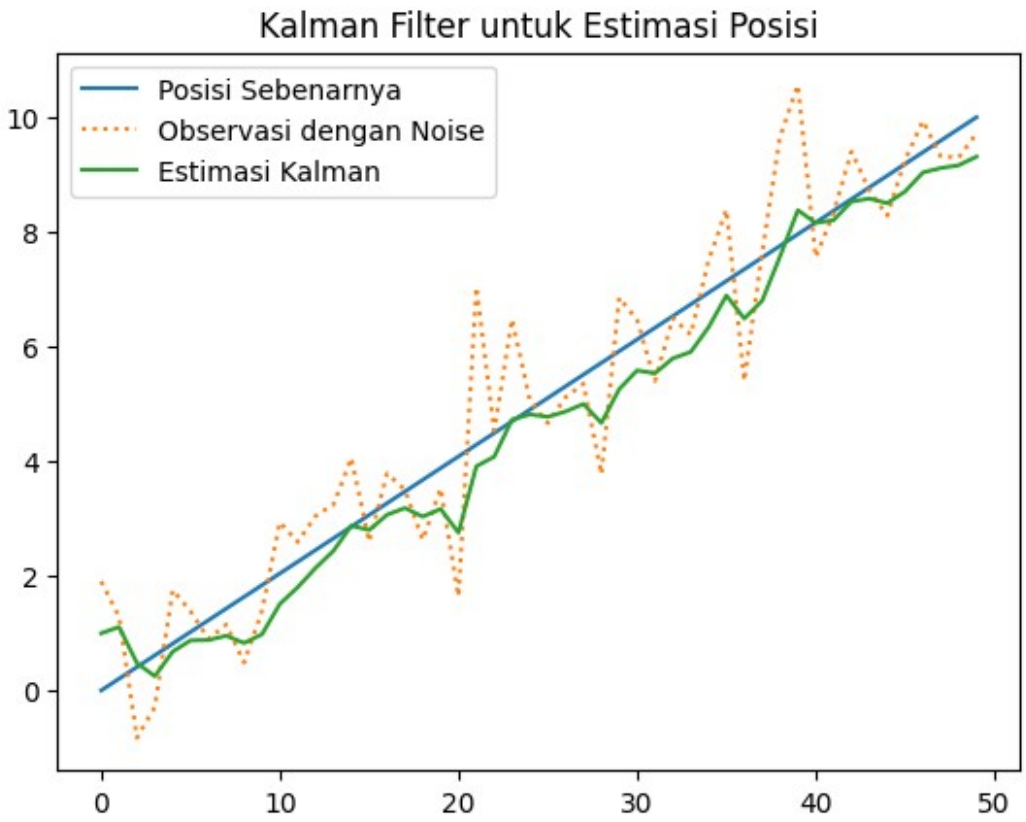
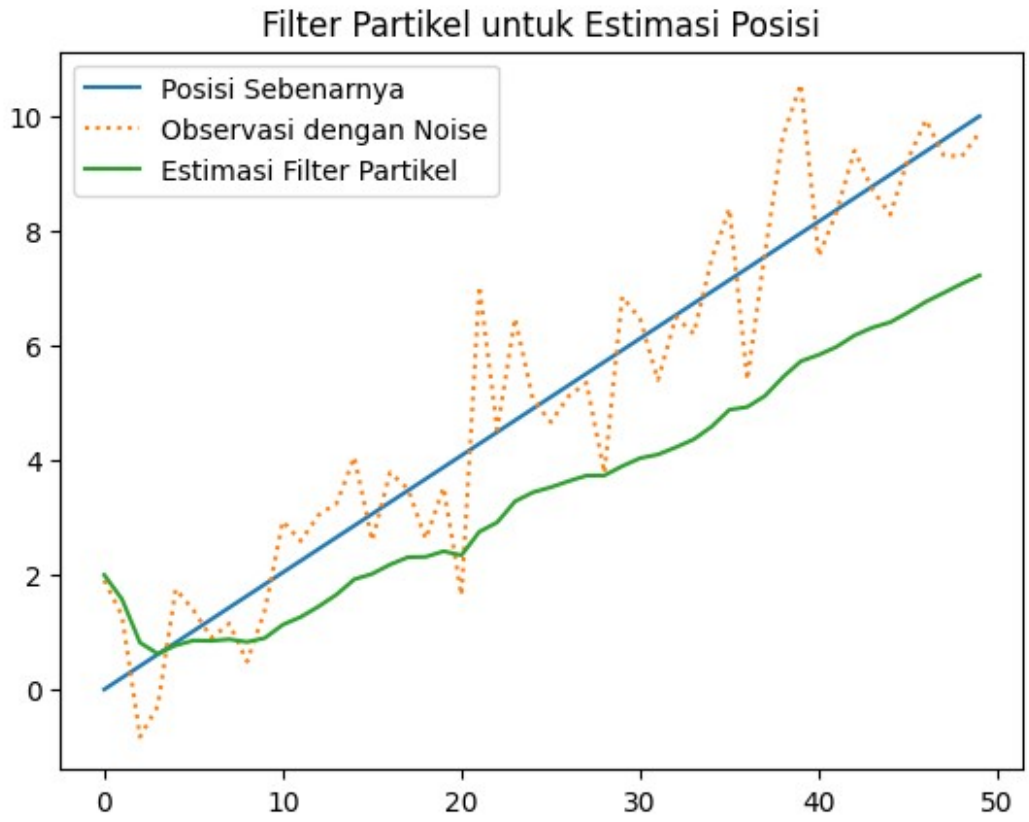
```python
import numpy as np
import matplotlib.pyplot as plt

num_steps = 50
actual_position = np.linspace(0, 10, num_steps)  # Posisi sebenarnya
(untuk simulasi)
observations = actual_position + np.random.normal(0, 1, num_steps)  #
Observasi dengan noise

# Variabel Kalman Filter
estimated_position = []
estimate = 0
estimate_uncertainty = 1
measurement_uncertainty = 1
process_uncertainty = 0.1

# Loop Kalman Filter
for i in range(num_steps):
    # Langkah prediksi
    estimate_uncertainty += process_uncertainty

    # Langkah update
    kalman_gain = estimate_uncertainty / (estimate_uncertainty +
measurement_uncertainty)
    estimate = estimate + kalman_gain * (observations[i] - estimate)
    estimate_uncertainty = (1 - kalman_gain) * estimate_uncertainty

    estimated_position.append(estimate)

# Plot hasil
plt.figure()
plt.plot(actual_position, label='Posisi Sebenarnya')
plt.plot(observations, label='Observasi dengan Noise',
linestyle='dotted')
plt.plot(estimated_position, label='Estimasi Kalman')
plt.legend()
plt.title('Kalman Filter untuk Estimasi Posisi')
plt.show()
```



Kalman Filter untuk memperkirakan posisi suatu objek meskipun ada gangguan atau noise dalam pengukurannya. Pertama, kode ini memprediksi posisi objek, lalu mengupdate prediksi berdasarkan pengukuran yang baru. Dengan cara ini, hasil estimasi menjadi lebih akurat, meskipun data yang diterima tidak sempurna. Hasil akhirnya adalah plot yang menunjukkan posisi objek yang lebih jelas dibandingkan dengan pengukuran yang berisik.

```
def particle_filter(observations, num_particles=1000):
    particles = np.random.uniform(0, 10, num_particles)  #
Inisialisasi partikel
    weights = np.ones(num_particles) / num_particles  # Bobot awal
sama rata

    estimates = []
    for obs in observations:
        # Langkah prediksi (menambahkan noise pada partikel)
        particles += np.random.normal(0, 0.1, num_particles)

        # Update bobot berdasarkan observasi
        weights *= np.exp(-0.5 * ((particles - obs) ** 2))
        weights /= np.sum(weights)

        # Resampling partikel berdasarkan bobot
        indices = np.random.choice(range(num_particles),
num_particles, p=weights)
```

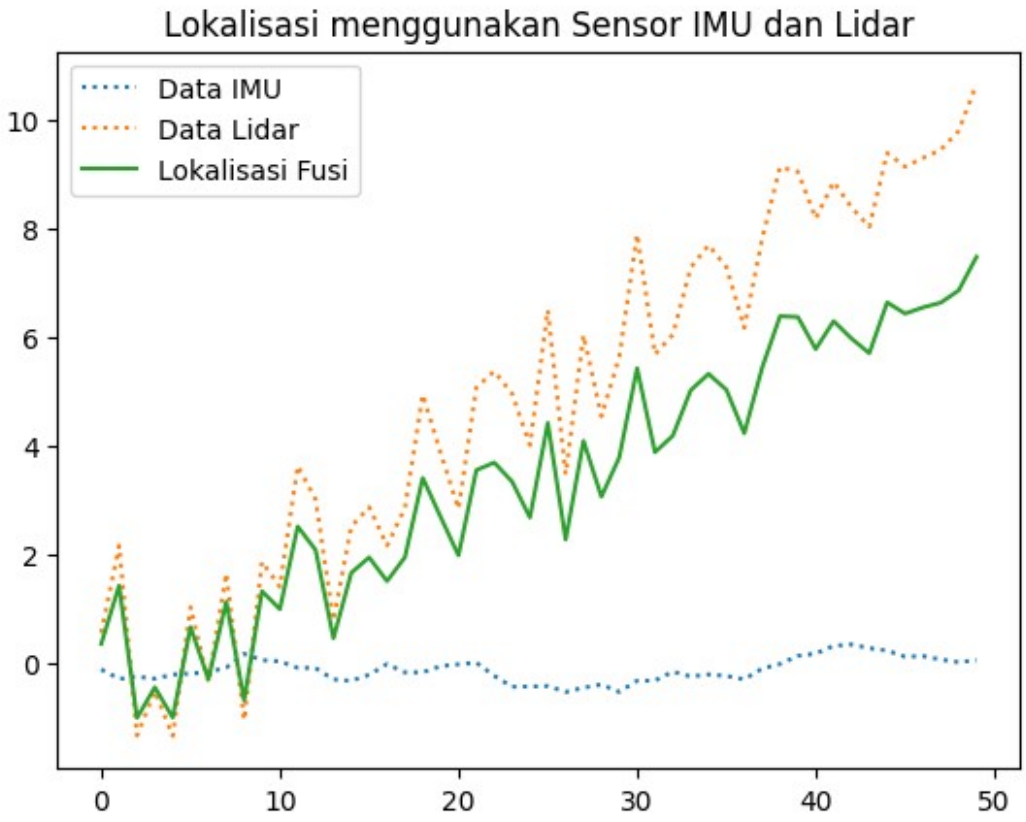
```
        particles = particles[indices]
        weights = np.ones(num_particles) / num_particles

        # Estimasi posisi sebagai rata-rata partikel
        estimates.append(np.mean(particles))

    return estimates

# Jalankan filter partikel dan plot
particle_estimates = particle_filter(observations)
plt.figure()
plt.plot(actual_position, label='Posisi Sebenarnya')
plt.plot(observations, label='Observasi dengan Noise',
linestyle='dotted')
plt.plot(particle_estimates, label='Estimasi Filter Partikel')
plt.legend()
plt.title('Filter Partikel untuk Estimasi Posisi')
plt.show()
```



metode filter partikel untuk memperkirakan posisi objek meskipun ada gangguan dalam pengukurannya. Setiap partikel diwakili sebagai posisi yang mungkin, dan partikel-partikel ini diperbarui setiap kali ada pengamatan baru. Proses ini memungkinkan kita untuk mendapatkan estimasi posisi yang lebih akurat meskipun data yang diterima tidak sempurna. Plot akhirnya

menunjukkan perbandingan antara posisi objek yang sebenarnya, pengamatan dengan noise, dan estimasi posisi berdasarkan filter partikel.

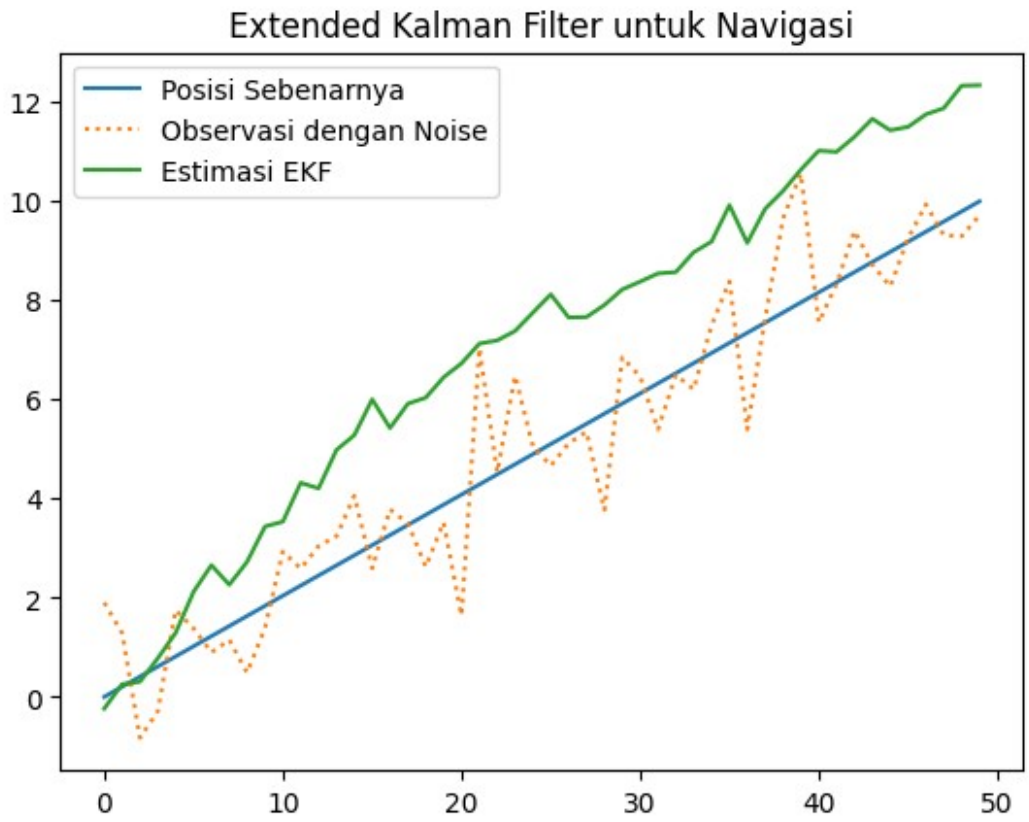
```
def imu_lidar_localization(num_steps):
    imu_data = np.cumsum(np.random.normal(0, 0.1, num_steps))  # Data
IMU simulasi
    lidar_data = np.linspace(0, 10, num_steps) + np.random.normal(0,
1, num_steps)  # Data Lidar simulasi

    # Fusi data IMU dan Lidar (rata-rata berbobot)
    localization = 0.7 * lidar_data + 0.3 * imu_data

    return imu_data, lidar_data, localization

imu, lidar, fused = imu_lidar_localization(num_steps)
plt.figure()
plt.plot(imu, label='Data IMU', linestyle='dotted')
plt.plot(lidar, label='Data Lidar', linestyle='dotted')
plt.plot(fused, label='Lokalisasi Fusi')
plt.legend()
plt.title('Lokalisasi menggunakan Sensor IMU dan Lidar')
plt.show()
```



Menggabungkan data dari dua sensor, IMU dan Lidar, untuk mendapatkan estimasi posisi yang lebih akurat. Data IMU menggambarkan pergerakan objek, sedangkan data Lidar memberikan

informasi posisi yang lebih jelas meskipun terpengaruh noise. Dengan menggabungkan kedua data ini menggunakan bobot tertentu, kita mendapatkan estimasi posisi yang lebih baik. Plot yang dihasilkan menunjukkan perbandingan antara data IMU, Lidar, dan hasil gabungan dari keduanya.

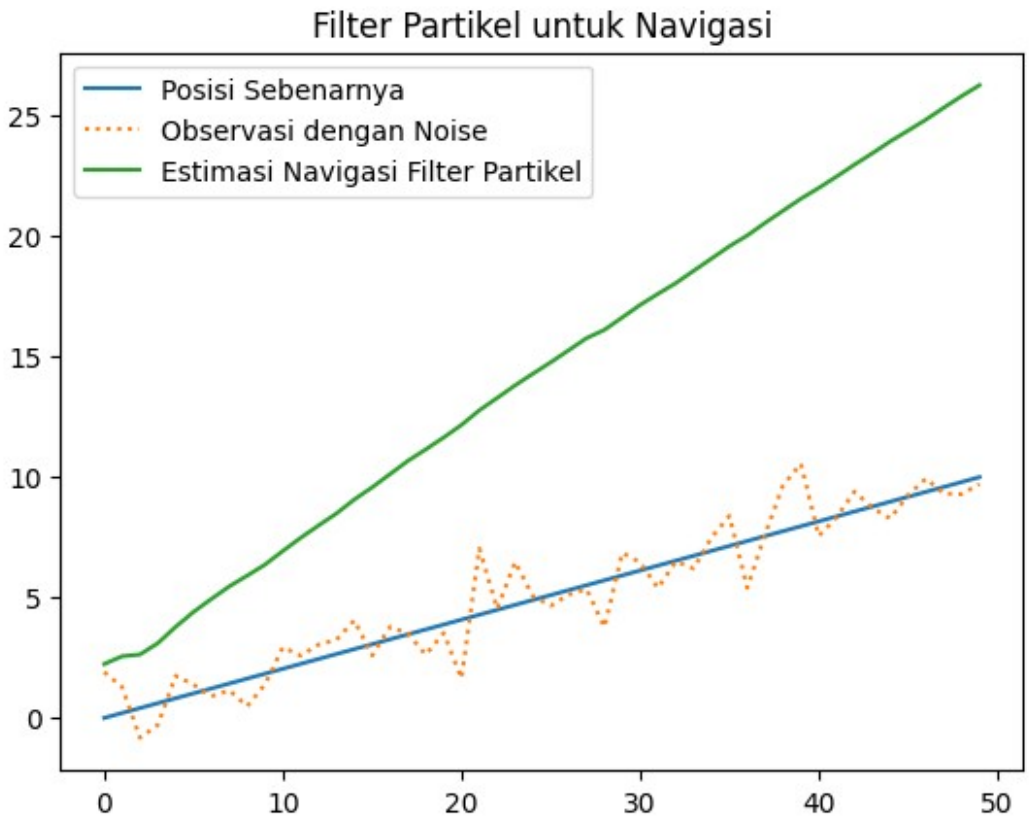
```
def ekf_navigation(num_steps):
    # Vektor keadaan: [posisi, kecepatan]
    state = np.array([0, 1])  # Posisi dan kecepatan awal
    state_uncertainty = np.eye(2)

    # Observasi simulasi (posisi)
    observations = np.linspace(0, 10, num_steps) + np.random.normal(0,
1, num_steps)

    states = []
    for obs in observations:
        # Langkah prediksi
        state = np.array([state[0] + state[1], state[1]])  # Update
posisi dan kecepatan
        state_uncertainty += np.eye(2) * 0.1  # Noise proses

        # Langkah update
        kalman_gain = state_uncertainty @ np.array([[1], [0]]) /
(state_uncertainty[0, 0] + 1)
        state = state + kalman_gain.flatten() * (obs - state[0])
        state_uncertainty = (np.eye(2) - kalman_gain @ np.array([[1,
0]])) @ state_uncertainty

        states.append(state[0])

    return states

# Jalankan EKF dan plot
ekf_states = ekf_navigation(num_steps)
plt.figure()
plt.plot(actual_position, label='Posisi Sebenarnya')
plt.plot(observations, label='Observasi dengan Noise',
linestyle='dotted')
plt.plot(ekf_states, label='Estimasi EKF')
plt.legend()
plt.title('Extended Kalman Filter untuk Navigasi')
plt.show()
```



Extended Kalman Filter (EKF) untuk memperkirakan posisi dan kecepatan objek meskipun pengukurannya terpengaruh oleh noise. Prosesnya dimulai dengan memprediksi posisi dan kecepatan berdasarkan model pergerakan, kemudian memperbarui estimasi posisi dengan membandingkan prediksi dengan pengukuran yang diterima. Hasil akhir adalah estimasi posisi yang lebih tepat, meskipun data pengukuran tidak sempurna. Plot yang dihasilkan memperlihatkan perbandingan antara posisi sebenarnya, observasi yang berisik, dan estimasi posisi menggunakan EKF.

```
def particle_filter_navigation(observations, num_particles=1000):
    particles = np.random.uniform(0, 10, num_particles)  # 
Inisialisasi partikel
    velocities = np.random.normal(1, 0.1, num_particles)  # Kecepatan 
awal
    weights = np.ones(num_particles) / num_particles  # Bobot awal 
sama rata

    estimates = []
    for obs in observations:
        # Langkah prediksi
        particles += velocities  # Update partikel berdasarkan 
kecepatan
        particles += np.random.normal(0, 0.1, num_particles)  # Tambah 
noise
```

```
        # Update bobot berdasarkan observasi
        weights *= np.exp(-0.5 * ((particles - obs) ** 2))
        weights /= np.sum(weights)

        # Resampling partikel berdasarkan bobot
        indices = np.random.choice(range(num_particles),
num_particles, p=weights)
        particles = particles[indices]
        velocities = velocities[indices]
        weights = np.ones(num_particles) / num_particles

        # Estimasi posisi sebagai rata-rata partikel
        estimates.append(np.mean(particles))

    return estimates

# Jalankan filter partikel untuk navigasi dan plot
particle_navigation_estimates =
particle_filter_navigation(observations)
plt.figure()
plt.plot(actual_position, label='Posisi Sebenarnya')
plt.plot(observations, label='Observasi dengan Noise',
linestyle='dotted')
plt.plot(particle_navigation_estimates, label='Estimasi Navigasi
Filter Partikel')
plt.legend()
plt.title('Filter Partikel untuk Navigasi')
plt.show()
```



Filter partikel untuk memperkirakan posisi objek dengan memanfaatkan serangkaian partikel yang mewakili posisi yang mungkin. Setiap partikel bergerak berdasarkan kecepatan yang memiliki noise, dan posisi diperbarui setiap langkah. Kemudian, filter ini menyesuaikan posisi partikel berdasarkan seberapa baik posisi tersebut cocok dengan pengukuran yang diterima. Setelah beberapa iterasi, kita mendapatkan estimasi posisi yang lebih akurat meskipun data yang diterima tidak sempurna. Plot yang dihasilkan memperlihatkan perbandingan antara posisi objek yang sebenarnya, pengamatan dengan noise, dan estimasi posisi yang lebih halus.