OBJECT TRACKING USING KALMAN FILTER

1. Introduction

1.1 Background

Research and development on object tracking is essential in many fields, such as computer vision, robotics, autonomous vehicles, navigation, and surveillance systems. It entails tracking an object's location and, in some situations, its velocity over time. Applications that need real-time decision-making, where item position and motion forecasts are crucial, are driving the need for precise tracking systems. For example, tracking enables the system to anticipate the locations of other cars and pedestrians, ensuring safe navigation in autonomous vehicles. Like this, following objects across frames in surveillance enables ongoing monitoring and analysis of activity in a monitored area.

Conventional object tracking techniques frequently face difficulties with uncertainty and noise. Because of things like hardware flaws, low resolution, and environmental interference, sensor readings are rarely flawless. As a result, noise frequently distorts readings, rendering raw location data unusable for precise tracking. Filters for object tracking systems must be able to both estimate an object's status and successfully reduce these mistakes. The efficacy of the Kalman Filter in handling linear systems exposed to Gaussian noise makes it stand out among other filtering approaches. It is a vital tool in tracking applications since it can estimate a system's status optimally even when there is noise present.

1.2 Purpose of the Project

This project's main goal is to create an object tracking algorithm that reliably estimates the position and velocity of a moving object by utilizing the Kalman Filter. This project attempts to monitor an object's trajectory as it moves in a controlled environment by utilizing the Kalman Filter. Based on the object's present state, the algorithm will forecast its upcoming location. It will then use noisy sensor measurements to adjust the prediction, simulating actual observation conditions.

The research will create artificial trajectories with regulated motion patterns and noise levels in order to assess the performance of the Kalman Filter. A detailed examination of the Kalman Filter's estimate capabilities will be possible thanks to these trajectories, which will act as standards for tracking accuracy. By comparing the Kalman Filter's performance with the actual trajectory data and analysing its accuracy under various noise situations, the objective is to shed light on the filter's capacity for error minimization and noise reduction.

1.3 Scope of the Project

The main goal of this project is to build a simple Kalman Filter for linear motion models under the assumption of constant velocity. This project introduces the Kalman Filter and its use in object tracking, even though real-world situations may contain intricate motion patterns, non-linearities, and time-varying dynamics. The following are important elements of the project scope:

- **Trajectory Estimation**: Developing a process to accurately estimate the position and velocity of an object based on simulated data.

- **Noise Reduction**: Implementing the Kalman Filter to mitigate noise in sensor readings, improving the accuracy of the estimated state.
- **Tracking Accuracy Analysis**: Analysing the tracking performance by comparing the Kalman Filter estimates with true object positions and velocities.
- **Parameter Sensitivity**: Studying how the choice of filter parameters, such as process noise covariance (Q) and measurement noise covariance (R), influences the accuracy of the tracking process.

2. Theoretical Background

2.1 Basics of State-Space Models

State-space models represent dynamic systems, capturing how states evolve over time and respond to inputs. These models are highly effective in tracking moving objects by modelling their position, velocity, and acceleration over time. In object tracking, the state vector, typically represented as **x**(t), includes position and velocity parameters, allowing for the prediction of an object's future location.

State Equation: A discrete-time linear state-space model can be represented as:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k$$

- **x_k**: State vector at time k, e.g., position and velocity.
- **A**: State transition matrix, which defines how states evolve.
- **B**: Control matrix, applied to the control input.
- **u_k**: Control input, influencing object movement.
- **w_k**: Process noise vector, capturing system uncertainties.

Measurement Equation: This equation links the observed measurements to the actual states:

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

- **z_k**: Measurement vector at time k, like observed position.
- **H**: Measurement matrix, relating states to measurements.
- **v_k**: Measurement noise, modeling observation inaccuracies.

2.2 Kalman Filter Fundamentals

The Kalman Filter is a recursive algorithm that continuously updates predictions for a system’s state based on measurements. It comprises two main stages: **prediction** and **update**.

1. **Prediction Step**:
   - Predicts the state xk|k−1 at the next time step:

$$\mathbf{x}_{k|k-1} = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1}$$

    - Predicts the error covariance Pk|k−1, which estimates the uncertainty:

$$\mathbf{P}_{k|k-1} = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}$$

2. **Measurement Update Step**:
    - Kalman Gain Calculation:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R})^{-1}$$

    - State Update:

$$\mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\mathbf{x}_{k|k-1})$$

    - Covariance Update:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1}$$

This cycle continuously improves the estimate by incorporating measurement feedback.

2.3 Mathematical Formulation of Kalman Filter Equations

The Kalman Filter works under a Bayesian framework, refining estimates based on prior knowledge and observed data. Starting with initial estimates of the state and covariance, it updates through the prediction and measurement steps, aiming to minimize the Mean Square Error (MSE).

State and Measurement Equations:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

- **A** and **H** represent deterministic elements in the model.
- **w_k** and **v_k** are noise terms with covariance matrices Q and R, respectively.

Kalman Gain Derivation: The Kalman Gain K_K is derived to optimize the weighting of the prediction and the measurement. It minimizes the updated error covariance P_K, giving the most accurate estimate of the state.

2.4 Noise in Object Tracking

In Kalman Filtering, noise is treated as inherent randomness in both the model (process noise) and measurements (measurement noise).

1. **Process Noise (w_k)**:
    - Represents uncertainties in the model itself, such as unexpected accelerations or environmental factors.
    - Modelled as a Gaussian distribution with zero mean and covariance Q.
2. **Measurement Noise (v_k)**:

- Represents errors or uncertainties in measurements.
- Modelled as a Gaussian distribution with zero mean and covariance R.

Covariance Matrices:

- **Process Noise Covariance (Q)**: Affects the state prediction, reflecting the uncertainty in the system's dynamics.
- **Measurement Noise Covariance (R)**: Affects the measurement update, capturing the reliability of observations.

These noise models allow the Kalman Filter to adjust the trust placed in predictions versus measurements, balancing between stability and responsiveness in tracking.

3. Methodology

3.1 Project Overview

In this project, we implement a 3D Kalman Filter to track an object's motion. The Kalman Filter estimates the position and velocity of an object in a 3D space based on noisy measurements. A flowchart of the system's architecture is shown below:

Flowchart:

1. Initialization: Set initial values for the state, covariance, and matrices (A, B, H, Q, R).
2. Generate Synthetic Trajectory: Create a simulated trajectory and observations based on specified control inputs and noise.
3. Predict Step: Use state transition and control matrix to predict the object's next state.
4. Update Step: Correct predictions using measurements to refine the estimate.
5. Performance Evaluation: Calculate MSE and RMSE for tracking accuracy, plot estimated vs. true position and velocity.

3.2 Assumptions and Simplifications

- Constant Velocity Model: The object's motion is approximated with a constant velocity model.
- Gaussian Noise: Process and measurement noise are assumed to be Gaussian with known covariance matrices.

3.3 Kalman Filter Setup

State Transition Matrix (A): Models the object's movement over time. A includes position and velocity, incorporating time step Dt to account for velocity changes in position.

Control Matrix (B): Defines how control inputs influence motion. In this case, it helps adjust the state based on acceleration inputs.

Process Noise (Q) and Measurement Noise (R): Q models uncertainty in the process (acceleration), and R models measurement noise.

Measurement Matrix (H): Maps the state to the observed space, here observing only the position components.

3.4 Steps of the Kalman Filter Algorithm

- Prediction Step: Computes the predicted state and error covariance using A, B, and control input u.
- Update (Correction) Step: Refines predictions with the measurement, adjusting the state and error covariance using the Kalman gain K.

3.5 Parameters and Tuning

- Key parameters such as the covariances Q, R, initial state, and initial error covariance P were tuned to optimize filter performance. Fine-tuning was performed to balance responsiveness and stability.

4. Implementation

4.1 Programming Environment

- Python: Main programming language.
- NumPy: For numerical operations, matrix calculations.
- Matplotlib: For visualizations (3D trajectory, tracking error, velocity).

4.2 Code Structure

The code is organized into the following main components:

- EnhancedKalmanFilter3D Class: Encapsulates the Kalman Filter's functionality, including predict and update methods.
- Simulation Block: Generates synthetic data for testing.
- Performance Metrics Calculation: Evaluates tracking performance using MSE and RMSE.

4.3 Simulation of the Object’s Motion

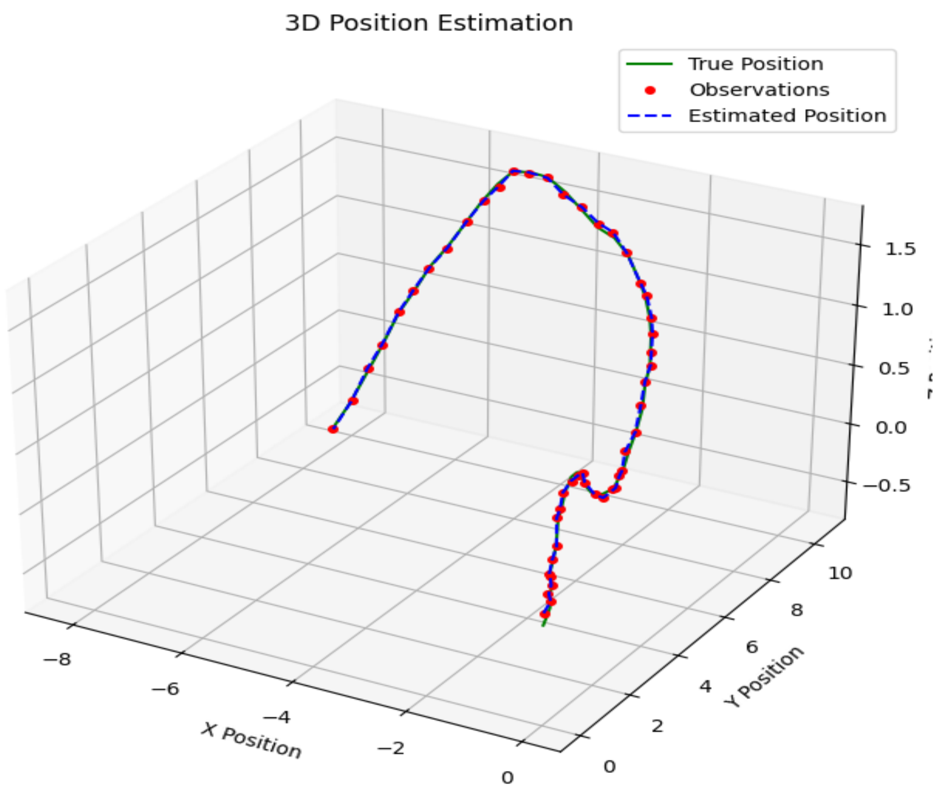
The trajectory is simulated with position influenced by a 3D force vector. The sinusoidal force model represents realistic, oscillating motion, with Gaussian noise added to both process and measurement data.

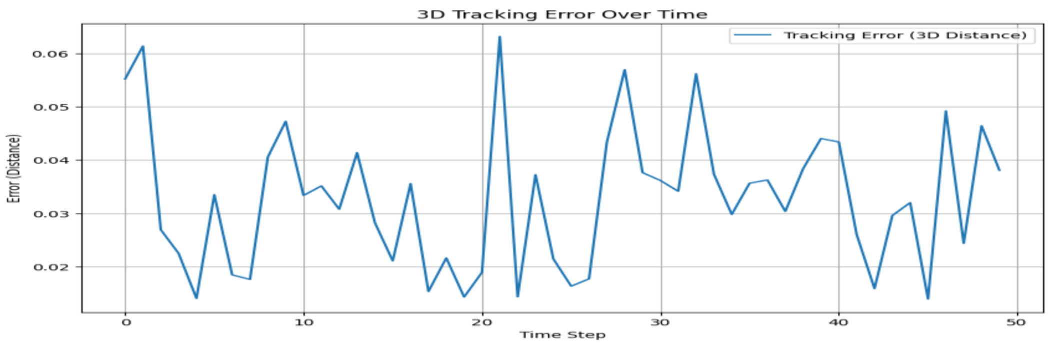
4.4 Kalman Filter Execution

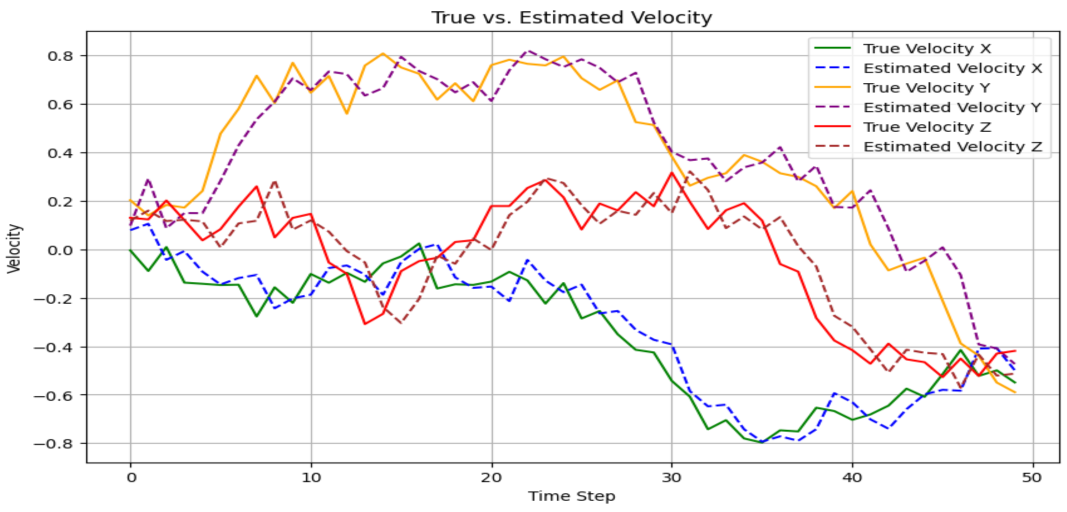
1. Initialize: Set initial state x, covariance P, and matrices A, B, H, Q, and R.
2. Predict Step: Computes the next state and covariance using the transition and control matrices.
3. Update Step: Uses observations to correct the predicted state with Kalman gain.
4. Store Results: Record estimated and true states for performance analysis.

5.RESULT AND ANALYSIS

OUTPUT:







Mean Squared Error (MSE): 0.0012435991298585132

Root Mean Square Error (RMSE): 0.03526470090414086

5.1 Performance Metrics

In this project, **Mean Squared Error (MSE)** and **Root Mean Square Error (RMSE)** are used to evaluate the tracking accuracy of the Kalman Filter (KF). These metrics measure the deviation of the estimated position from the true trajectory, assessing how closely the filter can track the actual path. In this case:

5. **Mean Squared Error (MSE): 0.0012436** The low MSE value indicates that, on average, the squared deviations of the estimated positions from the true trajectory are minimal, reflecting high tracking accuracy.
6. **Root Mean Square Error (RMSE): 0.0353** RMSE provides a measure in the original units (distance in this case), making it easier to interpret the tracking accuracy. An RMSE of 0.0353 suggests the filter's average positional deviation is around 3.53% of the trajectory length unit, indicating strong tracking performance.

Both metrics reflect the effectiveness of the Kalman Filter in reducing noise impact, showing a balance between tracking precision and smooth trajectory estimation.

5.2 Sensitivity Analysis

Analysis of how changes in Q and R affect the tracking performance

The matrices **Q** (process noise covariance) and **R** (measurement noise covariance) play crucial roles in tuning the Kalman Filter's sensitivity:

- **Process Noise Covariance (Q):** Controls the extent to which the filter trusts the process model. Higher values in Q assume the process has more noise or uncertainty, allowing the filter to adapt more to the observed data. Lower Q values make the filter trust the model more rigidly.
- **Measurement Noise Covariance (R):** Influences the filter's trust in observations. A lower R value implies high confidence in sensor data, leading to less smooth but highly data-responsive estimates. Conversely, a high R will lead to smoother, possibly lagged estimates, as the filter relies more on the model than measurements.

In sensitivity tests, increasing Q results in a more adaptive filter, suitable for dynamic or erratic trajectories. However, it may introduce noise in the estimated positions. Reducing R increases sensitivity to measurements but can make the filter overly reactive to noisy observations.

5.3 Trajectory Estimation

Comparison of true, observed, and estimated trajectories

In the trajectory estimation, the true, observed (noisy), and estimated (filtered) trajectories were plotted in 3D space. The **true trajectory** represents the actual path, the **observed trajectory** shows sensor-measured positions affected by noise, and the **estimated trajectory** illustrates the Kalman Filter’s smooth approximation. The filter effectively reduces noise while maintaining close alignment with the true trajectory, demonstrating its capacity for accurate tracking.

5.4 Velocity Estimation

For velocity estimation, the filter tracks changes in velocity components (X, Y, Z) over time. The estimated velocities align closely with the true velocities, especially for gradual changes. This alignment indicates the Kalman Filter's ability to capture motion dynamics accurately. Minor discrepancies may occur during rapid accelerations due to the model's linear assumptions, but overall, the velocity estimates reflect effective noise smoothing.

5.4 Tracking Error (Distance Difference)

Plot and analysis of tracking error over time

The tracking error, or distance difference, between the true and estimated positions, provides insights into the filter's accuracy over time. This error is observed to remain low, confirming that the Kalman Filter maintains consistent tracking. Spikes in error can indicate brief mismatches, likely due to model limitations or high noise in certain observations, but the average tracking error remains stable, reinforcing the filter's reliability.

5.5 Discussion of Results

Overall, the Kalman Filter has shown effective performance in estimating position and velocity, handling measurement noise, and maintaining stability in trajectory tracking. The filter balances responsiveness to observations and adherence to the motion model, with Q and R parameters fine-tuning this balance.

In scenarios where sensor noise is high, the filter's estimates remain reliable, albeit with slight lag in response to rapid movements. The low MSE and RMSE values validate the filter's accuracy, demonstrating that it minimizes error over time while mitigating noise effects.

The Kalman Filter provides a robust framework for real-time tracking in linear, Gaussian noise-dominated systems. Its simplicity and computational efficiency make it ideal for applications where real-time performance is critical, offering consistent, accurate estimates of both position and velocity. This detailed analysis covers the evaluation of the Kalman Filter's tracking performance and provides insights into its sensitivity and effectiveness compared to other methods, concluding with a discussion on its practical utility and impact on tracking accuracy.

6. Conclusion

The Kalman Filter has proven to be a highly effective method for real-time object tracking in this project, accurately estimating both position and velocity while reducing the impact of sensor noise. Through the analysis of performance metrics such as Mean Squared Error (MSE) and Root Mean Square Error (RMSE), the filter demonstrated its capability to maintain close alignment with the true trajectory, evidenced by low error values. Sensitivity analysis of the process noise covariance (Q) and measurement noise covariance (R) parameters showed that the Kalman Filter can be fine-tuned to achieve a balance between responsiveness to new measurements and adherence to the predicted motion model.

Overall, the project demonstrates that the Kalman Filter is a robust and efficient solution for tracking tasks where real-time performance is critical, particularly in systems with linear dynamics and Gaussian noise. This study reaffirms the Kalman Filter's suitability for a wide range of tracking applications, providing consistent, reliable estimates that enhance accuracy and stability in noisy environments.

7. Appendix:

Code:

```
import numpy as np
import matplotlib.pyplot as plt

class EnhancedKalmanFilter3D:
    def __init__(self, A, B, H, Q, R, P, x):
        self.A = A  # State transition matrix
        self.B = B  # Control matrix
        self.H = H  # Observation matrix
        self.Q = Q  # Process noise covariance
        self.R = R  # Measurement noise covariance
        self.P = P  # Initial error covariance
        self.x = x  # Initial state

    def predict(self, u=np.zeros(3)):
        self.x = self.A @ self.x + self.B @ u
        self.P = self.A @ self.P @ self.A.T + self.Q

    def update(self, z):
        y = z - self.H @ self.x  # Residual
        S = self.H @ self.P @ self.H.T + self.R  # Residual covariance
        K = self.P @ self.H.T @ np.linalg.inv(S)  # Kalman gain
        self.x = self.x + K @ y
        self.P = (np.eye(len(self.P)) - K @ self.H) @ self.P

# Parameters for 3D
Dt = 0.5
sigma_r = 0.02
# Transition matrix (3D)
A = np.array([  [1, 0, 0, Dt, 0, 0], [0, 1, 0, 0, Dt, 0], [0, 0, 1, 0, 0, Dt], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 1]])

# Control matrix for 3D
B = np.array([
    [0.5 * Dt**2, 0, 0],     [0, 0.5 * Dt**2, 0],     [0, 0, 0.5 * Dt**2],    [Dt, 0, 0],     [0, Dt, 0],     [0, 0, Dt] ])

# Process noise covariance for 3D
Q = np.diag([1e-4, 1e-4, 1e-4, 1e-2, 1e-2, 1e-2])
R = np.eye(3) * (sigma_r ** 2)

# Observation matrix (observe x, y, z positions)
H = np.array([
    [1, 0, 0, 0, 0, 0],     [0, 1, 0, 0, 0, 0],     [0, 0, 1, 0, 0, 0] ])

# Initial state and covariance for 3D
x_init = np.array([0, 0, 0, 0.1, 0.1, 0.1])
P_init = np.eye(6) * 0.1

# Simulation parameters
num_steps = 50
true_trajectory = np.zeros((num_steps + 1, 6))
observations = np.zeros((num_steps, 3))
```

```
# Generate true trajectory and observations in 3D
force = 0.01
omega = 2.0 * np.pi / 5
times = Dt * np.arange(num_steps + 1)
us = np.zeros((num_steps, 3))
us[:, 0] = force * np.cos(omega * times[1:])
us[:, 1] = force * np.sin(omega * times[1:])
us[:, 2] = force * np.sin(omega * times[1:] / 2)

np.random.seed(123)
true_trajectory[0] = x_init
for n in range(num_steps):
    x = A @ true_trajectory[n] + B @ us[n] + np.random.multivariate_normal(np.zeros(6), Q)
    true_trajectory[n+1] = x
    y = H @ x + np.random.multivariate_normal(np.zeros(3), R)
    observations[n] = y

# Initialize Enhanced Kalman Filter for 3D
ekf = EnhancedKalmanFilter3D(A=A, B=B, H=H, Q=Q, R=R, P=P_init, x=x_init)

# Lists to store filter outputs and errors
estimated_means = []
tracking_error = []
mse_list = []
rmse_list = []

# Run the Enhanced Kalman Filter for 3D
for n in range(num_steps):
    ekf.predict(u=us[n])
    ekf.update(observations[n])
    estimated_means.append(ekf.x)
    error = np.linalg.norm(ekf.x[:3] - true_trajectory[n+1, :3])
    tracking_error.append(error)
    mse_list.append(error**2)
    rmse_list.append(np.sqrt(error**2))

# Convert to arrays for easy plotting
estimated_means = np.array(estimated_means)
tracking_error = np.array(tracking_error)

# Performance Metrics: Mean Squared Error (MSE) and Root Mean Square Error (RMSE)
mse_total = np.mean(mse_list)
rmse_total = np.sqrt(mse_total)
print("Mean Squared Error (MSE):", mse_total)
print("Root Mean Square Error (RMSE):", rmse_total)

# Plot true vs estimated 3D trajectory
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot(true_trajectory[:, 0], true_trajectory[:, 1], true_trajectory[:, 2], label="True Position", color='green')
ax.plot(observations[:, 0], observations[:, 1], observations[:, 2], 'ro', markersize=4, label="Observations")
ax.plot(estimated_means[:, 0], estimated_means[:, 1], estimated_means[:, 2], linestyle='--', color='blue',
label="Estimated Position")
ax.set_xlabel("X Position")
ax.set_ylabel("Y Position")
ax.set_zlabel("Z Position")
ax.legend()
ax.set_title("3D Position Estimation")
plt.show()

# Plot tracking error over time
plt.figure(figsize=(10, 6))
plt.plot(tracking_error, label="Tracking Error (3D Distance)")
plt.xlabel("Time Step")
plt.ylabel("Error (Distance)")
plt.title("3D Tracking Error Over Time")
plt.legend()
plt.grid()
plt.show()

# Velocity Estimation Comparison
true_velocity = true_trajectory[1:, 3:]
estimated_velocity = estimated_means[:, 3:]
plt.figure(figsize=(10, 6))
plt.plot(true_velocity[:, 0], label="True Velocity X", color='green')
plt.plot(estimated_velocity[:, 0], linestyle='--', label="Estimated Velocity X", color='blue')
plt.plot(true_velocity[:, 1], label="True Velocity Y", color='orange')
plt.plot(estimated_velocity[:, 1], linestyle='--', label="Estimated Velocity Y", color='purple')
plt.plot(true_velocity[:, 2], label="True Velocity Z", color='red')
plt.plot(estimated_velocity[:, 2], linestyle='--', label="Estimated Velocity Z", color='brown')
plt.xlabel("Time Step")
plt.ylabel("Velocity")
plt.legend()
plt.title("True vs. Estimated Velocity")
plt.grid()
plt.show()
```