КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Катедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ВИЗНАЧЕННЯ ПРОСТОРОВОЇ ОРІЄНТАЦІЇ ЗА ДОПОМОГОЮ ІНТЕГРАЦІЇ ПОКАЗІВ АВТОМОБІЛЬНИХ ДАТЧИКІВ

Виконав студент 4 курсум
Клячкін Андрій Вадимович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Панченко Тарас Володимирович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні катедри теорії та технології програмування
«____»_____202_ р.
протокол №_____

Завідувач катедри

Нікітченко М.С.

(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи: 45 сторінок, 19 ілюстрацій, 14 джерел посилань.

БЕЗЗАПАХОВИЙ ФІЛЬТР КАЛМАНА, ВИЗНАЧЕННЯ ПРОСТОРОВОЇ ОРІЄНТАЦІЇ, ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЗА ВІДСУТНОСТІ GPS, ІНТЕГРАЦІЯ ПОКАЖЧИКІВ ДАТЧИКІВ, ФІЛЬТР КАЛМАНА, SENSOR FUSION.

Об’єктом аналізу є алгоритм фільтру Калмана та його варіанти, його застосування до інтеграції показів автомобільних сенсорів. Об’єктом розробки є програмна реалізація беззапахового фільтру Калмана, що дозволяє визначити місцезнаходження та орієнтацію автівки в умовах слабкого або відсутнього сигналу GPS.

Метою роботи є дослідження алгоритму фільтра Калмана та методів його застосування, а також створення програмного засобу для визначення положення та орієнтації автівки в умовах слабкого або відсутнього сигналу GPS з використанням цього алгоритму.

Методи розробки: комп’ютерне моделювання, проєктування архітектури програми, каскадна методологія розробки.

Інструменти розробки: інтегроване середовище розробки PyCharm, мова програмування Python.

Результати роботи: досліджено теоретичні засад фільтру Калмана там сам алгоритм. Проаналізовано різні види цього алгоритму, різні моделі руху автівки. Розроблено програмний продукт, що дозволяє визначати положення та орієнтацію автомобіля за умов слабкого або відсутнього сигналу GPS.

Розроблений програмний продукт може застосовуватися для покращення автомобільної навігації в умовах щільної забудови або іншої місцевості, що перешкоджає сигналу GPS.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки: Зараз у переважній більшості автомобілів для просторової орієнтації використовується система GPS. Це – перевірена часом система, що використовується для навігації у всьому світі. Із самого початку її роботи, її точність постійно збільшується – станом на 2018 рік, за оптимальних умов найновіші приймачі визначають положення в межах 0.3 м.

Але оптимальні умови досягаються далеко не завжди. Сигналу GPS можуть перешкоджати високі будівлі, дахи тонелів, або різноманітні природні об’єкти. В таких умовах точність навігації значно зменшується, а у деяких випадках сигнал зникає повністю. За таких умов доцільно коригувати дані GPS за допомогою локальних даних про стан автівки – швидкість, прискорення, кут нахилу керма. Цей процес називається інтеграцією сенсорів (Sensor Fusion), та здійснюється за допомогою низки алгоритмів, найрозповсюдженішим з яких є фільтр Калмана і його похідні.

Фільтр Калмана розроблявся П. Сверлінгом, Р. Калманом та Т.Н. Тіле з 1958 по 1961 роки, і відразу його було використано у космічній програмі США для позиціонування космічного корабля «Аполлон» [1]. З того моменту початковий алгоритм зазнав багатьох модифікацій та покращень, що призвели до створення таких нових алгоритмів, як розширений та беззапаховий фільтри Калмана.

Актуальність роботи та підстави для її виконання: Точна навігація завжди була важливою темою для будь-якої індустрії. Зараз, із розвитком технологій безпілотних автівок такими компаніями, як Tesla та Google, це ще більш актуально, адже у такому разі в транспортному засобі немає водія, що може коректувати курс. До того ж, навіть за оптимальних умов точності GPS може не вистачати для визначення більш дрібних деталей позиції тіла – наприклад, в якій смузі багатосмугової траси знаходиться автомобіль, чи не пересікає він суцільну лінію, або чи можливе зіткнення, якщо прямувати встановленим курсом.

Іншою галуззю, де потрібна висока точність позиціонування та визначення орієнтації, є просунені системи допомоги водію (Advanced Driver Assistance Systems, ADAS). Це – електронні системи, що допомагають водієві керувати автомобілем. Наприклад, система може виводити на вітрове скло прокладений маршрут, накладаючи його на дорогу, створюючи таким чином гібридний інтерфейс, або система може допомагати при парковці, чи навіть повністю автоматизувати цей процес. Для виконання цих завдань системі потрібен неперервний потік точної інформації про позицію та орієнтацію автомобіля.

Отже, наразі існує багато галузей, де постає проблема пошуку надійного джерела точної інформації про стан автівки. У цій роботі буде розглянуто один з алгоритмів, що вирішує цю проблему, та реалізовано його.

Мета роботи: дослідження роботи беззапахового фільтру Калмана, його застосування для Sensor Fusion, та розробка програмного засобу, що реалізує алгоритм.

Завдання роботи:

- Дослідити алгоритм фільтру Калмана та деяких його варіацій
- Дослідити наявні математичні моделі руху автомобіля
- Розробити програму, що реалізує алгоритм із використанням обраних моделей руху
- Протестувати програму на реальних даних

Об'єкт, предмет та методи дослідження: Об’єктом дослідження є процес визначення точної позиції та орієнтації автівки за допомогою інтеграції сенсорних даних.

Предметом дослідження є використання безапахового фільтру Калмана для визначення точної позиції та орієнтації автівки за допомогою інтеграції сенсорних даних.

Розробці ПЗ передувало дослідження теоретичних засад фільтру Калмана; створення математичної моделі руху автівки; систематизація та аналіз сенсорних

даних та виду, у якому вони надходять; аналіз та реалізація методів перетворення сенсорних даних у необхідні системи виміру.

Інструментом створення програмного засобу є середовище PyCharm та мова програмування Python.

Можливі сфери застосування. Розроблений програмний продукт може застосовуватися для покращення просторової орієнтації автівок, а також для використання як компонент просунених систем допомоги водієві.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ОЦІНЮВАННЯ СТАНУ

Задача визначення положення та орієнтації авто є, насамперед, задачею обробки показників сенсорів – за допомогою інформації про початкове положення автівки, швидкість та напрям можна, за умов достатньої точності інформації, точно визначити поточне положення. Але, на жаль, всі датчики мають похибку. Отже, доцільно враховувати точність кожного сенсора під час обчислення наступного положення. Зазвичай, це робиться за допомогою теорії ймовірності. Галузь, що вивчає цю проблему – ймовірне рекурсивне оцінювання стану.

1.1 Вступ до ймовірнісного рекурсивного оцінювання стану

Ціллю оцінювання стану є оцінювання таких значень (компонент стану), що їх неможливо визначити напряму, але можна вивести з доступної інформації. Положення та орієнтація об’єкту (наприклад, машини) у просторі є такими значеннями. Їх необхідно розраховувати з наявної інформації з різних джерел – наприклад, датчики швидкості, інерціальні датчики та інші. Ця задача ускладнюється тим, що сенсорні дані можуть мати шум.

Ймовірнісне оцінювання стану – процес обчислення ймовірності перебування у різних станах. Базуючись на цих розподілах, можна зробити висновок, у якому стані насправді знаходиться об’єкт.

Стан – це набір всіх параметрів об’єкта та навколишнього середовища, що можуть вплинути на майбутнє. Так, наприклад, у позиціонуванні автівки станом може бути його положення, швидкість, прискорення та орієнтація. У більш складних задачах у стан також може бути записане, наприклад, місцезнаходження людей та інших автівок навколо.

Взаємодія об’єкта із навколишнім середовищем відбувається через **спостереження** і **керувальні дії**.

Спостереження надають об’єктові інформацію про стан – наприклад, автівка може отримати дані з GPS, що містять інформацію про позицію, швидкість та орієнтацію, і може отримати інформацію з сенсорів.

Керувальні дії (керування) – дії, що змінюють стан. Прикладом такої дії є збільшення швидкості.

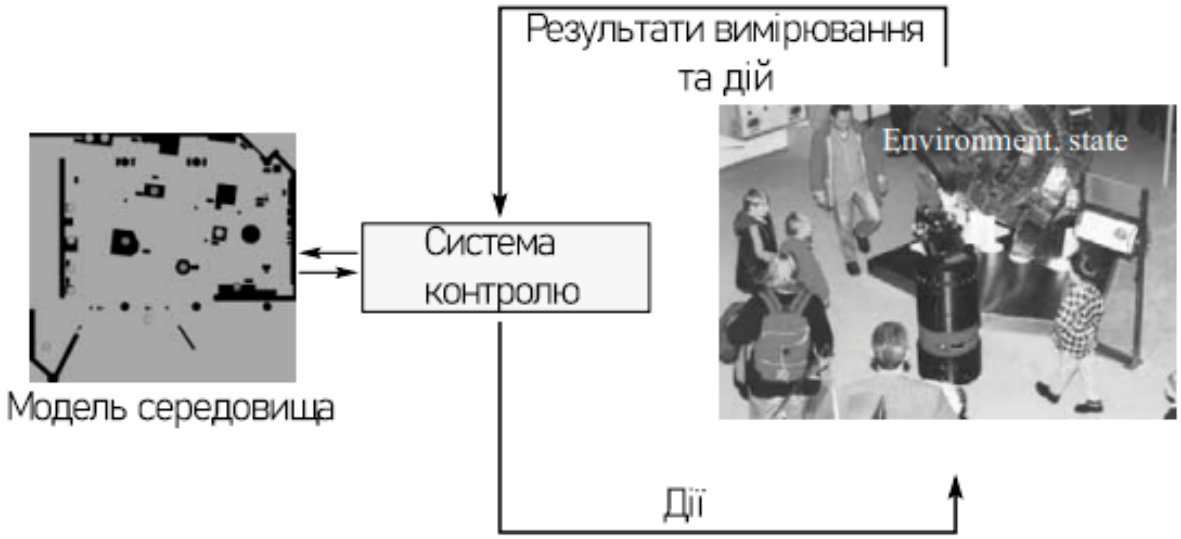


Рисунок 1 - взаємодія об'єкта з середовищем

Дані, що стосуються спостережень на кроці i, позначатимемо z_i. Дані, що стосуються керування на кроці i, позначатимемо u_i. Дані керування можуть містити, напирклад, поточне прискорення, чи поточну швидкість. Припустимо, що заміри відбуваються після керування – тобто, спочатку об’єкт зазнає керування u_t, і це впливає на стан x_t, і потім об’єкт проводить замір z_t, що відповідає стану x_t.

Перехід зі стану у стан, проведення спостереження та керування – стохастичні процеси. Це значить, що стан x_t утворюється випадково зі стану x_{t-1}. Таким чином, перехід з одного стану у наступний можна характерезувати функцією розподілу ймовірностей:

$$p(x_t|x_{0,\dots,t-1}, u_{0,\dots,t}, z_{0,\dots,t-1}) \quad (1)$$

Важливим і необхідним припущенням рекурсивного оцінювання станів є припущення, що у системі, що розглядається, стани є **повними**. Це значить, що, щоб передбачити наступний стан, достатньо інформації з поточного, і точність передбачення не підвищиться, якщо використовувати інформацію і з попередніх станів. Стани з x_0 по x_{t-1} ніяк не впливають на стан x_{t+1}, окрім як через стан x_t.

Враховуючи повноту станів, (1) спростити, так зі станів лише попередній впливає на поточний:

$$p(x_t|x_{t-1}, u_{0,\dots,t}, z_{0,\dots,t-1}) \quad (2)$$

Можна помітити, що, оскільки стан є повним, то він вже включає в себе інформацію про всі попередні керування і заміри, а отже, можна ще спростити (2) до:

$$p(x_t|x_{t-1}, u_t) \quad (3)$$

Цю ймовірність називають **ймовірністю переходу**. Ймовірність переходу описує еволюцію станів з часом об'єкту як функцію від керувань та (не обов'язково) часу.

Процес спостереження також є стохастичним процесом, що описується функцією розподілу:

$$p\big(z_t\big|x_{0\dots t}, u_{0,\dots,t}, z_{0,\dots,t-1}\big), \quad (4)$$

і спрощується до

$$p(z_t|x_t) \quad (5)$$

Цю ймовірність називають **ймовірністю спостереження**. Вона описує сприйняття стану об'єктом як функцію від стану та (не обов'язково) часу.

На рис. 2 зображено процес еволюції стану та спостережень. Стан в момент часу t залежить лише від стану у момент $t-1$ та керування у момент t, спостереження в момент часу t залежить виключно від стану у момент t. Таку модель еволюції станів називають **прихованою марківською моделлю**, або **динамічною байєсівською мережею**.

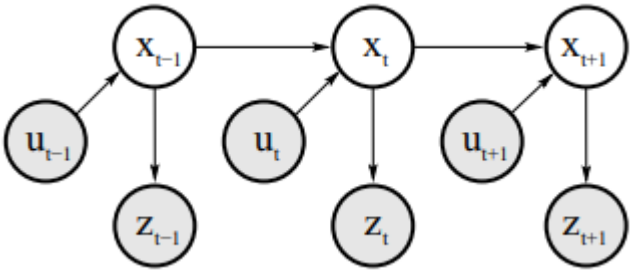


Рисунок 2 – еволюція станів; стрілки ілюструють залежності

Але об’єкт (робот, автівка) не може знати справжнього значення стану. Він може лише вгадувати його значення, базуючись на наявній інформації. Для стану в момент t ця інформація – керування $u_{1\dots t}$ та спостереження $z_{1\dots t}$. Вгадане значеня стану називають **припущенням** (belief) і вираховують через **розподіл припущення**. Цей розподіл співставляє кожній гіпотезі щодо справжнього значення стану її апостеріорну ймовірність:

$$bel(x_t) = p(x_t|z_{1\dots t}, u_{1\dots t}) \tag{6}$$

Іноді (зокрема, в фільтрі Калмана) необхідно обчислювати $bel(x_t)$ без знання z_t:

$$\overline{bel}(x_t) = p(x_t|z_{1\dots t-1}, u_{1\dots t}) \tag{7}$$

Це значення називають **передбаченням**, бо воно є передбаченням наступного стану, що ґрунтується на попередьому припущенні без врахування спостереження у момент t. Процес врахування спостереження, тобто, обрахунку $bel(x_t)$ з $\overline{bel}(x_t)$, називають **уточненням,** або **корекцією**.

1.2 Байєсів фільтр

Байєсів фільтр – найпростіший та найбільш загальний алгоритм для рекурсивної ймовірнісної оцінки стану. На ньому базуються майже всі інші алгоритми. Алгоритм рекурсивно обчислює розподіл припущення з інформації про

останнє спостереження, останнє керування та попередній стан. Алгоритм викладено на рис. 3:

1. ***алгоритм байєсівський_фільтр***($bel(x_{t-1}), u_t, z_t$):
2. *для всіх можливих значень стану* x_t*:*
3. $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$ *// Передбачення*
4. $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$ *// Уточнення*
5. *повернути* $bel(x_t)$

Рисунок 3 – алгоритм фільтру Байєса [2]

Для кожного з можливих значень стану x_t алгоритм складається з двох кроків: передбачення і уточнення.

Передбачення відбувається у рядку 3. Алгоритм обробляє керування u_t, обчислюючи припущення щодо стану x_t з керування та припущення щодо попереднього стану. Він робить це, обчислюючи інтеграл добутку двох розподілів: ймовірності, що керування u_t приведе об’єкт з попереднього стану саме у цей, та припущення щодо попереднього стану.

Уточнення ж відбувається в рядку 4. Алгоритм обчислює значення розподілу припущення для x_t як добуток ймовірності отримати спостереження z_t за такого значення стану на передбачене значення $\overline{bel}(x_t)$. Алгоритм **уточнює** значення розподілу, враховуючи останнє спостереження. Результат добутку не завжди знаходитиметься на проміжку [0,1], отже, не завжди є ймовірністю. Через це необхідно множення на константу нормалізації η.

Оскільки алгоритм обчислює розподіли рекурсивно зі значень попередніх розподілів, необхідно задати початкове значення $bel(x_0)$. Найбільш розповсюдженими є два крайові випадки: значення початкового стану відомо точно

і значення початкового стану не відомо взагалі. У першому випадку функцію задають так:

$$bel(x_0) = \begin{cases} 1, & x_0 = a \\ 0, & \text{інакше} \end{cases}$$

де a – точно відоме значення стану у момент 0.

У разі того, що значення не відомо взагалі, $bel(x_0)$ задається рівномірним розподілом по всім можливим значенням початкового стану. Також можливо таке, що значення відомо з певною точністю, але не точно. У такому разі $bel(x_0)$ можна задати відповідним нерівномірним розподілом.

Байєсівський фільтр є обмеженим алгоритмом у тому, що кроки передбачення та уточнення вимагають або можливості аналітично обчислити інтеграл у рядку 3 алгоритму та добуток у рядку 4, або кінцевого набору станів, щоб замінити інтеграл скінченною сумою. Це накладає обмеження на модель об’єкту (бо обмежується або набір можливих функцій розподілу припущення, або набір моживих станів).

1.3 Фільтр Калмана

На базі Байєсового фільтру розроблено багато алгоритмів – фільтр Калмана, гістограмний фільтр, частинковий фільтр [3] та інші. Найбільш розповсюджений з них – фільтр Калмана. У цьому алгоритмі припущення у момент t представлено середнім значенням μ_t та матрицею коваріації Σ_t. Найпростішим та найпершим його варіантом є лінійний фільтр Калмана.

Для коректної роботи ЛФК необхідно, щоб система задовольняла, крім припущення про повноту стану, трьом обмеженням:

1. Функція ймовірності переходу $p(x_t|u_t, x_{t-1})$ має бути лінійною за u_t та x_{t-1} з доданням Гаусівського шуму:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{8}$$

де x_{t-1}, x_t – вектори стану;

u_t – вектор керування;

ε_t – Гаусівський шум.

Вектор стану має вигляд:

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix},$$

вектор керування має вигляд:

$$u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{m,t} \end{pmatrix}.$$

A_t – матриця розміру $n \times n$, де n – розмір вектору стану x_t.

B_t – матриця розміру $n \times m$, де m – розмір вектору керування u_t.

ε_t – гаусівський вектор з середім значенням, що дорівнює нулю, та матрицею коваріації R_t. Виміри вектора ε_t такі самі, як виміри вектору стану. Цей вектор моделює неточність переходу зі стану у стан під впливом керування (у випадку з автівкою це може бути спричинене, наприклад, буксуванням або впливом вітру).

Таким чином, розподіл ймовірності переходу є багатовимірним гаусовим розподілом:

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \times$$

$$\times \exp\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T {R_t}^{-1}(x_t - A_t x_{t-1} - B_t u_t)\} \quad (9)$$

2. Ймовірність спостереження $p(z_t|x_t)$ має бути лінійною функцією за x_t з доданим Гаусовим шумом:

$$p(z_t|x_t) = C_t x_t + \delta_t, \quad (10)$$

де x_t – вектор стану;

C_t – матриця розміру $k \times n$, де k – розмір вектору спостережень;

δ_t – гаусівський вектор з середім значенням, що дорівнює нулю, та матрицею коваріації Q_t. Він моделює неточність спостережень – наприклад, неточність датчиків.

Таким чином, розподіл ймовірності спостережень також є багатовимірним гаусовим розподілом:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - C_t x_t)^T {Q_t}^{-1}(z_t - C_t x_t)\} \quad (11)$$

3. Початкове припущення має бути випадковою величиною з гаусовим розподілом:

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x_0 - \mu_0)^T {\Sigma_0}^{-1}(x_0 - \mu_0)\} \quad (12)$$

Якщо всі ці три умови виконуються, то для будь-якого моменту часу t результат ітерації фільтру – апостеріорне припущення $bel(x_t)$ – буде нормально розподіленим.

Алгоритм викладено на рис. 4:

1. ***алгоритм лінійний_фільтр_калмана***$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$:
2. $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$ *// Передбачення*
3. $\bar{\Sigma}_t = A_t\Sigma_{t-1}{A_t}^T + R_t$
4. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$ *// Уточнення*
6. $\Sigma_t = (I - K_t C_t)\,\bar{\Sigma}_t$
7. *повернути* $< \mu_t, \Sigma_t >$

Рисунок 4 – алгоритм лінійного фільтру Калмана [2]

У рядках 2 та 3 за результатами керування обчислюється наступний стан, представлений середнім значенням та коваріацією. Це робиться підставленням у

формулу (4) замість x_{t-1} середнього значення μ_{t-1} у рядку 2 та коваріації Σ_{t-1} у рядку 3.

У рядку 4 обчислюється центральне значення алгоритму – передавальний коефіцієнт Калмана (англ. Kalman Gain). Це значення задає, наскільки сильно спостереження впливатиме на результат. Воно обернено пропорційне значенню $C_t\bar{\Sigma}_t C_t^T + Q_t$ – результату підстановки передбаченої коваріації у рівняння спостереження, тобто, коваріації спостереження.

У рядках 5 та 6 відбувається уточнення. Виразом $z_t - C_t\bar{\mu}_t$ обчислюється **іновація** – відстань між очікуваним та дійсним спостереженнями.

Рисунки 5 та 6 ілюструють роботу фільтру Калмана. На рисунку 5 зображено результат роботи фільтру протягом 50 моментів часу (50 ітерацій). На рисунку 6 детально зображено роботу однієї ітерації. Середня квадратична помилка результатів роботи фільтру відносно справжніх станів в два рази менша за середню квадратичну помилку спостережень.

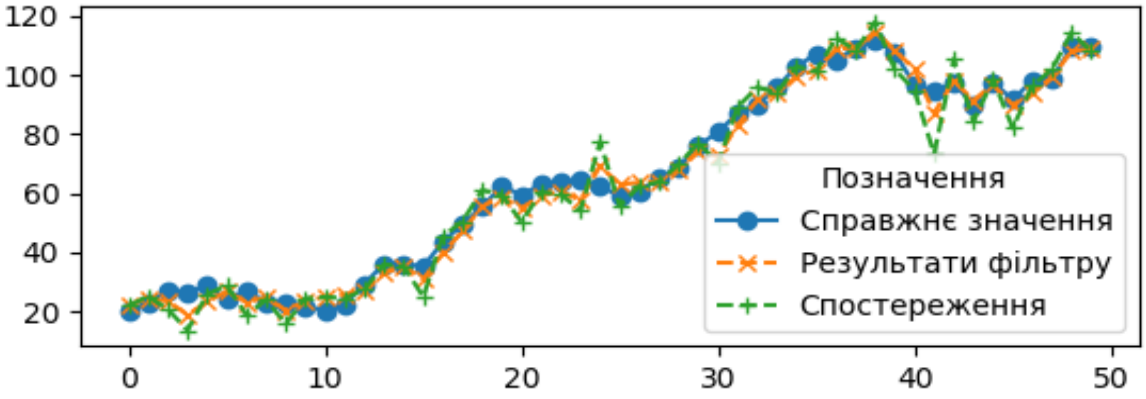


Рисунок 5 – результат роботи фільтру

1.4 Фільтри Калмана для нелінійних систем

Незважаючи на свою ефективність, немодифікований фільтр Калмана не підходить до вирішення поставленої задачі через обмеження на лінійність, яке він накладає на функцію переходу та спостережень – рух автівки неможливо достатньо

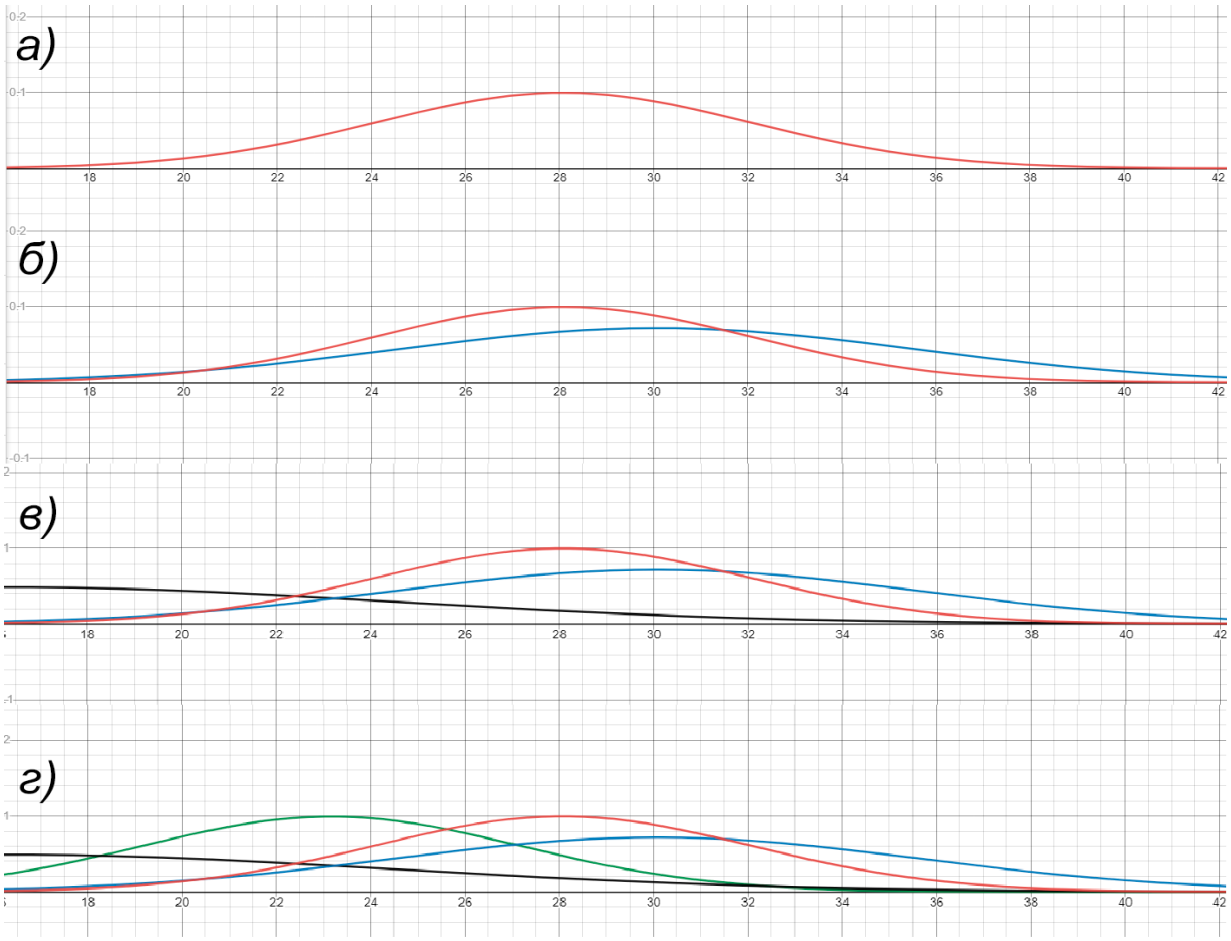


Рисунок 6 – детальний розбір одної ітерації лінійного фільтру Калмана: а – розподіл попереднього стану (черв.); б – розподіл попереднього стану (черв.) та розподіл наступного стану після передбачення (бл.); в – вищеозначені розподіли та розподіл поточного спостереження (чорн.); г – вищеозначені розподіли та розподіл наступного стану після уточнення (зел.)

точно передати лінійною системою (докладніше про це – у розділі 2). У підрозділі 1.4.1 буде стисло описано розширений фільтр Калмана, у підрозділі 1.4.2 буде викладено беззапахового фільтру Калмана та беззапахового перетворення, яке є основою його роботи.

1.4.1 Розширений фільтр Калмана

Для зняття обмеження на лінійність розширений фільтр Калмана використовує лінеаризацію нелінійних функцій, розкладаючи їх в ряд Тейлора першого порядку. Замість матриць переходу та спостереження A_t та B_t відповідно, в ньому використовуються матриці Гесе – матриці, що складаються з часткових похідних – функцій переходу та спостереження.

Цей підхід є досить точним для багатьох використань, але сильно втрачає точність зі збільшенням нелінійності моделі та варіації стану. До того ж, він вимагає обчислення похідних щоітерації. Наступна модифікація фільтру Калмана – беззапаховий фільтр Калмана – виправляє ці недоліки.

1.4.2 Беззапаховий фільтр Калмана

Для зняття обмеження на лінійність беззапаховий фільтр Калмана використовує так зване **беззапахове перетворення**. Це – алгоритм, що дозволяє наблизити розподіл випадкової величини, що є результатом застосування нелінійної функції до іншої випадкової величини. Він є набагато ефективнішим, ніж метод Монте-Карло, з невеликою втратою точності.

Нехай є задача дізнатися параметри (середнє значення та матрицю коваріації) розподілу випадкової величини Y, та

$$Y = g(X)$$

де g – нелінійна функція;

X – випадкова величина з середнім значенням $\bar{X}$ та матрицею коваріації $\sum_X$. Алгоритм беззапахового перетворення дозволяє це зробити таким чином:

1. Обчислити значення параметру λ:

$$\lambda = \alpha^2(L + \kappa) - L,$$

де: α – параметр, що контролює відстань від сігма-точок до середнього значення;

β – параметр, що дозволяє зберегти інформацію про розподіл X;

κ – другорядний параметр, що контролює відстань від СТ до середнього значення;

L – розмірність випадкової величини X.

2. Обчислити $2L + 1$ сігма-точок χ_i:

$$\chi_0 = \bar{X}; \tag{13}$$

$$\chi_i = \bar{X} + \sqrt{(L+\lambda)\sum_{X_i}} \quad i = 1, \dots, L \tag{14}$$

$$\chi_i = \bar{X} - \sqrt{(L+\lambda)\sum_{X_i}} \quad i = L+1, \dots, 2L \tag{15}$$

де $\sqrt{(L+\lambda)\sum_{X_i}}$ – i-тий рядок матриці $(L+\lambda)\sum_X$;

3. Обчислити для кожної з сігма-точок відповідну вагу:

$$W_0^{(m)} = \frac{\lambda}{L+\lambda}; \tag{16}$$

$$W_0^{(c)} = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta); \tag{17}$$

$$W_i^{(c)} = W_i^{(m)} = \frac{1}{2(L+\lambda)}, \quad i = 1, \dots, 2L \tag{18}$$

4. Обчислити результат застосування цільової функції g до сігма-точок:

$$y_i = g(\chi_i), \qquad i = 0, \dots, 2L \tag{19}$$

5. Обчислити середнє значення та матрицю коваріації розподілу Y:

$$\bar{y} = \sum_{i=0}^{2L} W_0^{(m)} y_i \tag{20}$$

$$\Sigma_y = \sum_{i=0}^{2L} W_i^{(c)} (y_i - \bar{y})(y_i - \bar{y})^T \tag{21}$$

Якщо X має нормальний розподіл, параметри α, β, κ задаються значеннями 10^{-3}, 2 та 0 відповідно. Інші розповсюджені варіанти задання параметрів включають: $< \alpha = 1, \beta = 2, \kappa = 0 >$; $< \alpha = \sqrt{3}, \beta = 2, \kappa = 1 >$; $< \alpha = 1, \beta = 0, \kappa = 3 >$.

Дію цього алгоритму ілюструє рис. 7. Зліва зображено результат методу Монте-Карло, по центру – результат лінеаризації функції g, яка використовуєтсья у розширеному фільтрі Калмана, справа – результат беззапахової трансформації. Як можна побачити, БТ досягає точності, близької до точності методу Монте-Карло, і набагато вищої за точність лінеаризації, при цьому обчислюючи набагато менше (у цьому випадку – $2 * 2 + 1 = 5$) значень.

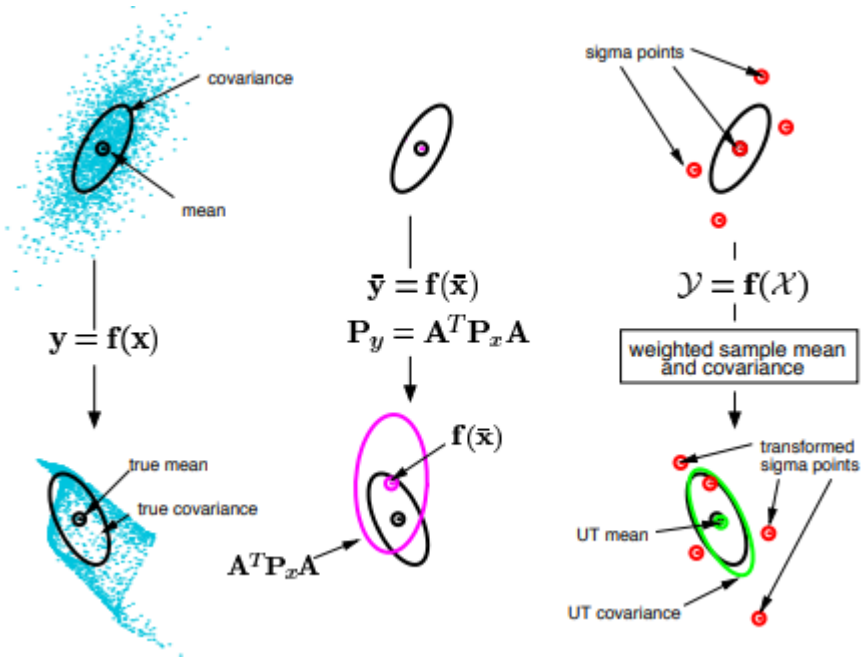


Рисунок 7 – результати застосування методу Монте-Карло, лінеаризації та беззапахової трансформації для визначення параметрів трансформованого розподілу

Беззапахове перетворення у беззапаховому фільтрі Калмана застосовується для того, щоб зняти обмеження на лінійність функцій переходу та спостереження. Нехай система має функцію переходу $F(x)$ та функцію спостереження $H(x)$. Завдяки використанню беззапахового перетворення, немає значення, чи є вони лінійними за своїми аргументами, чи ні. Псевдокод однієї ітерації алгоритму зображено на рис. 8. Важливим зауваженням є те, що ваги $W^{(m)}$ та $W^{(c)}$ можна не обчислювати щоітерації, бо їх значення залежать лише від константних параметрів.

1. ***алгоритм беззапаховий_фільтр_калмана***$(x_{t-1}, \Sigma_{t-1}, u_t, z_t)$:
2. $\chi_t = [\bar{x}_{t-1} \quad \bar{x}_{t-1} \pm \sqrt{(L+\lambda)\sum_{t-1}}]$
3. $W^{(m)} = [\frac{\lambda}{L+\lambda} \quad \frac{1}{2(L+\lambda)}]$
4. $W^{(c)} = [\frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \quad \frac{1}{2(L+\lambda)}]$
5. $\chi_t^f = F(\chi_t, u_t)$
6. $\bar{x}_t^{pr} = \sum_{i=0}^{2L} W_i^{(m)} \chi_{i,t}^f$
7. $\sum_t^{pr} = \sum_{i=0}^{2L} W_i^{(c)} \{\chi_{i,t}^f - \bar{x}_t^{pr}\}\{\chi_{i,t}^f - \bar{x}_t^{pr}\}^T$
8. $y_t = [\bar{x}_t^{pr} \quad \bar{x}_t^{pr} \pm \sqrt{(L+\lambda)\sum_t^{pr}}]$
9. $z_t^{pr} = H(y_t)$
10. $\bar{z}_t^{pr} = \sum_{i=0}^{2L} W_i^{(m)} z_{i,t}^{pr}$
11. $\sum_{z_t} = \sum_{i=0}^{2L} W_i^{(c)} \{z_{i,t}^{pr} - \bar{z}_t^{pr}\}\{z_{i,t}^{pr} - \bar{z}_t^{pr}\}^T$
12. $\sum_{x_t z_t} = \sum_{i=0}^{2L} W_i^{(c)} \{\chi_{i,t}^f - \bar{x}_t^{pr}\}\{z_{i,t}^{pr} - \bar{z}_t^{pr}\}^T$
13. $K_t = \sum_{x_t z_t} \sum_{z_t}^{-1}$
14. $\bar{x}_t = \bar{x}_t^{pr} + K_t(z_t - \bar{z}_t^{pr})$
15. $\Sigma_t = \sum_t^{pr} - K_t \sum_{z_t} K_t^T$
16. ***повернути*** $<\bar{x}_t, \Sigma_t>$

Рисунок 8 – алгоритм беззапахового фільтру Калмана [4]

З точки зору фільтру Калмана, у рядках з 1 по 7 здійснюється передбачення, а у рядках з 8 по 15 – уточнення. У 2 рядку обчислюються сігма-точки, у 3 та 4 – значення ваг. У 5 рядку сігма-точки пропускаються через функцію переходу F. У рядках 6 та 7 обчислюються передбачені (апріорні) середнє значення та коваріація стану у момент t. У рядку 8 обчислюються сігма-точки для розподілу передбачених значень. З 9 по 11 нових сігма-точок застосовується функція спостереження і

обчислюються передбачені значення спостереження. У рядку 12 обчислюється коваріація передбачень та передбачених спостережень. У рядку 13 обчислюється коефіцієнт Калмана. У 14 та 15 рядках обчислюється результат роботи ітерації – остаточні значення стану у момент часу t і його матриця коваріації.

Результат роботи беззапахового фільтру Калмана протягом 10 ітерацій над системою, де:

$$F(x_{t-1}, u) = x_{t-1} + u^2 + \varepsilon_f$$

$$H(x_t) = x_t + \varepsilon_h,$$

зображено на рис. 9.

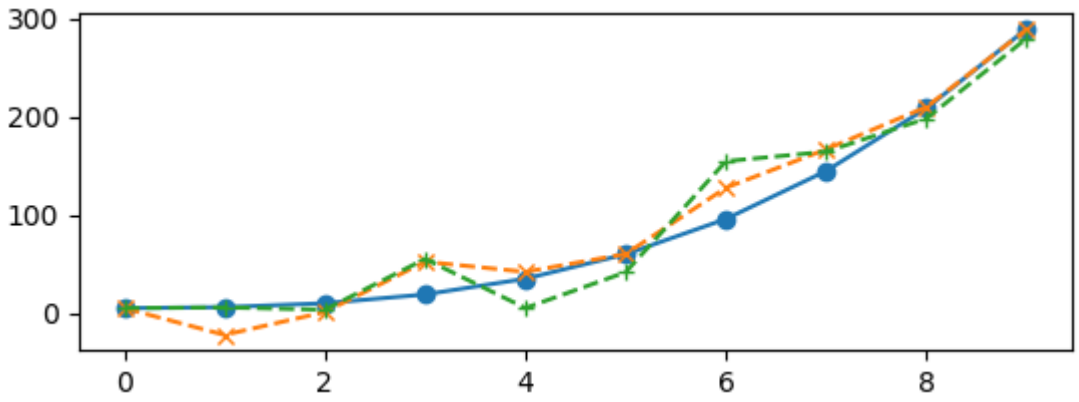


Рисунок 9 – результат роботи беззапахового фільтру Калмана. Зеленим кольором позначено спостереження, помаранчевим – результат роботи фільтру

Точність алгоритму вимірювалася середньою абсолютною різницею між виходом алгоритму та справжнім станом. Алгоритм досяг середньої точності, в 1.5 разів більшої, ніж точність спостережень.

РОЗДІЛ 2. МОДЕЛЬ РУХУ АВТІВКИ

Метою роботи є застосування розгляненого алгоритму – беззапахового фільтру Калмана – до задачі визначення розташування автівки. Для роботи алгоритму потрібна функція переходу – у цьому випадку, модель руху автівки. У підрозділах цього розділу буде розглянуто моделі, що існують, їх переваги та недоліки.

2.1 Класифікація моделей

Моделі руху автівок можна класифікувати за багатьма критеріями, але найбільш ключовою відмінністю між ними є складність, яку можна означити як кількість параметрів, які у моделі вважаються константними (рис. 10) [5].

Найпростішими є моделі CV (Constant Velocity – константна швидкість) та CA (Constant Acceleration – константне прискорення). Їх також називають **моделями лінійного руху**, оскільки вони моделюють прямолінійний рух, і не здатні моделювати поворот автівки. Їх перевагою є те, що вони моделюють зміну координат автівки за допомогою лінійних рівнянь.

Більш складними є **криволінійні** моделі. До таких відносяться модель CTRV (Constant Turn Rate and Velocity – константні швидкість повороту та

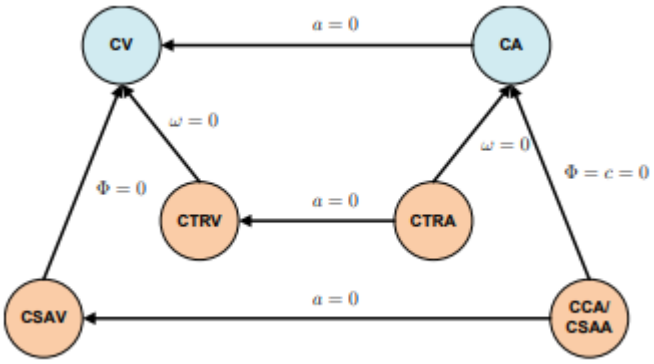


Рисунок 10 – схема, що ілюструє відмінності між різними моделями руху

швидкість) та CTRA (Constant Turn Rate and Acceleration – константні швидкість повороту та прискорення). Вони додають до стану параметр, що відображає орієнтацію автівки (кут повороту) та швидкість зміни орієнтації.

Існують і більш складні моделі – наприклад, CCA та CSAV, але їх недоцільно розглядати у контексті визначення положення та орієнтації цивільної автівки через складність їх реалізації та складність отримання даних, необхідних для їх роботи (наприклад, дані про ковзання шин).

2.2 Моделі CV та CA

Ці моделі є найпростішими в реалізації та найкращими за швидкістю роботи, але мають дуже низьку точність. В термінах фільтру Калмана, їх вектори стану та рівняння переходу мають вигляд:

а. Для CV:

$$x(t) = (x, v_x, y, v_y) \tag{22}$$

$$x(t + \Delta t) = x(t) + \begin{pmatrix} \Delta t * v_x \\ 0 \\ \Delta * v_y \\ 0 \end{pmatrix} \tag{23}$$

де x, y – положення об’єкту за вісями x та y відповідно;

v_x, v_y – швидкість об’єкту за вісями x та y відповідно;

Δt – час, що пройшов з моменту останнього передбачення стану.

б. Для CA:

$$x(t) = (x, v_x, a_x, y, v_y, a_y) \tag{23}$$

$$x(t + \Delta t) = x(t) + \begin{pmatrix} \Delta t * v_x + \frac{1}{2} a_x \Delta t^2 \\ a_x \Delta t \\ \Delta t * v_y + \frac{1}{2} a_y \Delta t^2 \\ a_y \Delta t \end{pmatrix} \tag{24}$$

де x, y – положення об’єкту за вісями x та y відповідно;

v_x, v_y – швидкість об’єкту за вісями x та y відповідно;

a_x, a_y – прискорення об’єкту за вісями x та y відповідно;

Δt – час, що пройшов з моменту останнього передбачення стану.

Хоча модель CA точніша за модель CV, головним недоліком обох цих моделей все ще є точність. Мала точність спричинена тим, що ці моделі не враховують зміну орієнтації об’єкта (кута повороту). Їх достатньо для вирішення простих задач, які не включають в себе визначення орієнтації об’єкту, але для більш складних задач, таких як задача, що поставлена в цій роботі, забезпечуваної їми точності не вистачає.

2.3 Модель CTRV

Модель CTRV є наступною за складністю після лінійних моделей. Вона додає у вектор стану кут повороту та швидкість повороту.

Вектор стану та рівняння переходу цієї моделі мають вигляд:

$$x(t) = (x, y, \theta, v, \omega) \tag{25}$$

$$x(t + \Delta t) = x(t) + \begin{pmatrix} \frac{v}{w}(\sin(\omega \Delta t + \theta) - \sin(\theta)) \\ -\frac{v}{w}(\cos(\omega \Delta t + \theta) - \cos(\theta) \\ \omega \Delta t \\ 0 \\ 0 \end{pmatrix} \tag{26}$$

де x, y – положення об’єкту за вісями x та y відповідно;

θ – кут повороту об’єкту (орієнтація);

v – швидкість руху об’єкту;

ω – швидкість зміни кута повороту об’єкту (кутова швидкість).

Ця модель забезпечує набагато більшу точність, ніж моделі лінійного руху, завдяки включенню зміни кута повороту у вектор стану та рівняння переходу. Зазвичай, саме цю модель використовують у задачах визначення положення та орієнтації як автівок, так і у авіації.

2.4 Модель CTRA

Модель CTRA – розвиток моделі CTRV, де до вектору стану додається прискорення об’єкту. Модель має вигляд:

$$x(t) = (x, y, \theta, v, a, \omega) \tag{27}$$

$$x(t + \Delta t) = x(t) + \begin{pmatrix} v * \frac{(\sin(\theta + \omega\Delta t) - \sin(\theta))}{\omega} + \\ + \frac{a * (\cos(\theta + \omega\Delta t) - \cos(\theta)) + a\omega * \Delta t * \sin(\theta + \omega\Delta t)}{\omega^2} \\ v * \frac{(\cos(\theta + \omega\Delta t) - \cos(\theta))}{\omega} + \\ + \frac{a * (\sin(\theta + \omega\Delta t) - \sin(\theta)) - a\omega * \Delta t * \cos(\theta + \omega\Delta t)}{\omega^2} \\ \omega\Delta t \\ a\Delta t \\ 0 \\ 0 \end{pmatrix} \tag{28}$$

де x, y – положення об’єкту за вісями x та y відповідно;

θ – кут повороту об’єкту (орієнтація);

v – швидкість руху об’єкту;

a – прискорення об’єкту у напрямку руху;

ω – швидкість зміни кута повороту об’єкту (кутова швидкість)

[6].

Ця модель є найбільш точною з розглянутих, і теж є широковикористовуваною у задачах визначення місцезнаходження та орієнтації чи наближення траєкторії.

2.5 Вибір моделі

Постає питання вибору з урахуванням переваг та недоліків розглянутих моделей. Моделі CTRA, або CTRV є компромісом між складністю реалізації та точністю. Вони точніші за моделі прямолінійного руху, а також простіші у реалізації за моделі CCA та CSAV з незначною втратою точності. У цій роботі буде розглянено деталі реалізації обох моделей.

РОЗДІЛ 3. ВХІДНІ ДАНІ

Робота моделей руху та фільтра Калмана вцілому потребує великої кількості вхідних даних. Ці дані можуть надходити з різних джерел – з датчиків GPS, з інерційного вимірювального пристрою автівки та з багатьох інших. Ці дані необхідно звести до виду, який максимізує точність та мінімізує складність роботи програми.

У цьому розділі буде розглянено формат вхідних даних, формат, у який їх буде приведено для використання у фільтрі, та способи приведення.

3.1 Дані GPS

Дані з GPS надходять у форматі, зображеному на рис. 11.

systemTimestamp	LAT	LON	VDOP	HDOP	PDOP	velocity	altitude	orientation	fix_qual	sats
1517704507002782	48.17842	11.63454	1.27000	0.68000	1.44000	0.00309	507.90000	0.00000	2	12.00000
1517704508002781	48.17842	11.63454	1.27000	0.68000	1.44000	0.00514	507.90000	0.00000	2	12.00000
1517704509002776	48.17842	11.63454	1.27000	0.68000	1.44000	0.00875	507.80000	0.00000	2	12.00000
1517704510002772	48.17842	11.63454	1.27000	0.68000	1.44000	0.00257	507.70000	0.00000	2	12.00000
1517704511002776	48.17842	11.63454	1.27000	0.68000	1.44000	0.00720	507.60000	0.00000	2	12.00000
1517704512002766	48.17842	11.63454	1.27000	0.68000	1.44000	0.00566	507.40000	0.00000	2	12.00000
1517704513002764	48.17842	11.63454	1.27000	0.68000	1.44000	0.00617	507.20000	0.00000	2	12.00000
1517704514002760	48.17842	11.63454	1.27000	0.68000	1.44000	0.43213	507.10000	291.15000	2	12.00000
1517704515002755	48.17842	11.63453	1.27000	0.68000	1.44000	1.33447	507.10000	290.29000	2	12.00000
1517704516002753	48.17843	11.63450	1.27000	0.68000	1.44000	2.58251	507.10000	291.38000	2	12.00000
1517704517002750	48.17844	11.63446	1.27000	0.68000	1.44000	2.76205	507.10000	290.72000	2	12.00000
1517704518002746	48.17844	11.63443	1.27000	0.68000	1.44000	2.33866	507.20000	290.54000	2	12.00000
1517704519002744	48.17845	11.63441	1.41000	0.79000	1.62000	1.93637	507.20000	301.18000	2	12.00000
1517704520002742	48.17846	11.63439	1.41000	0.78000	1.61000	1.61896	507.50000	311.38000	2	12.00000
1517704521002737	48.17847	11.63438	1.27000	0.68000	1.44000	1.18734	507.70000	328.32000	2	12.00000
1517704522002734	48.17848	11.63438	1.27000	0.68000	1.44000	0.90079	507.80000	338.40000	2	12.00000
1517704523002731	48.17849	11.63438	1.27000	0.68000	1.44000	0.90902	507.90000	353.27000	2	12.00000

Рисунок 11 – приклад даних з GPS.

Тлумачення стовпчиків у даних:

- SystemTimestamp: час надходження даних (μc);
- LAT: Широта (град.);
- LON: Довгота (град.);
- VDOP, HDOP, PDOP – значення, що характеризують точність даних;
- velocity – швидкість (м/с);

– altitude – висота;
– orientation – орієнтація (град.);
– fix_qual – значення, що характеризує точність даних;
– sats – кількість супутників GPS, за допомогою яких отримано дані.

Дані надходять у стандартній системі GPS – WGS84. Зберігати у векторі стану позицію в цій системі незручно, адже моделі руху описують зміну позиції у метрах відносно якоїсь точки відліку. Щоітерації переводити значення стану у широту та довготу і навпаки є джерелояк затримок у роботі, так і неточності. Натомість, у векторі стану позиція зберігатиметься у локальній системі ENU (East-North-Up). Перевод позиції з WGS84 у ENU складається з двох кроків – спочатку, WGS84 переводиться у систему ECEF (Earth-Centered, Earth-Fixed), а з неї – у ENU. Детальніше ці системи і процес переводу буде розглянуто в наступних пунктах.

Варто зауважити, що орієнтація надходить як кут між північчю і вектором напрямку у бік сходу (за годинниковою стрілкою).

3.1.1 Система WGS84

Саме у цій системі координат за замовчуванням приходять дані з датчиків GPS. Центр координат у цій системі – центр мас Землі, визначений с точністю до 2 см. Земля в ній моделюється як сфероїд з екваторіальним радіусом у 6378137 м на екваторі та фактором приплюснутості 1/298.257223563 (рис. 12) [9].

Місцезнаходження об’єкту у цій системі представляється як трійка з широти, довготи та висоти – (ϕ, λ, h).

3.1.2 Система ECEF

Система координат ECEF (Earth-Centered, Earth-Fixed) – ще одна глобальна система координат. Вона представляє позицію об’єкта як точку $< X, Y, Z >$ у системі Декартових координат з центром у центрі мас Землі.

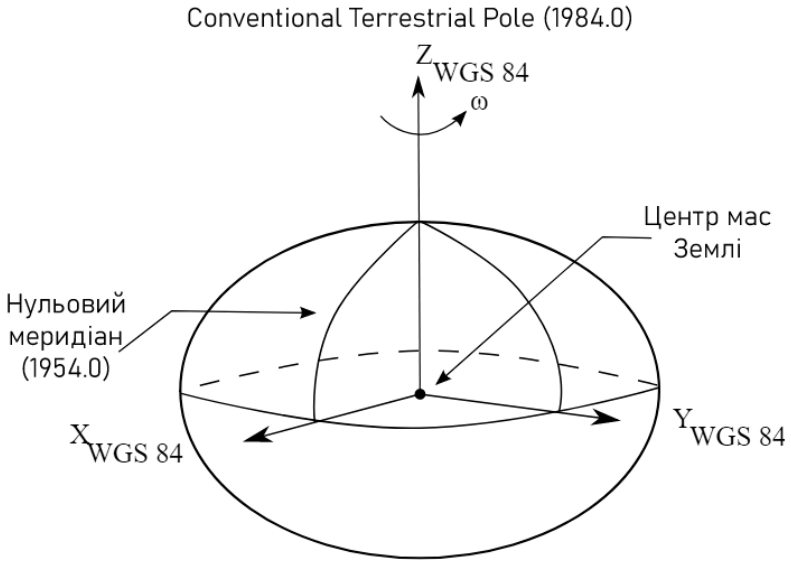


Рисунок 12 – система координат WGS84 [8]

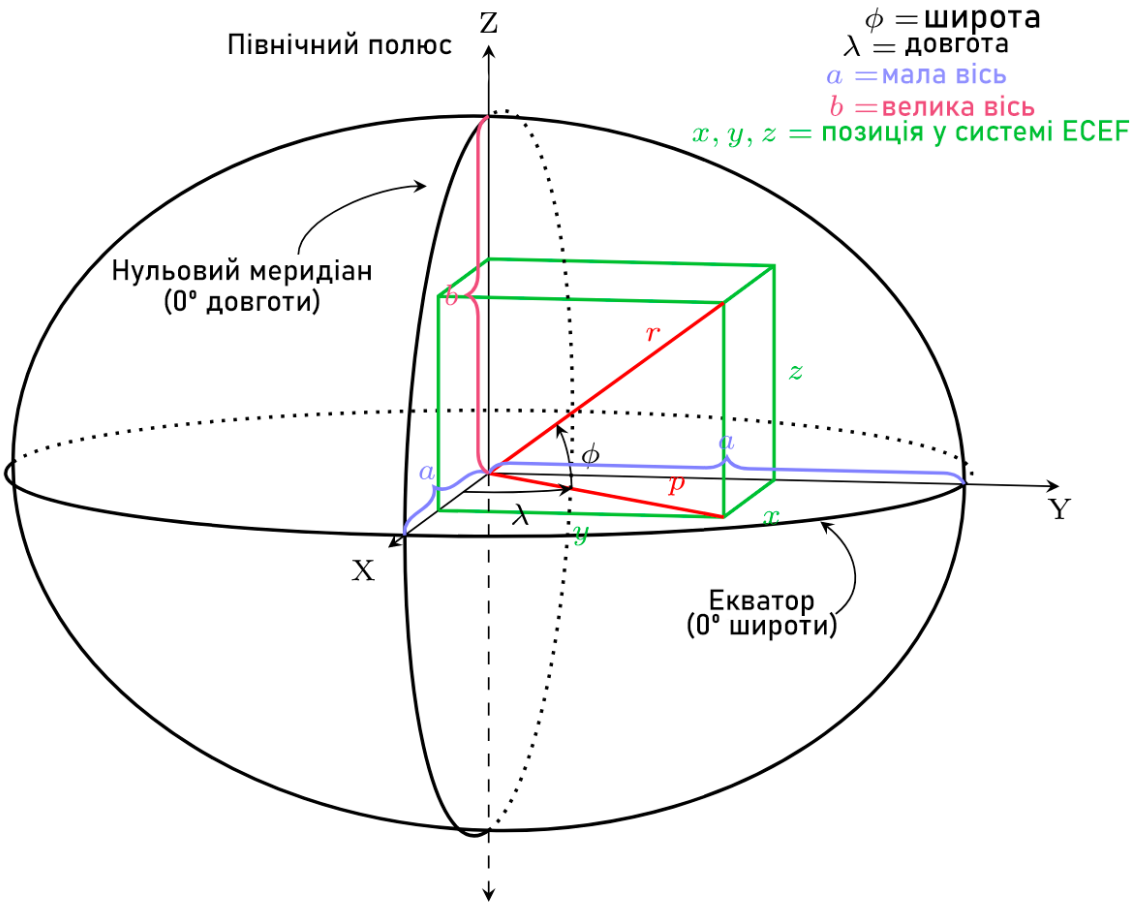


Рисунок 13 – схема системи координат ECEF

3.1.3 Система ENU

Позиція у системі ENU (East, North, Up) є трійкою (X, Y, Z) – координатами об’єкту в Декартовій системі координат з початком координат у довільній точці, де вісь X спрямована з початку координат на схід, вісь Y – на північ, а вісь Z – вгору.

Ця система координат є найкращим варіант для зберігання місцезнаходження у векторі стану в контексті цієї роботи, адже дані в форматі ENU можна використовувати як вхідні дані для моделі руху без жодних перетворень.

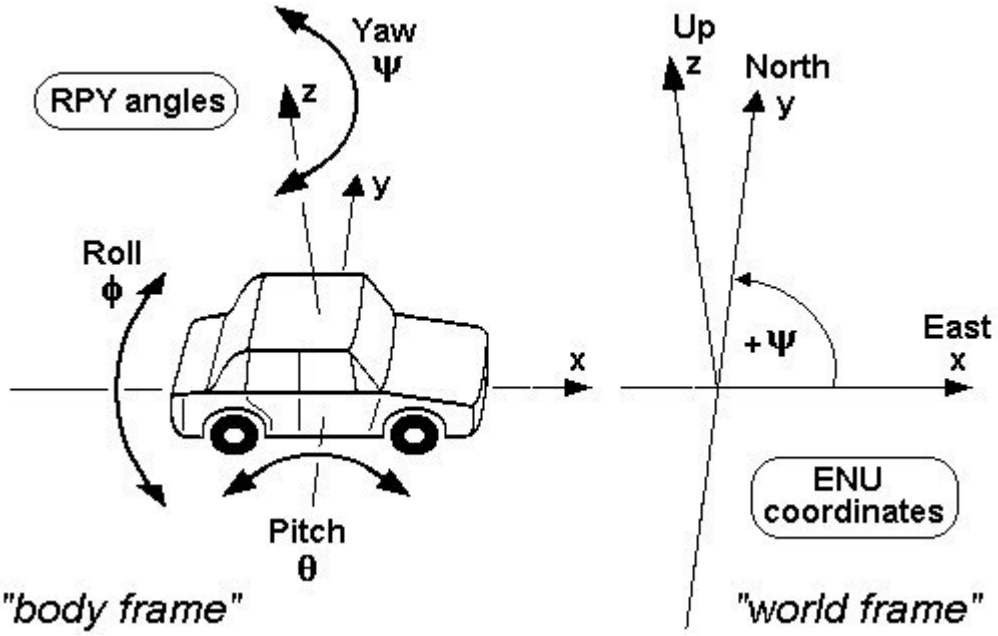


Рисунок 14 – схема системи координат ENU

3.1.4 Переведення даних між системами

Як вже було зазначено, перевод складається з двох етапів: перевод з WGS84 у ECEF та з ECEF у ENU.

Нехай задано точку A з координатами (ϕ, λ, h) у WGS84. Її координати (x, y, z) у системі ECEF визначаються за формулами:

$$x = \left(\frac{a}{\chi} + h\right)\cos(\phi)\cos(\lambda) \tag{29}$$

$$y = \left(\frac{a}{\chi} + h\right)\cos(\phi)\sin(\lambda) \tag{30}$$

$$z = \left(\frac{a * (1 - e^2)}{\chi} + h\right)\sin(\phi) \tag{31}$$

де a – мала вісь Землі;

e^2 – ексцентриситет Землі;

$\chi = \sqrt{1 - e^2 \sin^2 \chi}$.

Цю точку, у свою чергу, можна перевести у систему ENU з початком координат у точці з координатами WGS84 (ϕ_0, λ_0, h_0) та ECEF (x_0, y_0, z_0) переведенням у інший базис:

$$\begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} -\sin\lambda_0 & \cos\lambda_0 & 0 \\ -\sin\phi_0\cos\lambda_0 & -\sin\phi_0\sin\lambda_0 & \cos\phi_0 \\ \cos\phi_0\cos\lambda_0 & \cos\phi_0\sin\lambda_0 & \sin\phi_0 \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{pmatrix} \tag{32}$$

Отриманий вектор є шуканими координатами точки A в системі ENU [7].

Незважаючи на те, що всередині алгоритму всі значення обробляються і зберігаються в системі ENU, результат все одно необхідно переводити в WGS84 для використання, наприклад, в засобах для візуалізації положення або траєкторії. Таке «обернене» переведення також складається з двох етапів: переведення з ENU у ECEF і з ECEF у WGS84.

Так як перехід між ECEF та ENU є всього лише зміною базиса, обернений перехід досягається за допомогою трансформаці рівняння (32):

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\sin\lambda_0 & -\sin\phi_0\cos\lambda_0 & \cos\phi_0\cos\lambda_0 \\ \cos\lambda_0 & -\sin\phi_0\sin\lambda_0 & \cos\phi_0\sin\lambda_0 \\ 0 & \cos\phi_0 & \sin\phi_0 \end{pmatrix} \begin{pmatrix} e \\ n \\ u \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \qquad (33)$$

Переведення з ECEF у WGS84 складніше, ніж з WGS84 у ECEF, і для нього існує декілька алгоритмів. У цій роботі буде використано покращений варіант алгоритму Жу [10]:

$$w^2 = x^2 + y^2$$

$$l = e^2/2$$

$$m = w^2/a^2$$

$$n = \left((1 - e^2)\frac{z}{b}\right)^2$$

$$p = (m + n - 4l^2)/6$$

$$G = m * n * l^2$$

$$H = 2p^3 + G$$

$$C = \sqrt[3]{\frac{H + G + 2\sqrt{HG}}{2}}$$

$$i = -(2l^2 + m + n)/2$$

$$P = p^2$$

$$\beta = \frac{i}{3} - C - P/C$$

$$k = l^2(l^2 - m - n)$$

$$t = \sqrt{\sqrt{\beta^2 - k} - (\beta + i)/2} - \text{sign}(m - n)\sqrt{|(\beta - i)|/2}$$

$$F = t^4 + 2it^2 + 2l(m - n)t + k$$

$$\frac{dF}{dt} = 4t^3 + 4it + 2l(m - n)$$

$$\Delta t = \frac{-F}{dF/dt}$$

$$u = t + \Delta t + l$$

$$v = t + \Delta t - l$$

$$w = \sqrt{w^2}$$

$$\phi = \arctan(zu, wv)$$

$$\Delta w = w\left(1 - \frac{1}{u}\right)$$

$$\Delta z = z\left(1 - \frac{(1 - e^2)}{v}\right)$$

$$h = \text{sign}(u - 1)\sqrt{(\Delta w)^2 + (\Delta z)^2}$$

$$\lambda = \arctan(y, z)$$

де e – ексцентриситет Землі;

a – мала вісь Землі;

b – велика вісь Землі.

Ця формула дозволяє обчислити значення широти, довготи та висоти точки, заданої координатами в системі ECEF. Оскільки висота у задачі для автівки не має значення, кроки з обчислення Δw, Δz та h можна оминути.

3.2 Дані інерційного вимірювального пристрою

Друге джерело даних у автівці – інерційний вимірювальний пристрій (Inertial Measurement Unit, IMU; далі – ІВП). Це – пристрій, що включає в себе акселерометри та гіроскопи, і вимірює сили, що діють на тіло та його прискорення. Дані надходять у наступному вигляді:

	systemTimestamp	accelTimestamp	accelX	accelY	accelZ	gyroTimestamp	gyroX	gyroY	gyroZ
0	1517704506343448	1517704506343448	-9.58235	-0.37674	0.01076	1517704506343448	0.03514	-0.07666	-0.02768
1	1517704506343522	1517704506343522	-9.66009	-0.43654	0.02512	1517704506343522	0.03088	-0.07879	-0.02236
2	1517704506343550	1517704506343550	-9.62062	-0.30498	0.09688	1517704506343550	0.03514	-0.06814	-0.03301
3	1517704506343575	1517704506343575	-9.50461	-0.40784	0.02512	1517704506343575	0.02981	-0.08411	-0.02023
4	1517704506343600	1517704506343600	-9.67684	-0.27388	0.10644	1517704506343600	0.03514	-0.06708	-0.03514
5	1517704506343624	1517704506343624	-9.56202	-0.52624	-0.00957	1517704506343624	0.02981	-0.08411	-0.01597
6	1517704506343649	1517704506343649	-9.64813	-0.33249	0.09568	1517704506343649	0.03514	-0.06921	-0.03514
7	1517704506343675	1517704506343675	-9.53332	-0.39707	0.08611	1517704506343675	0.03301	-0.07773	-0.02236

Рисунок 15 – дані ІВП

Тлумачення стовпчиків:

- systemTimestamp – системний час у момент надходження запису, μс;
- accelTimestamp – системний час у момент зняття показників акселерометрів, μс;
- accelX, accelY, accelZ – прискорення за осями X, Y та Z відповідно, що розташовані, як показано на рис. 13, м^2/с;
- gyroTimestamp – системний час у момент зняття показників гіроскопу, μс;
- gyroX, gyroY, gyroZ – швидкість повороту навколо осей X, Y та Z, рад/с;

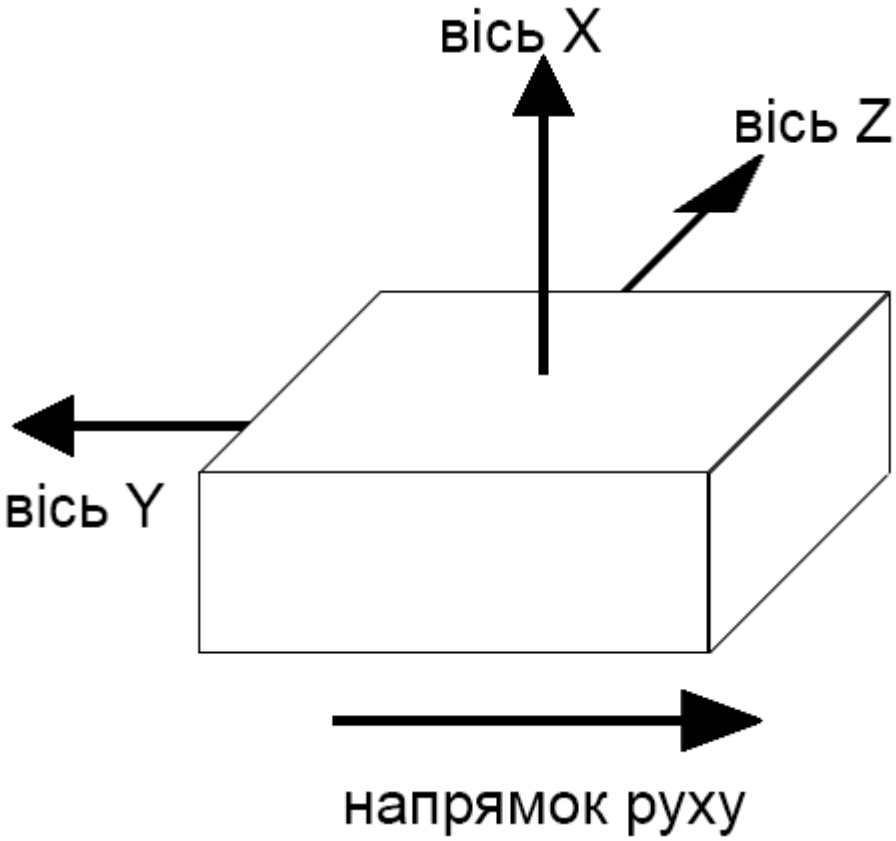
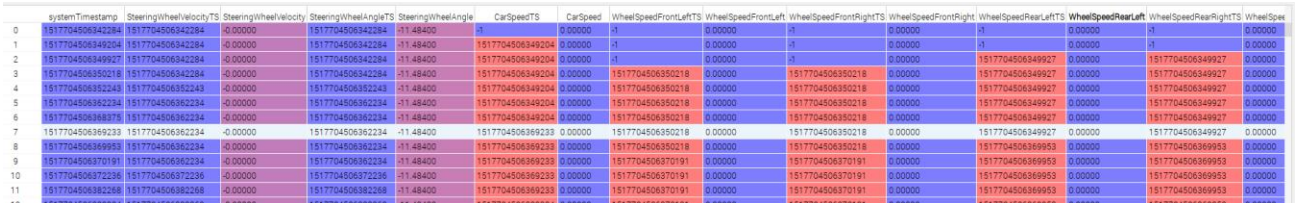


Рисунок 16 – розміщення осей ІВП

Варто зазначити, що швидкість повороту, як і орієнтація з датчику GPS, додатня за годинниковою стрілкою (з півночі на схід).

3.3 Дані інших датчиків

Для виконання завдання також доступні дані великої кількості датчиків щодо стану автівки, як-то стан фар, сигналів повороту, швидкість коліс, кут повороту руля і так далі. Їх занадто багато, щоб описати всі у цій роботі, тож буде описано лише ті, що надають інформацію, потрбіну для роботи моделі.



	systemTimestamp	SteeringWheelVelocityTS	SteeringWheelVelocity	SteeringWheelAngleTS	SteeringWheelAngle	CarSpeedTS	CarSpeed	WheelSpeedFrontLeftTS	WheelSpeedFrontLeft	WheelSpeedFrontRightTS	WheelSpeedFrontRight	WheelSpeedRearLeftTS	WheelSpeedRearLeft	WheelSpeedRearRightTS	WheelSpee
0	1517704506342284	1517704506342284	-0.00000	1517704506342284	-11.48400	-1	0.00000	-1	0.00000	-1	0.00000	-1	0.00000	-1	0.00000
1	1517704506349204	1517704506342284	-0.00000	1517704506342284	-11.48400	1517704506349204	0.00000	-1	0.00000	-1	0.00000	-1	0.00000	-1	0.00000
2	1517704506349927	1517704506342284	-0.00000	1517704506342284	-11.48400	1517704506349204	0.00000	-1	0.00000	-1	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
3	1517704506350218	1517704506342284	-0.00000	1517704506342284	-11.48400	1517704506349204	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
4	1517704506352243	1517704506352243	-0.00000	1517704506352243	-11.48400	1517704506349204	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
5	1517704506362234	1517704506362234	-0.00000	1517704506362234	-11.48400	1517704506349204	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
6	1517704506368375	1517704506362234	-0.00000	1517704506362234	-11.48400	1517704506349204	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
7	1517704506369233	1517704506362234	-0.00000	1517704506362234	-11.48400	1517704506369233	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506349927	0.00000	1517704506349927	0.00000
8	1517704506369953	1517704506362234	-0.00000	1517704506362234	-11.48400	1517704506369233	0.00000	1517704506350218	0.00000	1517704506350218	0.00000	1517704506369953	0.00000	1517704506369953	0.00000
9	1517704506370191	1517704506362234	-0.00000	1517704506362234	-11.48400	1517704506369233	0.00000	1517704506370191	0.00000	1517704506370191	0.00000	1517704506369953	0.00000	1517704506369953	0.00000
10	1517704506372236	1517704506372236	-0.00000	1517704506372236	-11.48400	1517704506369233	0.00000	1517704506370191	0.00000	1517704506370191	0.00000	1517704506369953	0.00000	1517704506369953	0.00000
11	1517704506382268	1517704506382268	-0.00000	1517704506382268	-11.48400	1517704506369233	0.00000	1517704506370191	0.00000	1517704506370191	0.00000	1517704506369953	0.00000	1517704506369953	0.00000

Рисунок 17 – приклад даних інших датчиків

Інформація з цих датчиків представлена у стовпчиках wheelSpeedRearRight та wheelSpeedRearLeft. Вони відображають швидкість обертання правого та лівого задніх коліс відповідно. Ці дані є найбільш надійним джерелом інформації про швидкість автівки, що її можливо отримати «зсередини». Для обчислення швидкості автівки використовується формула:

$$v = \frac{v_l + v_r}{2} * R \tag{34}$$

де v_l – швидкість обертання лівого заднього колеса;

v_r – швидкість обертання правого заднього колеса;

R – радіус колеса автівки; $R = 0.376$ м.

РОЗДІЛ 4. Реалізація

У цьому розділі буде описано реалізацію описаного алгоритму та описаних моделей. У підрозділі 4.1 буде зроблено огляд використаних технологій та наведено аргументи щодо їх вибору. У підрозділах з 4.2 по 4.4 описано реалізацію моделей руху, беззапахового перетворення та беззапахового фільтру Калмана[1].

4.1 Огляд використаних технологій

Предметна область програми, що розроблюється, накладає на неї деякі обмеження. У першу чергу, це обмеження у швидкості роботи, адже це програмне забезпечення спрямоване на те, щоб у реальному часі визначати положення автівки. Це обмеження значно зменшує простір мов програмування, що підходять для розробки програми.

Було розглянуто два основні варіанти – C++ та Python. Перевагою C++ над Python і багатьма іншими мовами високого рівня є висока швидкодія, але при цьому ця мова є важчою в опануванні, що може як ускладнити процес розробки, так і спричинити появу критичних багів. Python, навпаки, є простою мовою для опанування, але з низькою швидкодією, особливо у праці з масивами. На щастя, це компенсується наявністю відомих математичних бібліотек NumPy та SciPy, де операції, що потребують великої кількості обчислень, обробляються кодом на мовах C та C++. Їх використання мінімізує втрати у швидкодії відносно інших мов.

Враховуючи наведені вище аргументи, для реалізації було обрано мову Python та бібліотеки NumPy і SciPy [11].

[1] Реалізацію решти необхідних функцій – операцій з перетворення GPS, генерації результатів – наведено в додатках А та Б.

4.2 Реалізація моделей руху

Моделі руху реалізовано за допомогою двох функцій: transit_ctrv та transit_ctra, що відповідають функціям переходу моделей CTRV та CTRA. Параметрами обох функцій є поточний стан, керування та час Δt.

```
def transit_ctrv(state: np.ndarray, u: np.ndarray, delta_t: float) ->
np.ndarray:
    """State transition function for CTRV model.


    :param state: current state [x, y, sin(angle), cos(angle)];
    :param u: control vector [velocity, yaw rate];
    :param delta_t: delta time
    :returns: new state [x, y, sin(angle), cos(angle)]
    """
    v = u[0]
    w = -u[1]
    new_state = np.empty_like(state)
    new_state[:] = state
    if abs(w) < 1e-6:
        new_state += [
            v * state[3] * delta_t,
            v * state[2] * delta_t,
            0,
            0
        ]
    else:
        vdivw = v / w
        new_angle = np.arctan2(state[2], state[3]) + w * delta_t
        sin_new_angle = np.sin(new_angle)
        cos_new_angle = np.cos(new_angle)
        new_state += [
            vdivw * (sin_new_angle - state[2]),
            -vdivw * (cos_new_angle - state[3]),
            0,
            0
        ]
        new_state[2] = sin_new_angle
        new_state[3] = cos_new_angle
    return new_state


def transit_ctra(state: np.ndarray, u: np.ndarray, delta_t: float) ->
np.ndarray:
    """State transition function for CTRA model.
    :param state: current state [x, y, v, sin(angle), cos(angle)];
    :param u: control vector [acceleration, yaw rate];
```

```
    :param delta_t: delta time
    :returns: new state [x, y, v, sin(angle), cos(angle)]
    """
    a = u[0]
    w = -u[1]
    new_state = np.empty_like(state)
    new_state[:] = state
    new_angle = np.arctan2(state[2], state[3]) + w * delta_t
    sin_new_angle = np.sin(new_angle)
    cos_new_angle = np.cos(new_angle)
    if abs(w) < 1e-6:
        new_state += [
            ((state[2]+a*w*delta_t)*sin_new_angle + a*cos_new_angle -
state[2]*w*state[3] - a*state[4]),
            (-(state[2]+a*w*delta_t)*cos_new_angle + a*sin_new_angle +
state[2]*w*state[4] - a*state[3]),
            a*delta_t,
            0,
            0
        ]
    else:
        new_state += [
            (1/(w**2)) * (w*(state[2]+a*delta_t)*sin_new_angle +
a*cos_new_angle - state[2]*w*state[3] - a*state[4]),
            (1/(w**2)) * (-w*(state[2]+a*delta_t)*cos_new_angle +
a*sin_new_angle + state[2]*w*state[4] - a*state[3]),
            a*delta_t,
            0,
            0
        ]
    new_state[3] = sin_new_angle
    new_state[4] = cos_new_angle
    return new_state
```

4.3 Реалізація беззапахового перетворення

Алгоритм беззапахового перетворення було разбито на три функції: функція обчислення параметру λ, функція обчислення ваг та функція обчислення сігма-точок. Код цих функцій наведено нижче.

```
def calculate_lambda(L: int, alpha: float = 1, k: float = 0):
    return (alpha ** 2) * (L + k) - L

def calc_weights(alpha: float, beta: float, L: int, _lambda: float):
    w_m = np.full(shape=(2 * L + 1), fill_value=1 / (2 * (L + _lambda)))
    w_m[0] = _lambda / (L + _lambda)
    w_c = np.empty_like(w_m)
```

```
    w_c[:] = w_m
    w_c[0] += 1 - alpha ** 2 + beta
    return w_m, w_c

def calc_sigma_points(x_mean: np.array, x_cov: np.array, _lambda: float):
    dim: int = x_mean.shape[0]
    matrix = (dim + _lambda) * x_cov
    eigval, eigvec = np.linalg.eig(matrix)
    if len(eigval[eigval < 0]) > 0:
        eigval[eigval <= 0] = 1e-4
        matrix = eigvec.dot(eigval *
np.identity(dim)).dot(np.linalg.inv(eigvec))
    sq_rt_matrix = cholesky(matrix)
    sigma_vectors = np.full(shape=(2 * dim + 1, dim),
fill_value=x_mean.astype(float))
    sigma_vectors[1:(dim + 1)] += sq_rt_matrix[0:dim]
    sigma_vectors[(dim + 1):] -= sq_rt_matrix[0:]
    return sigma_vectors
```

Рядки з 4 по 6 включно функції calc_sigma_points потребують пояснення. Одним з кроків алгоритму БЗП є взяття квадратного кореня від матриці, отриманої із матриці коваріації початкового розподілу. Необхідною умовою для цього є невід'ємновизначність цієї матриці. В теорії, це не є проблемою, оскільки, як відомо, матриця коваріації завжди є невід'ємновизначеною, але на практиці це може не виконуватися через, по-перше, неможливість точно задати матрицю коваріації на початку роботи алгоритму, а по-друге, через неточности, спричинені обмеженнями мови програмування на точність чисел з рухомою комою, а також похибками числових методів, що використовуються у програмуванні. Рішення цієї проблеми походить з визначення невід'ємновизначеної матриці – матриця розкладається на власні числа та вектори та від'ємні власні числа замінюються на маленькі додатні значення (тут – 10^{-9}), після чого матриця «збирається» знову множенням нового вектору власних чисел на матрицю власних векторів. Отримана матриця гарантовано задовільняє передумови взяття квадратного кореня.

4.4 Реалізація беззапахового фільтру Калмана

Фільтр Калмана було винесено в окремий клас UnscentedKF. Значення, що не змінюються під час роботи алгоритму – наприклад, параметри для перетворення та обчислені ваги – реалізовано як поля класу. Клас містить, серед інших, два методи predict та update, що відповідають етапам передбачення та корекції у алгоритмі.

```
class UnscentedKF:
    def __init__(self, f, h, R, Q, L, alpha, beta, kappa):
        self.state_trans_func = f  # state transition function g(u_t,
mu_(t-1))
        self.obs_func = h  # observation function h(prior_mu_t)
        self.R = R  # state transition uncertainty covariance matrix
(process noise cov)
        self.Q = Q  # measurement error covariance matrix (msmt noise cov)
        self.L = L  # dimension
        self.alpha = alpha
        self.beta = beta
        self.kappa = kappa
        self._lambda = ut.calculate_lambda(L=self.L, alpha=self.alpha,
k=self.kappa)
        wm, wc = ut.calc_weights(alpha=alpha, beta=beta, L=L,
_lambda=self._lambda)
        self.wm = wm
        self.wc = wc

    def predict(self, prev_mean, prev_cov, u, trans_f, delta_t=1):
        # Calculate sigma points for state
        sigma_points_state = ut.calc_sigma_points(x_mean=prev_mean,
x_cov=prev_cov, _lambda=self._lambda)
        # Propagate through state transition function
        sigma_points_state_propagated = []
        for sp in sigma_points_state:
            sigma_points_state_propagated.append(trans_f(sp, u, delta_t))
        sigma_points_state_propagated =
np.array(sigma_points_state_propagated)
        # Predict state mean and covariance
        predict_state_mean = np.dot(self.wm, sigma_points_state_propagated)
        dif_state = sigma_points_state_propagated - predict_state_mean
        predict_state_cov = (self.wc * dif_state.T).dot(dif_state) + self.R
        return predict_state_mean, predict_state_cov, dif_state,
sigma_points_state_propagated

    def update(self, predict_mean, predict_cov, dif_state, z,
measurement_f, sp_propagated):
        sigma_points_obs_propagated = measurement_f(sp_propagated)
        # Calculate mean and covariance of observation
```

```
        obs_mean = np.dot(self.wm, sigma_points_obs_propagated)
        dif_obs = sigma_points_obs_propagated - obs_mean
        obs_cov = (self.wc * dif_obs.T).dot(dif_obs) + self.Q
        # Cross-covariance between predicted state and predicted
observation
        cross_cov = (self.wc * dif_state.T).dot(dif_obs)
        KalmanGain = cross_cov.dot(np.linalg.inv(obs_cov))
        state_mean_post = predict_mean + KalmanGain.dot((z - obs_mean).T)
        state_cov_post = predict_cov -
KalmanGain.dot(obs_cov).dot(KalmanGain.T)
        return state_mean_post, state_cov_post

    def propagate(self, mean: np.ndarray, cov: np.ndarray, u: np.ndarray,
trans_f, z: np.ndarray,
                  measurement_f, delta_t_u=1):
        predicted_next_mean, predicted_next_cov, dif, sp_propagated =
self.predict(mean, cov, u, trans_f, delta_t_u)
        return self.update(predicted_next_mean, predicted_next_cov, dif, z,
measurement_f, sp_propagated)
```

ВИСНОВКИ

В результаті роботи було:

- Досліджено теоретичні засади оцінювання стану.
- Досліджено теоретичні засади беззапахового фільтру Калмана – від Байєсівського фільтру, до лінійного алгоритму, до беззапахового перетворення.
- Досліджено фізичні моделі руху автівки. Виявлено, що моделі прямолінійного руху – CA та CV – є занадто простими для того, щоб досягнути достатньої точності, у той час як моделі CA та CSAV – занадто складні у реалізації при незначній перевазі у точності порівняно з моделями CTRV та CTRA.
- Проаналізовано та систематизовано вхідні дані з декількох типів сенсорів. Обрано систему координат, в якій зберігатимуться дані GPS.
- Розроблено програмний засіб, що реалізує алгоритм.
- Протестовано ПЗ на реальних даних.

Фільтр протестовано на реальних даних заїзду тестової автівки Мюнхеном. Заїзд включав як відкриті ділянки з хорошим сигналом GPS, так і ділянки із щільною забудовою, що перешкоджає сигналу, та проїзд у тонелі, де сигнал GPS відсутній.

На рисунку 18 порівняно результат роботи фільтру з даними «чистого» GPS. На відкритих ділянках траєкторії відрізняються в рамках одного метру. Різницю краще видно при порівнянні траєкторії на ділянці з тонелем (рис. 19). Як видно, фільтр справляється з поставленою задачею – визначення місцезнаходження автівки в умовах поганого чи відсутнього сигналу GPS.

Рисунок 18 – траєкторія за даними GPS (зліва) та за фільтром з моделлю CTRV (справа)

Рисунок 19 – Траєкторія під час в’їзду у тонель. Траєкторія, отримана з фільтру (справа) є набагато плавнішою і набагато ближча до справжньої, ніж отримана з GPS (зліва)

Для повноцінного впровадження у галузях систем допомоги водіям, безпілотних автівок та інших, програмний засіб потребує подальшої роботи. Серед можливих допрацювань є:

– Інтегрований інструментарій візуалізації результатів;
– Впровадження більш складної, але більш точної моделі руху (CCA, CSAV);

- Впровадження алгоритму перевірки точності сигналу GPS за кількістю доступних супутників, значенням HDOP, тощо [12];
- Проведення тестування у складніших умовах (гірська місцевість, щільна багатоповерхова забудова);
- Реалізація та впровадження більш швидкого алгоритму перетворення даних GPS з WGS84 у ENU, що не потребують частого обрахунку тригонометричних функцій [13];
- Впровадження варіанту беззапахового фільтру Калмана, відомого як Square-Root Unscented Kalman Filter (Квадратнокореневий беззапаховий фільтр Калмана). Цей алгоритм має більшу швидкодію, ніж звичайний БФК, оскільки замість матриці коваріації $\sum$ передає зі стану у стан її квадратний корінь, позбуваючись необхідності обчислення квадратного кореня від $\sum$ щоітерації [14].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives] / M. S. Grewal, A. P. Andrews – DOI 10.1109/MCS.2010.936465.
2. Probabilistic Robotics / S. Thrun, W. Burgard, D. Fox. – [б.м.], The MIT Press, 2005. – 672 с. – ISBN 978-0262201629.
3. Nonlinear filtering: Interacting particle resolution / Pierre Del Moral // Comptes Rendus de l'Académie des Sciences, 1997. – Series I Mathematics, Volume 325, Issue 6 – с. 653-658. – ISSN 0764-4442.
4. "The unscented Kalman filter for nonlinear estimation" / E. A. Wan, R. Van Der Merwe // IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, 2000. – с. 153-158. – DOI 10.1109/ASSPCC.2000.882463.
5. "Comparison and evaluation of advanced motion models for vehicle tracking" / R. Schubert, E. Richter, G. Wanielik // 11th International Conference on Information Fusion, 2008. – с. 1-6.
6. "Unscented Kalman filter design for curvilinear motion models suitable for automotive safety applications" / M. Tsogas, A. Polychronopoulos, A. Amditis // 7th International Conference on Information Fusion, 2005. – с. 8. – DOI: 10.1109/ICIF.2005.1592006.
7. Transformations between ECEF and ENU coordinates [Електронний ресурс] // European Space Agency – Режим доступу до ресурсу: https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates.
8. Defense Mapping Agency Technical Report: World Geodetic System: Its Definition And Relationships With Local Geodetic Systems [Електронний ресурс] // National Imagery & Mapping Agency. – 1991. – Режим доступу до ресурсу: https://apps.dtic.mil/dtic/tr/fulltext/u2/a280358.pdf.

9. World Geodetic System 1984 [Електронний ресурс] – Режим доступу до ресурсу: https://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf
10. Accurate Conversion of Earth-Fixed Earth-Centered Coordinates to Geodetic Coordinates / Karl Osen – Trondheim, Norwegian University of Science and Technology, 2017.
11. Бібліотека NumPy [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/
12. Dilution of Precision / R. Langley – [б.м., б.в.], 1999.
13. Converting GPS Coordinates (φλh) to Navigation Coordinates (ENU) / S. Drake – [б.м.], DSTO Electronics and Surveillance Research Laboratory, 2002.
14. The square-root unscented Kalman filter for state and parameter-estimation / R. Van der Merwe, E. A. Wan // IEEE International Conference on Acoustics, Speech, and Signal Processing, 2001. – Vol. 6. – с. 3461-3464. – DOI 10.1109/ICASSP.2001.940586.

ДОДАТОК А

Код операцій з перетворення між системами координат GPS

```
def wgs_to_enu(lat: float, lon: float, alt: float, ecef0: np.ndarray,
ref_matrix: np.ndarray, ) -> np.ndarray((3,)):
    """Convert WGS84 coordinates of a point to ENU coordinates with a
specified point of reference (0, 0, 0).

    :param lat: Latitude of point IN RADIANS
    :param lon: Longitude of point IN RADIANS
    :param alt: Altitude of point (NOT IN RADIANS XDD)))00))0)
    :param ecef0: Reference point in ECEF coordinates (easy to
precalculate)
    :param ref_matrix: Matrix for ECEF->ENU transformation (easy to
precalculate)
    :return: ENU coordinates of point in an ndarray [x,y,z]
    :rtype: np.ndarray
    """
    sp = np.sin(lat)
    cp = np.cos(lat)
    sl = np.sin(lon)
    cl = np.cos(lon)
    ecef1 = wgs_to_ecef(alt, sp, cp, sl, cl)
    return ecef_to_enu(ecef0, ecef1, ref_matrix)

def wgs_to_ecef_raw(phi: float, lam: float, h: float) -> np.ndarray((3,)):
    """Convert WGS84 coordinates to ECEF coordinates.

    :param phi: Latitude (IN RADIANS)
    :param lam: Longitude (IN RADIANS)
    :param h: height
    :return: ECEF coordinates in a ndarray [X, Y, Z]
    :rtype: np.ndarray
    """
    sp = np.sin(phi)
    cp = np.cos(phi)
    sl = np.sin(lam)
    cl = np.cos(lam)
    N = a / (np.sqrt(1 - e2 * (sp ** 2)))
    temp = (N + h) * cp
    x = temp * cl
    y = temp * sl
    z = (((b / a) ** 2) * N + h) * sp
    return np.array([x, y, z])

def wgs_to_ecef(h: float, sp: float, cp: float, sl: float, cl: float) ->
np.ndarray((3,)):
    """Convert WGS84 coordinates to ECEF coordinates. DON'T FORGET TO
CONVERT TO RADIANS!

    :param h: height
```

```
    :param sp: sin latitude
    :param cp: cos latitude
    :param sl: sin longitude
    :param cl: cos longitude
    :return: ECEF coordinates in a ndarray: [X, Y, Z]
    :rtype: np.ndarray
    """
    N = a / (np.sqrt(1 - e2 * (sp ** 2)))
    temp = (N + h) * cp
    x = temp * cl
    y = temp * sl
    z = (((b / a) ** 2) * N + h) * sp
    return np.array([x, y, z])

# Source: ESA
def ecef_to_enu(ecef0: np.ndarray, ecef1: np.ndarray, transform_matrix:
np.ndarray) -> np.ndarray((3,)):
    enu = transform_matrix @ (ecef1 - ecef0)
    return enu

# Source: ESA
def enu_to_ecef(ecef0: np.ndarray, enu: np.ndarray, transform_matrix:
np.ndarray) -> np.ndarray((3,)):
    ecef = transform_matrix @ enu + ecef0
    return ecef

# Source: accurate_conversion...
def ecef_to_wgs(x: float, y: float, z: float = 0) -> np.ndarray((3,)):
    w2 = x ** 2 + y ** 2
    l = e2 / 2
    l2 = l ** 2
    m = w2 / (a ** 2)
    n = ((1 - e2) * z / b) ** 2
    p = (m + n - 4 * l2) / 6
    G = m * n * l2
    H = 2 * p ** 3 + G
    C = np.cbrt(((H + G + 2 * np.sqrt(H * G)) / 2))
    i = -(2 * l2 + m + n) / 2
    P = p ** 2
    beta = i / 3 - C - P / C
    k = l2 * (l2 - m - n)
    t = np.sqrt(np.sqrt(beta ** 2 - k) - (beta + i) / 2) - np.sign(m - n) *
np.sqrt(abs((beta - i) / 2))
    F = t ** 4 + 2 * i * t ** 2 + 2 * l * (m - n) * t + k
    dFdt = 4 * t ** 3 + 4 * i * t + 2 * l * (m - n)
    deltat = -F / dFdt
    u = t + deltat + l
    v = t + deltat - l
    w = np.sqrt(w2)
    phi = np.arctan2(z * u, w * v)
    deltaw = w * (1 - 1 / u)
    deltaz = z * (1 - (1 - e2) / v)
    h = np.sign(u - 1) * np.sqrt(deltaw ** 2 + deltaz ** 2)
    lam = np.arctan2(y, x)
```

```
    return np.array([phi, lam, h])

def enu_to_wgs(ecef0: np.ndarray, enu: np.ndarray, transform_matrix:
np.ndarray) -> np.ndarray((3,)):
    ecef = enu_to_ecef(ecef0, enu, transform_matrix)
    return ecef_to_wgs(ecef[0], ecef[1], ecef[2])
```

ДОДАТОК Б

Код, що використовується для генерування результатів

```
# 0: Timestamp, 1: Lat, 2: Lon, 3: Pdop, 4: velocity, 5: altitude, 6:
orientation,
# 7: accelX, 8: accelY, 9: accelZ, 10:gyroX, 11: wsrr, 12: wsrl
def run_CTRV(inputs: np.ndarray):
    states = []
    current_state = np.zeros(5)
    current_cov = np.identity(5) * 1e-9
    last_update = 0.
    reference_ecef = np.zeros(3)
    reference_matrix = np.zeros((3, 3))
    reference_matrix_T = np.zeros((3, 3))
    initialized = False
    current_control = 0.
    current_measurement = np.zeros(5)
    ukf = ukfilter.UnscentedKF(process_speed_ctra, measmt_func_spd,
np.identity(5) * 1e-9, np.identity(5) * 1e-4, 5,
                               alpha=1, beta=0, kappa=3)
    for input in inputs:
        # if sensor (speed) data is available
        if not (np.isnan(input[11]) or np.isnan(input[12])):
            obs_speed = ((input[11] + input[12]) / 2) * wheel_r
            # Predict with last velocity and yaw rate, update with last gps
            current_state, current_cov = ukf.propagate(current_state,
current_cov,
                                                       current_control,
ctrv.transit_ctrv,
                                                       obs_speed,
ctrv.measmt_func_speed,
                                                       (input[0] -
last_update) * 1e-6)
            last_update = input[0]
            state_gps = np.degrees(ctrv.state_to_latlon(current_state,
reference_ecef, reference_matrix_T))
            states.append(state_gps)
        # if IMU (yaw rate) data is available
        if not np.isnan(input[10]):
            current_control = input[10] - gyro_static
        # if GPS data is available
        if not np.isnan(input[2]):
            # Transform GPS to ENU coordinates
            gps_radians = np.radians(input[1:3])
            gps_enu = wgs_to_enu(lat=gps_radians[0], lon=gps_radians[1],
alt=0,
                                 ecef0=reference_ecef,
ref_matrix=reference_matrix)
            current_measurement[:2] = gps_enu[:2]
            current_measurement[2] = input[4]
            gps_orient = np.radians(90-input[6])
```

```
            current_measurement[3] = np.sin(gps_orient)
            current_measurement[4] = np.cos(gps_orient)
            # Setup initial state and ENU reference
            if not initialized:
                reference_ecef, reference_matrix =
get_enu_reference(gps_radians[0], gps_radians[1])
                reference_matrix_T = reference_matrix.T
                current_state = current_measurement
                last_update = input[0]
                initialized = True
            # Predict with last velocity and yaw rate, update with last gps
            current_state, current_cov = ukf.propagate(current_state,
current_cov,
                                                       current_control,
ctrv.transit_ctrv,
                                                       current_measurement,
ctrv.measmt_func_gps,
                                                       (input[0] -
last_update) * 1e-6)
            last_update = input[0]
            state_gps = np.degrees(ctrv.state_to_latlon(current_state,
reference_ecef, reference_matrix_T))
            states.append(state_gps)
    return states


# 0: Timestamp, 1: Lat, 2: Lon, 3: Pdop, 4: velocity, 5: altitude, 6:
orientation,
# 7: accelX, 8: accelY, 9: accelZ, 10:gyroX, 11: wsrr, 12: wsrl, 13: car
speed
def run_CTRA(inputs: np.ndarray):
    states = []
    current_state = np.zeros(5)
    current_cov = np.identity(5) * 1e-9
    last_timestamp = 0.
    reference_ecef = np.zeros(3)
    reference_matrix = np.zeros((3, 3))
    reference_matrix_T = np.zeros((3, 3))
    initialized = False
    # Control vector: [velocity, yaw speed]
    current_control = np.zeros(2)
    current_measurement = np.zeros(5)
    ukf = ukfilter.UnscentedKF(process_speed_ctra, measmt_func_spd,
np.identity(5) * 1e-9, np.identity(5) * 1e-9, 5,
                               alpha=np.sqrt(3), beta=2, kappa=1)
    for input in inputs:
        # if sensor (speed) data is available
        if initialized and not (np.isnan(input[11]) or
np.isnan(input[12])):
            current_measurement[2] = input[13]
            # Predict with last velocity and yaw rate, update with last gps
            current_state, current_cov = ukf.propagate(current_state,
current_cov,
                                                       current_control,
```

```
ctra.transit_ctra,
                                                        current_measurement,
ctra.measmt_func_sensor,
                                                        (input[0] -
last_timestamp) * 1e-6)
            last_timestamp = input[0]
            states.append(np.degrees(ctra.state_to_latlon(current_state,
reference_ecef, reference_matrix_T)))
        # if IMU (yaw rate) data is available
        if not np.isnan(input[10]):
            current_control[0] = current_state[2] * input[8] +
current_state[3] * input[9]
            current_control[1] = input[10]
        # if GPS data is available
        if not np.isnan(input[2]):
            # Transform GPS to ENU coordinates
            gps_radians = np.radians(input[1:3])
            # Setup initial state and ENU reference
            if not initialized:
                reference_ecef, reference_matrix =
get_enu_reference(gps_radians[0], gps_radians[1])
                reference_matrix_T = reference_matrix.T
                current_state = current_measurement
                last_timestamp = input[0]
                initialized = True
            gps_enu = wgs_to_enu(lat=gps_radians[0], lon=gps_radians[1],
alt=0,
                                 ecef0=reference_ecef,
ref_matrix=reference_matrix)
            current_measurement[:2] = gps_enu[:2]
            current_measurement[2] = input[4]
            gps_orient = np.radians( input[6])
            current_measurement[3] = np.sin(gps_orient)
            current_measurement[4] = np.cos(gps_orient)
            # Predict with last velocity and yaw rate, update with last gps
            current_state, current_cov = ukf.propagate(current_state,
current_cov,
                                                       current_control,
ctra.transit_ctra,
                                                       current_measurement,
ctra.measmt_func_gps,
                                                       (input[0] -
last_timestamp) * 1e-6)
            last_timestamp = input[0]
            states.append(np.degrees(ctra.state_to_latlon(current_state,
reference_ecef, reference_matrix_T)))
    return states


if __name__ == '__main__':
    data = pd.read_csv('inputs/total.csv')
    data = data.to_numpy()[2588:]
    reslist = run_CTRV(data)
```

```
res = pd.DataFrame(reslist, columns=['latitude', 'longitude', 'alt'])
res.to_csv('results.txt')
```