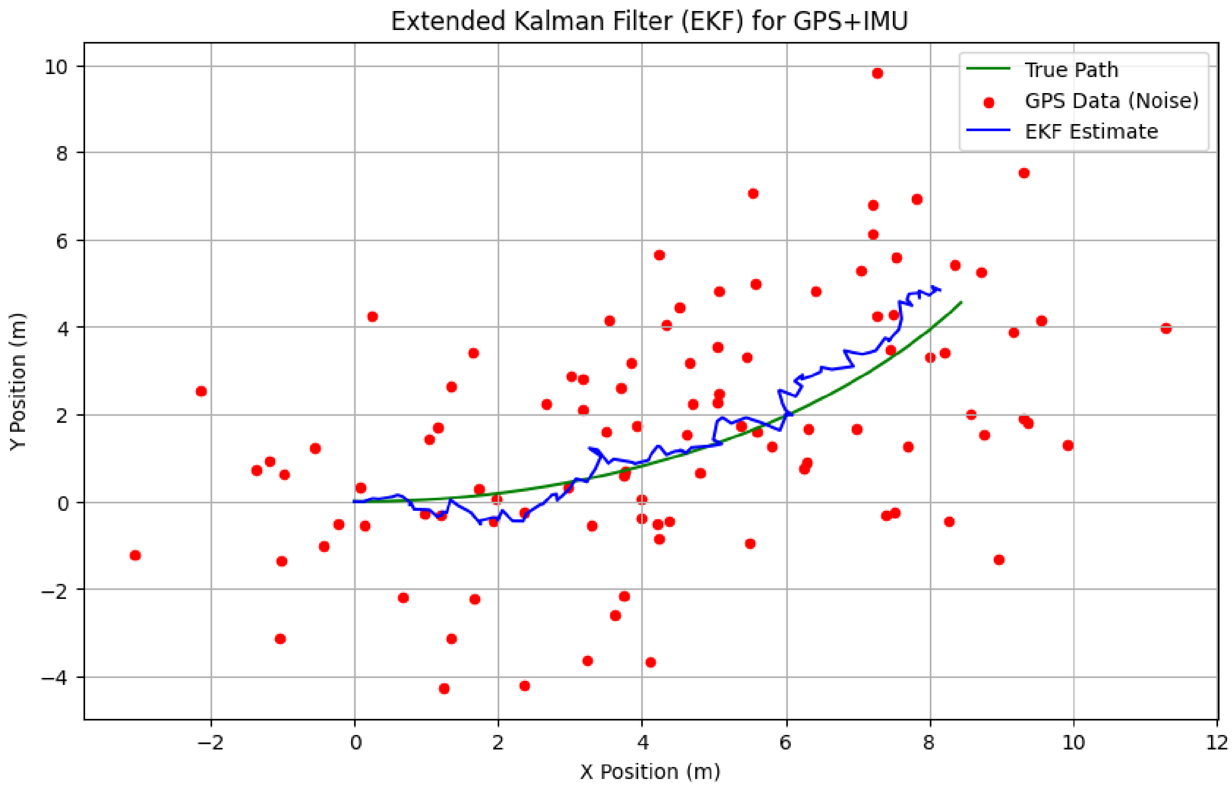
```python
import numpy as np
import matplotlib.pyplot as plt

# Parameter EKF
dt = 0.1  # Waktu langkah
state = np.array([0, 0, 0])  # [x, y, theta] posisi awal
covariance = np.eye(3) * 0.1  # Covariance matrix awal
process_noise = np.diag([0.01, 0.01, 0.001])  # Proses noise (Q)
measurement_noise = np.diag([5, 5])  # Noise GPS (R)

# Model Gerak
def motion_model(state, control, dt):
    x, y, theta = state
    v, omega = control
    x_new = x + v * np.cos(theta) * dt
    y_new = y + v * np.sin(theta) * dt
    theta_new = theta + omega * dt
    return np.array([x_new, y_new, theta_new])

# Jacobian untuk model gerak
def jacobian_motion(state, control, dt):
    _, _, theta = state
    v, _ = control
    F = np.array([
        [1, 0, -v * np.sin(theta) * dt],
        [0, 1,  v * np.cos(theta) * dt],
        [0, 0,  1]
    ])
    return F

# Model Pengamatan (GPS)
def measurement_model(state):
    return state[:2]

# Jacobian untuk pengamatan
def jacobian_measurement():
    return np.array([
        [1, 0, 0],
        [0, 1, 0]
    ])

# Data GPS dan IMU Simulasi
np.random.seed(42)
true_positions = [np.array([0, 0, 0])]
gps_data = []
imu_controls = []

for t in range(100):
    # Kontrol IMU (kecepatan linear dan sudut)
    v = 1.0
```

```
    omega = 0.1
    imu_controls.append([v, omega])

    # Posisi sebenarnya
    true_position = motion_model(true_positions[-1], [v, omega], dt)
    true_positions.append(true_position)

    # Data GPS dengan noise
    gps = measurement_model(true_position) +
np.random.multivariate_normal([0, 0], measurement_noise)
    gps_data.append(gps)

# EKF Implementasi
estimated_positions = [state]
for i in range(len(gps_data)):
    # Predict step
    control = imu_controls[i]
    state_pred = motion_model(estimated_positions[-1], control, dt)
    F = jacobian_motion(estimated_positions[-1], control, dt)
    covariance_pred = F @ covariance @ F.T + process_noise

    # Update step
    z = gps_data[i]
    H = jacobian_measurement()
    y = z - measurement_model(state_pred)
    S = H @ covariance_pred @ H.T + measurement_noise
    K = covariance_pred @ H.T @ np.linalg.inv(S)

    state_est = state_pred + K @ y
    covariance = (np.eye(3) - K @ H) @ covariance_pred
    estimated_positions.append(state_est)

# Plot Hasil
true_positions = np.array(true_positions)
gps_data = np.array(gps_data)
estimated_positions = np.array(estimated_positions)

plt.figure(figsize=(10, 6))
plt.plot(true_positions[:, 0], true_positions[:, 1], 'g-', label='True
Path')
plt.scatter(gps_data[:, 0], gps_data[:, 1], c='r', s=20, label='GPS
Data (Noise)')
plt.plot(estimated_positions[:, 0], estimated_positions[:, 1], 'b-',
label='EKF Estimate')
plt.legend()
plt.title("Extended Kalman Filter (EKF) for GPS+IMU")
plt.xlabel("X Position (m)")
plt.ylabel("Y Position (m)")
plt.grid()
plt.show()
```



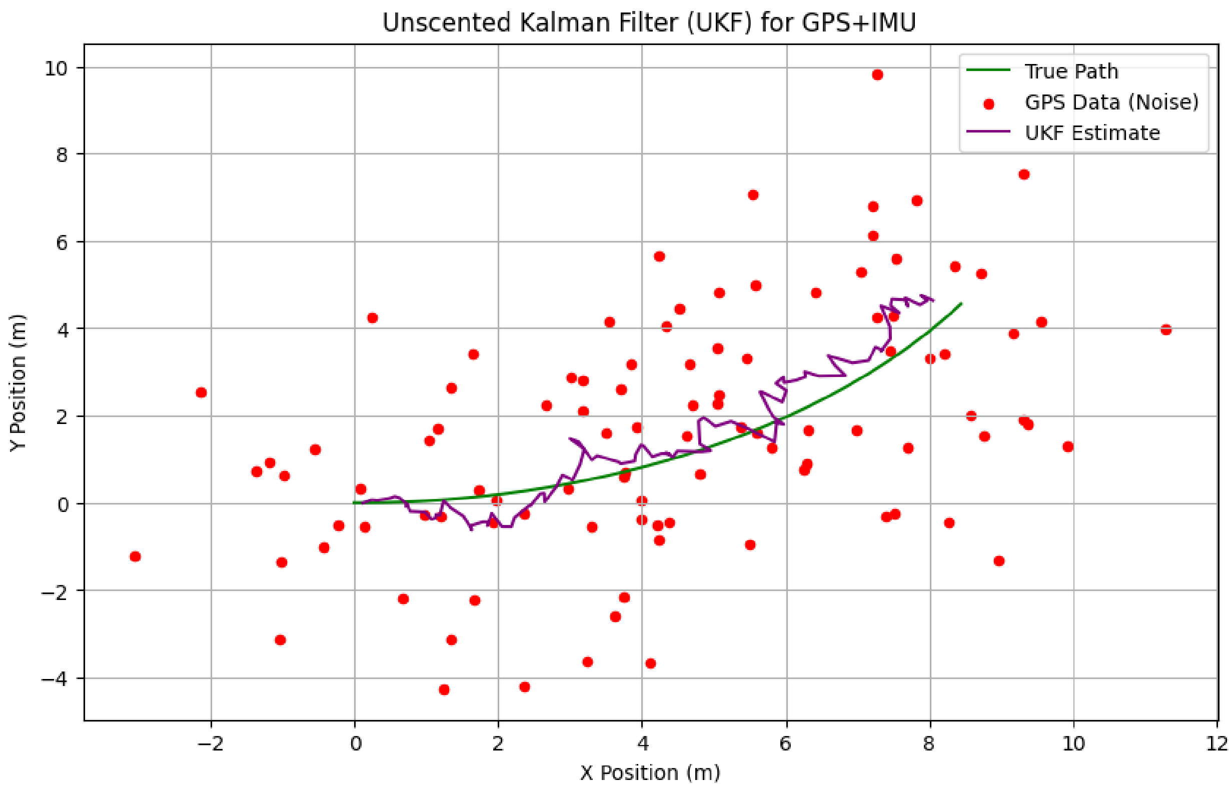
Extended Kalman Filter (EKF) untuk memperkirakan jalur sebenarnya dari suatu objek berdasarkan data GPS yang bising dan kontrol IMU (kecepatan dan rotasi). Simulasi mencakup generasi posisi sebenarnya, data GPS dengan noise, dan estimasi posisi menggunakan EKF. EKF bekerja dalam dua langkah utama: (1) Prediksi, di mana posisi berikutnya diproyeksikan berdasarkan kontrol IMU dan model gerak; (2) Pembaruan, di mana prediksi diperbaiki menggunakan data GPS bising. Grafik menunjukkan bahwa jalur estimasi EKF (biru) mendekati jalur sebenarnya (hijau), meskipun data GPS sangat bising (titik merah), menandakan bahwa EKF berhasil mengurangi pengaruh noise pada estimasi posisi.

```
!pip install filterpy

Collecting filterpy
  Downloading filterpy-1.4.5.zip (177 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.0/178.0 kB 2.6 MB/s eta
0:00:00
etadata (setup.py) ... ent already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from filterpy) (1.26.4)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from filterpy) (1.13.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from filterpy) (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(24.2)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->filterpy)
(2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->filterpy) (1.17.0)
Building wheels for collected packages: filterpy
  Building wheel for filterpy (setup.py) ... e=filterpy-1.4.5-py3-
none-any.whl size=110458
sha256=f0e27056371c96690d240c513afcb572811af7bc63985e070a376a17e1a3387
9
  Stored in directory:
/root/.cache/pip/wheels/0f/0c/ea/218f266af4ad626897562199fbbcba521b849
7303200186102
Successfully built filterpy
Installing collected packages: filterpy
Successfully installed filterpy-1.4.5

# Import modul yang diperlukan
import numpy as np
import matplotlib.pyplot as plt
from filterpy.kalman import UnscentedKalmanFilter as UKF
from filterpy.kalman import MerweScaledSigmaPoints

# UKF Setup
def fx(state, dt, control):
    x, y, theta = state
    v, omega = control
    x_new = x + v * np.cos(theta) * dt
    y_new = y + v * np.sin(theta) * dt
    theta_new = theta + omega * dt
    return np.array([x_new, y_new, theta_new])
```

```
def hx(state):
    return state[:2]  # Observasi (x, y)

# Sigma points untuk UKF
points = MerweScaledSigmaPoints(n=3, alpha=0.1, beta=2., kappa=1)
ukf = UKF(dim_x=3, dim_z=2, fx=fx, hx=hx, dt=0.1, points=points)
ukf.x = np.array([0., 0., 0.])  # State awal
ukf.P *= 0.1
ukf.Q = np.diag([0.01, 0.01, 0.01])  # Noise proses
ukf.R = np.diag([5, 5])  # Noise pengamatan GPS

# Simulasi Data
np.random.seed(42)
dt = 0.1
gps_data = []
controls = []
true_states = [np.array([0, 0, 0])]

for t in range(100):
    # Kontrol gerakan (kecepatan dan rotasi)
    control = np.array([1.0, 0.1])
    controls.append(control)

    # Gerak robot sebenarnya
    true_state = fx(true_states[-1], dt, control)
    true_states.append(true_state)

    # Pengamatan GPS dengan noise
    gps = true_state[:2] + np.random.multivariate_normal([0, 0],
np.diag([5, 5]))
    gps_data.append(gps)

# Jalankan UKF
ukf_positions = []
for i, control in enumerate(controls):
    ukf.predict(control=control)
    ukf.update(gps_data[i])
    ukf_positions.append(ukf.x)

# Plot hasil
true_states = np.array(true_states)
gps_data = np.array(gps_data)
ukf_positions = np.array(ukf_positions)

plt.figure(figsize=(10, 6))
plt.plot(true_states[:, 0], true_states[:, 1], 'g-', label='True
Path')  # Jalur sebenarnya
plt.scatter(gps_data[:, 0], gps_data[:, 1], c='r', s=20, label='GPS
Data (Noise)')  # Data GPS
plt.plot(ukf_positions[:, 0], ukf_positions[:, 1], '-',
```

```
color='purple', label='UKF Estimate')  # Estimasi UKF
plt.legend()
plt.title("Unscented Kalman Filter (UKF) for GPS+IMU")
plt.xlabel("X Position (m)")
plt.ylabel("Y Position (m)")
plt.grid()
plt.show()
```

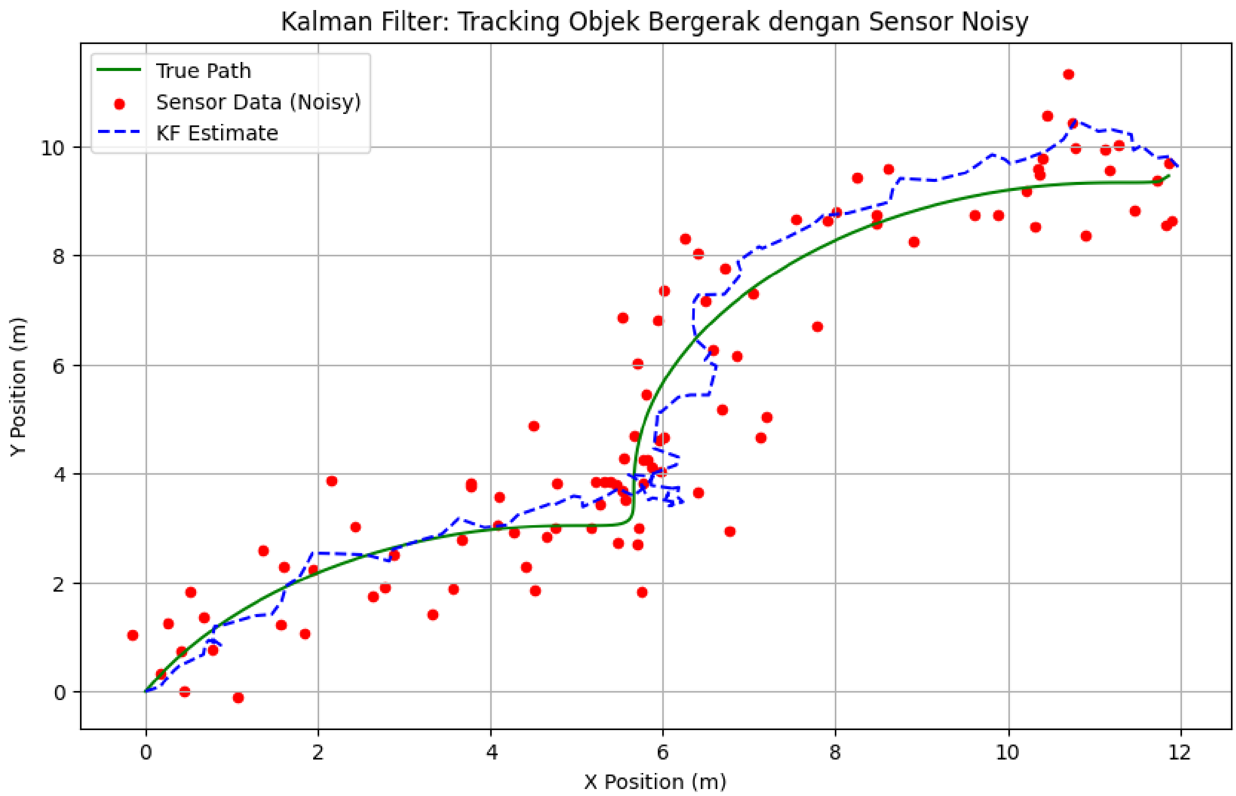


Unscented Kalman Filter (UKF) untuk memperkirakan jalur sebenarnya dari suatu objek dengan memadukan data kontrol IMU (kecepatan dan rotasi) serta pengamatan GPS yang bising. UKF bekerja dengan menggunakan sigma points untuk menangkap distribusi non-linear dari dinamika gerak. Simulasi dimulai dengan membuat data posisi sebenarnya, data GPS yang bising, dan kontrol IMU, kemudian UKF memperkirakan posisi berdasarkan langkah prediksi (menggunakan kontrol gerak) dan pembaruan (mengoreksi prediksi dengan data GPS). Hasilnya menunjukkan bahwa estimasi UKF (garis ungu) mendekati jalur sebenarnya (garis hijau), meskipun data GPS (titik merah) mengandung noise, membuktikan efektivitas UKF dalam menangani dinamika non-linear dan noise tinggi.

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

# Fungsi Model Gerak (Linear)
def motion_model(state, dt):
    # State: [posisi_x, kecepatan_x, posisi_y, kecepatan_y]
    F = np.array([
```

```
        [1, dt, 0,  0],
        [0,  1, 0,  0],
        [0,  0, 1, dt],
        [0,  0, 0,  1]
    ])
    return F @ state

# Model Pengamatan (Hanya Posisi)
def measurement_model(state):
    return np.array([state[0], state[2]])  # [posisi_x, posisi_y]

# Jacobian untuk Pengamatan
def jacobian_measurement():
    return np.array([
        [1, 0, 0, 0],
        [0, 0, 1, 0]
    ])

# Inisialisasi Variabel
dt = 0.1  # Timestep
state = np.array([0, 1, 0, 1])  # [pos_x, vel_x, pos_y, vel_y]
covariance = np.eye(4) * 0.1  # Covariance Matrix
process_noise = np.eye(4) * 0.01  # Proses noise (Q)
measurement_noise = np.eye(2) * 0.5  # Noise sensor posisi (R)

# Simulasi Data
np.random.seed(42)
true_states = [state]
measurements = []

for t in range(100):
    # Gerak objek sebenarnya (sinusoidal)
    state[0] += np.sin(0.1 * t) * 0.1  # Posisi X
    state[2] += np.cos(0.1 * t) * 0.1  # Posisi Y
    state = motion_model(state, dt)
    true_states.append(state)

    # Sensor membaca posisi dengan noise
    measurement = measurement_model(state) +
np.random.multivariate_normal([0, 0], measurement_noise)
    measurements.append(measurement)

# Jalankan Kalman Filter
estimated_states = [np.array([0, 1, 0, 1])]
for i in range(len(measurements)):
    # Predict step
    F = np.array([
        [1, dt, 0,  0],
        [0,  1, 0,  0],
        [0,  0, 1, dt],
```

```
        [0,  0, 0,  1]
    ])
    state_pred = F @ estimated_states[-1]
    covariance_pred = F @ covariance @ F.T + process_noise

    # Update step
    z = measurements[i]
    H = jacobian_measurement()
    y = z - H @ state_pred
    S = H @ covariance_pred @ H.T + measurement_noise
    K = covariance_pred @ H.T @ np.linalg.inv(S)

    state_est = state_pred + K @ y
    covariance = (np.eye(4) - K @ H) @ covariance_pred
    estimated_states.append(state_est)

# Plot Hasil
true_states = np.array(true_states)
measurements = np.array(measurements)
estimated_states = np.array(estimated_states)

plt.figure(figsize=(10, 6))
plt.plot(true_states[:, 0], true_states[:, 2], 'g-', label='True
Path')  # Jalur sebenarnya
plt.scatter(measurements[:, 0], measurements[:, 1], c='r', s=20,
label='Sensor Data (Noisy)')  # Data Sensor
plt.plot(estimated_states[:, 0], estimated_states[:, 2], 'b--',
label='KF Estimate')  # Estimasi KF
plt.legend()
plt.title("Kalman Filter: Tracking Objek Bergerak dengan Sensor
Noisy")
plt.xlabel("X Position (m)")
plt.ylabel("Y Position (m)")
plt.grid()
plt.show()
```

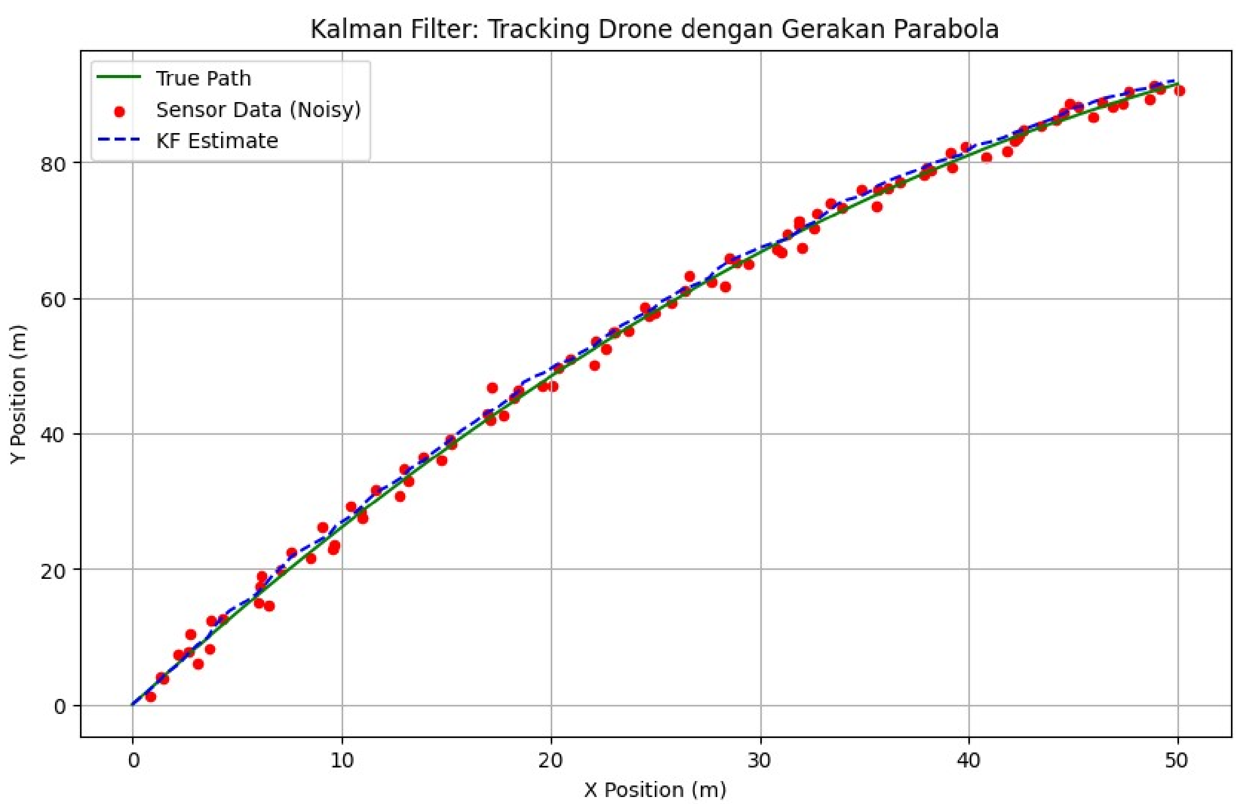


Kalman Filter (KF) untuk melacak posisi sebuah objek yang bergerak secara sinusoidal dengan data sensor posisi yang bising. Simulasi dimulai dengan membuat jalur sebenarnya dari objek (berbasis model gerak linear) dan menghasilkan pengamatan posisi yang dicampur dengan noise sensor. Kalman Filter bekerja melalui dua langkah utama: (1) Prediksi, di mana posisi dan kecepatan objek dihitung berdasarkan model gerak; (2) Pembaruan, di mana prediksi diperbaiki menggunakan data sensor. Hasilnya menunjukkan bahwa estimasi Kalman Filter (garis biru putus-putus) berhasil mendekati jalur sebenarnya (garis hijau), meskipun data sensor (titik merah) mengandung noise signifikan. Ini menegaskan kehandalan KF dalam menyaring noise dari data pengamatan untuk menghasilkan estimasi posisi yang lebih akurat.

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

# Fungsi Model Gerak
def motion_model(state, dt):
    # State: [posisi_x, kecepatan_x, posisi_y, kecepatan_y]
    F = np.array([
        [1, dt, 0,  0],
        [0,  1, 0,  0],
        [0,  0, 1, dt],
        [0,  0, 0,  1]
    ])
    return F @ state

# Model Pengamatan (Hanya Posisi)
```

```
def measurement_model(state):
    return np.array([state[0], state[2]])  # [posisi_x, posisi_y]

# Jacobian untuk Pengamatan
def jacobian_measurement():
    return np.array([
        [1, 0, 0, 0],
        [0, 0, 1, 0]
    ])

# Inisialisasi Variabel
dt = 0.1  # Timestep
state = np.array([0, 5, 0, 15])  # [pos_x, vel_x, pos_y, vel_y]
covariance = np.eye(4) * 0.1  # Covariance Matrix
process_noise = np.eye(4) * 0.01  # Proses noise (Q)
measurement_noise = np.eye(2) * 0.5  # Noise sensor posisi (R)

# Simulasi Data
np.random.seed(42)
true_states = [state]
measurements = []

for t in range(100):
    # Gerakan parabola: Y dipengaruhi gravitasi
    state[3] -= 0.98 * dt  # Gravitasi (penurunan kecepatan Y)
    state = motion_model(state, dt)
    true_states.append(state)

    # Sensor membaca posisi dengan noise
    measurement = measurement_model(state) +
np.random.multivariate_normal([0, 0], measurement_noise)
    measurements.append(measurement)

# Jalankan Kalman Filter
estimated_states = [np.array([0, 5, 0, 15])]
for i in range(len(measurements)):
    # Predict step
    F = np.array([
        [1, dt, 0,  0],
        [0,  1, 0,  0],
        [0,  0, 1, dt],
        [0,  0, 0,  1]
    ])
    state_pred = F @ estimated_states[-1]
    covariance_pred = F @ covariance @ F.T + process_noise

    # Update step
    z = measurements[i]
    H = jacobian_measurement()
    y = z - H @ state_pred
```

```
    S = H @ covariance_pred @ H.T + measurement_noise
    K = covariance_pred @ H.T @ np.linalg.inv(S)

    state_est = state_pred + K @ y
    covariance = (np.eye(4) - K @ H) @ covariance_pred
    estimated_states.append(state_est)

# Plot Hasil
true_states = np.array(true_states)
measurements = np.array(measurements)
estimated_states = np.array(estimated_states)

plt.figure(figsize=(10, 6))
plt.plot(true_states[:, 0], true_states[:, 2], 'g-', label='True 
Path')  # Jalur sebenarnya
plt.scatter(measurements[:, 0], measurements[:, 1], c='r', s=20, 
label='Sensor Data (Noisy)')  # Data Sensor
plt.plot(estimated_states[:, 0], estimated_states[:, 2], 'b--', 
label='KF Estimate')  # Estimasi KF
plt.legend()
plt.title("Kalman Filter: Tracking Drone dengan Gerakan Parabola")
plt.xlabel("X Position (m)")
plt.ylabel("Y Position (m)")
plt.grid()
plt.show()
```



Kalman Filter (KF) untuk melacak jalur gerak parabola, yang mencerminkan pergerakan drone atau objek yang dipengaruhi gravitasi. Gerakan ini dimodelkan dengan perubahan kecepatan vertikal (sumbu Y) karena gaya gravitasi, sedangkan posisi horizontal (sumbu X) tetap konstan. Simulasi menghasilkan jalur sebenarnya dari objek, data pengamatan posisi dari sensor yang dicampur dengan noise, dan hasil estimasi Kalman Filter. KF bekerja dengan dua langkah utama: (1) Prediksi, memperkirakan posisi objek berdasarkan model gerak; (2) Pembaruan, mengoreksi prediksi dengan data sensor. Hasilnya menunjukkan bahwa estimasi KF (garis biru putus-putus) secara konsisten mendekati jalur sebenarnya (garis hijau), meskipun pengamatan sensor (titik merah) mengandung noise. Ini menegaskan kehandalan KF dalam melacak gerakan parabola secara akurat meskipun dengan pengamatan yang tidak sempurna.