# SEMTA Radar Simulation System

## Overview

### Summary

* This document describes the use and functionality of the SEMTA simulation, signal processing, post-processing, and testing system implemented in MATLAB.

### Requirements

* General:
  + MATLAB
  + Signal Processing Toolbox
* Radar Simulation:
  + Communications Systems Toolbox
  + Phased Array Systems Toolbox
* Fixed Point Implementation (Required for modification but not execution):
  + Fixed Point Designer
  + DSP Toolbox
* Parallel Processing (Optional, increases speed of large simulations):
  + Parallel Computing Toolbox

### Simulation & Processing Description

#### Overview

The simulation and processing system is intended to perform end-to-end simulation of all aspects of the SEMTA system. This includes physical simulation of the radar signal response from the target, signal processing of the radar data cube, data processing of radar measurements, and multistatic data processing of the results of multiple radar units. This document details the operation of the single-unit radar simulation. Multistatic post-processing is described in ‘*Documentation/Post Processing Server.docx’* Block diagram overview of the system is shown in Figure 1, at right.

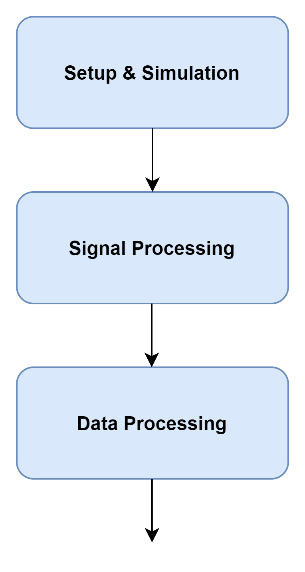


Figure 1. Basic overview of system operation.

#### Radar Signal Response Simulation

The first segment of the SEMTA system is the simulation component. This segment is intended to provide receive signal data which can then be used as input to the signal processing system, validating its correct operation. The simulation system is intended to be self-contained, so that it can be replaced with a real data parsing system without requiring large changes in the code base.

The position of the target and of each radar unit is determined by a few parameters, illustrated in Figure 3. The target’s trajectory is defined by a speed along the “track”, the distance of its sinusoidal excursion from the center of the track, and the period of the excursion. The array of radar units is defined by the distance from each unit to the center of the target’s trajectory, and by the distance between each radar unit.

The target uses a fluctuating RCS model which simulates the dependence of a real target’s RCS on the frequency and aspect angle

of the reflection. The purely fluctuating component of the RCS characteristic is generated by calculating the RCS of an ensemble of point scatterers which occupy the same cross-sectional area as the real target. In addition, a peak caused by specular reflection of a semi-cylindrical aircraft is added to the RCS. Plot of an example target RCS is shown in Figure 4.

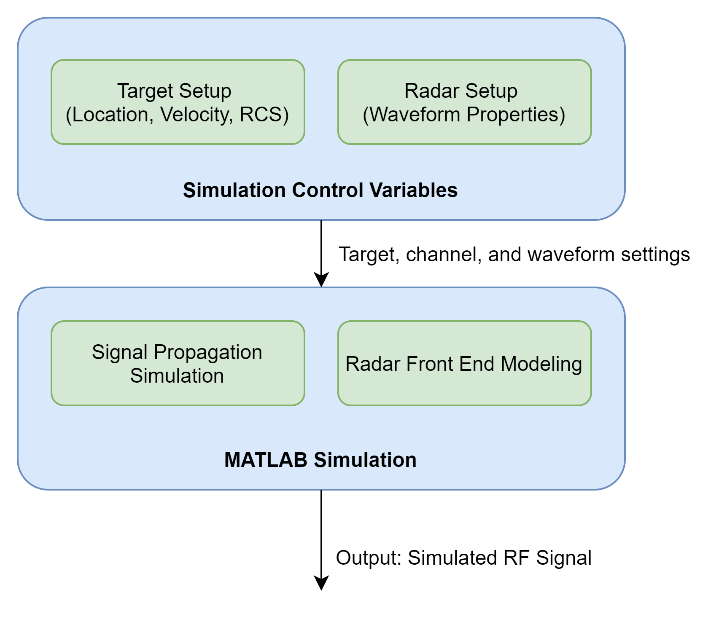


Figure 2. Simulation system components.

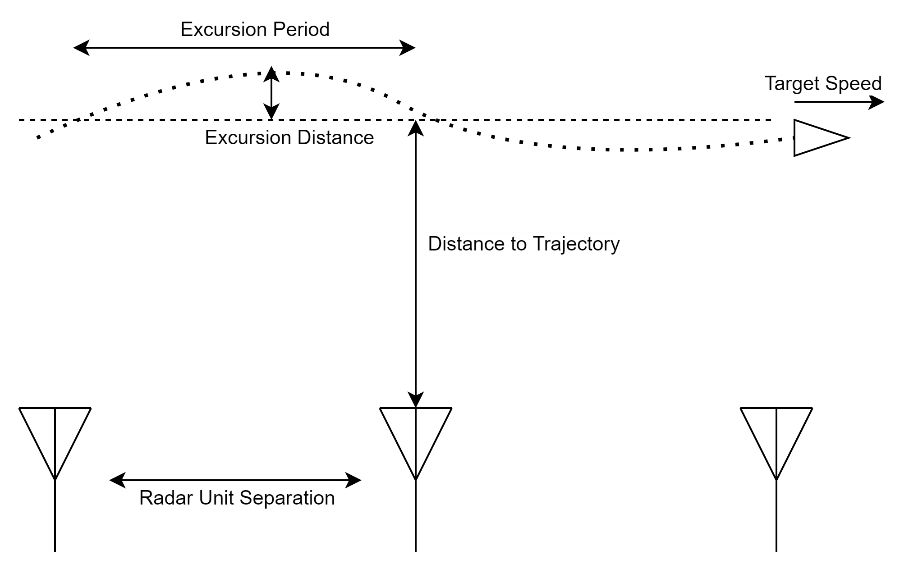


Figure 3. Target and radar system positioning.

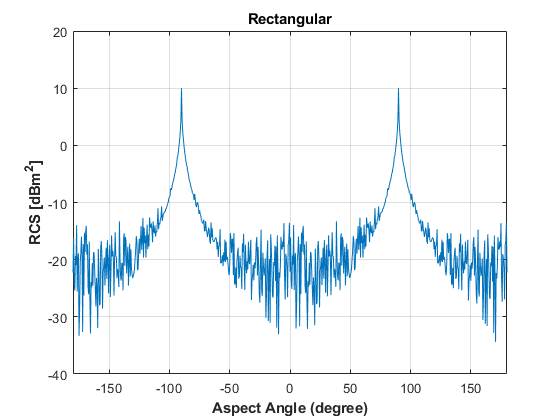


Figure 4. Example target RCS model.

Simulation of radar transmission, reflection, and reception is implemented using tools from Mathworks’ “Phased Array Toolbox” for MATLAB. This library provides functions which simulate anteanna patterns, channel effects, signal reflections, and receiver characteristics. Using these tools, a receive signal is calculated from the transmit waveform, and sent to the input of the signal processing system.

#### Signal Processing

The signal processing system performs operations on the radar data cube during each CPI of operation. This includes range processing, Doppler processing, CFAR detection, integration, and coordinate estimation. Block diagram of this segment is shown in Figure 5.

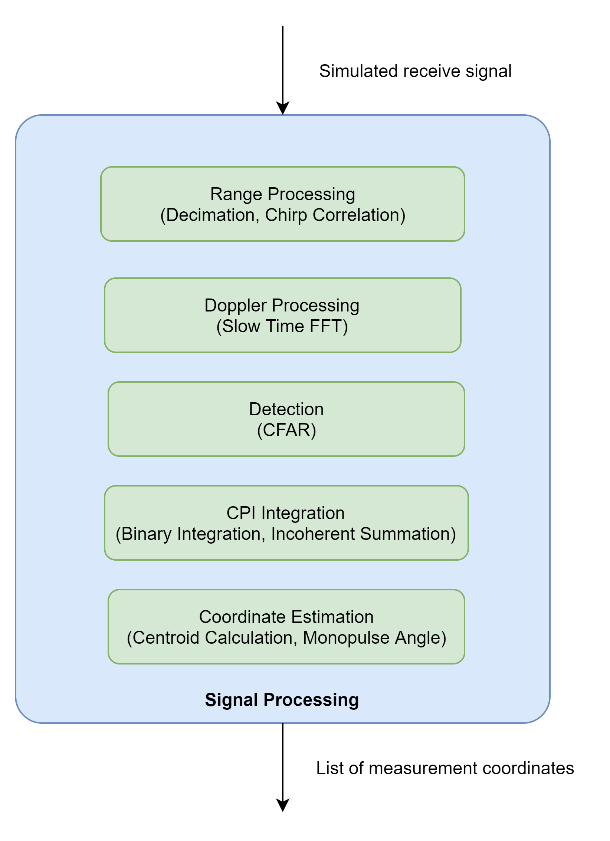


Figure 5. Overview of signal processing system.

Range processing is applied to the simulated receive signal, by performing a correlation with a windowed version of the transmit signal. Doppler processing is performed by taking the FFT of the range processed data in the slow time dimension, resulting in the range-Doppler data cube. Detection is performed using a 2D-CFAR algorithm.

The system must perform both binary m-of-n integration and incoherent integration, depending on whether the radar unit is in search or track mode. In the case of binary integration, a range bin migration compensation algorithm is applied to all detections, to mitigate the effects of high-speed targets moving between range bins between CPI. This is done by extending each detection through the swath of range bins that a target in that bin would cover by moving at the given Doppler velocity for the remainder of the frame. An illustration of this compensation algorithm is shown in Figure 6.

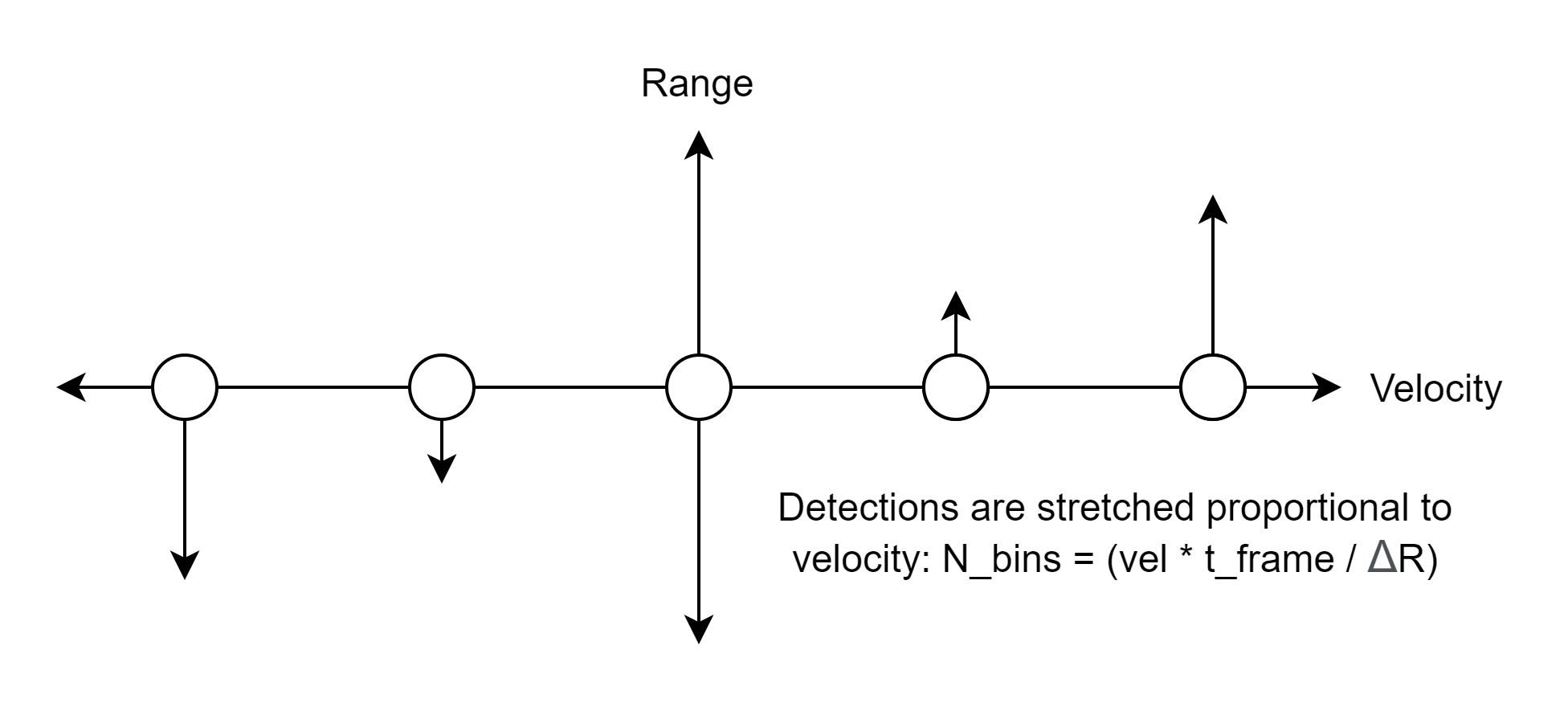


Figure 6. Illustration of range bin migration compensation used in binary integration algorithm.

After integration is performed, the “Hoshen-Kopelman algorithm” is used to determine all connected components of the binary detection data cube. Each connected component is treated as a single detection, which is passed to the coordinate estimation algorithm.

Coordinate estimation is performed using a power-weighted centroid method. The centroid value is computed for the target’s range and velocity by using its position in the data cube, and an estimation of the target’s angle-of-arrival is computed by taking a power-weighted average of the amplitude monopulse value in each bin of detection.

The range, velocity, and angle estimations are saved to a list of targets along with a timestamp and signal power estimate. This list of targets is then provided as the input to the data processing system.

#### Single Unit Data Processing

Using the list of estimated target coordinates, each unit performs gating and track association, followed by tracking using a Kalman filter. Block diagram overview of data processing system is shown in Figure 7.

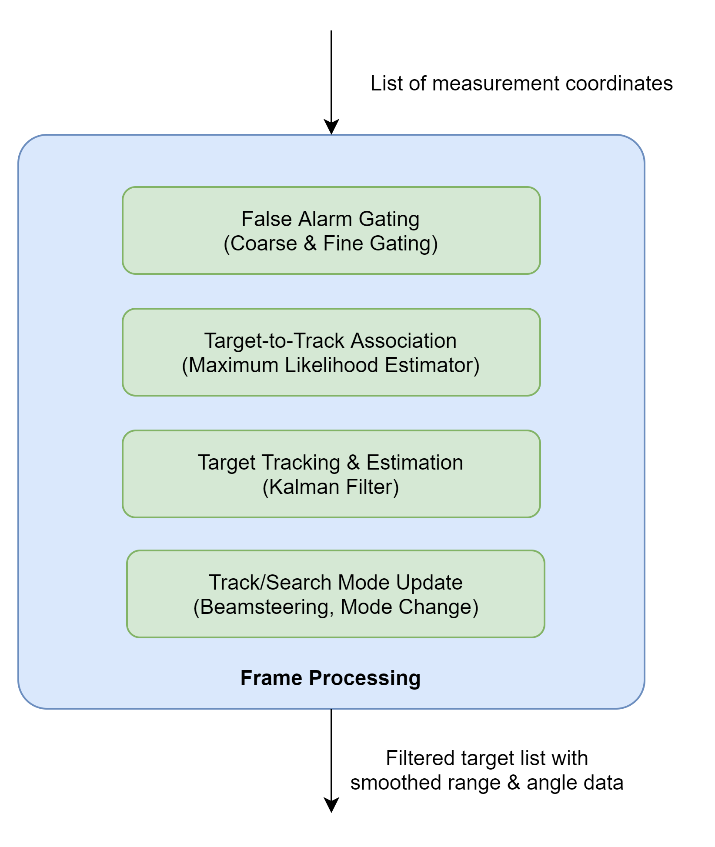


Figure 7. Block diagram of single unit data processing system.

Both coarse and fine gating to determine if any detections are not plausible as new inputs to existing tracks. Coarse gating eliminates all detections which have a larger change in range than is possible during the frame time, based on a maximum target velocity parameter. Fine gating takes the “Mahanalobis distance”, a measure of the statistical distance between a measurement and a distribution, using the previous track’s covariance matrix.

Target-to-track association is performed by using the remaining measurement which has the lowest Mahanalobis distance from the predicted location of the tracked target. This measurement is used as input to the Kalman tracking filter, along with an estimate of the error uncertainty of the measurement which is calculated using empirical curves determined by previous Monte Carlo testing.

In order to model the frame-to-frame behavior of the radar system, there is a single callback from the results of the tracking system. The estimated angle of the next target position is used to set the simulated beamsteering angle, and whether the tracking system counts a new detection or a missed detection is used to determine whether the system should simulate the radar in track mode or in search mode.

## User Guide:

### Execution

To run system, start by setting parameters as desired in the four scripts found in *Scripts/Setup*. See next section for detailed description. Then, run either of the three scripts in the main directory, *Fullsystem.m, FullySystem\_FixedPoint.m,* or *FullSystem\_NoSimulation.m*, depending on desired function. If a post-processing server is live, then *FullSystem\_PostServer.m* can be used to run simulation and post-processing simultaneously.

### Setup

Setup is handled in four scripts within the folder *Scripts/Setup*. Each one defines a struct of key-value pairs which hold each of the parameters required to define the simulation/processing setup. Each parameter is defined below:

* **SetupRadarScenario.m** – Defines “scenario.radarsetup”, which holds parameters describing the waveform, antennas, processing, detection, and tracking.
  + f\_c – Center frequency (Hz)
  + f\_s – ADC Sampling frequency (Hz)
  + t\_p – Duration of LFM pulse (s)
  + bw – Bandwidth of LFM pulse (Hz)
  + prf – Pulse repetition frequency (Hz)
  + n\_p – Number of LFM pulses per CPI (#)
  + n\_ant – Number of elements in antenna array (#)
  + tx\_pow – Peak transmit power for single channel (W)
  + rx\_ant\_gain – Gain of Rx antenna array (dB)
  + tx\_ant\_gain – Gain of Tx antenna array (dB)
  + rx\_nf – Noise figure of receiver (dB)
  + range\_off – Apply range offset curve defined in *Results/Error Curves* to centroid algorithm (T/F)
  + beamwidth – Antenna array beamwidth (deg)
  + mono\_coeff – Emperically calculated monopulse coefficient (unitless)
  + phase\_bits – Number of bits in phase shifter resolution (#)
  + r\_win – Window function applied to range processing (string)
  + d\_win – Window function applied to doppler processing (string)
  + detect\_type – Type of detection algorithm to apply on data cube (string)
  + thresh – Threshold value if *detect\_type* is set to “threshold” (dB)
  + Pfa – False alarm rate if *detect\_type* is set to “CFAR” (unitless)
  + num\_guard – Number of guard cells used for CFAR window, in range x doppler directions (#)
  + num\_train – Number of training cells used for CFAR window, in range x doppler directions (#)
  + rng\_limits – Minimum and maximum range to search with CFAR. Useful in removing false alarms from noise dropoff at high and low range. (m)
  + det\_m – Value of M for M-of-N processing if *int\_type* is set to ‘binary’
  + min\_SNR – Minimum SNR for likely detection, used only if *fast\_simulation* is set to true in *Setup\_Simulation.m* (dB)
  + max\_vel – Maximum possible velocity of target for coarse gating and variance estimation (m/s)
  + max\_acc – Maximum possible acceleration of target for variance estimation (m/s^2)
  + dist\_thresh – Mahanalobis distance threshold for fine gating (unitless)
  + miss\_max – Maximum number of missed detection before track is deactivated (#)
  + EKF – Use Extended Kalman Filter (T/F) (OBSOLETE)
  + sigma\_v – Target motion uncertainty in x,y directions (unitless)
  + sigma\_v\_multi – Multilateration target motion uncertainty (unitless) (OBSOLETE)
  + bi\_multi – Use bidirectional tracking for multilateration (T/F)
  + bi\_single – Use bidirectional tracking for single-unit post-processing (T/F)
  + limitSensorFusion – Limit data fusion to only two units (T/F) (OBSOLETE)
* **SetupSimulation.m** – Defines “scenario.simsetup”, which hold parameters describing MATLAB options and configurations
  + readout –Write target data to command window (T/F)
  + clear\_cube – Delete large data structures from scenario object (T/F) (OBSOLETE)
  + send\_alert – Send email alert after system is complete (T/F)
  + attach\_zip – Attach .zip file of generated files to alert email (T/F)
  + alert\_address – Email address for email alerts (string)
  + par\_cfar – Use Parallel Processing Toolbox to parallelize CFAR, for faster processing (T/F)
  + fast\_simulation – Skip frames where target detection is extremely unlikely, for faster simulation.
  + filename – File name prefix for saving files (string)
  + timestampfile – Append timestamp to filename (T/F)
  + save\_format – File extensions for saving figures, set in save\_format.list (string)
  + save\_figs – Save figures to file (T/F)
  + save\_mat – Save .mat file of scenario object to file (T/F)
  + reduce\_mat –Remove large data structures from scenario object before saving (T/F)
  + save\_track – Save results of tracking to .mat file for post processing (T/F)
  + save\_track\_single – Save results of single-unit tracking to .mat file for post processing (T/F)
  + server\_IP – IP address and port of post-processing server (string)
* **SetupTarget.m** – Defines “scenario.rcs” and “scenario.traj” which hold parameters describing the target RCS and motion
  + rcs\_model – Type of model to use, either “model” or “constant” (string)
  + ave\_rcs – Average RCS of target (dBm^2)
  + dim – Estimated size of target for RCS simulation (m)
  + n\_sc – Number of point scatterers used for RCS simulation (#)
  + res\_a – Angular resolution of RCS simulation (deg)
  + freq – Frequency range to model in RCS simulation (Hz)
  + specular\_on – Use specular reflection model (T/F)
  + theta\_zero – Angle at which specular response is 0dBm^2 (deg)
  + theta\_ave – Angle at which specular response is equal to *ave\_rcs* (deg)
  + peak\_rcs – RCS at specular peak (dBm^2)
  + alt – Altitude of target trajectory (m)
  + yvel – Along-track velocity of target (m/s)
  + exc – Distance of cross-track trajectory excursion (m)
  + per – Period of cross-track excursion (unitless)
  + pos\_st – Set position if *model* is set to ‘static’ (m)
  + vel\_st – Set velocity if *model* is set to ‘static’ or ‘linear’ (m)
  + model – Type of model to use, either ‘static’, ‘linear’, or ‘model’ (string)
* **SetupMulti.m** – Defines “scenario.multi” which holds parameters describing multistatic setup
  + num\_receivers – Number of radar units to simulate (#)
  + dist\_from\_center – Cross track distance from trajectory center to radar line (m)
  + unit\_spacing – Along track distance between radar units (m)
  + spacing\_offset – Along track offset of radar units (m)
  + simulation\_time – Amount of time to simulate (s)
  + timing\_jitter – Max random deviation in measurement time between units (s)

### Plotting & Visualization

The following functions can be run to display results of simulation & processing. Each are methods of the RadarScenario class, and can be found in *Definitions/RadarScenario.m*.

* Target Model Plots:
  + *viewRCSFreq* – Plot RCS of target over frequency
  + *viewRCSAng* – Plot RCS of target over aspect angle
  + *viewTraj*  - Plot trajectory in top-down view
* Signal Processing Plots:
  + *viewRxSignal* – Plots receive signal in time domain
  + *viewCorrelationFFT* – Plots correlation signal before IFFT in range processing
  + *viewRangeTimeDomain* – Plots results of single chirp range processing in time domain
  + *viewRangeCube* – Plots results of full range processing cube
  + *viewRDCube –* Plots results of range doppler processing
  + *viewIncoherentCube –* Plots incoherently integrated power cube
  + *viewDetections –* Plots detections in power cube
  + *viewSNR –* Plots SNR across frames
  + *viewTrackingSingle –* Plots tracking results of a single radar unit
  + *viewTrackingMulti –* Plots tracking results of multistatic processing
  + *viewErrorsSingle –* Plots single-unit error from ground truth trajectory
  + *viewErrorsMulti –* Plots multistatic processing error from ground truth trajectory

### Data Object Structure

All data and parameters in simulation are attached to a data object named “scenario”. The use of each subfield is described below:

* *multi –* Data temporarily stored between frames and units, before saved to tracking results
* *tracking\_single –* Each cell contains the results of a single unit’s tracking system
* *tracking\_multi –* Results of multistatic tracking
* *rcs –* Parameters and functions regarding the target’s RCS
* *traj –* Parameters and functions regarding the target’s trajectory
* *radarsetup –* Physical parameters and system details
* *simsetup –* MATLAB simulation options
* *sim –* Containerholding simulation objects used by MATLAB Phased Array Toolbox
* *rx\_sig* – Most recent receive signal of a full frame
* *cube* – Radar data cube(s)
* *detection* – Information on detected targets from CFAR system and coordinate estimator
* *flags* – Variables used to track progress in system
* *timing* – Variables used to track remaining time in simulation

## File & Function Description

### File Directories:

* **Definitions** – Contains the definition of the “RadarScenario” class
  + *Radar Scenario.m* - This class is used as the main scenario object which holds all relevant simulation data, and contains methods which mostly perform plotting & visualization
* **Documentation** – User-written documentation
  + *Radar Simulation.docx* – This document
  + *Post Processing Server.docx* – Describes the post-processing Python server
* **Figures** – Output folder for *SaveFigures* function. Not included in git repo.
* **Fixed Point –** Contains files used by Fixed Point Designer tool
  + **codegen**  - Files generated by the FP Designer, including fixed point versions of MATLAB functions, wrappers to use these functions in floating point code, and .mex binary files which are compiled with fixed point data types.
  + **Function** – Functions written as input to the FP Designer tool
  + **Project** – Project files, which open FP designer and document conversion process
  + **Reference** – .mat files containing test bench inputs and reference values
  + **Test Bench** – Scripts which test the functions in “Function” and “codegen”
* **Functions –** Functions used in main system
  + **Data Processing –** Functions which operate on single-unit data in coordinate form, after signal processing.
    - *BeamSteeringUpdate.m –* Updates beamsteering based on tracking results and current steering mode
    - *KalmanFilter\_SingleUnit.*m – Kalman filter for single-unit tracking
    - *MahanalobisDistance.*m – Statistical distance measurement used in fine gating for single-unit tracking
    - *ModeCheck.*m – Updates beamsteering mode using tracking results
    - *Tracking\_SingleUnit.*m – Outer function for single unit tracking, including Kalman filter, fine and course gating, and track management.
    - *Tracking\_SingleUnit\_Bidirectional.*m – Single-unit tracking which is capable of processing both forward in time and backwards in time. Used for post-processing.
    - *Tracking\_SingleUnit\_Post.*m – Runs both forward and backwards tracking, and performs data fusion. Used for post-processing.
  + **Multistatic Processing –** Functions which operate on data derived from multiple radar units, after single-unit Data Processing
    - *DataFusion.m –* Uses inverse-variance-weighted averaging to combine results from single-unit tracking systems.
    - *Tracking\_Multi.*m – Outer function for multiple unit tracking, running both forwards and backwards in time and implementing a Kalman filter.
    - *Tracking\_Multi\_Bidirectional.*m – Runs both forwards and backwards tracking, and performs data fusion.
    - *ErrorEstimation.*m – Compares results of simulated data to ground truth of target trajectory.
    - *SendToTrackingServer.*m – Sends data prepared by *Functions/Utilities/SaveTrackingSingle.m* to external post-processing server using HTTP request.
  + **Signal Processing –** Functions which operate on raw received signals or the radar data cube.
    - *Detection.m –* Performs CFAR detection and data integration on radar power cube, then performs coordinate estimation using a centroid algorithm and an amplitude monopulse algorithm.
    - *Detection\_FixedPoint.*m – Same as previous function but with fixed-point CFAR algorithm implemented.
    - *RealDataInput.*m – Function to decode live test data files into format used by simulation and processing system. Requires revision before use, see note in ‘Open Tasks’ below.
    - *SignalProcessing.*m – Range and doppler processing of input signals
    - *SignalProcessing\_FixedPoint.*m – Same as previous function but with fixed-point range and doppler processing functions implemented
    - *SimulateAxes.*m – Generates range and doppler axes without running signal processing. Used for scenarios without simulation.
  + **Simulation –** Functions used in simulation of the real radar scenario, not used in a real test setup
    - *CalculateSNR.m –* Calculate ideal SNR of point target at given coordinates. Used to verify signal responses of simulation system.
    - *CalculateVariance.m –* Using empirical curves from Monte Carlo tests, estimate error variance of measurement at set coordinates. Used to initialize variance matrices in Kalman filter.
    - *PhasedSetup.*m – Sets up Phased Array Toolbox objects used by simulation
    - *RadarSimulation*.m – Runs simulation of single frame radar transmission, reflection, and reception. This is the main script for simulating received signals.
    - *SimulateDetections*.m – Using empirical results from simulation tests, generates detection coordinates without running simulation. Used for rapid Monte Carlo testing.
    - *TargetRCSModel.m* – Implements fluctuating target RCS model with specular reflection.
    - *TrajectoryModel.m* – Implements trajectory simulation for target motion.
  + **Utilities –** Useful functions such as file handling and memory management
    - *EmailAlert.m* – Sends email from external SMTP server. Useful for notifying user if long tests are complete.
    - *EmailSetup*.m – Sets up email alert system.
    - *SaveFigures*.m – Saves open figures to file.
    - *SaveScenario*.m – Saves scenario data object to file
    - *SaveTracking*.m – Saves tracking results to file, for use in previous version of post processing Python server. (DEPRECATED)
    - *SaveTrackingSingle*.m – Saves results of single unit tracking to file, for use in post-processing server.
* **Input –** Folder which contains live test data to be processed. Not included in git repo.
* **PostProcessing –** External post-processing HTTP server running in Python. See *Documentation/Post Processing Server.docx* for more information.
  + **Input –** Files uploaded to HTTP server. Not included in git repo.
  + **Output –** Files generated by post processing server. Not included in git repo.
  + **Instance –** Configuration files generated per instance of post-processing server. Not included in git repo.
  + *Server.py* – HTTP server which collects files and initializes post processing. Also presents a web interface using Dash.
  + *Tracking*.py – Post processing and result generation functions.
  + *requirements*.txt – Python package list for pip.
  + *Dockerfile* – Dockerfile which describes build process of docker container.
  + *docker*-*compose.yml*  - File which describes Docker Compose settings used to run Docker container continuously.
* **References –** Reference documents
  + **Phase I** – Documents from SBIR Phase I
    - *SEMTA Kickoff\_190930.pdf –* Slides from Phase I kickoff
    - *SEMTA Signal Processing.*docx – Phase I signal processing description
  + *33265 AF-SEMTA II Kickoff Meeting 210629.pdf –* Slides from Phase II kickoff
* **Results –** Folder used to save results by *SaveScenario* and *SaveTracking.* Not included in git repo.
* **Scratch** – Assorted scratch files. Not maintained.
* **Scripts** – Matlab programs, not written as functions, used for setup and execution of system.
  + **Functional** – Starting tasks, stopping tasks, and main simulation loop
    - *Main.m* – Main simulation/processing loop
    - *Main\_FixedPoint*.m – Same as above, but calls fixed point signal processing system
    - *Main\_NoSimulation*.m – Same as above, but generates detection coordinates form empirical curves instead of running simulation.
    - *Main\_PostServer.m* – Same as above, but uses HTTP request to send data to external post-processing server after each radar unit’s simulation completes.
    - *Main\_Realdata.m ­*– Modified processing loop for the use of real data input. Requires modification to run, see note in ‘Open Tasks’ below.
    - *StartProcess.m –* Tasks to run at beginning of simulation/processing.
    - *EndProcess.*m – Tasks to run at end of simulation/processing .
  + **Setup** – Scripts which declare parameters and simulation setup
    - *SetupRadarScenario.m –* Declares parameters and variables of radar system, waveform, processing, and tracking
    - *SetupTarget.*m – Declares RCS and trajectory of target.
    - *SetupMulti.*m – Declares number of receivers, their positions, and the number of frames.
    - *SetupSimulation.*m – Sets up parameters of system, such as whether to use parallel processing or whether to save files.
    - *SetupRealData.m –* Sets up parameters for real data processing, using a combination of previous scripts in this directory, modified for live testing cases.
* **Test** – Files used for Monte Carlo testing. Not maintained.
* *FullSystem.m –* Main script for running system.
* *FullSystem\_FixedPoint.*m – Main script for running system with integrated fixed point functions
* *FullSystem\_NoSimulation.*m – Main script for running system, skipping simulation and signal processing.
* *FullSystem\_PostServer.m –* Main script for running system only for live processing, then sending results to post-processing server using HTTP request.
* *FullSystem\_RealData.m –* Main script for running system with real data files from live test. Will need to be modified to run, see note in ‘Open Tasks’ below.

## Notes

### Open Tasks

* To use real data form live tests, a function will need to be written to read and interpret data file input.
  + A function script has been created, *Functions/Signal Processing/RealDataInput.m*, which provides a description of the exact details of the script which will need to be written.
  + Two wrapper scripts, *FullSystem\_RealData.m* and *Main\_RealData.m,* have been written which hold the basic control structure which can be used to process live test data.