# SEMTA Post-Processing Server

## Overview

* The SEMTA system uses a remote server to collect data and perform post-processing tasks such as data fusion and tracking.
* This system is implemented in two Python scripts. Data processing is performed by Tracking.py, while Server.py implements an HTTP server using Flask and Dash.
* Processed data can be viewed using the web interface, which can be found at the server’s web address, followed by ‘/dashboard’. For example, ‘http://localhost:5000/dashboard’.

## Processing Description

Block diagram overview of the SEMTA post-processing system is shown in Figure 1 below.

The first step in the post-processing system is performing tracking on the results of each individual radar unit. During live processing, this is done during each frame of operation, sending new radar measurements as input to the live tracking system, which performs coarse and fine gating, target-to-track association, and Kalman filter tracking.

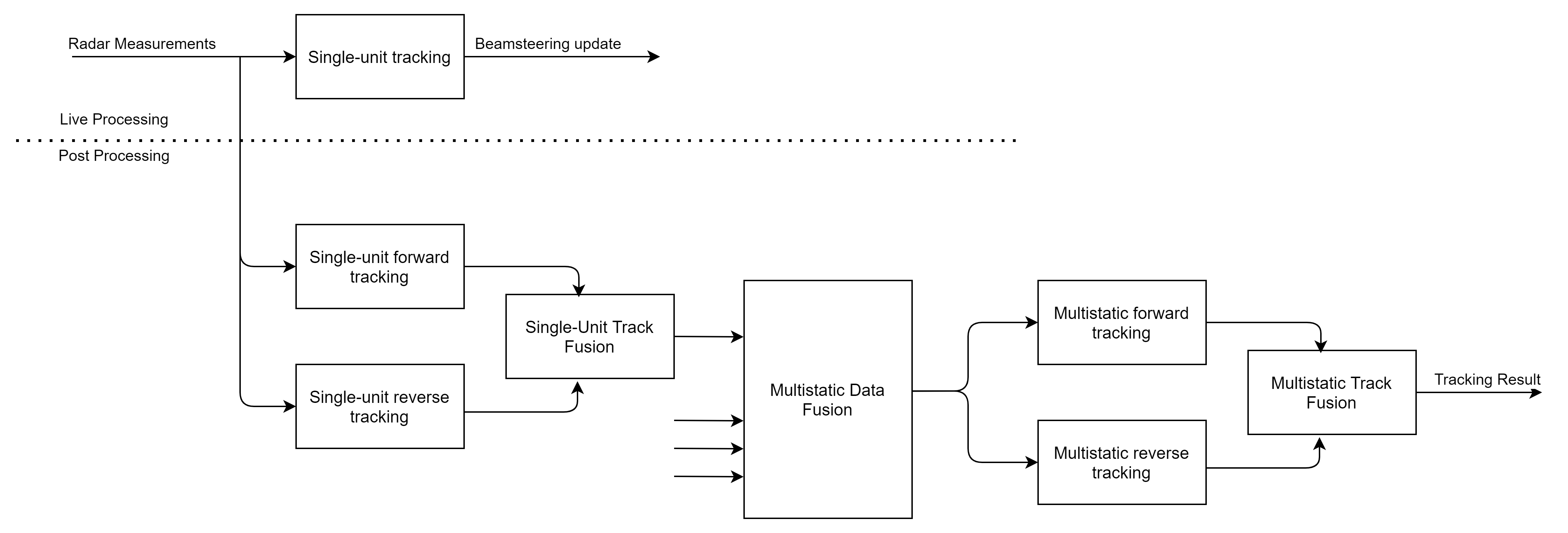


Figure 1. Block diagram overview of post-processing system

For post-processing, the system has the benefit of performing smoothing as well as tracking by applying Kalman filter estimation both forward in time and backwards in time. Additionally, since gating and target-to-track association has been performed by the single unit system, these functions can be bypassed, using only the radar measurements which passed track association.

For each radar unit’s data, the forward and reverse Kalman filters are applied, and the two results are fused using inverse-variance weighted averaging, which produces both an improved state estimate and a reduced variance estimate, which is used to seed the multistatic tracking system.

To combine the results of multiple radar systems, all smoothed measurements are combined into a vector of measurement data objects, which is then sorted in order of the measurement’s time stamp. A second pass of tracking is then performed, applying both a forward and reverse tracking system to the combined vector of measurements, and again using inverse-variance weighted averaging to produce the final result.

## Server Deployment

* A Docker container using Docker Compose has been configured for installation & deployment to any target operating system:
  + Install Docker and Docker Compose using preferred package manager
  + Set desired port in PostProcessing/docker-compose.yml, defaults to 5000
  + Issue the following commands within the “PostProcessing” directory:
    - docker-compose build
    - docker-compose up -d

## Web Interface

* This server implements a simple web interface using Plotly and Dash.
* Web interface can be reached at ‘[SERVER IP ADDRESS:PORT]/dashboard’.
* Includes menu to modify tracking filter parameters and repeat processing
* Plots on web interface can be adjusted and saved to local system.
* See example web interface readout in Figure 2.

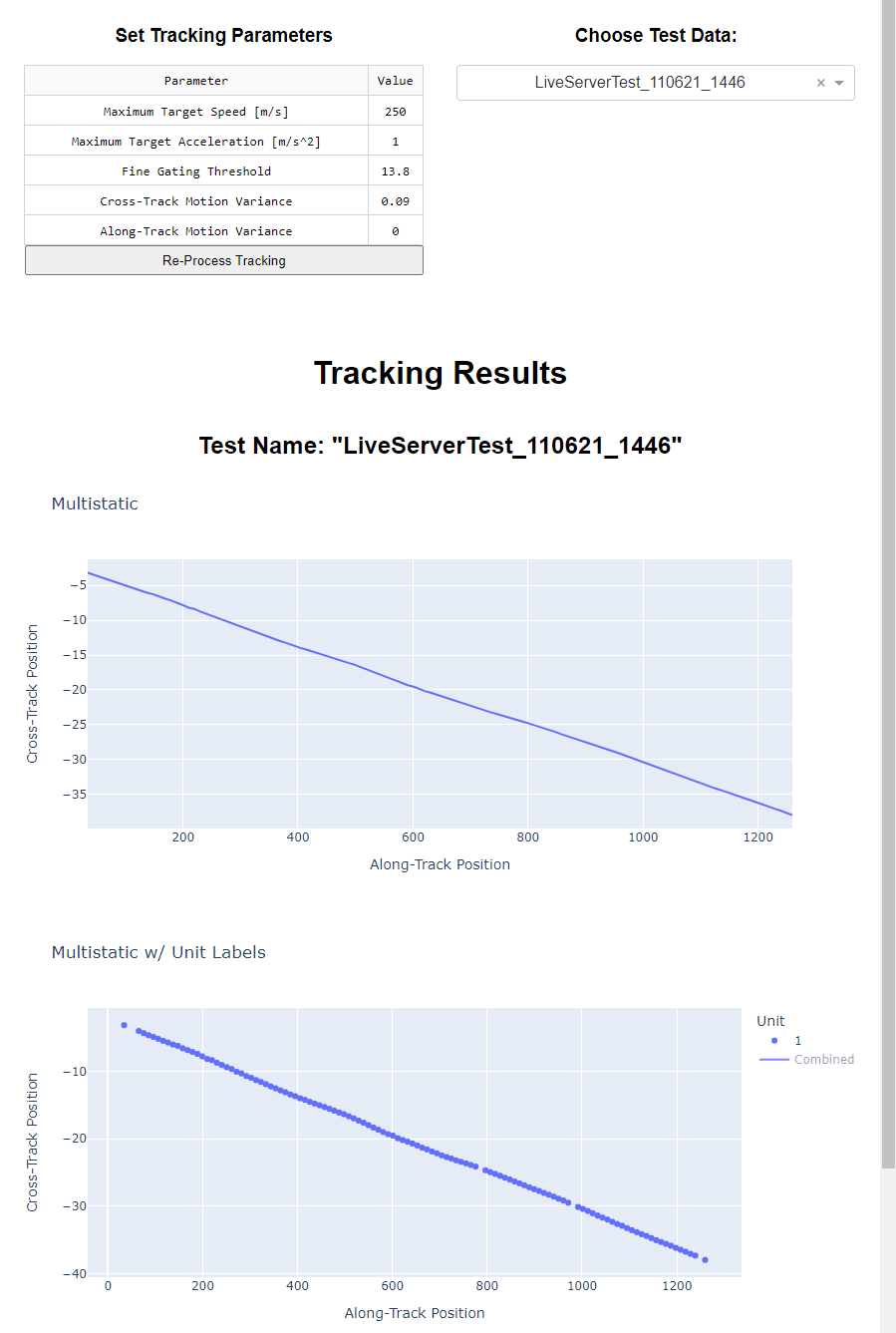


Figure 2. Example of web interface

## Client Usage

* Radar data is currently packaged in a .mat file, for use with MATLAB simulation and processing results. This format should be modified for file format of final use case.
  + The .mat file contains the following fields:
    - ‘n\_fr’
      * Number of frames of measurement data
      * Integer value
    - ‘range’
      * Estimated range to target, in meters
      * Floating point vector of length ‘n\_fr’
    - ‘vel’
      * Estimated Doppler range rate of target, in meters per second
      * Floating point vector of length ‘n\_fr’
    - ‘az’
      * Estimated azimuth bearing of target, in degrees
      * Floating point vector of length ‘n\_fr’
    - ‘steer’
      * Antenna array steering direction, in degrees
      * Floating point vector of length ‘n\_fr’
    - ‘SNR’
      * Estimated signal-to-noise ratio of target reflection, in decibels
      * Floating point vector of length ‘n\_fr’
    - ‘time’
      * Timestamps of measurements
      * Floating point vector of length ‘n\_fr’
    - ‘radar\_pos’
      * Radar position, in meters
      * Floating point vector of length 3, describing down-range x cross-range x vertical position
* Radar measurement data is uploaded to server using HTTP POST request.
  + Default URI is ‘/’, for example http://127.0.0.1:5000/
  + File is attached to request body using ‘form-data’ format, with key “file”
* If file is successfully sent, it will be saved in the “PostProcessing/Input” folder.
  + A folder within “PostProcessing/Output” is generated using the name of the file.
  + .csv files of tracked target coordinates, along with .png files of resulting scatter plots, are saved to the new output file.
  + One of each type of output file is created for each single-unit tracking result, plus one for the multistatic results.
* A browser or REST API client can be used to verify connection and correct operation.
  + GET request to root URI of ‘/’ should return HTTP status 200 if connected successfully

## Open Issues

* Tracking.py file will need to be modified to accommodate file type used as output of radar system.
  + .mat file is currently used, as it is directly output from MATLAB.
  + File unpacking code is found in lines 552-573 of Tracking.py.
* HTTP server and port will need to be configured and exposed to WAN.
  + HTTP server currently runs on localhost:5000.
  + Port 5000 can be modified by editing Dockerfile and rebuilding Docker image.