

# Multi Object 3D Tracking

## Architectural report

By

Spencer Randolph

# Inspiration:

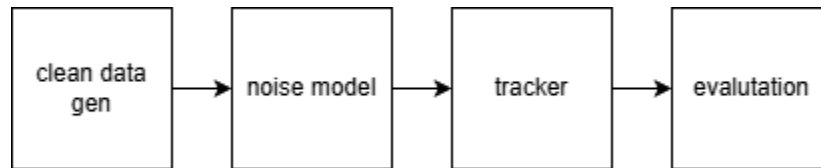
In a perfect system with no noise, and a super high refresh rate, tracking can be thought of as a simple combinatorial problem, matching detections[t-1] to detections t with minimizing the total distance between pairs. Unfortunately in real world systems noise exists and the refresh rate has to be realistic. So a simple combinatorial approach will not work.

While working with end to end systems I noticed nearest neighbors tracking was not viable in real world systems since it used this unrealistic combinatorial approach. This project aims to use motion models in a real time system design to improve tracking.

This document aims to explain the design of the system and how data flows through it.

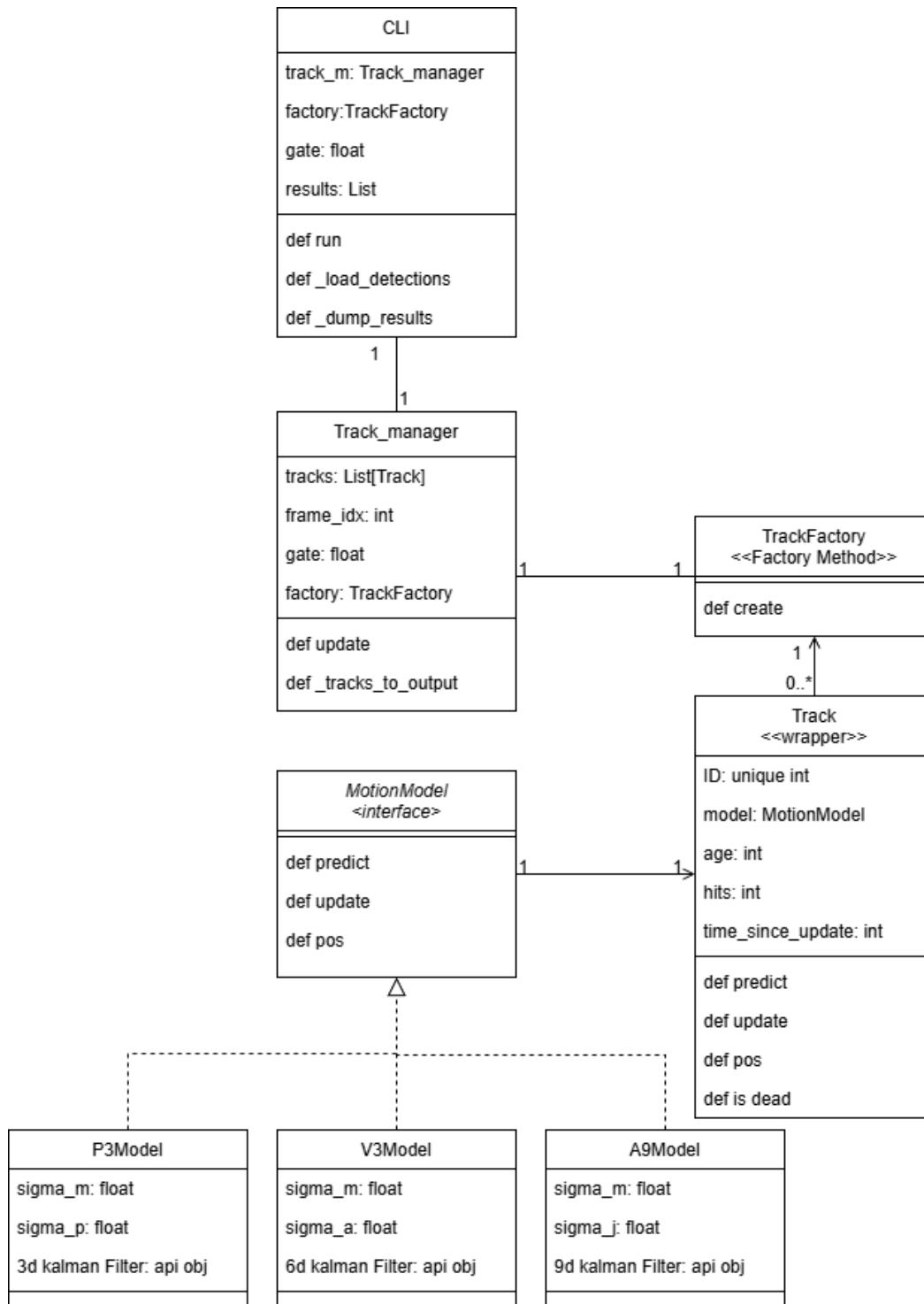
# Overview:

The system can be split up into 3 primary components; data generation, tracking and evaluation. Data generation and evaluation are done primarily through python scripts and will be covered in the results report. This document will primarily deal with tracker.



The tracker uses motion models to predict detections, building off of the previous motion of the object. By using the predictions for tracking instead of the previous location the system is able to make better matches through noise.

# Tracker:



## CLI

The CLI serves as a test interface for running repeatable experiments. It instantiates the TrackManager and TrackFactory, loads detections from CSV, runs the per-frame tracking loop, and writes results and metrics artifacts.

## Track\_manager

TrackManager owns the active set of tracks and runs the per-frame tracking cycle. Each update step performs prediction, data association between predicted track positions and incoming detections, applies measurement updates to matched tracks, spawns new tracks for unmatched detections, and prunes stale tracks based on time-since-update.

## TrackFactory

TrackFactory centralizes creation of Track instances. It assigns consistent unique track IDs and constructs the selected MotionModel implementation (e.g., p3, v6, v9) so that track initialization is uniform and the tracking pipeline can swap motion models without changing TrackManager.

## Track

Track represents a single tracked object and serves as a thin wrapper around a MotionModel. It stores track-level lifecycle state (unique ID, age, hits, time-since-update) and exposes a consistent interface (predict(), update(pos), pos(), is\_dead()) while delegating all estimation to the underlying motion model. This keeps the track-management logic independent of the specific filter/state dimension and lets motion models be swapped without changing the rest of the pipeline.

## MotionModel

A MotionModel defines how a single object's state evolves over time and how new measurements update that state. It interfaces (predict, update, pos) so the rest of the system doesn't care what motion model is used. Each concrete motion model encapsulates its own internal state representation (e.g., Kalman filter matrices, noise parameters, covariance) and is responsible for producing the best current position estimate used by TrackManager for association.

## P3Model

P3Model is a 3D position-only motion model, since its transitions function is just the identity matrix, its guess is just its last positions. This results in a nearest neighbor like baseline.

## V6Model

The V6Model implements a 6D constant-velocity Kalman filter, predicting the next position by propagating the current velocity forward each frame. It consumes only 3D position measurements, using the update step to correct both position and velocity through the filter's covariance coupling. This model works well for smooth motion and can be made more responsive to maneuvering by increasing the process noise parameter.

## A9Model

The A9Model implements a 9D constant-acceleration Kalman filter with state, predicting motion by integrating acceleration into velocity and velocity into position each frame. It consumes only 3D position measurements, so velocity and acceleration are inferred indirectly via the filter's dynamics and covariance, it should better handle curving/accelerating targets, it can be made more responsive to maneuvering by increasing the process noise parameter.