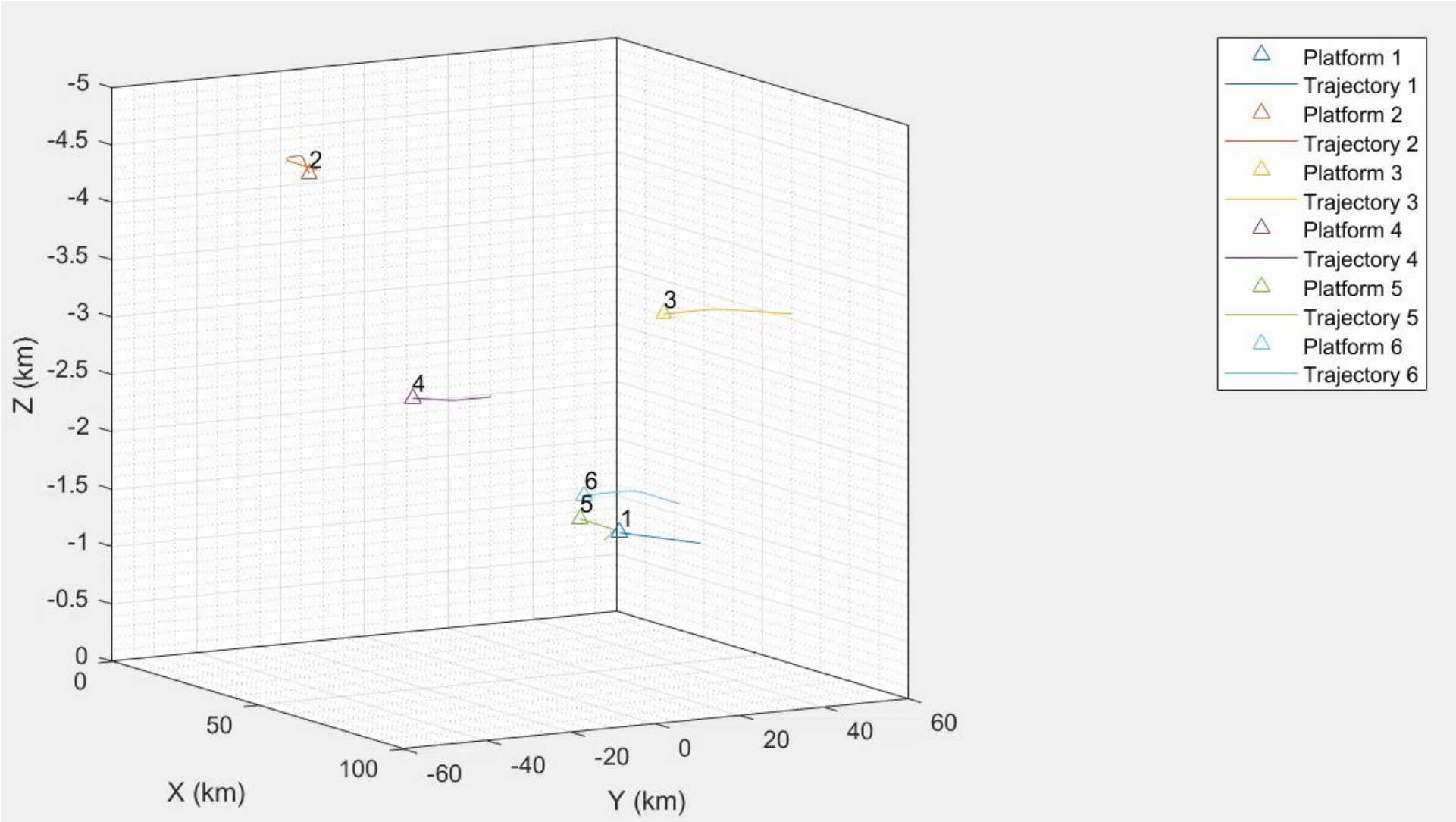
```
tower = platform(scene);
tower.Trajectory.Position = [-10000 0 0];
radar = monostaticRadarSensor(1, 'UpdateRate',10, ... % In Hz
    'MountingLocation', [0 0 -30], 'FieldOfView', [1.5;45],...
    'MechanicalScanLimits', [-60 60; 0 0], 'HasElevation', true)
radarLocation = tower.Trajectory.Position + radar.MountingLocation;
rPlotter = platformPlotter(tp, 'DisplayName', 'Radar', 'Marker', 'o', 'MarkerFaceColor', [0 0 0]);
plotPlatform(rPlotter, tower.Trajectory.Position);
```

```
beamPlotter = trajectoryPlotter(tp, 'DisplayName', 'Radar Beam', 'LineStyle', '-');
detsPlotter = detectionPlotter(tp, 'DisplayName', 'Radar Detections', 'Marker', 'o', 'MarkerFaceColor', [1 0 0]);
restart(scene);
while advance(scene)
    % Move the platform
    p = pose(plat,'true');
    plotPlatform(aPlotter, p.Position);

    % Get the platform pose in the coordinates of the tower
    tgtPose = targetPoses(tower);

    % Simulate and plot the radar detections
    [detections, numDets] = radar(tgtPose, scene.SimulationTime);
    detPos = zeros(numDets,3);
    for i=1:numDets
        detPos(i,:) = radarLocation + detections{i}.Measurement';
    end
    if ~isempty(detPos)
        plotDetection(detsPlotter, detPos);
    end

    % Get and plot the radar beam in the horizontal plane
    angle = radar.MechanicalAngle(1);
    beamPosition = radarLocation + radar.MaxUnambiguousRange*[0 0 0; cosd(angle) sind(angle) 0];
    plotTrajectory(beamPlotter, {beamPosition})
end
```



Example here

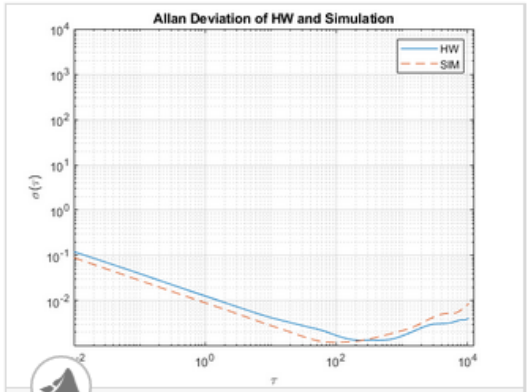
Test Tracker Performance on Pre-Built Benchmark Trajectories



Reference

W.D. Blair, G. A. Watson, T. Kirubarajan, Y. Bar-Shalom, "Benchmark for Radar Allocation and Tracking in ECM." Aerospace and Electronic Systems IEEE Trans on, vol. 34. no. 4. 1998

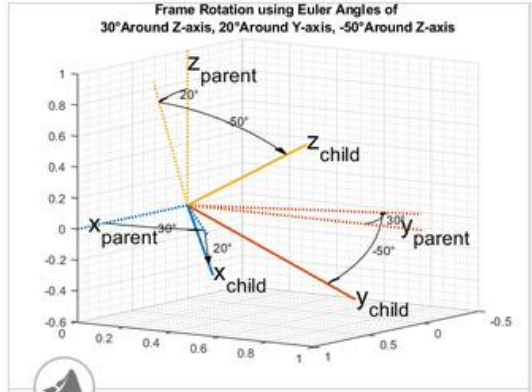
To go further on localization, see also



Inertial Sensor Noise Analysis Using Allan Variance

Use the Allan variance to determine noise parameters of a MEMS gyroscope. These parameters can be used to model the gyroscope in

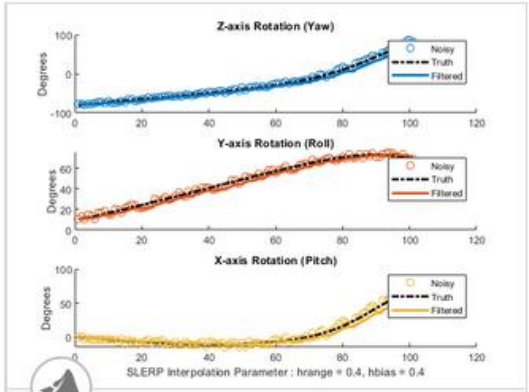
R2018b Open Script



Rotations, Orientation and Quaternions

Reviews concepts in three-dimensional rotations and how quaternions are used to describe orientation and rotations.

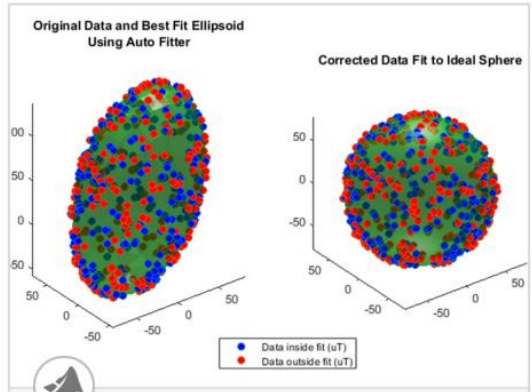
R2018b Open Script



Lowpass Filter Orientation Using Quaternion SLERP

Use spherical linear interpolation (SLERP) to create sequences of quaternions and lowpass filter noisy trajectories. SLERP is a commonly

R2018b Open Script



Magnetometer Calibration

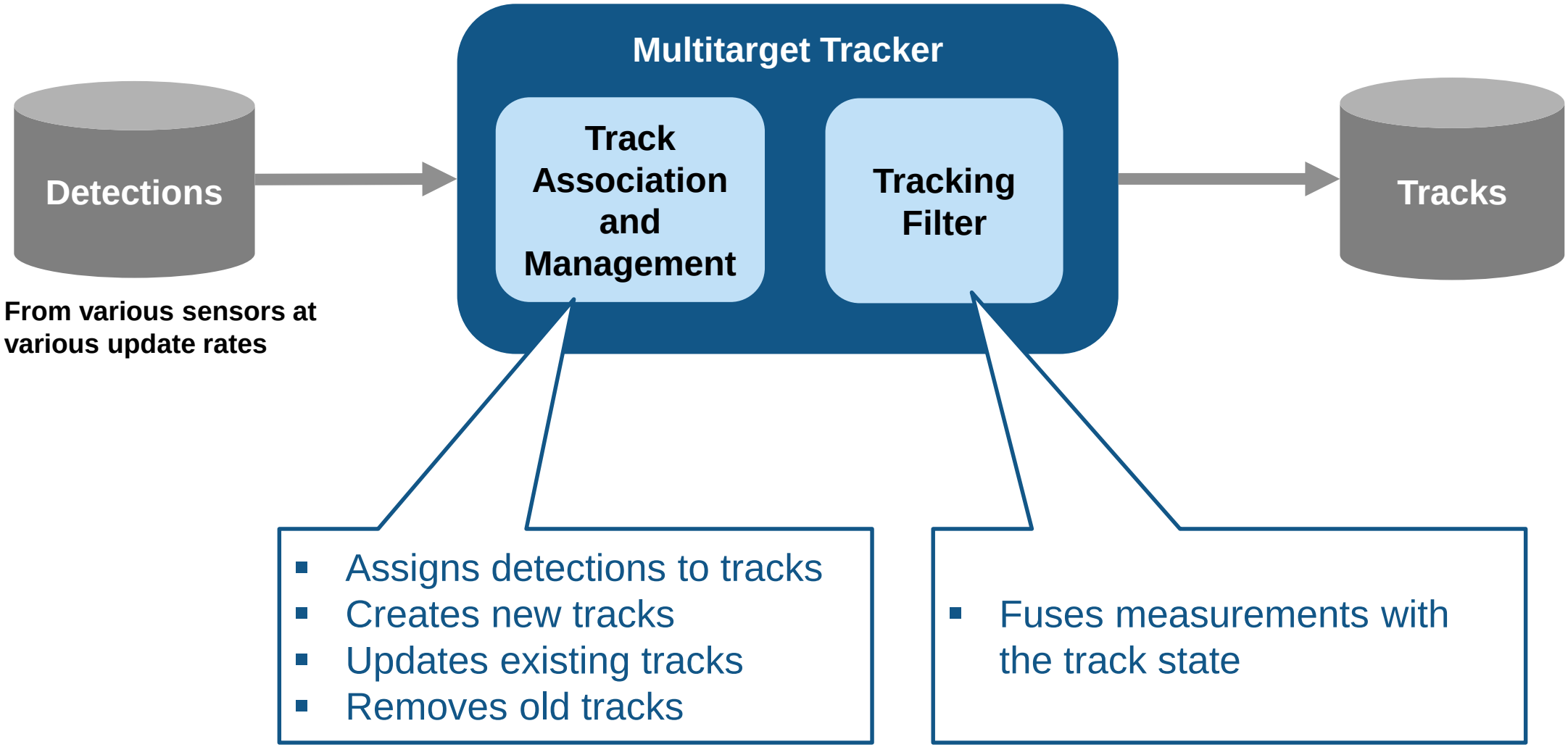
Magnetometers detect magnetic field strength along a sensor's X,Y and Z axes. Accurate magnetic field measurements are essential for

R2019a Open Script

Agenda

- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

A Multi-object Tracker is More than a Kalman Filter



Tracking Algorithm Development Workflow

Recorded Sensor Data

Scenario Definition and Sensor Simulation

Ownship Trajectory Generation → INS Sensor Simulation →

Actors/ Platforms → Radar, IR, & Sonar Sensor Simulation →

objectDetection

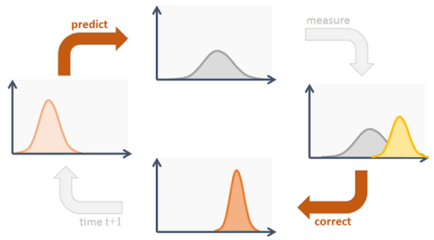
Tracking Algorithms

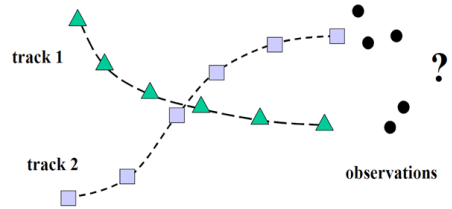
GNN, MHT, etc..

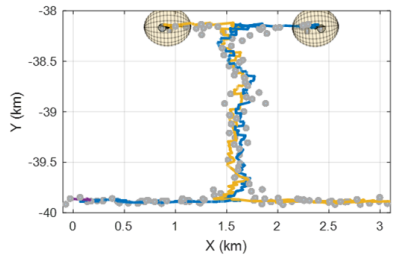
tracks

Visualization & Metrics

A rich library of tracking algorithms







Filters

- Alpha Beta filter
- Kalman filters
  - Linear, EKF, UKF, CKF, MSCEKF
- Particle filter
- Multiple models
  - GSF, IMM

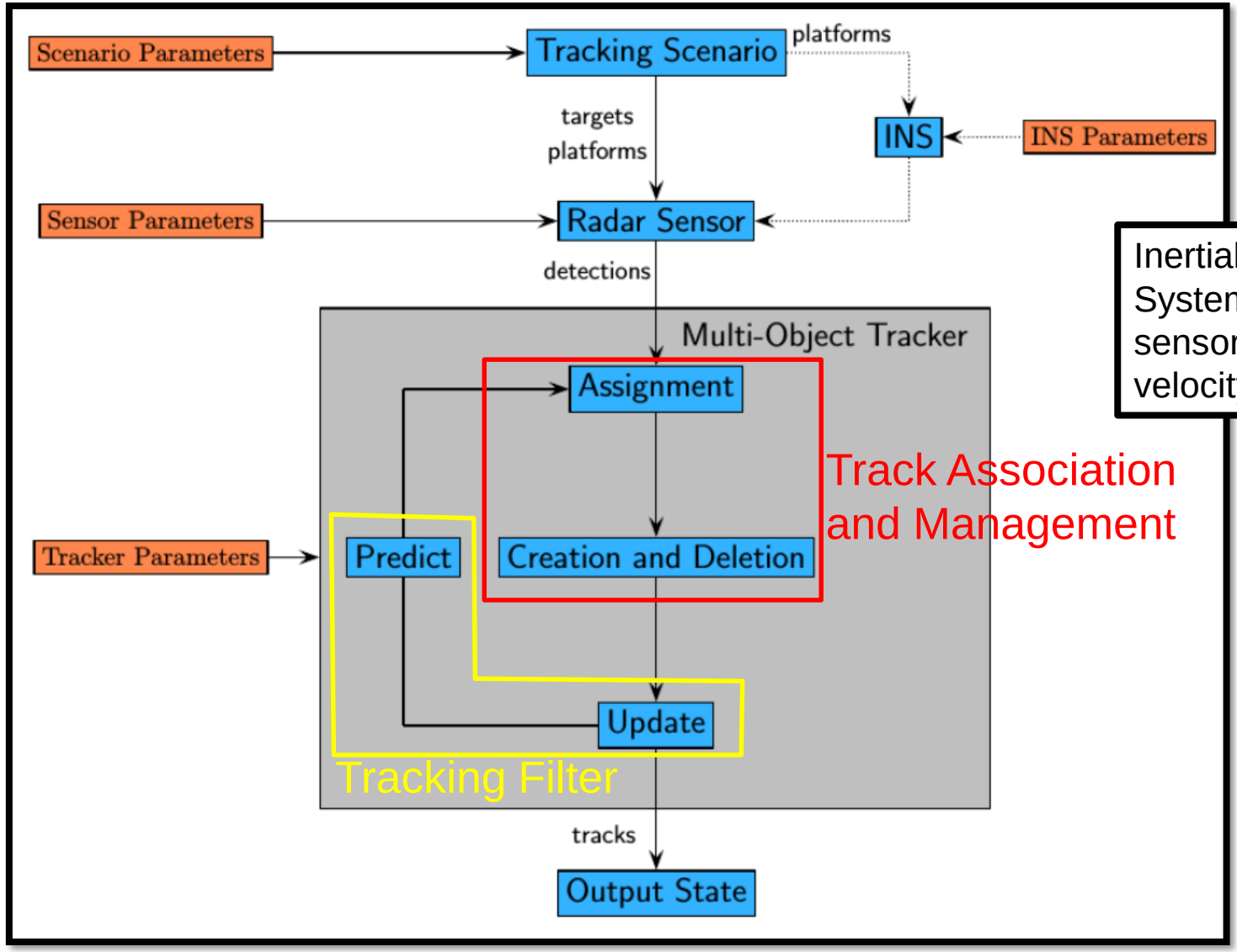
Data Association

- 2D assignment
- S-D assignment
- K-best assignmemt

Trackers

- GNN, ***JPDA, PHD***
- MHT (track-oriented)
- Trackers components
  - History and score logic
  - etc……

Components of a Multi-Object Tracker

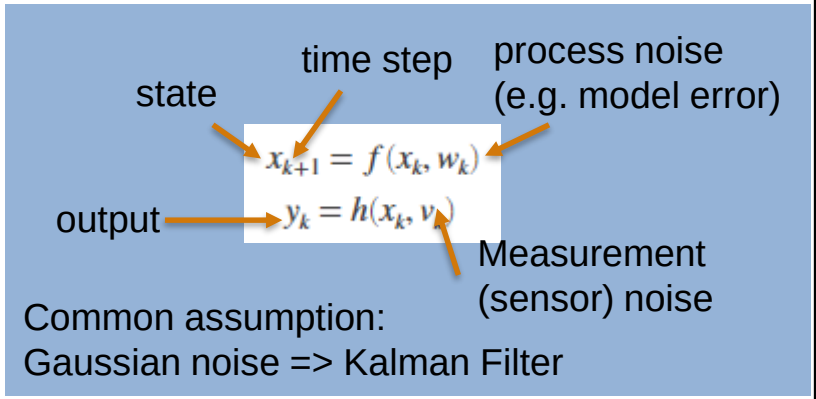


Inertial Navigation System provides radar sensor platform position, velocity and orientation

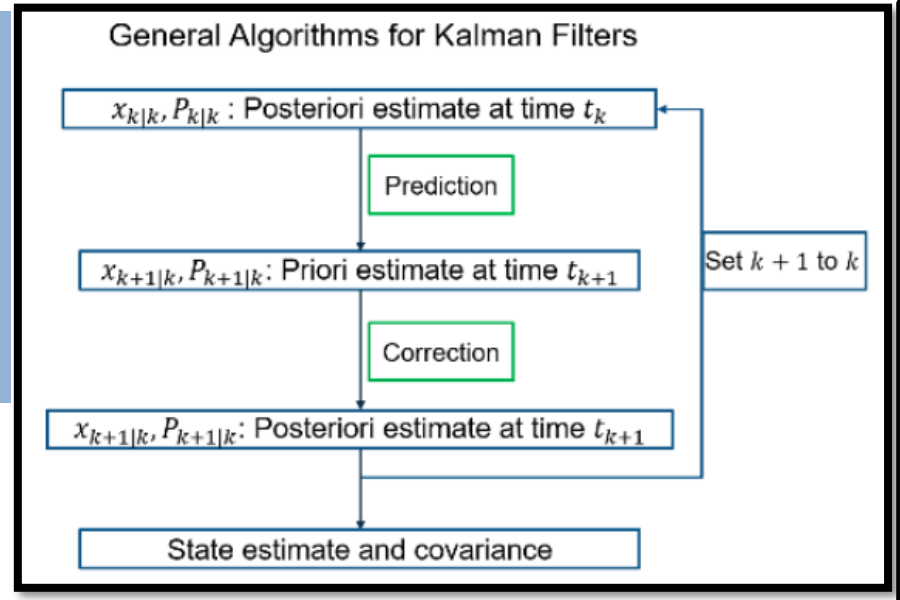
More information here

Components of a Multi-Object Tracker

Tracking Filter



Common assumption:
Gaussian noise => Kalman Filter



Covariance estimate assumes system is linear:
If $y_1(t) = H\{x_1\}$, $y_2(t) = H\{x_2\}$, then
$Ay_1(t) + By_2(t) = H\{Ay_1(t) + By_2(t)\}$

Linear Kalman Filter – Assumes linear system
Extended Kalman Filter – Linearizes the nonlinear system around state estimate
Unscented Kalman Filter – Samples the covariance distribution and propagates through non-linear model

Gaussian Sum Filter – good for partially observable cases (e.g. range only measurements)
Interactive Multiple Model Filter – good for tracking manoeuvring targets
Particle Filter – doesn't require gaussian noise

Linear Kalman Filter Example

1. Create the measured positions from a constant-velocity trajectory

```
vx = 0.2;
vy = 0.1;
T  = 0.5;
pos = [0:vx*T:2;5:vy*T:6]';
```

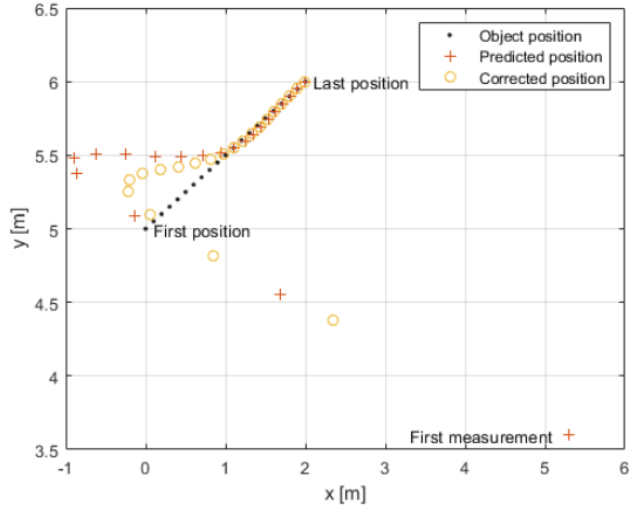
2. Specify initial position and velocity

```
x = 5.3;
y = 3.6;
initialState = [x;0;y;0];
KF = trackingKF('MotionModel','2D Constant Velocity','State',initialState);
```

State Initialisation (e.g. constant velocity/acceleration/turn and Motion Models available or use your own

3. Run Kalman Filter

```
for k = 1:size(pos,1)
    pstates(k,:) = predict(KF,T);
    cstates(k,:) = correct(KF,pos(k,:));
end
```



More information here

Components of a Multi-Object Tracker

Track Association and Management

One or more sensors generating multiple detections from multiple targets – detections must be:

1. Gated – determine which detections are valid candidates to update existing tracks

2. Assigned* – make a track to detection assignment. Assignment approaches include:

> **Global Nearest Neighbour** – Minimise overall distance of track to detection assignments
> **Joint Probability Data Association** – Soft assignment so all gated detections make weighted contributions to a track
> **Track Orientated Multiple Hypothesis Tracking** – Allows data association to be postponed until more information is received

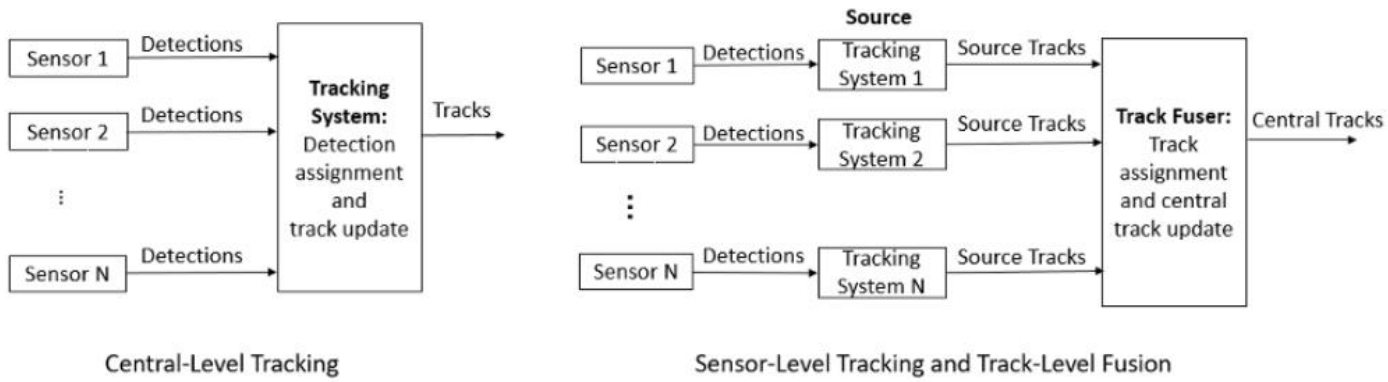
Lowest Complexity

Best Performance

Track maintenance is required for creation (tentative status), confirmation, deletion of tracks (after coasting)

> Can use history or score based logic

Advanced Topic – Track to Track Fusion:



Central-Level Tracking

Sensor-Level Tracking and Track-Level Fusion

More information here

* Some trackers (e.g. PHD Filter) don't require assignment

Components of a Multi-Object Tracker

Track Association and Management

More information here

Track Creation

```
>> h=trackerGNN

h =

  trackerGNN with properties:

                TrackerIndex: 0
      FilterInitializationFcn: 'initcvekf'
                  Assignment: 'MatchPairs'
        AssignmentThreshold: [30 Inf]
                MaxNumTracks: 100
               MaxNumSensors: 20

                  TrackLogic: 'History'
        ConfirmationThreshold: [2 3]
           DeletionThreshold: [5 5]

          HasCostMatrixInput: false
  HasDetectableTrackIDsInput: false
             StateParameters: [1×1 struct]

                   NumTracks: 0
          NumConfirmedTracks: 0
```

Data

Time

```
>> h({objectDetection(1,[1;2;3])},1)

ans =

  0×1 objectTrack array with properties:

    TrackID
    BranchID
    SourceIndex
    UpdateTime
    Age
    State
    StateCovariance
    StateParameters
    ObjectClassID
    TrackLogic
    TrackLogicState
    IsConfirmed
    IsCoasted
    IsSelfReported
    ObjectAttributes
```

```
>> h({objectDetection(2,[1;2;3])},2)

ans =

  objectTrack with properties:

            TrackID: 1
           BranchID: 0
        SourceIndex: 0
         UpdateTime: 2
                Age: 2
              State: [6×1 double]
    StateCovariance: [6×6 double]
    StateParameters: [1×1 struct]
       ObjectClassID: 0
         TrackLogic: 'History'
    TrackLogicState: [1 1 0 0 0]
       IsConfirmed: 1
        IsCoasted: 0
     IsSelfReported: 1
   ObjectAttributes: [1×1 struct]
```

```
h =

  trackerGNN with properties:

                TrackerIndex: 0
      FilterInitializationFcn: 'initcvekf'
                  Assignment: 'MatchPairs'
        AssignmentThreshold: [30 Inf]
                MaxNumTracks: 100
               MaxNumSensors: 20

                  TrackLogic: 'History'
        ConfirmationThreshold: [2 3]
           DeletionThreshold: [5 5]

          HasCostMatrixInput: false
  HasDetectableTrackIDsInput: false
             StateParameters: [1×1 struct]

                   NumTracks: 1
          NumConfirmedTracks: 1
```

Track Deletion

```
>> h({},3)

ans =

  objectTrack with properties:

            TrackID: 1
           BranchID: 0
        SourceIndex: 0
         UpdateTime: 3
                Age: 3
              State: [6×1 double]
    StateCovariance: [6×6 double]
    StateParameters: [1×1 struct]
       ObjectClassID: 0
         TrackLogic: 'History'
    TrackLogicState: [0 1 1 0 0]
       IsConfirmed: 1
        IsCoasted: 1
     IsSelfReported: 1
   ObjectAttributes: [1×1 struct]
```

```
>> h({},4)
>> h({},5)
>> h({},6)
>> h({},7)

ans =

  0×1 objectTrack array with properties:

    TrackID
    BranchID
    SourceIndex
    UpdateTime
    Age
    State
    StateCovariance
    StateParameters
    ObjectClassID
    TrackLogic
    TrackLogicState
    IsConfirmed
    IsCoasted
    IsSelfReported
    ObjectAttributes
```

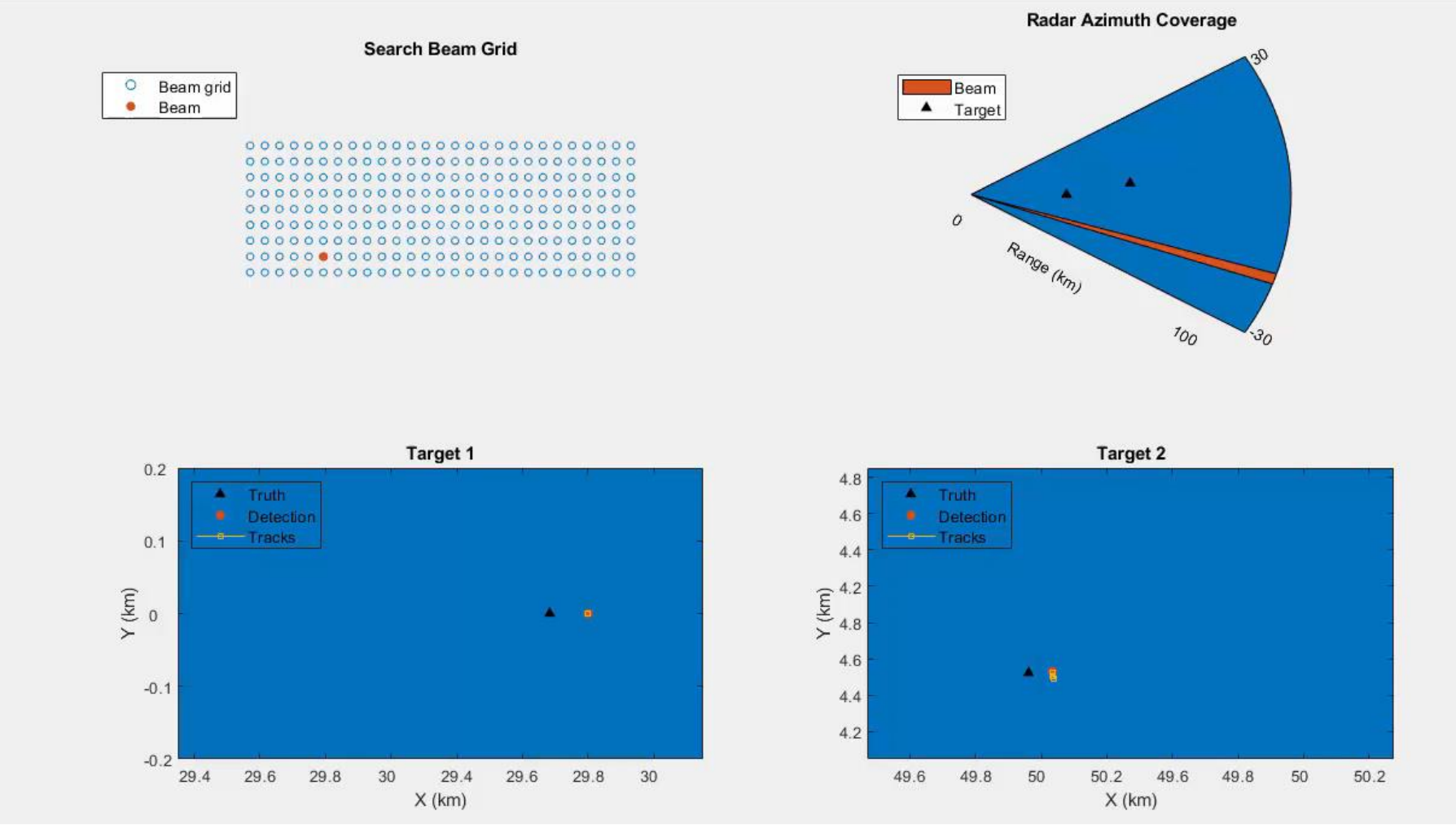
```
h =

  trackerGNN with properties:

                TrackerIndex: 0
      FilterInitializationFcn: 'initcvekf'
                  Assignment: 'MatchPairs'
        AssignmentThreshold: [30 Inf]
                MaxNumTracks: 100
               MaxNumSensors: 20

                  TrackLogic: 'History'
        ConfirmationThreshold: [2 3]
           DeletionThreshold: [5 5]

          HasCostMatrixInput: false
  HasDetectableTrackIDsInput: false
             StateParameters: [1×1 struct]

                   NumTracks: 0
          NumConfirmedTracks: 0
```

Example of Multi-Object Tracking:
Multifunction Radar: Search and Track

Search

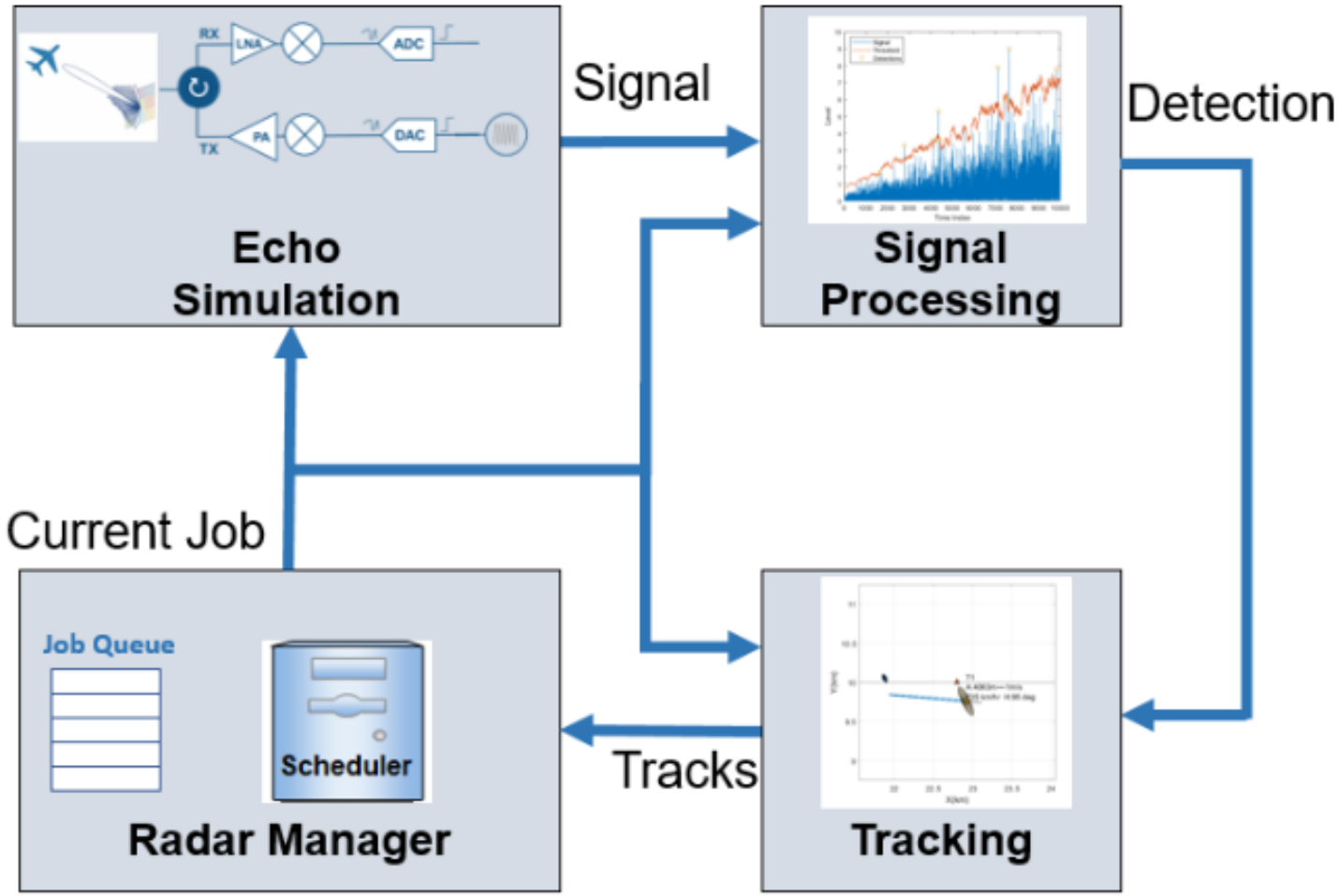
Track (confirm)

Track (update)

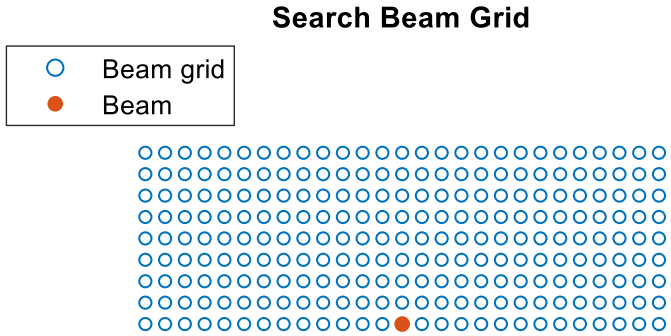


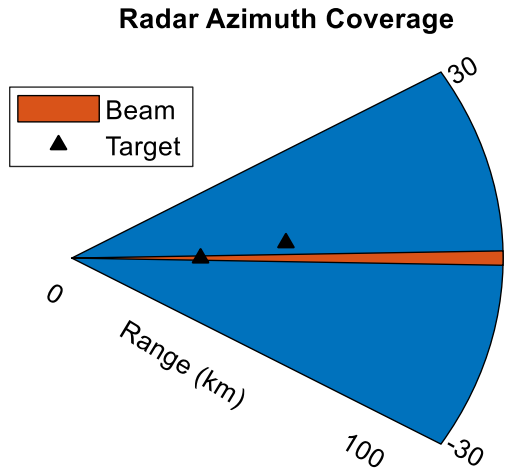
Example here

Example of Multi-Object Tracking:
Multifunction Radar: Search and Track

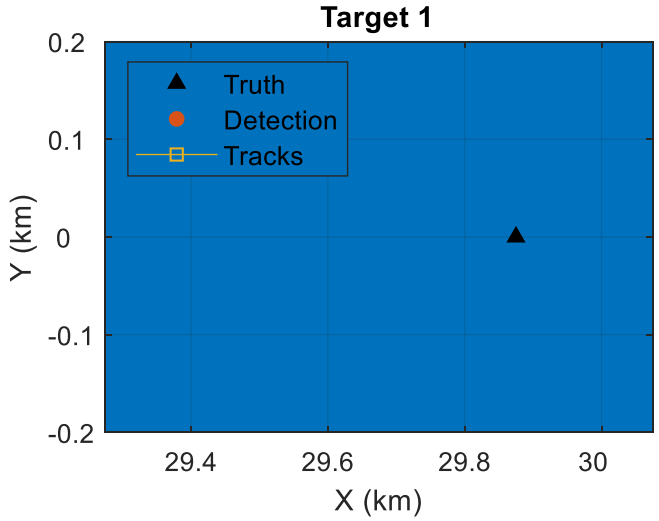


Example of Multi-Object Tracking:
Multifunction Radar: Search and Track



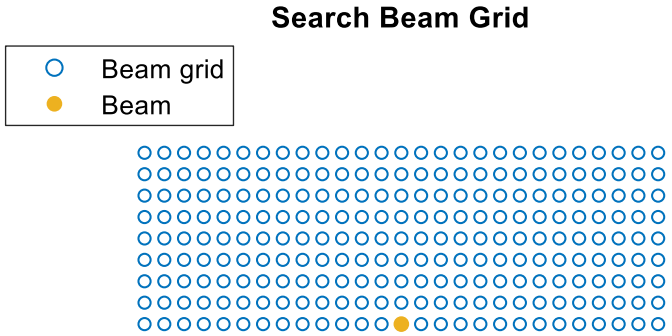


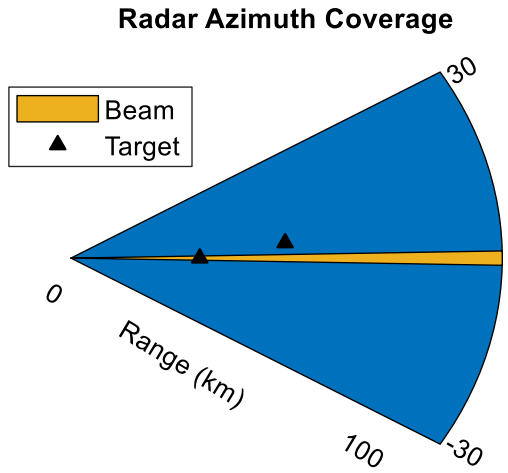
Target 1 Detected



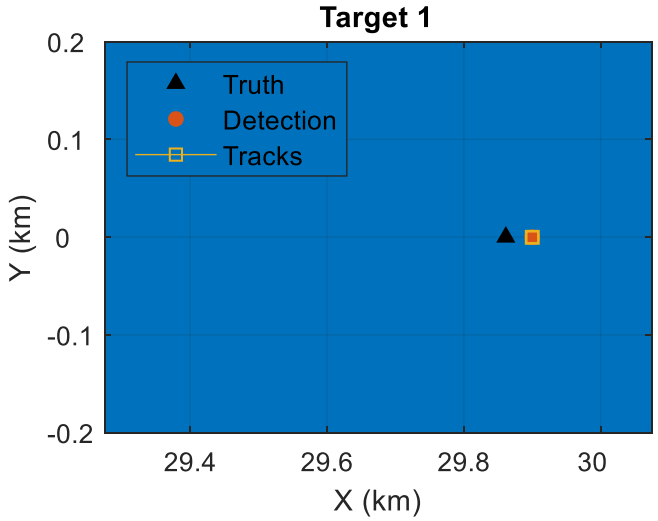

```
0.000000 sec:    Search   [-30.000000 0.000000]
0.010000 sec:    Search   [-27.692308 0.000000]
0.020000 sec:    Search   [-25.384615 0.000000]
0.030000 sec:    Search   [-23.076923 0.000000]
0.040000 sec:    Search   [-20.769231 0.000000]
0.050000 sec:    Search   [-18.461538 0.000000]
0.060000 sec:    Search   [-16.153846 0.000000]
0.070000 sec:    Search   [-13.846154 0.000000]
0.080000 sec:    Search   [-11.538462 0.000000]
0.090000 sec:    Search   [-9.230769 0.000000]
0.100000 sec:    Search   [-6.923077 0.000000]
0.110000 sec:    Search   [-4.615385 0.000000]
0.120000 sec:    Search   [-2.307692 0.000000]
0.130000 sec:    Search   [0.000000 0.000000] Target detected at 29900.
```

Example of Multi-Object Tracking:
Multifunction Radar: Search and Track



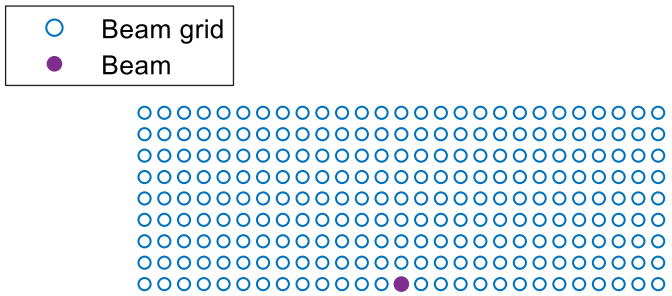


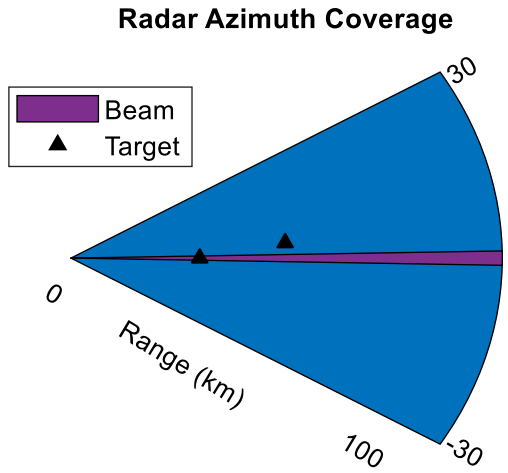
Detection Confirmed and Track 1 Created



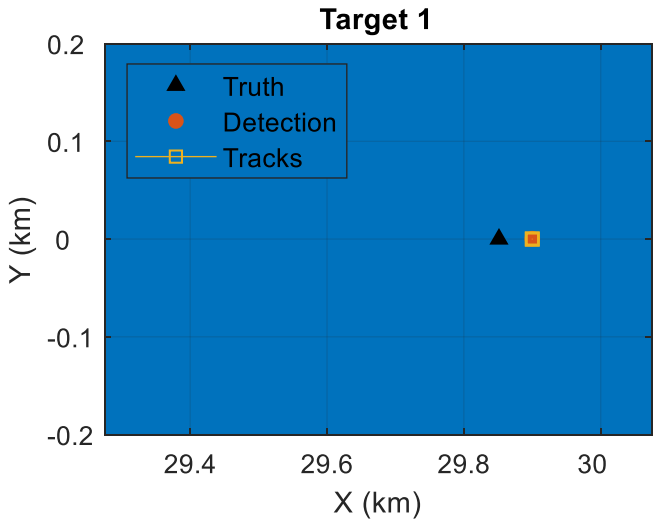

```
0.000000 sec:    Search   [-30.000000 0.000000]
0.010000 sec:    Search   [-27.692308 0.000000]
0.020000 sec:    Search   [-25.384615 0.000000]
0.030000 sec:    Search   [-23.076923 0.000000]
0.040000 sec:    Search   [-20.769231 0.000000]
0.050000 sec:    Search   [-18.461538 0.000000]
0.060000 sec:    Search   [-16.153846 0.000000]
0.070000 sec:    Search   [-13.846154 0.000000]
0.080000 sec:    Search   [-11.538462 0.000000]
0.090000 sec:    Search   [-9.230769 0.000000]
0.100000 sec:    Search   [-6.923077 0.000000]
0.110000 sec:    Search   [-4.615385 0.000000]
0.120000 sec:    Search   [-2.307692 0.000000]
0.130000 sec:    Search   [0.000000 0.000000] Target detected at 29900.000000 m
0.140000 sec:    Confirm  [-0.000586 -0.000034]   Created track 1 at 29900.000000 m
```

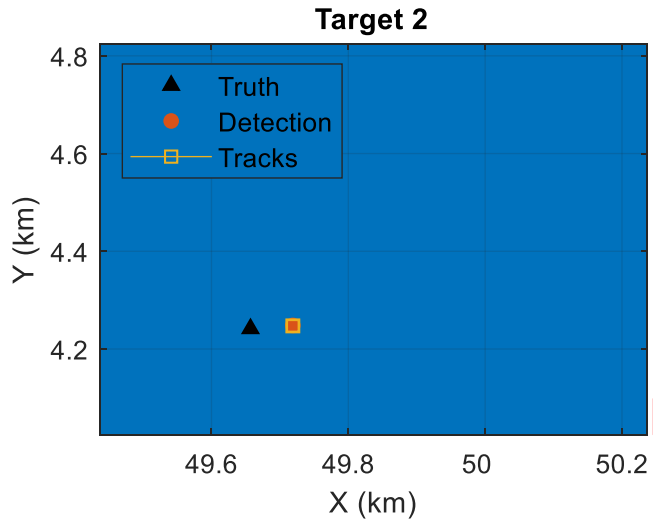
Example of Multi-Object Tracking:
Multifunction Radar: Search and Track





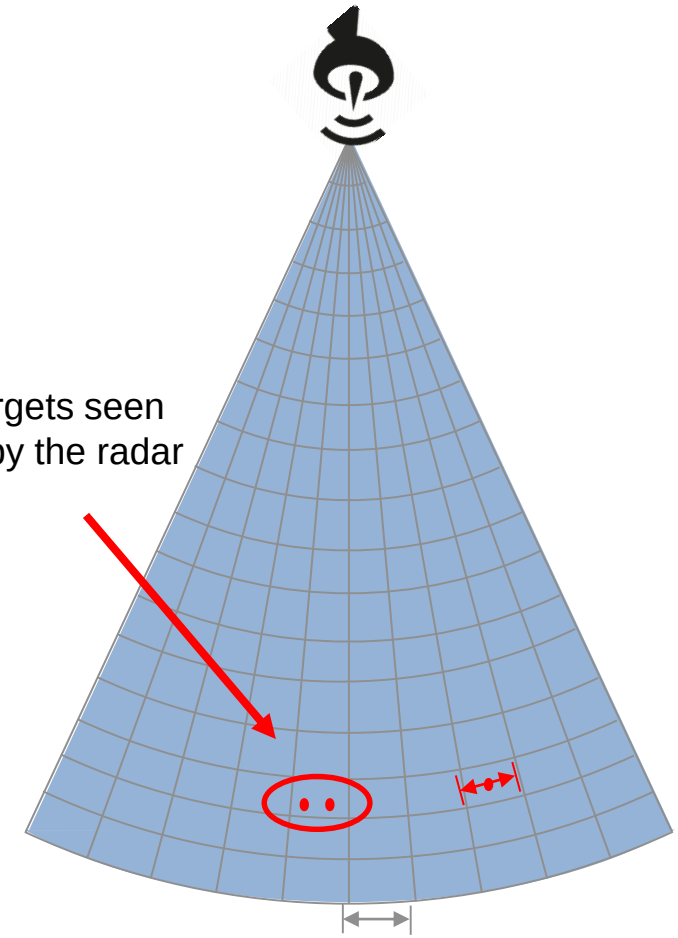
Track 1 Updated

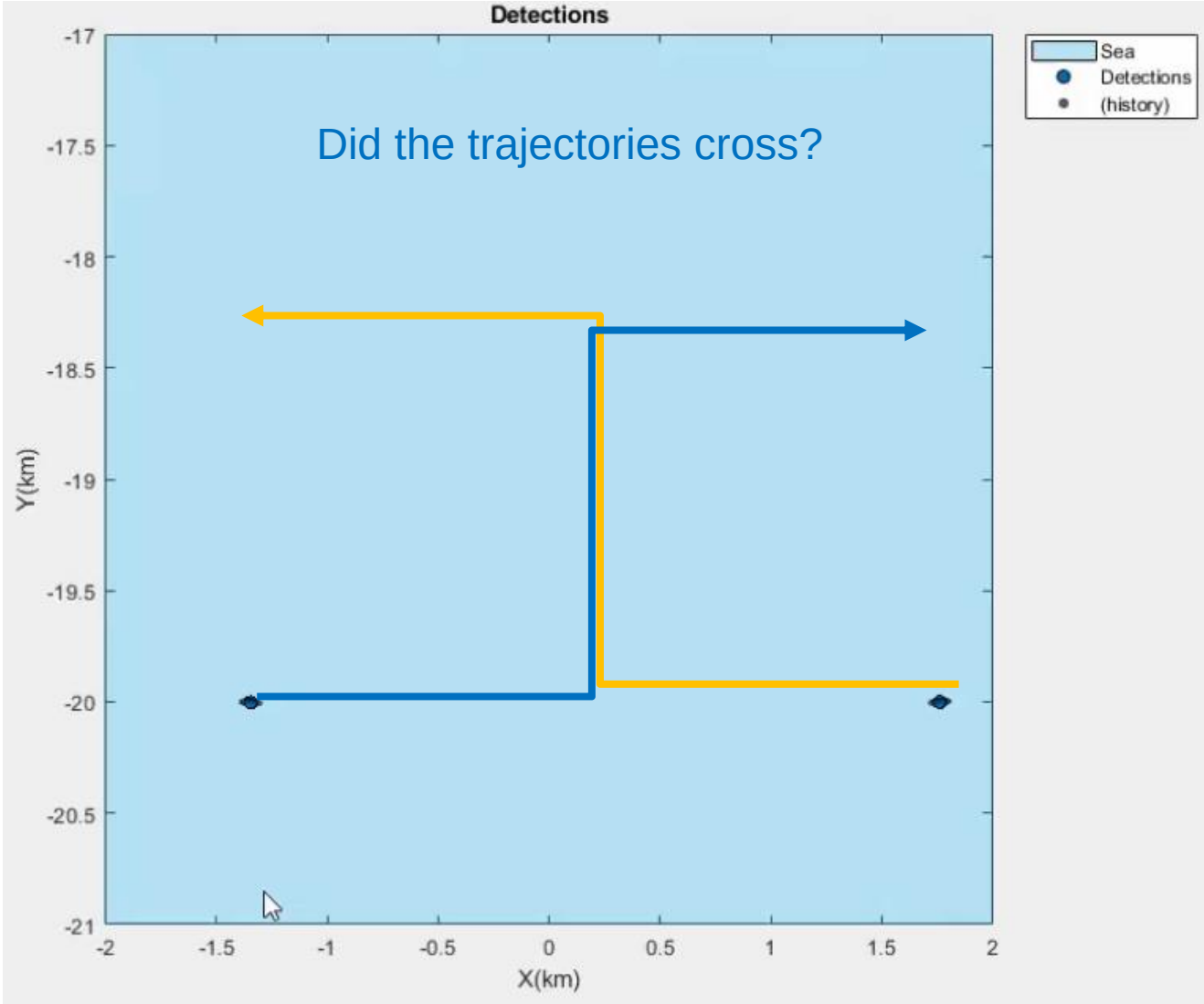





```
0.000000 sec:   Search   [-30.000000 0.000000]
0.010000 sec:   Search   [-27.692308 0.000000]
0.020000 sec:   Search   [-25.384615 0.000000]
0.030000 sec:   Search   [-23.076923 0.000000]
0.040000 sec:   Search   [-20.769231 0.000000]
0.050000 sec:   Search   [-18.461538 0.000000]
0.060000 sec:   Search   [-16.153846 0.000000]
0.070000 sec:   Search   [-13.846154 0.000000]
0.080000 sec:   Search   [-11.538462 0.000000]
0.090000 sec:   Search   [-9.230769 0.000000]
0.100000 sec:   Search   [-6.923077 0.000000]
0.110000 sec:   Search   [-4.615385 0.000000]
0.120000 sec:   Search   [-2.307692 0.000000]
0.130000 sec:   Search   [0.000000 0.000000] Target detected at 29900.000000 m
0.140000 sec:   Confirm  [-0.000586 -0.000034]   Created track 1 at 29900.000000 m
0.150000 sec:   Search   [2.307692 0.000000]
0.160000 sec:   Search   [4.615385 0.000000] Target detected at 49900.000000 m
0.170000 sec:   Confirm  [4.881676 0.000739] Created track 2 at 49900.000000 m
0.180000 sec:   Search   [6.923077 0.000000]
0.190000 sec:   Search   [9.230769 0.000000]
0.200000 sec:   Search   [11.538462 0.000000]
0.210000 sec:   Search   [13.846154 0.000000]
0.220000 sec:   Search   [16.153846 0.000000]
0.230000 sec:   Search   [18.461538 0.000000]
0.240000 sec:   Track    [-0.000399 0.000162]    Track 1 at 29900.000000 m>>
```

Example of Multi-Object Tracking:
Performing What-If Analysis

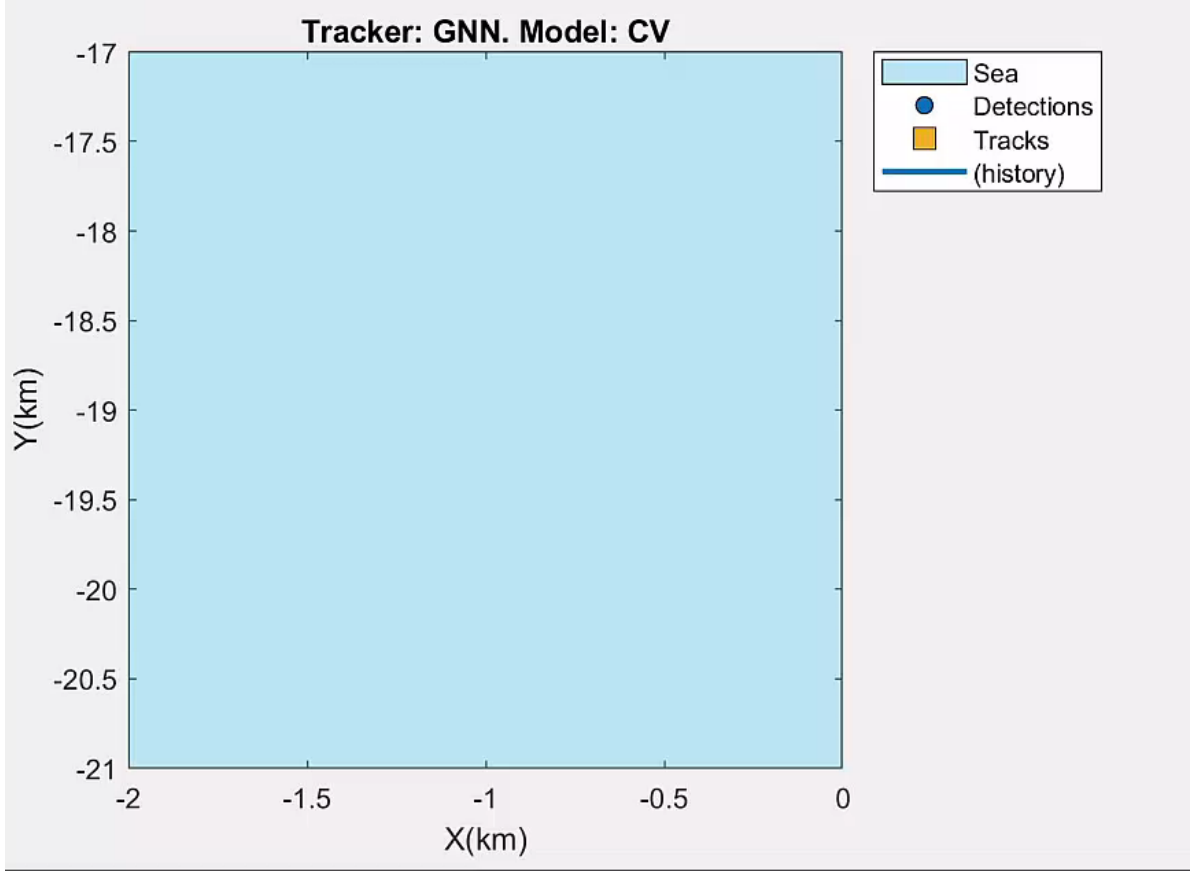




MATLAB EXPO 2019

Example here

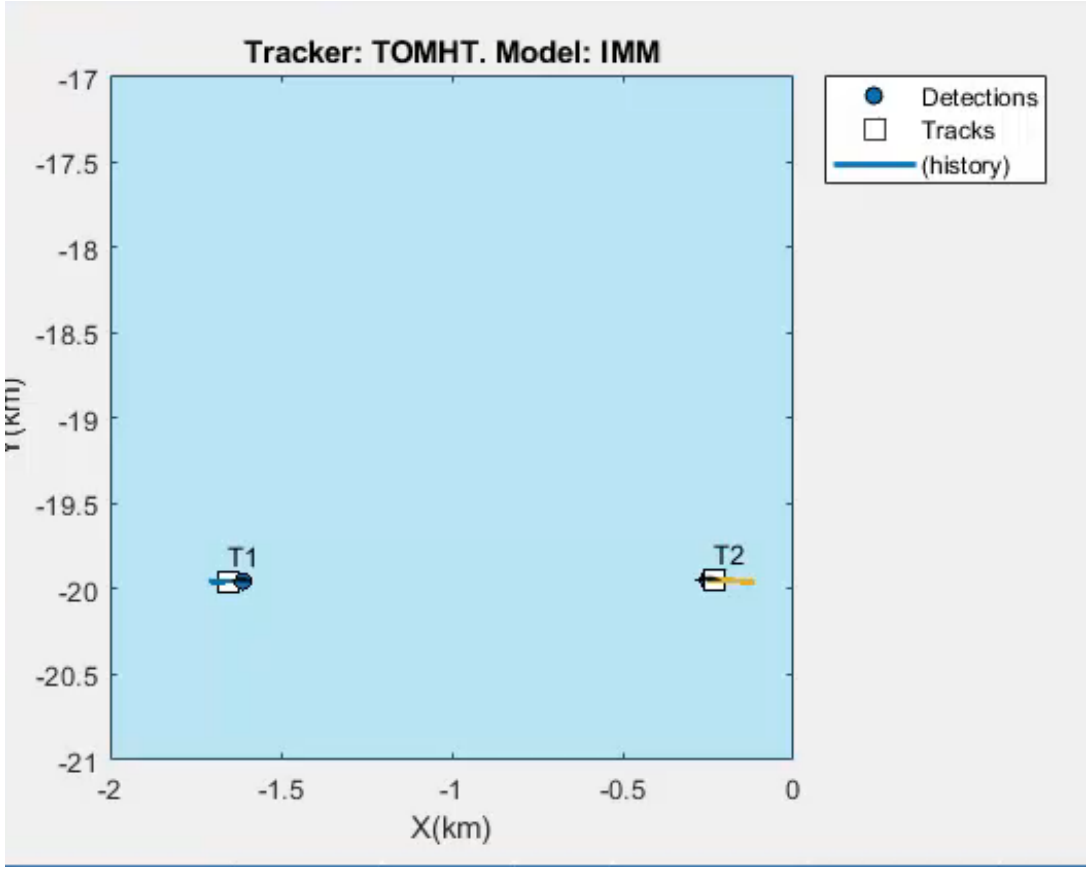
Example of Multi-Object Tracking:
Performing What-If Analysis




```
tracker = trackerGNN( ...
  'FilterInitializationFcn',@initCVFilter,...
  'MaxNumTracks', numTracks, ...
  'MaxNumSensors', 1, ...
  'AssignmentThreshold',gate, ...
  'TrackLogic', 'Score', ...
  'DetectionProbability', pd, ...
  'FalseAlarmRate', far, ...
  'Volume', vol, 'Beta', beta);
```

```
tracker = trackerGNN( ...
  'FilterInitializationFcn',@initIMMFilter,...
  'MaxNumTracks', numTracks, ...
  'MaxNumSensors', 1, ...
  'AssignmentThreshold',gate, ...
  'TrackLogic', 'Score', ...
  'DetectionProbability', pd, ...
  'FalseAlarmRate', far, ...
  'Volume', vol, 'Beta', beta);
```

Example of Multi-Object Tracking:
Performing What-If Analysis



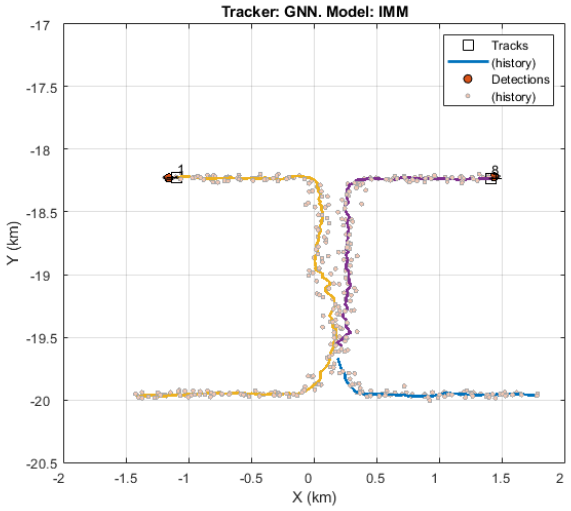

```
tracker = trackerTOMHT( ...
  'FilterInitializationFcn',@initIMMFilter,...
  'MaxNumTracks', numTracks, ...
  'MaxNumSensors', 1, ...
  'AssignmentThreshold',[0.2,1,1]*gate, ...
  'TrackLogic', 'Score', ...
  'DetectionProbability', pd, ...
  'FalseAlarmRate', far, ...
  'Volume', vol, 'Beta', beta, ...
  'MaxNumHistoryScans', 10, ...
  'MaxNumTrackBranches', 5,...
  'NScanPruning', 'Hypothesis', ...
  'OutputRepresentation', 'Tracks');
```

Example of Multi-Object Tracking:
Comparing Trackers and Tracking Filters

False track
Dropped track

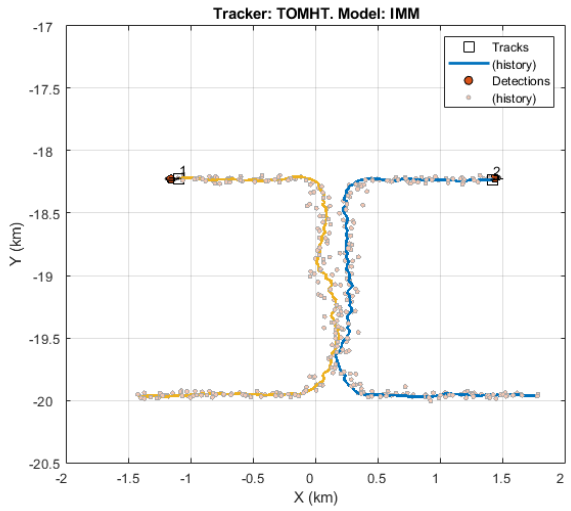
TrackID	AssignedTruthID	Surviving	TotalLength	DivergenceStatus
1	2	true	190	false
2	NaN	false	77	true
8	3	true	111	false

TruthID	AssociatedTrackID	TotalLength	BreakCount	EstablishmentLength
2	1	192	0	4
3	8	192	1	2



TrackID	AssignedTruthID	Surviving	TotalLength	DivergenceStatus
1	2	true	190	false
2	3	true	191	false

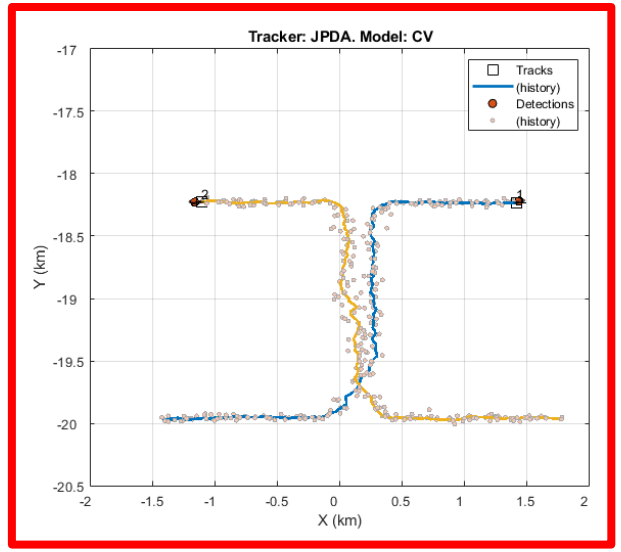
TruthID	AssociatedTrackID	TotalLength	BreakCount	EstablishmentLength
2	1	192	0	2
3	2	192	0	2



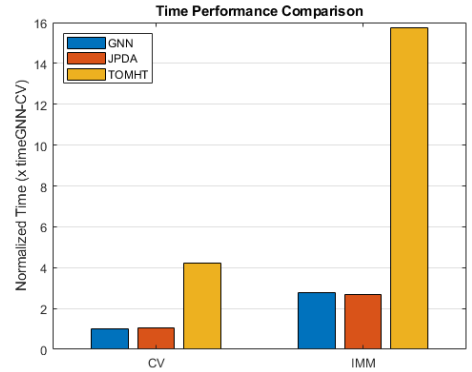
Slower

TrackID	AssignedTruthID	Surviving	TotalLength	DivergenceStatus
1	2	true	191	false
2	3	true	191	false

TruthID	AssociatedTrackID	TotalLength	BreakCount	EstablishmentLength
2	1	192	0	1
3	2	192	0	2

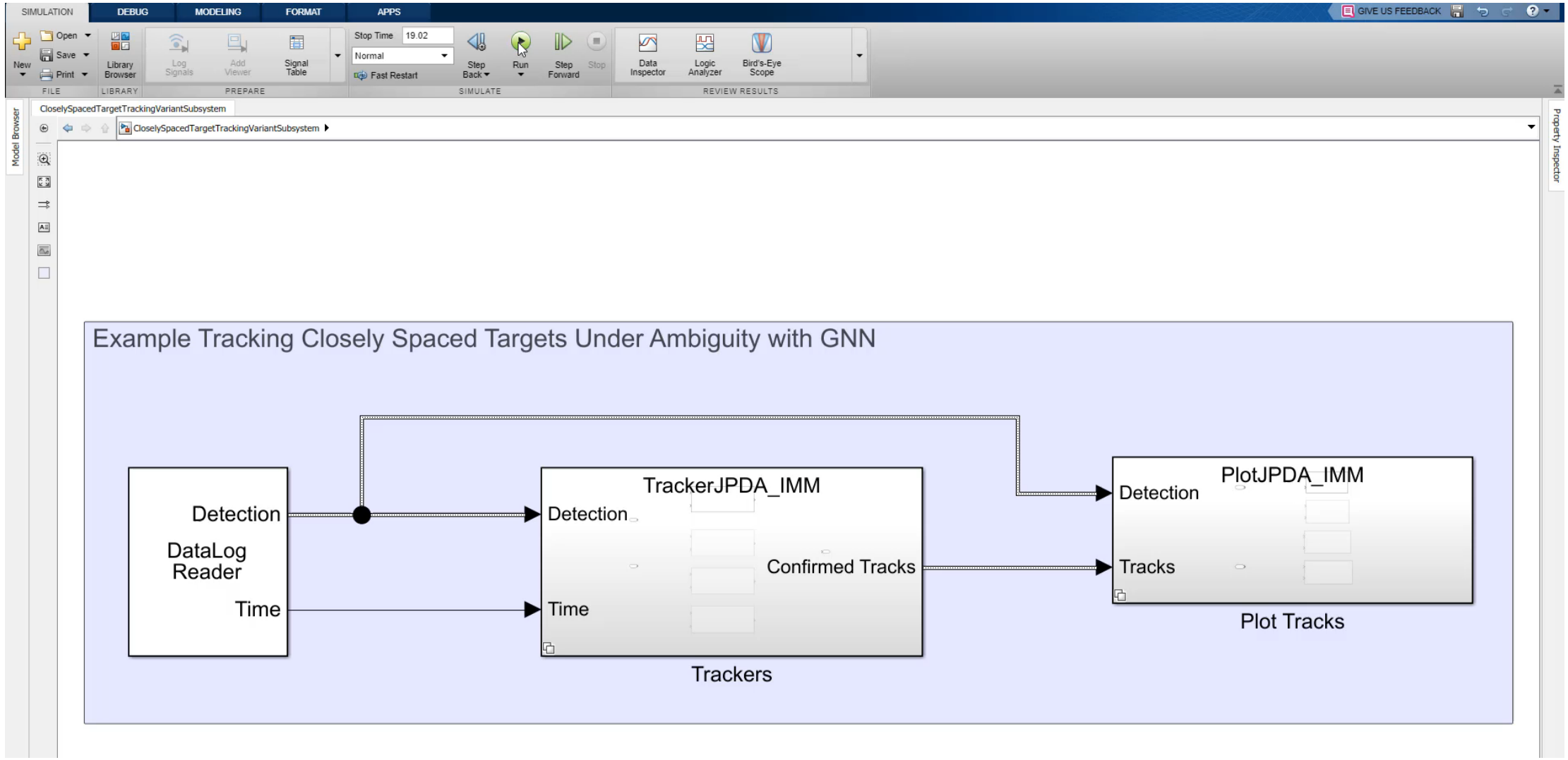


Faster



Simulink Support for Multi-Object Tracking

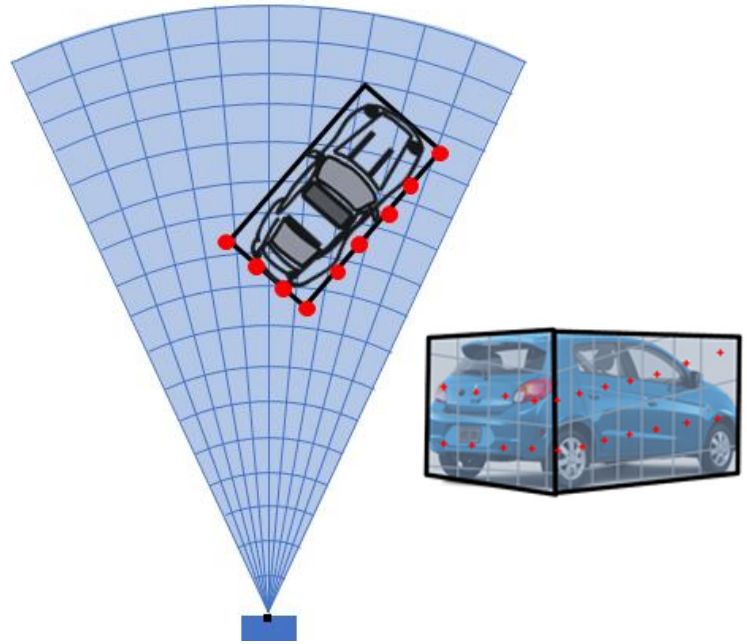
R2019b



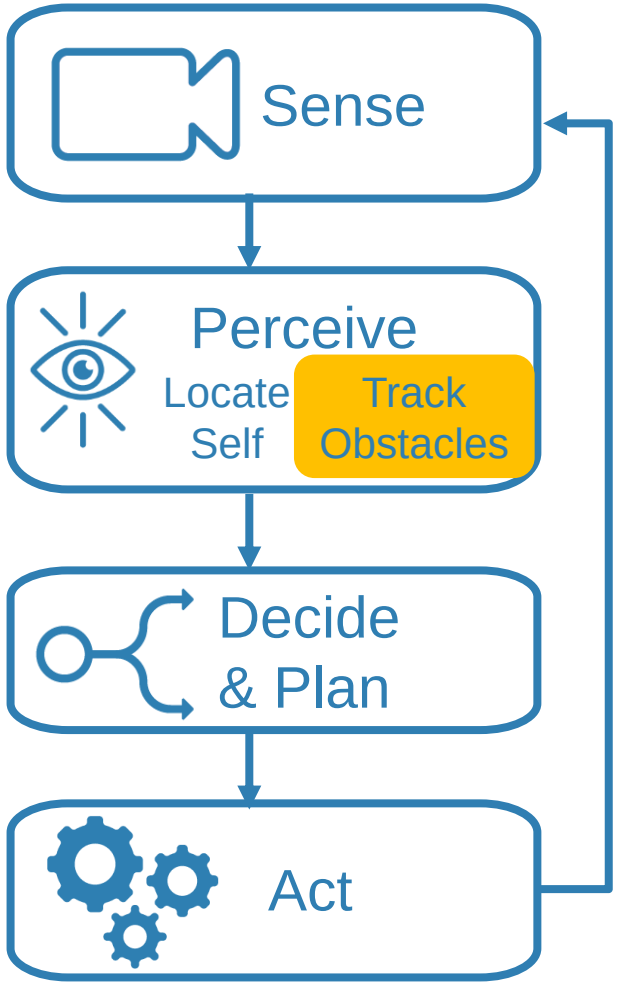
Point object vs. Extended object

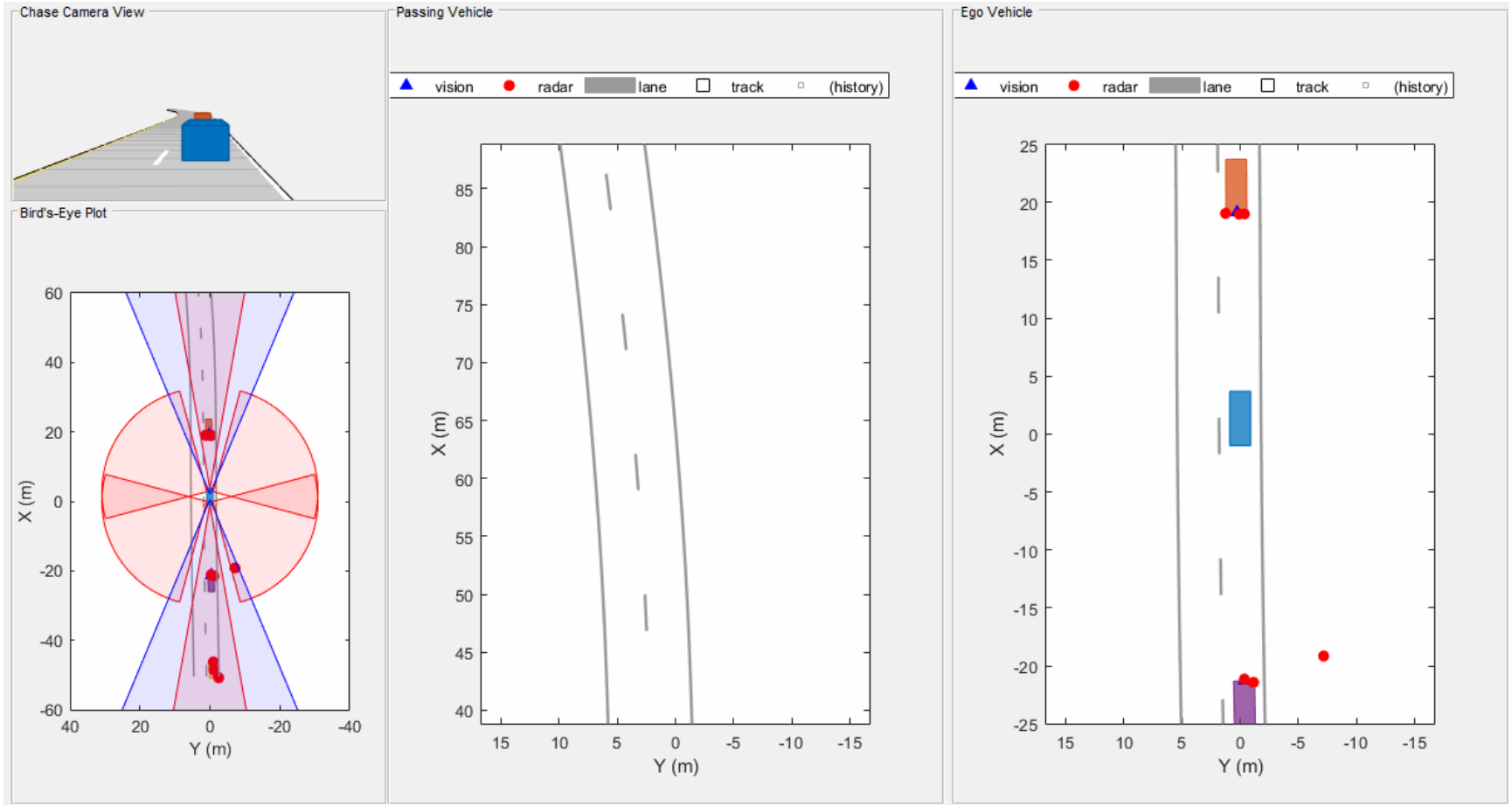
- **Point object**
  - Distant object represented as a single point
  - One detection per object per scan

- **Extended object**
  - High resolution sensors generate multiple detections per object per scan

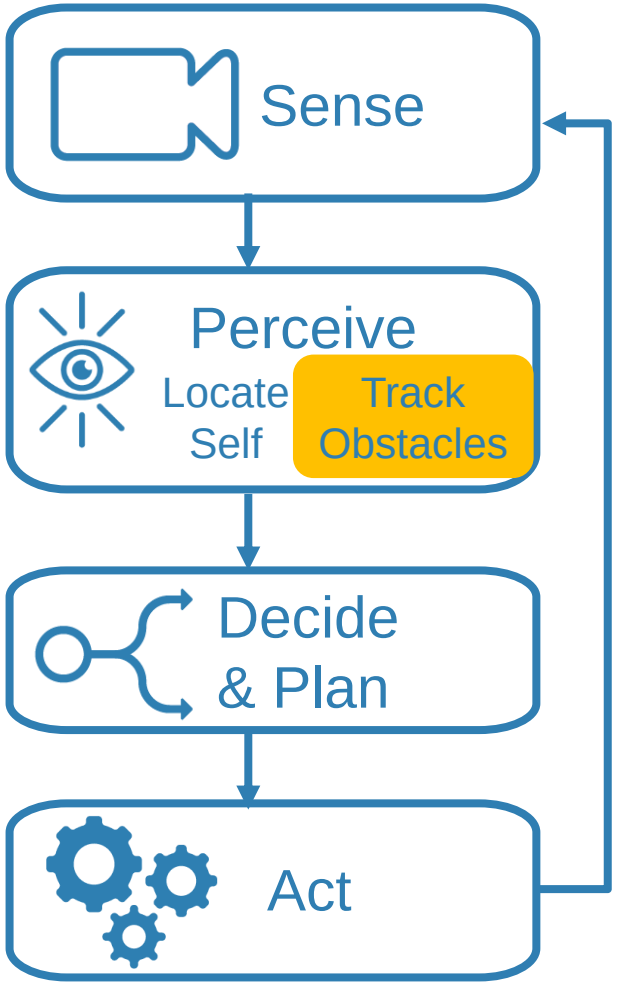


Extended Object Tracking Example

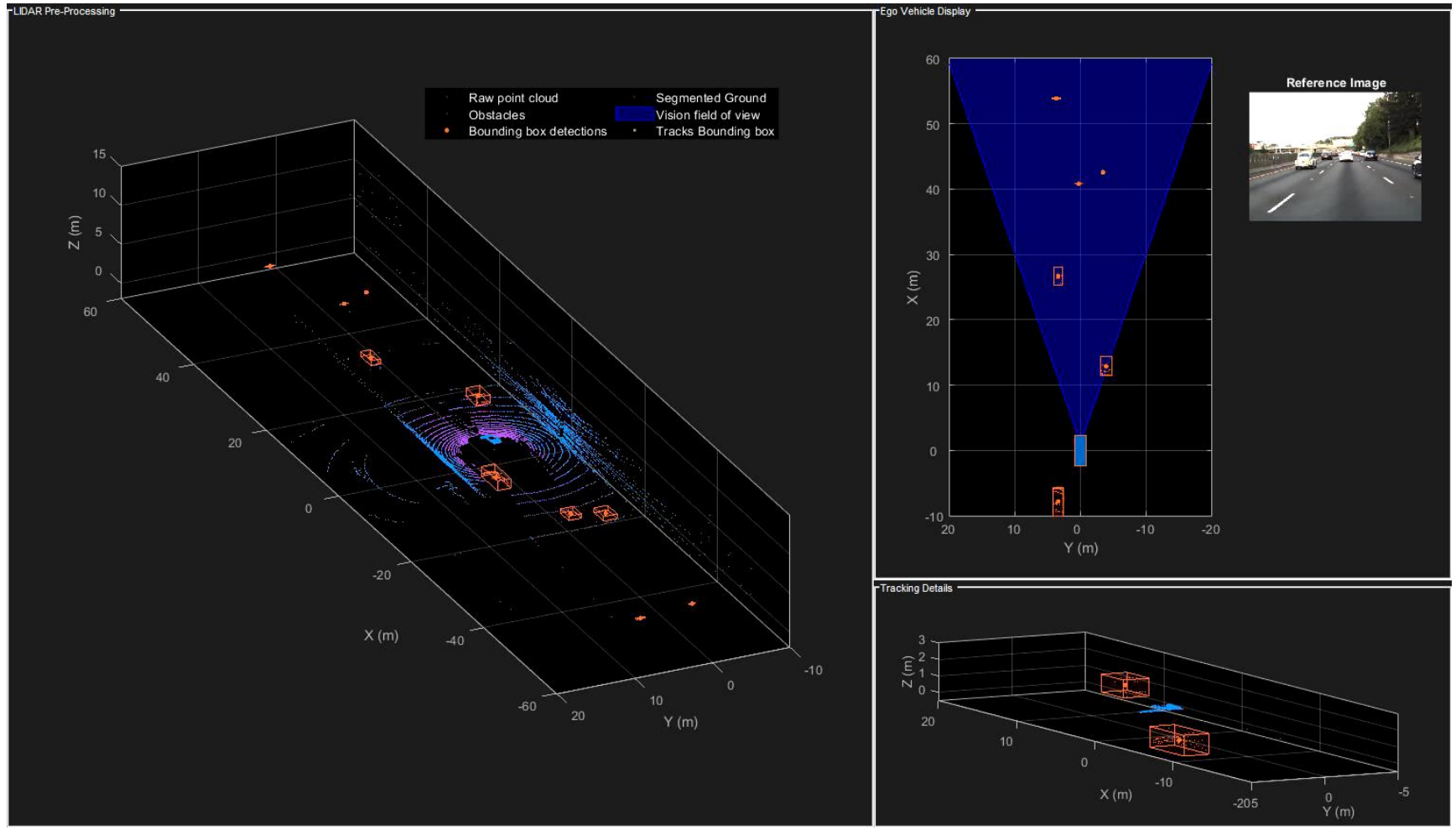




Tracking with Lidar (Even more points!!)

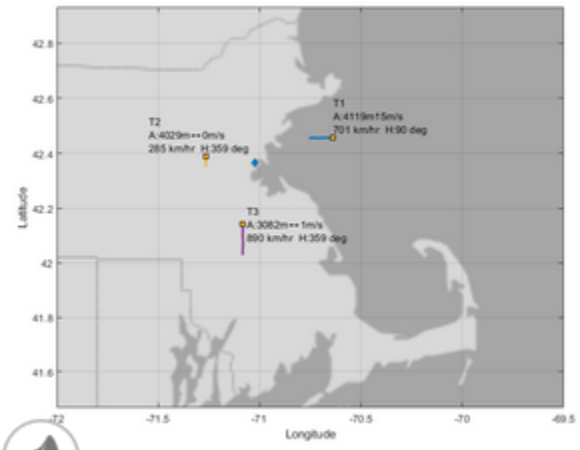


JPDA Tracker with IMM



Pre-process point cloud data to extract objects of interest. Example here.

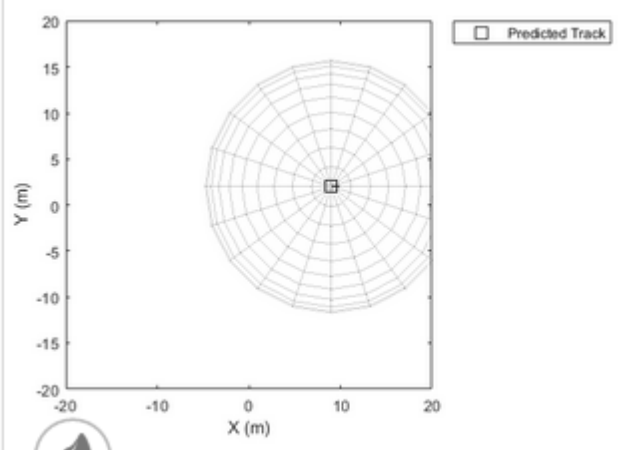
To go further on tracking with a single sensor, see also



Air Traffic Control

Generate an air traffic control scenario, simulate radar detections from an airport surveillance radar (ASR), and configure a global

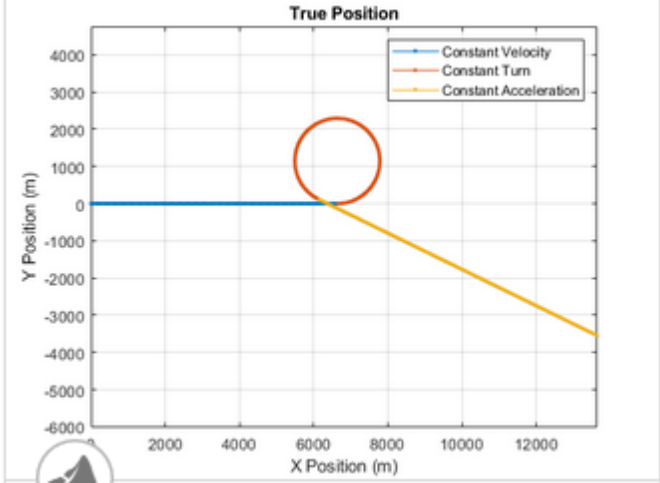
Open Script



Introduction to Using the Global Nearest Neighbor Tracker

Configure and use the global nearest neighbor (GNN) tracker.

Open Script



Tracking Maneuvering Targets

Track maneuvering targets using various tracking filters. The example shows the difference between filters that use a single motion model and

Open Script

Agenda

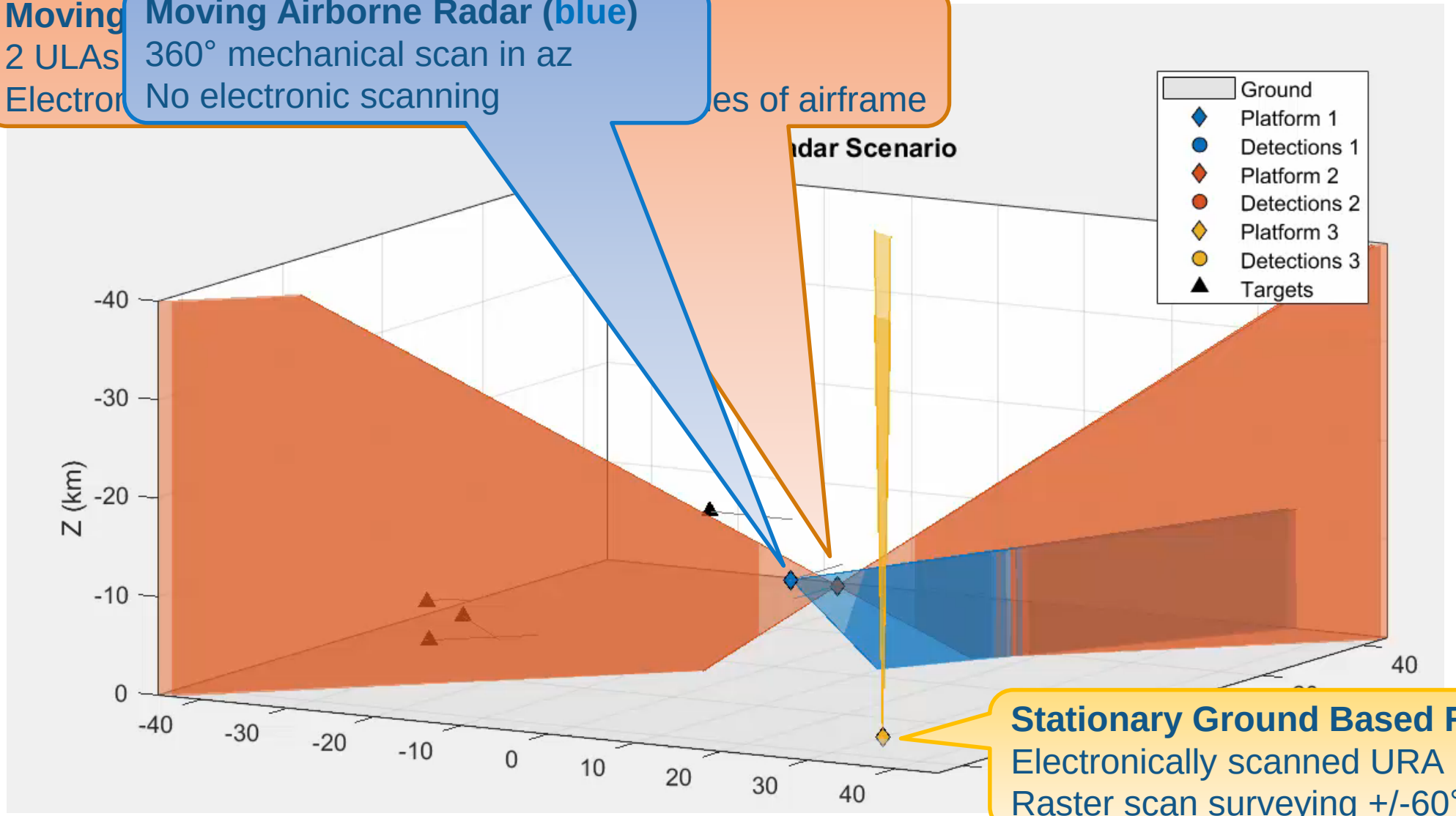
- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Multi-platform Scenario Generation

Moving
2 ULAs
Electron

Moving Airborne Radar (blue)
360° mechanical scan in az
No electronic scanning

es of airframe

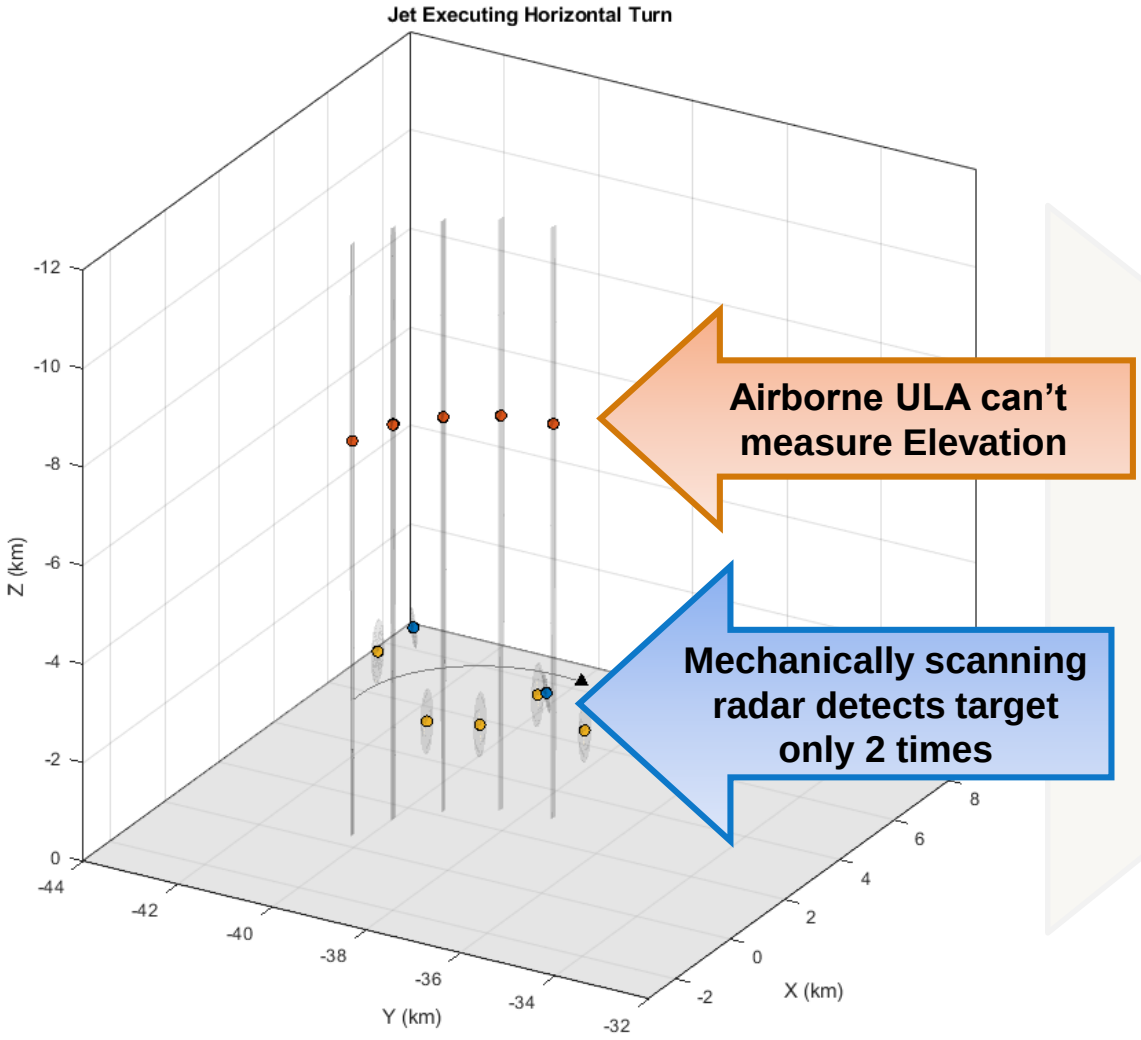


Stationary Ground Based Radar (yellow)
Electronically scanned URA
Raster scan surveying +/-60° az and -20 to 0° el

Example here

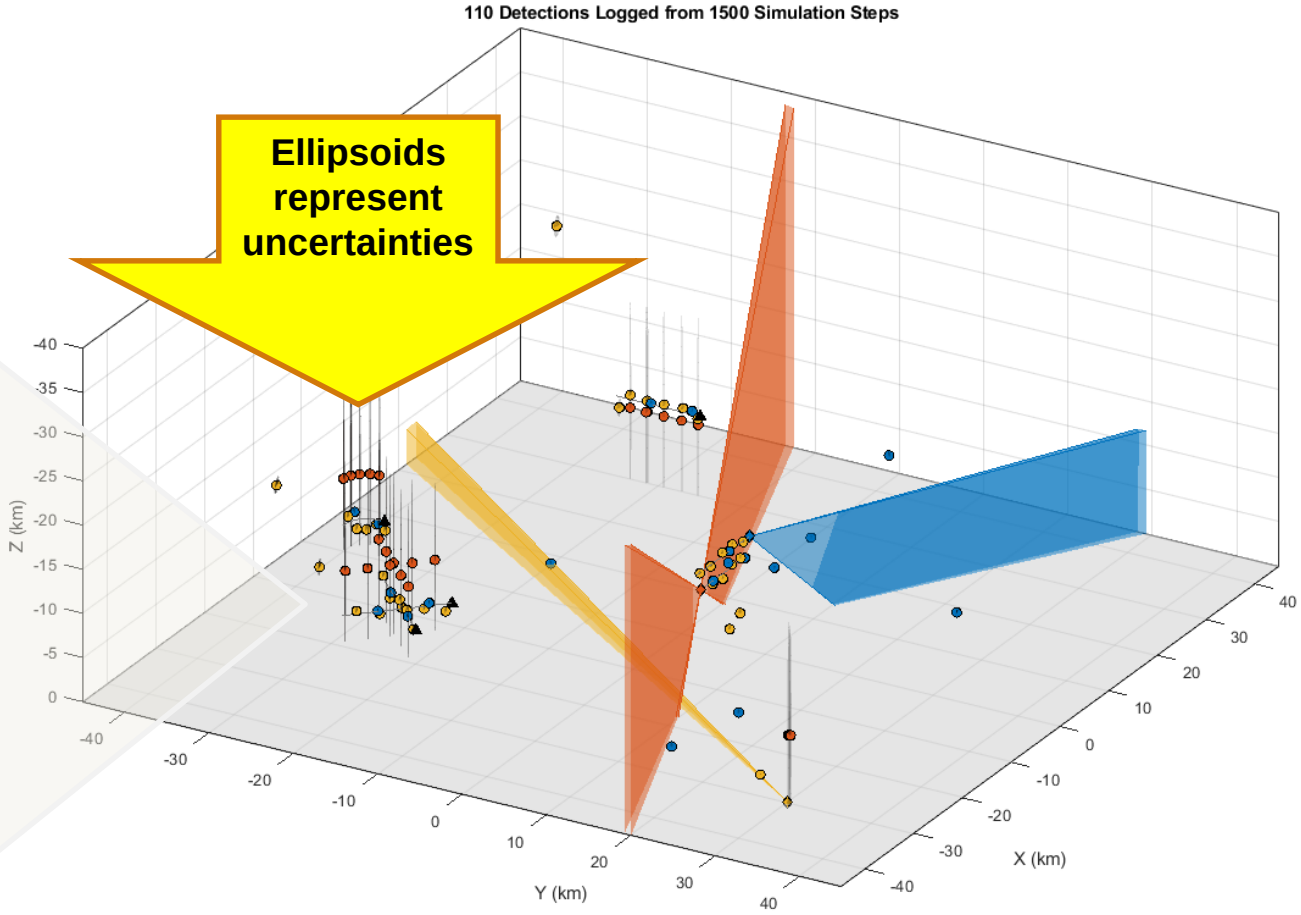
Multi-platform Scenario Generation

Visualize Detections and Measurement Uncertainties



Airborne ULA can't measure Elevation

Mechanically scanning radar detects target only 2 times

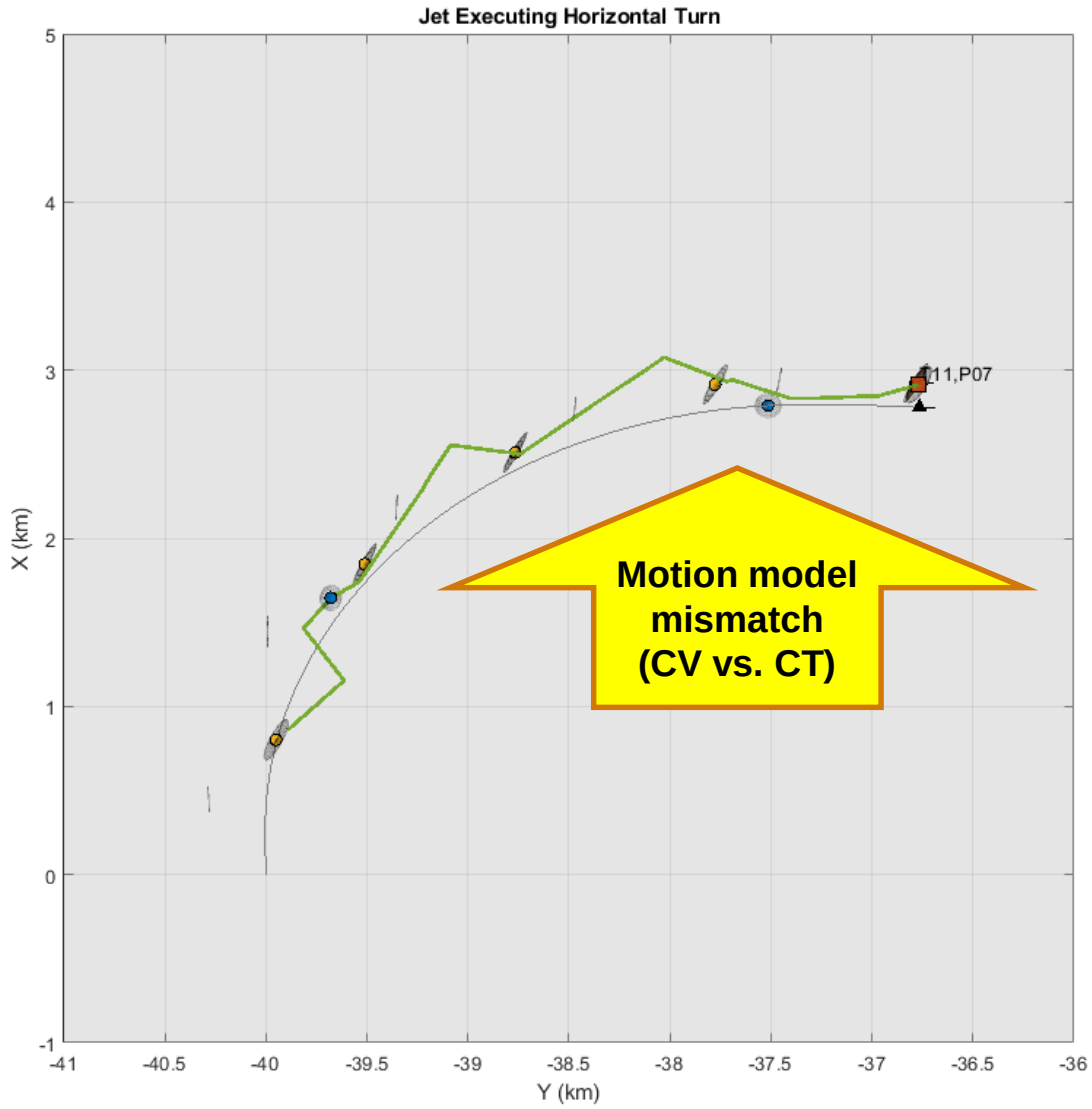


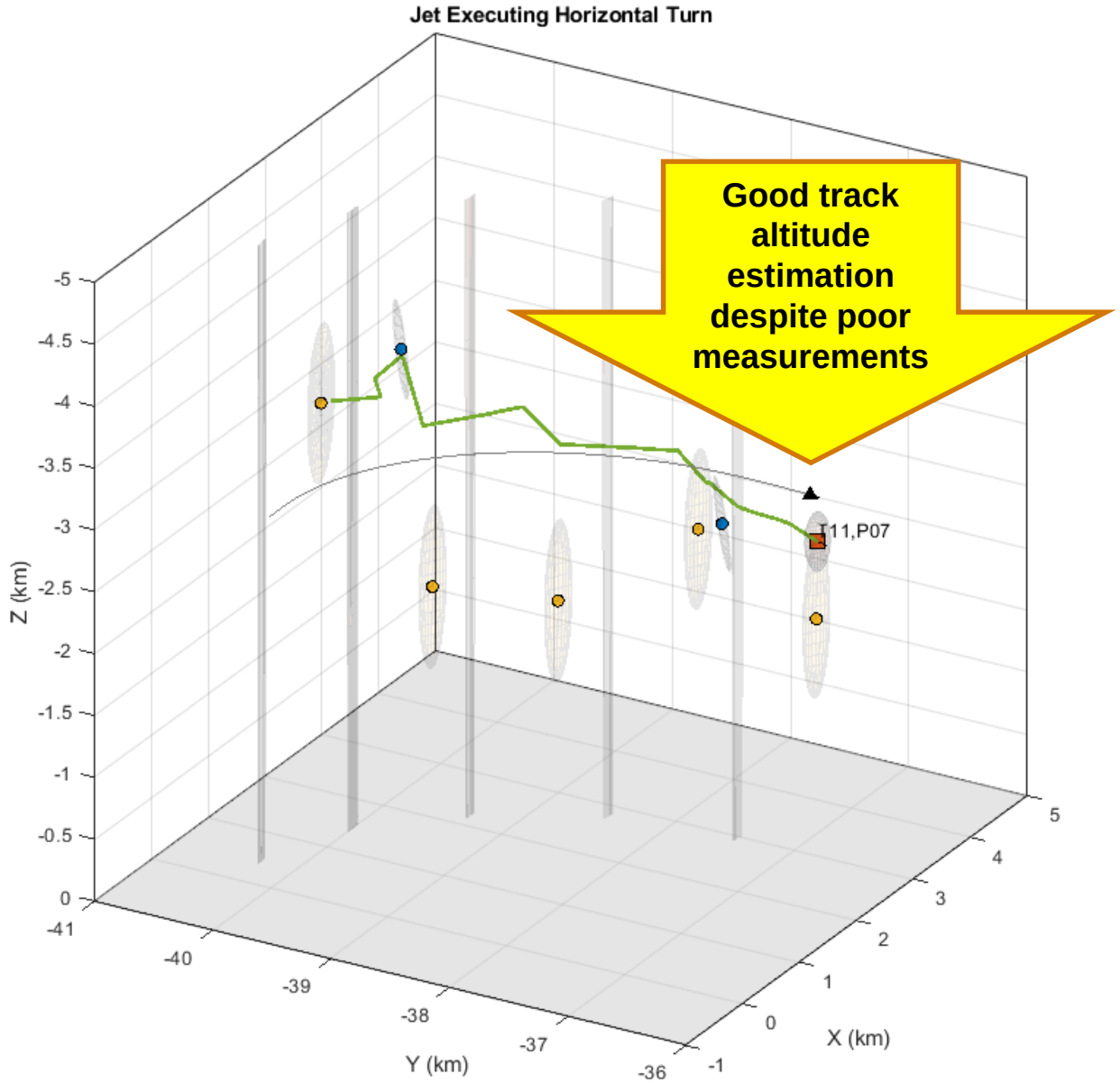
Ellipsoids represent uncertainties

Example here

Multi-platform Scenario Generation

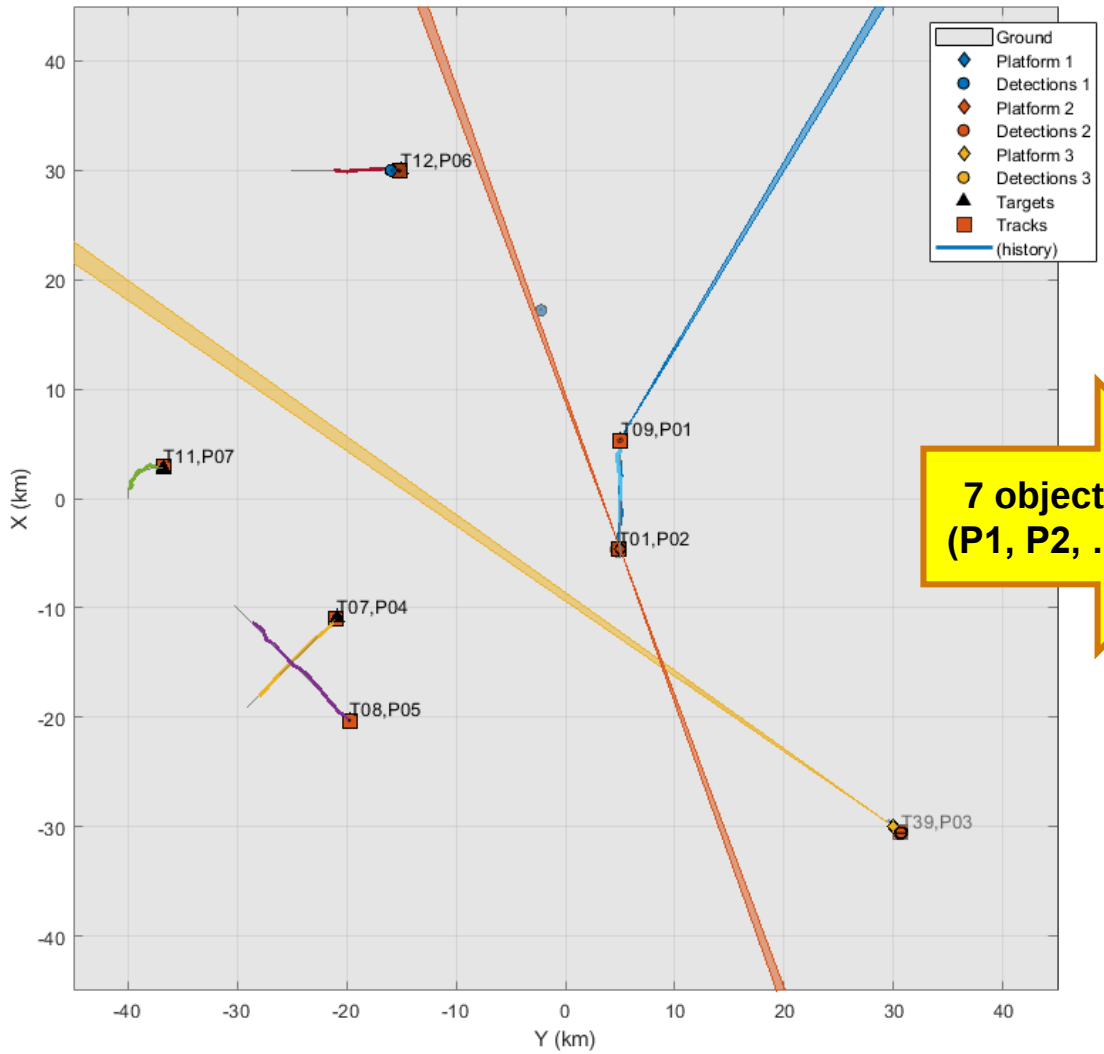
Visualize Track Accuracy



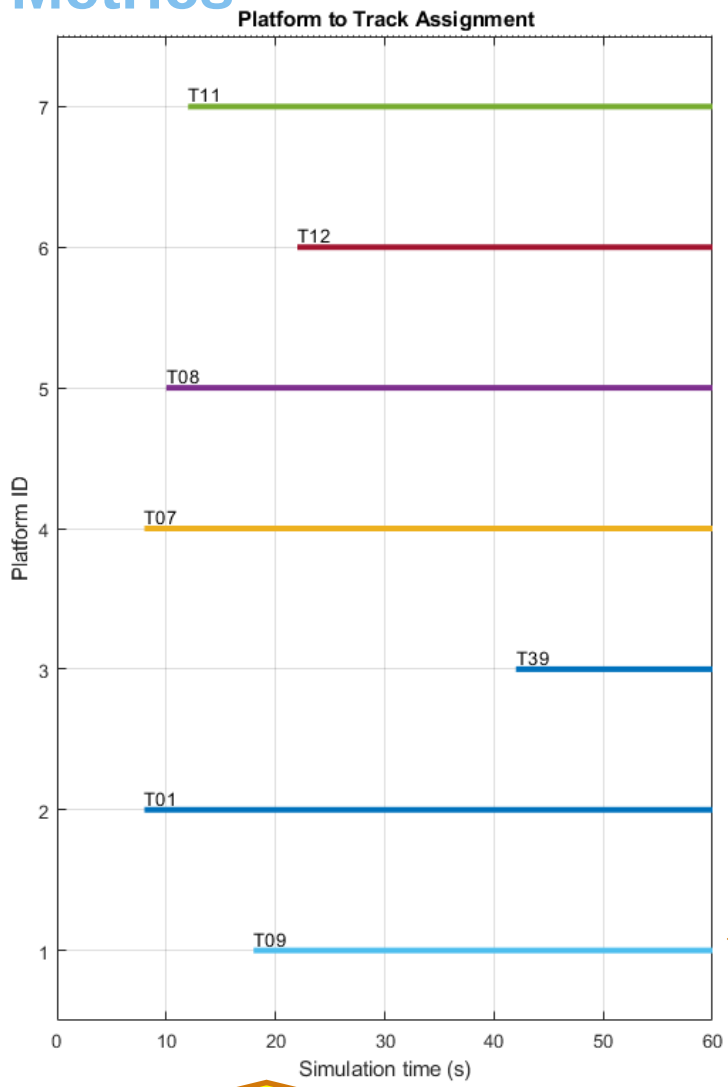


Multi-platform Scenario Generation

Tune and Compare Trackers with Assignment Metrics



7 objects (P1, P2, …)

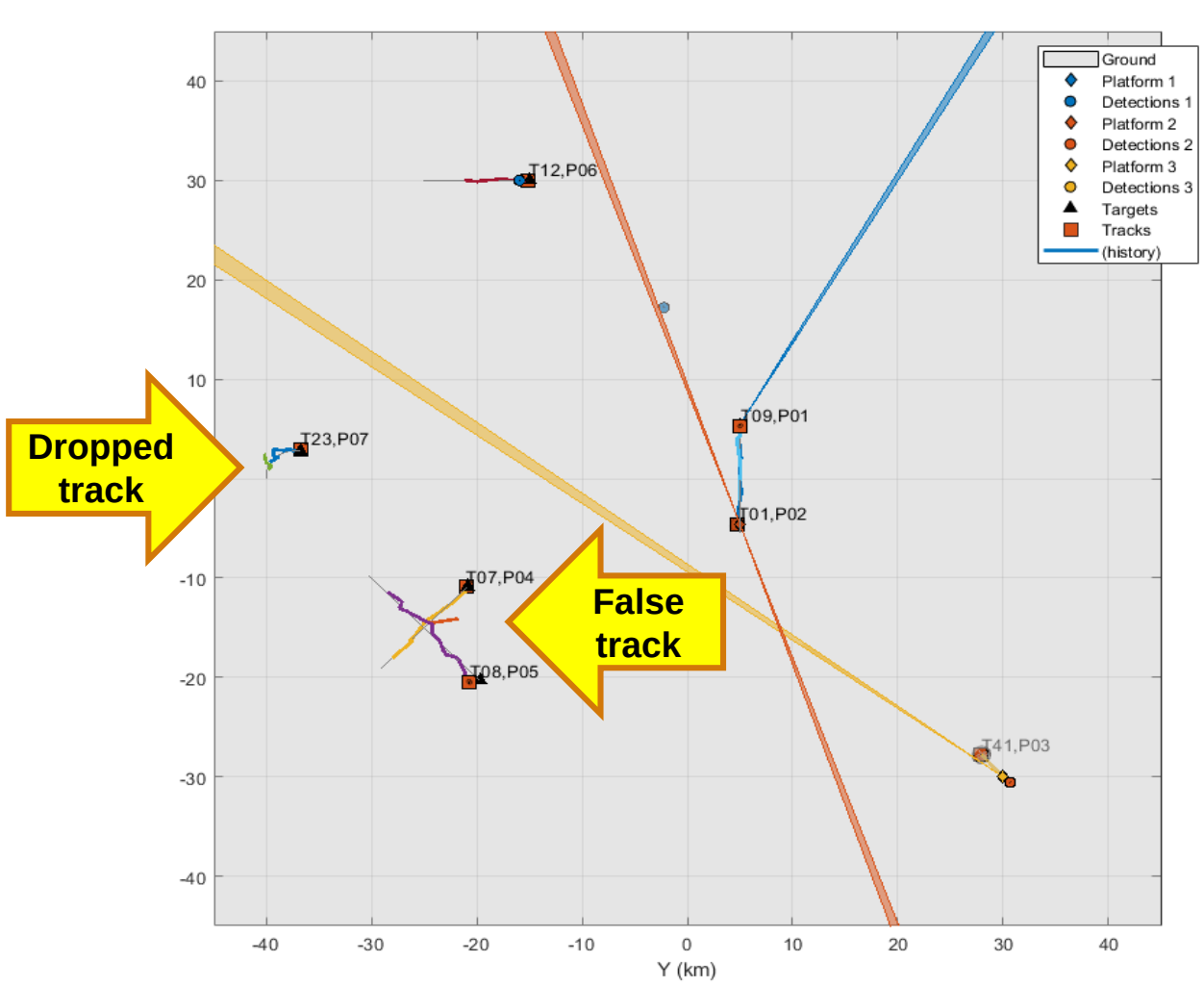


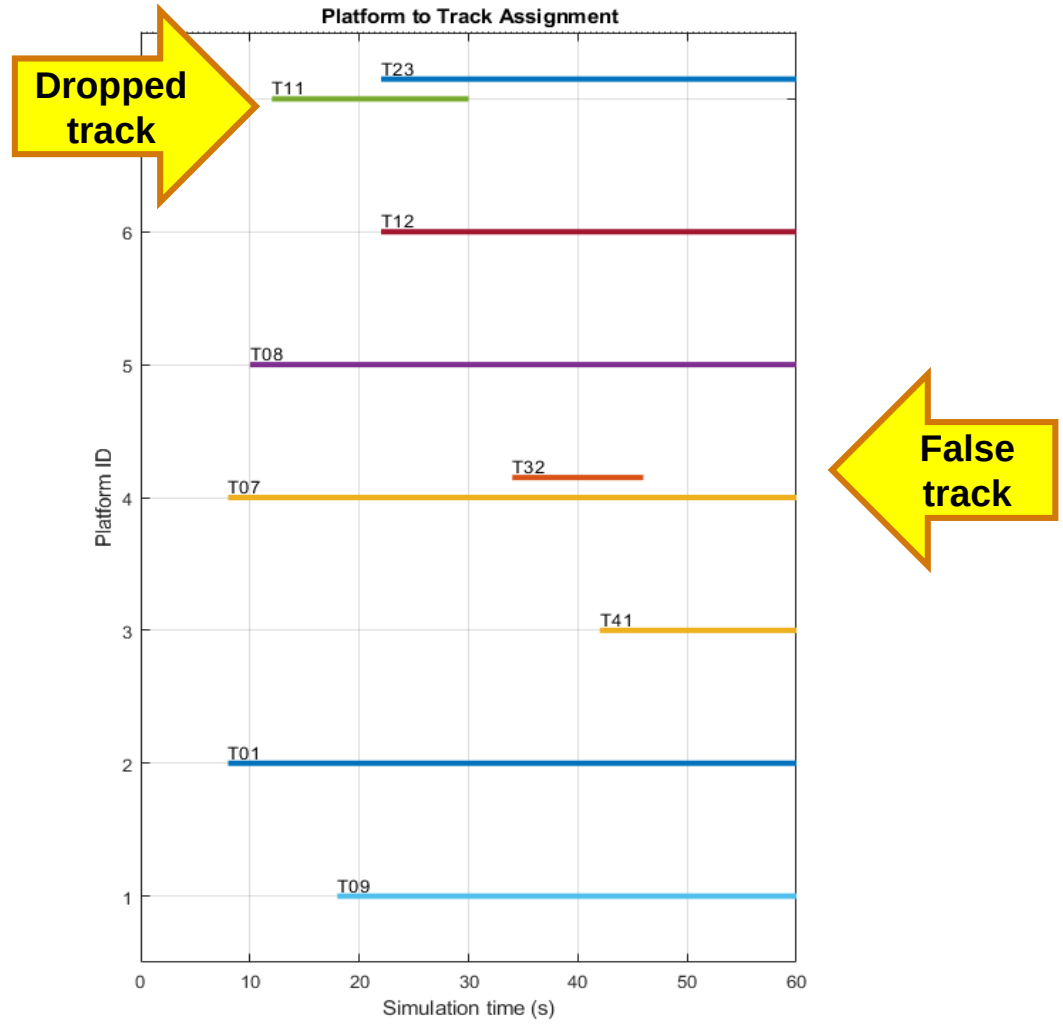
Track T09 for P1

Time to confirm tracks

Multi-platform Scenario Generation

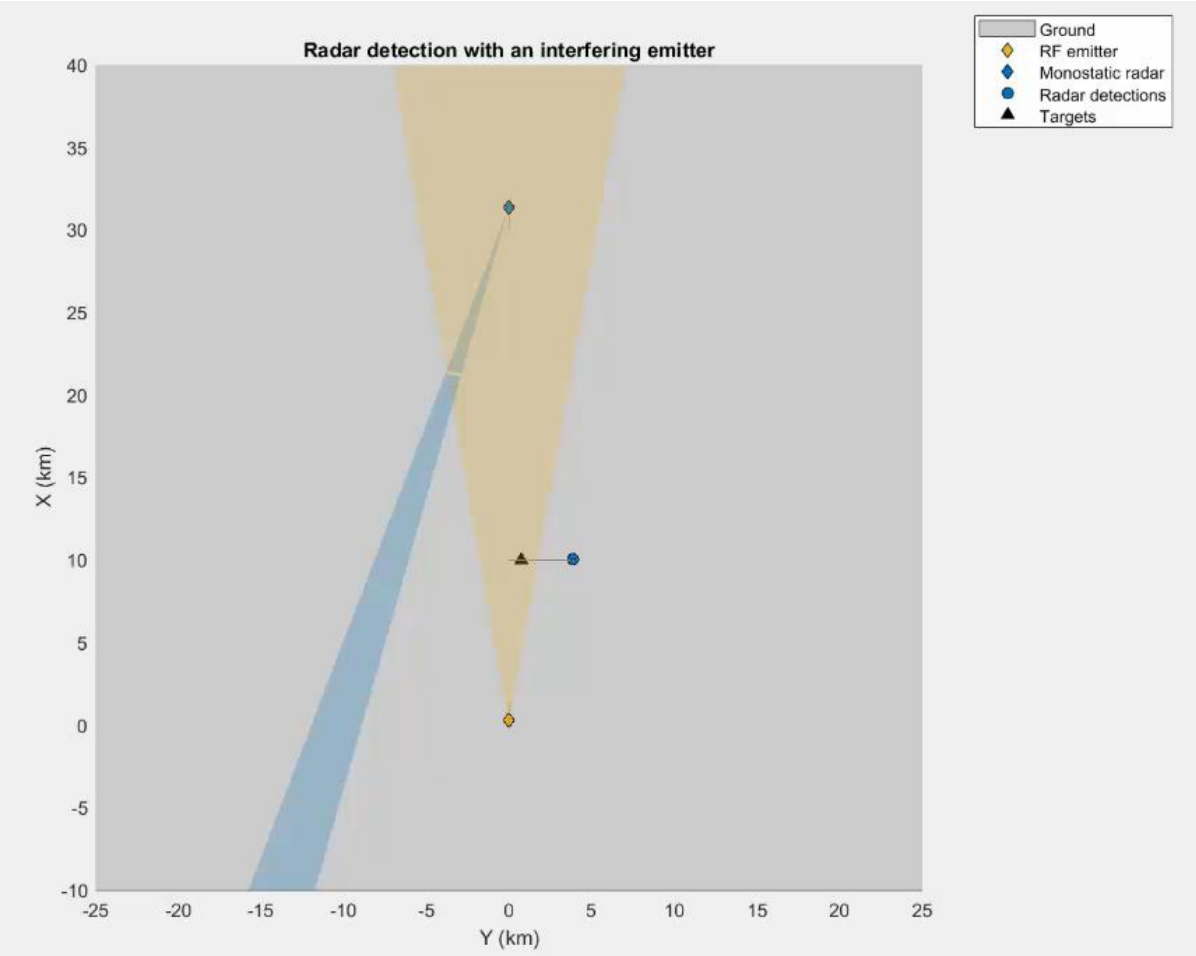
Assess Tracker Performance with Assignment Metrics

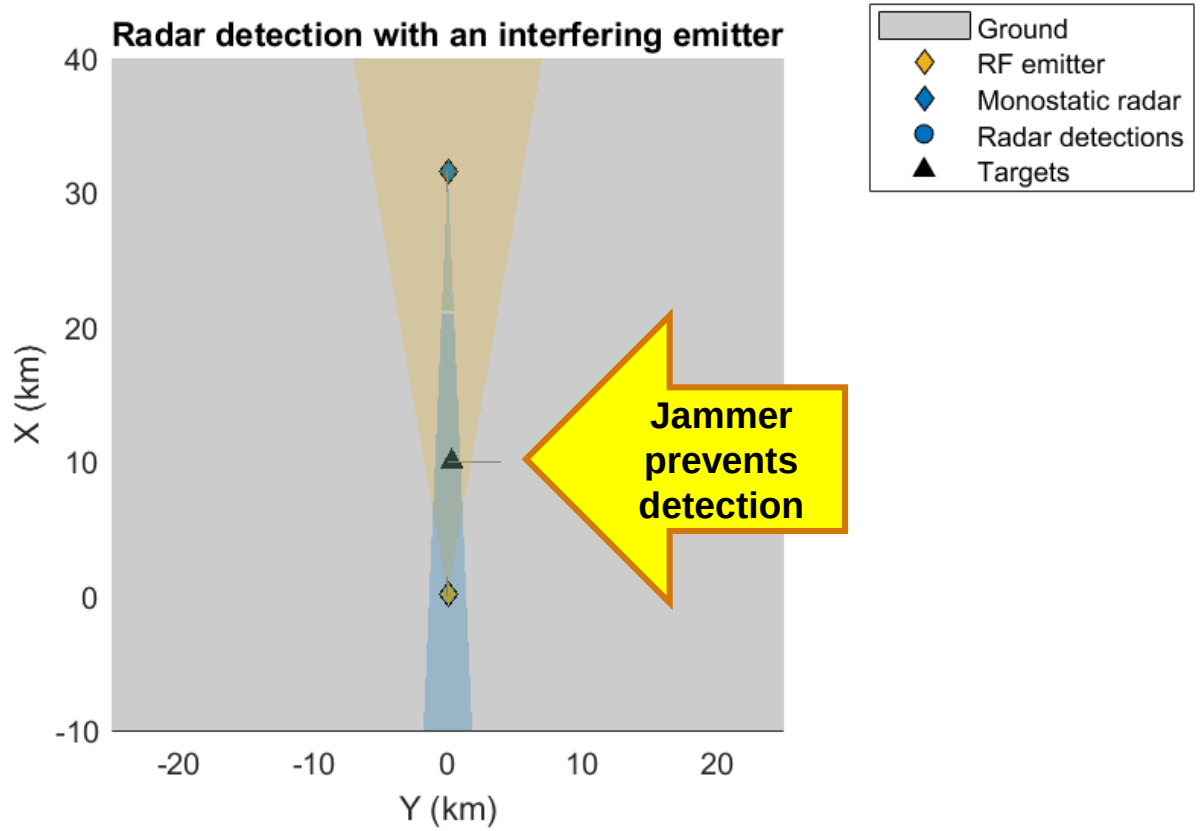




Multi-platform Scenario Generation

Jamming Scenario

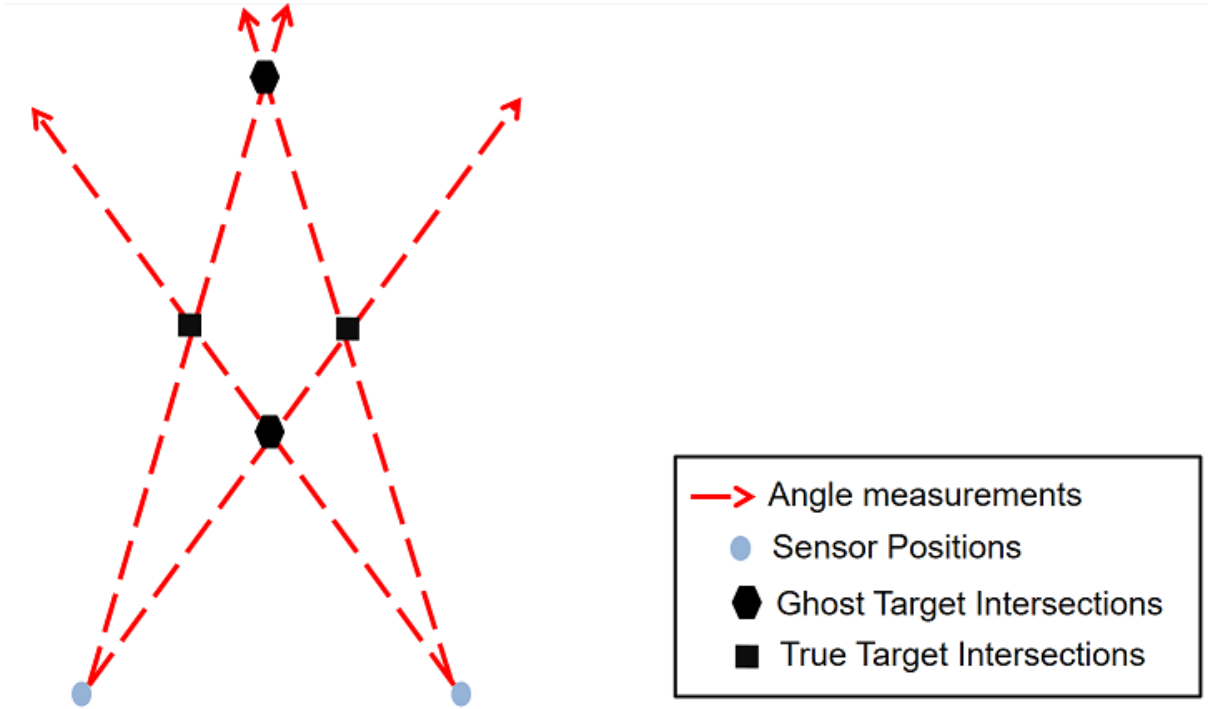


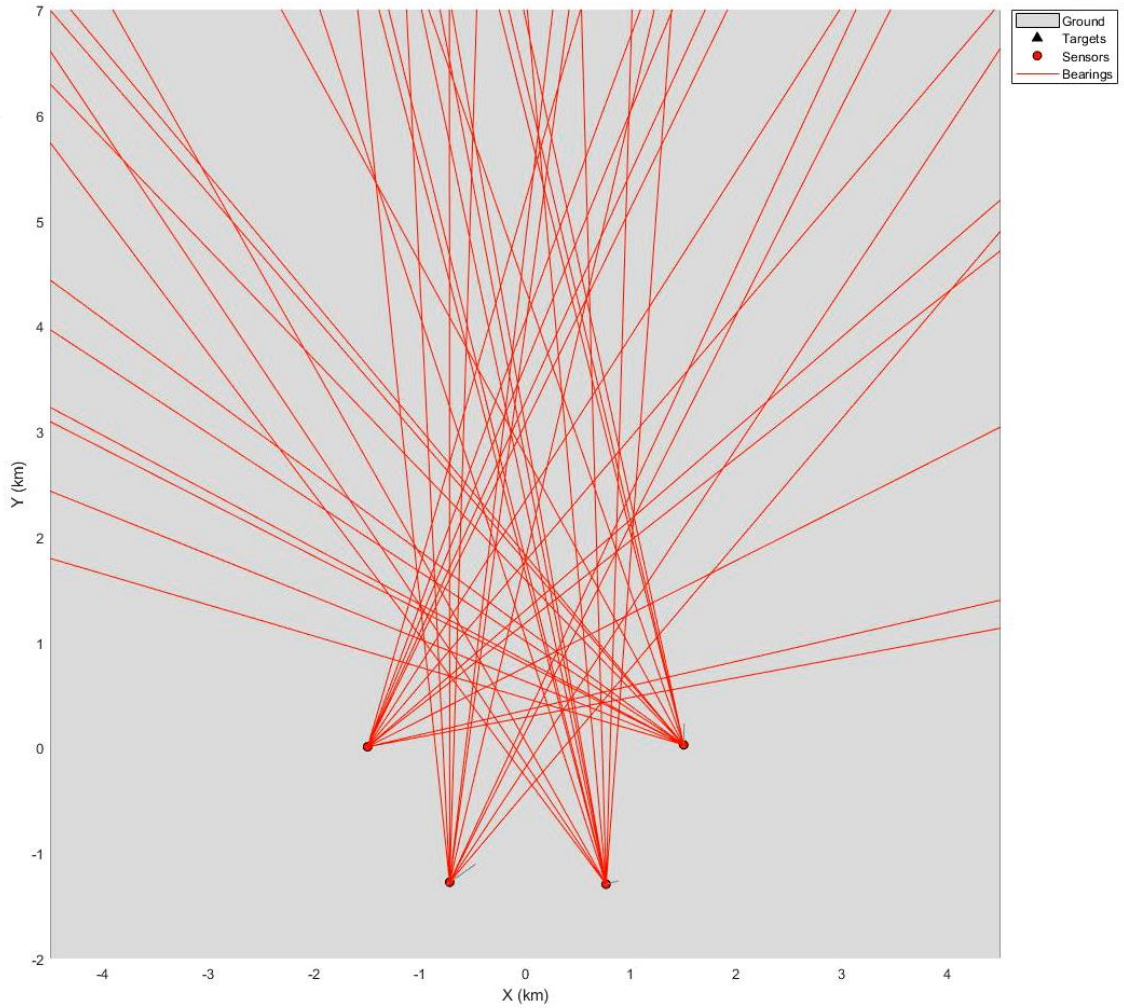


Example here

Multi-platform Scenario Generation
Positioning with direction only measurements

Where are the <u>real</u> targets? How to remove the ghosts?



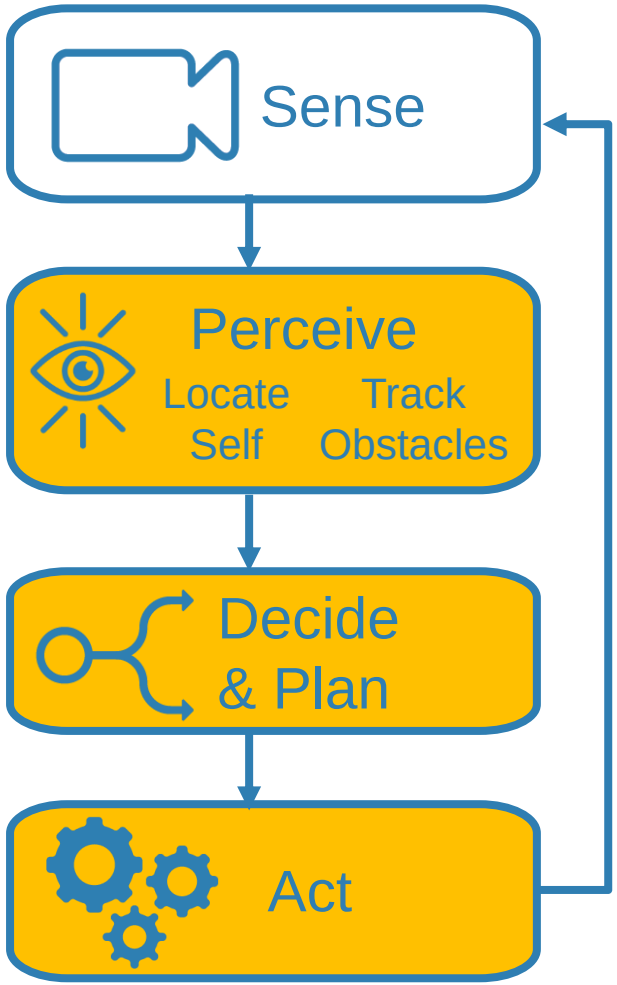


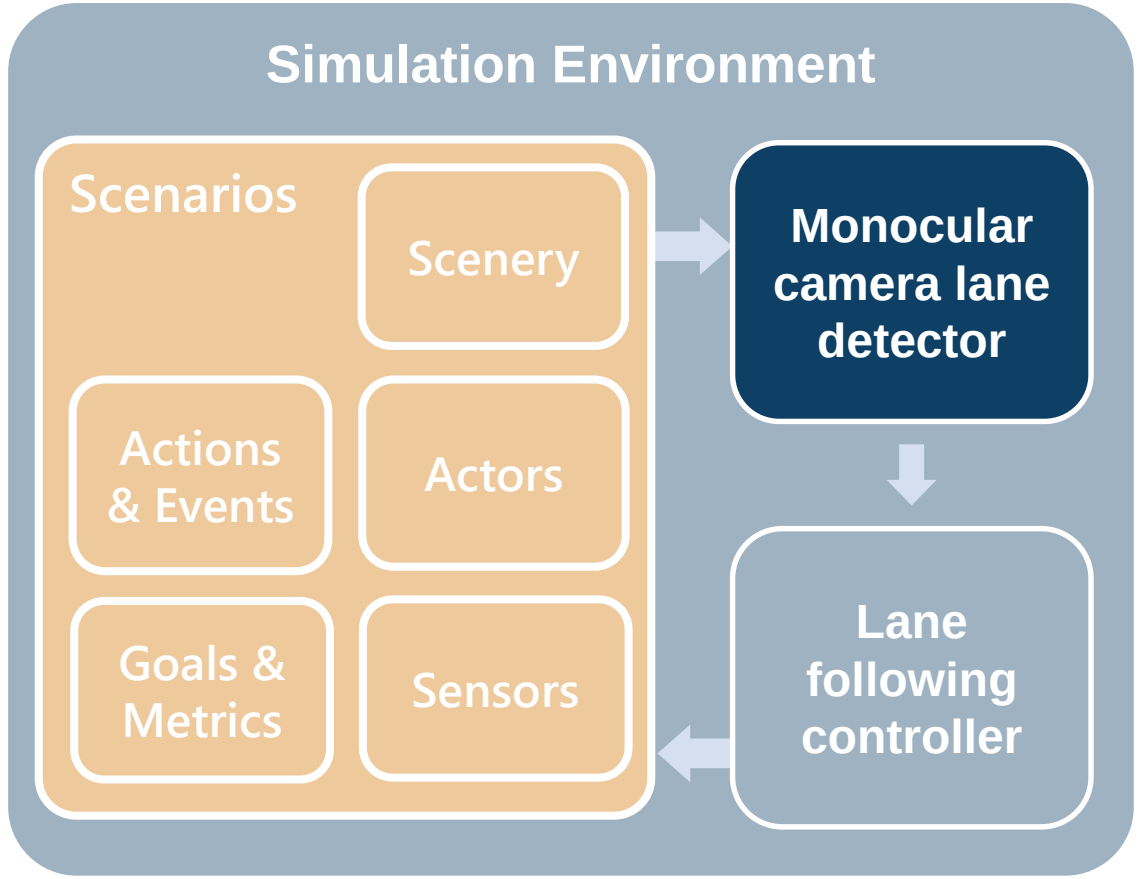
Example here

Agenda

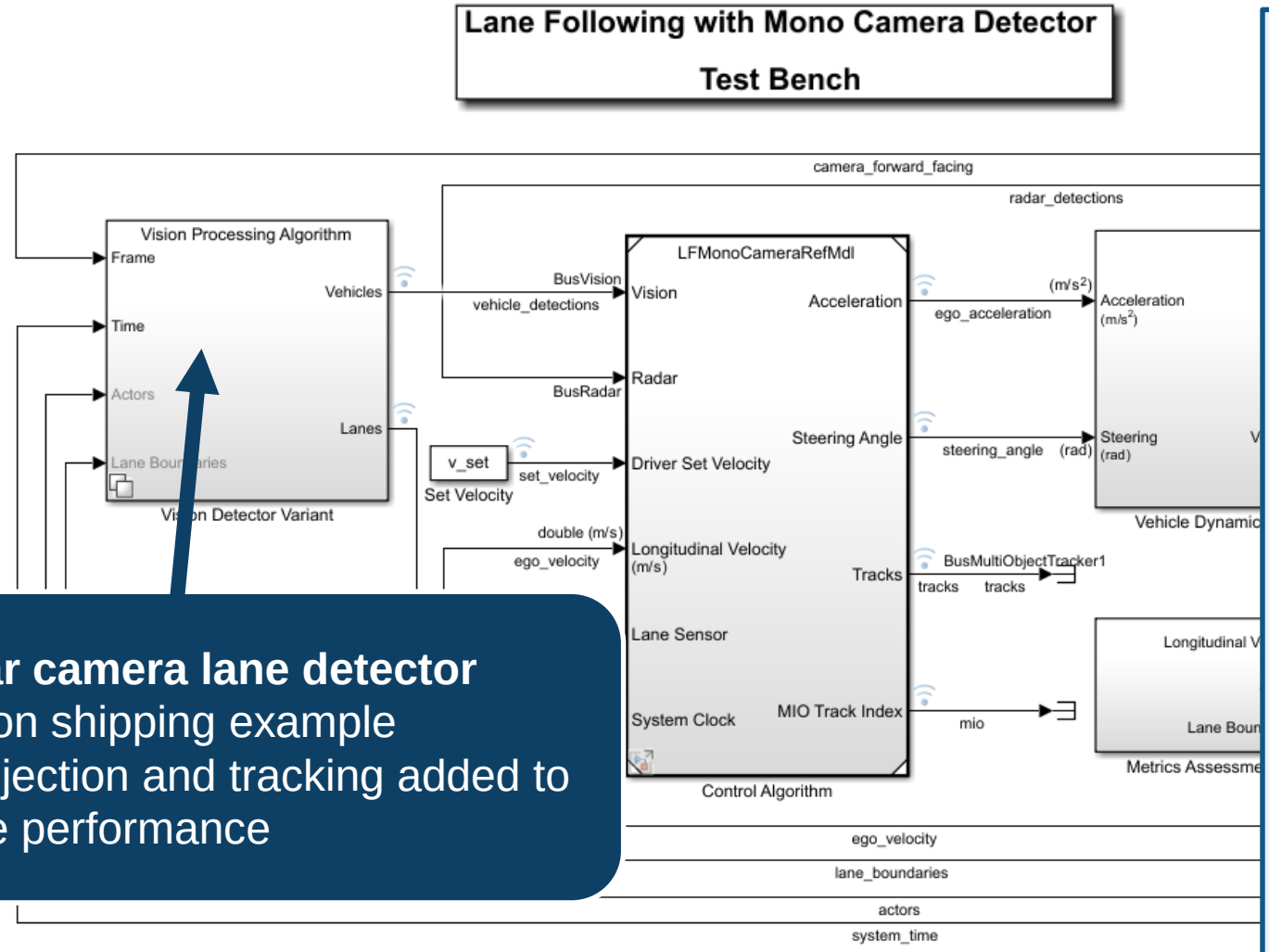
- Introduction
- Technology overview of perception
- Sensor models for sensor fusion and tracking
- Building simulation scenarios
- Developing a Multi-Object Tracker
- Tracking from multiple platforms
- Connecting trackers to a control system
- Q&A

Simulate a lane detection and lane following system





Simulate a lane detection and lane following system



Monocular camera lane detector

- Based on shipping example
- Lane rejection and tracking added to improve performance

helperMonoSensor.m

```
40
41  classdef helperMonoSensor < handle
42
43      properties
44          % Sensitivity for the lane segmentation
45          LaneSegmentationSensitivity = 0.25;
46
47
48
49
50
51
```

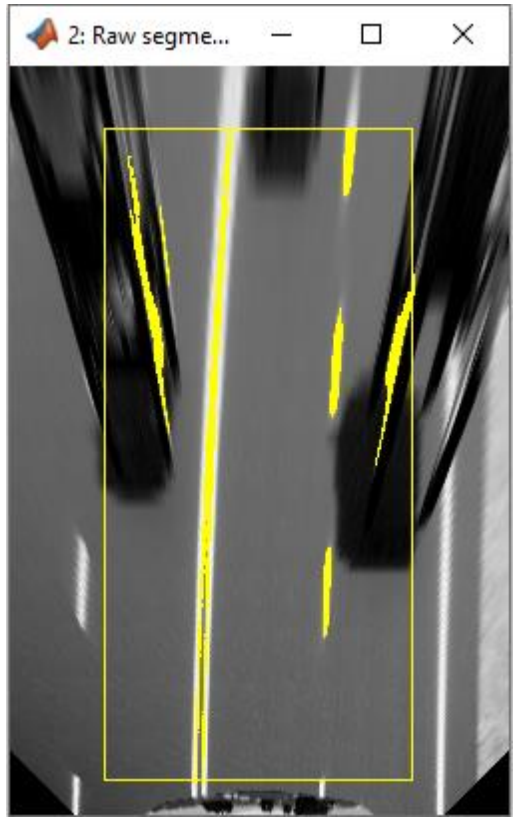
[Visual Perception Using Monocular Camera]

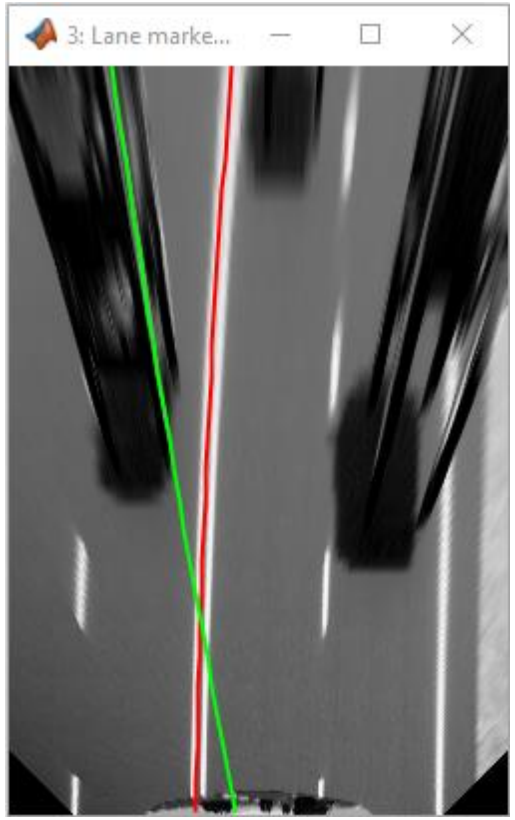
Automated Driving Toolbox™

R2017a

Simulate a lane detection and lane following system

Challenge with noisy lanes

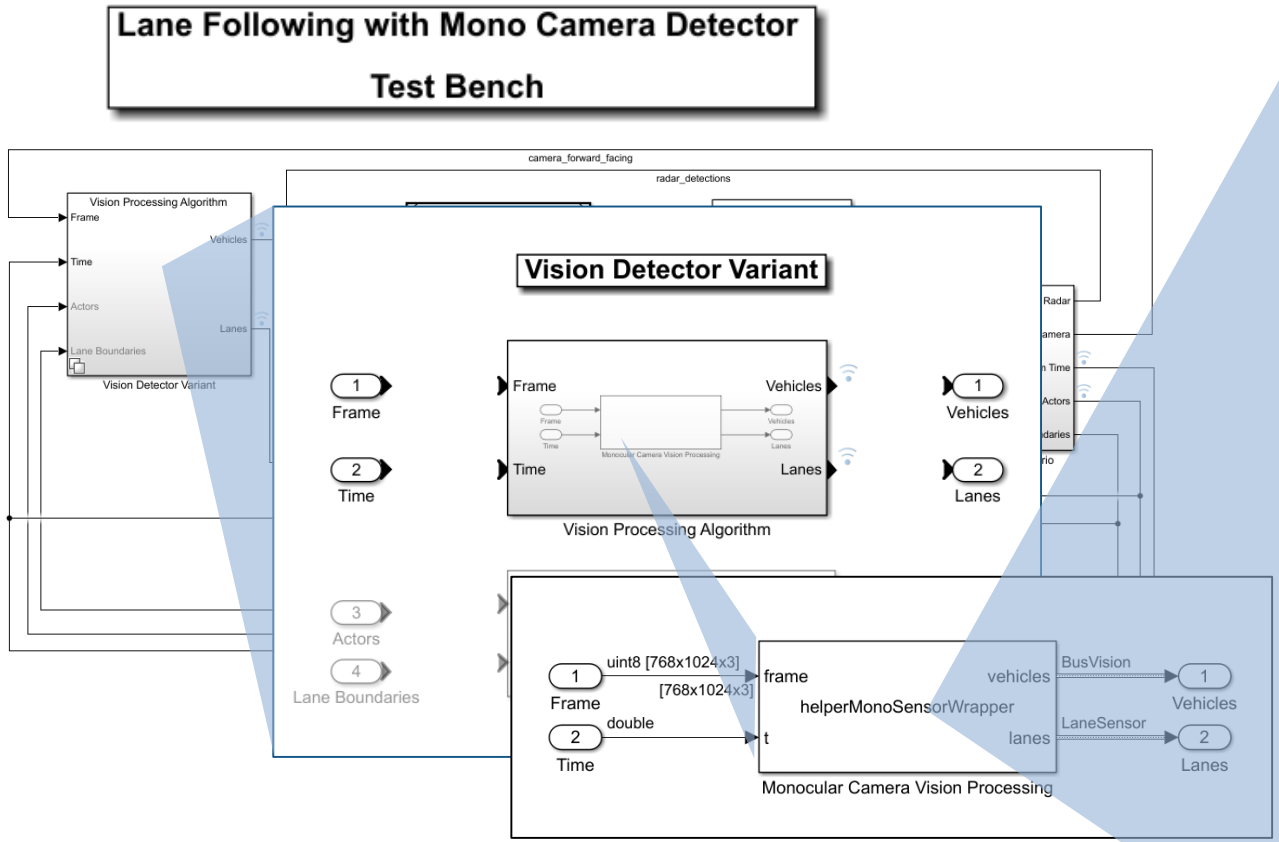


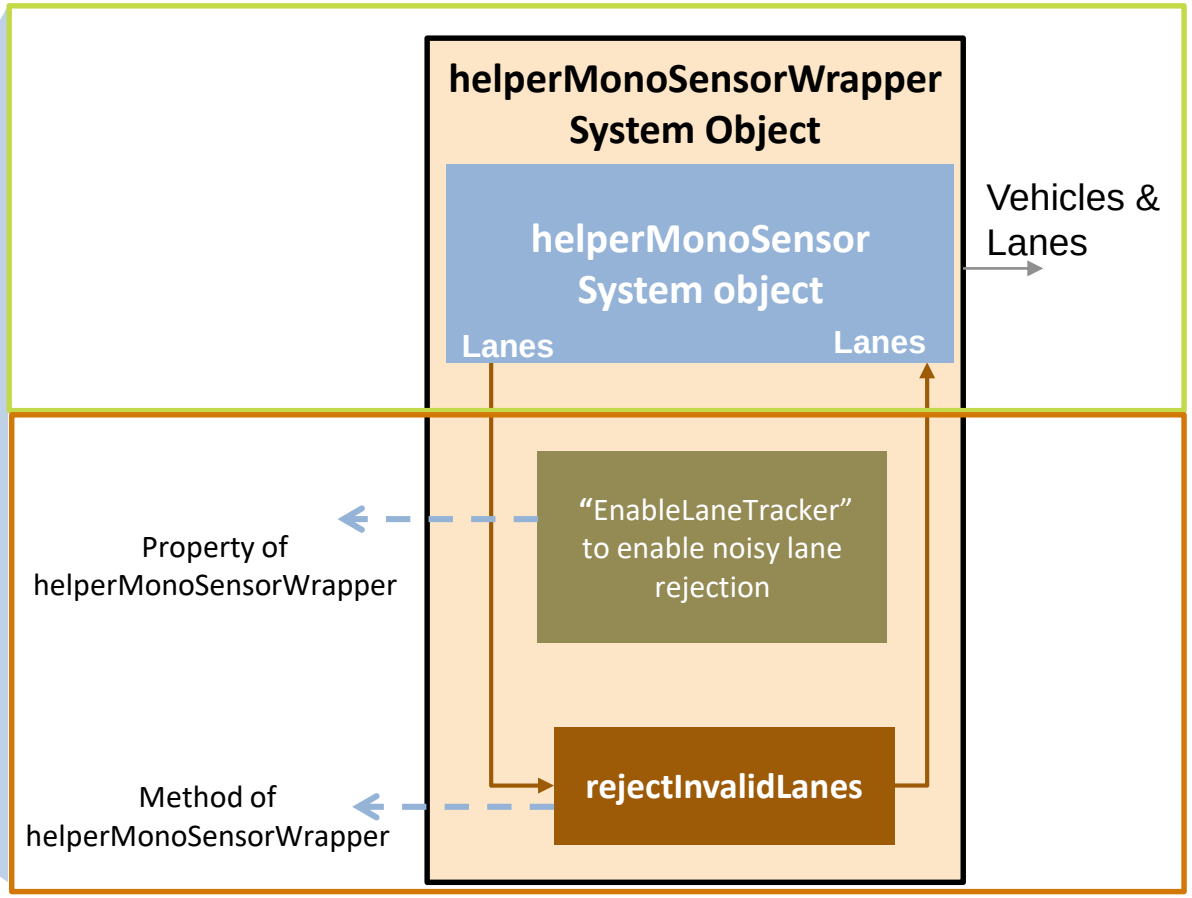


Approaching car gets confused with road

Simulate a lane detection and lane following system

Integrate noisy lane rejection and tracking





Simulate a lane detection and lane following system

Integrate noisy lane rejection and tracking

Example: Before Correction

Example: After Correction

Simulate a lane detection and lane following system

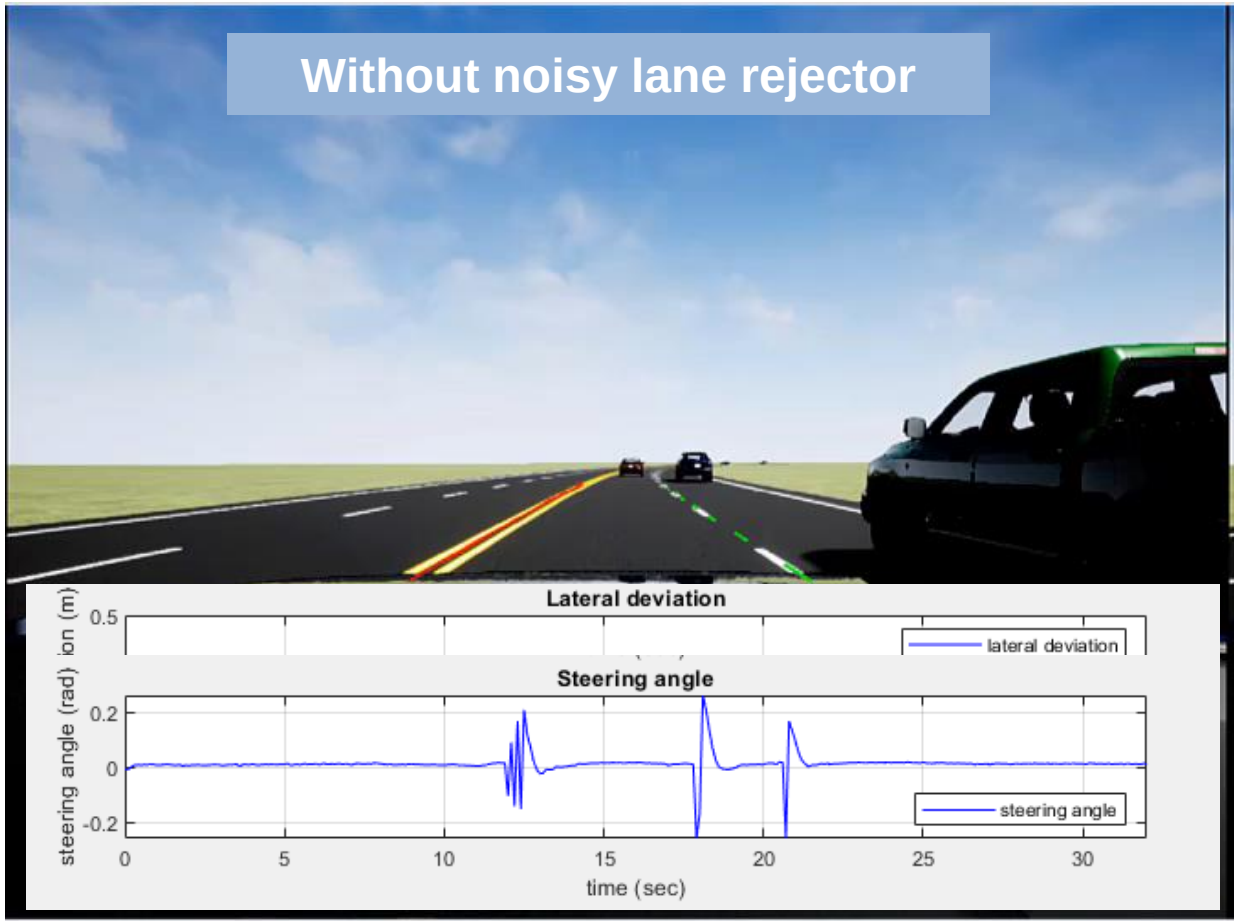
Tracker Setup and Configuration

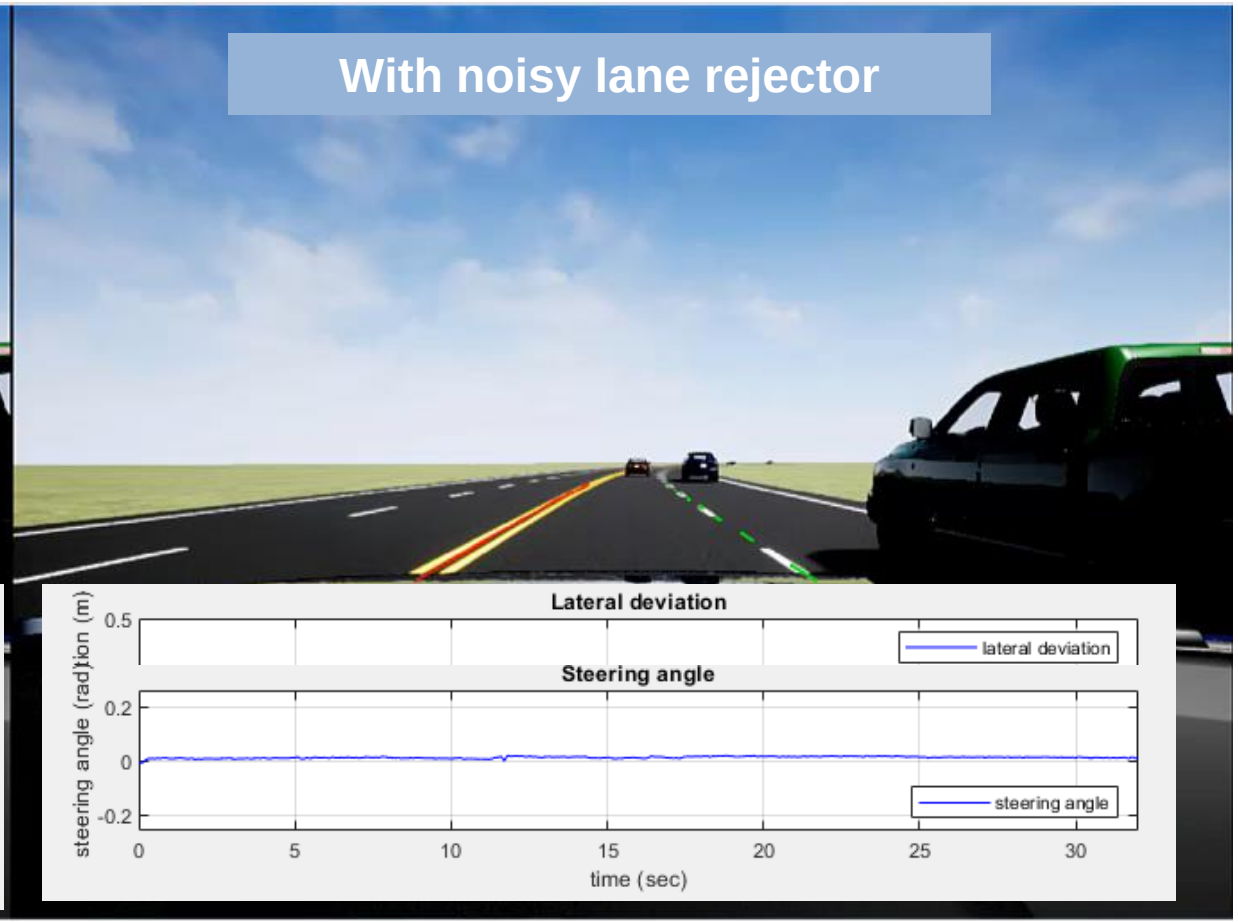
```
leftCur: 1x1 parabolicLaneBoundary =

  parabolicLaneBoundary with properties:

      Parameters: [-6.0503e-04 0.0047 2.0271]
    BoundaryType: Solid
        Strength: 15.6101
         XExtent: [6.9960 28.0080]
           Width: 0
```

- Two instances of Kalman tracker to track left and right lanes independently.
- Initialize trackers for first valid lanes using “ConfigureKalmanFilter”
- Input Parameters for tracking: A,B,C coefficients of parabolic lane boundaries
- Motion model : Constant Acceleration
- Correct tracker for every valid lane

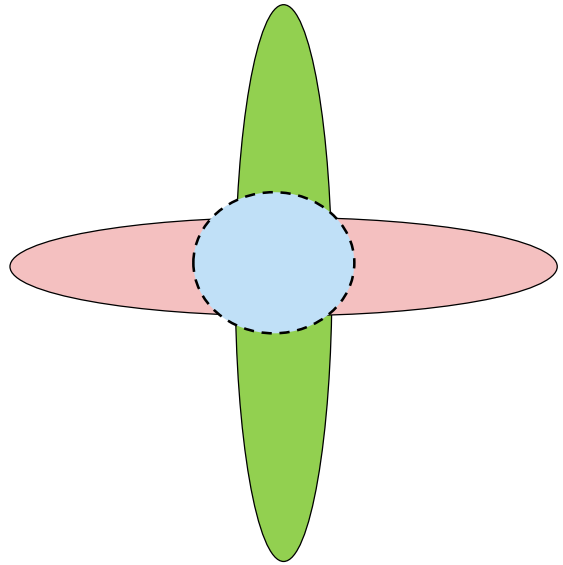
Simulate a lane detection and lane following system

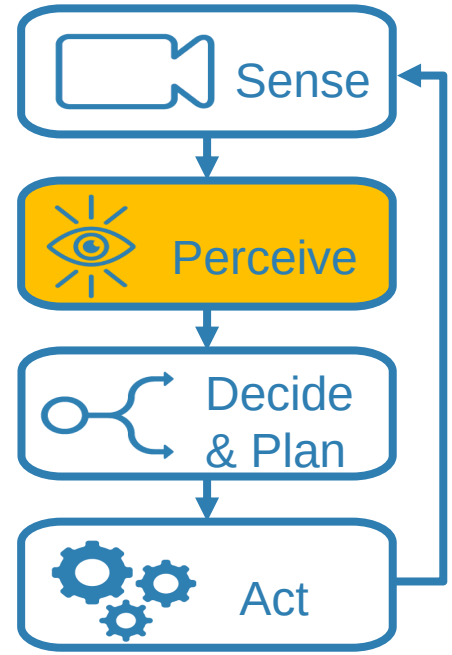
Steering Angle deviation for simulation run





Sensor Fusion and Tracking …





Is Ubiquitous

Leverages Sensor Strengths

Enables Autonomy

Q&A

Marc Willerton (mwillert@mathworks.com)