**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра математичних основ комп’ютерних наук

«До захисту допущено»  
Завідувач кафедри  
М. П. Крамаренко

(підпис)

«      »                         20      р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

за спеціальністю122 Комп’ютерні науки

на тему:

**НАЗВА ВАШОЇ ДИПЛОМНОЇ РОБОТИ ВЕЛИКИМИ ЛІТЕРАМИ ЖИРНИМ ШРИФТОМ**

Виконала студентка 4 курсу  
Головко Михайлина Вікторівна

(підпис)

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Морковченко Микола Олександрович

(підпис)

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

Студент

(підпис)

Київ – 2017

# РЕФЕРАТ

# ЗМІСТ

# ВСТУП

**Оцінка сучасного стану об’єкта розробки**: Зараз у більшості автомобілів для просторової орієнтації використовується лише GPS-навігатор.

Починаючи з його винайдення у 1960 р., фільтр Калмана широко використовується для високоточної просторової орієнтації в багатьох галузях, включно з автомобільною.

**Актуальність роботи та підстави для її виконання**. Тесла маск гугл хуйо майо

**Мета й завдання роботи.**

**Об'єкт, методи й засоби розроблення.**

**Можливі сфери застосування**. хаха машинки go brrrrrrrrr

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ФІЛЬТР

## 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.

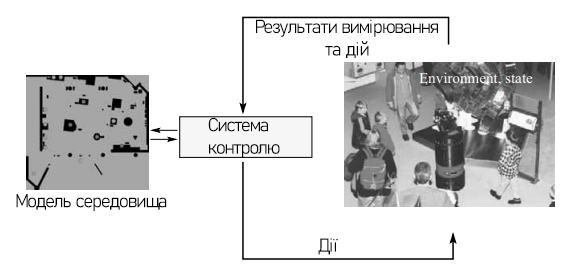
Але що таке стан? Стан – це набір всіх параметрів об’єкта та навколишнього середовища, що можуть вплинути на майбутнє. Так, наприклад, у позиціонуванні автівки станом може бути його положення, швидкість, прискорення та орієнтація. У більш складних задачах у стан також може бути записане, наприклад, місцезнаходження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

Рисунок 1 - взаємодія об'єкта з середовищем

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.

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

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

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  |  |

Важливим і необхідним припущенням рекурсивної оцінки 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 по ніяк не впливають на стан , окрім як через стан .

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

|  |  |  |
| --- | --- | --- |
|  |  | (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:

|  |  |  |
| --- | --- | --- |
|  |  |  |

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:

|  |  |  |
| --- | --- | --- |
|  |  |  |

і спрощується до

|  |  |  |
| --- | --- | --- |
|  |  |  |

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 залежить лише від стану у момент та керування у момент , спостереження в момент часу залежить виключно від стану у момент . Таку модель еволюції станів називають **прихованою марківською моделлю**, або **динамічною байєсівською мережею**.

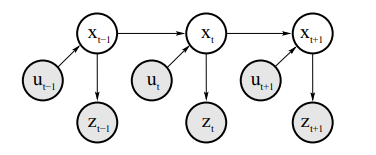


Рисунок 2 – еволюція станів; стрілки ілюструють залежності

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 ця інформація – керування та спостереження . Вгадане значеня стану називають belief і вираховують через belief distribution. Цей розподіл співставляє кожній гіпотезі щодо справжнього значення стану її апостеріорну ймовірність:

Іноді (зокрема, в фільтрі Калмана) необхідно обчислювати без знання :

Цей розподіл називають **передбаченням**, бо він надає можливість передбачити наступний стан, знаючи попередній belief до врахування спостереження у момент . Процес врахування спостереження, тобто, обрахунку з , називають **уточненням**.

## 1.2 Байєсівський фільтр

Байєсівський фільтр – найпростіший та найбільш загальний алгоритм для рекурсивної ймовірнісної оцінки стану. Алгоритм рекурсивно обчислює belief distribution з інформації про останнє спостереження, останнє керування та попередній стан. Алгоритм має вигляд:

1. ***алгоритм байєсівський\_фільтр***
2. *для всіх можливих значень стану :*
3. // *Передбачення*
4. // *Уточнення*
5. *повернути*  (3)

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

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

Уточнення ж відбувається в рядку 4. Алгоритм обчислює значення розподілу belief для як добуток ймовірності отримати спостереження за такого значення стану на передбачене значення . 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 .

Оскільки алгоритм обчислює розподіли рекурсивно зі значень попередніх розподілів, необхідно задати початкове значення . Найбільш розповсюдженими є два крайові випадки: значення початкового стану відомо точно і значення початкового стану не відомо взагалі. У першому випадку функцію задають так:

де – точно відоме значення стану у момент 0.

У разі того, що значення не відомо взагалі, задається рівномірним розподілом по всім можливим значенням початкового стану.

Також можливо таке, що значення відомо з певною точністю, але не точно. У такому разі 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 belief, або набір моживих станів).

## 1.3 Фільтр Калмана

Найбільш розпоширений алгоритм на базі байєсівського фільтра – фільтр Калмана. У цьому алгоритмі belief у момент представлено середнім значенням та матрицею коваріації . 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 має бути лінійною за та з доданням Гаусівського шуму:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

де , – вектори стану;

– вектор керування;

– Гаусівський шум.

Вектор стану має вигляд:

вектор керування має вигляд:

.

– матриця розміру , де – розмір вектору стану .

– матриця розміру , де – розмір вектору керування .

– гаусівський вектор з середім значенням, що дорівнює нулю, та матрицею коваріації . Виміри вектора 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:

2. Ймовірність спостереження має бути лінійною функцією за з доданим Гаусовим шумом:

де – вектор стану;

– матриця розміру , де – розмір вектору спостережень;

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

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

3. Початковий belief має бути випадковою величиною з гаусовим розподілом:

Якщо всі ці три умови виконуються, то для будь-якого моменту часу результат ітерації фільтру – апостеріорний belief – буде нормально розподіленим.

Алгоритм має вигляд:

1. ***алгоритм лінійний\_фільтр\_калмана***
2. // *Передбачення*

5. // *Уточнення*

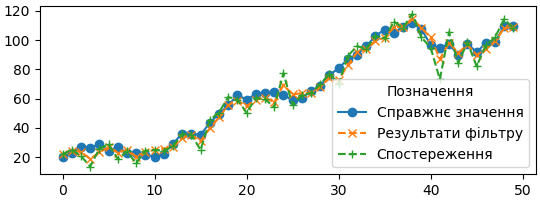
**7.** *повернути*

У рядках 2 та 3 за результатами керування обчислюється наступний стан, представлений середнім значенням та коваріацією. Це робиться підставленням у формулу (4) замість середнього значення у рядку 2 та коваріації 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 – результату підстановки передбаченої коваріації у рівняння спостереження, тобто, коваріації спостереження.

У рядках 5 та 6 відбувається уточнення. Виразом обчислюється **іновація** – відстань між очікуваним та дійсним спостереженнями.

Рисунки 3 та 4 ілюструють роботу фільтру Калмана. На рисунку 3 зображено результат роботи фільтру протягом 50 моментів часу (50 ітерацій). Середня квадратична помилка результатів роботи фільтру відносно справжніх станів в два рази менша за середню квадратичну помилку спостережень. На рисунку 4 детально зображено роботу однієї ітерації.



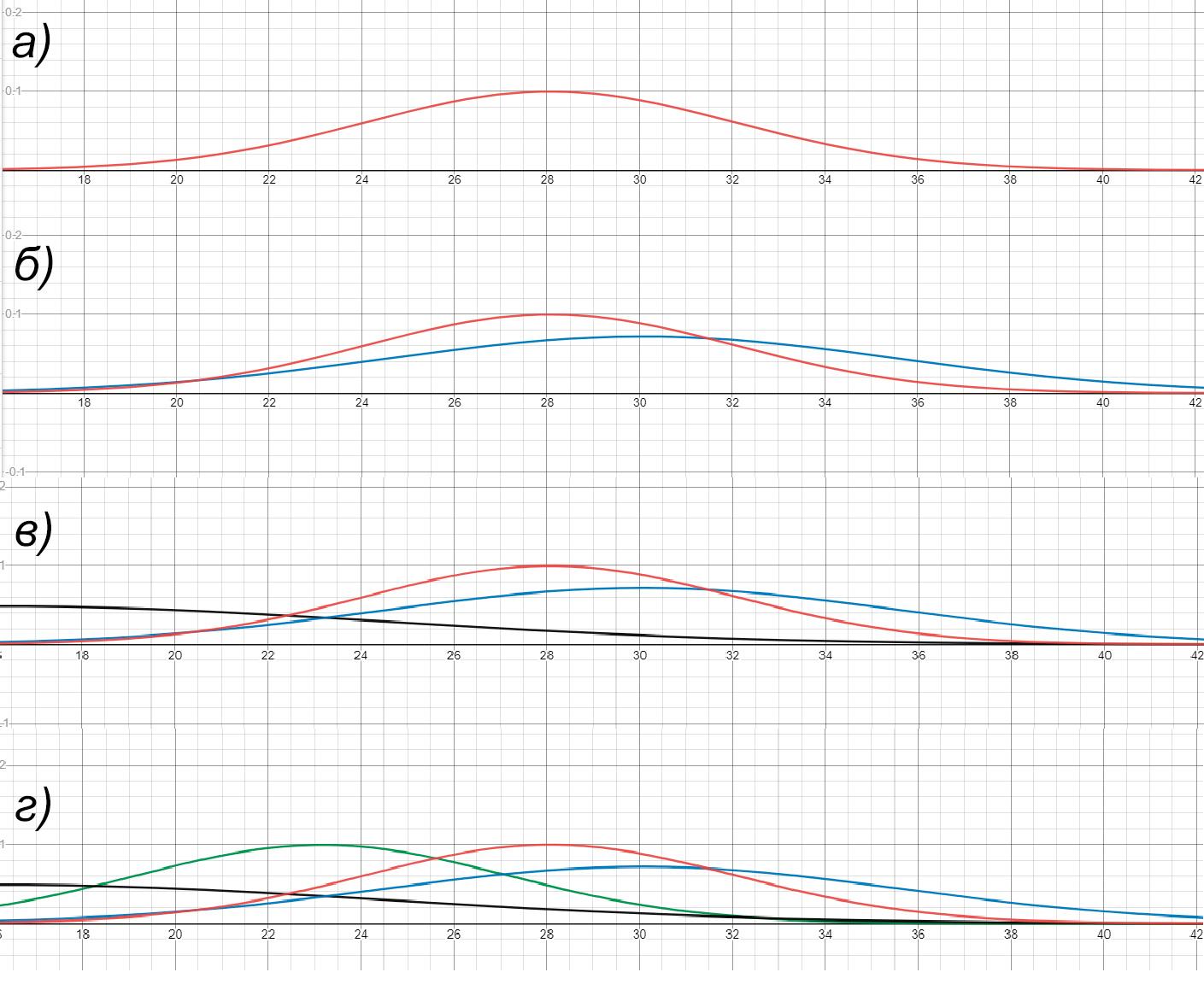
Рисунок 3 – результат роботи фільтру

Рисунок 4 – одна ітерація фільтру: 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.)

## 1.4 Фільтри Калмана для нелінійних систем

Незважаючи на свою ефективність, лінійний фільтр Калмана нечасто використовується на практиці через обмеження на лінійність, які він накладає на функцію переходу та спостережень, які потрібні для того, щоб кінцевий belief мав гаусів розподіл. Протягом ХХ сторіччя було розроблено декілька алгоритмів, заснованих на фільтрі Калмана, що знімають це обмеження – розширений фільтр Калмана та беззапаховий фільтр Калмана.

**1.4.1 Розширений фільтр Калмана**

Розширений фільтр Калмана використовує лінеаризацію нелінійних функцій.

**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 , та

де – нелінійна функція;

𝑋 – випадкова величина з середнім значенням та матрицею коваріації . Алгоритм беззапахового перетворення дозволяє це зробити таким чином:

1. Обчислити значення параметру

,

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

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

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

*–* розмірність випадкової величини *X*.

1. Обчислити сігма-точок :

;

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

де – *і*-тий рядок матриці ;

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

;

;

|  |  |  |
| --- | --- | --- |
|  |  | ; |

1. Обчислити результат застосування цільової функції до сігма-точок:

|  |  |  |
| --- | --- | --- |
|  |  | ; |

1. Обчислити середнє значення та матрицю коваріації розподілу :

Якщо *X* має нормальний розподіл, параметри задаються значеннями відповідно. Інші розповсюджені варіанти задання параметрів включають:

.

Дію цього алгоритму ілюструє рис. 5. Зліва зображено результат методу Монте-Карло, по центру – результат лінеаризації функції , 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 - ) значень.

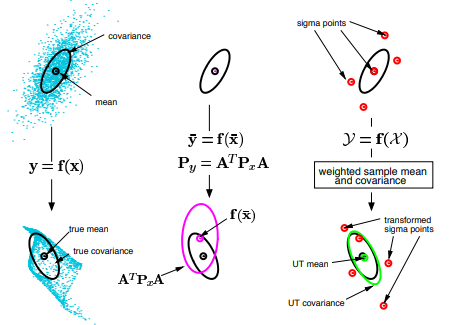


Рисунок 5 – порівняння результатів застосування методу Монте-Карло, лінеаризації та беззапахової трансформації для визначення параметрів розподілу випадкової величини після застосування до неї нелінійної функції

Беззапахове перетворення у беззапаховому фільтрі Калмана застосовується для того, щоб зняти обмеження на лінійність функцій переходу та спостереження. Нехай система має функцію переходу та функцію спостереження . Завдяки використанню беззапахового перетворення, немає значення, чи є вони лінійнимі за своїми аргументами, чи ні. Псевдокод однієї ітерації алгоритму зображено нижче. Важливим зауваженням є те, що ваги та можна не обчислювати щоітерації, бо їх значення залежать лише від параметрів, що задаються на початку роботи алгоритму.

1. ***алгоритм беззапаховий\_фільтр\_калмана***













16. ***повернути*** <>

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 . У рядках 6 та 7 обчислюються передбачені (апріорні) середнє значення та коваріація стану у момент . 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 і його матриця коваріації.

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

Зображено на рис. 6.

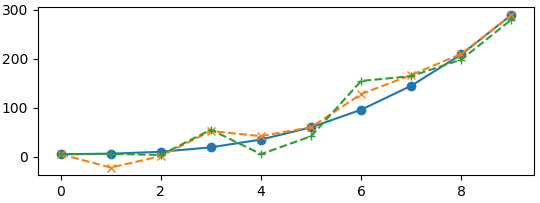


Рисунок 6 – результат роботи беззапахового фільтру Калмана. Зеленим кольором позначено спостереження, помаранчевим – результат роботи фільтру

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.

**1.2.1** **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.

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

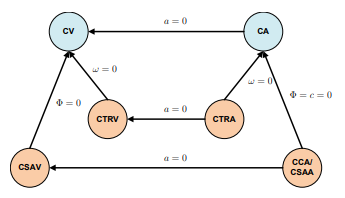


Рисунок 7 – схема, що ілюструє відмінності між різними моделями руху

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.

Найскладнішою з моделей, що будуть розглянуті, є CCA (Constant Curvature and Acceleration – константні ... та прискорення). Ця модель враховуює залежність швидкості повороту автівки від швидкості руху.

Існують і більш складні моделі, 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).

## 1.2.2 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:

1. Для CV:

де – положення об’єкту за вісями та відповідно;

– швидкість об’єкту за вісями та відповідно;

– час, що пройшов з моменту останнього передбачення стану.

1. Для CA:

де – положення об’єкту за вісями та відповідно;

– швидкість об’єкту за вісями та відповідно;

– прискорення об’єкту за вісями та відповідно;

– 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.

## 1.2.3 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:

де – положення об’єкту за вісями та відповідно;

– кут повороту об’єкту (орієнтація);

– швидкість руху об’єкту;

– 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.

## 1.2.4 Модель CTRA

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

де – положення об’єкту за вісями та відповідно;

– кут повороту об’єкту (орієнтація);

– швидкість руху об’єкту;

– прискорення об’єкту у напрямку руху;

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

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

## 1.2.5 Модель CCA

## 1.2.5 Вибір моделі

Постає питання вибору з урахуванням переваг та недоліків розглянутих моделей.

Найбільш точною є модель CCA, але вона також є найбільш складною у реалізації. Через це найбільш логічним вибором є або модель CTRA, або CTRV – компроміс між складністю реалізації та точністю. У цій роботі буде розглянено деталі реалізації обох моделей та порівняно їх результати.

# 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. 8.

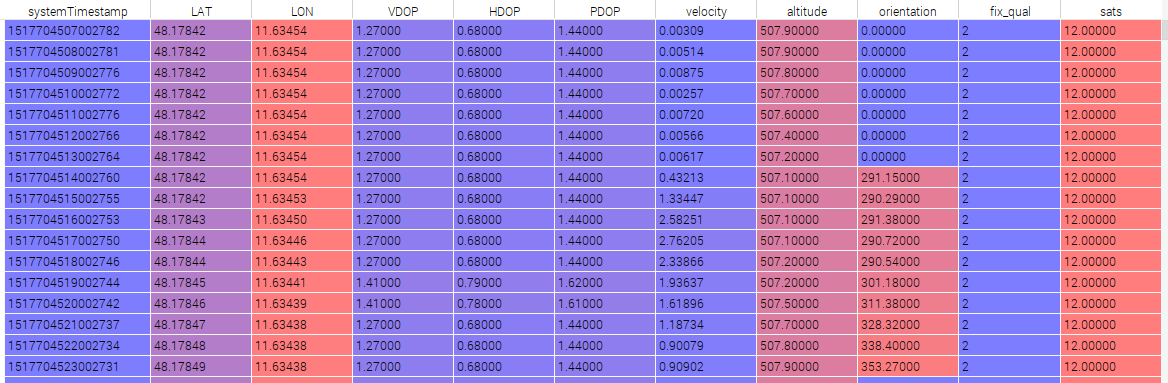


Рисунок 8 – приклад даних з GPS.

Значення стовпчиків у даних:

* SystemTimestamp: час надходження даних ();
* 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**

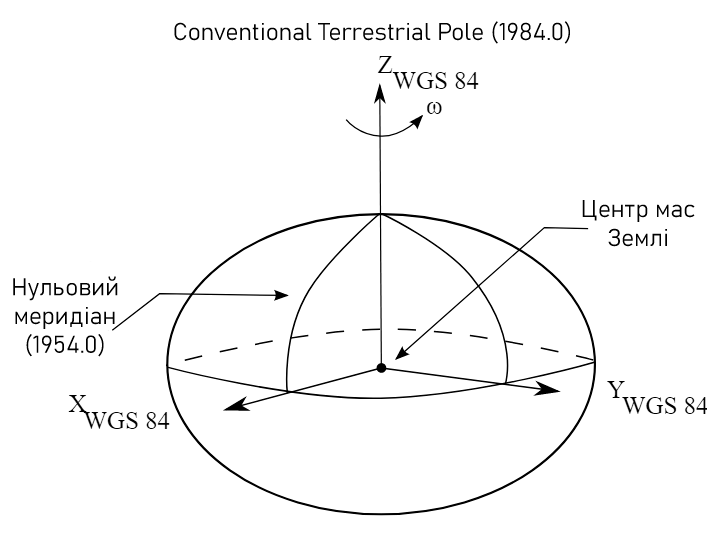
 Саме у цій системі координат за замовчуванням приходять дані з датчиків

Рисунок 9 – система координат WGS84

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.

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

## 3.1.2 Система ECEF

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

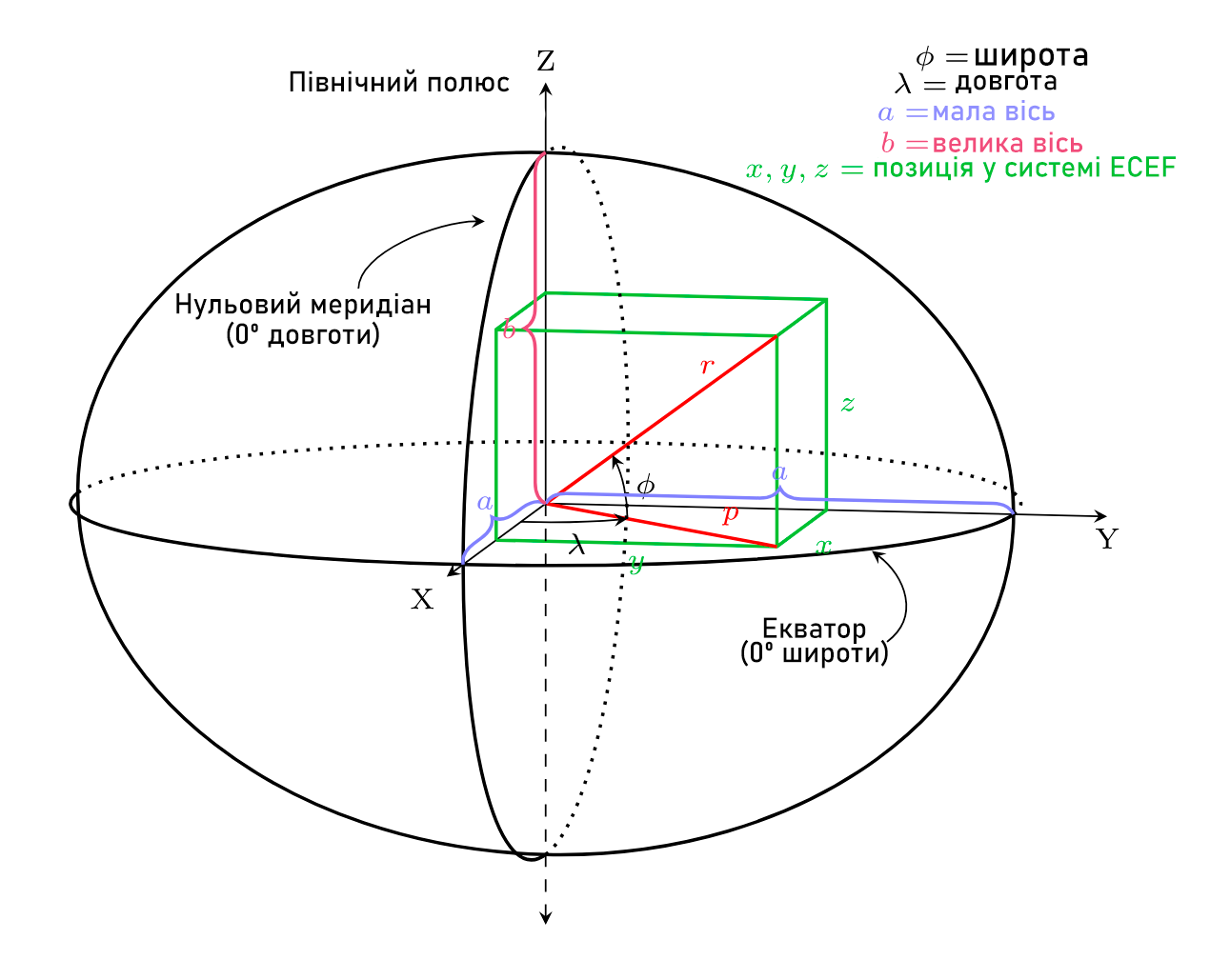
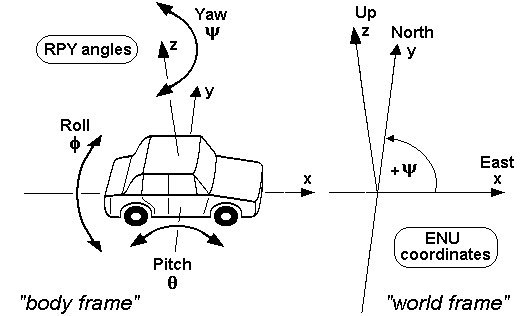


Рисунок 10 – схема системи координат ECEF

## 3.1.3 Система ENU

Позиція у системі ENU (East, North, Up) є трійкою – координатами об’єкту в Декартовій системі координат з початком координат у довільній точці, де вісь спрямована з початку координат на схід, вісь – на північ, а вісь – 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 11 – 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 у WGS84. Її координати у системі ECEF визначаються за формулами:

де – мала вісь Землі;

– ексцентриситет Землі;

.

Цю точку, у свою чергу, можна перевести у систему ENU з початком координат у точці з координатами WGS84 та ECEF переведенням у інший базис:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Отриманий вектор є шуканими координатами точки А в системі ENU.

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 (3.1):

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

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |
|  |  | (3.3) |
|  |  | (3.4) |
|  |  | (3.5) |
|  |  | (3.6) |
|  |  | (3.7) |
|  |  | (3.8) |
|  |  | (3.9) |
|  |  | (3.10) |
|  |  | (3.11) |
|  |  | (3.12) |
|  |  | (3.13) |
|  |  | (3.14) |
|  |  | (3.15) |
|  |  | (3.16) |
|  |  | (3.17) |
|  |  | (3.18) |
|  |  | (3.19) |
|  |  | (3.20) |
|  |  | (3.21) |
|  |  | (3.22) |
|  |  | (3.23) |
|  |  | (3.24) |
|  |  | (3.25) |

де – ексцентриситет Землі;

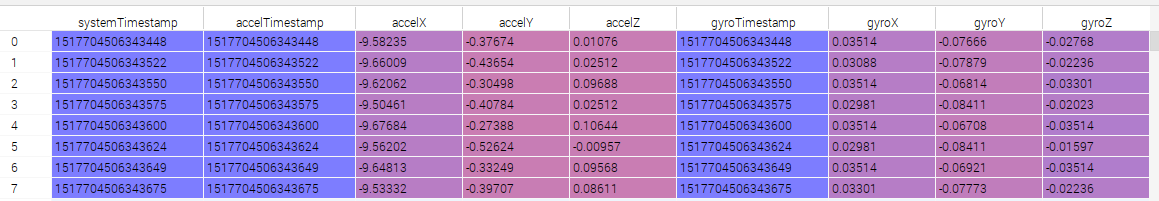
– мала вісь Землі;

– 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 (3.22), (3.23) та (3.24) 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:

Рисунок 12 – дані ІВП

Тлумачення стовпчиків

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

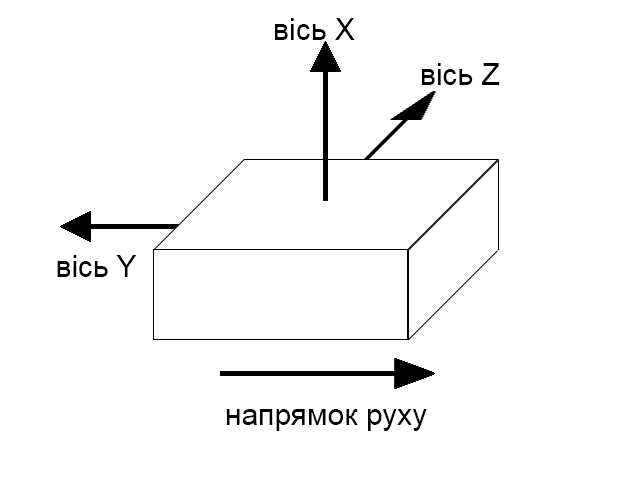


Рисунок 13 – розміщення осей ІВП

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.

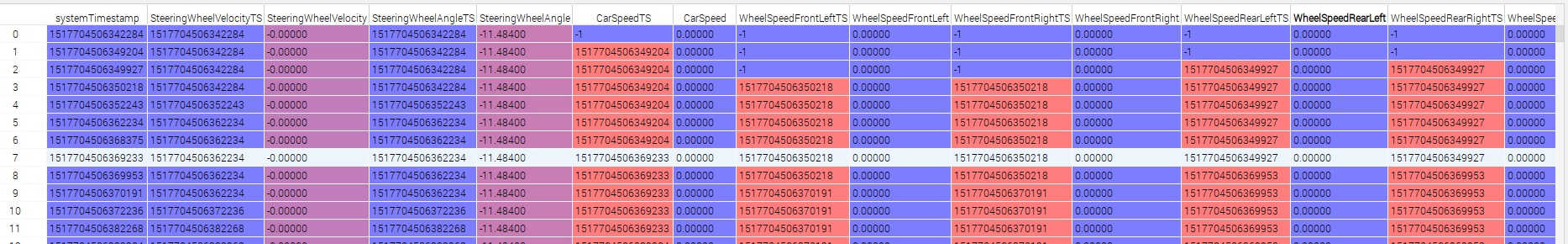


Рисунок 14 – приклад даних інших датчиків

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:

де – швидкість обертання лівого заднього колеса;

– швидкість обертання правого заднього колеса;

– радіус колеса автівки;

# 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]](#footnote-1). Нарешті, у підрозділі 4.х буде наведено та проаналізовано результати роботи програми.

## 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.

## 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 .

**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 CTRV 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 , 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 – , 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 виведено 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)**

## 4.5 Результати роботи алгоритму

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.

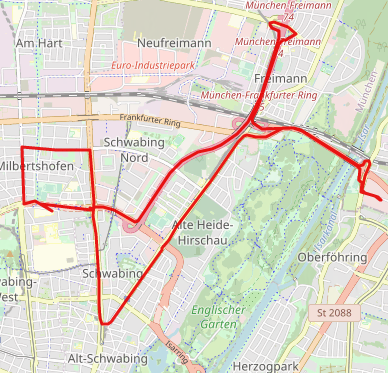


Рисунок 15 – траєкторія за фільтром з моделлю CTRV (зліва) та дані лише з GPS (справа)

На рисунку 15 порівняно результат роботи фільтру з даними «чистого» GPS. На відкритих ділянках траєкторії відрізняються в рамках 1м. Різницю краще видно при порівнянні траєкторії на ділянці з тонелем (рис. 16). Як видно, фільтр справляється з поставленою задачею – визначення місцезнаходження автівки в умовах поганого чи відсутнього сигналу GPS.



Рисунок 16. Траєкторія під час в’їзду у тонель. Траєкторія, отримана з фільтру (зліва) є набагато плавнішою і набагато ближча до справжньої, ніж отримана з GPS (справа)

1. Реалізацію решти необхідних функцій – операцій з перетворення GPS, генерації результатів – наведено в додатку 1. [↑](#footnote-ref-1)