Кафедра СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської кваліфікаційної роботи на тему

РЕАЛІЗАЦІЯ АЛГОРИТМУ ФІЛЬТРАЦІЇ ТРАЄКТОРІЇ РУХУ GPS-МОДУЛЯ ЗА РЕЗУЛЬТАТАМИ ВИМІРЮВАННЯ КООРДИНАТ

Студент групи КН-412 **Рибачук Остап Андрійович**
(шифр, прізвище та ініціали)

Керівник проекту _____ (**Наталія Спас**)

Консультанти _____ (_____)

_____ (_____)

_____ (_____)

_____ (_____)

Завідувач кафедри Михайло ЛОБУР

«___» _____ 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут Інститут комп'ютерних наук та інформаційних технологій

Кафедра Систем автоматизованого проектування

Спеціальність F3 "Комп'ютерні науки"

„ЗАТВЕРДЖУЮ”

Завідувач

кафедри САП

Михайло ЛОБУР

«___» _____ 2025 р.

ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу студента групи КН-412 ОКР Бакалавр

Рибачук Остап Андрійович

(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Реалізація алгоритму фільтрації траєкторії руху GPS-модуля за результатами вимірювання координат

затверджена наказом по університету від 30.04.2025 р. № 1566-4-08

2. Термін подання студентом закінченої роботи (проекту) 06.06.2025

3. Вихідні дані для роботи (проекту)

- існуючі технології фільтра Калмана в навігаційних системах;
- стандарти навігаційних системах;.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які треба розробити)

- адаптивність фільтра Калмана в навігаційній системі;
- програмна реалізація фільтра Калмана.

5. Перелік графічного матеріалу

- Гіперпараметризована модель обертання алгоритму фільтра Калмана для безпілотних об'єктів;
- Формування навігаційних сигналів із сукупності супутників.

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту)

GPS, Arduino, Raspbery Py, Python

7. Консультування роботи (проекту), із зазначенням розділів роботи

Розділ	Консультант	Завдання видав		Завдання прийняв	

8. Дата, коли видано завдання 01.05.2025

Керівник **Наталія Спас**
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1.	Ознайомлення з предметною областю, вивчення		
	літератури, подібних систем	01.05 – 5.05	
2.	Робота над першим розділом	6.05 – 11.05	
3.	Робота над другим розділом	12.05 – 17.05	
4.	Розробка і тестування системи	18.05 – 22.05	
5.	Робота над третім розділом	23.05 – 26.05	
6.	Оформлення текстової і графічної частини		
	бакалаврської кваліфікаційної роботи	27.05 – 30.05	

Студент _____
(підпис)

Керівник _____
(підпис)

АНОТАЦІЯ

Рибачук О., Спас Н.Я. Реалізація алгоритму фільтрації траєкторії руху GPS-модуля за результатами вимірювання координат. Бакалаврська кваліфікаційна робота. – Національний університет «Львівська політехніка», Львів, 2025 р.

Розширена анотація

Супутникова навігаційна система складається з групи від 2 до 30 орбітальних супутників, що передають спеціальні радіосигнали, наземної системи керування та приймального обладнання користувача «супутниковими навігаторами». Для підвищення точності визначення координат може використовуватися система наземних станцій, а також системи, що передають користувачеві точні координати та супутникові годинники.

Положення приймача в супутниковій навігаційній системі розраховується за допомогою діапазонів різних супутникових сузір'їв. Радіус сфери неможливо точно визначити через такі фактори, як затримки в іоносфері та тропосфері, релятивістські ефекти та вплив прийнятих відбитих сигналів. Без будь-якої корекції точність позиціонування може досягати 30-50 метрів [1].

Наразі в експлуатації або готуються до розгортання такі супутникові навігаційні системи: NAVSTAR - GPS (Global Positioning System) - це супутникова навігаційна система, яка вимірює відстань, час та визначає місцезнаходження. Він дозволяє визначати положення та швидкість будь-якої точки на Землі практично за будь-яких погодних умов, а також положення та швидкість об'єктів у космічному просторі поблизу Землі. Система була розроблена, впроваджена та експлуатується Міністерством оборони США.

Основний принцип використання цієї системи полягає у визначенні положення шляхом вимірювання відстані від відомої точки координат до об'єкта. Ця відстань розраховується на основі часу затримки, необхідного для проходження сигналу від супутника до приймальної антени GPS. Тобто, для визначення тривимірних координат GPS-приймачу необхідно знати відстані до

трьох супутників та час системи GPS. Тому для визначення координат та висоти приймача необхідні сигнали щонайменше від чотирьох супутників.

Кожна система транслює один і той самий контент на двох частотах. Це робиться для того, щоб розрахувати іоносферну затримку, оскільки іоносферна затримка залежить від частоти, і ця залежність відома. Однак через використання ключів шифрування цивільні користувачі не можуть отримати доступ до даних на другій частоті.

Найважливішими сферами застосування супутникової навігації є: геодезичні роботи, кадастрова зйомка: використання навігаційних систем для визначення точних координат меж земельних ділянок, картографія: створення карт за допомогою супутникових навігаційних систем, навігація: навігаційні системи використовуються для морської, дорожньої, авіаційної та інших видів навігації, супутниковий моніторинг транспортних засобів: використання навігаційних систем для відстеження місцезнаходження та швидкості транспортних засобів і контролю їх руху, місцезнаходження прстрою, геофізичні дослідження: використання навігаційних систем для моніторингу руху та коливань літосферних плит.

Кожен регіон, який використовує певну систему супутникової навігації, має свої власні вимоги до точності позиціонування. Наприклад, у кадастрових роботах похибка не повинна перевищувати 10 сантиметрів. Натомість, у дорожній навігації зазвичай достатньо точності 5 метрів. Для задоволення цих вимог використовуються різні датчики та методи.

Метою цієї роботи є розробка алгоритму фільтрації даних, отриманих від супутникових навігаційних систем, з використанням модуля супутникової навігаційної системи.

Об'єкт дослідження – дані, отримані від супутникових навігаційних систем.

Методом дослідження є фільтрація Калмана. Аналіз продуктивності фільтра Калмана. Вибір навігаційного модуля для отримання даних про місцезнаходження. Аналіз даних, відфільтрованих кожним модулем.

Практична цінність роботи полягає в обробці даних, отриманих за допомогою цього модуля, для зниження співвідношення сигнал/шум. Аналітичні та обчислювальні матеріали, представлені в ній, правильно та об'єктивно відображають стан досліджуваних процесів

Ключові слова: фільтр Кальмана, навігаційна система, супутниковий зв’язок, навігаційний модуль.

Список використаних джерел.

1. O. Belej, N. Nestor, I. Artyshchuk and N. Spas, "Modeling of Wireless Sensor Network Based on Functioning Parameters of Unevenly Distributed Nodes," 2023 17th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Jaroslaw, Poland, 2023, pp. 23-27, doi: 10.1109/CADSM58174.2023.10076512.

2. O. Belej, J. Sadeckii and T. Lohutova, "The Features of Security of Transfer and Storage Data for the Internet of Things in Cloud Database," 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 2019, pp. 7-12, doi: 10.1109/PICST47496.2019.9061416. - **Date of Conference:** 8-11 Oct. 2019.

3. O. Belej, N. Nestor and O. Polotai, "Developing a Local Positioning Algorithm Based on the Identification of Objects in a Wi-Fi Network of the Mall," *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, Polyana, Ukraine, 2019, pp. 32-36. - doi: 10.1109/MEMSTECH.2019.8817385.

4. O. Belej, I. *Artyshchuk*, W. Sitek The Controlling of Transmission of Chaotic Signals in Communication Systems Based on Dynamic Models. / CEUR Workshop Proceedings / vol. 2353 (2019). online: http://ceur-ws.org/Vol-2353/paper38.pdf

ABSTRACT

Rybachuk O., Spas N.Ya. Implementation of the GPS module trajectory filtering algorithm based on coordinate measurement results. Bachelor's thesis. - Lviv Polytechnic National University, Lviv, 2025.

Extended abstract

A satellite navigation system consists of a group of several (from 2 to 30) orbiting satellites transmitting special radio signals, a ground control system and user receiving equipment (the so-called "satellite navigators"). To increase the accuracy of determining coordinates, a system of ground stations can be used, as well as systems that transmit precise coordinates and satellite clocks to the user.

The position of the receiver in a satellite navigation system is calculated using the ranges of various satellite constellations. The radius of the sphere cannot be determined precisely due to factors such as delays in the ionosphere and troposphere, relativistic effects and the influence of received reflected signals. Without any correction, the positioning accuracy can reach 30-50 meters. [1]

The following satellite navigation systems are currently in operation or are being prepared for deployment: NAVSTAR - GPS (Global Positioning System) - is a satellite navigation system that measures distance, time and determines location. It allows you to determine the position and velocity of any point on Earth (except for the polar regions) in almost any weather conditions, as well as the position and velocity of objects in outer space near the Earth. The system was developed, implemented and operated by the US Department of Defense.

The basic principle of using this system is to determine the position by measuring the distance from a known coordinate point to the object. This distance is calculated based on the delay time required for the signal to travel from the satellite to the GPS receiving antenna. That is, to determine the three-dimensional coordinates, the GPS receiver needs to know the distances to three satellites and the GPS system time. Therefore, signals from at least four satellites are required to determine the coordinates and altitude of the receiver.

Each system broadcasts the same content on two frequencies. This is done in order to calculate the ionospheric delay, since the ionospheric delay depends on the frequency, and this dependence is known. However, due to the use of encryption keys, civilian users cannot access data on the second frequency.

The most important areas of application of satellite navigation are: eodetic works, cadastral surveying: use of navigation systems to determine exact coordinates of boundaries of land plots, cartography: creation of maps using satellite navigation systems, navigation: navigation systems are used for marine, road, aviation and other types of navigation, satellite vehicle monitoring: use of navigation systems to track the location and speed of vehicles and control their movement, hive: if your phone has satellite navigation, you can find its location, geophysical research: use of navigation systems to monitor the movement and oscillations of lithospheric plates.

Each region using a particular satellite navigation system has its own requirements for positioning accuracy. For example, in cadastral works, the error should not exceed 10 centimeters. In contrast, in road navigation, an accuracy of 5 meters is usually sufficient. To meet these requirements, various sensors and methods are used.

The purpose of this work is to develop an algorithm for filtering data received from satellite navigation systems using the GNSS module.

The object of research is data received from satellite navigation systems.

The research method is Kalman filtering. Analysis of the performance of the Kalman filter. Selection of a navigation module for obtaining location data. Analysis of data filtered by each module.

The practical value of the work lies in processing the data received using this module to reduce the signal-to-noise ratio. The analytical and computational materials presented in it correctly and objectively reflect the state of the processes under study

Keywords: Kalman filter, navigation system, satellite communication, navigation module.

References.

1. O. Belej, N. Nestor, I. Artyshchuk and N. Spas, "Modeling of Wireless Sensor Network Based on Functioning Parameters of Unevenly Distributed Nodes," 2023 17th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Jaroslaw, Poland, 2023, pp. 23-27, doi: 10.1109/CADSM58174.2023.10076512.

2. O. Belej, J. Sadeckii and T. Lohutova, "The Features of Security of Transfer and Storage Data for the Internet of Things in Cloud Database," 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 2019, pp. 7-12, doi: 10.1109/PICST47496.2019.9061416. - **Date of Conference:** 8-11 Oct. 2019.

3. O. Belej, N. Nestor and O. Polotai, "Developing a Local Positioning Algorithm Based on the Identification of Objects in a Wi-Fi Network of the Mall," *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, Polyana, Ukraine, 2019, pp. 32-36. - doi: 10.1109/MEMSTECH.2019.8817385.

4. O. Belej, I. *Artyshchuk,* W. Sitek The Controlling of Transmission of Chaotic Signals in Communication Systems Based on Dynamic Models. / CEUR Workshop Proceedings / vol. 2353 (2019). online: http://ceur-ws.org/Vol-2353/paper38.pdf

ЗМІСТ

ВСТУП

Актуальність роботи. Супутникова навігаційна система (GNSS) – це складна електронно-технологічна система, що складається з набору наземних та космічних засобів, що використовуються для визначення положення, а також параметрів руху для різних цілей. Це положення визначається трьома координатами супутникового навігаційного приймача в певній системі координат. Найпоширенішими системами координат є ECEF та WGS84. Якщо положення розглядається у сферичній системі координат і третя координата (висота) невідома, то позиціонування є двовимірним. Якщо координати приймача не змінюються під час процесу позиціонування, таке позиціонування називається статичним. В іншому випадку мова йде про спортивне позиціонування.

Існує кілька способів вирішення задач супутникової навігації. Найпопулярнішим методом у минулому був різницевий метод. Ці методи вимагають, щоб принаймні одна наземна станція була розташована досить близько до місця вимірювання. Ці методи дозволяють використовувати алгебраїчні перетворення для зменшення більшості помилок, зокрема іоносферних та тропосферних затримок, а також помилок синхронізації між супутником та приймачем. Однак, чим далі від наземної станції, тим нижча точність позиціонування. Це головний недолік диференціальних методів, але, незважаючи на це, їм приділялася велика увага в минулому, тоді як абсолютним методам приділялося менше уваги. Однак ситуація почала змінюватися з появою наземних сервісів, здатних обчислювати різну додаткову інформацію. Сьогодні найбільшим таким сервісом є IGS (Міжнародна глобальна навігаційна супутникова система). Останнім часом все більшої популярності набуває точкове позиціонування. Такий підхід передбачає врахування максимальної кількості даних для корекції різних помилок та затримок.

Визначення вашого місцезнаходження в супутниковій навігаційній системі фактично є пошуком трьох координат. Окрім цих величин, багато відносних параметрів можна виміряти за допомогою інерціальних датчиків (ІД):

акселерометрів, гіроскопів та магнітометрів. Акселерометр може бути використаний для вимірювання трьох координат прискорення, гіроскоп – для вимірювання трьох координат кутової швидкості, а магнітометр – для вимірювання трьох координат вектора магнітного схилення. Ці дані також можна використовувати для розв'язання навігаційних рівнянь. Система, яка коригує координати, обчислені супутниковим навігаційним приймачем на основі інерціальних даних, називається слабозв'язаною системою. Тіснозв'язані системи передбачають використання інерціальних датчиків для корекції даних, отриманих від супутникових навігаційних пристроїв, перед обчисленням положення.

Метою цієї роботи є розробка алгоритму фільтрації даних, отриманих від супутникових навігаційних систем, з використанням модуля супутникової навігаційної системи.

Для досягнення цієї мети необхідно розглянути такі завдання:

- вивчення теорії фільтрації та створення прогнозів;

- адаптивність фільтра Калмана в навігаційній системі;

- програмна реалізація фільтра Калмана;

- використання цього модуля для експериментального вимірювання координат рухомого об'єкта.

Об'єкт дослідження – дані, отримані від супутникових навігаційних систем.

Предмет дослідження є здатність фільтра Калмана обробляти дані, отримані від систем супутникової навігації.

Практична цінність роботи полягає в обробці даних, отриманих за допомогою розробленого модуля, для зниження співвідношення сигналу і шумів. Аналітичні та обчислювальні матеріали, представлені в ній, правильно та об'єктивно відображають стан досліджуваних процесів

Бакалаврська кваліфікаційна робота за структурою складається із вступу, переліку умовних позначень, трьох розділів, висновків, списку використаних джерел та додатків.

1. КЛАСИЧНІ МЕТОДИ ПОЗИЦІОНУВАННЯ

1.1. Позиціонування методом сферичного перетину

Кожна система супутникової навігації містить щонайменше 24 супутники. Супутники безперервно передають пакети даних, що містять час запуску та координати супутника на момент запуску.

Швидкість передачі даних відома - c (швидкість світла у вакуумі). Після отримання пакета даних приймач обчислює час передачі сигналу

$$\Delta t = tr - ts, \qquad (1.1)$$

де tr - час прийому, а ts - час відправки. Це означає, що відстань від супутника (під час передачі) до приймача (під час прийому) є добутком часу та швидкості:

$$d = \Delta t \times s. \qquad (1.2)$$

Отже, приймач може бути розташований у будь-якій точці сфери радіуса d з центром у точці, де супутник знаходився під час передачі.

Якщо додати повідомлення від другого супутника, приймач буде розташований на перетині двох сфер. Отже, додавши передавач, ми можемо визначити точне положення. (Рис. 1.1) схематично зображує цю ситуацію.

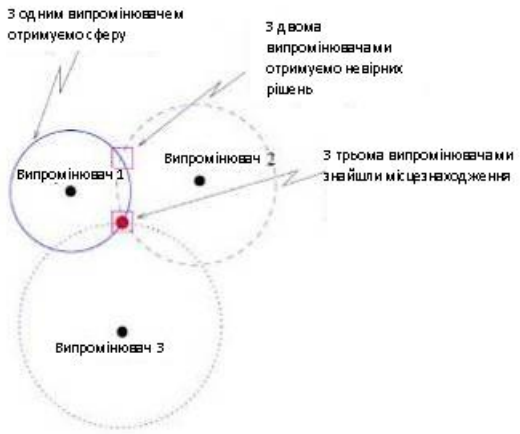


Рис. 1.1: Визначення положення шляхом перетину трьох сфер.

Однак цей метод не дозволяє нам визначити точне місцезнаходження. Це пояснюється тим, що відстань від супутника до приймача визначається неточно: різні помилки та затримки враховуються в розрахунку, але не враховуються.

Ця ситуація показана на рис. 1.2.

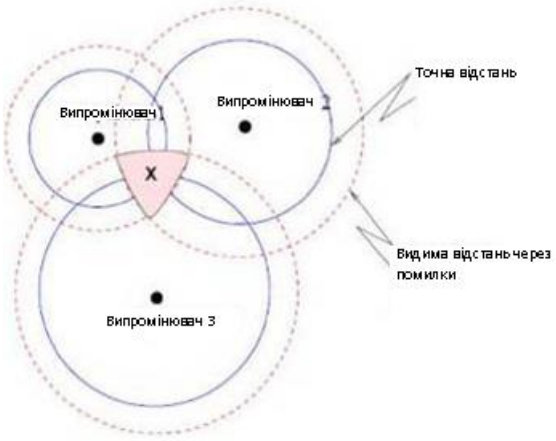


Рис. 1.2: Місцезнаходження визначається неточно

Чим більше помилок ви усунете, тим точніше буде ваше місцезнаходження.

1.2. Помилки вимірювання та їх порядок

В аналітичній формі представлено більшість помилок та затримок, що виникають під час вимірювання відстані від супутника до приймача.

Помилка – це будь-яке відхилення в координатах супутника або годинниках приймача та супутника. Затримка – це помилка, яка спричиняє зміну швидкості, з якою сигнал поширюється між супутником і приймачем у певний момент часу.

Припустимо, у нас є приймач і кілька супутників. Передбачається, що існує спільна шкала часу, а саме час, коли годинники супутника та приймача збігаються. Середній час за Гринвічем (час на початковому меридіані) майже завжди використовується як спільна шкала часу. Згідно з [6], значення псевдодальності ми називаємо

$$Pi = cH\,[t(T2) - ti(T1)] \qquad (1.3)$$

де:

с - швидкість світла у вакуумі

t(T2) - час, необхідний годиннику приймача для отримання сигналу від приймача

ti (T1) - час, коли сигнал супутника був відправлений годинником супутника номер і

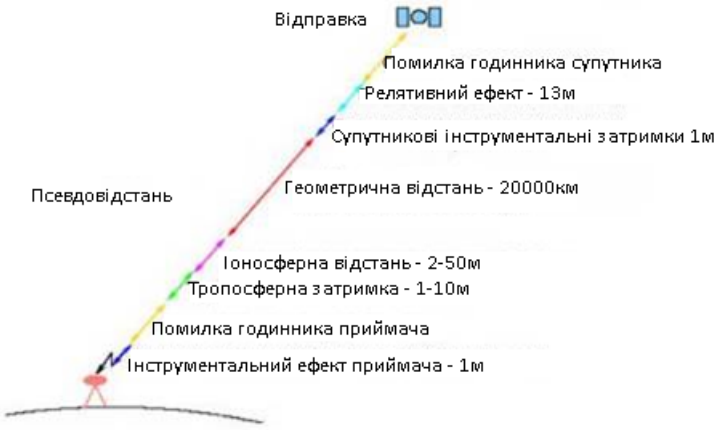


Рис. 1.3: Помилки та їх послідовність

Під час визначення відстані від і-го супутника до приймача псевдодальність також включає такі похибки: похибки синхронізації годинників супутника та приймача, затримки атмосферних сигналів (тропосфери та іоносфери), релятивістські ефекти, затримки приладів, інтерференція типу "перезображення" (багатопроменеве поширення), інтерференція тощо.

Враховуючи вищесказане, перепишемо формулу (1.4) наступним чином:

$$P_i = p_i + c \text{ Ч } (dt - dt_i) + rel_i + T_i + a \text{ Ч } I_i + K_i + M_i + \varepsilon_i \qquad (1.4)$$

1.3. Метод корекції помилок

Існує кілька груп методів корекції помилок та затримок. Ці методи можна використовувати окремо або в різних комбінаціях.

По можливості, методи моделювання безпосередньо обчислюють конкретні помилки. Наступні помилки можна обчислити безпосередньо:

• Корекція координат за допомогою геофізичних моделей (припливи, рухи літосферних плит тощо)

• Корекція тропосферної затримки

• Врахування зміщення між фазовими центрами приймача та супутника

• Корекція помилок супутникових приладів (помилки можуть бути різними для кожного супутника)

• Корекція релятивістського ефекту

• Корекція гравітаційного впливу інших планет

Усі вищезазначені корекції реалізовані в бібліотеці GPSTk [4].

Двочастотний приймач – це приймач, який отримує однакові дані від супутників, але використовує різні частоти. Основним завданням двочастотного приймача є усунення іоносферної затримки. Це можливо, оскільки іоносферна затримка залежить від частоти, і ця залежність відома. Для двочастотних приймачів іоносферну затримку можна обчислити безпосередньо.

Однак, дводіапазонні приймачі занадто дорогі для цивільного використання.

Виправлення кожної з вищезазначених помилок видається дуже складним завданням. Був винайдений клас методів, які можуть усунути всі помилки (повні або часткові) одночасно - різницевий метод. При використанні цих методів положення хоча б однієї наземної станції має бути достатньо точним. Ці помилки можна усунути, зробивши емпіричне припущення, що чим ближче розташовані приймачі, тим вища кореляція помилок, зумовлених атмосферними затримками. Отже, відстань між приймачами зменшить кореляцію і, отже, точність позиціонування. Диференціальний метод застосовний для динамічного спостереження в радіусі від 20 до 50 кілометрів від базової станції для двочастотних приймачів, а одночастотний приймач застосовний для динамічного спостереження в радіусі 8 кілометрів від базової станції.

Розглянемо ідею диференціального методу на прикладі подвійного диференціального методу, припускаючи, що є лише одна наземна станція. Тому:

• Є приймач A - наземна станція з відомими координатами

• Є приймач B, координати якого потрібно знайти.

• Станція A та приймач B розташовані досить близько

• Кожен приймач спостерігається як супутником 0, так і супутником 1 одночасно (протягом кількох секунд)

Алгоритм подвійної різниці:

1. Побудуйте рівняння типу (2.2) для двох механізмів прийому для кожного супутника. Позначте відповідні псевдодальності як PA0, PA1, PB0, PB1.

2. Віднімiть рівняння для супутника 0: PA0 - PB0. Віднімiть рівняння для супутника 1: PA1 - PB1.

• Серед цих різниць мінімізуються наступні:

• Похибки годинника супутника

• Похибки, спричинені перешкодами від супутника A

• Часткова атмосферна похибка

3. Віднімiть отриману різницю: (PA0 - PB0) - (PA1 - PB1). Вони перетинаються тут.

• Похибка годинника приймача

• Похибки, спричинені перешкодами від приймача

4. Ми маємо нове рівняння майже без похибки.

Отже, диференціація, особливо подвійна диференціація, може миттєво усунути майже всі помилки. Однак ці методи вимагають наявності принаймні однієї наземної станції поблизу. Це не завжди можливо, тому ці методи не підходять для таких застосувань, як мобільне картографування. [1]

Високоточне точкове позиціонування (ВТП) – це метод позиціонування в системах супутникової навігації. Він належить до методу абсолютного позиціонування (тобто без використання наземних станцій). Його важливою особливістю є безпосереднє обчислення похибок. ВТП був спочатку розроблений у 1997 році для використання в дводіапазонних приймачах. З часом методи вимірювання вдосконалилися, і тепер можливо визначити положення 24-годинного статичного спостереження з похибкою близько 1 см [5]. ВТП вже являє собою клас методів, кожен з яких означає:

• Надає додаткові дані з точними координатами супутника

• Наявність даних про похибку супутникового годинника

• Повне або майже повне усунення іоносферної затримки

• Надає карти іоносфери та тропосфери

• Існує набір рішень, які можна усереднити за допомогою фільтра Калмана.

Для двочастотних приймачів у ВТП іоносферну затримку можна повністю усунути. Для одночастотних приймачів іоносферну затримку можна розрахувати за допомогою іоносферної карти IONEX [6].

Бібліотека GPSTk [4] реалізує PRR для двочастотних приймачів за статичних умов спостереження.

1.4. Фільтр Калмана

Фільтр Калмана – це ефективний (тобто робустний) рекурсивний фільтр, який оцінює вектор стану динамічної системи, використовуючи набір неповних та зашумлених вимірювань. Названий на честь Рудольфа Калмана.

Щоб дати повніше визначення фільтра Калмана, необхідно ввести деякі необхідні терміни та символи. Для деяких процесів існують фізичні моделі. Нехай цей процес також пов'язаний з деякими вимірюваннями. Фільтр Калмана припускає дискретний час. У k-й момент часу вектор стану xkЄRn виражається вектором стану xk-1 у (k-1)-й момент часу:

$$xk = Axk\text{-}1 + Buk\text{-}1 + wk\text{-}1 \qquad (1.5)$$

де:

A-nЄn - матриця моделі фізичного процесу;

B-nЄs - матриця керування процесом;

- u – вектор керування розмірністю s;

- w – n-вимірний вектор помилки, що представляє помилку у фізичній моделі

Ми також можемо отримати дані вимірювання в будь-який час. Зв'язок між вектором вимірювання zkЄRm та вектором стану такий:

$$Zk=Hxk+vk \qquad (1.6)$$

де: z – вектор розмірності m;

H-mЄn – матриця, що представляє зв'язок між вектором стану та вектором вимірювання;

v – вектор помилки розмірності m, що представляє помилку вимірювання.

Рівняння (1.6) визначає деякий фізичний процес, а вектор z – це вимірювання, яке ми можемо виконати за допомогою певного обладнання. Формула (1.4) виражає зв'язок між вектором стану та вектором вимірювання. Зауважимо, що в загальному випадку матриці A, B, H можуть залежати від k, тобто вони можуть залежати від k безпосередньо. Матриця B може бути ненульовою лише тоді, коли процес знаходиться під зовнішнім керуванням. Загалом, рівняння можуть бути нелінійними, але алгоритм Калмана завжди дозволяє їх лінеаризувати.

Вектори v та w розглядаються як випадкові величини з коваріаційними матрицями R та Q відповідно. Ці матриці також можуть залежати від часу k. Якщо компоненти вектора стану незалежні одна від одної, то матриця R буде діагональною, а i-й діагональний елемент представлятиме середньоквадратичну похибку xi. Те саме стосується матриці Q та вектора w.

Алгоритм Калмана – це наступний двокроковий алгоритм для обчислення стану динамічної системи:

• Крок 1 – прогнозування.

• Крок 2 – Корекція:

$$\widehat{x_k} = \widehat{x_k^-} + K_k(z_k - H\widehat{x_k^-}) \qquad (1.7)$$

Матриця K в алгоритмі Калмана називається матрицею коефіцієнта посилення Калмана.

Матриця P в алгоритмі Калмана – це матриця коваріаційної помилки вектора стану x. Це означає, що на кожному кроці k фільтр Калмана разом з вектором стану x_k забезпечує можливість оцінки точності цього стану.

Перше наближення в алгоритмі може бути будь-яким. Однак, як перший крок, ми повинні вказати матрицю P на додаток до вектора стану. Якщо перше наближення (початкові умови) точне, то матриця P має бути нульовою, і навпаки

– якщо наближення неточне, то відповідна коваріація вектора стану cov(xi,xj) буде великим числом.

З формули (1.6) зрозуміло, що першим кроком алгоритму Калмана є обчислення вектора стану в наступний момент часу, спираючись лише на фізичну модель. Але на другому кроці алгоритму (1.7) вектор стану модифікується на основі результатів вимірювання. Наступна теорема обговорює оптимальність вибору матриці Калмана.

Матриця K мінімізує суму квадратів математичного очікування помилок оціненого вектора стану.

Фільтр Калмана є різновидом рекурсивних фільтрів . Для обчислення оцінки стану системи на поточний такт роботи йому потрібна оцінка стану у вигляді оцінки стану системи та оцінки похибки визначення цього стану на попередньому такті роботи та вимірювання на поточному такті. Ця властивість відрізняє його від пакетних фільтрів, що вимагають у поточний такт роботи знання історії вимірювань та/або оцінок. Далі під записомx^n|m розумітимемо оцінку істинного вектораху момент n з урахуванням вимірів з початку роботи і з момент m включно.

Стан фільтра визначається двома змінними:

$\widehat{x_k}$- апостеріорна оцінка стану об'єкта в момент k, отримана за результатами спостережень аж до моменту k включно часто позначається $\widehat{x_k}$, де $\widehat{x_k}$ означає "оцінка", а k - номер такту, на якому вона отримана;

$\frac{P_k}{k}$ — апостеріорна коваріаційна матриця помилок, що задає оцінку точності отриманої оцінки вектора стану і включає оцінку дисперсій похибки обчисленого стану і коваріації, що показують виявлені взаємозв'язки між параметрами стану системи часто позначається $\widehat{\mathrm{D}_k}$.

Кожна ітерація фільтра Калмана поділяється на дві фази: екстраполяція та корекція. Під час екстраполяції фільтр отримує попередню оцінку стану системи $\widehat{x_k}$ на поточний крок за підсумковою оцінкою стану з попереднього кроку. Цю

попередню оцінку також називають апріорною оцінкою стану, тому що для її отримання не використовуються спостереження відповідного кроку. У фазі корекції апріорна екстраполяція доповнюється відповідними поточними вимірами корекції оцінки. Скоригована оцінка також називається апостеріорною оцінкою стану, або просто оцінкою вектора стану $\widehat{x_k}$. Зазвичай ці дві фази чергуються: екстраполяція проводиться у разі корекції до наступного спостереження, а корекція проводиться разом із доступними наступному етапі спостереженнями. Однак можливий і інший розвиток подій: якщо з певної причини спостереження виявилося недоступним, то етап корекції може бути пропущений та виконано екстраполяцію за некоригованою оцінкою апріорною екстраполяцією. Аналогічно, якщо незалежні виміри доступні лише в окремі такти роботи, все одно можливі корекції зазвичай з використанням іншої матриці спостережень H_k .

1.5. Висновок до розділу 1

Фільтр Калмана – це потужний інструмент фільтрації даних. Основний принцип полягає у використанні фізичної інформації про саме явище для фільтрації. Зокрема, якщо ви фільтруєте дані спідометра автомобіля, у вас є підстави вважати, що різкі зміни швидкості зумовлені похибками вимірювання, спричиненими інерцією автомобіля. Фільтр Калмана привабливий тим, що він, у певному сенсі, є найкращим фільтром.

Фільтри, особливо фільтри Калмана, є незамінними інструментами для забезпечення найкращої точності та стабільності результатів вимірювань. Під час використання алгоритмів фільтрації важливо дотримуватися балансу між точністю, найкращими результатами вимірювань та використанням ресурсів мікроконтролера, оскільки різні фільтри вимагають специфічних обчислень, що може призвести до небажаних наслідків, якщо їх неправильно налаштувати або

використовувати. Але для кожного завдання слід знайти та реалізувати правильний фільтр, враховуючи вищезазначені фактори.

2. АНАЛІЗ СИСТЕМИ СУПУТНИКОВОЇ НАВІГАЦІЇ

2.1. Модулі системи супутникової навігації наступного покоління

L76 та L70 – це найновіші модулі GNSS, що включають усі найновіші та найдосконаліші технології.

Рис. 2.1. Модуль GNSS L70

Таблиця 2.1. Технічні характеристики модуля

Характеристики модулів		L70	L76
Живлення		2.8V~4.3В	3.1V~4.3В
Струмоспоживання	Виявлення	21mA@ 3.3В	25mA@ 3.3В
	Стеження	18mA@ 3.3В	18mA@ 3.3В
Чутливість	Виявлення	-148дБм	-147 дБм
	Стеження	-163 дБм	-160 дБм
	Повторне виявлення	-160 дБм	-160 дБм
TTFF@ 130дБм	Гарячий старт	<1с	<1с
	Теплий старт	<5с (EASY™)	<5с (EASY™)
	Холодний старт	<15с (EASY™)	<15с (EASY™)
Точне позиціонування		2.5м	2.5м
Точність часу синхронізації	1PPS	10нс	10нс
Частота оновлення даних		До 10Гц	До 10Гц

L76 — один із найменших модулів GNSS у світі. Його розміри становлять 10,1x9,7x2,5 мм.

Цей модуль встановлено на друкованій платі поверхневого монтажу. Він має загалом 18 контактів (1,1 мм), а саме:

Рис. 2.2. Контакт модуля L76

Розмір модуля:

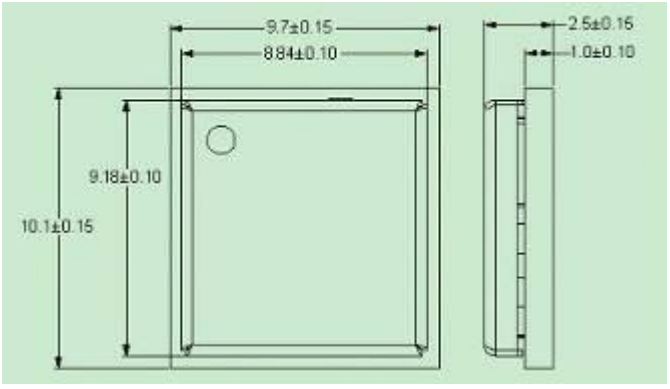


Рис. 2.3. Розмір модуля L76

L76 базується на однокристальній мікросхемі GNSS, яка включає радіосекцію, секцію базової смуги, фільтр SAW та температурно-компенсований кварцовий генератор (ТККГ).

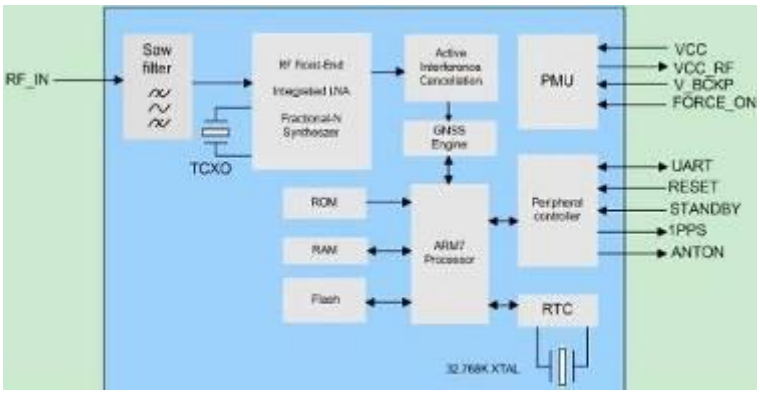


Рис. 2.4. Однокристальна мікросхема системи супутникової навігації

Живлення модуля: Контакт ТККГ використовується для живлення секції базової смуги, радіосекції, вводу/виводу та годинника реального часу ТККГ. Струм навантаження змінюється залежно від рівня ТККГ, навантаження процесора та кількості супутників, що відстежуються. Для контакту ТККГ типовий піковий струм у режимі збору даних про супутники становить 40 мА. Отже, сигнал, що подається на висновок ТККГ, повинен мати стабільні та високоякісні характеристики. Пульсації напруги живлення повинні відповідати наступним вимогам: максимум 54 мВ (RMS) при F = 0,3 МГц та максимум 15 мВ (RMS) при F>3 МГц.

Вивід V_BCKP забезпечує живлення для RTC. Для V_BCKP рекомендується пара конденсаторів ємністю 4,7 мкФ та 100 нФ. Діапазон

напруги RTC становить від 1,5 В до 4,5 В. Для досягнення швидкого TTFF (часу до першої фіксації) RTC повинен отримувати безперервне живлення. Пам'ять SRAM, яка також потребує безперервного живлення, є частиною RTC та містить всю інформацію, необхідну для швидкого запуску GPS, включаючи деякі налаштування користувача.

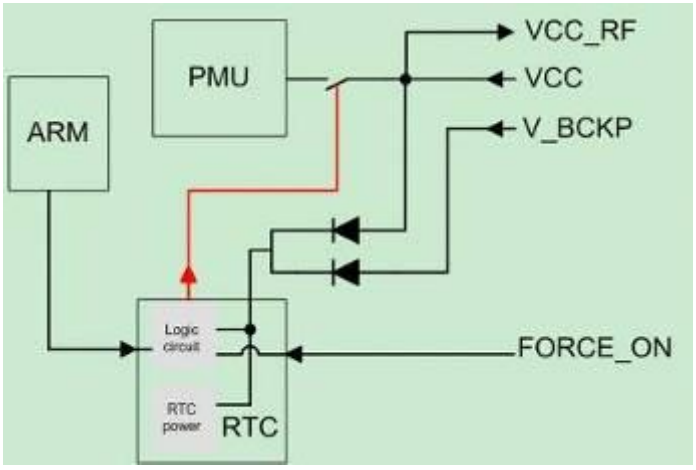


Рис. 2.5. Принципова схема розподілу живлення всередині модуля

Залежно від напруги живлення GNSS можна визначити середній струм та споживання енергії. Випробування проводилися на відкритому повітрі з використанням супутників GPS.

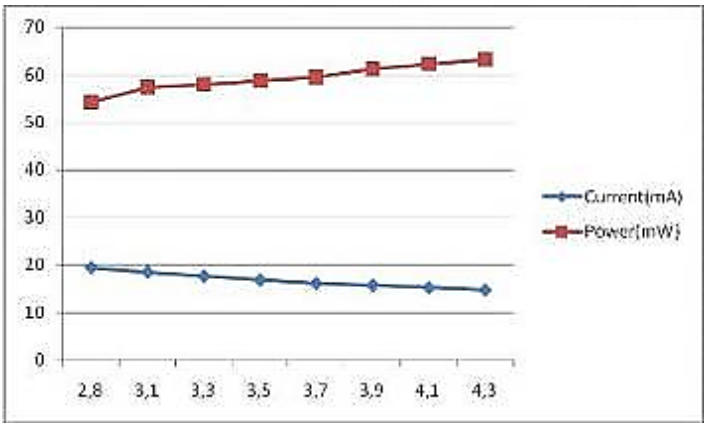


Рис. 2.6. Зв'язок між середнім струмом та споживанням потужності та напругою живлення модуля системи супутникової навігації

Перелік випробувань, проведених для визначення надійності модуля:

Таблиця 2.1. Перелік випробувань модуля системи супутникової навігації

Випробуваний виріб	Стан	Стандартні дані
Термічний шок	-30єС…+80єС, 144 cycles	GB/T 2423.22-2002 Test Na IEC 68-2-14 Na
Вологе тепло, циклічне	+55єС; >90% Rh 6 cycles for 144 hours	IEC 68-2-30 Db Test

Вібраційний шок	5~20Hz, 0.96m^2/s^3; 20-500Hz, 0.96m^2/s^3-3dB/oct, 1hour/axis; no function	2423.13 - 1997 Test Fdb IEC 68-2-36 Fdb Test
Випробування на нагрівання	85єC, 2 hours, Operational	GB/T 2423.1-2001Ab IEC 68-2-1 Test
Випробування на холод	-40єC, 2 hours, Operational	GB/T 2423.1-2001Ab IEC 68-2-1 Test
Витримування в теплі	90єC, 72 hours, Non-Operational	GB/T 2423.2-2001Ab IEC 68-2-2 Test
Витримування в холоді	-45єC, 72 hours, Non-Operational	GB/T 2423.2-2001Ab IEC 68-2-2 Test

Однією з головних переваг модуля L76 є можливість одночасного використання двох систем ГЛОНАСС та GPS, оскільки діапазон L1 може приймати сигнали як від GPS. Одночасне використання GPS ефективно подвоює кількість видимих супутників. Цей модуль дозволяє використовувати 33 канали відстеження, 99 каналів режиму виявлення та 210 каналів PRN. L76 може відстежувати будь-яку комбінацію сигналів GPS та ГЛОНАСС. L76 має наднизьке енергоспоживання 18 мА в режимі відстеження. За тих самих умов середнє енергоспоживання інших модулів на ринку становить близько 55 мА.

Цей модуль використовує найсучаснішу технологію AGPS EASY™ для скорочення часу гарячого запуску до 1 секунди. Технологія AlwaysLocate™ значно покращує точність позиціонування. Найнижче середнє енергоспоживання в режимі безперервного позиціонування становить лише 3 мВт.

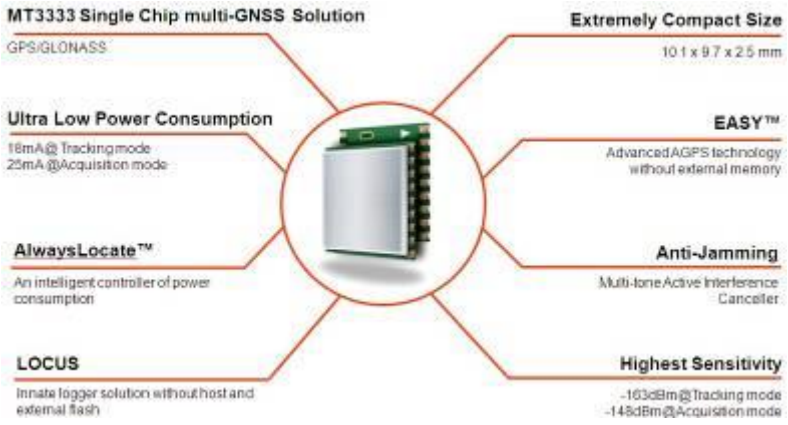


Рис. 2.7. Функції, що підтримуються L76

L70 – це ще один модуль Quectel GNSS, який є одним з лідерів ринку завдяки своїм функціям. L70 підтримує всі нові функції L76 та пропонує високу

продуктивність з точки зору чутливості, швидкості, енергоспоживання та сумісності з GSM-модемами Quectel. Додаткова вбудована функція реєстратора під назвою LOCUS дозволяє модулю L70 зберігати інформацію про місцезнаходження у внутрішній флеш-пам'яті з інтервалом у 15 секунд (за замовчуванням), створюючи журнал понад 16 годин. L76 та L70 підтримують такі функції:

• EASY™, самогенероване прогнозування ефемерид для швидкого позиціонування

• Дуже низьке енергоспоживання - 18 мА в режимі @tracking

• AlwaysLocate™, інтелектуальний алгоритм енергозбереження

• LOCUS, вбудований логер, немає потреби в підключенні зовнішньої пам'яті

• Висока чутливість: -163 дБм відстеження, -148 дБм прийом

• Підтримка DGPS, QZSS, SBAS (WASS/EGNOS/MSAS/GAGAN)

• Захист від глушіння, багатотонове активне шумозаглушення

• Технологія Self-AGPSEASY™

EASY™ - це абревіатура від так званої вбудованої системи допомоги позиціонуванню. Завдяки Easy™, GPS-процесор автоматично обчислює та прогнозує одну ефемериду (до 3 днів) при ввімкненні та зберігає її в енергонезалежній пам'яті. GPS-процесор може використовувати цю інформацію для визначення положення, коли інформації про супутники недостатньо. Ця функція може допомогти покращити позиціонування та зменшити TTFF у приміщенні та міських умовах.

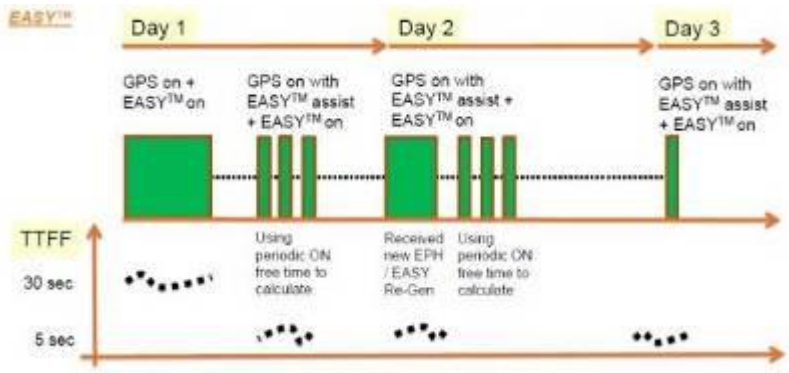


Рис. 2.8. Переваги EASY™

Періодичний режим сну дозволяє регулярно контролювати час увімкнення/вимкнення, щоб зменшити середнє споживання енергії. Час увімкнення/вимкнення можна встановити за допомогою інструкцій PMTK, будь ласка, зверніться до рис. 2.9 для отримання детальної інформації. Періодичний режим сну можна ввімкнути, надіславши таку команду PMTK: $PMTK255, тип, час роботи, час сну, другий час роботи, другий час сну

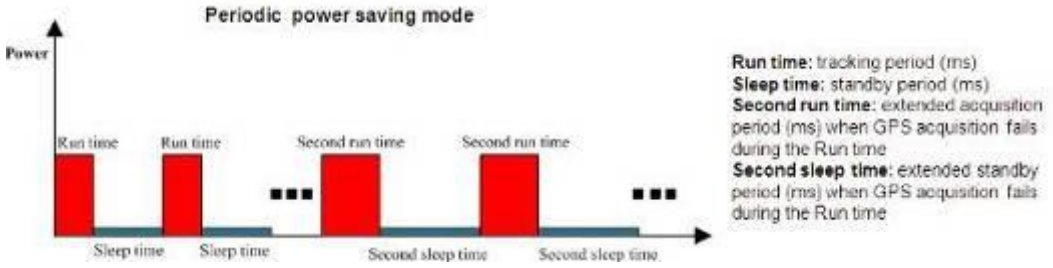


Рис. 2.9. Налаштування часу увімкнення/вимкнення

Зазвичай модуль GNSS переходить у періодичний режим після успішного позиціонування, але він також може перейти в цей режим, навіть якщо позиціонування не вдається.

Якщо позиція не зафіксована в першому робочому циклі, рекомендується встановити довший інтервал у другому робочому циклі, щоб забезпечити успішне повторне блокування.

AlwaysLocate™ – це інтелектуальний контролер для періодичних операцій. L76 та L70 можуть адаптивно регулювати час увімкнення/вимкнення відповідно до зовнішніх умов та моделей руху для досягнення балансу між точністю позиціонування та споживанням енергії.

Таким чином, середнє споживання енергії в режимі енергозбереження AlwaysLocate™ нижче, ніж у режимі періодичного енергозбереження. Типова середня потужність становить 3 МВт.

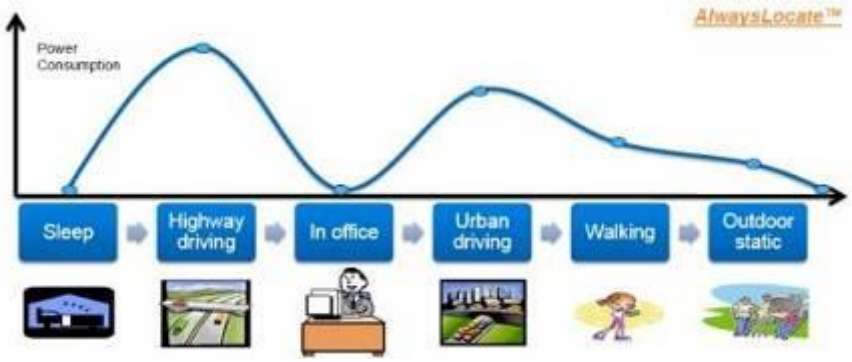


Рис. 2.10. Система AlwaysLocate™

Водночас модуль L76 також має інші режими роботи для зменшення споживання енергії.

Режим очікування – це режим низького споживання енергії. У режимі очікування ядро та контакти вводу/виводу залишаються активними, але радіо та кристал вимкнені. Модуль припиняє пошук супутників та навігацію. uaRT1 все ще може використовуватися для команд PMTK або будь-яких інших даних, але вихідні повідомлення NEMA не надсилатимуться.

Існує два способи входу та виходу з режиму очікування.

1) Використання контактів режиму очікування: Коли на контакти подається низька напруга, модуль переходить у режим очікування, а відсутність сигналу (контакти всередині модуля знаходяться у високому стані) відновлює повну функціональність модуля. Але завжди пам'ятайте, що замикання STANDBY на землю призведе до додаткового споживання струму, в цьому випадку до 600 мкА при VCC = 3,3 В.

2) Використовуйте команду PMTK: Надсилання команди PMTK "$PMTK161.0*28" переведе модуль у режим очікування. Надсилання будь-яких даних через UART1 розбудить модуль, оскільки uaRT1 все ще доступний. Після виходу з режиму очікування модуль використовуватиме всю внутрішню додаткову інформацію (наприклад, час GPS, ефемериди, останню позицію тощо), щоб мінімізувати TTFF під час гарячого або теплого запуску. У цей час споживання струму в режимі очікування становить близько 500 мкА при VCC = 3,3 В.

У режимі очікування модуль споживає менше енергії, ніж у режимі очікування. У цьому режимі модуль припиняє пошук та відстеження супутників. uaRT1 недоступний. Однак, живлення продовжує подаватись до енергонезалежної пам'яті RTC, яка містить всю необхідну інформацію GPS, необхідну для швидкого запуску, а також зберігає параметри конфігурації. Технологія EASY також зберігає ефемериди в цьому режимі. Споживання струму в цьому режимі становить приблизно 7 мкА.

Існує три способи входу та виходу з режиму очікування.

1) Надішліть команду: "$PMTK225,4*2F", щоб остаточно перейти в режим очікування. Єдиний спосіб увімкнути модуль - це подати високу напругу на FORCE_ON.

2) Надішліть команду: “$PMTK291,7.0,<час>,1*<контрольна сума>”. Модуль перейде в режим очікування на одну мілісекунду, а потім автоматично ввімкнеться. Тут максимальне значення часу досягає 518400000 секунд. Окрім режиму синхронного виходу, подача високої напруги на FORCE_ON також може ввімкнути модуль вищезазначеним способом.

3) Відключення джерел живлення VCC та V_BCKP переведе модуль у режим очікування. Якщо живлення буде знову подано на VCC, модуль негайно відновить повну функціональність. Однак цей метод не рекомендується [7].

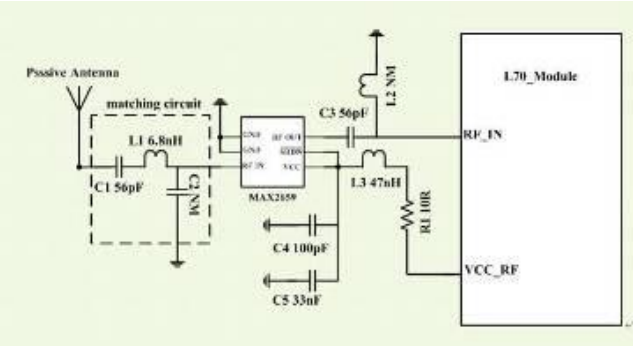


Рис. 2.11. Принципова схема модуля L70 та малопотужного підсилювача MAX2659

Модуль LEA-5 – це 50-канальний надчутливий GPS/GALILEO приймач, який не потребує жодних додаткових зовнішніх компонентів, окрім блоку живлення, антени та додаткового акумулятора для годинника реального часу для роботи. Перевагою цього модуля є використання технології SuperSense® від u-blox для внутрішніх GPS-систем, яка забезпечує дуже швидке відстеження навіть у густонаселених міських районах.

Рис. 2.12. Модулі LEA-5

Має флеш-пам'ять для зберігання налаштувань користувача. Вбудований малошумний підсилювач можна використовувати як з активними, так і з пасивними антенами. Модуль LEA-5S не має вбудованої флеш-пам'яті, а LEA-5A не підтримує режим KickStart.

2.2. Моделювання за фільтром Калмана

Позначимо за x_k величину, яку ми вимірюватимемо, а потім фільтруватимемо. Це може бути координата, швидкість, прискорення, вологість, температура, тиск і т.д.

Почнемо із простого прикладу, який і приведе нас до формулювання спільного завдання. Уявіть собі, що у нас є радіокерований об'єкт, що рухається, який може переміщатися тільки вперед і назад. Ми, знаючи вагу об'єкта, форму, покриття поверхні, якою він переміщається тощо, розрахували як контролюючий джойстик впливає швидкість руху v_k .

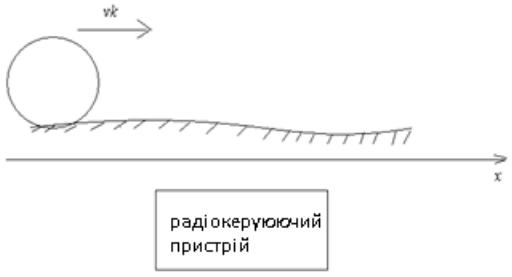


Рис. 2.13. Об'єкт, що рухається

Тоді координата об'єкта змінюватиметься згідно із законом:

$$x_{k+1} = x_k + v_k\,dt \qquad (2.1)$$

У реальному житті ми не можемо врахувати в наших розрахунках малі обурення, що діють на об'єкт (вітер, нерівності поверхні, перешкоди), тому справжня швидкість об'єкта буде відрізнятися від розрахункової. До правої частини написаного рівняння додасться випадкова величина $про_k$:

$x_{k+1} = x_k + v_k\,dt + {}_k$ *(2.2)*

У нас є встановлений на об'єкті GPS сенсор, який намагається міряти справжню координату x_k об'єкта, і, звичайно ж, не може її поміряти точно, а міряє з помилкою $з_k$, яка теж випадковою величиною.

У результаті з сенсора ми отримуємо помилкові дані:

$z\,k = x\,k + з\,k$ (2.3)

Завдання полягає в тому, що знаючи невірні показання сенсора z_k знайти хороше наближення для істинної координати об'єкта x_k. Це гарне наближення ми будемо позначати як .

У формулюванні ж загального завдання, за координату x_k може відповідати все, що завгодно (температура, вологість тощо), а член, який відповідає за контроль системи ззовні, ми позначимо за u_k (у прикладі с об'єктом $u_k = v_k\,dt$).

Рівняння для координати та показання сенсора виглядатимуть так:

$z_k = x_k + з_k$ *(2.4)*

І так, що нам відомо:

· U_k - це відома величина, яка контролює еволюцію системи. Ми її знаємо із побудованої нами фізичної моделі.

· Помилка моделі $про_k$ та помилка сенсора $з_k$ – випадкові величини. І закони розподілу не залежать від часу (від номера ітерації k).

· Середні значення помилок дорівнюють нулю: $Ео_k = Ез_k = 0$.

· Сам закон розподілу випадкових величин може бути нам і не відомий, але відомі їх дисперсії та . Зауважимо, що дисперсії не залежить від k , оскільки закони розподілу залежить від нього.

· Передбачається, що всі випадкові помилки незалежні одна від одної: яка помилка буде в момент часу k зовсім не залежить від помилки в інший момент часу k '.

Незайвим буде відзначити, що завдання фільтрації - це завдання згладжування. Ми не прагнемо згладжувати дані із сенсора, ми прагнемо отримати найближче значення до реальної координати x_k .

Ми будемо міркувати щодо індукції. Уявіть собі, що на k -му кроці ми вже знайшли відфільтроване значення із сенсора, яке добре наближає справжню координату системи x_k . Не забуваємо, що ми знаємо рівняння, яке контролює зміну нам невідомої координати:

$$x_{k+1} = x_k + u_k + o_k, \qquad (2.5)$$

тому, не отримуючи значення з сенсора, ми можемо припустити, що у кроці $k+1$ система еволюціонує відповідно до цього закону і сенсор покаже щось близьке до . На жаль, поки що ми не можемо сказати нічого більш точного. З іншого боку, на кроці $k+1$ у нас буде неточне показання сенсора z_{k+1} .

Ідея Калмана полягає в тому, щоб отримати найкраще наближення до істинної координати x_{k+1} , ми повинні вибрати золоту середину між показанням z_{k+1} неточного сенсора і - нашим передбаченням того, що ми очікували від нього побачити. Показанню сенсора ми дамо вагу K, а на передбачене значення залишиться вага $(1-K)$:

$$x_{k+1}^{opt} = K \times z_{k+1} + (1 - K) \times (x_k^{opt} + u_k) \qquad (2.6)$$

Коефіцієнт K називають коефіцієнтом Калмана. Він залежить від кроку ітерації, тому правильніше було б писати K_{k+1} , але поки, щоб не захаращувати формули розрахунків, ми опускатимемо його індекс.

Ми повинні вибрати коефіцієнт Калмана K таким, щоб оптимальне значення координати, що вийшло, було б найбільш близько до істинної координати x_{k+1} . Наприклад, якщо ми знаємо, що наш сенсор дуже точний, то ми більше довірятимемо його показанню і дамо значення z_{k+1} більше ваги (K близько одиниці). Якщо ж сенсор, навпаки, зовсім не точний, тоді більше орієнтуватимемося на теоретично передбачене значення.

У загальному випадку, щоб знайти точне значення коефіцієнта Калмана K потрібно просто мінімізувати помилку:

$$e_{k+1} = x_{k+1} - x_k^{opt} \quad (2.7)$$

Тепер потрібно розібратися, що таке мінімізувати помилку. Адже помилка, як ми бачимо, сама по собі є випадковою величиною і щоразу набуває різних значень.

Насправді немає однозначного підходу до визначення те, що означає, що помилка мінімальна.

Так само як і у випадку з дисперсією випадкової величини, коли ми намагалися оцінити характерну ширину її розкиду, так і тут виберемо найпростіший для розрахунків критерій. Ми мінімізуватимемо середнє значення від квадрата помилки.

З того що всі випадкові величини, що входять у вираз (2.7) для e_{k+1}, незалежні, випливає, що всі "перехресні" члени дорівнюють нулю $E(k_k+1)E(e_k з_{k+1}) = 0$.

Також $Eз_{k+1} = Eo_k = 0$ тоді формула для дисперсій виглядає набагато простіше.

Вираз (2.7) набуває мінімального значення, коли (прирівнюємо похідну до нуля):

$$K_{k+1} = \frac{Ee_k^2 + \sigma_\delta^2}{Ee_k^2 + \sigma_\delta^2 + \sigma_\eta^2} \quad (2.8)$$

Підставляємо у вираз (2.8) для середньоквадратичної помилки мінімізуюче значення коефіцієнта Калмана (2.9) . Отримуємо:

$$E(e_{k+1}^2) = \frac{\sigma_\eta^2(Ee_k^2 + \sigma_\delta^2)}{Ee_k^2 + \sigma_\delta^2 + \sigma_\eta^2} \quad (2.9)$$

Наше завдання вирішено. Ми отримали ітераційну формулу (2.9) для обчислення коефіцієнта Калмана. [9]

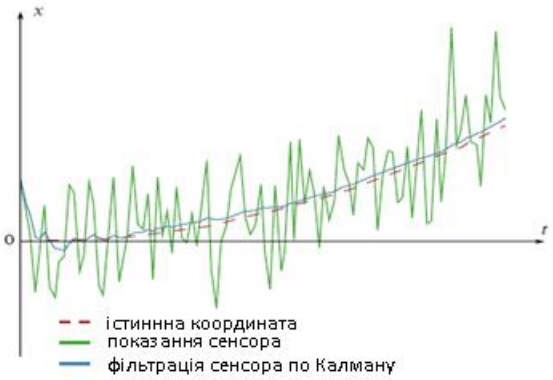


Рис. 2.14. Дані оброблені фільтром Калмана

Якщо простежити, як із кроком ітерації k змінюється коефіцієнт Калмана K_k, можна показати, що він завжди стабілізується до певного значення K_{stab}. Зокрема, коли середньоквадратичні помилки сенсора і моделі ставляться один до одного як десять до одного, графік коефіцієнта Калмана в залежності від кроку ітерації виглядає так:

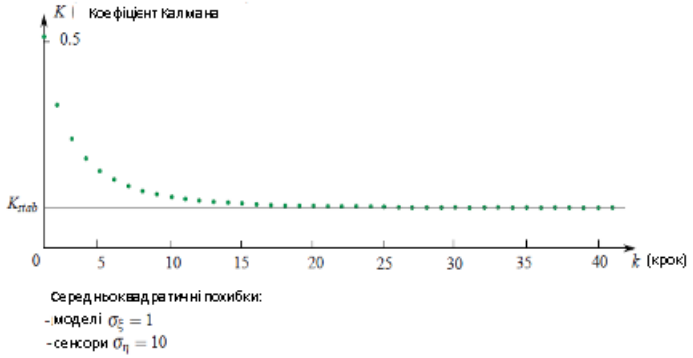


Рис. 2.15. Коефіцієнти Калмана як функція кроку ітерації

Основна ідея фільтра Калмана полягає в тому, щоб знайти коефіцієнт K такий, щоб відфільтроване значення:

$$x_k^{opt} = Kz_{k+1} + (1 - K)(x_k^{opt} + u_k) \qquad (2.10)$$

У середньому координата xk+1 має найменше відхилення від фактичного значення. Ми знаходимо, що відфільтроване значення є лінійною функцією показань датчика zk+1 та попереднього відфільтрованого значення. Попереднє відфільтроване значення є лінійною функцією показань датчика zk та попереднього відфільтрованого значення. І так далі, доки весь ланцюжок фільтрів не буде повністю розгорнутий. Іншими словами, відфільтроване значення лінійно пов'язане з усіма попередніми показаннями датчика:

$$x_k^{opt} = \lambda + \lambda_0 z_0 + \cdots + \lambda_{k+1} z_{k+1} \quad (2.11)$$

Ось чому фільтр Калмана називають лінійним фільтром.

2.3. Простий фільтр Калмана

Основний принцип полягає в тому, що фільтр враховує фізичну модель системи та інформацію з попередніх вимірювань для прогнозування. Він рекурсивно оновлює оцінку стану системи, тим самим зменшуючи вплив шуму та підвищуючи точність наступних вимірювань.

Цей фільтр часто використовується в галузі вимірювань, включаючи навігацію (для фільтрації початкових значень та отримання більш стабільних результатів). Варто зазначити, що цей фільтр є спрощеною його версією, але існують й інші модифіковані версії, такі як «Розширений фільтр Калмана».

Цей фільтр містить такі коефіцієнти та змінні:

1. P – коваріація помилки прогнозування, яка відображає точність прогнозування стану системи на наступному кроці вимірювання. Це значення оновлюється після кожного кроку прогнозування;

2. Xe – оцінка поточного стану системи, яка оновлюється на основі виміряних та прогнозованих значень; 3. Pc – коваріація помилки прогнозування, яка використовується для подальшого оновлення значення P;

4. G – коефіцієнт Калмана, який використовується для визначення ступеня врахування виміряного значення під час оновлення оцінки стану;

5. Zp – виміряне значення, передбачене для наступного кроку, яке використовується для порівняння з фактичним виміряним значенням;

6. Xp – стан системи, передбачений для наступного кроку, який використовується для подальшого оновлення значення Xe;

7. Початкове значення вимірювання;

8. Значення швидкості відгуку фільтра;

9. Середнє значення відхилення.

Після короткого ознайомлення з суттю фільтра можна інтуїтивно зрозуміти, як алгоритм обробляє рукописні дані. Для цієї мети я використовував Python (JupyterLab в Anaconda), оскільки Python надає багато функцій візуалізації та аналізу даних, враховуючи завантаження даних вимірювань з файлів .csv, що я особисто реалізував за допомогою Raspberry Pi Pico.

На зображенні нижче показано клас фільтра, де слід зазначити, що цей клас також можна запускати на Raspberry Pi Pico.

,

[5]:
```
#Базовий клас фільтра Калмана

class KalmanFilter:
    def __init__(self, initial_value, speed_of_reaction_to_change, average_deviation_value):
        self.P = 1.0 #Початкове значення варіації помилки
        self.Xe = initial_value #Початкове значення фільтра
        self.speed_of_reaction_to_change = speed_of_reaction_to_change
        self.average_deviation_value = average_deviation_value

    def filtration_process(self, value):
        #Оновлення прогнозу коваріації помилки з урахуванням шуму процесу
        self.Pc = self.P + self.speed_of_reaction_to_change

        #Обчислення Калманівського коефіцієнта
        self.G = self.Pc / (self.Pc + self.average_deviation_value)

        #Оновлення коваріації помилки на основі Калманівського коефіцієнта
        self.P = (1 - self.G) * self.Pc

        self.Xp = self.Xe #Збереження поточної оцінки стану для прогнозу
        self.Zp = self.Xp #Збереження прогнозованого стану

        #Оновлення оцінки стану на основі різниці між новим вимірюванням та прогнозом,
        #зваженої Калманівським коефіцієнтом
        self.Xe = self.G * (value - self.Zp) + self.Xp

        return self.Xe
```

Рис. 2.16. Клас фільтра Калмана на Python

Цей клас містить конструктор ініціалізації __init__, який визначає ключові вхідні дані, які потрібно встановити, а саме: початкове вхідне вимірювання, чутливість фільтра та середнє відхилення. Важливо зазначити, що середнє відхилення обчислюється шляхом проведення певної кількості тестових вимірювань перед оголошенням екземпляра класу фільтра. Вибір правильного параметра чутливості також важливий, але це значення буде змінюватися від випадку до випадку (я тестував значення від 0,01 (повільно) до 0,1 (швидко)).

Наступний крок – використання користувацької функції (див. Рисунок 2) для отримання масиву випадкових значень у заданому діапазоні (у цьому випадку я взяв 250 значень у діапазоні від 10 до 40).

```
[80]: input_value_chanel_X = random_input_value_array_gen(250, 10, 40) #Масив вхідних значень X
```

Рис. 2.17. Отримання масиву випадкових значень для тестування

Наступний крок– створення екземпляра класу та встановлення параметрів фільтра, після чого процес фільтрації зациклюється. Тут він зациклюється, оскільки це симуляція вимірювання.

```
[23]: #Екземпляр класу фільтра каналу X
kalman_filter_channel_X = KalmanFilter(initial_value = initial_input_value_chanel_X,
                                       speed_of_reaction_to_change = speed_of_reaction_to_change,
                                       average_deviation_value = average_deviation_value_X)

[25]: #ПРОЦЕС ФІЛЬТРАЦІЇ ВХІДНИХ ДАНИХ

[27]: # Застосування фільтра Калмана до даних (на тестових даних)

#Канал X
for value in input_value_chanel_X:
    filtered_value = kalman_filter_channel_X.filtration_process(value)
    filtered_values_chanel_X.append(filtered_value)
```

Рис. 2.18 Приклад процесу створення категорії та фільтрації

Візуалізуючи це, ми отримуємо наступний рисунок, на якому показано вхідні зашумлені дані (жовтий графік) та ті ж дані після застосування фільтра Калмана (синій графік). Для кращого розуміння я додав пунктирну лінію посередині.

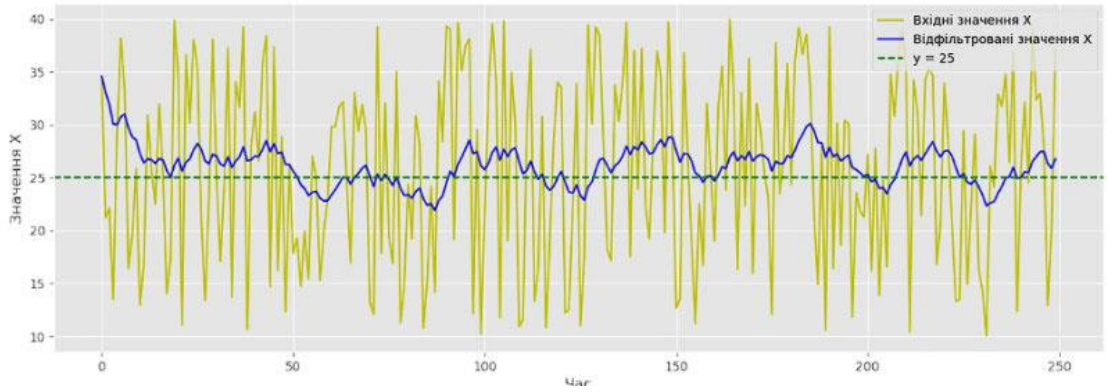


Рис. 2.19. Візуалізація процесу фільтраціїм згідно простого фільтрк Калмана

Зрозуміло, що фільтр значно згладив зашумлені дані, в цьому випадку його коливання знаходяться приблизно на рівні центральної лінії.

Після того, як ви ознайомитеся з алгоритмом, ви можете спробувати адаптувати фільтр до платформи Arduino та використовувати його для фільтрації вимірювань у режимі реального часу. У цьому випадку я використовував датчик тиску BMP280. Усі вихідні файли програми додані в один файл.

Щоб спростити структуру програми, я вирішив відокремити алгоритм фільтрації та створити бібліотеку, що містить клас фільтра Калмана.

```h
#ifndef KALMANFILTER_H
#define KALMANFILTER_H
#include <Arduino.h>

class KalmanFilter {
  private:
    float m_P; // Коваріація помилки прогнозу
    float m_Xe; // Оцінка поточного стану системи
    float m_Pc; // Прогнозована коваріація помилки
    float m_G; // Коефіцієнт Калмана
    float m_Zp; // Прогнозоване вимірювання наступного кроку
    float m_Xp; // Прогнозований стан системи наступного кроку
    float m_speedOfReaction; // Значення швидкості реакції
    float m_averageDeviationValue; // Значення середнього відхилення

  public:
    void UpdateCoefficients(float initValue, float speedOfReaction, float averageDeviationValue);
    float FiltrationProcess(float inputValue);
};

#endif
```

Рис. 2.20. Файл бібліотеки фільтра Калмана .h

```cpp
#include "KalmanFilter.h"

//--------------------------------------------------------------------------------------------

// Функція оновлення коефіцієнтів
void KalmanFilter::UpdateCoefficients(float initValue, float speedOfReaction, float averageDeviationValue) {
  m_Xe = initValue;
  m_P = 1.0;
  m_speedOfReaction = speedOfReaction;
  m_averageDeviationValue = averageDeviationValue;
}

// Функція фільтрації
float KalmanFilter::FiltrationProcess(float inputValue) {
  m_Pc = m_P + m_speedOfReaction;                 // Оновлення прогнозу коваріації помилки з урахуванням шуму процесу
  m_G = m_Pc / (m_Pc + m_averageDeviationValue);  // Обчислення Калманівського коефіцієнта
  m_P = (1 - m_G) * m_Pc;                         // Оновлення коваріації помилки на основі Калманівського коефіцієнта
  m_Xp = m_Xe;                                    // Збереження поточної оцінки стану для прогнозу
  m_Zp = m_Xp;                                    // Збереження прогнозованого стану
  m_Xe = m_G * (inputValue - m_Zp) + m_Xp;
  return m_Xe;
}

//--------------------------------------------------------------------------------------------
```

Рис. 2.21. Файл бібліотеки фільтра Калмана .cpp

Цей клас є повною адаптацією алгоритму, який я раніше демонстрував у JupyterLab на прикладі тестових даних, але написаний на C++. Я хотів би виділити одну відмінність від попередньої версії: замість використання

конструктора класу нами створено окрему функцію для оновлення коефіцієнтів, яку можна запускати за потреби.

Також створено бібліотеку, що містить версію функції, яку я написав для обчислення середнього відхилення на основі даних вимірювань, записаних в одновимірному масиві.

Наступний крок – додати бібліотеку до проекту, створити об'єкт класу та визначити значення швидкості відгуку фільтра для спрощення процесу зміни значення.

```
4   #include "KalmanFilter.h"
5   #include "AverageDeviation.h"
6
7   #define BMP280_ADDRESS 0x76 // Адреса BMP280
8   #define SPEED_OF_REACTION 0.01 // Швидкість реакції для фільтра Калмана
9
10  Adafruit_BMP280 bmp;  // Оголошення об'єкта BMP280 I2C
11  KalmanFilter PressureFilter; //Оголошення об'єкта класа фільтра Калмана
```

Рис. 2.22. Створення екземпляра класу фільтра

Наступний крок для фільтра – обчислити середнє відхилення та встановити перший параметр у функції Setup.

```
46  //Середнє відхилення для фільтра калмана
47  for(int i = 0; i < maxIteration; i++) {
48    readPressureValueArray[i] = (bmp.readPressure() - averageDeviationOffset);
49    delayMicroseconds(2000);
50  }
51
52  averageDeviationFilter = AverageDeviationValueCalc(readPressureValueArray, maxIteration);
53  readPressureValue = bmp.readPressure() - averageDeviationOffset;
54  PressureFilter.UpdateCoefficients(readPressureValue, SPEED_OF_REACTION, averageDeviationFilter);
55 }
```

Рис. 2.23. Налаштування фільтра Калмана

Останній крок – застосувати фільтр до даних вимірювання. Зображення до та після фільтрації відображаються на графіку. Значення затримки встановлюється для кращої візуалізації значення затримки можна і потрібно змінювати за потреби.

```
57  void loop() {
58    readPressureValue = (bmp.readPressure() - averageDeviationOffset);
59    filtredPressureValue = PressureFilter.FiltrationProcess(readPressureValue);
60    Serial.print(readPressureValue);
61    Serial.print(",");
62    Serial.println(filtredPressureValue);
63    delay(50);
64  }
```

Рис. 2.24. Застосування фільтра Калмана

Ця інструкція показує лише використання фільтра Калмана та не показує всі аспекти налаштування вимірювання.

Наступний крок – візуалізація результатів вимірювання та фільтрації в режимі реального часу. Для візуалізації ми використовуємо структуру виведення інформації та часову діаграму.

На рисунку нижче показано процес вимірювання та фільтрації для SPEED_OF_REACTION = 0,05.

Рис. 2.25. Візуалізація вимірювання та фільтрації без зовнішнього впливу

Рис. 2.26. Візуалізація вимірювання та фільтрації із зовнішнім впливом

Цей рисунок показує подібний ефект до тестових даних, але варто зазначити, що він показує хід зміни значення через тимчасовий зовнішній вплив.

Коли зовнішній вплив збільшується, збільшення значення після фільтрації є плавнішим, ніж коли воно не фільтрується; подібний ефект спостерігається, коли значення зменшується, коли зовнішній вплив зменшується. Тому значення не стрибає і не падає різко, що дуже корисно для систем, чутливих до таких різких стрибків.

Ми також візуалізували два фільтри Калмана з різними значеннями швидкості:

1. Помаранчевий - SPEED_OF_REACTION = 0,01;
2. Зелений - SPEED_OF_REACTION = 0,1.

Рис. 2.27. Візуалізація виміряних значень та 2 фільтрів без зовнішніх впливів

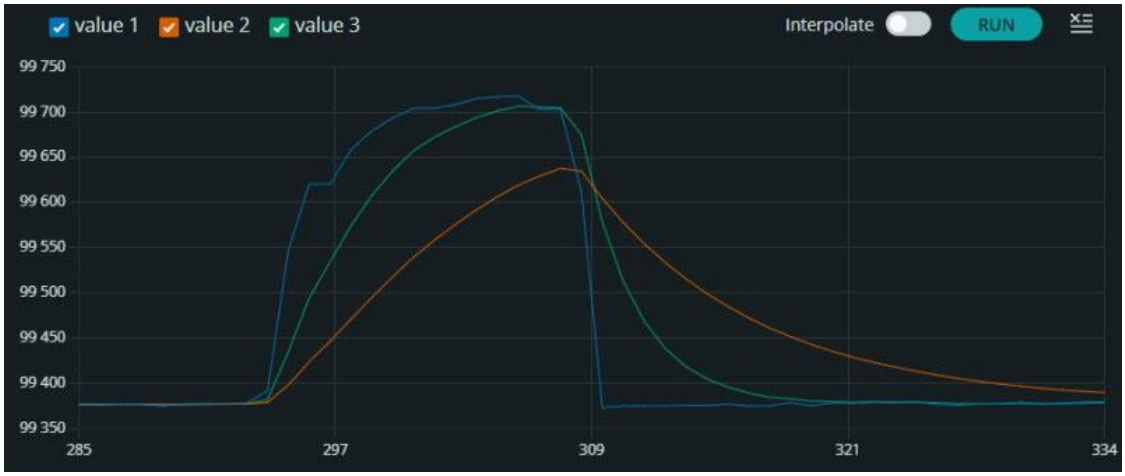


Рис. 2.28. Візуалізація виміряних значень та 2 фільтрів із зовнішніми впливами

Аналізуючи результати фільтрації для різних значень швидкості реакції, можна зробити висновок, що це значення має суттєвий вплив на початкові результати. Отже, можна зробити такі висновки:

- Графік з нижчим індексом продуктивності (помаранчевий) має плавніші зміни індикатора, але це також впливає на результати під зовнішніми впливами, оскільки значення зростають повільніше та довше падають;

- Графік з вищим індексом продуктивності (зелений) має менш плавні зміни індикатора (порівняно з помаранчевим), але швидше реагує на зміни, що підтверджується на другому рисунку. Водночас він також має хороший фільтруючий ефект.

2.4. Висновок до розділу 2

Фільтри, особливо фільтри Калмана, є незамінним інструментом для забезпечення найкращої точності та стабільності результатів вимірювань. Під час використання алгоритмів фільтрації найважливіше — дотримуватися балансу між точністю, найкращими результатами вимірювань та використанням ресурсів у порівнянні Arduino та Raspberry Pi Pico, оскільки різні фільтри вимагають специфічних розрахунків, що може призвести до небажаних наслідків, якщо їх не вказати або не використовувати належним чином. Але для

кожного завдання варто знайти правильний фільтр та реалізувати його, враховуючи вищезазначені аспекти.

Під час використання алгоритмів фільтрації найважливіше — дотримуватися балансу між точністю, найкращими результатами вимірювань та використанням ресурсів Raspberry Pi Pico, оскільки різні фільтри вимагають специфічних розрахунків, що може призвести до небажаних наслідків, якщо їх не вказати або не використовувати належним чином. Але для кожного завдання варто знайти правильний фільтр та реалізувати його, враховуючи вищезазначені аспекти.

3. РЕАЛІЗАЦІЯ АЛГОРИТМУ ФІЛЬТРАЦІЇ ТРАЄКТОРІЇ РУХУ GPS-МОДУЛЯ ЗА РЕЗУЛЬТАТАМИ ВИМІРЮВАННЯ КООРДИНАТ

3.1. Формування наборів результатів вимірювання координат із супутникових навігаційних систем

Фактична реалізація фільтра Калмана вимагає отримання супутникових даних. Для цього ми використовуватимемо модуль L76 та модуль LEA-5 для передачі даних супутників GPS)

Наступні внутрішні дані отримуються за допомогою модулів L76 та LEA-5.

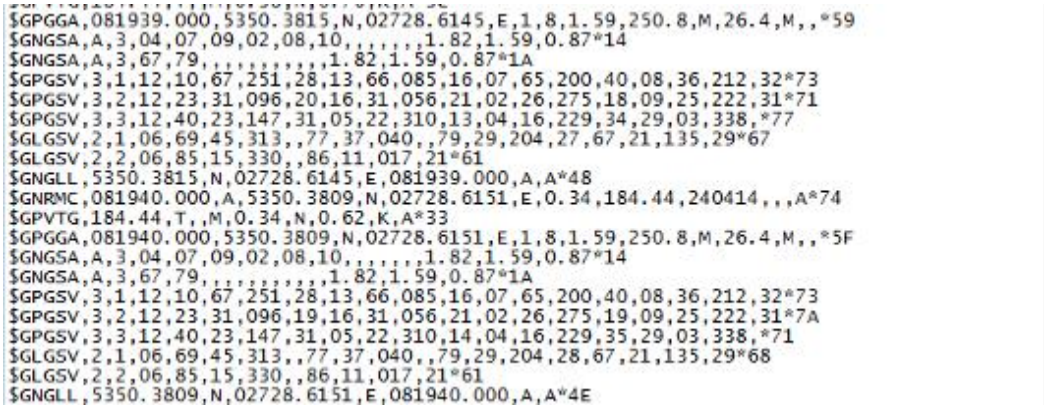


Рис. 3.1. Пакет даних, отриманий за допомогою модуля L76

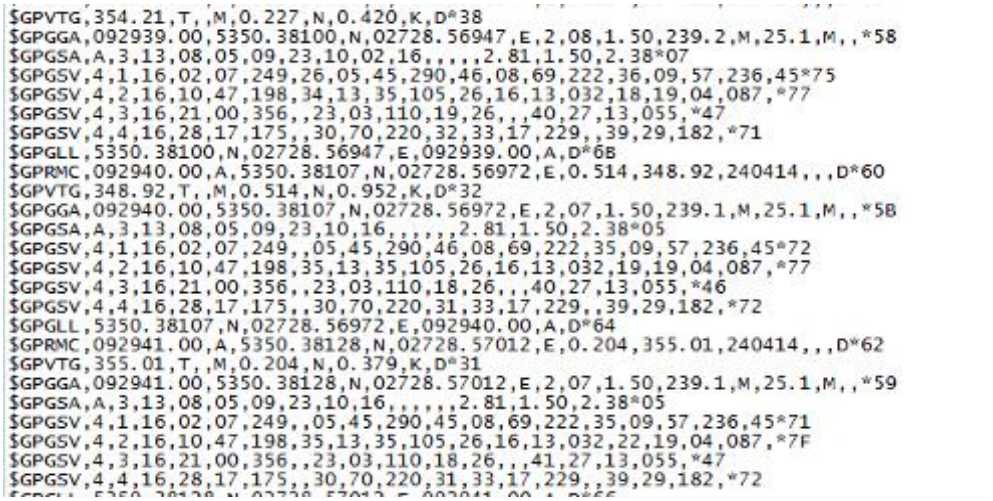


Рис. 3.2. Пакет даних, отриманий за допомогою модуля LEA-5

Дані отримуються від супутника у форматі NMEA. Це формат, який використовується для передачі повідомлень між бортовими приладами. Він містить систему повідомлень для обміну інформацією між GPS-навігаційними приймачами та споживачами навігаційної інформації. Усі команди та

повідомлення передаються у текстовому форматі ASCII, а команди та повідомлення, пов'язані з GPS-приймачами, починаються з $GP, а символи <CR><LF> повинні бути включені в кінець рядка повідомлення. Контрольну суму поточного повідомлення (починаючи з роздільника *) можна вказати в останньому полі повідомлення. 8-бітна контрольна сума (XOR) виконується для всіх символів між роздільниками $ та * у повідомленні, включаючи та виключаючи пробіли. Шістнадцятковий результат перетворюється на два символи ASCII (0-9, A-F).

Частина повідомлення протоколу NMEA:

$GPGGA містить положення GPS, час збору даних, якість даних, кількість використаних супутників, коефіцієнт горизонтального розпаду положення (HDOP), інформацію про корекцію диференціалу та вік.

$GPGLL містить широту та довготу GPS, а також час збору даних.

$GPGSA відображає режим приймача GPS, параметри супутника, що використовуються для вирішення навігаційної задачі. Результат відображається в повідомленні $GPGGA та значенні коефіцієнта точності положення.

$GPGSV вказує кількість видимих супутників, номер супутника, кут місця, кут азимута та співвідношення сигнал/шум для кожного супутника.

$GPRMC містить дані про час, місцезнаходження, курс та швидкість, надіслані приймачем навігації GPS. Це повідомлення повинно містити контрольну суму та надсилатися з інтервалом не більше 2 секунд. Усі поля даних мають бути готові, перш ніж самі дані можна буде використовувати. Недійсні поля можна використовувати, доки дані тимчасово не будуть готові.

$GPVTG містить поточний істинний курс (COG) та швидкість відносно землі (SOG).

$GPZDA містить час UTC, календарний день, місяць, рік та місцевий часовий пояс.

GGA - дані позиціонування GPS 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

$GPGGA, hhmmss.ss, 1111.11, a, yyyyy.yy, a, x, xx, x.x, xxx, M, x.x, M, x.x, xxxx*hh

1. Середній час за Гринвічем, коли було визначено місцезнаходження.

2. Широта місцезнаходження.

3. Північ/Південь (Пн/Пд).

4. Довгота місцезнаходження.

5. Південно-східний (Пн/З).

6. Індикатор якості сигналу GPS:

0 = немає фіксації або погана фіксація;

1 = Режим GPS нормальної точності, місцезнаходження можна зафіксувати;

2 = Диференціальний режим GPS, місцезнаходження можна зафіксувати;

3 = Режим GPS високої точності, місцезнаходження можна зафіксувати.

7. Кількість супутників, що використовуються (00-12, може відрізнятися від показаної кількості).

8. HDOP – максимальна висота над/нижче рівня моря.

9. Висота приймальної антени над/нижче рівня моря.

10. Висота антени в метрах.

11. Похибка геоїда - різниця між еліпсоїдом Землі WGS-84 та рівнем моря, "-" = рівень моря нижче еліпсоїда.

12. Похибка в метрах.

13. Вік диференціальних даних GPS – час у секундах з моменту останнього оновлення SC104 типу 1 або 9, доповнений нулями, якщо диференціальний режим не використовується.

14. Ідентифікатор станції диференціальної корекції, ID, 0000-1023.

15. Рядкова контрольна сума.

Приклад повідомлення:

$GPGGA,004241.47,5532.8492,N,03729.0987,E,1,04,2.0,-0015,M,,,,*31

GLL – Географічне розташування – Широта/Довгота 1 2 3 4 5 6 7

$GPGLL, 1111.11, a, yyyyy.yy, a, hhmmss.ss, A*hh < CR><LF>

1. Географічна широта розташування.

2. Північ/Південь (N/S).

3. Географічна довгота місцезнаходження.

4. Схід/Захід (E/W).

5. Середній час за Гринвічем, коли було визначено місцезнаходження.

6. Статус A = Дані дійсні

V = Дані недійсні

7. Контрольна сума рядка.

Приклад повідомлення:

$GPGLL,5532.8492,N,03729.0987,E,004241.469,A*33

GSA - GPS DOP та активні супутники

Це повідомлення показує режим роботи GPS-приймача, параметри супутника, що використовуються для вирішення навігаційної задачі. Результати відображаються в повідомленні $GPGGA та значенні DOP, що використовується для визначення координат 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

$GPGSA, a, x, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, x.x, x.x, x.x*hh <CR><LF>

1. Режим: M = Ручний, примусово 2D або 3D;

A = Автоматичний, дозволити автоматичний вибір 2D/3D.

2. Режим: 1 = Без позиції, 2 = 2D, 3 = 3D

3-14. Номер супутника PRN, що використовується для визначення позиції (нуль означає невикористаний).

15. Коефіцієнт PDOP.

16. Коефіцієнт HDOP.

17. Коефіцієнт VDOP.

18. Контрольна сума рядка.

Приклад повідомлення:

$GPGSA,A,3,01,02,03,04,,,,,,,,,,,2.0,2.0,2.0*34

GSV - Видимі супутники GPS. Це повідомлення показує кількість видимих супутників (SV), їхні номери PRN, їхню висоту над локальним горизонтом, їхній азимут та співвідношення сигнал/шум. Кожне повідомлення може містити повідомлення щонайбільше для чотирьох супутників, а решта надсилається в

наступних повідомленнях $GPGSV. Перші два поля кожного повідомлення вказують загальну кількість надісланих повідомлень та номер поточного повідомлення.

1 2 3 4 5 6 7 8 15 16 17 18 19 20 $GPGSV, x, x, xx, xx, xx, xxx, xx..., xx, xx, xxx, xx*hh <CR><LF> 1. Загальна кількість повідомлень, від 1 до xx 999.

2. Номер повідомлення, від 1 до 9.

3. Загальна кількість видимих супутників.

4. PRN номер супутника.

5. Кут висоти, градуси (максимум 90°).

6. Істинний азимут, градуси, від 000° до 359°.

7. Діапазон співвідношення сигнал/шум: від 00 до 99 дБ, нуль за відсутності сигналу.

8-11. Співвідношення сигнал/шум другого супутника таке ж, як 4-7.

12-15. Співвідношення сигнал/шум третього супутника таке ж, як 4-7.

16-19. Співвідношення сигнал/шум четвертого супутника таке ж, як 4-7.

20. Контрольна сума.

Приклад повідомлення: $GPGSV,3,1,12,02,86,172,,09,62,237,,22,39,109,,27,37,301,*7A $GPGSV,3,2,12,17,28,050,29,21,31,2,12,17,28,050,29,21,14,146,1246,1246,1246,1246,1246,1246 $GPGSV,3,3,12,07,08,231,,10,08,043,,04,06,170,,30, 00,281,*77 RMC — Рекомендований мінімальний обсяг даних GPS/навігації 1 2 3 4 5 6 7 819 1000 метрів. cc, A, 1111.11, A, yyyy. yy, a, x. x, x. x, дд.мм.рр, x. x, A *yy <CR><LF> 1. Точний час UTC 2. Статус: A = дійсний, V = попередження навігаційного приймача 3,4.

7. Швидкість відносно землі (SOG) у вузлах

8. Істинний курс у градусах

9. Дата: дд/мм/рр

10. Магнітне схилення в градусах

11. Схід/Захід (E/W)

12. Рядок контрольної суми (обов'язково) Приклад повідомлення:

$GPRMC,113650.0,A,5548.607,N,03739.387,E,000.01,255.6,210403,08.7,E* 69 VTG - $TG, x 4. Рядок контрольної суми Приклад повідомлення: $GPZDA. Для обробки пакетів даних, отриманих модулем, ми використовуємо фільтр Калмана, реалізований на C#. Отримані дані слід конвертувати з формату DDD MM у формат MMM (D означає градуси, а M – хвилини). DDDDD, наприклад: 087° західної довготи 38.185 38.185 60 = 0.636417, кут: 87.636417°. Північ (N) та Схід (E) – додатні числа, південь (S) та захід (W) – від’ємні числа. Західна довгота t" від'ємна: -87.636417°

Щоб відобразити координати на площині, їх потрібно перетворити зі стереографічної проекції на прямокутні координати:

a=6378137 - екваторіальний радіус

e = 8.181919084E-2 - ексцентриситет

lat = геодезична широта (радіани)

lon = довгота (радіани)

alt = висота відносно еліпсоїда WGS84 (метри)

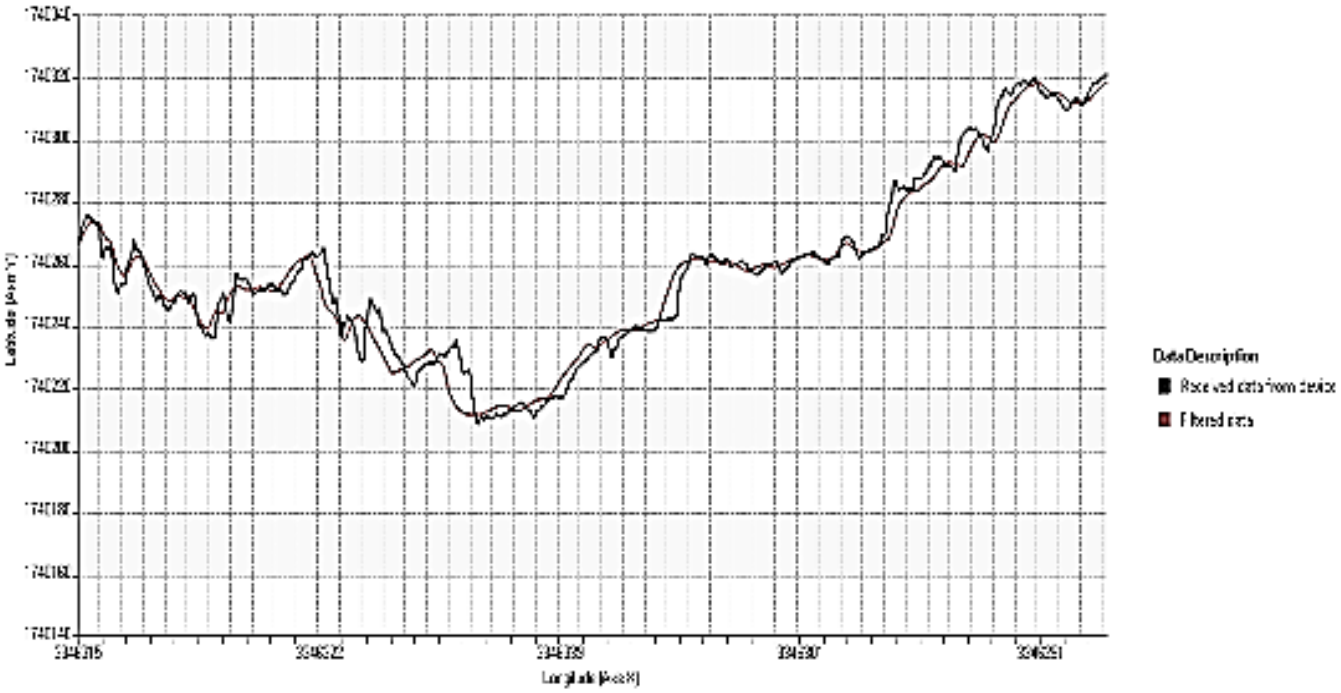


Рис. 3.3. Дані, отримані та відфільтровані модулем L76

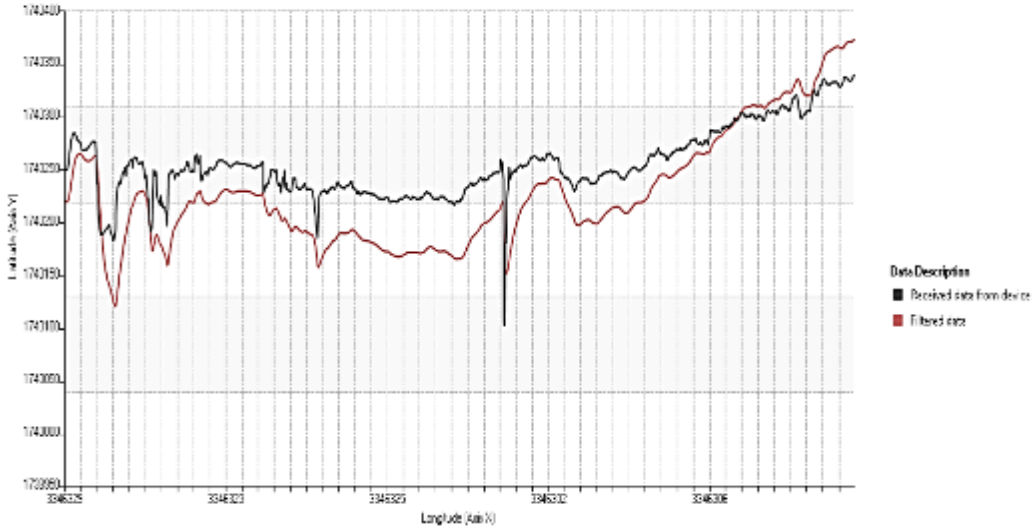


Рис. 3.4. Дані, отримані та відфільтровані модулем LEA-5

На рис. 3.3 та 3.4 чітко видно, що крива відфільтрованих даних є більш плавною та має менше піків. Це означає, що співвідношення сигнал/шум зменшилося, і ми отримали точніші значення.

Для оцінки даних нам потрібно обчислити стандартне відхилення:

$$\sigma = \sqrt{\frac{(a_1-a)^2+(a_2-a)^2+\cdots+(a_n-a)^2}{n}} \qquad (3.1)$$

де:

$$a = \frac{a_1+a_2+\cdots+a_n}{n} \qquad (3.2)$$

Порівнюючи два рисунки, ми виявляємо, що дані, отримані модулем L70, є точнішими, ніж дані, отримані модулем LEA-5. Це пояснюється тим, що модуль L76 використовує комбіновану систему позиціонування GPS/GLONASS, тоді як система GPS, яка використовується модулем LEA-5 для отримання даних, має деякі недоліки.

Загальним недоліком використання будь-якої радіонавігаційної системи є те, що в деяких випадках сигнал може не досягти приймача або може надходити з серйозними спотвореннями чи затримками. Зокрема, глибоко в квартирі, підвалі або тунелі всередині залізобетонної будівлі практично неможливо визначити своє точне місцезнаходження. Оскільки робочі частоти GPS знаходяться в дециметровому діапазоні радіохвиль, прийом супутникового сигналу значно погіршується в умовах густої рослинності або щільної хмарності.

Перешкоди від багатьох наземних джерел радіосигналу, а також магнітні бурі можуть впливати на нормальний прийом GPS-сигналу. Низький нахил орбіти GPS становить 55°, що суттєво знижує точність позиціонування в полярних регіонах Землі, оскільки зона покриття супутників GPS не виходить за горизонт. Фундаментальною особливістю GPS є те, що умови прийому сигналу повністю залежать від Міністерства оборони США. Комбінований приймач супутникової радіонавігаційної системи GPS відповідає всім ринковим вимогам до GPS-приймачів і додає нові якості - надійність і точність позиціонування в несприятливих умовах - за рахунок збільшення кількості прийнятих сигналів. Для невійськових користувачів поточні дані щодо точності GPS показують, що при використанні від 6 до 11 супутників діапазон похибки широти та довготи становить від 2,00 до 8,76 метрів. При використанні від 7 до 8 супутників діапазон похибки широти та довготи GPS становить від 4,46 до 7,38 метрів. При одночасному використанні обох систем похибка може бути зменшена до 1,5-3 метрів. У повсякденному використанні надійний прийом важливіший за точність. Незалежно від того, чи це автомобільна навігаційна система, чи туристична, важливо підтримувати стабільний зв'язок із супутниками та завжди чітко визначати поточне положення. У ситуаціях, коли сигнал нестабільний на вузьких вулицях між висотними будівлями або на дорогах, оточених густими лісами, сигнали двох систем перекриваються. Коли навігаційна система, що використовує лише GPS, втрачає супутники, пристрої, що використовують обидві системи одночасно, показують хороші результати. Попередні результати тестування показують, що в певних сценаріях використання цього приймача використання другого GPS може значно покращити характеристики навігації.

3.2. Формування сигналів глобальної навігаційної системи як основи оптимізаційного алгоритму

Цей сигнал схожий на обгорнутий фізичний носій, що показує річницю зірки-супутника. Такий підхід гарантує, що ваша річниця буде «чітко видно» за тисячі кілометрів, незалежно від темряви, опале листя чи інших погодних умов.

Структура функції S(t) відома та описана в файлах керування системою. На основі математичної моделі сигналу вона використовується для визначення кількості годин у сигналі та вирівнювання його зі старим сигналом.

Навігаційний сигнал приймається за наявності шуму та інших сигналів. Інтенсивність цих ін'єкцій, і це важливо, у сто разів або більше перевищує інтенсивність самого навігаційного сигналу. Тому обробка сигналу не виконується, доки функція не прийме від'ємне значення. Такий підхід будує локальну копію сигналу на основі математичної моделі сигналу, а потім налаштовує параметри цієї копії на максимальну швидкість.

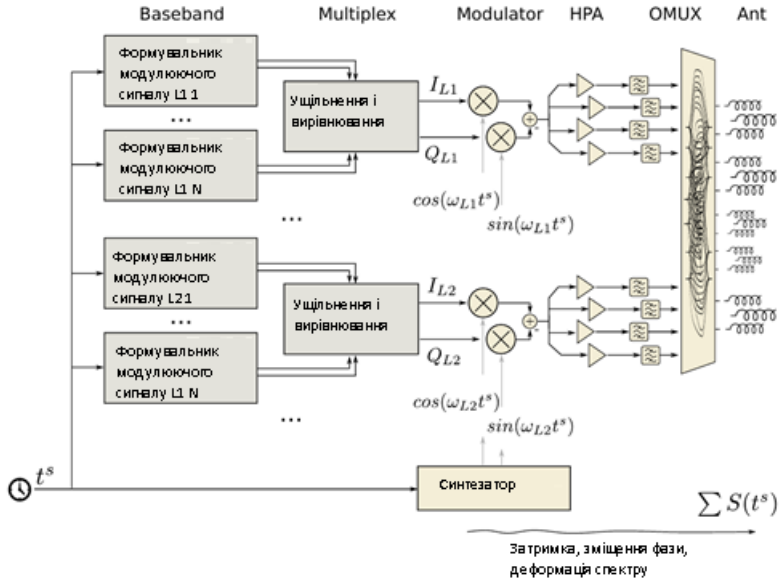


Рис. 3.5. Формування навігаційних сигналів із сукупності супутників

Алгоритм обробки навігаційного сигналу визначається математичною моделлю навігаційного сигналу. Тепер нас дивує дорожня поліція. Різні системи, здається, генерують різні сигнали. Однак, кожен супутник не обмежується одним сигналом, а являє собою комбінацію всіх типів сигналів. На нових супутниках можна побачити до 14 різних навігаційних сигналів/ Це означає понад 50 типів сигналів для всіх систем.

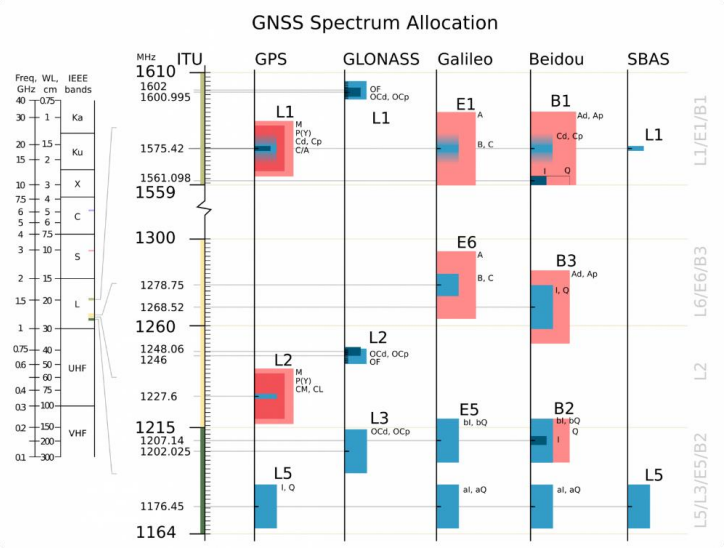


Рис. 3.6. Розподіл різних типів сигналів у низькому діапазоні L

Якщо спробувати побудувати модель OSI до радіонавігації, різні системи та сигнали визначають різні протоколи фізичної площини.

Але не поспішайте відкривати своє IDE.

1. Зі збільшенням розміру прошивки дуже важливо підтримувати та тестувати нейтральні реалізації таких функцій, але це дорого.

2. В обробці сигналів деякі функції реалізуються апаратно в модулях FPGA або HVIC, що означає, що нам не потрібно використовувати спеціальні канали та виконувати багато налаштувань.

Перетворити різні сигнали в математичну модель та використовувати параметри цієї моделі для опису нюансів різних сигналів. Нами знайдено цей підхід після розробки кількох навігаційних інструментів. Використовуйте загальні канали для зараження та ініціалізуйте ці канали необхідними параметрами перед початком обробки та використання певних сигналів.

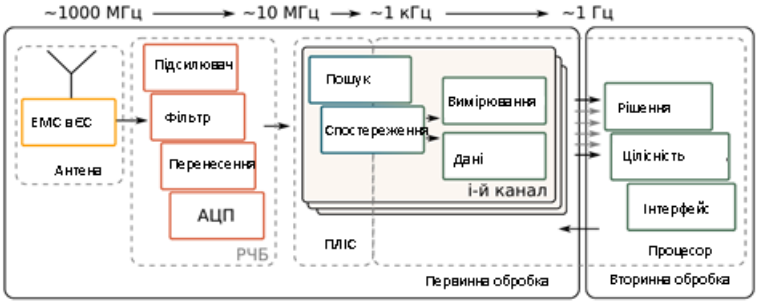


Рис. 3.7. Функціональна схема навігаційного приймача

Положення водія оцінюється на основі рівня шуму сигналу, тому навігаційний сигнал спеціально розроблений для легкої оцінки на основі шуму та важливих факторів.

Згідно з формулою Вудворда, чим більша площа, яку займає сигнал, і чим концентрованіший спектр на межі видимого діапазону, тим менше шум впливає на оцінку його вихідної затримки. Супутники передають навігаційні дані про своє положення з низькою швидкістю, тому спектр сигналу не може бути повністю розширений. Тому розширені послідовності можуть ще більше покращити точність оцінки синхронізації сигналу.

Найпоширенішими сигналами, які можуть виявляти смартфони, ймовірно, є сигнали GPS L1C/A. Ці сигнали скануються за допомогою послідовностей спектру коду дальньої дії з періодом 1 мілісекунда, скануючи до 1023 та 511 символів одночасно. Ці послідовності доповнюються фіктивними елементами та генеруються за допомогою додаткових лінійних генераторів на окремих буферах.

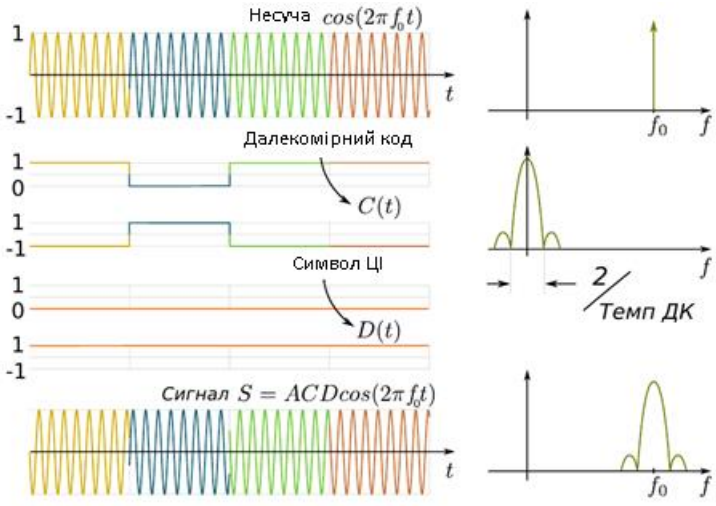


Рис. 3.8. Сигнал GPS з кодом модуляції відстані та навігаційним оголошенням

Після десятиліть роботи зі застарілими сигналами, система GPS почала модернізуватися, включаючи додавання нових сигналів. Система обробки сигналів Galileo була адаптована з системи BeiDou, яка була побудована з нуля. Детективи сигналів почали намагатися з'єднати всі сигнали одночасно та героїчно подолали численні виклики, з якими вони зіткнулися:

• висока ефективність, висока точність та низьке енергоспоживання;

• проста інфраструктура та низьке енергоспоживання;

• мінімізація витоку сигналу до інших радіосистем;

• стабільність сигналу (висока CCD супутникових підсилювачів) та посилення старих сигналів;

• вибір частот, що передують розподілу частот супутникової навігації;

• підтримка чутливості та допуску до перевантаження за вищих швидкостей передачі даних.

Схеми модуляції стали складнішими. Наприклад, GPS L2C розроблений для посилення двох компонентів сигналу під час передачі: один символ для одного компонента сигналу та інший символ для іншого. Ця схема також використовується для базових сигналів коду GPS. У деяких сигналах допоміжний код зупиняється. Це допомагає зменшити відносну силу сигналів, оскільки прийом одного сигналу менш важливий, ніж прийом іншого.

Коли спектр сигналу розділяється, генеруються два сигнали. Найкраще помістити новий сигнал на той самий сигнал, що й старий, щоб вони були спектрально розділені один від одного, зменшуючи можливість однозначного злиття. Варто зазначити, що ця техніка називається BOC (Binary Offset Carrier), AltBOC, TMBOC, CBOC, QMBOC тощо.

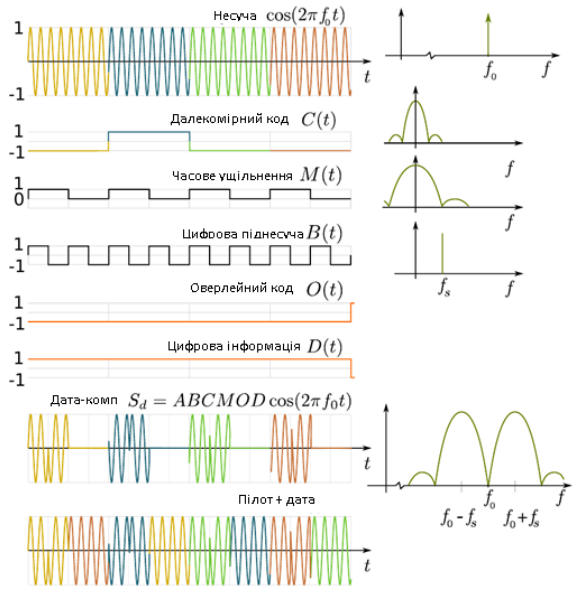


Рис. 3.9. Сигнал GPS з видаленням коду, цифровий супутник, накладений код та навігаційне повідомлення

Просту структуру сигналу можна отримати з вихідної моделі, відключивши відповідні множники або припустивши, що їхні множники рівні.

Система GPS пройшла через кілька поколінь супутників, і обладнання від блоку IIA до блоку III продовжуватиме працювати.

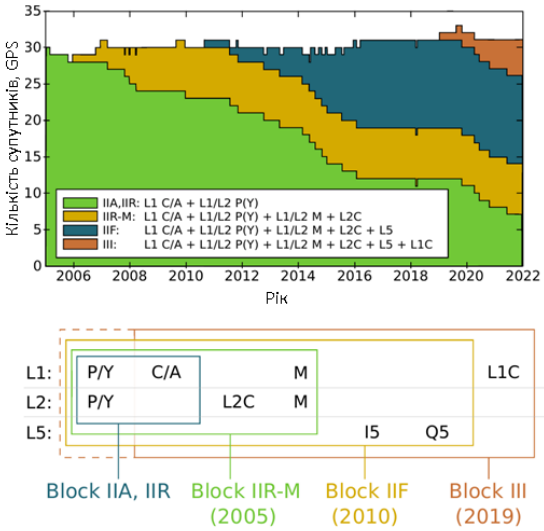


Рис. 3.10. Зв'язок між переданим сигналом та типом GPS-навігаційного супутника

Під час процесу оновлення цивільний сигнал L2C був доданий до іншої неносіючої частоти – військового зашифрованого сигналу безпеки M. Блок IIA почав впроваджувати сигнали L5 на третій частоті, що переносить лише 20 МГц, тоді як Блок III забезпечував сигнали L1C. Решта сигналів незабаром зможуть замінити сигнал C/A в наших смартфонах і забезпечити цивільних користувачів функцією цифрового підпису для запобігання спуфінгу.

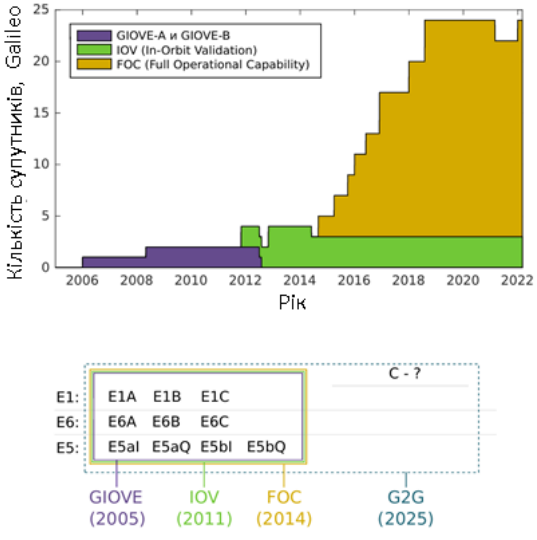


Рис. 3.11. Сигнали Galileo

Система Galileo ідеально підходить для частотного планування навігаційних систем. За параметрами цей сигнал E1 близький до сигналу GPS L1C та сигналу BeiDou B1C. У тестовому режимі цифровий підпис можна негайно замінити, щоб запобігти спуфінгу. Сигнал E5 використовує модуляцію Vicor AltBOC для створення широкосмугового агрегованого сигналу до 50 МГц з усіх джерел сигналу. Під час розробки системи сигнал E6 був перетворений на комерційний сигнал за додаткову плату. Через кілька днів було опубліковано специфікацію C/NAV.

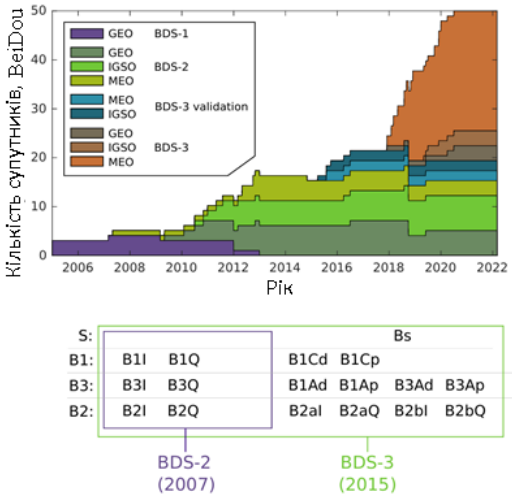


Рис. 3.12. Система BeiDou

Система BeiDou оновлюється з вражаючою швидкістю. Наразі система перебуває на третьому етапі, і очікується, що супутникова мережа буде повністю оновлена через 2-3 роки. Сигнал третього етапу випереджатиме сигнал Galileo та краще доповнюватиме системи США та Європи.

3.3. Реалізація оптимізаційного алгоритму фільтрації Калмана за результатами вимірювання координат

Як і багато оптимізаційних алгоритмів, нелінійні ФК повинні розуміти в який бік їм потрібно змінювати параметри, щоб максимально наблизитися до мінімуму функції з мінімальною розбіжністю між попереднім станом системи і

мінімальна розбіжність між тим, що ми очікуємо побачити від моделюваного датчика і самими даними від реального датчика. Пошук цього градієнта залежить від виду лінеаризації.

Усі нелінійні види фільтрації Калмана (ФК) так чи інакше використовують лінеаризацію. Поширені 2 види лінеаризації ФК:

1. Розраховані людиною матриці Якобі, записані в аналітичному вигляді (Extented Kalman Filter EKF). Іншими словами – це лінеаризація щодо дотичної.
2. Лінеаризація методом пробних точок (Sigma Point Kalman Filter, Unscented Kalman Filter UKF). Іншими словами - це лінеаризація по січеній.

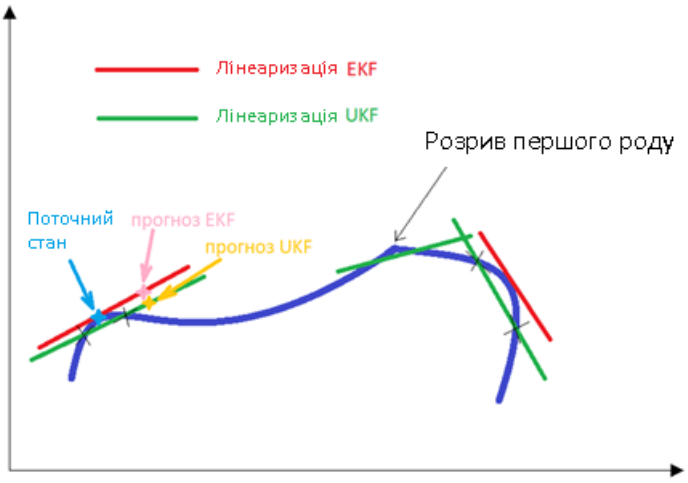


Рис. 3.13. EKF та UKF лінеаризації

В інженерному контексті для нас можна виділити такі особливості: У першому випадку із переваг – швидка швидкість роботи. З недоліків – ручний запис в аналітичному вигляді матриці Якобі, менша стабільність та точність на сильних нелінійних взаємозв'язках, функції еволюції та моделі датчика мають бути диференційовані. У другому випадку з переваг - найкраща стабільність і точність на сильних нелінійних взаємозв'язках, відсутність необхідності запису матриць Якобі, потрібна лише запис у прямому вигляді еволюції та моделі датчика.

Згідно з цілями, знизити поріг входження в тему, дати універсальний алгоритм для швидкого прототипування і перевірки гіпотез в умовах слабкої

обчислювальної бази і малої кількості пам'яті кращим вибором буде другий вид ФК - UKF.

Крім UKF, існують наступні варіації сигма-точкового фільтра Калмана: CDKF (central difference Kalman filter – фільтр Калмана центральної різниці), SRUKF (square-root unscented Kalman filter – ансцентний фільтр Калмана квадратного кореня), SRCDKF (square-root central difference їх ітеративні модифікації IUKF, ICDKF, ISRUKF, ISRCDKF.

В інженерному контексті завдання оптимальним вибором вважаю т.зв. Square Root Central Difference Kalman Filter (SRCDKF). Вибір сигма точок здійснюється згідно з центральною різницею (в негативний напрямок від лінеаризованої точки і в позитивний напрямок) пропорційно матриці квадратного кореня коваріації. При цьому не використовується розкладання Холецького для матриці коваріації, яке може бути не стабільним в умовах із присутністю одночасно малих та великих значень у матриці.

Це універсальне ядро SRCDKF, яке можна використовувати у різних додатках, реалізуючи різні математичні моделі. Почнемо відразу з реальної інженерної задачі, без прикладів на одновимірних випадках, тому що в них не виявиться вся міць таких алгоритмів. Паралельно розберемо вихідний код.

Припустимо, є квадрокоптер, який може летіти в будь-який бік, будь-якою стороною борту, тобто в нього немає переда чи зад. На квадрокоптері встановлений акселерометр і датчик кутової швидкості (IMU), датчик глобальної навігаційної системи (GNSS), в нашому прикладі немає магнітометра та барометра, хоча це не важливо.

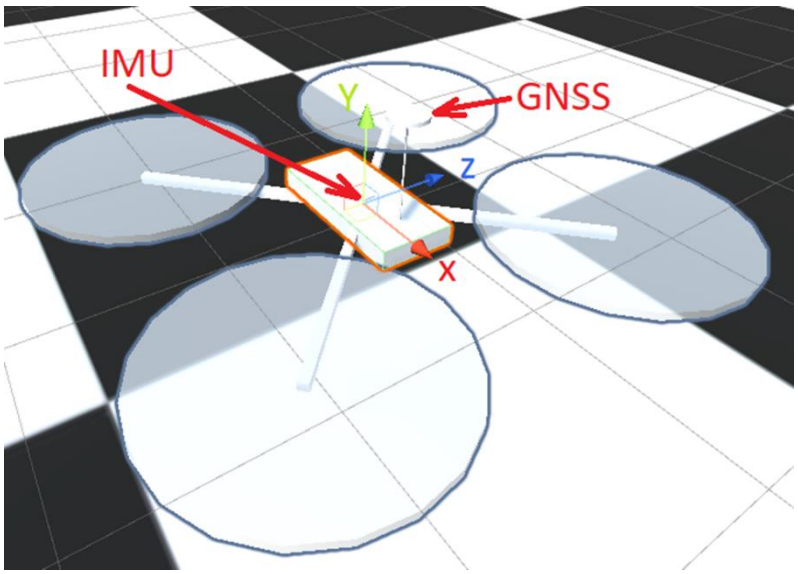


Рис. 3..14. Умовний квадрокоптер, який може летіти в будь-який бік

Нам для управління квадрокоптером потрібно знати становище та орієнтацію у просторі в кожний момент часу.

Якщо я сказав, що завдання з реального світу, а реальний світ тривимірний, то ми будемо розглядати математичну модель опису обертання та положення у тривимірному просторі. Для опису обертання тривимірному просторі скористаємося кватерніоном, можна сказати стандартом таких випадків. А для опису положення – звичайним 3-мірним вектором.

Хто працював із дешевими MEMS IMU знає про необхідність калібрування для нівелювання масштабних спотворень, міжосьових взаємозв'язків, а також зміщень (biases).

Так виглядають дані з акселерометра при обертанні по всіх осях. Центр еліпсоїда зміщений деякий вектор (bias). Сам еліпсоїд масштабно спотворений.

Потрібно придумати математичну модель еволюції системи, тобто як вона змінюватиметься згодом, які дані від сенсорів будуть моделюватися, які дані будуть апріорними, в якій системі координат виражатиметься вектор стану глобальної, прив'язаної до географічних осей північний схід або локальної, в системі координат квадрокоптера, яка модель датчика GNSS. Щоб не ускладнювати дослідження, я пропоную скористатися не надто складною і водночас робочою математичною моделлю.

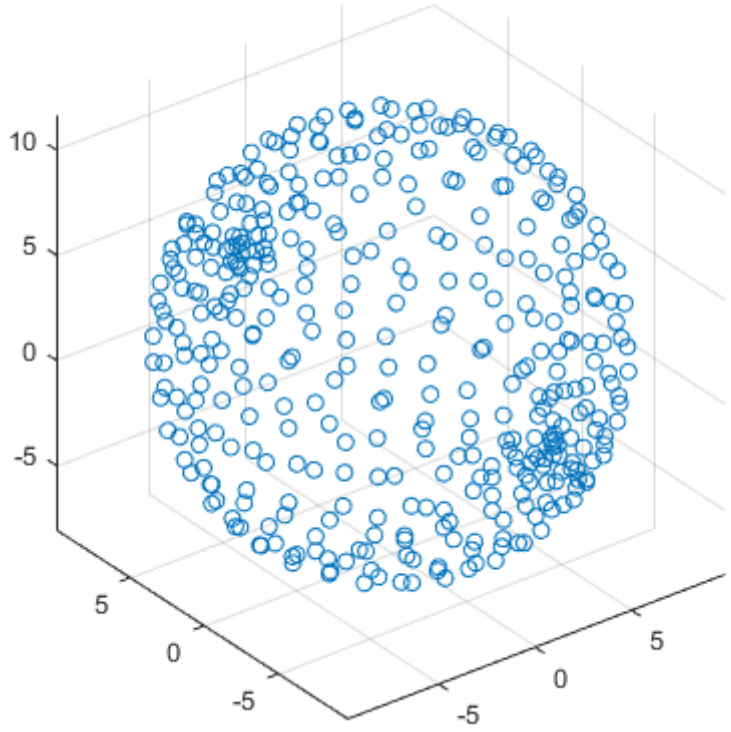


Рис. 3.15. Калібрування для нівелювання масштабних спотворень сигналів

Для більш-менш достатньої для нашої точності математичної моделі потрібно 3 параметри для опису зміщення (bias) акселерометра (в СК IMU), 3 параметра для опису зміщення (bias) датчика кутової швидкості (в СК IMU), 3 параметра опису положення (3-х мірний вектор в глобальній СК), 4 параметр виражає вектор з СК коптера в глобальну СК), додатково включимо 3-мірний вектор швидкості в глобальній СК. Разом 16 параметрів. Це називається вектор стану системи, тому що він описує повний стан у контексті нашого завдання нашого квадрокоптера в конкретний момент часу, тобто яке має положення, орієнтацію, швидкість і т.д.

Еволюція системи описується тим, як змінюватиметься вектор стану у часі, тобто залежність нинішнього моменту часу від минулого, ще цей етап називається прогнозом.

Зміщення IMU не залежать від інших змінних у векторі стану, тому в даний час вони майже рівні минулим значенням. Майже тому що все-таки вони трохи змінюються з часом температура може змінюватися, потрібно щоб ФК розумів, що це не константи і мав деяку невизначеність у значенні, для цього є два

параметри, які є векторами стандартних відхилень шумів прогнозу для зсувів IMU.

Кілька слів про вектори стандартних відхилень шумів прогнозу. Чим більшими будуть ці значення шумів, тим менша довіра прогнозу, невизначеність відповідної змінної вектора стану зростатиме з кожною ітерацією прогнозу і етап корекції вноситиме суттєве коригування в них. Тобто ця змінна буде схильна до великої волатильності. ФК користуватиметься цією "крутилкою" більше, ніж тими, у яких мале значення стандартного відхилення шуму.

Також варто відзначити, що, на відміну від інших видів ФК, модель шуму як процесу, так і датчика може бути не адитивною, і даний вид ФК зможе врахувати і нелінійну природу шуму. Ця перевага забезпечена способом лінеаризації за допомогою сигма-крапок, які пропускаються в тому числі і через модель шуму.

Кватерніон - це гіперпараметризована модель обертання, оскільки має 4 параметри у просторі обертання з 3 ступенями свободи. Так як повноцінно описати обертання 3 параметрами не виходить (Gimbal lock), доводиться вдаватися до таких математичних конструкцій як кватерніони. Для переходу з 4-мірного простору в 3-мірний простір і назад є спеціальні операції. Це можна уявити як сферу (кватерніон) і дотичну площину (3-мірний вектор обертання).

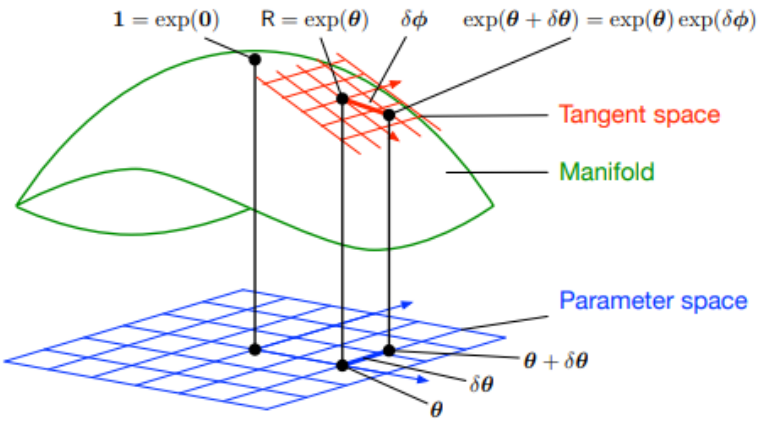


Рис. 3.16. Гіперпараметризована модель обертання алгоритму фільтра Калмана для безпілотних об'єктів

ФК не вміє працювати з параметрами, які мають обмеження, а у кватерніону має бути одинична довжина. У такому разі можна піти на невелику хитрість і після кожної ітерації прогнозу нормалізувати кватерніон. Для нашої мат. моделі такий спосіб пригодиться, однак для більш нелінійних систем доведеться використовувати спеціальну арифметику та операції з кватерніонами всередині ФК.

У нашому випадку є лише 2 датчики IMU та GNSS. IMU - використовується безпосередньо в моделі прогнозу як апріорні значення та не моделюються. Чим вище, тим менше довіра GNSS, дану величину можна варіювати в залежності від якості даних з датчика.

Вектори стандартних відхилень шумів прогнозу і моделі датчиків можна взяти з датішитів, або провести експеримент із реальними датчиками, розрахувати самому. У нашому випадку вони вже розраховані, якщо захочете використати іншу матюку. модель - від чого відштовхуватиметься.

Крім реальних реальних датчиків, можна моделювати псевдодатчики. Наприклад, при старті, коли коптер ще не злетів, і ми точно впевнені, що він не рухається, можна використовувати псевдовимірювання кутової та лінійної швидкостей, які дорівнюють нулю. Тим самим повідомляючи ФК, що наразі руху немає, і ця інформація, у свою чергу, допоможе в уточненні вектора стану.

Коротко розповім подробиці симулятора: Сам ФК у Unity використовується як динамічна бібліотека, для WebGL – сирі вихідники. Коптер не управляється за фізичними законами, використовуються просто сили та моменти для імітації польоту коптера у скрипті Control.cs. Для імітації IMU та GNSS створено 2 скрипти IMUModel.cs та GNSSModel.cs. Обгортка та взаємодія з Unity ФК реалізована в KalmanFilter.cs. Для графіків використовувався відкритий Asset .

Для фізики використовується внутрішній двигун PhysX. Для демонстрації 60 Гц – достатньо.

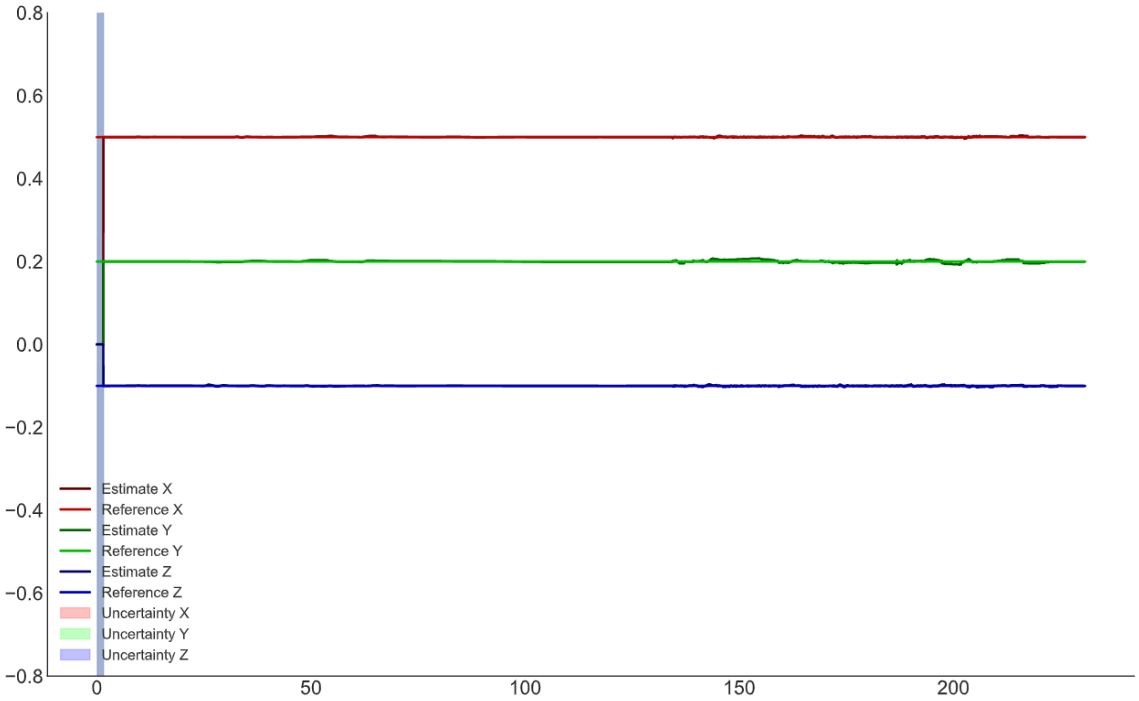


Рис. 3.17. Вектор зсувів супутникової навігаційної системи

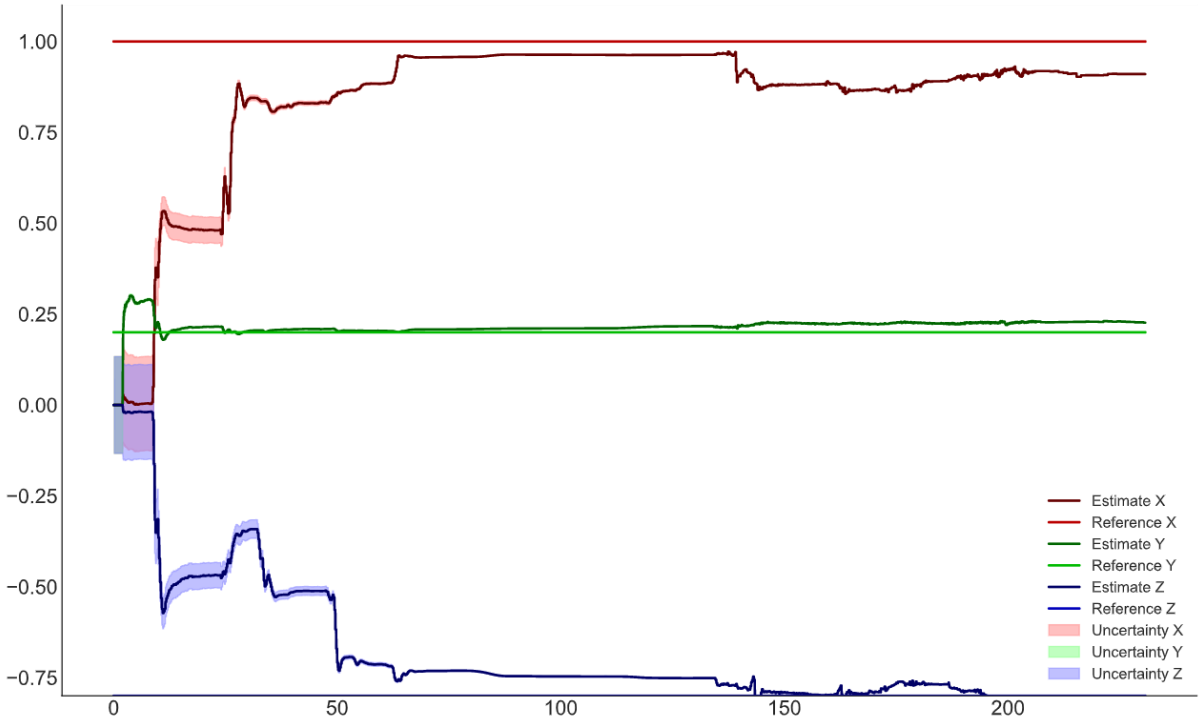


Рис. 3.18. Вектор акселерометра зсувів

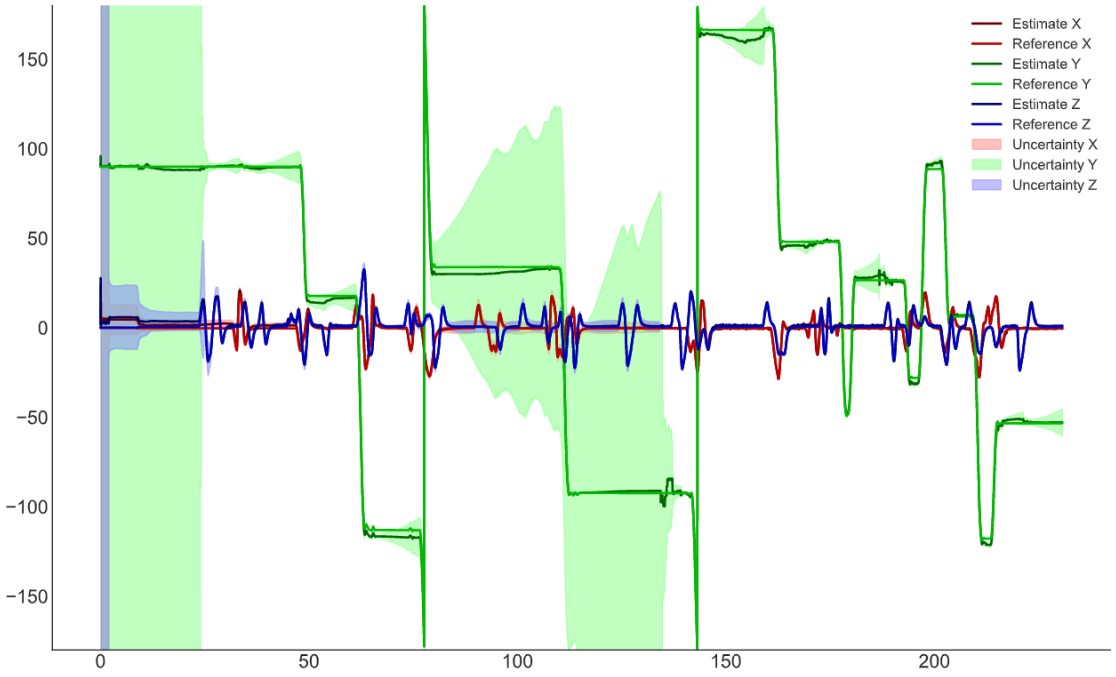


Рис. 3.19. Графік кутів Ейлера

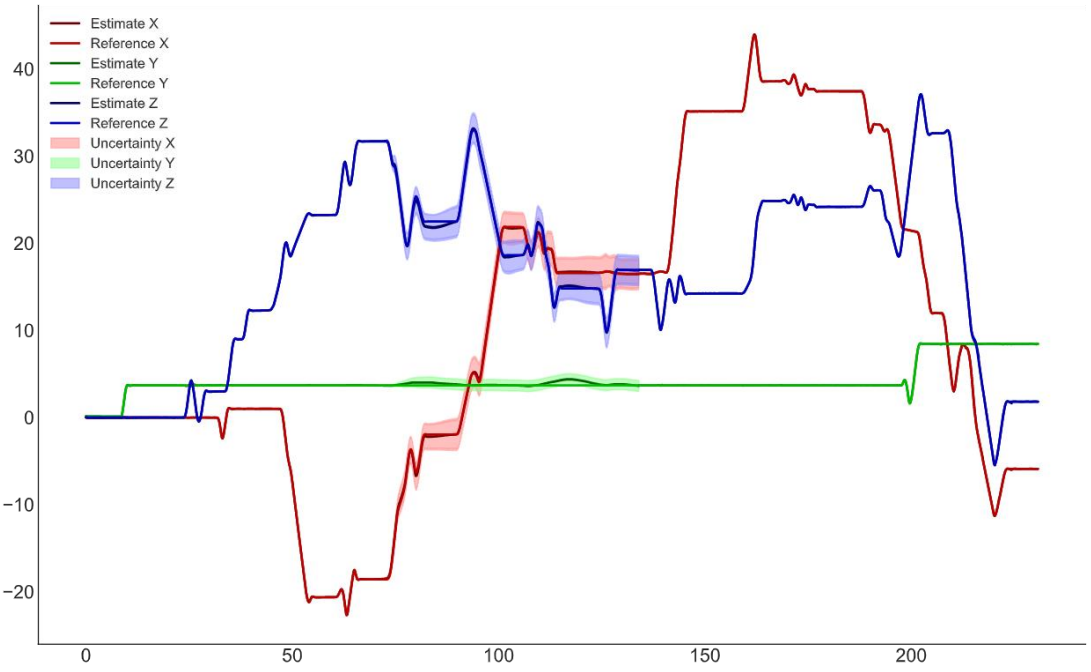


Рис. 3.20. Вектор положення

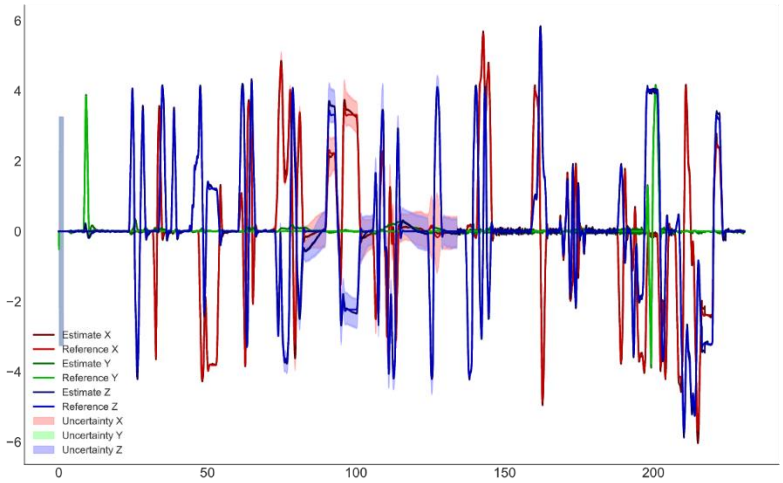


Рис. 3.21. Вектор швидкостей

Пояснення за графіками: Estimate – це оцінка ФК, Reference – це фактична величина, Uncertainty – дисперсія з матриці коваріації *100.

У центрі графіків видно, що помилка зросла, це сталося через зменшення впливу даних GNSS.

Можна помітити, що магнітометр не використовувався для корекції курсу. Якщо застосовувати звичайний фільтр Magwick або Mahony тангаж і крен більш-менш спостерігаються, коли немає лінійних прискорень, то кут курсу буде пливти і взяти його звідки. А якщо постійно присутні лінійні, доцентрові прискорення, то всі кути спливуть при корекції акселерометром, це актуально для літаків, наприклад.

У разі застосування ФК та даної мат. моделі такого не буде: за наявності руху за допомогою датчика GNSS відбуватиметься поділ сирого вектора прискорення з акселерометра на лінійну частину і на вектор прискорення вільного падіння, через що кути тангажу та крену будуть максимально точними, а кут курсу не буде спливати. До того ж ми отримуємо ще й точніші вектори положення та швидкості. Цю особливість помітив не тільки я:

У цьому полягає перевага ФК - знаходженні неявних корисних зворотних залежностей між вектором стану і датчиками.

Далі представляємо графік, що ілюструють корекцію курсового кута, навіть при початковому неправильному значенні в 45 градусів.

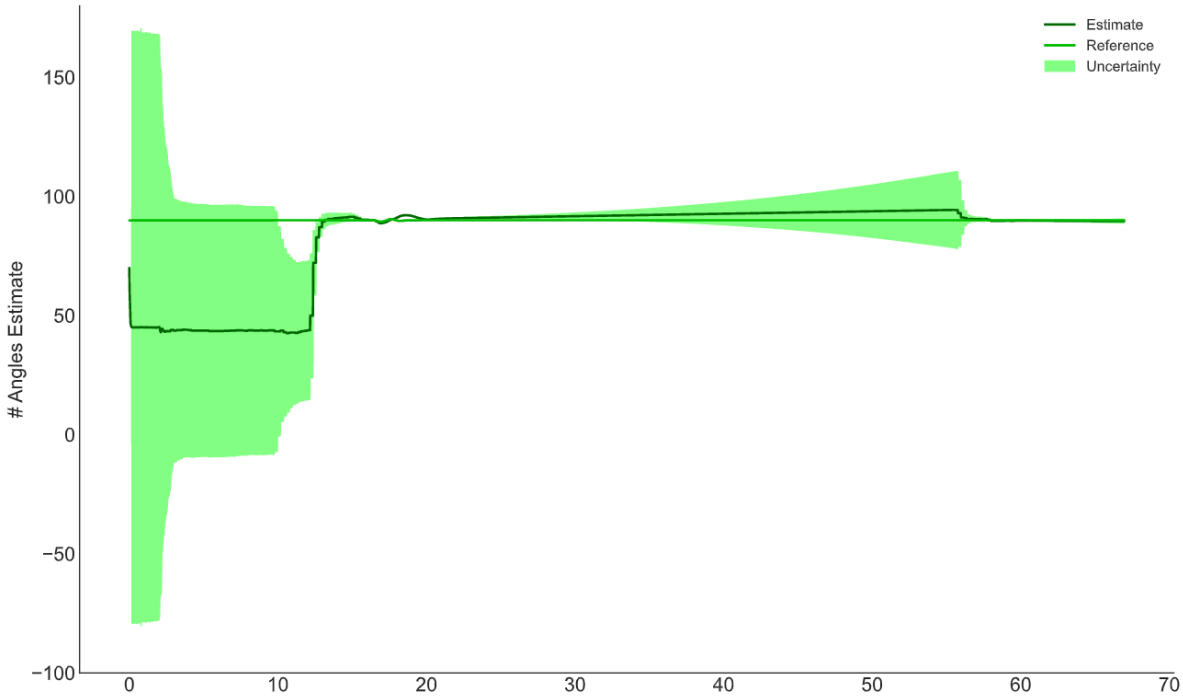


Рис. 3.22. Кут курсу з початковою помилкою в 45 градусів

В інтерактивній демці багато чого немає, з чим можна пограти безпосередньо в Unity Edior, а саме: різні значення bias IMU, їх зміни в часі, різні величини стандартних відхилень шумів прогнозу та датчика GNSS, затримки між датчиками, неправильні початкові умови тощо. Так що раджу завантажити вихідний вихід і безпосередньо в Unity Editor пограти з різними умовами.

Найкорисніше, це те, що той самий C++ код, який працює в демке (kalman lib), можна цілком без переробок запустити на мікроконтролері, на реальному залізі в реальних умовах.

3.4. Висновок до розділу 3

Враховуючи різноманітність доступних наразі навігаційних сигналів GNSS, ми можемо спростити їх до вичерпного опису, таким чином уніфікувавши програмні та апаратні модулі, що складають ці сигнали. Це об'єднання спростить розробку та тестування пристроїв та визначить, як проектувати програмні архітектури та інтерфейси. Пропускаючи кроки аналізу, виконуючи їх у скоригованій моделі, а потім зводячи їх до необхідних сигналів.

Ця інформація описується як високошвидкісна та не заважає реалізації модулів, які обробляють такі сигнали. Крім того, зберігається канальний рівень із синхронізацією потужності, декодуванням та зберіганням. Це також пов'язано з тим, що він неофіційний, але про це ми поговоримо пізніше.

ВИСНОВКИ

Найважливішими сферами застосування супутникової навігації є:

• геодезія, кадастрові роботи: за допомогою навігаційних систем визначаються точні координати меж земельних ділянок;

• картографія: використання супутникових навігаційних систем для складання карт;

• навігація: використання навігаційних систем для морської, дорожньої, авіаційної та інших видів навігації;

• супутниковий моніторинг дорожнього руху: за допомогою навігаційних систем можна відстежувати положення та швидкість транспортних засобів, а також контролювати їх рух;

• стільниковий зв'язок: якщо мобільний телефон оснащений супутниковим навігаційним обладнанням, можна визначити його положення;

• геофізичні дослідження: за допомогою навігаційних систем можна спостерігати рух та вібрацію літосферних плит.

Кожен регіон, де використовуються супутникові навігаційні системи, має свої вимоги до точності позиціонування. Наприклад, кадастрові роботи вимагають похибки не більше 10 см. Дорожня навігація зазвичай вимагає точності 5 метрів. Для задоволення цих вимог використовуються різні датчики та різні методи.

Існує багато способів вирішення проблеми супутникової навігації. У минулому диференціальні методи були найпопулярнішими. Ці методи вимагають, щоб принаймні одна наземна станція була достатньо близько до місця вимірювання. Ці методи можуть зменшити більшість помилок алгебраїчних операцій, особливо іоносферні та тропосферні затримки, тим самим зменшуючи помилки годинників супутника та приймача. Однак точність позиціонування зменшується, чим далі обчислення виконується від наземної станції. Це головний недолік диференціальних методів, але, незважаючи на це, їм приділялася велика увага в минулому, тоді як абсолютні методи рідко розглядалися. Але це почало змінюватися з появою наземних служб, які

обчислюють різні додаткові дані, такі як точні орбіти супутників, помилки годинника супутника, атмосферні карти тощо. Сьогодні найбільшою такою службою є Міжнародна служба глобальної навігаційної супутникової системи. В останні роки високоточні методи також стають все більш популярними. Такі методи вимагають врахування найбільшої кількості даних для корекції різних помилок та затримок.

Було проведено аналіз продуктивності фільтра Калмана. Для отримання даних позиціонування обрано навігаційні модулі. Проаналізовано відфільтровані дані з різних модулів.

Суть ФК можна розширити на оптимізацію як поточного вектора стану в конкретний час, а й оптимізацію, наприклад, цілого набору станів, включених у різні моменти часу Sliding-Window Optimization, bundle adjustment/graph optimization. Такий підхід вимагає набагато більшої обчислювальної потужності та реалізується іншими методами, проте набагато точніший і стабільніший.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Belej O., Kolesnyk K., Polotai O. (2021) Application of Neural Networks in Intrusion Monitoring Systems for Wireless Sensor Networks. In: Shakhovska N., Medykovskyy M.O. (eds) Advances in Intelligent Systems and Computing V. CSIT 2020. Advances in Intelligent Systems and Computing, vol 1293. Springer, Cham. - https://link.springer.com/chapter/10.1007%2F978-3-030-63270-0_74 - DOI: 10.1007/978-3-030-63270-0_74

2. Belej O., Lohutova T., Halkiv L. (2021) Development of Evaluation Templates for the Protection System of Wireless Sensor Network. In: Ageyev D., Radivilova T., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 69. Springer, Cham. https://doi.org/10.1007/978-3-030-71892-3_10

3. Belej, O., Bokla, N., & Halkiv, L. (2021). Detection of attacks in wireless networks of IoT. Paper presented at the CEUR Workshop Proceedings, , 2923 153-163. - https://www.scopus.com/inward/record.uri?eid=2-s2.0-85112365162&partnerID=40&md5=b42660c3ea5a30236c29b7782f55a737 - Scopus

4. Belej, O., Fedirko, Y., Markelov, O. Ensuring accuracy of personal data processed in blockchain systems. (2024) CEUR Workshop Proceedings, 3723, pp. 13-41. - CEUR Workshop Proceedings, Volume 3723, Pages 13 – 41, 2024, 6th International Workshop on Modern Data Science Technologies Workshop, MoDaST 2024, Lviv, 31 May 2024 through 1 June 2024, Code 200872 – ISSN: 16130073 - https://ceur-ws.org/Vol-3723/paper2.pdf

5. Neinavaie M., Khalife J., Kassas Z. M. Acquisition, Doppler Tracking, and Positioning With Starlink LEO Satellites: First Results. IEEE Transactions on Aerospace and Electronic Systems. 2022. Vol. 58, № 3. P. 2606–2610. DOI: 10.1109/TAES.2021.3127488.

6. Albano M., Hadzic S., Rodriguez J. Use of Negative Information in Positioning Algorithms. Mobile Multimedia Communications. Springer, Berlin, Heidelberg. 2012. P. 206–217. DOI: 10.1007/978-3-642-35155-6_17. (date of access: 27.12.2023)

7. .Saroufim J., Hayek S. W., Kassas Z. M. Simultaneous LEO Satellite Tracking and Differential LEO-Aided IMU Navigation. ION Position, Location and Navigation Symposium (PLANS), 24 April 2023, Monterey, USA. IEEE. 2023. P. 179–188. DOI: 10.1109/plans53410.2023.10140087. (date of access: 27.12.2023)

8. Kozhaya S., Kanj H., Kassas Z. M. Multi-Constellation Blind Beacon Estimation, Doppler Tracking, and Opportunistic Positioning with OneWeb, Starlink, Iridium NEXT, and Orbcomm LEO Satellites. ION Position, Location and Navigation Symposium (PLANS), 24 April 2023, Monterey, USA. IEEE. 2023. P.1184–1195. DOI: 10.1109/plans53410.2023.10139969. (date of access: 27.12.2023)

9. Chi Y.-E., Park J., Park S.-O. Hybrid Multibeamforming Receiver With HighPrecision Beam Steering for Low Earth Orbit Satellite Communication. IEEE Transactions on Antennas and Propagation. 2023. Vol. 71, № 7. P. 5695–5707. DOI: 10.1109/TAP.2023.3277195. (date of access: 27.12.2023)

10. Portillo I., Cameron B. G., Crawley E. F. A technical comparison of three low earth orbit satellite constellation systems to provide global broadband. Acta Astronautica. 2019. Vol. 159. P. 123–135. 10.1016/J.ACTAASTRO.2019.03.040. (date of access: 27.12.2023)

11. Wang J., Zhou Z., Jiang W., Cai B., Shangguan W. A Multi-constellation Positioning Method Based on Optimal Stochastic Modelling. China Satellite Navigation Conference (CSNC) 2020 Proceedings: Volume I. Lecture Notes in Electrical Engineering. Vol. 650 / ed. Sun J., Yang C., Xie J. Singapore: Springer, 2020.

12. Rabjerg J., Leyva-Mayorga I., Soret B., Popovski P. Exploiting topology awareness for routing in LEO satellite constellations. IEEE Global Communications Conference (GLOBECOM), 07–11 December 2021, Madrid, Spain. IEEE. 2021. P. 16. DOI: 10.1109/GLOBECOM46510.2021.9685686. (date of access: 27.12.2023)

13. Peng Y., Scales W. Ionospheric Remote Sensing with GNSS. Encyclopedia. 2021. Vol.1. № 4. P.1246–1256. DOI: 10.3390/encyclopedia1040094. (date of access: 27.12.2023)

14. Groves P. Shadow Matching: A New GNSS Positioning Technique for

Urban Canyons. Journal of Navigation. 2011. Vol. 64. P. 417–430. DOI: 10.1017/S0373463311000087. (date of access: 27.12.2023)

15. .Jardak N., Adam R. Practical Use of Starlink Downlink Tones for Positioning. Sensors. 2023. Vol. 23. № 6. P. 3234. DOI: 10.3390/s23063234.

16. Won S. M. et al. Multimodal Sensing with a Three-Dimensional Piezoresistive Structure. ACS Nano. 2019. Vol. 13. № 10. C. 10972–10979. DOI: 10.1021/acsnano.9b02030. (date of access: 27.12.2023)

17. Shaker M., Ghaffari A., Maghsoodpour A., Khodayari A. GPS/INS Integration for Vehicle Navigation based on INS Error Analysis in Kalman Filtering. International journal of automotive engineering. 2017. Vol. 7. P. 2562–2570. DOI: 10.22068/IJAE.7.4.2562. (date of access: 27.12.2023)

18. Nourmohammadi H., Keighobadi J. Decentralized INS/GNSS System With MEMS-Grade Inertial Sensors Using QR-Factorized CKF. IEEE Sensors Journal. 2017. Vol. 17. P. 3278–3287. DOI: 10.1109/JSEN.2017.2693246. (date of access: 27.12.2023)

19. Ertürk S. Real-Time Digital Image Stabilization Using Kalman Filters. Real Time Imaging. 2002. Vol. 8. P. 317–328. DOI: 10.1006/rtim.2001.0278. (date of access: 27.12.2023)

20. Hoang M., Carratu M., Paciello V., Pietrosanto A. Noise Attenuation on IMU Measurement For Drone Balance by Sensor Fusion. International Instrumentation and Measurement Technology Conference (I2MTC), 17 May 2021, Glasgow, United Kingdom. IEEE. 2021. P. 1–6. DOI: 10.1109/I2MTC50364.2021.9460041. (date of access: 27.12.2023)

21. Fang T.H., Kim Y., Park S.G., Seo K., Park S.H. GPS and eLoran Integrated Navigation for Marine Applications Using Augmented Measurement Equation Based on Range Domain. International Journal of Control, Automation and Systems. Springer. 2020. Vol. 18, № 9. P. 2349–2359. DOI: 10.1007/s12555-019-0287-y. (date of access: 27.12.2023)

22. .Xu K., Zhang G., Jia P., Hao X., Li D. Bias Compensation Model for Sensor Orientation Under Weak Conditions. Geoscience and Remote Sensing Letters. IEEE.

2021. Vol. 18. P. 172–176. DOI: 10.1109/LGRS.2020.2968503. (date of access: 27.12.2023)

23. Zhuang Y. et al. Multi-sensor integrated navigation/positioning systems using data fusion: From analytics-based to learning-based approaches. Information Fusion. Elsevier BV. Vol. 95. 2023. P. 62–90. DOI: 10.1016/j.inffus.2023.01.025. (date of access: 27.12.2023)

ДОДАТОК А

Фільтр Калмана, Matlab

```
clear all;
N=100 % number of samples
a=0.1 % acceleration
sigmaPsi=1
sigmaEta=50;
k=1: N
x=k
x (1) =0
z (1) =x (1) +normrnd (0,sigmaEta);
for t=1: (N-1)
x (t+1) =x (t) +a*t+normrnd (0,sigmaPsi);
z (t+1) =x (t+1) +normrnd (0,sigmaEta);
end;
%kalman filter
xOpt (1) =z (1);
eOpt (1) =sigmaEta; % eOpt (t) is a square root of the error dispersion (variance). It's not a random variable.
for t=1: (N-1)
eOpt (t+1) =sqrt ( (sigmaEta^2) * (eOpt (t) ^2+sigmaPsi^2) / (sigmaEta^2+eOpt (t) ^2+sigmaPsi^2))
K (t+1) = (eOpt (t+1)) ^2/sigmaEta^2
xOpt (t+1) = (xOpt (t) +a*t) * (1-K (t+1)) +K (t+1) *z (t+1)
end;
plot (k,xOpt,k,z,k,x)
```

ДОДАТОК Б

Фільтр Калмана на C#

MathHelper. cs

```
using System;
namespace _KalmanFilterApp
{
public static class MathHelper
{
public static double ToRadian (double degreeAngle)
{
return degreeAngle * Math. PI / 180;
}
public static Coord3D ToCartesian (double h, double lat, double lon)
{
const double a = 6378140.0000;
const double f = 6356755.2881575287;
const double eqr = (a * a - f * f) / (a * a);
var b = ToRadian (lat);
var l = ToRadian (lon);
var sinB = Math. Sin (b);
var cosB = Math. Cos (b);
var sinL = Math. Sin (l);
var cosL = Math. Cos (l);
var n = a / Math. Sqrt (1.0 - eqr * sinB * sinB);
var coord = new Coord3D { X = (n + h) * cosB * cosL, Y = (n + h) * cosB * sinL, Z = (n *
(1.0 - eqr) + h) * sinB };
return coord;
}
public static double ToDecimalAngle (double angle, double minutes, Direction direction)
{
var retAngle = angle + minutes / 60;
if (direction. HasFlag (Direction. W) || direction. HasFlag (Direction. S))
{
return - retAngle;
}
return retAngle;
}
}
}
```

DataReaderHelper. cs

```
using System;
using System. Collections. Generic;
using System. Globalization;
using System. IO;
using System. Linq;
namespace _KalmanFilterApp
{
public static class DataReaderHelper
{
public static IList<Coord3D> Read (Stream stream)
{
```

```
try
{
using (var reader = new StreamReader (stream))
{
var coords = new List<Coord3D> ();
string data = reader. ReadToEnd ();
var separators = new [] { '\r', '\n' };
var lines = data. Split (separators, StringSplitOptions. RemoveEmptyEntries)
. Where (dt =>! string. IsNullOrEmpty (dt) && dt. Contains ("$GPGGA")). ToArray ();
if (lines. Length == 0)
{
throw new Exception ("Data were not found");
}
for (int i = 0; i < lines. Length; i++)
{
var parts = lines [i]. Split (",". ToArray (), StringSplitOptions. RemoveEmptyEntries);
double result;
if (! double. TryParse (parts [2], NumberStyles. AllowDecimalPoint, CultureInfo. InvariantCulture, out result)
||! double. TryParse (parts [4], NumberStyles. AllowDecimalPoint, CultureInfo. InvariantCulture, out result))
{
Logger. Warning (string. Format ("Line {0} is wrong {1}", i + 1, lines [i]));
continue;
}
Direction latitudeDirect, longitudeDirect;
if (! Enum. TryParse (parts [3], out latitudeDirect)
||! Enum. TryParse (parts [5], out longitudeDirect))
{
Logger. Warning (string. Format ("Line {0} is wrong {1}", i + 1, lines [i]));
continue;
}
var geoCoordY = parts [2]. TrimStart ('0');
var geoCoordX = parts [4]. TrimStart ('0');
var latitudeAngle = MathHelper. ToDecimalAngle (
GetDegreeAngle (geoCoordY),
GetMinutes (geoCoordY),
latitudeDirect);
var longitudeAngle = MathHelper. ToDecimalAngle (
GetDegreeAngle (geoCoordX),
GetMinutes (geoCoordX),
longitudeDirect);
double h = double. Parse (parts [9], CultureInfo. InvariantCulture);
coords. Add (MathHelper. ToCartesian (h, latitudeAngle, longitudeAngle));
}
return coords;
}
}
catch (Exception exc)
{
Logger. Error (exc);
throw;
}
```

```
}
private static double GetDegreeAngle (string geoCoord)
{
var part = geoCoord. Split ("."). ToCharArray (), StringSplitOptions. RemoveEmptyEntries)
[0];
var strDegreeAngle = part. Substring (0, part. Length - 2);
var degreeAngle = double. Parse (strDegreeAngle, CultureInfo. InvariantCulture);
return degreeAngle;
}
private static double GetMinutes (string geoCoord)
{
var index = geoCoord. IndexOf ('. ') - 2;
var minutes = double. Parse (geoCoord. Substring (index), CultureInfo. InvariantCulture);
return minutes;
}
}
}
```

KalmanFiltreModel. cs

```
using System;
using System.componentModel;
using System. Linq;
using Emgu. CV;
namespace _KalmanFilterApp
{
public class KalmanFilterModel: IKalmanFilterModel, INotifyPropertyChanged
{
private double _sigmaPsi = Constants. SigmaPsi;
private double _sigmaEta = Constants. SigmaEta;
public Coord3D [] GetFilteredData (Coord3D [] points)
{
int count = points. Length;
if (count == 0)
{
return new Coord3D [0];
}
var coordsX = GetFilteredCoords (points. Select (pt => pt. X). ToArray ());
var coordsY = GetFilteredCoords (points. Select (pt => pt. Y). ToArray ());
var coordsZ = GetFilteredCoords (points. Select (pt => pt. Z). ToArray ());
var coords = new Coord3D [count];
for (int i = 0; i < count; i++)
{
coords [i]. X = coordsX [i];
coords [i]. Y = coordsY [i];
coords [i]. Z = coordsZ [i];
}
return coords;
}
public double SigmaPsi
{
get
```

```
{
return _sigmaPsi;
}
set
{
_sigmaPsi = value;
OnPropertyChanged ("SigmaPsi");
}
}
public double SigmaEta
{
get
{
return _sigmaEta;
}
set
{
_sigmaEta = value;
OnPropertyChanged ("SigmaEta");
}
}
private double [] GetFilteredCoords (double [] coords)
{
var count = coords. Length;
var xOpt = new double [count];
xOpt [0] = coords [0];
var eOpt = new double [count];
eOpt [0] = SigmaEta;
for (int i = 0; i < count - 1; i++)
{
eOpt [i + 1] =
Math. Sqrt (Math. Pow (SigmaEta,
2) *
(Math. Pow (eOpt [i],
2) +
Math. Pow (SigmaPsi,
2)) /
(Math. Pow (SigmaEta,
2) + Math. Pow (eOpt [i],
2) + Math. Pow (SigmaPsi,
2)));
double k = Math. Pow (eOpt [i + 1],
2) / Math. Pow (SigmaEta,
2);
xOpt [i + 1] = xOpt [i] * (1 - k) + k * coords [i + 1];
}
return xOpt;
}
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged (string propertyName)
{
if (PropertyChanged! = null)
{
```

```
PropertyChanged (this, new PropertyChangedEventArgs (propertyName));
}
}
}
}
```

ДОДАТОК В

Основний код ядра для фільтру Калмана

```
#include "mat_op.h"
#include "mat_def.h"
#include "math.h"

#include "stdlib.h"


void mat_init_f32(	matrix_instance_f32 * S,		uint16_t nRows,		uint16_t nColumns,	float * pData)
{
		/* Assign Number of Rows */
		S->numRows = nRows;

		/* Assign Number of Columns */
		S->numCols = nColumns;

		/* Assign Data pointer */
		S->pData = pData;
}
void fill_f32(		float value, float * pDst,		uint32_t blockSize)
{
		uint32_t blkCnt;			/* loop counter */


				/* Run the below code for Cortex-M4 and Cortex-M3 */
		float in1 = value;
		float in2 = value;
		float in3 = value;
		float in4 = value;

		/*loop Unrolling */
		blkCnt = blockSize >> 2u;

		/* First part of the processing with loop unrolling.  Compute 4 outputs at a time.
		** a second loop below computes the remaining 1 to 3 samples. */
		while (blkCnt > 0u)
		{
				/* C = value */
				/* Fill the value in the destination buffer */
				*pDst++ = in1;
				*pDst++ = in2;
				*pDst++ = in3;
				*pDst++ = in4;

				/* Decrement the loop counter */
				blkCnt--;
		}

		/* If the blockSize is not a multiple of 4, compute any remaining output samples here.
		** No loop unrolling is used. */
		blkCnt = blockSize % 0x4u;
		while (blkCnt > 0u)
		{
				/* C = value */
				/* Fill the value in the destination buffer */
				*pDst++ = value;

				/* Decrement the loop counter */
				blkCnt--;
		}
```

```
}
float* matrixInit(matrix_instance_f32 *m, int rows, int cols) {
	float *d;

	d = (float *)malloc(rows*cols*sizeof(float));

	mat_init_f32(m, rows, cols, d);
	fill_f32(0.0, d, rows*cols);
	return d;
}
float* MemInit(int size) {
	float *d;

	d = (float *)malloc(size);

	return d;
}
void matrixFree(matrix_instance_f32 *m)
{
	if (m&&m->pData)
	free((void*)m->pData);
}
void memFree(float *d)
{
	if (d)
		free((void*)d);
}

float nonZero(float x)
{

	//if ((x <  0.00000001f) && (x >  0.0f)) return 0.00000001f;
	//else if ((x <  0.0f) && (x > -0.00000001f)) return -0.00000001f;
	//else
	return x;
}
void mat_mult_f32(matrix_instance_f32 * pSrcA,	matrix_instance_f32 * pSrcB,	matrix_instance_f32 * pDst)
{
	float *pIn1 = pSrcA->pData;		/* input data matrix pointer A */
	float *pIn2 = pSrcB->pData;		/* input data matrix pointer B */
	float *pInA = pSrcA->pData;		/* input data matrix pointer A  */
	float *pOut = pDst->pData;		/* output data matrix pointer */
	float *px;				/* Temporary output data matrix pointer */
	float sum;				/* Accumulator */
	uint16_t numRowsA = pSrcA->numRows;	/* number of rows of input matrix A */
	uint16_t numColsB = pSrcB->numCols;	/* number of columns of input matrix B */
	uint16_t numColsA = pSrcA->numCols;	/* number of columns of input matrix A */



		/* Run the below code for Cortex-M4 and Cortex-M3 */

	float in1, in2, in3, in4;
	uint32_t col, i = 0u, j, row = numRowsA, colCnt;	/* loop counters */


	{
		/* The following loop performs the dot-product of each row in pSrcA with each column in pSrcB
*/
		/* row loop */
		do
		{
			/* Output pointer is set to starting address of the row being processed */
			px = pOut + i;
```

```
                    /* For every row wise process, the column loop counter is to be initiated */
                    col = numColsB;

                    /* For every row wise process, the pIn2 pointer is set
                    ** to the starting address of the pSrcB data */
                    pIn2 = pSrcB->pData;

                    j = 0u;

                    /* column loop */
                    do
                    {
                            /* Set the variable sum, that acts as accumulator, to zero */
                            sum = 0.0f;

                            /* Initiate the pointer pIn1 to point to the starting address of the column
being processed */
                            pIn1 = pInA;

                            /* Apply loop unrolling and compute 4 MACs simultaneously. */
                            colCnt = numColsA >> 2u;

                            /* matrix multiplication        */
                            while (colCnt > 0u)
                            {
                                    /* c(m,n) = a(1,1)*b(1,1) + a(1,2) * b(2,1) + .... +
a(m,p)*b(p,n) */
                                    in3 = *pIn2;
                                    pIn2 += numColsB;
                                    in1 = pIn1[0];
                                    in2 = pIn1[1];
                                    sum += in1 * in3;
                                    in4 = *pIn2;
                                    pIn2 += numColsB;
                                    sum += in2 * in4;

                                    in3 = *pIn2;
                                    pIn2 += numColsB;
                                    in1 = pIn1[2];
                                    in2 = pIn1[3];
                                    sum += in1 * in3;
                                    in4 = *pIn2;
                                    pIn2 += numColsB;
                                    sum += in2 * in4;
                                    pIn1 += 4u;

                                    /* Decrement the loop count */
                                    colCnt--;
                            }

                            /* If the columns of pSrcA is not a multiple of 4, compute any remaining
MACs here.
                            ** No loop unrolling is used. */
                            colCnt = numColsA % 0x4u;

                            while (colCnt > 0u)
                            {
                                    /* c(m,n) = a(1,1)*b(1,1) + a(1,2) * b(2,1) + .... +
a(m,p)*b(p,n) */
                                    sum += *pIn1++ * (*pIn2);
                                    pIn2 += numColsB;

                                    /* Decrement the loop counter */
```

```
                                        colCnt--;
                                    }

                                    /* Store the result in the destination buffer */
                                    *px++ = sum;

                                    /* Update the pointer pIn2 to point to the  starting address of the next
column */
                                    j++;
                                    pIn2 = pSrcB->pData + j;

                                    /* Decrement the column loop counter */
                                    col--;

                                } while (col > 0u);



                                /* Update the pointer pInA to point to the  starting address of the next row */
                                i = i + numColsB;
                                pInA = pInA + numColsA;

                                /* Decrement the row loop counter */
                                row--;

                        } while (row > 0u);
                }


}
void mat_trans_f32(    matrix_instance_f32 * pSrc,         matrix_instance_f32 * pDst)
{
                float *pIn = pSrc->pData;                /* input data matrix pointer */
                float *pOut = pDst->pData;               /* output data matrix pointer */
                float *px;                              /* Temporary output data matrix pointer */
                uint16_t nRows = pSrc->numRows;          /* number of rows */
                uint16_t nColumns = pSrc->numCols;       /* number of columns */

                        /* Run the below code for Cortex-M4 and Cortex-M3 */

                uint32_t blkCnt, i = 0u, row = nRows;         /* loop counters */


                {
                        /* Matrix transpose by exchanging the rows with columns */
                        /* row loop     */
                        do
                        {
                                /* Loop Unrolling */
                                blkCnt = nColumns >> 2;

                                /* The pointer px is set to starting address of the column being processed */
                                px = pOut + i;

                                /* First part of the processing with loop unrolling.  Compute 4 outputs at a time.
                                ** a second loop below computes the remaining 1 to 3 samples. */
                                while (blkCnt > 0u)       /* column loop */
                                {
                                        /* Read and store the input element in the destination */
                                        *px = *pIn++;

                                        /* Update the pointer px to point to the next row of the transposed matrix
*/
                                        px += nRows;
```

```
                                        /* Read and store the input element in the destination */
                                        *px = *pIn++;

                                        /* Update the pointer px to point to the next row of the transposed matrix
*/
                                        px += nRows;

                                        /* Read and store the input element in the destination */
                                        *px = *pIn++;

                                        /* Update the pointer px to point to the next row of the transposed matrix
*/
                                        px += nRows;

                                        /* Read and store the input element in the destination */
                                        *px = *pIn++;

                                        /* Update the pointer px to point to the next row of the transposed matrix
*/
                                        px += nRows;

                                        /* Decrement the column loop counter */
                                        blkCnt--;
                                }

                                /* Perform matrix transpose for last 3 samples here. */
                                blkCnt = nColumns % 0x4u;

                                while (blkCnt > 0u)
                                {
                                        /* Read and store the input element in the destination */
                                        *px = *pIn++;

                                        /* Update the pointer px to point to the next row of the transposed matrix
*/
                                        px += nRows;

                                        /* Decrement the column loop counter */
                                        blkCnt--;
                                }



                                i++;

                                /* Decrement the row loop counter */
                                row--;

                        } while (row > 0u);         /* row loop end  */

                }


}
int qrDecompositionT_f32(matrix_instance_f32 *A, matrix_instance_f32 *Q, matrix_instance_f32 *R) {
                int minor;
                int row, col;
                int m = A->numCols;
                int n = A->numRows;
                int min;

                // clear R
                fill_f32(0, R->pData, R->numRows*R->numCols);
```

```
min = MIN(m, n);

/*
* The QR decomposition of a matrix A is calculated using Householder
* reflectors by repeating the following operations to each minor
* A(minor,minor) of A:
*/
for (minor = 0; minor < min; minor++) {
		float xNormSqr = 0.0f;
		float a;

		/*
		* Let x be the first column of the minor, and a^2 = |x|^2.
		* x will be in the positions A[minor][minor] through A[m][minor].
		* The first column of the transformed minor will be (a,0,0,..)'
		* The sign of a is chosen to be opposite to the sign of the first
		* component of x. Let's find a:
		*/
		for (row = minor; row < m; row++)
			xNormSqr += A->pData[minor*m + row] * A->pData[minor*m + row];

		a = sqrtf(fabsf(xNormSqr));
		if (A->pData[minor*m + minor] > 0.0f)
			a = -a;

		if (a != 0.0f) {
			R->pData[minor*R->numCols + minor] = a;

			/*
			* Calculate the normalized reflection vector v and transform
			* the first column. We know the norm of v beforehand: v = x-ae
			* so |v|^2 = <x-ae,x-ae> = <x,x>-2a<x,e>+a^2<e,e> =
			* a^2+a^2-2a<x,e> = 2a*(a - <x,e>).
			* Here <x, e> is now A[minor][minor].
			* v = x-ae is stored in the column at A:
			*/
			A->pData[minor*m + minor] -= a; // now |v|^2 = -2a*(A[minor][minor])


/*

* Transform the rest of the columns of the minor:

* They will be transformed by the matrix H = I-2vv'/|v|^2.

* If x is a column vector of the minor, then

* Hx = (I-2vv'/|v|^2)x = x-2vv'x/|v|^2 = x - 2<x,v>/|v|^2 v.

* Therefore the transformation is easily calculated by

* subtracting the column vector (2<x,v>/|v|^2)v from x.

*

* Let 2<x,v>/|v|^2 = alpha. From above we have

* |v|^2 = -2a*(A[minor][minor]), so

* alpha = -<x,v>/(a*A[minor][minor])

*/
			for (col = minor + 1; col < n; col++) {
```

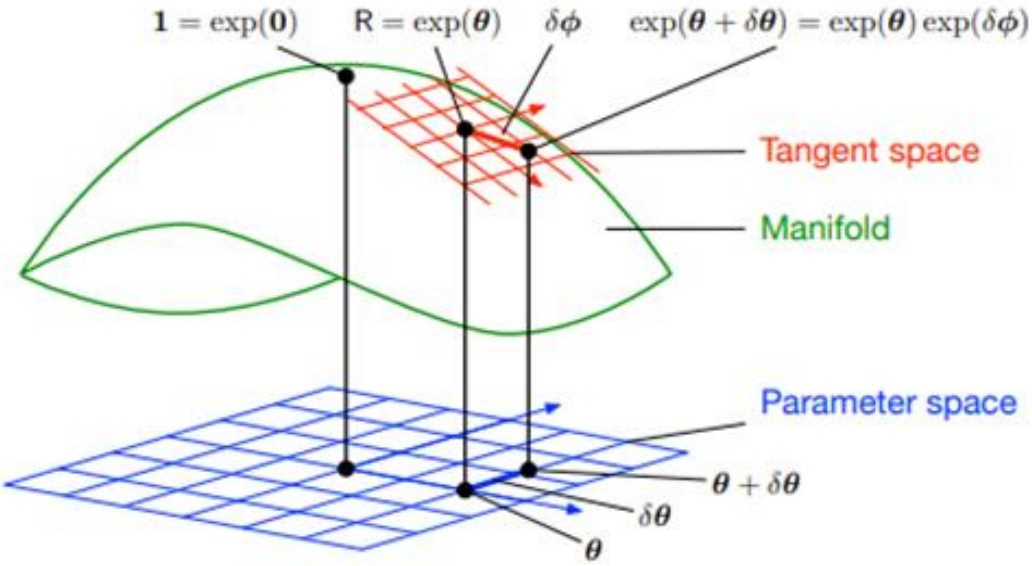
```
                                        float alpha = 0.0f;

                                        for (row = minor; row < m; row++)
                                                alpha -= A->pData[col*m + row] * A->pData[minor*m +
row];

                                        alpha /= nonZero(a*A->pData[minor*m + minor]);

                                        // Subtract the column vector alpha*v from x.
                                        for (row = minor; row < m; row++)
                                                A->pData[col*m + row] -= alpha*A->pData[minor*m + row];
                                }
                        }
                        // rank deficient
                        else
                                return 0;
        }

        // Form the matrix R of the QR-decomposition.
        //      R is supposed to be m x n, but only calculate n x n
        // copy the upper triangle of A
        for (row = min - 1; row >= 0; row--)
                for (col = row + 1; col < n; col++)
                        R->pData[row*R->numCols + col] = A->pData[col*m + row];

        // Form the matrix Q of the QR-decomposition.
        //      Q is supposed to be m x m

        // only compute Q if requested
        if (Q) {
                fill_f32(0, Q->pData, Q->numRows*Q->numCols);

                /*
                * Q = Q1 Q2 ... Q_m, so Q is formed by first constructing Q_m and then
                * applying the Householder transformations Q_(m-1),Q_(m-2),...,Q1 in
                * succession to the result
                */
                for (minor = m - 1; minor >= min; minor--)
                        Q->pData[minor*m + minor] = 1.0f;

                for (minor = min - 1; minor >= 0; minor--) {
                        Q->pData[minor * m + minor] = 1.0f;

                        if (A->pData[minor*m + minor] != 0.0f) {
                                for (col = minor; col < m; col++) {
                                        float alpha = 0.0f;

                                        for (row = minor; row < m; row++)
                                                alpha -= Q->pData[row*m + col] * A-
>pData[minor*m + row];

                                        alpha /= nonZero(R->pData[minor*R->numCols + minor] *
A->pData[minor*m + minor]);

                                        for (row = minor; row < m; row++)
                                                Q->pData[row*m + col] -= alpha*A-
>pData[minor*m + row];
                                }
                        }
                }
        }
        return 1;
}
```

```
void matrixDiv_f32(matrix_instance_f32 *X, matrix_instance_f32 *A, matrix_instance_f32 *B, matrix_instance_f32
*Q, matrix_instance_f32 *R, matrix_instance_f32 *AQ) {
	int i, j, k;
	int m, n;

	// this is messy (going into a class's private data structure),
	// but it is better than malloc/free
	Q->numRows = B->numRows;
	Q->numCols = B->numRows;
	R->numRows = B->numRows;
	R->numCols = B->numCols;
	AQ->numRows = A->numRows;
	AQ->numCols = B->numRows;

	m = A->numRows;
	n = B->numCols;

	qrDecompositionT_f32(B, Q, R);
	mat_mult_f32(A, Q, AQ);

	// solve for X by backsubstitution
	for (i = 0; i < m; i++) {
		for (j = n - 1; j >= 0; j--) {
			for (k = j + 1; k < n; k++)
				AQ->pData[i*n + j] -= R->pData[j*n + k] * X->pData[i*n + k];
			X->pData[i*n + j] = AQ->pData[i*n + j] / nonZero(R->pData[j*n + j]);
		}
	}
}

float dotVector3(float *va, float *vb) {
	return va[0] * vb[0] + va[1] * vb[1] + va[2] * vb[2];
}
void log_mapQuat(float *qIn, float *omega) {

	// define these compile time constants to avoid std::abs:
	static const float twoPi = 2.0f * M_PI, NearlyOne = 1.0f - 1e-8f,
		NearlyNegativeOne = -1.0f + 1e-8f;

	const float qw = qIn[0];
	// See Quaternion-Logmap.nb in doc for Taylor expansions
	if (qw >= NearlyOne) {
		// Taylor expansion of (angle / s) at 1
		// (2 + 2 * (1-qw) / 3) * q.vec();
		float tmp = (8.0f / 3.0f - 2.0f / 3.0f * qw);
		omega[0] = tmp * qIn[1];
		omega[1] = tmp * qIn[2];
		omega[2] = tmp * qIn[3];
	}
	else if (qw <= NearlyNegativeOne) {
		// Taylor expansion of (angle / s) at -1
		// (-2 - 2 * (1 + qw) / 3) * q.vec();
		float tmp = (-8.0f / 3.0f - 2.0f / 3.0f * qw);
		omega[0] = tmp * qIn[1];
		omega[1] = tmp * qIn[2];
		omega[2] = tmp * qIn[3];
	}
	else {
		// Normal, away from zero case
		float angle = 2.0f * acosf(qw), s = sqrtf(1.0f - qw * qw);
		// Important:  convert to [-pi,pi] to keep error continuous
		if (angle > M_PI)
			angle -= twoPi;
		else if (angle < -M_PI)
```

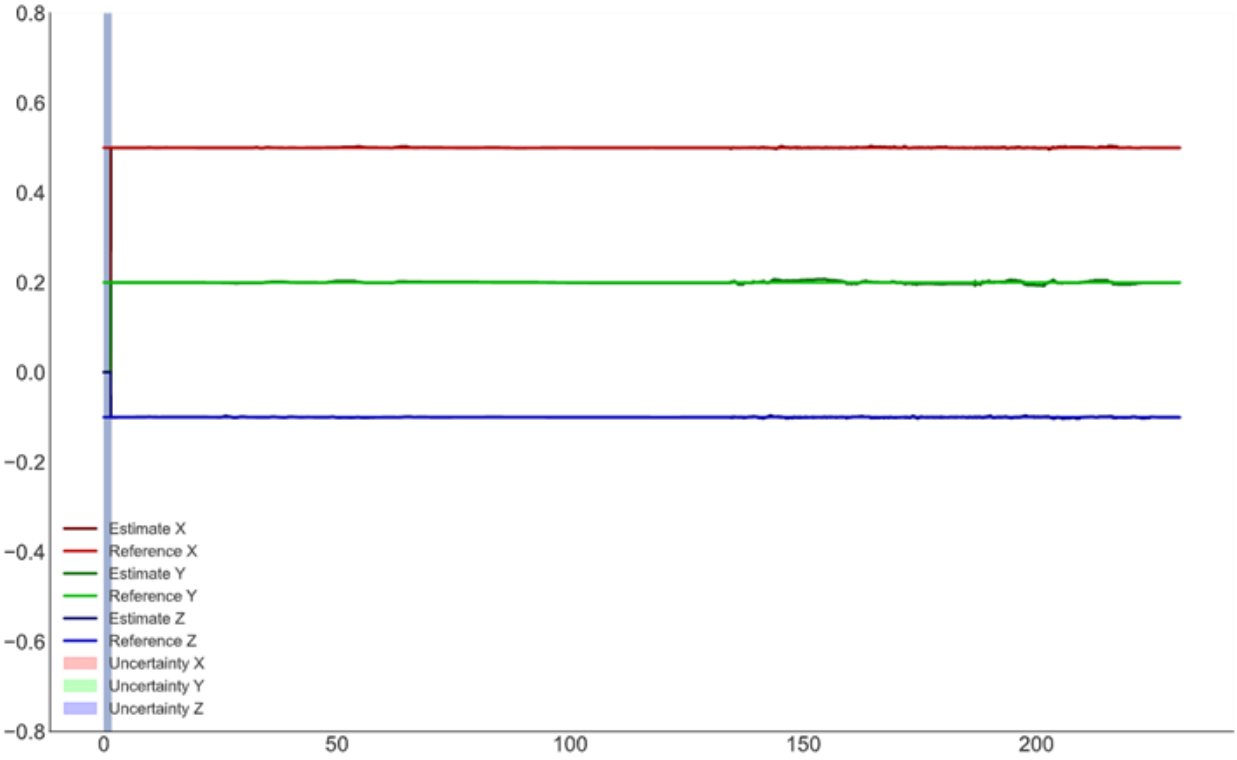
```
                        angle += twoPi;
                float tmp = (angle / s);
                omega[0] = tmp * qIn[1];
                omega[1] = tmp * qIn[2];
                omega[2] = tmp * qIn[3];
        }

}
void exp_mapQuat(float *omega, float *q) {
        float vec[3];
        float theta2 = dotVector3(omega, omega);
        if (theta2 > 5.96e-8f) {
                float theta = sqrtf(theta2);
                float ha = 0.5f * theta;
                float tmp = (sinf(ha) / theta);

                vec[0] = tmp * omega[0];
                vec[1] = tmp * omega[1];
                vec[2] = tmp * omega[2];
                q[0] = cosf(ha);
                q[1] = vec[0];
                q[2] = vec[1];
                q[3] = vec[2];
        }
        else {
                // first order approximation sin(theta/2)/theta = 0.5
                vec[0] = 0.5f * omega[0];
                vec[1] = 0.5f * omega[1];
                vec[2] = 0.5f * omega[2];
                q[0] = 1.0f;
                q[1] = vec[0];
                q[2] = vec[1];
                q[3] = vec[2];
        }
}
void box_plusQuat(float *qIn, float *vec, float *qOut) {
        float qexp[4];

        if ((vec[0] == 0.0f) && (vec[1] == 0.0f) && (vec[2] == 0.0f))
        {
                qOut[0] = qIn[0];
                qOut[1] = qIn[1];
                qOut[2] = qIn[2];
                qOut[3] = qIn[3];
                return;
        }
        exp_mapQuat(vec, qexp);
        quatMultiply(qOut, qexp, qIn);
}

void box_minusQuat(float *q1, float *q2, float *vec) {
        float q2inv[4], qr[4];
        if ((q1[0] == q2[0]) && (q1[1] == q2[1]) && (q1[2] == q2[2]) && (q1[3] == q2[3]))
        {
                vec[0] = 0.0f;
                vec[1] = 0.0f;
                vec[2] = 0.0f;
                return;
        }
        q2inv[0] = q2[0];
        q2inv[1] = -q2[1];
        q2inv[2] = -q2[2];
        q2inv[3] = -q2[3];
        quatMultiply(qr, q1, q2inv);
```

```
		log_mapQuat(qr, vec);
}
void quatMultiply(float *qr, float *q1, float *q2) {
		qr[0] = q2[0] * q1[0] - q2[1] * q1[1] - q2[2] * q1[2] - q2[3] * q1[3];
		qr[1] = q2[0] * q1[1] + q2[1] * q1[0] - q2[2] * q1[3] + q2[3] * q1[2];
		qr[2] = q2[0] * q1[2] + q2[1] * q1[3] + q2[2] * q1[0] - q2[3] * q1[1];
		qr[3] = q2[0] * q1[3] - q2[1] * q1[2] + q2[2] * q1[1] + q2[3] * q1[0];
}
void normalizeVec3(float *vr, float *v) {
		float norm;

		norm = sqrtf(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);

		vr[0] = v[0] / norm;
		vr[1] = v[1] / norm;
		vr[2] = v[2] / norm;
}
void normalizeQuat(float *qr, float *q) {
		float norm;

		norm = 1.0f / sqrtf(q[0] * q[0] + q[1] * q[1] + q[2] * q[2] + q[3] * q[3]);

		qr[0] *= norm;
		qr[1] *= norm;
		qr[2] *= norm;
		qr[3] *= norm;
}
void crossVector3(float *vr, float *va, float *vb) {
		vr[0] = va[1] * vb[2] - vb[1] * va[2];
		vr[1] = va[2] * vb[0] - vb[2] * va[0];
		vr[2] = va[0] * vb[1] - vb[0] * va[1];
}
void rotateVectorByQuat(float *vr, float *v, float *q) {
		float w, x, y, z;

		w = q[0];
		x = q[1];
		y = q[2];
		z = q[3];

		vr[0] = w * w*v[0] + 2.0f*y*w*v[2] - 2.0f*z*w*v[1] + x * x*v[0] + 2.0f*y*x*v[1] + 2.0f*z*x*v[2] - z *
z*v[0] - y * y*v[0];
		vr[1] = 2.0f*x*y*v[0] + y * y*v[1] + 2.0f*z*y*v[2] + 2.0f*w*z*v[0] - z * z*v[1] + w * w*v[1] -
2.0f*x*w*v[2] - x * x*v[1];
		vr[2] = 2.0f*x*z*v[0] + 2.0f*y*z*v[1] + z * z*v[2] - 2.0f*w*y*v[0] - y * y*v[2] + 2.0f*w*x*v[1] - x *
x*v[2] + w * w*v[2];
}
void rotateVectorByRevQuat(float *vr, float *v, float *q) {
		float qc[4];

		qc[0] = q[0];
		qc[1] = -q[1];
		qc[2] = -q[2];
		qc[3] = -q[3];

		rotateVectorByQuat(vr, v, qc);
}
void rotateVecByMatrix(float *vr, float *v, float *m) {
		vr[0] = m[0 * 3 + 0] * v[0] + m[0 * 3 + 1] * v[1] + m[0 * 3 + 2] * v[2];
		vr[1] = m[1 * 3 + 0] * v[0] + m[1 * 3 + 1] * v[1] + m[1 * 3 + 2] * v[2];
		vr[2] = m[2 * 3 + 0] * v[0] + m[2 * 3 + 1] * v[1] + m[2 * 3 + 2] * v[2];
}
void rotateVecByRevMatrix(float *vr, float *v, float *m) {
		vr[0] = m[0 * 3 + 0] * v[0] + m[1 * 3 + 0] * v[1] + m[2 * 3 + 0] * v[2];
```

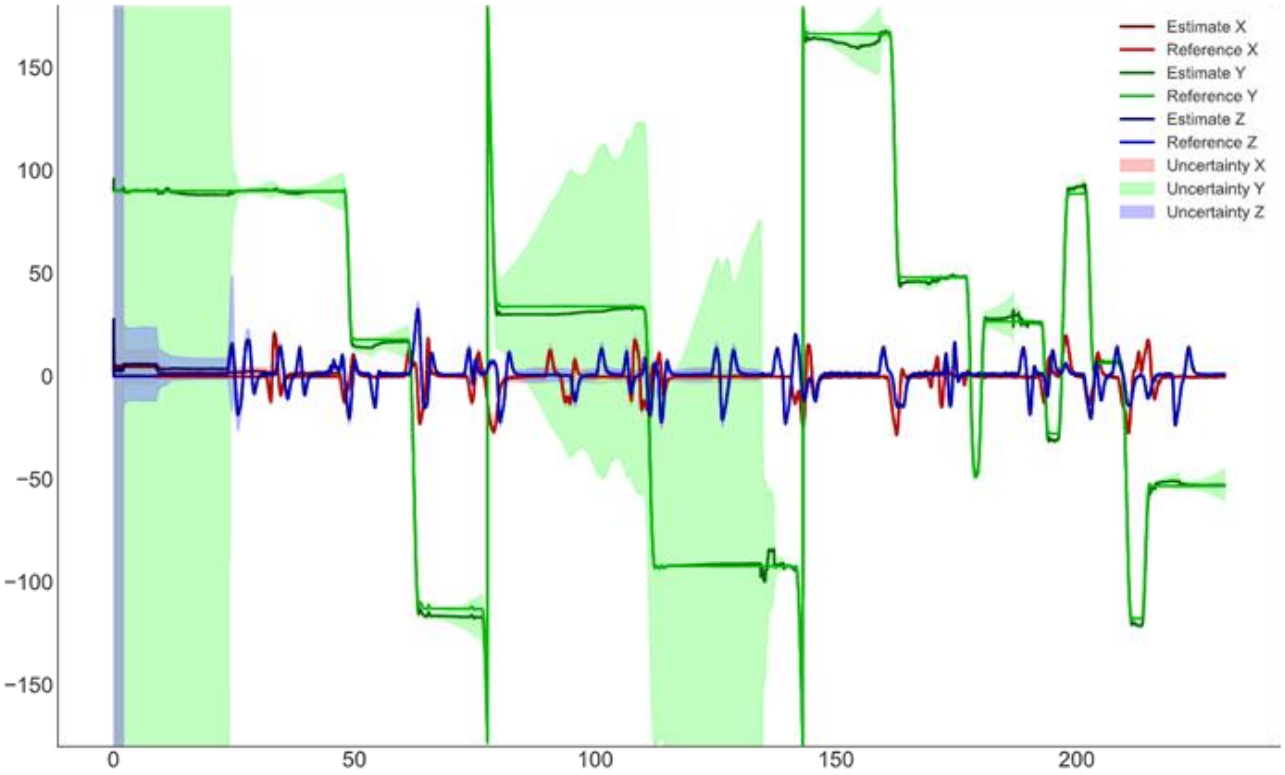
```
		vr[1] = m[0 * 3 + 1] * v[0] + m[1 * 3 + 1] * v[1] + m[2 * 3 + 1] * v[2];
		vr[2] = m[0 * 3 + 2] * v[0] + m[1 * 3 + 2] * v[1] + m[2 * 3 + 2] * v[2];
}
void quatToMatrix(float *m, float *q, int normalize) {
		float sqw = q[0] * q[0];
		float sqx = q[1] * q[1];
		float sqy = q[2] * q[2];
		float sqz = q[3] * q[3];
		float tmp1, tmp2;
		float invs;

		// get the invert square length
		if (normalize)
			invs = 1.0f / (sqx + sqy + sqz + sqw);
		else
			invs = 1.0f;

		// rotation matrix is scaled by inverse square length
		m[0 * 3 + 0] = (sqx - sqy - sqz + sqw) * invs;
		m[1 * 3 + 1] = (-sqx + sqy - sqz + sqw) * invs;
		m[2 * 3 + 2] = (-sqx - sqy + sqz + sqw) * invs;

		tmp1 = q[1] * q[2];
		tmp2 = q[3] * q[0];
		m[1 * 3 + 0] = 2.0f * (tmp1 + tmp2) * invs;
		m[0 * 3 + 1] = 2.0f * (tmp1 - tmp2) * invs;

		tmp1 = q[1] * q[3];
		tmp2 = q[2] * q[0];
		m[2 * 3 + 0] = 2.0f * (tmp1 - tmp2) * invs;
		m[0 * 3 + 2] = 2.0f * (tmp1 + tmp2) * invs;

		tmp1 = q[2] * q[3];
		tmp2 = q[1] * q[0];
		m[2 * 3 + 1] = 2.0f * (tmp1 + tmp2) * invs;
		m[1 * 3 + 2] = 2.0f * (tmp1 - tmp2) * invs;
}

void rotateQuat(float *qOut, float *qIn, float *rate) {
		float q[4];
		float r[3];

		r[0] = rate[0] * -0.5f;
		r[1] = rate[1] * -0.5f;
		r[2] = rate[2] * -0.5f;

		q[0] = qIn[0];
		q[1] = qIn[1];
		q[2] = qIn[2];
		q[3] = qIn[3];

		// rotate
		qOut[0] = q[0] + r[0] * q[1] + r[1] * q[2] + r[2] * q[3];
		qOut[1] = -r[0] * q[0] + q[1] - r[2] * q[2] + r[1] * q[3];
		qOut[2] = -r[1] * q[0] + r[2] * q[1] + q[2] - r[0] * q[3];
		qOut[3] = -r[2] * q[0] - r[1] * q[1] + r[0] * q[2] + q[3];
}
```



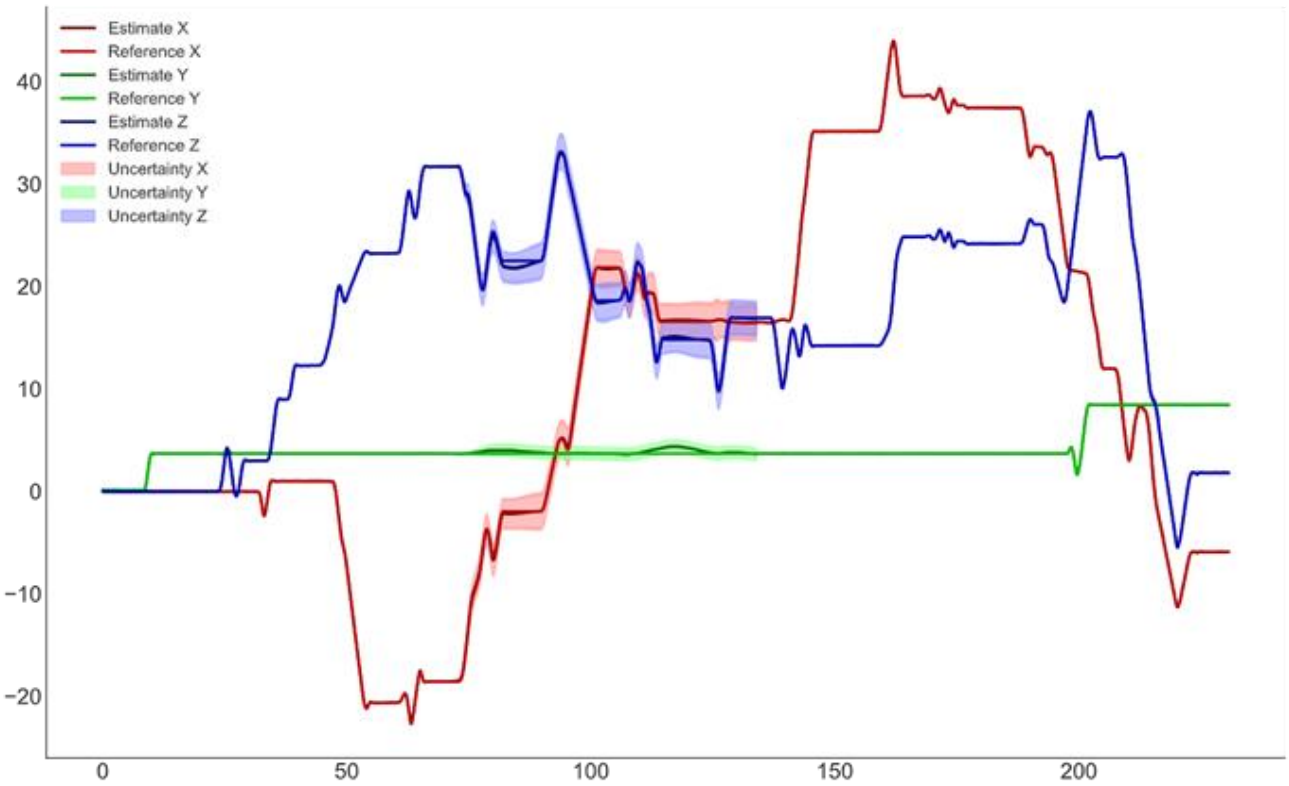
					Бакалаврська кваліфікаційна робота	
Зм.	Лист	Прізвище	Підпис	Дата	Гіперпараметризована модель обертання алгоритму фільтра Калмана для безпілотних об'єктів	Літера: Н; Мас; Масштаб
Дипломник		Рибачук О.				
Керівник		Спас Н.Я.				Аркуш 1; Аркушів 6
Консул.						
Консул.						Національний університет “Львівська політехніка” ІКНІ, КН-412
Реценз.						
Зав.каф.		Лобур М.В.				



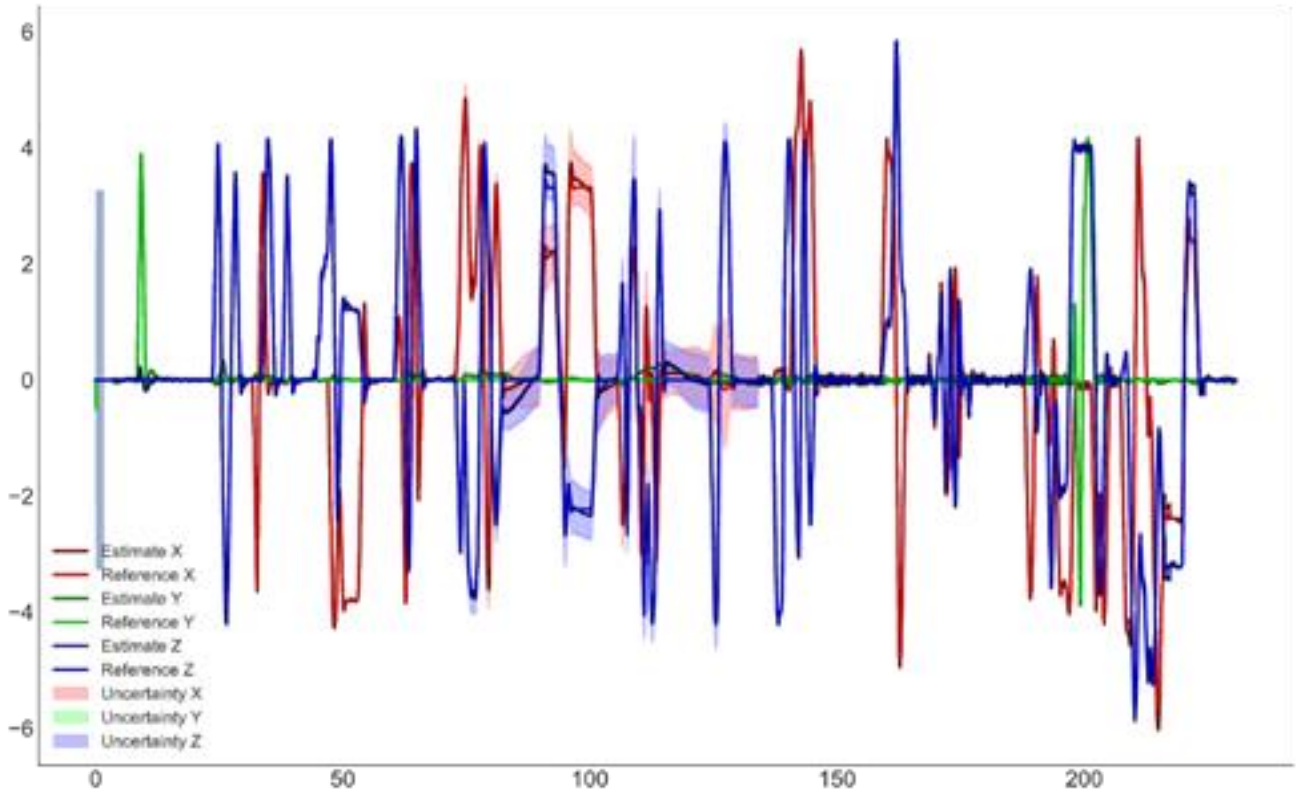
					Бакалаврська кваліфікаційна робота	
Зм.	Лист	Прізвище	Підпис	Дата	Вектор зсувів супутникової навігаційної системи	Літера: Н; Мас; Масштаб
Дипломник		Рибачук О.				Аркуш 2; Аркушів 6
Керівник		Спас Н.Я.				Національний університет “Львівська політехніка” ІКНІ, КН-412
Консул.						
Консул.						
Реценз.						
Зав.каф.		Лобур М.В.				



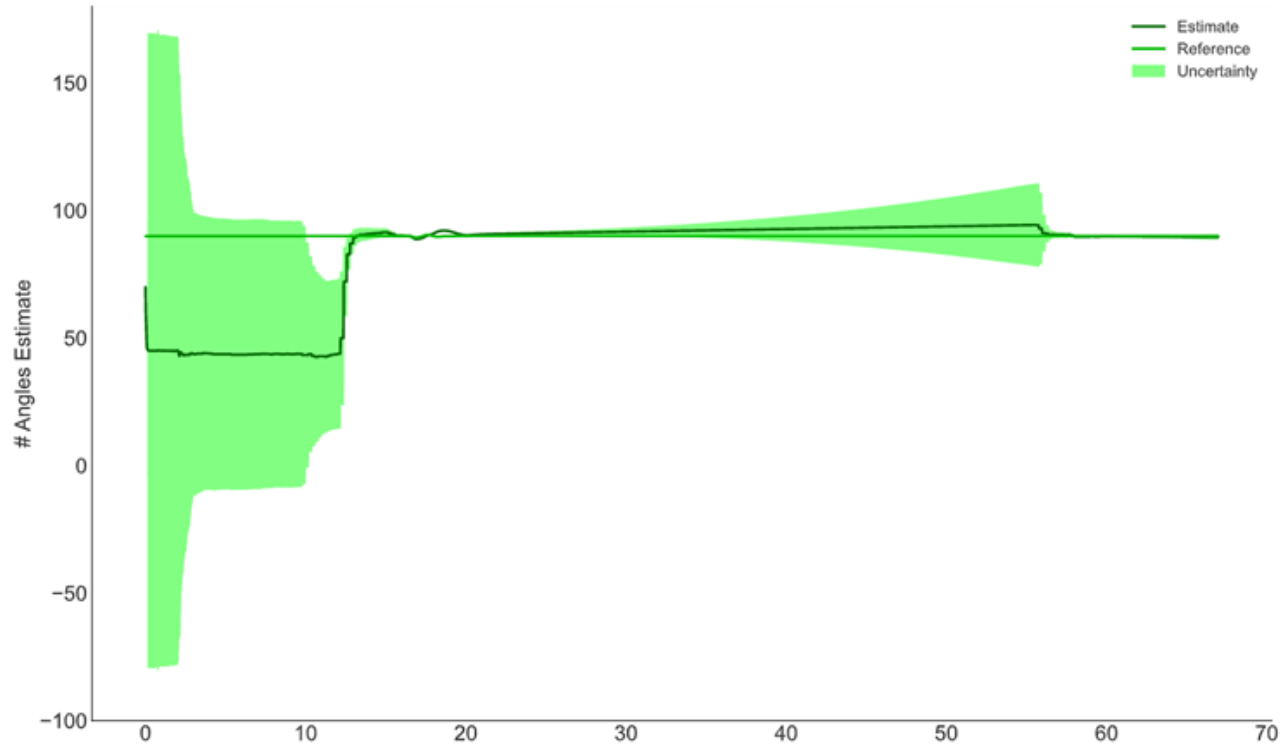
					Бакалаврська кваліфікаційна робота		
					Графік кутів Ейлера у фільтрі Калмана для безпілотних об'єктів	Літера: Н	Мас:	Масштаб:
Зм.	Лист	Прізвище	Підпис	Дата		Аркуш 3	Аркушів 6
Дипломник		Рибачук О.				Національний університет “Львівська політехніка” ІКНІ, КН-412	
Керівник		Спас Н.Я.					
Консул.							
Консул.							
Реценз.							
Зав.каф.		Лобур М.В.					



					Бакалаврська кваліфікаційна робота		
					Вектор положення безпілотних об'єктів	Літера	Мас / Масштаб
Зм.	Лист	Прізвище	Підпис	Дата		Н	
Дипломник		Рибачук О.				Аркуш 4	Аркушів 6
Керівник		Спас Н.Я.				Національний університет “Львівська політехніка” ІКНІ, КН-412	
Консул.							
Консул.							
Реценз.							
Зав.каф.		Лобур М.В.					



					Бакалаврська кваліфікаційна робота		
					Вектор швидкостей безпілотних об'єктів	Літера	Мас / Масштаб
Зм.	Лист	Прізвище	Підпис	Дата		Н	
Дипломник		Рибачук О.				Аркуш 5	Аркушів 6
Керівник		Спас Н.Я.				Національний університет “Львівська політехніка” ІКНІ, КН-412	
Консул.							
Консул.							
Реценз.							
Зав.каф.		Лобур М.В.					



					Бакалаврська кваліфікаційна робота		
					Кут курсу безпілотного об'єкту з початковою помилкою в 45 градусів	Літера	Мас / Масштаб
Зм.	Лист	Прізвище	Підпис	Дата		Н	
Дипломник		Рибачук О.				Аркуш 6	Аркушів 6
Керівник		Спас Н.Я.				Національний університет “Львівська політехніка” ІКНІ, КН-412	
Консул.							
Консул.							
Реценз.							
Зав.каф.		Лобур М.В.					