Sensor Fusion using Nonlinear Navigation Filters

Ashwin Disa
Robotics Engineering
Worcester Polytechnic Institute
Email: amdisa@wpi.edu

I. Introduction

Accurate and reliable state estimation is fundamental for the autonomy of aerial robots, particularly quadcopters operating in GPS-denied or cluttered environments. Inertial Measurement Units (IMUs) provide high-frequency measurements of angular velocity and linear acceleration, but these readings are corrupted by sensor noise and time-varying biases that, if uncorrected, lead to rapid drift. Conversely, vision-based pose measurements offer drift-free observations of position and orientation at lower rates. Fusing these complementary data sources through nonlinear estimation techniques enables real-time, drift-corrected pose estimation critical for tasks such as obstacle avoidance, mapping, and precision maneuvering.

In this project, we implement and evaluate two classes of nonlinear navigation filters to estimate the full fifteen-state pose vector which includes three-axis position, Z–X–Y Euler angles, linear velocity, and gyroscope and accelerometer biases using a combined inertial and vision-based measurement model.

First, a nonlinear Extended Kalman filter (EKF) is designed around a continuous-time process model that accounts for IMU noise, bias drift, and gravity compensation. The filter propagates the state and covariance through first-order linearization. Second, a Particle Filter is implemented using a vectorized approach to efficiently handle thousands of samples, incorporating the same noise covariance parameters to allow direct comparison between methods. Both filters utilize the vision-based measurement model developed in the previous assignment, which provides noisy observations of position and orientation for correction steps.

Each filter's performance is assessed across multiple flight trajectories by comparing estimated poses against motion capture ground truth and raw visual observations. Key evaluation metrics include three-dimensional trajectory visualizations and the mean-square error (MSE) of position estimates. Finally, a comparative discussion highlights the trade-offs in implementation complexity, runtime performance, and estimation accuracy between Kalman-based and particle-based approaches.

II. Methodology

In this implementation, the very first step is to load all camera and marker calibration data before processing any image. A simple parser reads the camera intrinsic matrix, distortion coefficients, and AprilTag layout from a text file. These values are converted into NumPy arrays and stored once at startup to ensure consistent use throughout the experiment.

Similar to previous assignment, for each video frame, we detect AprilTag corners in pixel coordinates and match them to their known positions in the world. A helper routine computes the 3D location of each tag's four corners based on the tag size and grid spacing. We collect these world–pixel point pairs and feed them into OpenCV's PnP solver, which returns the camera's rotation and translation relative to the world frame.

Since the camera is rigidly mounted on the drone with a known offset, we apply that fixed translation and rotation to the PnP result to recover the drone's body-frame position. We then convert the rotation matrix into roll, pitch, and yaw angles, which align with our chosen state representation of position, velocity, and orientation.

A. *Extended Kalman Filter Implementation*

The extended kalman filter maintains a fifteen-dimensional state vector comprising the vehicle position, velocity, Z–X–Y Euler angles, gyroscope biases, and accelerometer biases. Process noise covariance Q and measurement noise covariance R are provided at initialization. If no initial state is given, the state vector is set to zero and the covariance to a small diagonal matrix; otherwise the supplied estimates are used.

During each prediction step, the state is unpacked into its components: position p, velocity v, Euler angles (ϕ, θ, ψ), gyroscope bias b_g, and accelerometer bias b_a. A rotation matrix from body to world frame is formed from the Euler angles. The measured angular velocity and linear acceleration inputs are bias-corrected, and their contributions to the state derivative are computed using gravity compensation and a mapping matrix $G(\phi, \theta, \psi)$. Position is advanced by velocity, velocity by the

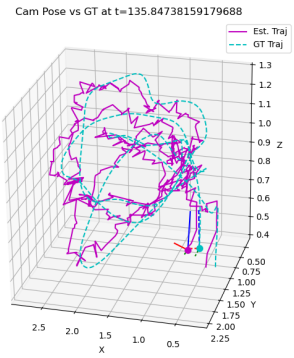
rotated, bias-corrected acceleration plus gravity, and angles by the mapped angular rates. Zero-mean bias derivatives implement a random-walk for the sensor biases. Each derivative is integrated over the time step Δt in an Euler update to advance the state.

A linearized transition matrix F is assembled by placing Δt terms in the velocity-to-position blocks and linearizing the bias corrections in the acceleration and angle blocks. The covariance is then propagated via

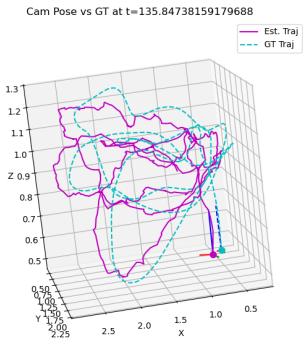
$$P \leftarrow F\,P\,F^{\top} + Q,$$

injecting process uncertainty.

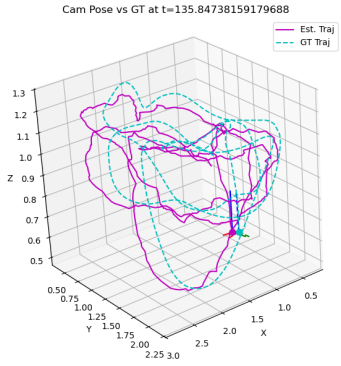
When a new pose measurement arrives, a measurement matrix H extracts the position and orientation components from the state. The residual between measured and predicted values is formed, with orientation differences wrapped into $[-\pi, \pi]$ to avoid discontinuities. The innovation covariance and Kalman gain are computed in the usual EKF fashion, and the state and covariance are corrected. The following comparison shows the plot where the noise R matrix is set with different standard deviations for the position. Lower the std dev, more accurate is the sensor and more weight is given to the measurement value, reflecting results closer to pure PnP (case of a perfect sensor). Higher the std dev values, more noisy is the sensor and the EKF gives more weight to the prediction from the process model. The results are shown in Fig. 1.



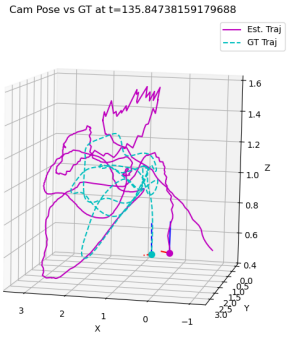
(a) Pure PnP



(b) Position std deviation set to 0.05



(c) Position std deviation set to 0.2



(d) Position std deviation set to 1

Fig. 1: PnP v/s IMU + PnP (EKF) with different standard deviations.

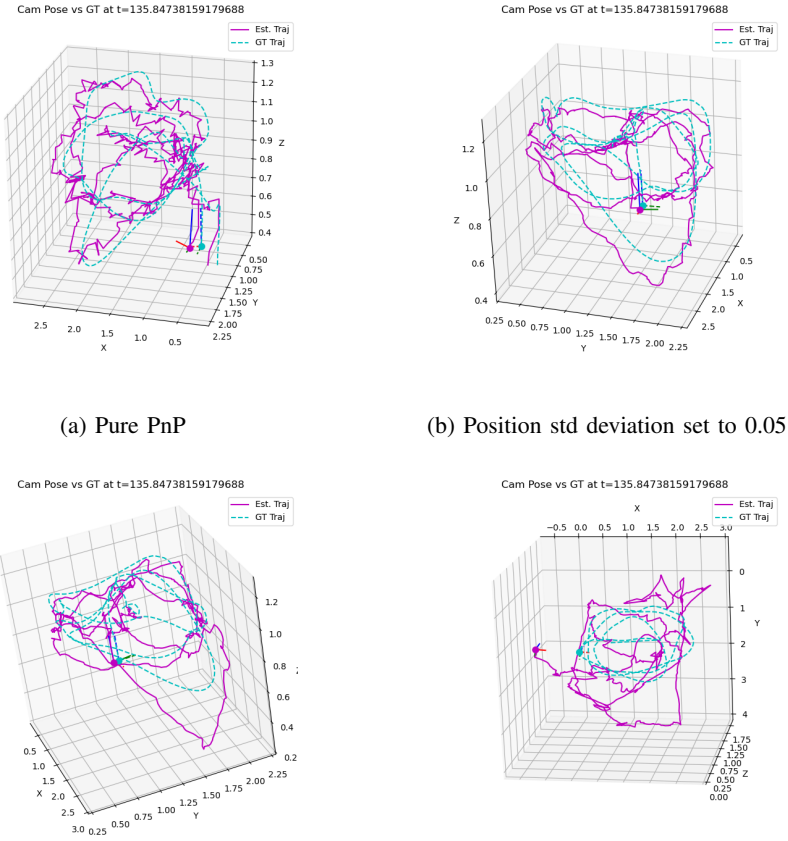
B. Extended Kalman Filter Anaysis

As the assumed measurement noise in the EKF's R matrix for the position channels was increased, the filter naturally shifts more weight onto its internal process model and prior estimate, which produces noticeably smoother, less noisy trajectories compared to raw PnP outputs. However, the downside is that the filter now under-reacts to actual position updates and drifts further from the true path. A "sweet spot" is observed for R where the trajectory is both smooth and remains close to ground truth. To handle this trade-off more systematically, an adaptive noise scheme can be introduced that inflates or deflates the measurement covariance based on the innovation residuals—when residuals grow unexpectedly large, temporarily down-weight the prediction and trust the measurements more, and vice versa.

C. *Particle Filter Implementation*

The particle filter begins by sampling a cloud of fifteen-dimensional state hypotheses from a Gaussian distribution centered at the first available pose estimate from PnP, with covariance equal to the initial predefined uncertainty. Each particle carries its own copy of position, velocity, orientation angles, and sensor biases, and all weights start equal. During each time step, the predict method advances every particle through a constant-velocity model by applying a state transition matrix that shifts position by velocity multiplied by the elapsed time. Process noise drawn from the covariance Q is added to each particle to maintain a spread of possible states and capture model uncertainty.

Once a new pose measurement is obtained via PnP, the update method computes for each particle a six-dimensional predicted observation by extracting its position and Euler angles. The difference between these predictions and the actual measurement forms a residual vector for each particle; the angular components of this residual are wrapped into the interval $[-\pi, \pi]$ to avoid discontinuities. Each particle's weight is then set proportional to the exponential of minus one half the Mahalanobis distance of its residual under the measurement noise covariance R. The weights are normalized so that they sum to one, concentrating probability on particles that better explain the new measurement. The Mahalanobis distance is a way of measuring how far a vector yy lies from the center of a Gaussian distribution with covariance R. Unlike the ordinary Euclidean distance, which treats every direction equally, the Mahalanobis distance "warps" space according to the shape of the covariance: directions in which the distribution is very uncertain count for less, and directions in which it is very certain count for more.



(a) Pure PnP

(b) Position std deviation set to 0.05

(c) Position std deviation set to 0.2

(d) Position std deviation set to 1

Fig. 2: PnP v/s IMU + PnP (PF with 5000 particles) with different standard deviations.

To prevent weight collapse, systematic resampling is performed: a random offset between zero and one over the number of particles is chosen, and equally spaced points in $[0, 1)$ are generated. These points are mapped through the cumulative distribution of weights to select a new set of particles, which are then all assigned uniform weight. The estimate method can report the particle with maximum weight, the arithmetic mean of all particles, or the weighted mean, offering flexibility in how the filter's output is interpreted.

In the main loop over image frames, the filter first computes the elapsed time since the previous frame and calls predict with the measured angular velocity and linear acceleration. After computing the new AprilTag-based measurement, update and resample are invoked when using the particle filter, and the current best state estimate is stored. This cycle of predict, update, and resample continues for each frame, allowing the particle filter to capture complex, non-Gaussian uncertainty patterns in the drone's pose over time. Some results are shown in Fig. 2 to compare the effects of noise from the sensor.

D. Particle Filter Analysis

During the particle-filter experiments, it became clear that matching the EKF's accuracy under the same noise settings required on the order of 4000/5000 particles. With only 250 particles, the mean-squared error ballooned to roughly two to three times higher, and the filter took many frames to concentrate its particle cloud around the true state after startup. This behavior stems from using the process model as the proposal distribution: early on, particles are spread too diffusely and only gradually collapse toward high-likelihood regions as measurements accumulate. To improve both convergence speed and computational efficiency, one can adopt measurement-informed proposals—such as the auxiliary particle filter or an unscented particle filter—that sample particles more directly from the predictive posterior rather than the prior.

TABLE I: Comparison with different number of particles for 3 trajectories

Trajectory	Particles	MSE in position (meters)
Trajectory 1	250	0.561
	500	0.455
	750	0.325
	1000	0.362
	2000	0.301
	3000	0.306
	4000	**0.224**
	5000	0.297
Trajectory 5	250	0.513
	500	0.225
	750	0.172
	1000	0.177
	2000	0.168
	3000	0.169
	4000	**0.161**
	5000	0.162
Trajectory 7	250	0.179
	500	0.165
	750	0.139
	1000	0.137
	2000	0.125
	3000	0.126
	4000	0.132
	5000	**0.124**

E. Results and Comparison discussion

The Extended Kalman Filter leverages a compact, analytic representation of both the process and measurement models. In each cycle, a single 15×1 state vector and 15×15 covariance matrix are propagated through a linearized transition (the F matrix) and updated via a closed-form gain computation. Because the EKF assumes locally Gaussian uncertainties and relies on first-order Taylor expansions, its predict and update steps consist primarily of a few dense matrix multiplies and inversions of a small (6×6) innovation covariance. In practice, this means that—even with moderate sensor noise—the EKF can rapidly converge to a consistent pose estimate with relatively little computational effort. The small, fixed cost per frame makes it well suited to real-time, onboard applications where CPU and memory are constrained.

By contrast, the Particle Filter represents its uncertainty as an ensemble of thousands of independent hypotheses. Each particle carries its own 15-state vector, and every predict step requires propagating all particles through the motion model and adding random draws from the process noise distribution. Likewise, each update entails computing a six-dimensional residual for each particle, evaluating a Mahalanobis distance, exponentiating to obtain weights, and then performing a systematic resample. Although this Monte Carlo approach can capture strongly non-Gaussian or multimodal posterior distributions that the EKF might miss, it incurs a computational cost roughly proportional to the number of particles. While experimenting, we found that fewer than a few thousand particles led to noisy, erratic estimates, whereas on the order of 5000 particles were required to approach the accuracy of the EKF under the same sensor noise settings.

Overall, while the Particle Filter offers greater theoretical flexibility—particularly in highly nonlinear regimes or when the error distribution departs significantly from Gaussian—the practical cost of sampling, weighting, and resampling thousands of particles at each timestep proved prohibitive. The EKF, with its single-vector propagation and matrix-based correction, not only ran an order of magnitude faster but also delivered more reliable pose estimates in our AprilTag-based visual-inertial fusion scenario. Unless one expects significant non-Gaussian effects or multiple hypotheses to persist, the EKF represents a more efficient and robust choice for onboard drone pose estimation.

Summarizing, between the two approaches, the EKF proved significantly simpler to get up and running: its predict–update cycle maps directly onto a handful of matrix multiplications and inversions, so implementing the motion and measurement models required only a few dozen lines of code. By contrast, the PF demanded custom routines for particle management—drawing samples, computing likelihoods, normalizing weights, and performing systematic resampling—which added both code complexity and opportunities for subtle bugs. When it came to parameter tuning, both filters share the same noise covariances Q and R, so adjusting those matrices felt comparably straightforward; the PF did introduce an extra tuning knob in the particle count, but once a sufficiently large ensemble was chosen, further tweaks to Q and R had similar effects in both methods. In terms of runtime, the EKF runs an order of magnitude faster than the PF and correcting a single state vector versus iterating over thousands of particles each frame—making the EKF far more suitable for real-time onboard processing. Finally, accuracy measurements confirmed that the EKF delivered tighter tracking of the ground truth than the PF with a practical number of particles; although the PF can in principle capture non-Gaussian uncertainties, matching the EKF's performance required on the order of 5000 particles, which compounded its computational burden without yielding noticeably better estimates.

TABLE II: Comparison of MSE between PnP, EKF and PF

Trajectory	**PnP**	**EKF**	**PF (5000)**
Trajectory 1	0.178	**0.177**	0.209
Trajectory 2	**0.156**	0.159	0.170
Trajectory 3	**0.151**	0.155	0.295
Trajectory 4	**0.126**	0.131	0.181
Trajectory 5	0.148	**0.147**	0.149
Trajectory 6	0.122	**0.113**	0.118
Trajectory 7	**0.097**	0.150	0.119
Trajectory 8	**0.095**	0.117	0.120

Below is the comparison between pure PnP, EKF and PF implementation for all trajectories including their position MSE and orientation Roll, Pitch and Yaw from Fig. 3 to Fig. 26.

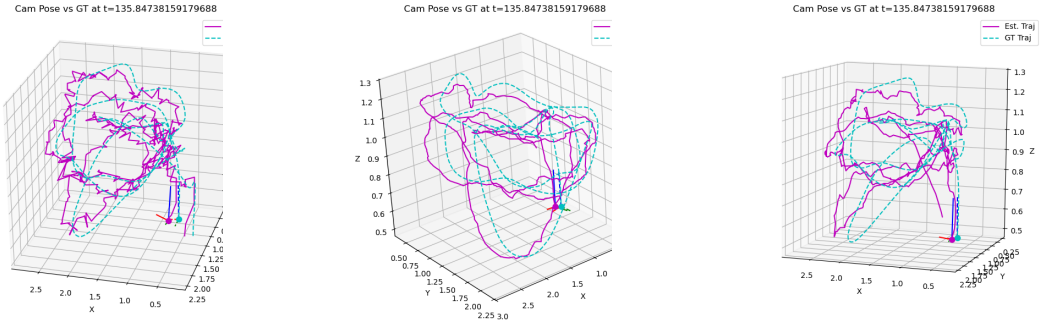
III. CONCLUSION

In this work, implemented and evaluated two classes of nonlinear navigation filters— Extended Kalman Filter (EKF) and a Particle Filter (PF)—for fusing inertial measurements with AprilTag-based vision observations to estimate a 15-state pose vector (position, velocity, orientation, and IMU biases) aboard a quadcopter. Goal is to assess each filter's estimation accuracy, and sensitivity to noise assumptions under identical process and measurement noise covariances.

Quantitatively, the EKF consistently matched or outperformed both pure PnP and PF estimates across eight diverse flight trajectories. As shown in Table II, the mean-square position error (MSE) of the EKF was on average within 0.01 m of pure PnP, whereas the PF—even with 5000 particles—exhibited higher MSE in 5 of 8 trajectories (e.g., 0.177 m for EKF vs. 0.209 m for PF on Trajectory 1) . The EKF's tightly constrained Gaussian assumption and first-order linearization delivered both smoothness and fidelity to ground truth.

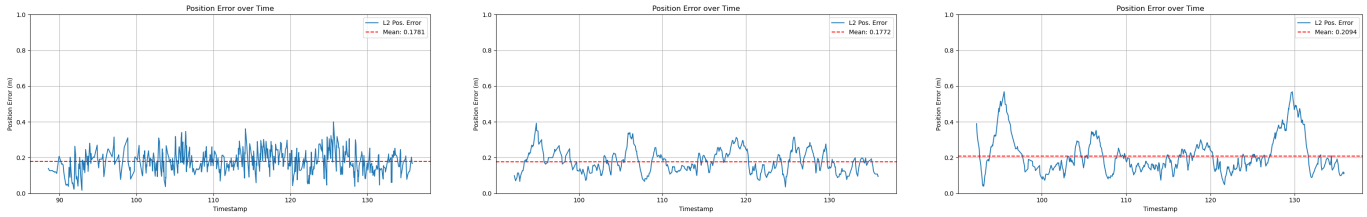
In contrast, the PF's ability to represent non-Gaussian and multimodal uncertainty came at a steep computational cost. Analysis revealed that fewer than 1000 particles yielded MSE two to three times larger than the EKF, and matching EKF accuracy required roughly 4000–5000 particles (MSE $\approx$ 0.224 m with 4000 particles) as shown in Table I.

Overall, while the PF offers theoretical advantages in highly nonlinear or multimodal scenarios, experiments indicate that, for AprilTag-based visual-inertial fusion, the EKF strikes the best balance of simplicity, robustness, and real-time performance. Future extensions may explore adaptive noise schemes within the EKF or measurement-informed proposal distributions for PFs, but under the tested conditions, the EKF remains the preferred choice for precise, drift-corrected drone navigation.



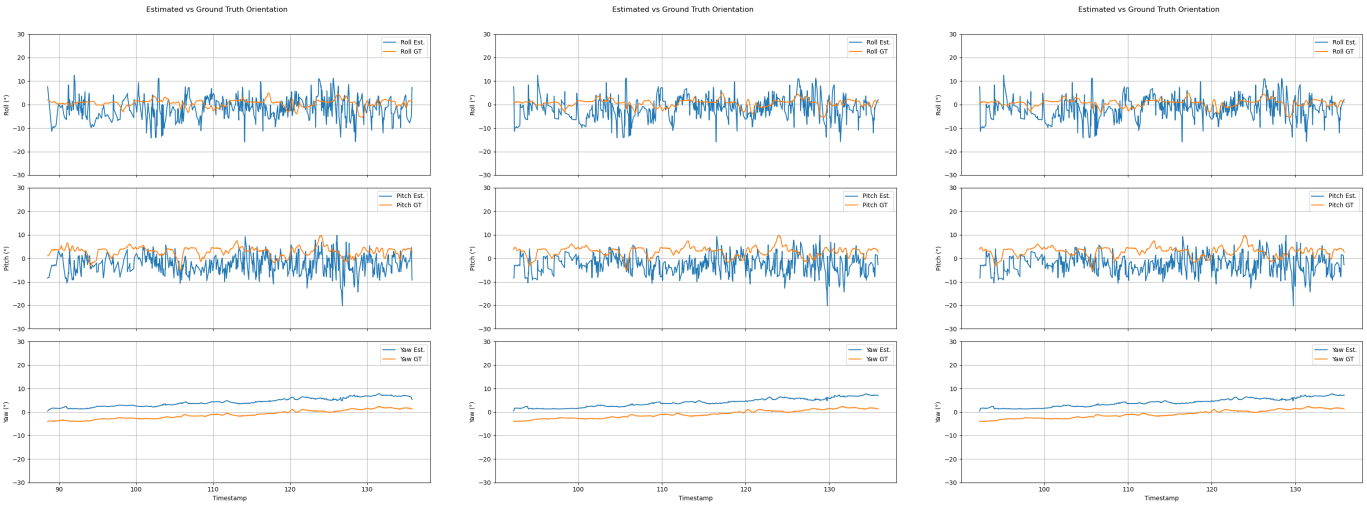
(a) PnP (b) EKF (c) PF

Fig. 3: Trajectory 1.



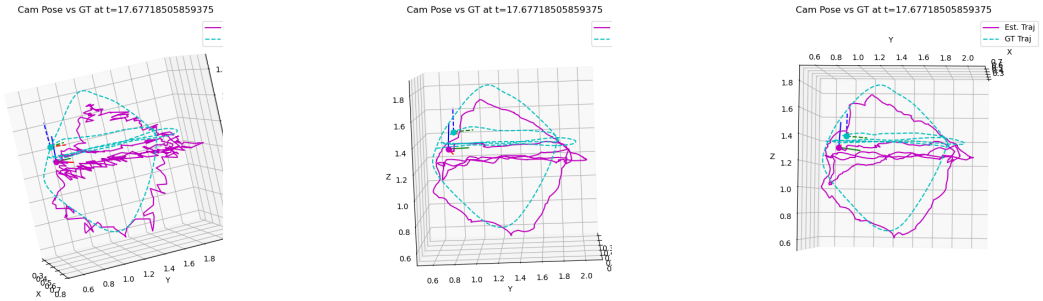
(a) PnP (b) EKF (c) PF

Fig. 4: MSE comparison for trajectory 1.



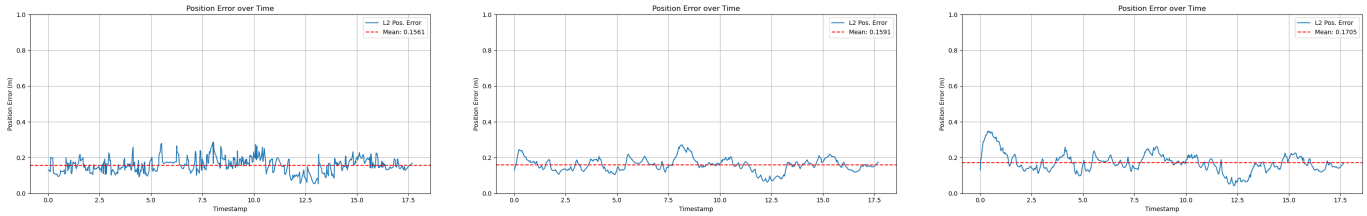
(a) PnP (b) EKF (c) PF

Fig. 5: Roll, Pitch, Yaw comparison for trajectory 1.



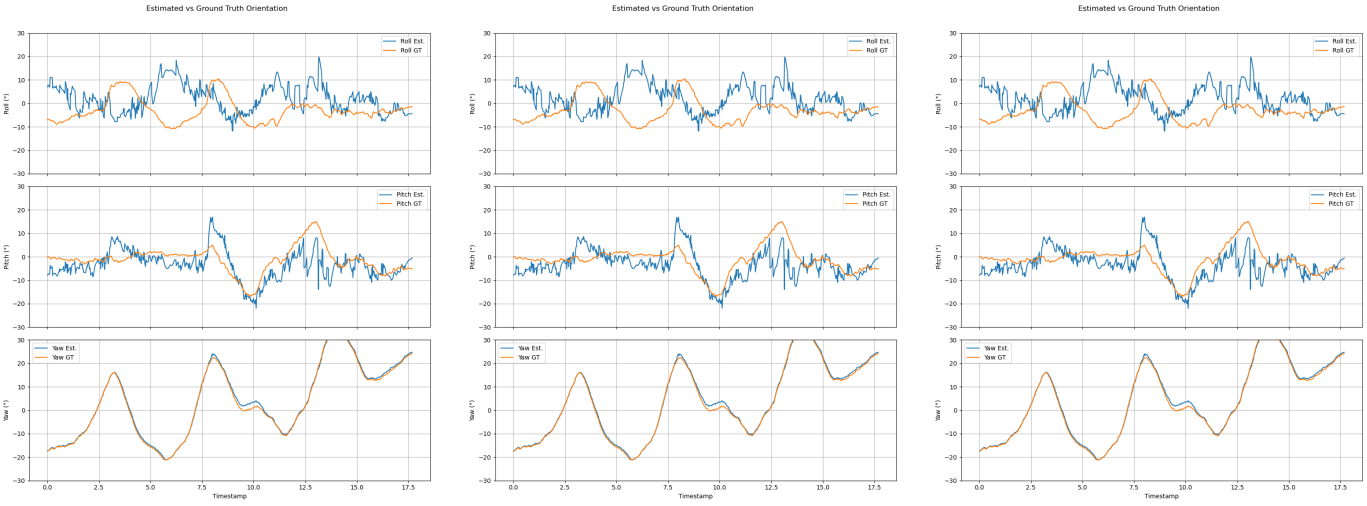
(a) PnP (b) EKF (c) PF

Fig. 6: Trajectory 2.



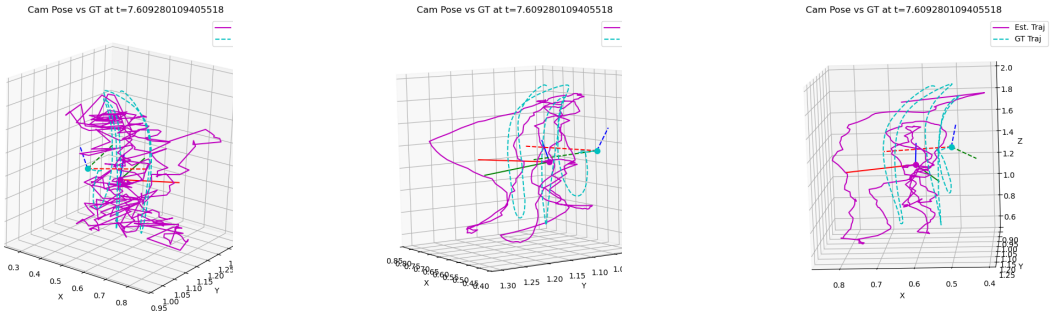
(a) PnP (b) EKF (c) PF

Fig. 7: MSE comparison for trajectory 2.



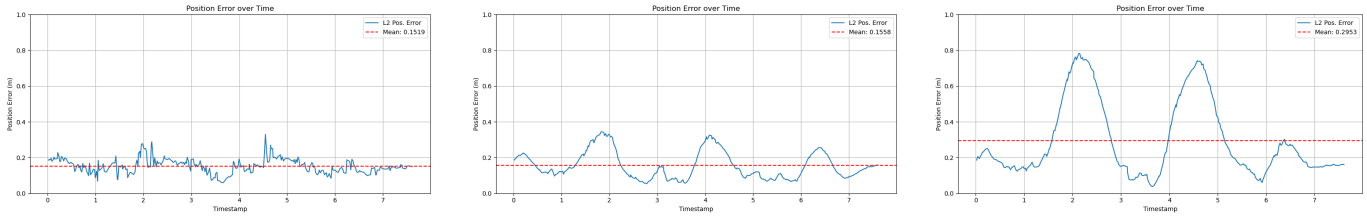
(a) PnP (b) EKF (c) PF

Fig. 8: Roll, Pitch, Yaw comparison for trajectory 2.



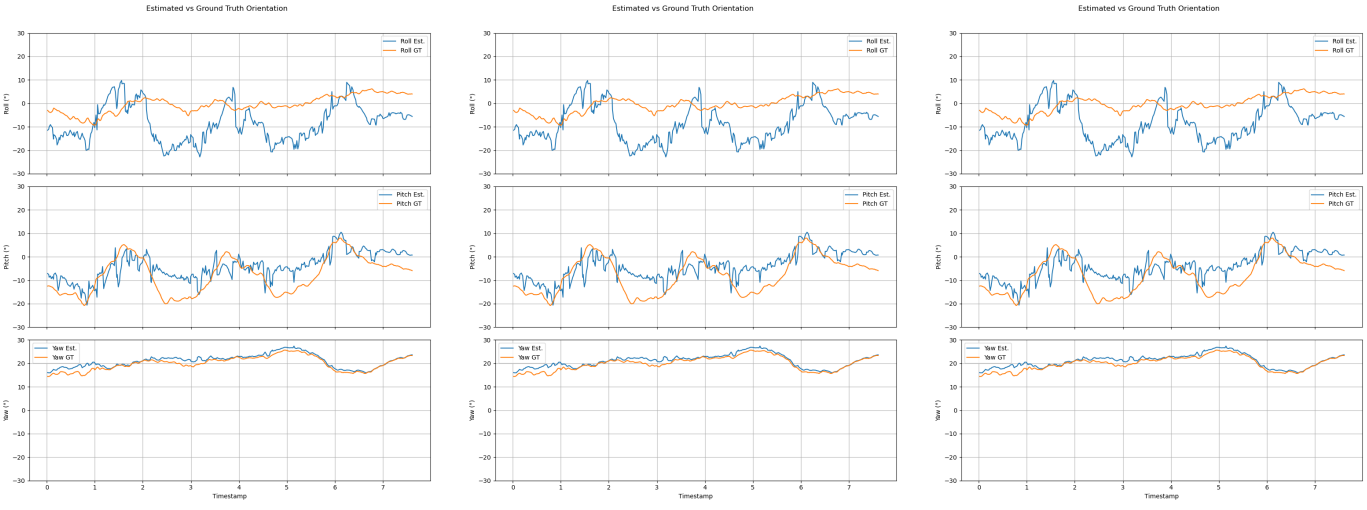
(a) PnP (b) EKF (c) PF

Fig. 9: Trajectory 3.



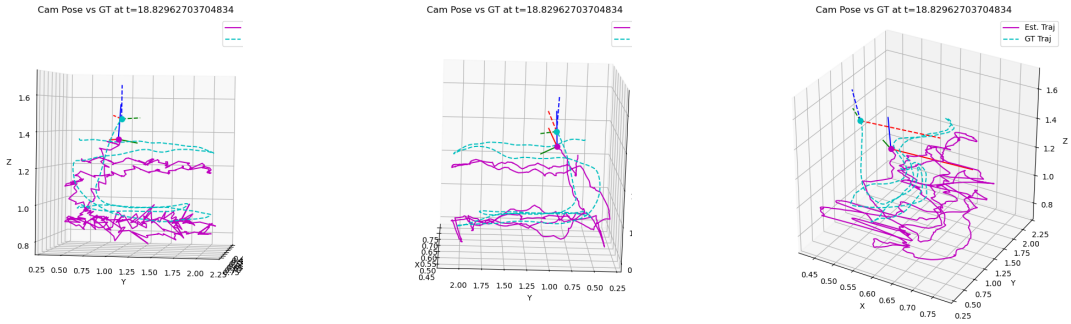
(a) PnP (b) EKF (c) PF

Fig. 10: MSE comparison for trajectory 3.



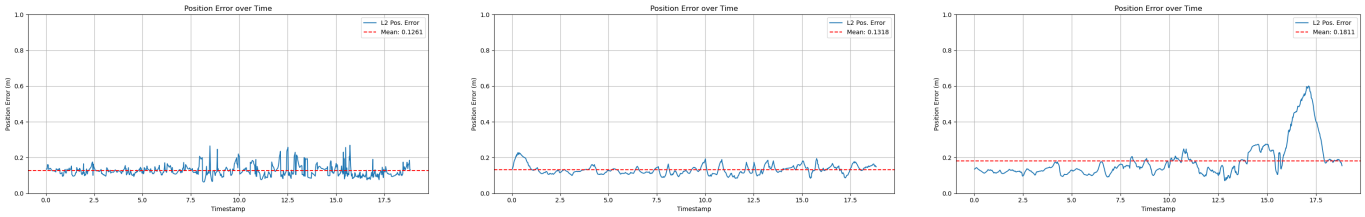
(a) PnP (b) EKF (c) PF

Fig. 11: Roll, Pitch, Yaw comparison for trajectory 3.



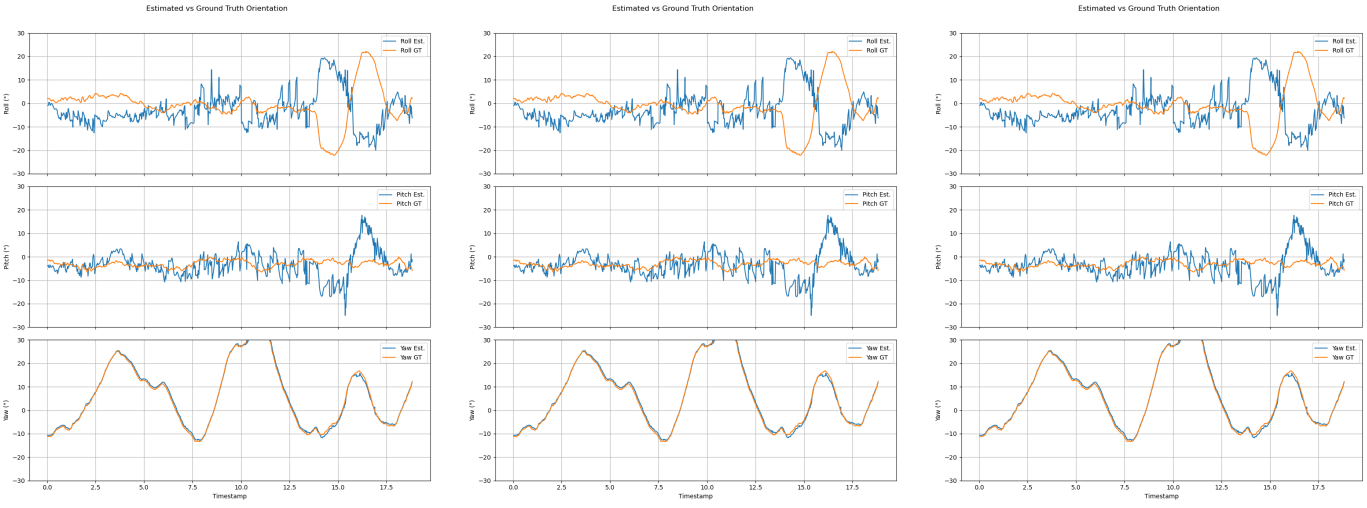
(a) PnP (b) EKF (c) PF

Fig. 12: Trajectory 4.



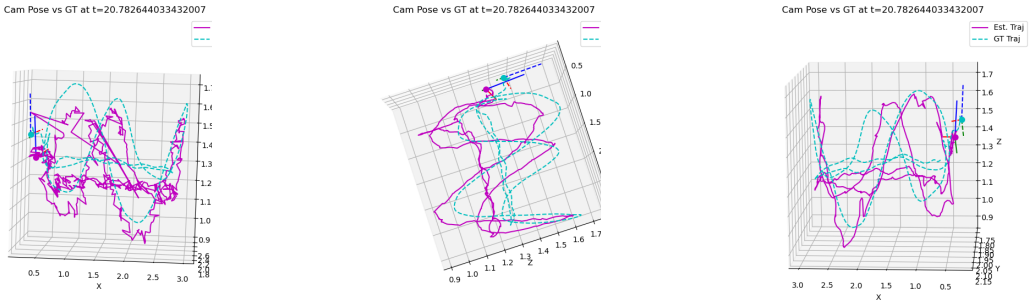
(a) PnP (b) EKF (c) PF

Fig. 13: MSE comparison for trajectory 4.



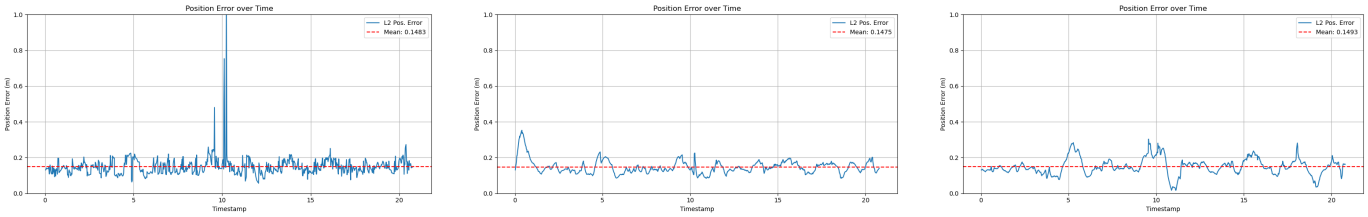
(a) PnP (b) EKF (c) PF

Fig. 14: Roll, Pitch, Yaw comparison for trajectory 4.



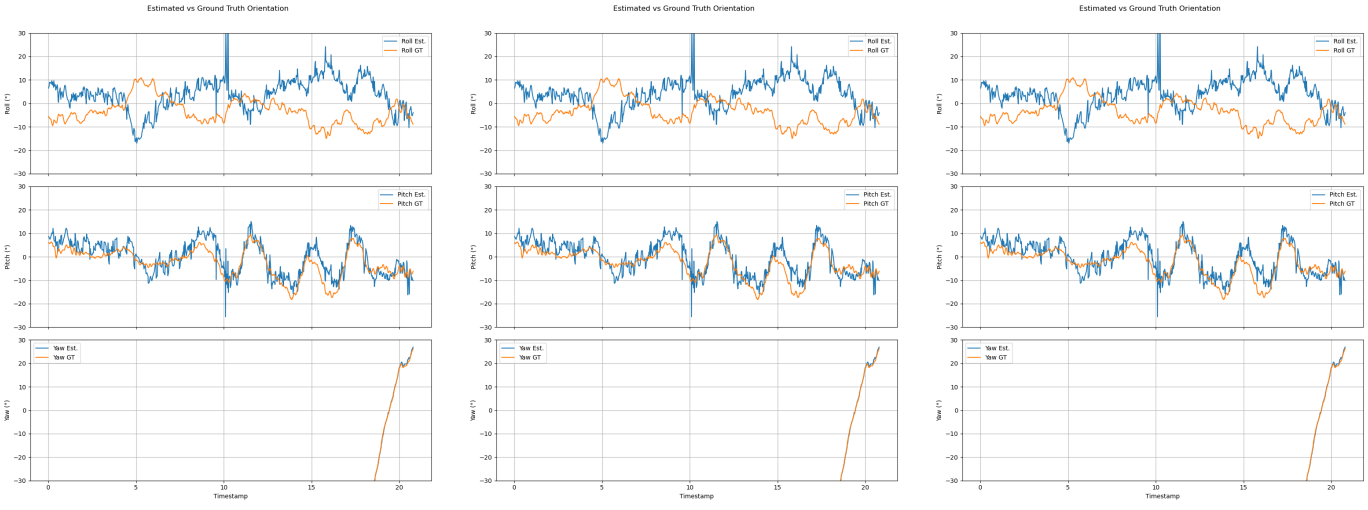
(a) PnP (b) EKF (c) PF

Fig. 15: Trajectory 5.



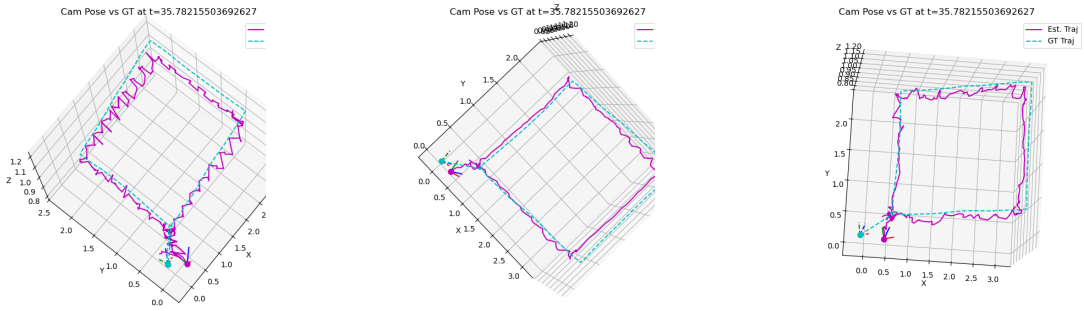
(a) PnP (b) EKF (c) PF

Fig. 16: MSE comparison for trajectory 5.



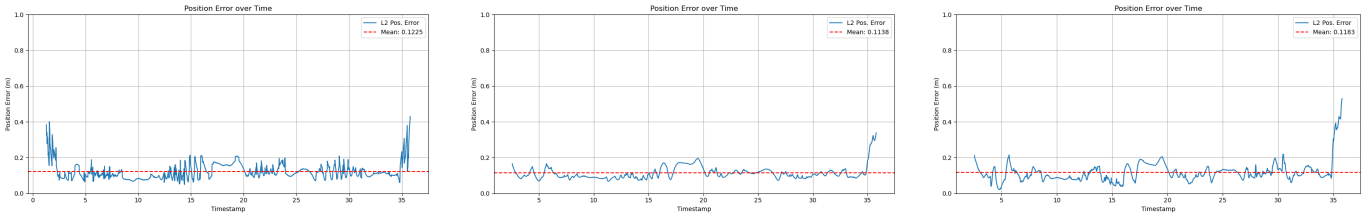
(a) PnP (b) EKF (c) PF

Fig. 17: Roll, Pitch, Yaw comparison for trajectory 5.



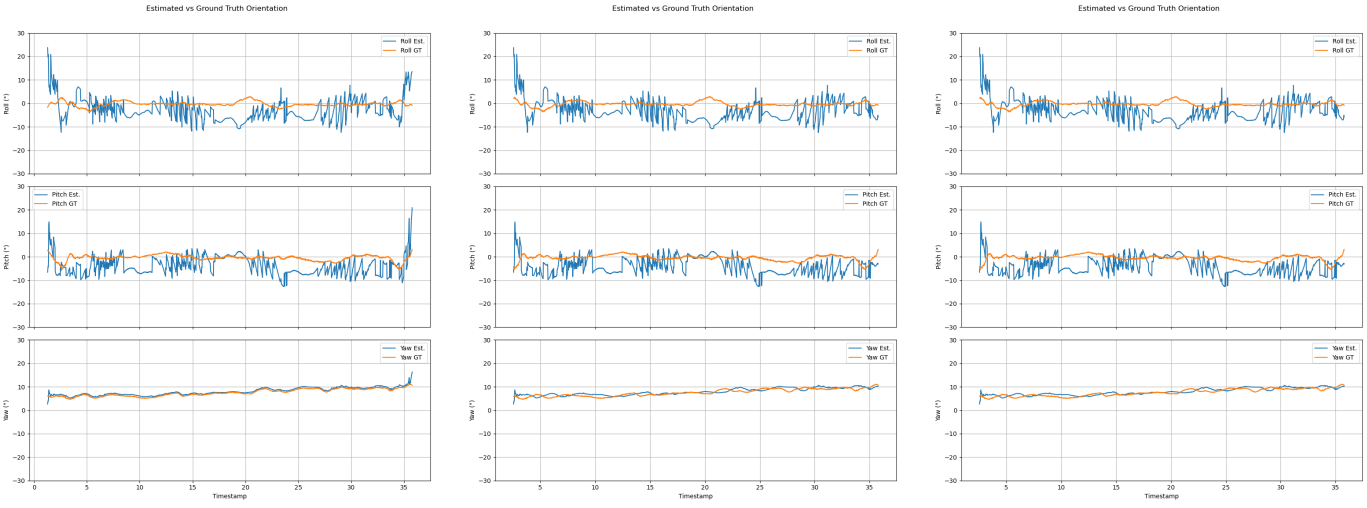
(a) PnP (b) EKF (c) PF

Fig. 18: Trajectory 6.



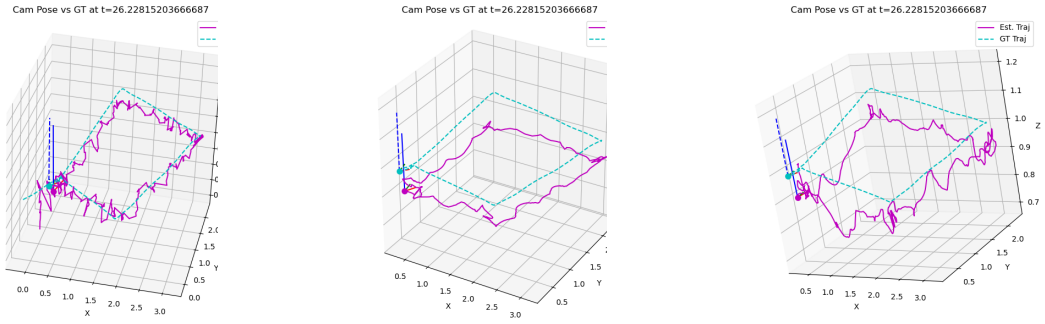
(a) PnP (b) EKF (c) PF

Fig. 19: MSE comparison for trajectory 6.



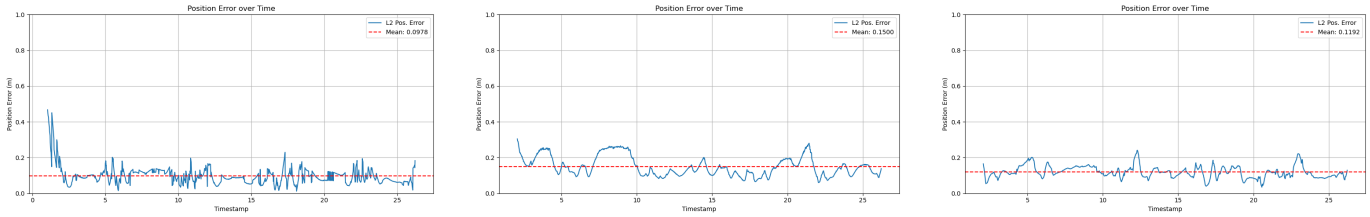
(a) PnP (b) EKF (c) PF

Fig. 20: Roll, Pitch, Yaw comparison for trajectory 6.



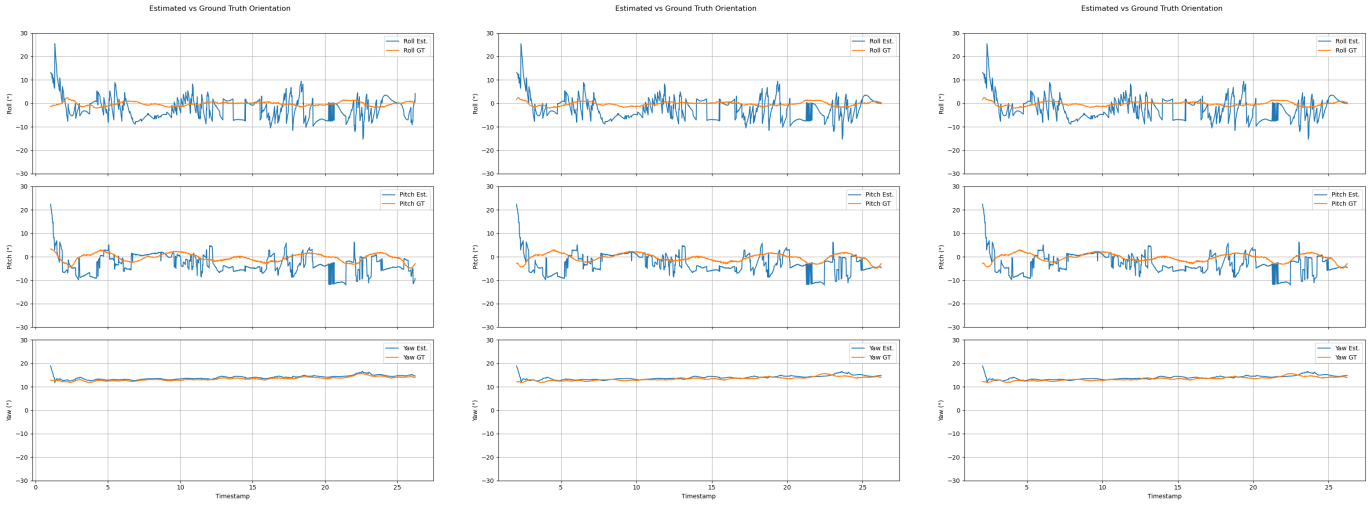
(a) PnP (b) EKF (c) PF

Fig. 21: Trajectory 7.



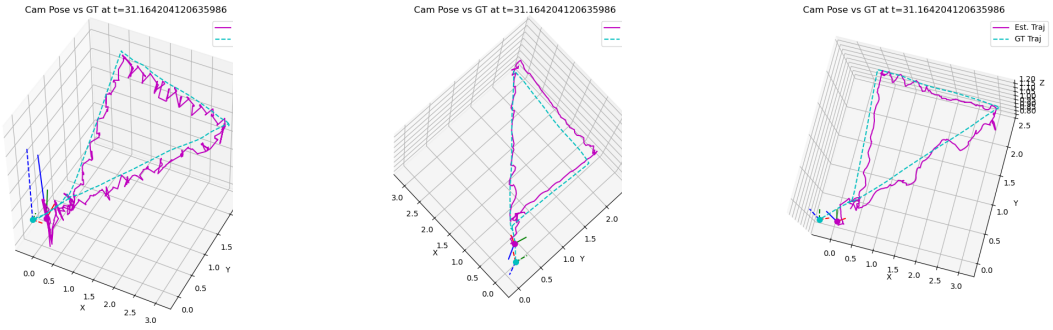
(a) PnP (b) EKF (c) PF

Fig. 22: MSE comparison for trajectory 7.



(a) PnP (b) EKF (c) PF

Fig. 23: Roll, Pitch, Yaw comparison for trajectory 7.

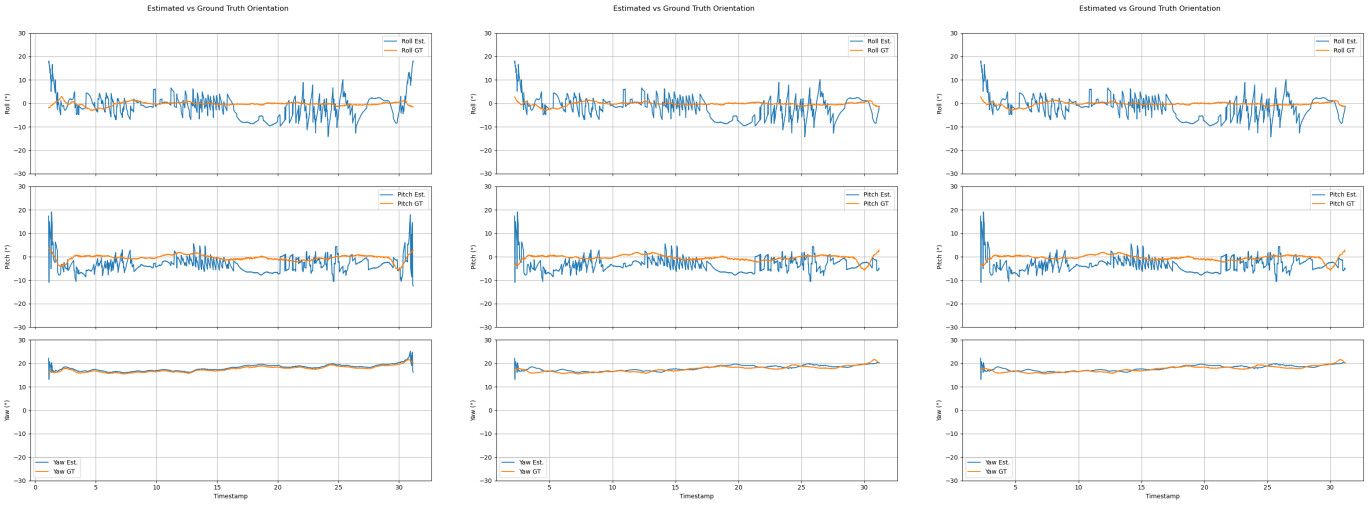


(a) PnP (b) EKF (c) PF

Fig. 24: Trajectory 8.

(a) PnP (b) EKF (c) PF

Fig. 25: MSE comparison for trajectory 8.



(a) PnP (b) EKF (c) PF

Fig. 26: Roll, Pitch, Yaw comparison for trajectory 8.