ROBOTICS AND PROGRAMMING
MEE 3023
PROJECT REPORT

PROJECT TITLE:

AIR TRAFFIC SURVEILLENCE RADAR SYSTEM

DONE BY:

Name:Kaavyaa.R.M
Register Number:19BPS1108

PREPARED FOR:

Programme	:	B.Tech(CPS)	Semester	:	FS 2021-22
Course	:	Robotics and Programming	Code	:	MEE 3023
Faculty	:	Dr.Arockia Selvakumar	Slot	:	G2

NOVEMBER 2021

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

TABLE OF CONTENTS

H.NUMBER	HEADING	PAGE NUMBER
1	Abstract	3
2	Introduction	5
3	Literature Survey	7
4	Methodology	9
5	Analysis	13
6	Conclusion	15
7	References	20

ABSTRACT:

Target tracking is a type of data processing used in surveillance systems to interpret the environment based on observations from one or more sensors. Goal of target tracking system is to partition the observations into sets such that all observations in a set are produced by the same target. These sets are called tracks. By further analysis of observations in a track, various quantities can be computed, including target's current and future kinematic state and targets classification characteristics.

Air surveillance is a type of surveillance focused on scanning the air space to detect and track airborne targets. One of the common sensors used for air surveillance is the air surveillance radar. Typical radar employs directional antenna (which is mechanically rotated at constant rate) to transmit pulsed waveforms and receive echoes reflected from targets. By analyzing received echoes, presence of targets can be detected and observations (called plots in radar jargon) can be "extracted". Plot is a measurement vector containing measurements of time, azimuth and range, at least.

This project shows how to generate an air traffic control scenario, simulate radar detections from an airport surveillance radar (ASR), and configure a global nearest neighbor (GNN) tracker to track the simulated targets using the radar detections. This enables you to evaluate different target scenarios, radar requirements, and tracker configurations without needing access to costly aircraft or equipment.

INTRODUCTION:

Air Traffic Control Radar (ATC-Radar) is the umbrella term for all radar devices used to secure and monitor civil and military air traffic in Air Traffic Management (ATM). They are usually fixed radar systems that have a high degree of specialization. Common applications of air traffic control radars include:

- en-route radar systems,
- Air Surveillance Radar (ASR) systems,
- Precision Approach Radar (PAR) systems,
- surface movement radars, and
- special weather radars.

Airport surveillance radar systems are capable of reliably detecting and tracking aircraft at altitudes below 25,000 feet (7,620 metres) and within 40 to 60 nautical miles (75 to 110 km) of their airport. Systems of this type have been installed at more than 100 major airports throughout the United States. One such system, the ASR-9, is designed to be operable at least 99.9 percent of the time, which means that the system is down less than 10 hours per year. This high availability is attributable to reliable electronic components, a "built-in test" to search for failures, remote monitoring, and redundancy .

A number of radar units can be monitored and controlled from a single location. When trouble occurs, the fault is identified and a maintenance person dispatched for repair.

SYSTEM REQUIREMENTS:

- MATLAB
- Robotics Toolbox
- Navigation toolbox
- Sensor fusion and tracking toolbox

LITERATURE SURVEY:

1.Real-time implementation of target tracking system for air surveillance radar applications

This paper presents a single-sensor multiple-target tracking system applicable with air surveillance radars. We describe the development process along with key details of system's design and implementation, including heuristic process of choosing system parameters to meet tracking performance requirements. In addition, we provide evaluation of system's real-time performance on general purpose computers.We have developed a system which performs real-time target tracking using plots from single radar. Thesystem was called Air Surveillance Radar Tracker (ASRT).

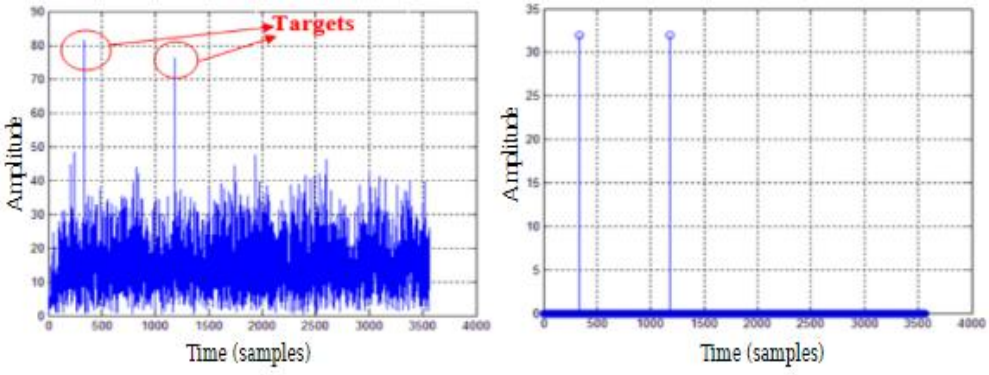
We chose single-hypothesis hard decision approach with optimal solution to the assignment problem. This approach is known as Global Nearest Neighbour (GNN) method. Munkres algorithm was selected for solving the assignment problem. Track Maintenance – For this block, an approach similar to described in was selected. Basically, ad-hoc rules are used – M/N rule for track confirmation, and Nd consecutive missed associations for track termination.

Interacting Multiple Model (IMM) algorithm was chosen, with two or three Kalman filters (KF). All KFs represent state vectors in Cartesian coordinate system, and employ Piecewise Constant White Acceleration Model (PCWAM) to model target dynamics.

To design and implement ASRT software, we used an iterative and incremental process. In design phase, we Combined object-oriented and algorithmic decomposition, resulting in several auxiliary classes with clearly separated responsibilities (e.g. assignment, KF, track number generation), and a central class which is algorithmically decomposed to provide various MTT functionalities. Implementation language is C++, augmented with boost libraries.

2 . An Industrial Design Approach, Implementation, and Application Perspectives for Surveillance Radar Systems

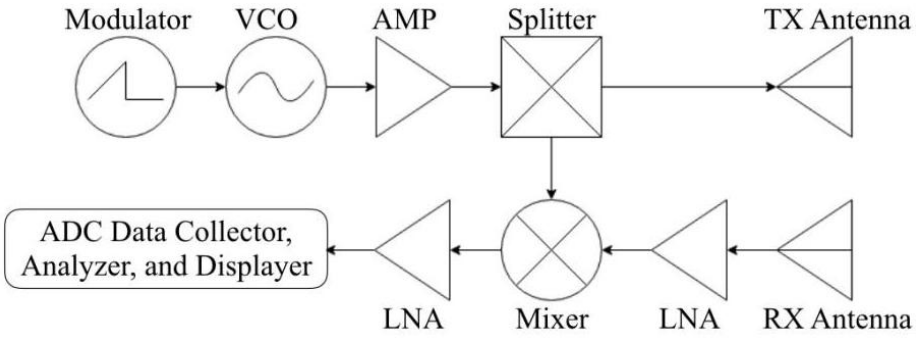
This paper is devoted to demonstrate the most important keys to interface the design and implementation of radar systems with industrial considerations to achieve a competitive radar product. The investigation is focusing on the industrial design ideas for radar systems by formulating the advanced design concepts, system engineering design requirements and disciplinary perspective for radar production. Industrial design is synergic the industrial society, which makes originally isolated disciplines contact and interact each other to form an organic unity
Implementation:
Display implantation is developed. Net framework is written in C# programming language that is very common language for different types of windows applications. PPI Implementation is done in layers architecture. Each layer represents the output software module and all layers are added to generate the final view for the PPI display.



3.Accessible Real-Time Surveillance Radar System for Object Detection

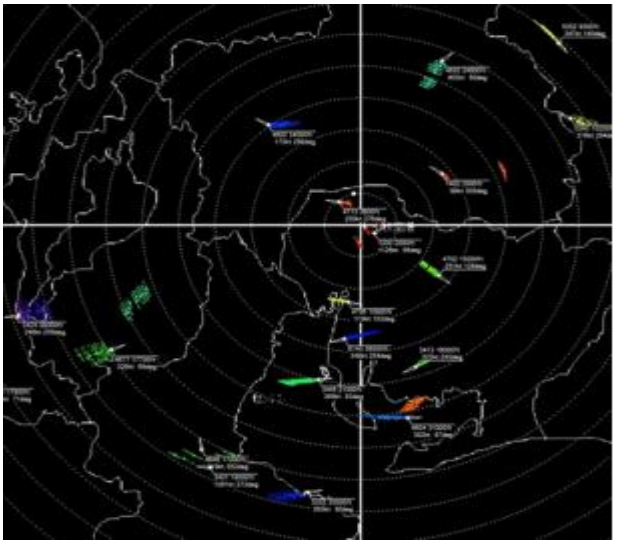
The paper details the iterative process on the system design and improvements with experiments to realize the system used for surveillance. We propose an inexpensive, lighter, safer, and smaller radar system than military-grade radar systems while keeping reasonable capability for use in monitoring public places.

The proposed system adopts basic FMCW radar system, which consists of several components as follows. A modulator that generates voltage within a certain voltage range with a certain interval; a voltage-controlled oscillator (VCO) that generates frequency according to the input voltage generated from the modulator; a splitter that splits the signal generated from VCO and sends the signal to both a transmit (TX) antenna and a mixer; two low-noise amplifiers (LNAs), respectively; placed in each side of the mixer that each amplifies the signal coming from the previous stage; a mixer that mixes the amplified reflected signal with the transmitted signal received from the splitter; and two directional antennas to transmit and receive signal.



4. DEVELOPMENT OF PASSIVE SURVEILLANCE RADAR

The passive surveillance radar referred to in this paper is equipment or a system for the purpose of monitoring positions of aircraft cruising in air space and taxing on the surface of an airport. The passive radar has a function of detecting the position of target objects, such as aircraft, having a reflective cross-sectional area above a certain level without any radio wave transmissions.In the latest models in 2014, the PSSR provided by the IRT Corp., has adopted a system configuration that uses a USB connection between the signal processing unit and the host PC.After connecting antennae to the PSSR Receiver & Processor, it is possible to obtain airspace monitoring information. The PSSR (#PRIRT1- PF-001A PSSR) is smaller than A4 size, at a 250mm width, 170mm depth, and 65mm thickness, and a power consumption of 3W or less.Airspace monitoring coverage of the PSSR is about more than 200km, and the PSSR can be operated with any conventional SSR Mode-A/C as the master SSR, including the SSR which is installed temporarily for some kinds of operations, such as disaster drills, by SelfDefense Forces, and so on. Figure 5 shows the trajectories of helicopters observed during sea disaster prevention training, which was conducted in the Seto-Naikai Sea in October 2012.



5. IMPROVEMENTS FOR AIR SURVEILLENCE RADAR

Both a near-term and a far-term radar concept for improved air-surveillance are described. The near-term concept is based on operating simultaneously with multiple frequencies over two radar bands to obtain the benefits of more uniform coverage in elevation, improved target detection, improved automatic tracking, target height-finding without the need for a 3D antenna, elementary target recognition, and other attributes that result from operating over a very wide band. The far-term concept is based on operating an air-surveillance radar with a very wide bandwidth as in the near-term approach, but with an architecture quite different from that used in current air-surveillance radars. This radar looks everywhere all the time with a number of fixed directive receiving beams and an omnidirectional transmitting beam. The spatial, temporal, and spectral domains are used along with digital beamforming and digital signal processing to allow this ubiquitous radar to perform multiple functions in parallel with variable data rates and without time sharing so as to achieve simultaneous weapon control and surveillance, reduced susceptibility to intercept, and other capabilities difficult to obtain with conventional analog beamforming methods.

METHODOLOGY:

Air Traffic Scenario

```
% Creating tracking scenario
scenario = trackingScenario;
% Adding a stationary platform to model the ATC tower
tower = platform(scenario);
```

Airport Surveillance Radar

Adding an airport surveillance radar (ASR) to the ATC tower. An ATC tower has a radar mounted 15 meters above the ground. This radar scans mechanically in azimuth at a fixed rate to provide 360 degree coverage in the vicinity of the ATC tower. Common specifications for an ASR are listed:

Sensitivity: 0 dBsm @ 111 km

Mechanical Scan: Azimuth only

Mechanical Scan Rate: 12.5 RPM

Electronic Scan: None

Field of View: 1.4 deg azimuth, 10 deg elevation

Azimuth Resolution: 1.4 deg

Range Resolution: 135 m

```
rpm = 12.5;
fov = [1.4;10];
scanrate = rpm*360/60;  % deg/s
updaterate = scanrate/fov(1); % Hz

radar = fusionRadarSensor(1,'Rotator', ...
'UpdateRate', updaterate, ...        % Hz
'FieldOfView', fov, ...              % [az;el] deg
```

```
'MaxAzimuthScanRate', scanrate, ...    % deg/sec
'AzimuthResolution', fov(1), ...       % deg
'ReferenceRange', 111e3, ...           % m
'ReferenceRCS', 0, ...                 % dBsm
'RangeResolution', 135, ...            % m
'HasINS', true, ...
'DetectionCoordinates', 'Scenario');
% Mount radar at the top of the tower
radar.MountingLocation = [0 0 -15];
tower.Sensors = radar;
```

Enabling elevation and setting the mechanical scan limits to span the radar ' s elevation field of view beginning at 2 degrees above the horizon.

```
% Enabling elevation scanning
radar.HasElevation = true;
% Setting mechanical elevation scan to begin at 2 degrees above the horizon
elFov = fov(2);
tilt = 2; % deg
radar.MechanicalElevationLimits = [-fov(2) 0]-tilt;
```

Set the elevation field of view to be slightly larger than the elevation spanned by the scan limits. This prevents raster scanning in elevation and tilts the radar to point in the middle of the elevation scan limits.

```
radar.FieldOfView(2) = elFov+1e-3;
```

The fusionRadarSensor models range and elevation bias due to atmospheric refraction. These biases become more pronounced at lower altitudes and for targets at long ranges. Because the index of refraction changes (decreases) with altitude, the radar signals propagate along a curved path. This results in the radar observing

targets at altitudes which are higher than their true altitude and at ranges beyond their line-of-sight range.

Add three airliners within the ATC control sector. One airliner approaches the ATC from a long range, another departs, and the third is flying tangential to the tower. Model the motion of these airliners over a 60 second interval.

TrackingScenario uses a North-East-Down (NED) coordinate frame. When defining the waypoints for the airliners below, the z-coordinate corresponds to down, so heights above the ground are set to negative values.

```
% Duration of scenario

sceneDuration = 60; % s

% Inbound airliner

ht = 3e3;

spd = 900*1e3/3600; % m/s

wp = [-5e3 -40e3 -ht;-5e3 -40e3+spd*sceneDuration -ht];

traj = waypointTrajectory('Waypoints',wp,'TimeOfArrival',[0
sceneDuration]);

platform(scenario,'Trajectory', traj);

% Outbound airliner

ht = 4e3;

spd = 700*1e3/3600; % m/s

wp = [20e3 10e3 -ht;20e3+spd*sceneDuration 10e3 -ht];

traj = waypointTrajectory('Waypoints',wp,'TimeOfArrival',[0
sceneDuration]);

platform(scenario,'Trajectory', traj);

% Tangential airliner
```

```
ht = 4e3;

spd = 300*1e3/3600; % m/s

wp = [-20e3 -spd*sceneDuration/2 -ht;-20e3 spd*sceneDuration/2 -ht];

traj = waypointTrajectory('Waypoints',wp,'TimeOfArrival',[0 sceneDuration]);

platform(scenario,'Trajectory', traj);
```

GNN Tracker

Create a trackerGNN to form tracks from the radar detections generated from the three airliners. Update the tracker with the detections generated after the completion of a full 360 degree scan in azimuth.

The tracker uses the initFilter supporting function to initialize a constant velocity extended Kalman filter for each new track. initFilter modifies the filter returned by initcvekf to match the target velocities and tracker update interval.

```
tracker = trackerGNN( ...

'Assignment', 'Auction', ...

'AssignmentThreshold',50, ...

'FilterInitializationFcn',@initFilter);
```

Visualize on a Map

You use trackingGlobeViewer to visualize the results on top of a map display. You position the origin of the local North-East-Down (NED) coordinate system used by the tower radar and tracker at the position of Logan airport in Boston. The origin is located at 42.36306 latitude and -71.00639 longitude and 50 meters above the sea level.

```
origin = [42.366978, -71.022362, 50];
```

```
mapViewer = trackingGlobeViewer('ReferenceLocation',origin,...
'Basemap','streets-dark');
campos(mapViewer, origin + [0 0 1e5]);
drawnow;
plotScenario(mapViewer,scenario);
snapshot(mapViewer);
```

ANALYSIS:

Simulate and Track Airliners

The following loop advances the platform positions until the end of the scenario has been reached. For each step forward in the scenario, the radar generates detections from targets in its field of view. The tracker is updated with these detections after the radar has completed a 360 degree scan in azimuth.

```
% Set simulation to advance at the update rate of the radar
scenario.UpdateRate = radar.UpdateRate;
% Create a buffer to collect the detections from a full scan of the
radar
scanBuffer = {};
% Initialize the track array
tracks = objectTrack.empty;
% Save visualization snapshots for each scan
allsnaps = {};
scanCount = 0;
% Set random seed for repeatable results
```

```
s = rng;
rng(2020)
while advance(scenario)

% Update airliner positions
plotPlatform(mapViewer, scenario.Platforms([2 3 4]), 'TrajectoryMode','Full');

% Generate detections on targets in the radar's current field of view
[dets,config] = detect(scenario);
scanBuffer = [scanBuffer;dets]; %#ok<AGROW>
% Plot beam and detections
plotCoverage(mapViewer,coverageConfig(scenario))
plotDetection(mapViewer,scanBuffer);

% Update tracks when a 360 degree scan is complete
simTime = scenario.SimulationTime;
isScanDone = config.IsScanDone;
if isScanDone
scanCount = scanCount+1;
% Update tracker
[tracks,~,~,info] = tracker(scanBuffer,simTime);
% Clear scan buffer for next scan
```

```
scanBuffer = {};
elseif isLocked(tracker)
% Predict tracks to the current simulation time
tracks = predictTracksToTime(tracker,'confirmed',simTime);
end


% Update map and take snapshots
allsnaps = snapPlotTrack(mapViewer,tracks,isScanDone,
scanCount, allsnaps);
end
allsnaps = [allsnaps, {snapshot(mapViewer)}];
```

Show the first snapshot taken at the completion of the radar's second scan.

```
figure
imshow(allsnaps{1});
```

The preceding figure shows the scenario at the end of the radar's second 360 degree scan. Radar detections, shown as light blue dots, are present for each of the simulated airliners. At this point, the tracker has already been updated by one complete scan of the radar. Internally, the tracker has initialized tracks for each of the airliners. These tracks will be shown after the update following this scan, when the tracks are promoted to confirmed, meeting the tracker's confirmation requirement of 2 hits out of 3 updates.

The next two snapshots show tracking of the outbound airliner.

```
figure
imshow(allsnaps{2});
figure
```

```
imshow(allsnaps{3});
```

The previous figures show the track picture before and immediately after the tracker updates after the radar's second scan. The detection in the figure before the tracker update is used to update and confirm the initialized track from the previous scan's detection for this airliner. The next figure shows the confirmed track's position and velocity. The uncertainty of the track's position estimate is shown as the yellow ellipse. After only two detections, the tracker has established an accurate estimate of the outbound airliner's position and velocity. The airliner's true altitude is 4 km and it is traveling north at 700 km/hr.

```
figure
imshow(allsnaps{4});
figure
imshow(allsnaps{5});
```

The state of the outbound airliner's track is coasted to the end of the third scan and shown in the figure above along with the most recent detection for the airliner. Notice how the track's uncertainty has grown since it was updated in the previous figure. The track after it has been updated with the detection is shown in the next figure. You notice that the uncertainty of the track position is reduced after the update. The track uncertainty grows between updates and is reduced whenever it is updated with a new measurement. You also observe that after the third update, the track lies on top of the airliner's true position.

```
figure
imshow(allsnaps{6});
```

The final figure shows the state of all three airliners' tracks at the end of the scenario. There is exactly one track for each of the

three airliners. The same track numbers are assigned to each of the airliners for the entire duration of the scenario, indicating that none of these tracks were dropped during the scenario. The estimated tracks closely match the true position and velocity of the airliners.

```
truthTrackTable = tabulateData(scenario, tracks) %#ok<NOPTS>
```

truthTrackTable=3 × 4 table

TrackID	Altitude		Heading		Speed	
	True	Estimated	True	Estimated	True	Estimated
"T1"	4000	4051	90	90	700	710
"T2"	4000	4070	0	359	300	300
"T3"	3000	3057	0	359	900	908

Visualize tracks in 3D to get a better sense of the estimated altitudes.

```
% Reposition and orient the camera to show the 3-D nature of the map

camPosition = origin + [0.367, 0.495, 1.5e4];

camOrientation = [235, -17, 0]; % Looking south west, 17 degrees below the horizon

campos(mapViewer, camPosition);

camorient(mapViewer, camOrientation);

drawnow
```

The figure below shows a 3-D map of the scenario. You can see the simulated jets in white triangles with their trajectories depicted as white lines. The radar beam is shown as a blue cone with blue dots representing radar detections. The tracks are shown in yellow, orange, and blue and their information is listed in their respective color. Due to the nature of the 3-D display, some of the markers may be hidden behind others.

You can use the following controls on the map to get different views of the scene:

To pan the map, you left click on the mouse and drag the map.

To rotate the map, while holding the ctrl button, you left click on the mouse and drag the map.

To zoom the map in and out, you use the mouse scroll wheel.

```
snapshot(mapViewer);
```

SUPPORTING FUNCTIONS USED:

initFilter

This function modifies the function initcvekf to handle higher velocity targets such as the airliners in the ATC scenario.

```
function filter = initFilter(detection)
filter = initcvekf(detection);
classToUse = class(filter.StateCovariance);
spd = 900*1e3/3600; % m/s
velCov = cast(spd^2,classToUse);
cov = filter.StateCovariance;
cov(2,2) = velCov;
cov(4,4) = velCov;
filter.StateCovariance = cov;
% Set filter's process noise to match filter's update rate
scaleAccelHorz = cast(1,classToUse);
scaleAccelVert = cast(1,classToUse);
Q = blkdiag(scaleAccelHorz^2, scaleAccelHorz^2, scaleAccelVert^2);
filter.ProcessNoise = Q;end
```

tabulateData

This function returns a table comparing the ground truth and tracks

```
function truthTrackTable = tabulateData(scenario, tracks)% Process truth data
platforms = scenario.Platforms(2:end); % Platform 1 is the radar
numPlats = numel(platforms);
trueAlt = zeros(numPlats,1);
trueSpd = zeros(numPlats,1);
trueHea = zeros(numPlats,1);for i = 1:numPlats
```

```
traj = platforms{i}.Trajectory;
waypoints = traj.Waypoints;
times = traj.TimeOfArrival;
trueAlt(i) = -waypoints(end,3);
trueVel = (waypoints(end,:) - waypoints(end-1,:)) / (times(end)-times(end-1));
trueSpd(i) = norm(trueVel) * 3600 / 1000; % Convert to km/h
trueHea(i) = atan2d(trueVel(1),trueVel(2));end
trueHea = mod(trueHea,360);
% Associate tracks with targets
atts = [tracks.ObjectAttributes];
tgtInds = [atts.TargetIndex];
% Process tracks assuming a constant velocity model
numTrks = numel(tracks);
estAlt = zeros(numTrks,1);
estSpd = zeros(numTrks,1);
estHea = zeros(numTrks,1);
truthTrack = zeros(numTrks,7);for i = 1:numTrks
estAlt(i) = -round(tracks(i).State(5));
estSpd(i) = round(norm(tracks(i).State(2:2:6)) * 3600 / 1000); % Convert to km/h;
estHea(i) = round(atan2d(tracks(i).State(2),tracks(i).State(4)));
estHea(i) = mod(estHea(i),360);
platID = tgtInds(i);
```

```
platInd = platID - 1;

truthTrack(i,:) = [tracks(i).TrackID, trueAlt(platInd), estAlt(i),
trueHea(platInd), estHea(i), ...

trueSpd(platInd), estSpd(i)];end

% Organize the data in a table

names =
{'TrackID','TrueAlt','EstimatedAlt','TrueHea','EstimatedHea','TrueS
pd','EstimatedSpd'};

truthTrackTable = array2table(truthTrack,'VariableNames',names);

truthTrackTable = mergevars(truthTrackTable, (6:7),
'NewVariableName', 'Speed', 'MergeAsTable', true);

truthTrackTable.(6).Properties.VariableNames =
{'True','Estimated'};

truthTrackTable = mergevars(truthTrackTable, (4:5),
'NewVariableName', 'Heading', 'MergeAsTable', true);

truthTrackTable.(4).Properties.VariableNames =
{'True','Estimated'};

truthTrackTable = mergevars(truthTrackTable, (2:3),
'NewVariableName', 'Altitude', 'MergeAsTable', true);

truthTrackTable.(2).Properties.VariableNames =
{'True','Estimated'};

truthTrackTable.TrackID = "T" +
string(truthTrackTable.TrackID);end
```

snapPlotTrack

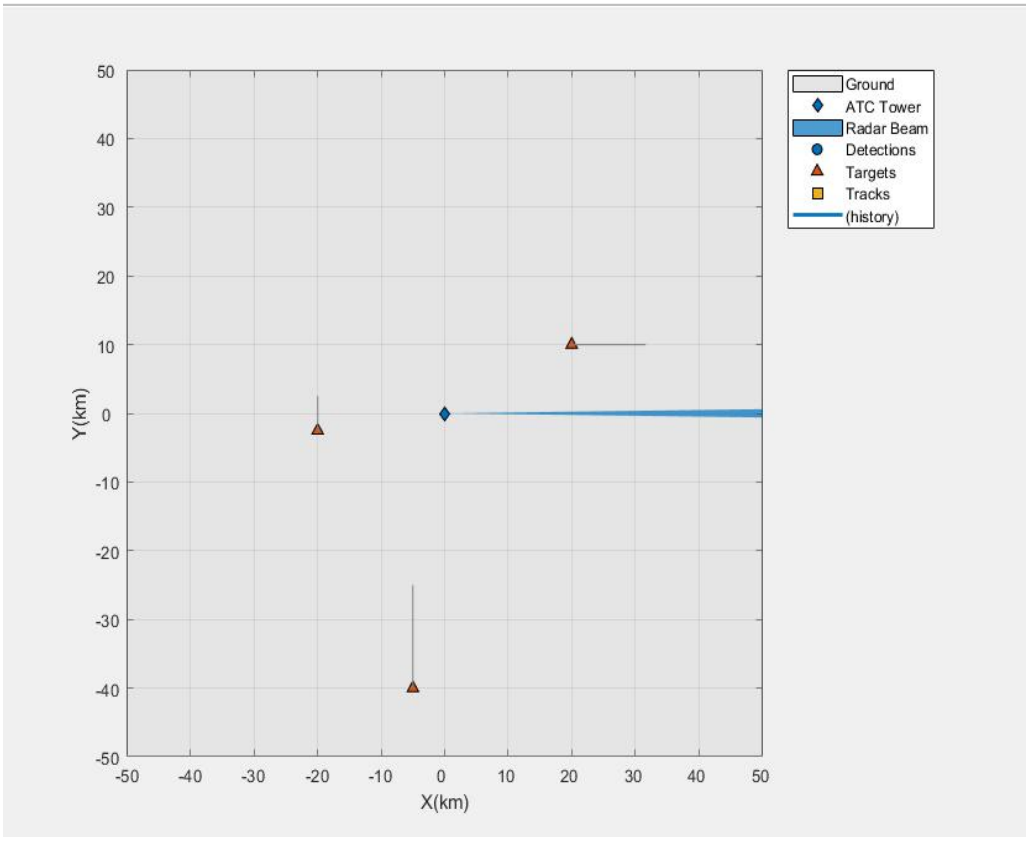
This function handles moving the camera, taking relevant snapshots and updating track visuals.

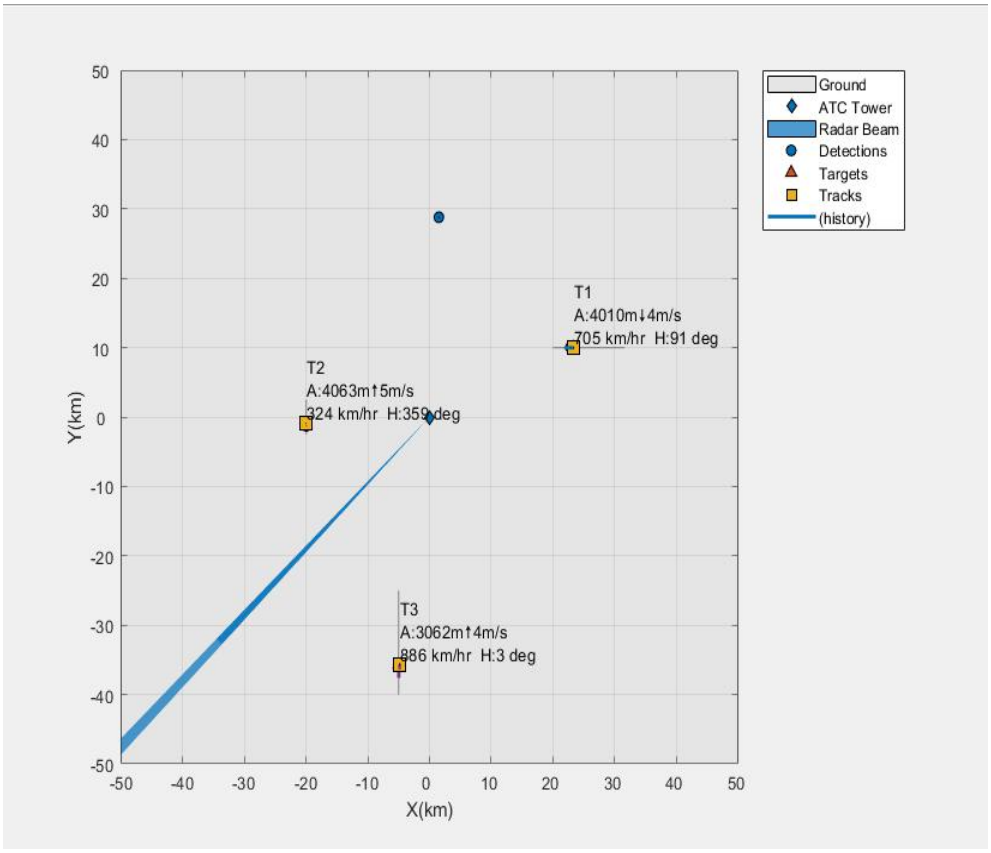
```
function allsnaps = snapPlotTrack(mapViewer,tracks,isScanDone, scanCount,allsnaps)% Save snapshots during first 4 scansif isScanDone && any(scanCount == [2 3])
newsnap = snapshot(mapViewer);
allsnaps = [allsnaps, {newsnap}];
%move camera
if scanCount == 2
% Show the outbound airliner
campos(mapViewer, [42.5650  -70.8990 7e3]);
drawnow
newsnap = snapshot(mapViewer);
allsnaps = [allsnaps, {newsnap}];
end
end
% Update display with current track positions
plotTrack(mapViewer,tracks, 'LabelStyle','ATC');
if isScanDone && any(scanCount == [2 3])
% Take a snapshot of confirmed track
drawnow
newsnap = snapshot(mapViewer);
allsnaps = [allsnaps, {newsnap}];
% Reset Camera view to full scene
if scanCount == 3
origin = [42.366978, -71.022362, 50];
campos(mapViewer, origin + [0 0 1e5]);
```

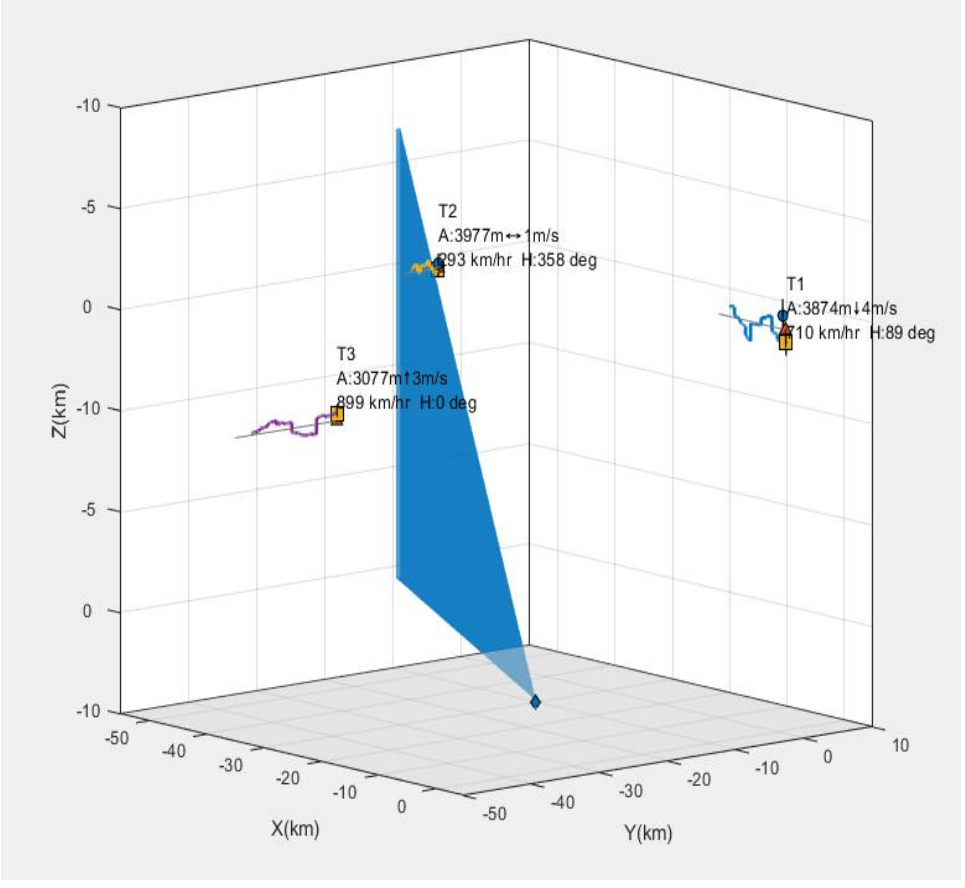
drawnow

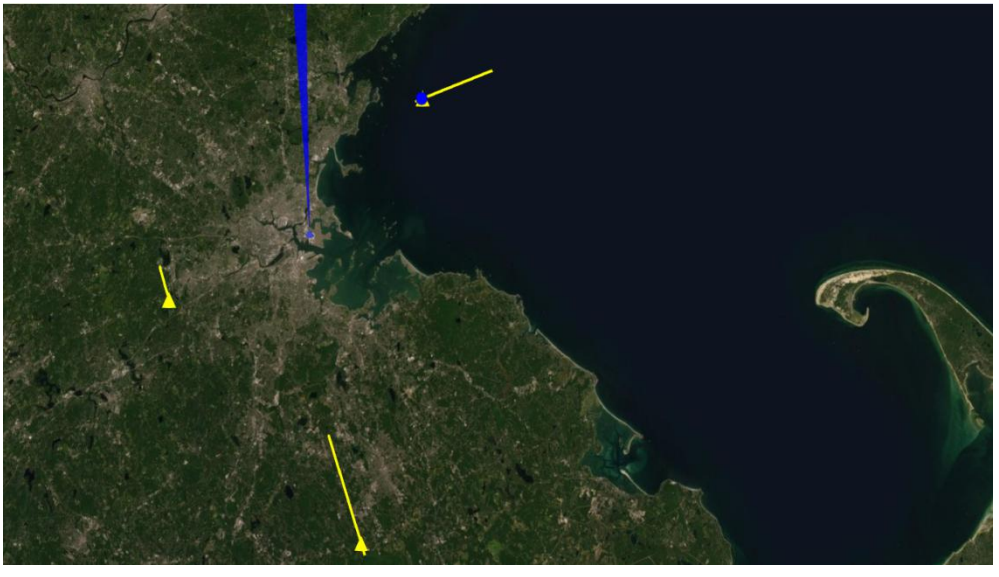
endendend

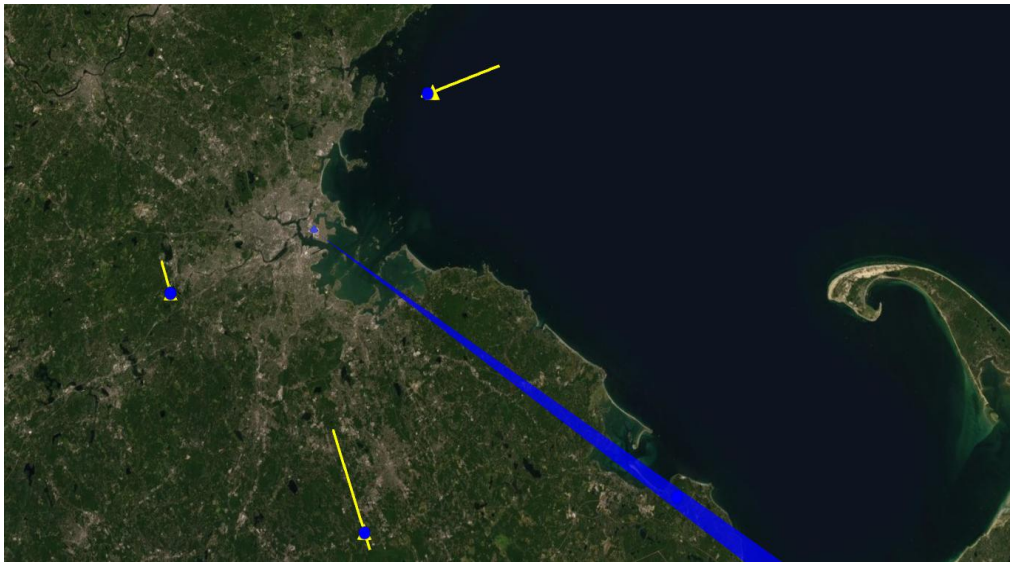
RESULTS:

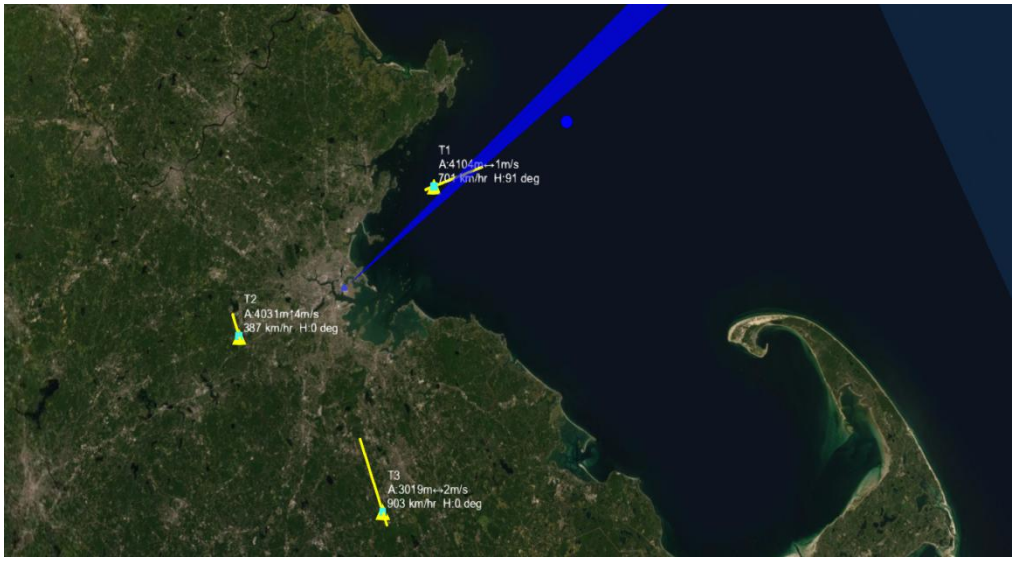



Ground
ATC Tower
Radar Beam
Detections
Targets
Tracks
(history)
T1
A:4010m↓4m/s
705 km/hr H:91 deg
T2
A:4063m↑5m/s
324 km/hr H:359 deg
T3
A:3062m↑4m/s
886 km/hr H:3 deg
Y(km)
X(km)
50
40
30
20
10
0
-10
-20
-30
-40
-50
-50 -40 -30 -20 -10 0 10 20 30 40 50


T2
A:3977m↔1m/s
893 km/hr H:358 deg
T1
A:3874m↓4m/s
810 km/hr H:89 deg
T3
A:3077m↑3m/s
899 km/hr H:0 deg
Z(km)
X(km)
Y(km)







CONCLUSION:

This report shows how to generate an air traffic control scenario, simulate radar detections from an airport surveillance radar (ASR), and configure a global nearest neighbor (GNN) tracker to track the simulated targets using the radar detections. In this example, you learned how the tracker's history based logic promotes tracks. You also learned how the track uncertainty grows when a track is

coasted and is reduced when the track is updated by a new detection.

REFERENCES:

1. A. K. Kamal, "Airport surveillance radar design for increased air traffic," *92 International Conference on Radar*, 1992, pp. 151-154.

2. Y.P. Saputera, M. Wahab, T. T. Estu, "Omnidirectional ultra-wideband antenna for radar detector applications", *Antenna Measurements & Applications (CAMA) 2017 IEEE Conference on*, pp. 253-256, 2017.

3. A Sree Sowmya Reddy, Anupama R K, Surya K, Kavitha V, "Design of S-Band Low Noise Block (LNB) for Radars with redundancy", *Communication information and Computing Technology (ICCICT) 2021 International Conference on*, pp. 1-4, 2021.

4. https://www.radartutorial.eu/02.basics/ATC-Radars.en.html

5. http://bionics.seas.ucla.edu/education/MAE_263D/RTB_MATLAB_Intro.pdf

6. https://www.faa.gov/air_traffic/technology/asr-11/

7. https://www.nec.com/en/global/solutions/cns-atm/surveillance/asr.html

8. https://www.britannica.com/technology/radar/Atmospheric-effects#ref237370

9. https://www.britannica.com/technology/air-traffic-control-radar-beacon-system

10. Bhatta, Niraj & Priya, M.. (2017). RADAR and its applications.