Tentu! Berikut ini adalah analisis dari semua kode dalam gaya bahasa mahasiswa non formal:

Kode 1: Extended Kalman Filter (EKF) dengan Data GPS dan IMU

Ringkasan: Kode ini menggunakan Extended Kalman Filter (EKF) untuk menggabungkan data GPS dan IMU guna memperkirakan posisi dan orientasi suatu objek.

Poin Penting:

- **State Vector:** state terdiri dari posisi (x, y) dan orientasi (theta).
- **Motion Model:** motion_model mengupdate posisi dan orientasi berdasarkan input kontrol (kecepatan v dan kecepatan sudut omega).
- **Measurement Model:** measurement_model memberikan perkiraan pengukuran GPS, yaitu koordinat x dan y.
- **Jacobian Matrices:** jacobian_motion dan jacobian_measurement menghitung matriks Jacobian untuk model gerak dan pengukuran.
- **Langkah EKF:**
    - **Prediction:** Memprediksi state baru dan matriks covariance.
    - **Update:** Mengupdate state dan covariance dengan pengukuran GPS baru menggunakan Kalman Gain.

Visualisasi: Hasil posisi, bersama dengan posisi sebenarnya dan data GPS yang penuh noise, dipetakan untuk menunjukkan performa EKF.

Kode 2: Unscented Kalman Filter (UKF) untuk Data GPS dan IMU

Ringkasan: Kode ini menggunakan Unscented Kalman Filter (UKF) untuk menggabungkan data GPS dan IMU guna memperkirakan posisi dan orientasi suatu objek.

Poin Penting:

- **State Vector:** Mirip dengan EKF, state vector ukf.x terdiri dari posisi (x, y) dan orientasi (theta).
- **Motion dan Measurement Models:** Didefinisikan oleh fx dan hx. fx memprediksi state baru berdasarkan input kontrol, sedangkan hx memberikan perkiraan pengukuran GPS.
- **Inisialisasi UKF:** Menggunakan MerweScaledSigmaPoints untuk menghasilkan sigma points dan menginisialisasi UKF.
- **Langkah UKF:**
    - **Prediction:** Memprediksi state baru dan covariance menggunakan model gerak dan input kontrol.
    - **Update:** Mengupdate state dan covariance dengan pengukuran GPS baru.

Visualisasi: Mirip dengan EKF, hasil posisi, posisi sebenarnya, dan data GPS yang penuh noise dipetakan untuk menunjukkan performa UKF.

Kode 3: Kalman Filter (KF) untuk Tracking Objek Bergerak

Ringkasan: Kode ini menggunakan Linear Kalman Filter (KF) untuk melacak objek bergerak dengan data sensor yang penuh noise.

Poin Penting:

- **State Vector:** state mencakup posisi dan kecepatan dalam arah x dan y.
- **Motion Model:** motion_model mengupdate state berdasarkan model linear dengan kecepatan konstan.
- **Measurement Model:** measurement_model memberikan pengukuran posisi (x, y) dari state.
- **Jacobian Matrix:** jacobian_measurement memberikan matriks Jacobian untuk model pengukuran.
- **Langkah KF:**
    - **Prediction:** Memprediksi state baru dan matriks covariance menggunakan model gerak.
    - **Update:** Mengupdate state dan covariance dengan pengukuran sensor baru menggunakan Kalman Gain.

Simulasi: Objek mengikuti jalur sinusoidal, dan data sensor yang penuh noise dihasilkan untuk mensimulasikan pengukuran dunia nyata. **Visualisasi:** Hasil estimasi posisi, posisi sebenarnya, dan data sensor yang penuh noise dipetakan.

Kode 4: Kalman Filter untuk Tracking Drone dengan Gerakan Parabola

Ringkasan: Kode ini juga menggunakan Linear Kalman Filter (KF), tapi kali ini untuk melacak drone yang bergerak dengan gerakan parabola yang dipengaruhi gravitasi.

Poin Penting:

- **State Vector:** state mencakup posisi dan kecepatan dalam arah x dan y.
- **Motion Model:** motion_model mengupdate state, termasuk efek gravitasi pada kecepatan-y.
- **Measurement Model:** measurement_model memberikan pengukuran posisi (x, y) dari state.
- **Jacobian Matrix:** jacobian_measurement memberikan matriks Jacobian untuk model pengukuran.
- **Langkah KF:**

- **Prediction:** Memprediksi state baru dan matriks covariance menggunakan model gerak.
- **Update:** Mengupdate state dan covariance dengan pengukuran sensor baru menggunakan Kalman Gain.

Simulasi: Kecepatan-y drone berkurang seiring waktu karena gravitasi, mensimulasikan jalur parabola. **Visualisasi:** Hasil estimasi posisi, posisi sebenarnya, dan data sensor yang penuh noise dipetakan.

Analisis Umum:

Konsep yang Dibahas:

1. **Kalman Filters:** EKF dan UKF memperluas KF dasar untuk menangani model gerak dan pengukuran non-linear.
2. **State Estimation:** Setiap filter memprediksi dan mengoreksi state vector berdasarkan input kontrol dan pengukuran sensor.
3. **Covariance Update:** Semua filter mengupdate matriks covariance untuk merepresentasikan ketidakpastian dalam estimasi state.
4. **Noise Handling:** Proses noise dan pengukuran noise dimodelkan dan digunakan dalam langkah update untuk menangani ketidakpastian dan ketidakakuratan dalam model dan data sensor.
5. **Visualisasi:** Setiap implementasi menyertakan visualisasi jalur sebenarnya, pengukuran noisy, dan jalur estimasi, memberikan wawasan tentang performa filter.

Analisis dan Penjelasan Code 1

Tujuan:

Code ini bertujuan untuk mengontrol robot di simulator Webots sehingga mampu bergerak melalui waypoints tertentu dengan algoritma navigasi berbasis waktu. Robot memiliki beberapa aktuator dan sensor, termasuk motor roda belakang, roda depan, aktuator kemudi, sensor posisi, akselerometer, dan IMU (Inertial Measurement Unit).

Struktur Code:

1. **Inisialisasi Robot**:
   - Robot adalah instance dari robot Webots.
   - timestep menentukan interval waktu simulasi.
2. **Konfigurasi Perangkat**:
   - Perangkat seperti motor belakang, kemudi, sensor posisi, akselerometer, dan IMU diinisialisasi dengan getDevice().

- Motor diatur agar memiliki posisi tak terbatas (setPosition(float('inf'))) dan kecepatan awal nol (setVelocity(0.0)).

3. **Parameter Dinamis**:
    - Variabel seperti left_speed, right_speed, phi, theta, state, dll., digunakan untuk menyimpan kondisi dinamis robot.
    - waypoints adalah daftar koordinat target yang harus dicapai robot.
4. **Fungsi setState(destx, destz)**:
    - Fungsi ini menentukan keadaan (‘state’) berdasarkan posisi relatif robot terhadap waypoint.
    - Contoh state:
        - 0: Rem.
        - 1: Belok ke kiri.
        - 2: Berputar ke kiri.
        - 8: Bergerak lurus.
5. **Fungsi moveToWaypoint(destx, destz, ...)**:
    - Mengatur kecepatan roda belakang dan depan serta posisi kemudi berdasarkan state yang ditentukan.
    - Logika melibatkan perhitungan sudut orientasi robot (theta) dan respons gerakan untuk menuju waypoint berikutnya.
6. **Loop Utama**:
    - Di dalam while robot.step(timestep):
        - Data dari sensor seperti akselerometer, IMU, dan sensor posisi diambil.
        - Posisi robot dihitung menggunakan odometri.
        - Fungsi moveToWaypoint() dipanggil untuk menentukan langkah robot.
        - Kecepatan motor dan posisi aktuator kemudi diperbarui.

Kelebihan:

- Implementasi waypoint yang jelas.
- Mendukung navigasi multi-sumbu (x, z).
- Menggunakan data IMU dan odometri untuk estimasi posisi.

Kekurangan:

- Tidak ada mekanisme penghindaran rintangan.

- Tidak menangani kesalahan sensor atau lingkungan non-ideal.
- State-based approach terlalu bergantung pada timing.

Analisis dan Penjelasan Code 2

Tujuan:

Code kedua mirip dengan code pertama tetapi memiliki beberapa tambahan dan modifikasi pada logika kontrol waypoint, sehingga mendukung simulasi robot dengan cara yang lebih dinamis.

Struktur Code:

1. **Inisialisasi dan Konfigurasi**:
    - Mirip dengan code pertama, perangkat dan parameter robot diatur.
2. **Fungsi setState()**:
    - Sama seperti code pertama, fungsi ini mengatur state. Namun, ada modifikasi pada logika waktu (timing) untuk transisi antar state.
3. **Fungsi moveToWaypoint()**:
    - Menggunakan logika yang sama untuk menentukan gerakan berdasarkan sudut theta.
    - Terdapat variabel tambahan seperti flip untuk menangani orientasi terbalik.
4. **Loop Utama**:
    - Robot membaca data sensor dan menghitung posisi relatifnya.
    - Tidak hanya navigasi ke waypoint, code ini juga mencetak posisi dan sudut orientasi robot untuk debugging.

Kelebihan:

- Menambahkan mekanisme flip untuk orientasi terbalik.
- Logging yang lebih baik untuk debugging.

Kekurangan:

- Masih tidak mendukung penghindaran rintangan.
- Logika timing dapat menyebabkan ketidakstabilan jika timestep berubah.

Kesimpulan

Kedua kode memiliki struktur serupa dengan tujuan menggerakkan robot melalui beberapa waypoint. Perbedaannya terletak pada tingkat penyempurnaan logika navigasi:

- Code pertama lebih sederhana dan cocok untuk pemahaman dasar navigasi robot.

- Code kedua memiliki logika yang lebih kompleks, mendukung kondisi orientasi terbalik, tetapi tetap memiliki keterbatasan pada situasi lingkungan dinamis.

Saran untuk Pengembangan

1. **Penghindaran Rintangan**:
    - Tambahkan sensor jarak atau LIDAR untuk mendeteksi dan menghindari rintangan.
2. **Navigasi Adaptif**:
    - Gunakan algoritma navigasi seperti A* atau Dijkstra untuk menentukan jalur dinamis.
3. **Penggunaan PID Controller**:
    - Implementasikan pengendali PID untuk memastikan kecepatan roda lebih stabil.
4. **Validasi dengan Sensor**:
    - Periksa data sensor seperti IMU untuk mengurangi kesalahan drift pada posisi robot.