Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
(повна назва інституту/факультету)

Кафедра телекомунікацій
(повна назва кафедри)

«На правах рукопису»
УДК _____

До захисту допущено
В.о. завідувача кафедри

_____ Явіся В.С.___
(підпис) (ініціали, прізвище)
“___”_____2018_р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 172 Телекомунікації та радіотехніка,
(код і назва)

спеціалізація Апаратно-програмні засоби електронних комунікацій

на тему: Дослідження можливості використання диференційної GPS поправки в мобільних пристроях Android.

Виконав (-ла): студент (-ка) _2_ курсу, групи ТЗ-71мп
(шифр групи)

Тимошенко Вадим Русланович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник професор, д.т.н., Ільченко М.Ю., _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ с.н.с, к.т.н Міночкін Д.А.___ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018

ЗМІСТ

ВСТУП

На сьогодні в світі використовується більше 5 мільярдів пристроїв з підтримкою GNSS, 80% з яких це смартфони. Більш ніж 50% додатків на Google Play та App Store використовують інформацію про місцезнаходження користувача. Вища точність визначення місцезнаходження, що може бути отримана на масовому ринку, буде лише підвищувати використання сматфонів та портативних пристроїв в напів-професійних застосуваннях для місцевизначення різноманітних об’єктів.

Типова точність мобільних пристроїв на масовому ринку знаходиться в діапазоні від декількох метрів до десятків метрів при складних умовах, таких як висока щільність забудови. Однак, одночасте використання декількох GNSS систем, двочастотних чіпсетів та використання зовнішньої інформації може підвищити точність до рівня менше метру.

Одним із способів отримання зовнішньої інформації є використання систем диференціальної корекції - методів поліпшення характеристик роботи навігаційної системи, такі, як точність, надійність і доступність, через інтеграцію зовнішніх даних в процесі розрахунку. Суть більшості методів диференціальної корекції полягає в обліку навігаційною апаратурою різного роду поправок, що одержані з альтернативних джерел. Джерелами корегувальної інформації є контрольно-коригуючі станції (ККС), опорні координати яких відомі з високою точністю. Як правило методи диференціальної корекції забезпечують поправками обмежену територію Землі. Каналами доставки даних диференціальної корекції можуть бути різними, традиційно це УКВ, стільниковий і супутниковий зв'язок.

Системи диференціальної корекції існують доволі тривалий час, проте застосування самих корекцій вимагає дорогих GPS приймачів. Необхідною умовою використання диференційних поправок є доступ до необробленої навігаційної інформації від GNSS-супутників. До 2016 року виробники GNSS чіпсетів для мобільних пристроїв не надвали доступ операційній системі

Android та клієнтським додаткам до необробленої навігаційної інформації, що унеможливлювало використання різного роду диференційних поправок. Однак, у травні 2016 року компанія Google оголосила, що починаючи з Android версії 7 мобільні додатки отримають доступ до необроблених GNSS вимірювань. Що дозволяє застосовувати власні алгоритми, для підвищення точності позиціонування, в Android додатках.

Незважаючи на переваги та можливості, що надає застосування необроблених GNSS вимірювань, воно не набуло широкого розповсюдження. Для цього є дві причини: по-перше, стандартні формати, такі як RINEX або NMEA, недоступні на платформі Android, тому неможливо використати вже напрацьовані алгоритми для обробки необроблениъ даних; по-друге, через новизну даного функціоналу в ОС Android не має готових рішень, які вже адаптовані під відповідні нові формати.

Таким чином, дослідження використання диференційних поправок в мобільних пристроях на базі ОС Android є актуальним завданням.

1. Глобальна система навігації

1.1 Архітектура GPS

Супутникова система навігації - це комплексна електронно-технічна система, що складається із сукупності наземного і космічного обладнання, призначена для визначення місцезнаходження (географічних координат і висоти) та часу, а також параметрів руху (швидкості і напрямку руху) для наземних, водних і повітряних об'єктів.

Глобальна система позиціонування (GPS) - супутникова радіо навигаційна

система, розроблена Міністерством оборони (DoD) США. Система використовує супутникові сузір'я розташовані на середньо земній орбіті, які передають мікрохвильові сигнали, що дозволяють GPS приймачам визначати місцеположення, швидкість та час.

Основа системи GPS тісно пов'язана із запуском першого радянського супутника в 1957 році, оскільки супутникова орбіта може бути визначена шляхом спостереження за ефектом Допплера. Використовуючи ці знання, США розробили супутникову систему, що називається TRANSIT. З TRANSIT позиція на землі може бути визначена шляхом вимірювання доплерівських зсувів сигналів. Але ця система не мала чітких пристроїв синхронізації на борту супутників, і розрахунок позиції приймача займав близько 15 хвилин. У 1973 році Міністерство оборони (DoD) вирішило розробити супутникову навігаційну систему на основі системи TRANSIT. З'явилася концепція системи GPS. У 1977 році були здійснені перші випробування приймачів з використанням псевдолітів (псевдосупутників), тобто передавачів, встановлених на земній поверхні. Перший оперативний супутник GPS був запущений в 1978 році. До 1985 року було запущено 11 супутників блоку I. У 1989 році був активований новий тип супутника, був запущений перший супутник II блоку. У 1993 році система досягла повного 24-супутникового сузір'я. У цьому році також було дозволено безкоштовне використання системи в цивільних цілях. У 1995 році була досягнута повна оперативна

спроможність (FOC) [1]. До 2000 року у систему вводилась спеціально похибка, що знижувала точність визначення координат цивільними особами. Згодом було оголошено деактивацію вибіркової доступності, що призвело до підвищення точності для цивільних користувачів від приблизно 100 м до 20 м. У 2005 році модернізація системи GPS розпочалася запуском першого супутника типу IIR-M, який підтримує новий військовий M-сигнал і другий цивільний сигнал L2C [2].

Основна концепція GPS полягає в тому, що сигнали передаються з космічних супутників і вони приймаються приймачами на або поблизу поверхні Землі. Використання інформації про час дозволяє визначити відстань від кожного супутника і тим самим, використовуючи процес триангуляції та знання позицій супутника, можна визначити положення приймача на Землі. Всі супутники надсилають інформацію про час передачі сигналу, тому приймач знає, коли повідомлення було відправлено. Оскільки радіосигнали поширюються зі швидкістю світла, шлях від супутника до приймача займає дуже короткий, проте визначений час. Супутники також передають інформацію про свої позиції. Таким чином, приймач здатний розрахувати відстань від супутника до приймача. Щоб отримати повне виправлення широти, довготи і висоти, потрібні чотири або більше супутників, а коли приймач знаходиться на відкритій місцевості, в зоні видимості весь час знаходиться більше чотирьох супутників. В ідеальних умовах виправлення лише широти і довготи можна отримати з трьох супутників.

За даними The Aerospace Corporation і Trimble, GPS-технологію можна виділити три сегменти:

- Космічний сегмент.
- Контрольний сегмент.
- Сегмент користувача

1.1.1 Космічний сегмент

Супутники GPS знаходяться на одній з шести орбіт. Вони знаходяться в площинах, які нахилені приблизно на 55 ° відносно екваторіальної площини, на на кожній орбіті знаходиться чотири супутники. Таке розташування забезпечує користувачам видимість від п'яти до восьми супутників у будь-який час з будь-якої точки Землі.

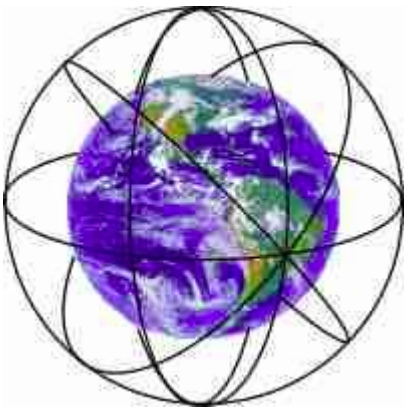


Рис. 1.1 Орбіти GPS супутників

Їх орбіти контролюються, оскільки відхилення від орбіти супутника перетворюються на помилки в визначенні місцезнаходження. Сигнали з часом передачі також точно контролюються. Супутники містять атомний годинник, щоб сигнали, які вони передають, мали високу точність. Тим не менш, ці годинники мають незначний дрейф, і для подолання цього, використовуються коригуючі сигнали від земних станцій.

Самі GPS супутники мають конструктивний термін у десять років, але для забезпечення цілісності сузір’я в разі несподіваних збоїв, запасні супутники знаходяться на орбіті і можуть бути введені в експлуатацію протягом короткого часу. Супутники забезпечують власне живлення через свої сонячні батареї. Вони простягаються приблизно до 5 метрів і генерують 700 Вт, необхідні для живлення супутника та його батарей, коли воно знаходиться під сонячним світлом. Супутник повинен працювати, коли він знаходиться на темній стороні Землі, коли сонячні панелі не виробляють енергію. Це означає, що під час наявності сонячного проміння сонячні панелі

повинні забезпечувати додаткову енергію для зарядки акумуляторів, а не лише живлення основних супутникових схем [3].

Використовується кілька різних позначень для ідентифікації супутників на їх орбітах. Одна номенклатура присвоює літеру кожної орбітальної площині (тобто A, B, C, D, E і F) до кожного супутника у цій площині, супутники позначаються номером від 1 до 4. Таким чином, супутник, на який позначається B3, відноситься до 3-го супутника в площині орбіти B.

Друге позначення, що використовується є супутниковим номером NAVSTAR, призначеним ВВС США. Це позначення у формі номера космічного апарату (SVN) 11 для позначення супутника NAVSTAR 11.

Третє позначення являє собою конфігурацію генератору (PRN) псевдовипадкового коду на борту супутника. Ці генератори коду PRN налаштовуються однозначно на кожному супутнику, таким чином створюючи унікальні версії як C/A-коду, так і P (Y)-коду. Таким чином, супутник може бути ідентифікований за кодами PRN, які він генерує [4].

Супутники постійно транслюють дві несучі хвилі. Ці несучі хвилі знаходяться в L-діапазоні (використовуються для радіо). Ці хвилі походять від основної частоти, створеної дуже точним атомним годинником:
• Носій L1 транслюється на частоті 1575,42 МГц (10,23 x 154)
• Носій L2 транслюється на частоті 1227,60 МГц (10,23 x 120).
Несучі можуть бути модульовані наступними кодами:

- C/A кодом
- P-кодом
- Y-кодом

Грубий код (C/A код) являє собою код псевдовипадкового шуму (PRN), який передається на частоті 1.023 МГЦ. Цей код повторюється кожну мілісекунду. Рівняння для декодування C/A коду відомі і незасекречені. Тому C/A код доступний для цивільного використання. C/A код використовується багатьма цивільними приймачами для потреб навігації і картографування.

Р-код - це другий код псевдовипадкового шуму (PRN), яким модулюються GPS сигнали. Він передається на частоті 10.23 МГц. Період повторення сигналу становить 267 днів. Цикл в 267 днів розділений на 38 сегментів по 7 днів. Шість з цих сегментів зарезервовані для експлуатаційних цілей або не використовуються. Кожен з 32-х залишилися 7-ми денних сегментів присвоєно певного супутника. Тому, кожен супутник має свій, властивий тільки йому, код. Рівняння для декодування Р-коду відомі і не засекречені, вони доступні для цивільного використання [5].

У-код можна вважати зашифрованою версією Р-коду. В дійсності, це Р-код, модульований за допомогою W-коду, який являє собою спеціальну послідовність шифрування. Рівняння, декодування У-коду засекречені і відомі тільки зареєстрованим користувачам.

1.1.2 Контролюючий сегмент

Контролюючий сегмент складається з однієї головної контрольної станції, 5-и моніторингових станцій та 4-ох наземних антен, розподілених між 5 станціями приблизно на екваторі Землі. Цей сегмент відповідальність за підтримку супутників та їх належне функціонування. Це включає в себе підтримку супутників у їх належних орбітальних позиціях та моніторингу стану та охорони супутникової підсистеми. Також відстежується технічний стан супутників, тобто стан супутникових сонячних батарей, рівні заряду акумуляторів та рівні палива, що використовуються для маневрів. Контролюючий сегмент відповідає за оновлення годинників кожного супутника, ефемерид, альманаху та інших індикаторів в навігаційному повідомленні раз на день або за потребою. Параметри ефемерид точно пов’язані з орбітами GPS супутників і дійсні лише в інтервалі часу від 4 до 6 годин залежно від часу з останнього завантаження контрольного сегменту.

Залежно від версії супутника, дані навігаційного повідомлення можуть зберігатися щонайменше від 14 днів до максимально 210 днів у 4-ох або 6-и годинних інтервалах. Альманах - це зменшена точна підмножина параметрів

ефемерид. Альманах складається з 7 із 15 параметрів орбіти ефемеридів. Дані альманаху використовуються для прогнозування приблизного положення супутника та допомоги у виборі супутникового сигналу. Крім того, вирішуються аномалії супутника, проводить вимірювання псевдодальностей і вимірювання різниці в відстані на віддалених станціях моніторингу, щоб визначити необхідні коригування супутникових годинників, альманаху та ефемерид. Для виконання вищевказаних функцій сегмент управління складається з трьох різних фізичних компонентів: основної контрольної станції (MCS), станцій моніторингу та наземних антен.

MCS це центр операцій контрольного сегменту і знаходиться на базі ВПС Falcon, Колорадо-Спрінгс, штат Колорадо. Станції моніторингу пасивно відслідковують супутники GPS, коли вони з’являються в зоні видимості та вимірювюють псевдодальності та похибку. Ці вимірювання проводяться за допомогою сигналів L1 та L2 GPS. Необроблені дані, а також отримане навігаційне повідомлення та місцеві дані про погоду, передаються в MCS через оборонну систему супутникової комунікації та інші наземні системи зв'язку. MCS обробляє дані від станцій моніторингу для контролю супутникової навігації. Дані з усіх станцій моніторингу використовуються для створення корекції супутникових годинників, ефемерид та альманахів даних для кожного супутника [6].

Станції моніторингу формують набір даних для контрольного сегмента. Станція моніторингу містить двочастотний GPS-приймач (L1/L2), який безперервно робить вимірювання псевдодальностей і похибки до кожного супутника. Точно відоме розташування фазового центру антени приймача. Станція моніторингу також містить два цезієвих годинники, на яких базується час GPS-системи. Вимірювання псевдодальності та похибки, зроблені для кожного супутника в зоні видимості, приймачем станції моніторингу, оновлює статистику фільтра Калмана на контрольній станції, щодо оцінки позиції, швидкості та часу (PVT) кожного супутника. Цей фільтр оновл*юється* кожні 15 хвилин, під час визначення координат супутника. Даний процес забезпечує

точну оцінку ефемериди супутника та зсуву годинника під час збору даних. Для користі користувачеві GPS, ці позиції повинні бути передбачені наперед. Позиція супутника та часові поправки передбачаються наперед з використанням точних моделей супутника та його середовища [4].

Антена підземної лінії зв'язку забезпечує засоби управління та керування супутниками та завантаження навігаційних повідомлень та інших даних. Наземна антена передачі зберігає та завантажує дані, які називаються даними TT & C (телеметрія, відстеження та командування). Унікальний набір даних TT & C (який включає навігаційне повідомлення) формується MCS для кожного супутника. Ці дані пересилаються на земну антену з MCS і зберігаються до появи конкретного супутника. Лінія передачі даних S-діапазону використовується для передачі даних на супутник для пересилання до процесору навігаційного супутника [6].

1.1.3 Сегмент користувача

Приймальне обладнання користувача, як правило, називається "GPS-приймачем", обробляє сигнали L-діапазону, що надсилаються з супутників, для визначення користувачем позиції, швидкості та часу. У технології приймальних пристроїв GPS спостерігається значна еволюція, паралельно з електронікою в цілому. Перші приймальні пристрої, виготовлені в середині 1970-х років як частина фази перевірки концепції, були в основному аналоговими пристроями для військового застосування, які були великими, громіздкими та важкими. Завдяки сучасній технології GPS приймач із подібними або більшими можливостями, як правило, важить кілька фунтів і займає невеликий обсяг. Структурна схема приймального набору GPS показана на рисунку 2. Комплект GPS складається з п'яти головних компонентів: антени, приймача, процесора, пристрою вводу / виводу та блоку живлення.

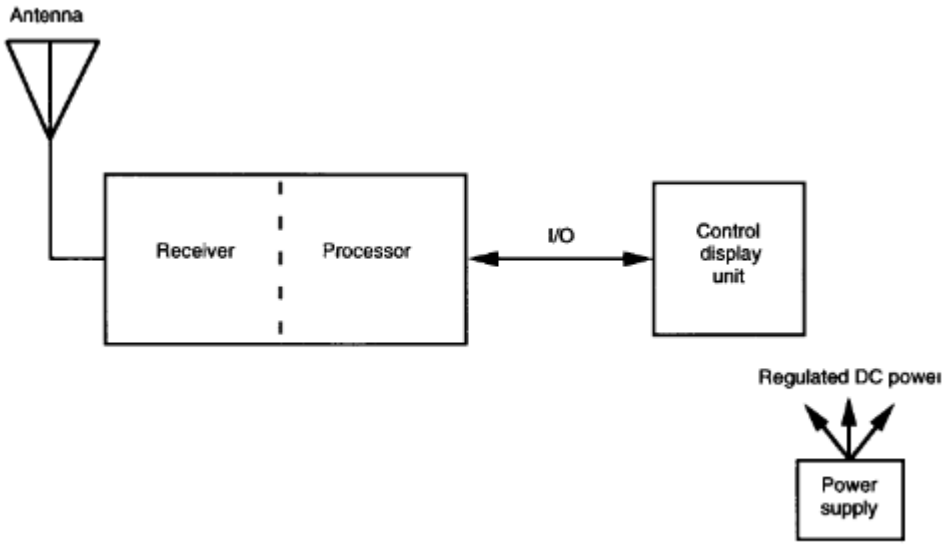


Рис. 1.2 Структурна схема GPS приймача

Супутникові сигнали приймаються через антену, яка має праву кругову поляризацію. З діаграмою спрямованості рівною 160°. GPS-приймачі, які відстежують P (Y) - код на L1 і на L2 частотах, повинні мати пропускну здатність ріну 20,46 МГц на обох частотах. Якщо приймач відстежує лише C/A-код на L1, то антена повинна мати пропускну здатність принаймні 2,046 Мгц. Приймачі розділяються на два основні типи: ті, які відслідковують як P (Y) -код, так і C/A-код, і ті, що відстежують тільки C/A-код. Сам прийчам відповідає за обробку та декодування прийнятих сигналів. Процесор, як правило, виконує функції керування приймачем, починаючи з отримання сигналу, продовжуючи відстеженням сигналу та збором даних. Крім того, процесор може також сформувати рішення PVT з вимірювань приймача [6].

1.2 GPS Час

Хоча GPS найкраще відома своїми можливостями позиціонування, вона також є одним з найбільш доступних та надійних джерел точного часу. Час повинен бути визначений узгоджено та уніфіковано. Тривалість секунди Temps Atomique International (TAI) була визначена в 1967 році як тривалість 9 192 631 770 періодів випромінювання, що відповідає переходу між двома надтонкими рівнями основного стану атома цезію 133 [7].

Універсальний час (UT) - це стандарт сонячного часу, що відображає середню швидкість обертання Землі, її найбільш часто використовуване визначення називається UT1. Координований універсальний час (UTC) - це компроміс між TAI та UT1. Фактично, як атомний час, UTC є таким же рівномірним, як може бути шкала TAI. Проте, для того, щоб слідувати варіаціям обертання Землі, він завжди знаходиться в межах 0,9 секунд відносно UT1. Це досягається шляхом додавання або віднімання певної кількості стрибкових секунд до/від TAI.

Основні поняття позиціонування GPS базуються на термінах - час прибуття та час передачі сигналів. GPS використовує свою систему еталонного часу - GPS Time (GPST).

Час GPS (GPST) безперервний, без стрибків секунд. Його відлік розпочався з опівночі (0 г UTC) з 5 на 6 січня 1980 року. У цю епоху різниця між TAI та UTC становила 19 с [25].

GPST передається за допомогою двох параметрів: номер тижня (WN) та час тижня (TOW). Перший розраховує тижні, що пройшли з початку часу GPS до поточного тижня. Тижні починаються опівночі в суботу. TOW підраховує секунди, що пройшли протягом поточного тижня.

У таблиці 1.1 представлені співвідношення між часовими інтервалами GPS та TAI, а також співвідношенням між UTC та TAI. На рисунку 3 показана різниця часу між еталонними часами та TAI.

Таблиця 1.1 Співвідношення часу GPS - TAI та UTC - TAI

Система	Взаємозв'язок
GPST – TAI	$TAI = GPST + 19s$
UTC – TAI	$UTC = TAI - leapsecond_{UTC-TAI}$

Відмінність часу у секундах від UTC до TAI визначається як $\text{leapsecond}_{UTC\text{-}TAI}$. Проте спільнота GNSS описує стрибок секунди як різницю часу між UTC і GPS. Різниця під може бути виражена як:

$$leapsecond = leapsecond_{UTC-TAI} - 19s \quad (1.1)$$

1.3. Навігаційне повідомлення та стан відстеження

Приймач повинен синхронізуватися із сигналом, що передає супутник. В даний час GPS передає чотири різні сигнали в L1 [9], причому сигнал кодування Coarse/Acquisition (C/A) є найважливішим і найбільш часто використовуваним сигналом для пристроїв масового ринку. Код PRN базується на коді тривалістю 1 мс на швидкості 1,023 чіпа на мс. Локальна копія коду C/A генерується з належною затримкою та Доплеровським зсувом. На рисунку 1.4 показана ідеальна синхронізація між періодичним кодом C/A, переданим супутником (повторюється кожні 1 мс), і його локальною реплікою, сформованою приймачем (синій колір). На цьому етапі приймач не може забезпечити повноцінну оцінку часу передачі, і доступні лише часткові дані. Наприклад, при $t0$ приймач може лише надати відносну затримку до початку періоду коду C/A. Точно так само при $t1$ приймач забезпечує затримку до початку поточного коду C/A. Кількість цілих кодів C/A між $t0$ та $t1$ невідома.

Якщо приймач синхронізується лише з кодом C/A, дійсний діапазон переданого часу становить від 0 до 1 мс, що робить обчислення часу передачі неоднозначним.

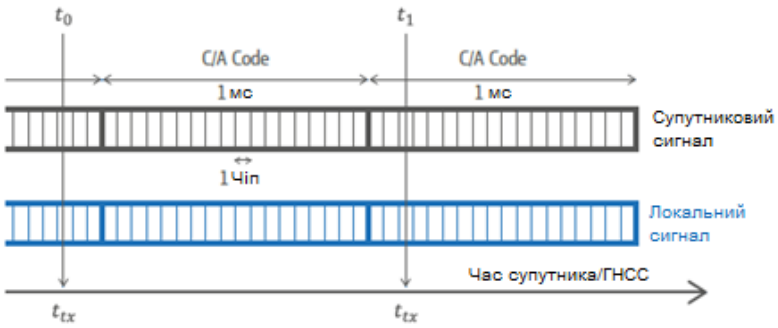


Рис. 1.3. Синхронізація з кодом C / A

Щоб вирішити неоднозначність цього часу, приймач може використовувати структуру навігаційного повідомлення. На малюнку 1.5 показані різні етапи синхронізації, необхідні приймачу для отримання часу супутника:

- Code Lock: приймач заблоковано до коду C / A. Дійсний діапазон 0-1 мс.
- Bit Sync: приймач синхронізується з бітами. Дійсний діапазон становить 0-20 мс.
- Синхронізація підкадрів: приймач синхронізується з підкадрами. Діючий діапазон становить 0-6 с.
- TOW декодовані: усі субкадри містять TOW. Тому після розшифровки TOW дійсний діапазон становить 0-1 тиждень.

В літературі описуються більш складні алгоритми, здатні обчислювати рішення PVT, використовуючи тільки часткову псевдодальність. Це навігаційне рішення відоме як проблема грубої навігації [10]. Крім того, приймачі A-GNSS також можуть отримати TOW через зовнішні системи і не потребують декодування переданого повідомлення.

У випадку з GPS, п'ять послідовних субкадрів утворюють кадр, а навігаційне повідомленням складається з 25 кадрів.

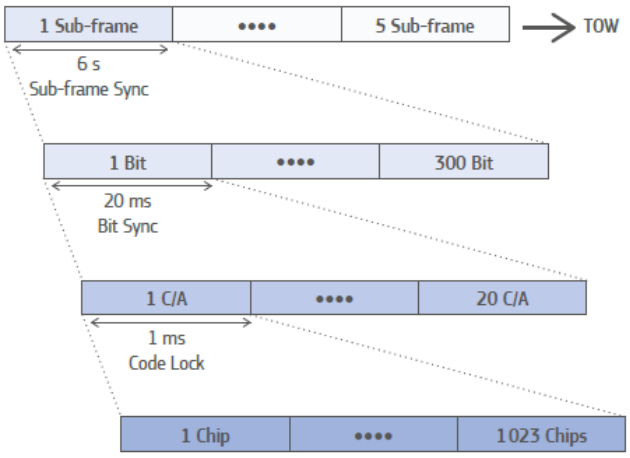


Рис. 1.4. Код L1 C/A системи GPS та структура навігаційного повідомлення

Частина навігаційного повідомлення являє собою набір номерів, які називаються ефемеридами, які разом описують орбіту супутника в просторі та позиції супутника на цій орбіті в певний момент час. Результатом є набір координат x, y та z, де супутник був, коли сигнал був переданий. Ці значення вказують на позицію супутника щодо системи координат, визначеної

Всесвітньою системою геодезичних параметрів Землі 1984 року (далі — WGS 84). Центр цієї системи координат знаходиться поблизу маси центру Землі, а її вісь z відповідає середній вісі обертання Землі. Додатній напрямок вісі z спрямований до північного полюса; додатній напрямок x з'являється з Землі на Грінвічському мередіані на екваторі (на південь від Гани і на захід від Габону, в Атлантичному океані). Вісь y з'являється на екваторі на 90-му меридіані східної довготи (у точці Індійського океану, південно-східний напрямок від Шрі-Ланки і на захід від Суматри), що визначається правою системою координат [11].

1.4 Визначення псевдодальностей

Незважаючи на те, що поняття псевдодальності є простим, їх розрахунок вимагає вимірювань часу. Приймачі GNSS обробляють прийняті сигнали для отримання часу передачі (t_{Tx}) та часу прийому (t_{Rx}). Різниця між ними - час проходження сигналу від супутника до приймача (при відсутності додаткових затримок через іоносферу, тропосферу та інших елементів). Псевдодальність може бути розрахована як:

$$\rho = (t_{Rx} - t_{Tx}) \cdot \mathrm{c}, \tag{1.2}$$

*д*е c — швидкість світла,

t_{Tx} — час передачі,

t_{Rx} — час прийому

Розрахунок псевдодальності та обчислення PVT можуть бути реалізовані за допомогою двох різних методів: використовуючи загальний час прийому або загальний час передачі [12].

1.4.1 Розрахунок на основі загального часу прийому

Загальний час прийому відомий як час вимірювання (t_{meas}^{Rx}). Чотири супутники передають сигнали одночасно, всі супутники передають однаковий TOW і той же час епохи (зліва на рисунку 7). Через різні шляхи поширення, всі сигнали надходять до приймача з різними затримками. Після чого приймач

обчислює часовий зсув між TOW і поточним часом епохи. Це відомо як час вимірювання (t_{meas}^{Rx}). В результаті, час, що передається супутником (наданий у системному часі GPS), в момент вимірювання може бути виражений як:

$$t_{Tx,Sat1} = t_{TOW}^{GNSS} + \Delta_1, (1.3)$$

$$t_{Tx,Sat2} = t_{TOW}^{GNSS} + \Delta_2, (1.4)$$

$$t_{Tx,Sat3} = t_{TOW}^{GNSS} + \Delta_3, (1.5)$$

$$t_{Tx,Sat3} = t_{TOW}^{GNSS} + \Delta_3, (1.6)$$

де t_{TOW}^{GNSS} це переданий TOW,

Δ_i це затримка між TOW та часом вимірювання.

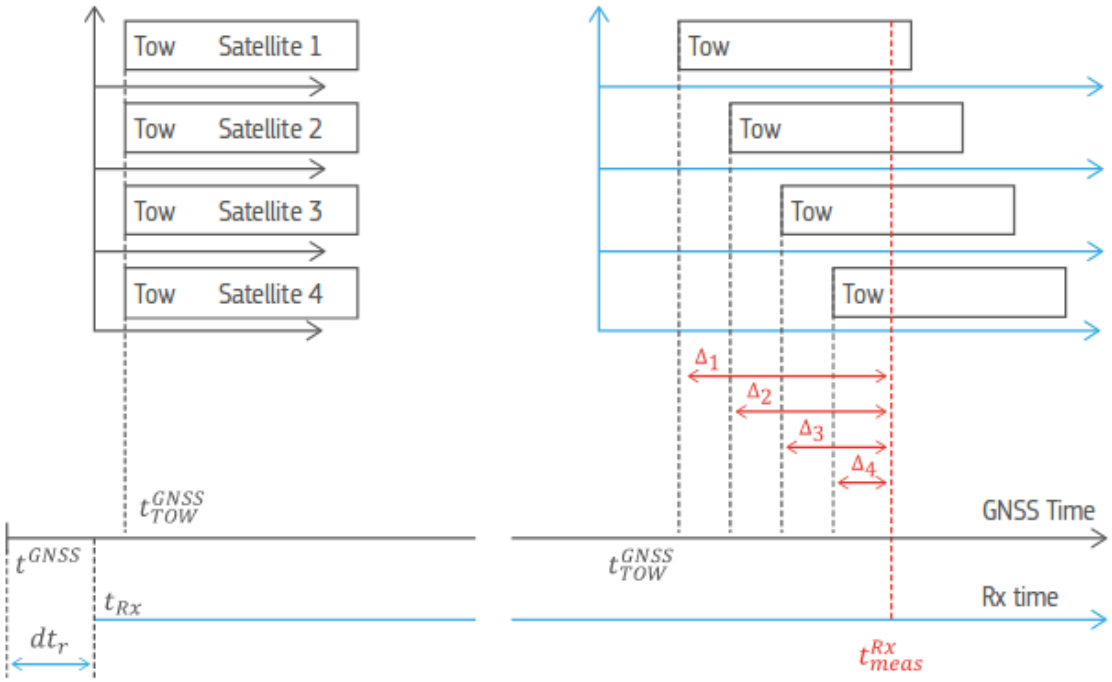


Рис. 1.5 Визначення псевдодальності на основі загального часу прийому

Перед тим як обчислювати перший PVT і декодувати TOW з принаймні одного супутника, приймач не має інформації про час GPS. Таким чином, приймач повинен робити припущення, щоб генерувати час прийому і обчислити перший набір псевдодальностей. Перший супутниковий сигнал, що надходить, використовується як опорний. Отриманий час (t_{meas}^{Rx}) - це час, що пройшов, плюс час опорного супутника (t_{ref}^{path}). Зазвичай передбачається стандартне значення від 65 до 85 мс. Тому перший виміряний час можна обчислити як

$$t_{meas}^{Rx}[1] = t_{Tx,Sat1}[1] + t_{ref}^{path} \text{ (1.7)}$$

Всі інші псевдодальності утворюються відносно першого. Оскільки присутня постійна помилка для всіх супутників (оскільки час розповсюдження був вибраний, не оцінений), вводиться упередженість між GNSS та отриманим часом (dt_r). Розрахований час в одиницях еталонного часу GNSS може бути виражений таким чином:

$$t_{meas}^{Rx,GNSS}[k] = t_{meas}^{Rx}[k] - dt_r[1], \text{ (1.8)}$$

де k - k-й час вимірювання.

Після того як був обчислений перший набір псевдодальностей і отриманий перший PVT, dt_r визначається і використовується для всіх наступних наборів псевдодальностей. Проте приймач обчислює та оновлює часову поправку у всіх наборах PVT. Отже, наступні псевдодальності можна обчислити як:

$$\rho_1[k] = \left(t_{meas}^{Rx}[k] - t_{Tx,Sat1}[k] + dt_r[1]\right)c, \text{ (1.9)}$$

$$\rho_2[k] = \left(t_{meas}^{Rx}[k] - t_{Tx,Sat2}[k] + dt_r[1]\right)c, \text{ (1.10)}$$

$$\rho_3[k] = \left(t_{meas}^{Rx}[k] - t_{Tx,Sat3}[k] + dt_r[1]\right)c, \text{ (1.11)}$$

$$\rho_4[k] = \left(t_{meas}^{Rx}[k] - t_{Tx,Sat4}[k] + dt_r[1]\right)c, \text{ (1.12)}$$

де c - швидкість світла у вакуумі.

1.4.2 Розрахунок з використанням загального часу передачі

На відміну від часу прийому, цей метод базується на часі передачі супутників. Фактично, як згадувалося раніше, всі супутники транслюють дані синхронно, але через різні затримки поширення користувач не отримує дані з кожного супутника одночасно. Коли еталонний біт (наприклад, перший початок підкадру (TLM)) ідентифікується в кожному каналі, приймач порівнює його з часом прибуття того ж еталону з іншого супутника. Як і в попередньому методі, канал з найменшим лічильником часу вибирається як еталон, його затримка поширення вибирається в діапазоні між 65 і 85 мілісекундами (приблизний час розповсюдження сигналів GPS), і всі інші канали будуть мати більшу затримку поширення, яка і буде вимірюватися. Це поняття накреслено на рисунку

На рисунку 1.6 опорний біт встановлено рівним найлегше ідентифікованому TLM, що прийшов по першому каналу. Якщо вважається, що TLM ідентифікований для всіх відслідкованих супутників, то приймач визначає різницю часу між часом приймоу кожного TLM відносно часу прийому еталонного супутника.

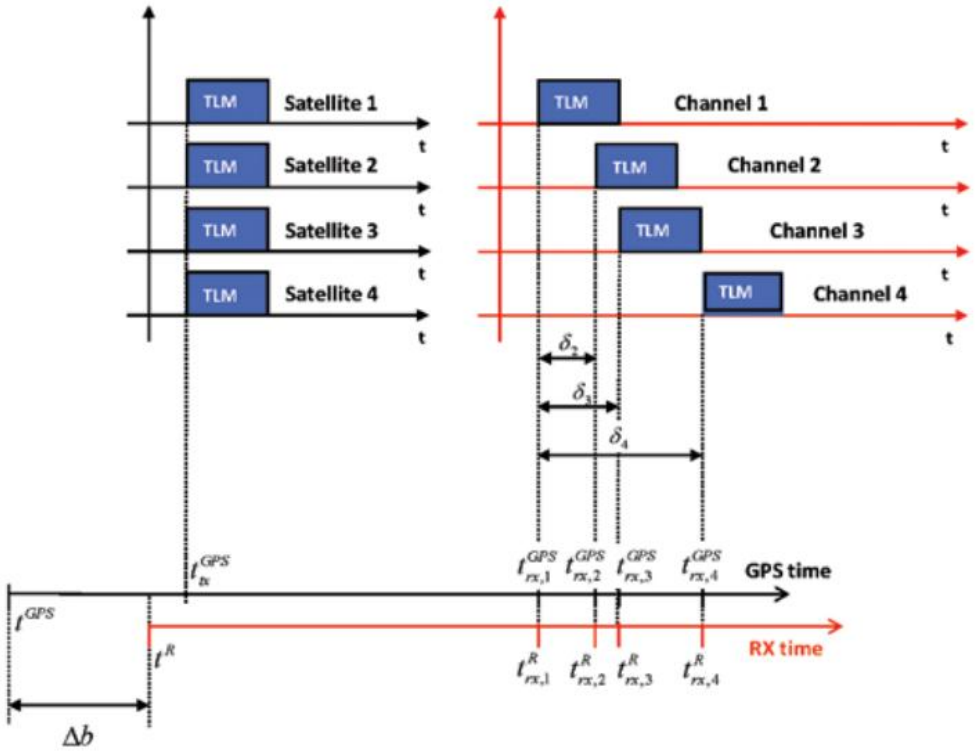


Рис. 1.6 Визначення псевдодальності на основі загального часу передачі

Посилаючись на рисунок 1.6, ця різниця часу δ, може бути записана як:

$$\delta i = (t_{Rxi} - t_{Rx1}), \text{(1.13)}$$

де $t_{Rx}i$ являє собою час прийому субкадра для і-го супутника, тоді як $t_{Rx}1$, це час прийому субкадру еталонного супутника.

Виходячи з цього псевдодальності можуть бути записані як:

$$\rho i = \rho_1 + c\Delta b + c\delta i \text{ (1.14)}$$

де Δb - це зміщення часу між годинником приймача та супутника,

с - швидкість світла,

ρ 1 - це псевдодальність еталонного каналу,

δі це різниця часту між і-им супутниковим каналом та еталонним

Цей підхід має деякі недоліки, коли він реалізується в режимі реального часу:

а) Необхідно чекати, доки всі канали не отримали однаковий біт даних для обчислення псевдодальностей.

б) Оскільки еталонні сигнали з кожного супутника надходить в різні моменти часу, в загальному випадку приймач матиме різні часові помилки для кожного розрахунку, що виникає внаслідок впливу частотного зсуву генератора. Тим не менш, якщо генератор приймача є достатньо стабільним, це не призведе до значних помилок.

в) У випадку одночасного використання декількох супутникових систем GPS / Galileo, реалізація цього підходу в режимі реального часу неможлива, тому що одержувачу доведеться працювати з двома незалежними супутниковими системами, які мають різні структури даних, а також використовувати два еталонних канали (один для GPS і один для Galileo). Отже, такий підхід змусить приймач зберігати велику кількість інформації в буфері з значною витратою ресурсів, що також означає затримку обчислення позиції, швидкості та часу (PVT) [12].

1.5 Визначення позиції

Визначивши псевдодальність до супутника, використовуючи просту геометрію, приймач визначає, що він знаходиться на поверхні сфери, з центром на цьому супутнику та радіусом рівним відстані від супутника. Дану модель зображено на рисунку N. Якщо годинник приймача точно синхронізований з часом GPS, приймач може негайно обчислити його положення за допомогою простої алгебри.

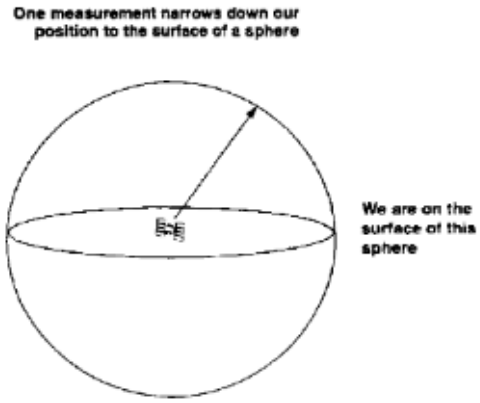


Рисунок 1.7 Умовна сфера на поверхні якої знаходиться користувач

Годинник приймача зазвичай має певні погрішності відносно часу GPS. Таким чином, псевдодальність складається не тільки з того, яку відстань пройшов сигнал від супутника до приймача, але і наскільки час приймача та час GPS відрізняються. Це називається зсувом годинника і являє собою четверту невідому (разом з координатами x, y та z приймача). Зсув годинника може бути як позитивним, так і негативним, оскільки годинник приймача може випереджати, або відставати від часу GPS. Оскільки ми знаємо, що сигнал пройшов до приймача за швидкістю світла (близько 300 000 000 метрів на секунду), можливо шкидко перетворити його на відстань, помноживши його на швидкість світла. Аналогічно, часовий зсув вимірюється одиницями часу, а також може бути перетворений на відстань. Ця відстань або помилка часу є загальною для всіх псевдодальностей, оскільки приймач використовує один і той же годинник, щоб вимірювати всі значення псевдодальності.

Маючи розраховані псевдодальності та місцезнаходження супутника в момент передачі повідомлення, визначене з навігаційного повідомлення, можливо розрахувати місцезнаходження приймача. Використовуючи відповідну математику, приймач обчислює свою позицію (x, y та z) та часовий зсув (Δt), процес визначення відображений на рисунку 3. Точки A, B та C - розташування трьох супутників у просторі координат x, y діаграми. Радіуси

d1, d2 та d3 представляють собою псевдодальності, які були виміряні для кожного супутника.

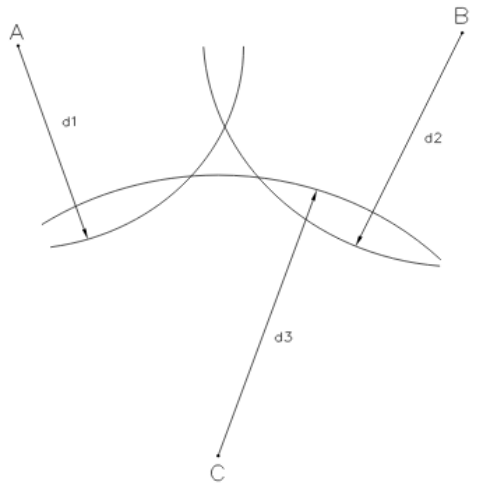


Рисунок 1.8 Визначення місцезнаходження приймача

Таким чином, позиція визначається як така, що розташована на перехресті трьох сфер, з центом що відповідає кожному супутнику та з радіусом, рівним відповідним псевдодальностям. В ідеальних умовах існуватиме лише дві точки перетину, одна з яких буде знаходитись біля поверхні Землі, а інша в космосі. Друга точка перетину відкидається. Але наявність похибки призводить до того, що три сфери не перетинаються в одній точці. Вони перетинаються, утворюючи трикутник.

Коли GPS-приймач отримує ряд вимірювань, які не перетинаються в одній точці, він починає віднімати (або додавати) час, поки він не отримає відповідь, що дозволить радіусам від усіх супутників проходити через одну точку. Виходячи з цього приймач визначає необхідний часовий зсув та вносить відповідні коригування.

На наступному зображенні відображена поправка Δt для кожної псевдодальності. Результат полягає в тому, що було скоригувано кожну псевдодальність на одне значення Δt, що призводить до того, що сфери перетинаються в одній точці. Координати цього перетину відображають позицію приймач, а Δt - часовий зсув. В результаті цього процесу визначається позиція приймача, а також визначається правильний час [13].

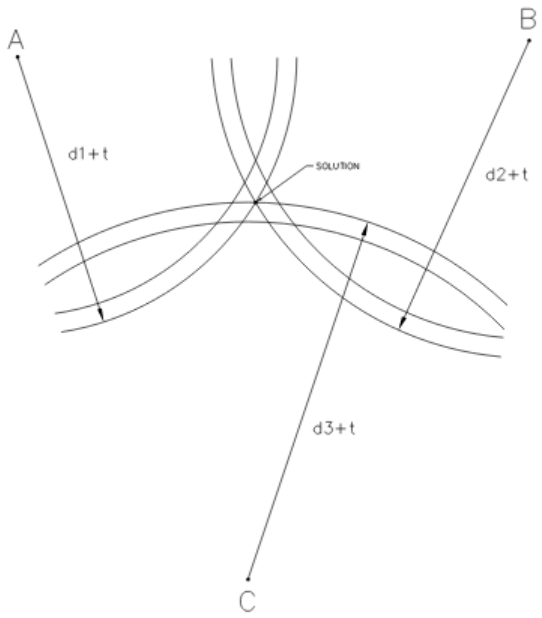


Рисунок 1.9 Застосування часового зсуву для визначення

1.6 Джерела похибок

Точність, з якою приймач може визначати своє положення або швидкість, чи синхронізувати час системи GPS, залежить від складної взаємодії різних факторів. Наприклад, відомий користувачеві зсув супутникових годинників відносно часу системи GPS, або компенсація помилки розповсюдження супутникового зв'язку. Відповідні помилки викликані кожним сегментом (контролюючим, космічним та сегментом користувача).

Процес передачі, прийому та виявлення сигналу GPS - це фізичний процес, який, як і будь-який інший фізичний процес, містить джерела помилок. Деякі помилки очевидні: супутникові годинники не є точно правильними, навіть якщо для коректування використовуються коригуючі сигнали. Місце розташування супутника в космосі не обов'язково правильне, оскільки це визначається лише спостереженнями, зробленими на землі, і ефемерідні значення отримують в межах точності приблизно 30 см. Приймач, що обчислює власну позицію, може лише вирішувати отримані сигнали з певною точністю, яка обумовлена довжиною хвилі або несучої (для вимірів фази несучої) або довжиною бітового коду (для кодового рішення), а також роздільну здатність коду або фазовий зсув у приймачі. Подальші обмеження

виникають у приймачі на основі точності розрахунків, де математичні процеси можуть скоротитись або бути округлені.

Деякі інші джерела помилок з'являються, під час розгляданні фізичного процесу передачі сигналу. Наприклад, супутник передає сигнал, що рухається за високих швидкостей у космосі. Оскільки малоймовірно, що приймач рухається в одному і тому ж напрямку і швидкості, буде відбуватися Доплерівський ефект, який впливає на ефективну довжину хвилі як коду, так і хвилі носія. Також сигнал повинен проходити через іоносферу над Землею, що має ефект переміщення сигналу і згинання його шляху. Він також проходить через тропосферу (нижній шар атмосфери, в якій відбувається більша частина погодніх явищ), що також впливає на шлях і швидкість сигналу. Оскільки сигнал супроводжує приймач, деякі з них можуть відбивати землю, воду або будівлі, водонапірні вежі, знаки, розташовані біля приймача, і досягати антени після розповсюдження на більшу відстань, ніж сигнал, який надходить безпосередньо від супутника (явище, яке називається багатопроменевістю сигналів).

1.6.1 Похибки годинників

Супутники містять атомні годинники, які контролюють всі операції синхронізації на борту, включаючи генерацію широкомовного сигналу. Хоча ці годинники дуже стійкі, поля поправки годинника в навігаційному повідомленні розраховуються таким чином, що відхилення між часом супутника та часом GPS може становити до 1 мс [14]. (Зсув на 1 мс призводить до 300-кілометрової похибки псевдодальності.) MCS визначає та передає параметри корекції годинника на супутники для ретрансляції в навігаційному повідомленні.

Приймачі використовують менш точні годинники, що може спричинити похибки в обчисленнях псевдодальностей. Рисунок відображає вплив похибки годинника на результат визначення псевдодальностей та розрахунку місцезнаходження приймача. Де P1, P2, P3 — умовні позначення супутників,

а дуги — проекція сфери з центром на супутнику та радіусом рівним визначеному значенню псевдодальності.

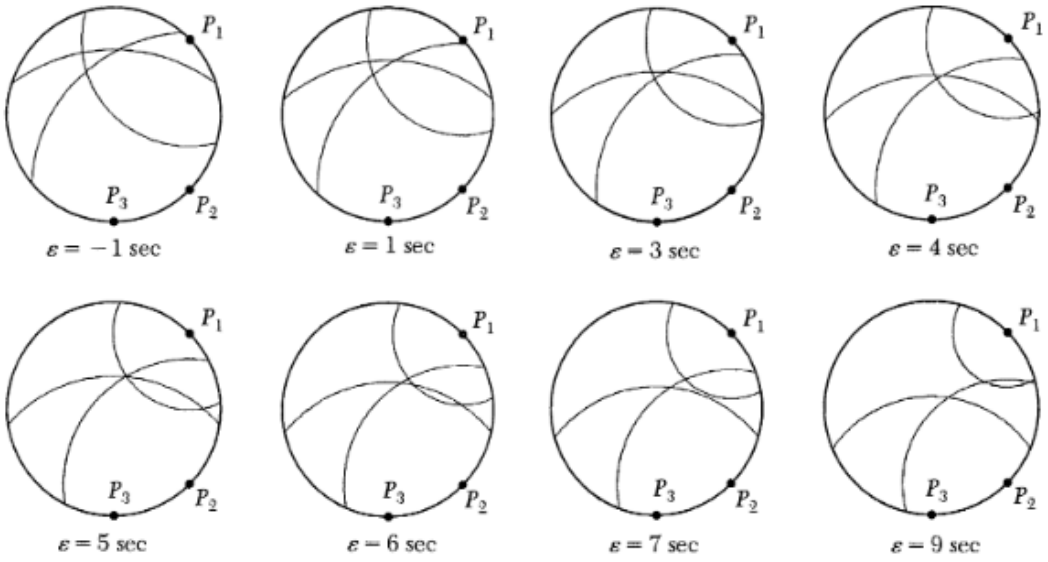


Рис. 1.10 Вплив похибки годинника на визначення псевдодальності.

1.6.2 Похибка ефемерид

Приймач очікує, що кожен супутник знаходиться в певному місці в певний час. Кожну годину або близько того, у своєму повідомленні, супутник повідомляє приймачу, де передбачається, що він буде в часі "t", отже. Якщо цей прогноз ефемериди невірний і супутник знаходиться не у передбаченому місці, то вимірювання відстані від антени приймача до супутника буде містити похибку.

Оцінки ефемерид для всіх супутників обчислюються та передаються до супутників разом з іншими параметрами навігаційного повідомлення для повторного передавання їх користувачу. Ці поправки генеруються на основі передбачення контролюючим сегментом позиції кожного супутника. Остаточна похибка положення супутника - це вектор dr, зображений на **рисунку**, з типовими величинами в діапазоні 1-6 м [15]. Ефемеридні помилки, як правило, найменші в радіальному (від супутника до центру Землі) напрямку. Компоненти помилок ефемерида вздовж шляху (миттєвий напрямок руху супутника) і перехресного шляху (перпендикулярно

повздовжньому та радіальному напрямках) значно більші. Повздовжні та перехресні компоненти складні для спостереження контрольним сегментом за допомогою центру моніторингу на поверхні Землі, о**скільки ці компоненти не значно перетворюють лінію прямої видимості на Землю**. Але, користувач не відчуває великих помилок вимірювання через кількість помилок, пов'язаних з ефекмерами, з тієї ж причини. Ефективна похибка псевдодальності або фази несучої, обумовлена помилками прогнозування ефемеридів, становить порядку 0,8 м [16].

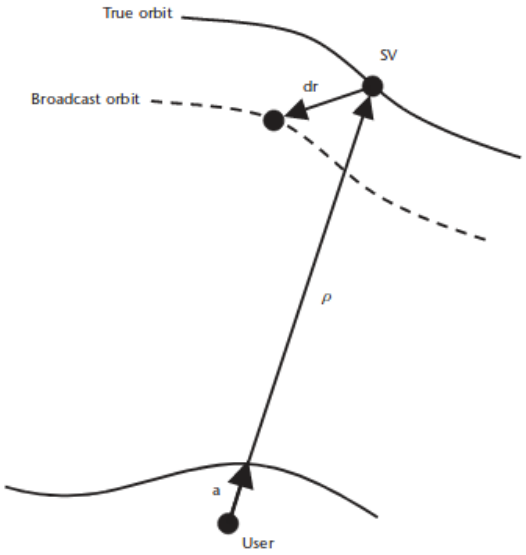


Рис. 1.11 Похибка положення супутника

1.6.3 Зниження точності

Зниження точності (DOP) також впливає на точність розрахунків GPS, але не прямим способом. Математично DOP - це співвідношення між стандартними відхиленнями заданого параметра та псевдодальністю. Наприклад, вертикальна DOP - це співвідношення між стандартним відхиленням вертикальної складової (висоти) приймача GPS та стандартним відхиленням псевдодальності. Для параметрів, що містять більше однієї змінної, такої як геометричний DOP, коефіцієнт визначається як квадратний корінь суми квадратів стандартного відхилення змінних (координат x, y, z та часу) та стандартним відхиленням псевдодальності. Фізично DOP описує вплив геометричної конфігурації видимих супутників на точність GPS. В ідеалі видимі супутники повинні розташовуватися на широких кутах відносно

один одного. Геометрія такої конфігурації супутників вважається сильною, а значення DOP низькі. І навпаки, якщо видимі супутники мають мале значення кута між собою, то конфігурація супутників має слабку геометрію та значення DOP є високими [17]. На **рисунку** показаний сценарій, в якому приймач GPS вимірює псевдодальності двох супутників. Похибка вимірювання псевдодальності в обох випадках залишається незмінною, рисунок , з більшим кутовим розмежуванням між двома супутниками, показує, що площа невизначеності в положенні приймача є меншою, ніж у другому випадку, рисунок .

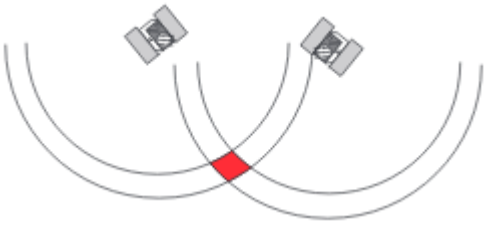


Рис. 1.12 Приклад низького значення DOP

Рис. 1.13 Приклад високого значення DOP

1.6.4 Багатопроменеве поширення

Багатопроменеве поширення виникає, коли антена приймача розташована поблизу великої відбивної поверхні, такої як озеро або будівля. Сигнал супутника поширюючись в сторону приймача потрапляє в найближчий об'єкт і відбивається в напрямку антени, створюючи похибки вимірювання.

Поки приймач відстежує прямий сигнал (який завжди надходить раніше, ніж будь-який сигнал зазнавший багатопроменевості), такі випадки мало впливають на продуктивність. Проте, сигнали відбиті від прилеглих об'єктів, або навіть відбиті від віддалених об'єктів, можуть надходити з короткими затримками затримки (наприклад, десятки або сотні наносекунд) після прибуття основного сигналу. Такі багатопроменеві копії сигналу спотворюють

кореляційну функцію між отриманим композитним (прямим) сигналом та локально згенерованою кодовою послідовністю в приймачі. Вони також спотворюють композитну фазу отриманого сигналу, вводячи помилки в вимірах псевдодальності та фази несучої, які відрізняються для сигналів різних супутників і, таким чином, створюють помилки в визначенні положення, швидкості та часу.

Затінення - це надмірне затухання при прямому поширенню сигналу, який зазвичай вводять коли прямий шлях проходить через листя або іншу перешкоду. В деяких випадках при багатопроменевому поширенні сигнал зазнає слабшошо затухання, отримана потужність сигналу зазнавшого багатопроменевості може бути навіть більшою, ніж прийнята потужність сигналу що зазнав затінення [18]. Таке явище може виникнути у ситуаціях на вулиці, як це зображено на рисунку 6.8, а також в приміщенні, наприклад, коли прямий шлях проходить через стіни або стелю та дах, тоді як багатопроменевий сигнал відбивається від іншої будівлі і надходить через вікно. У деяких випадках траєкторія прямого шляху може бути настільки щільно затіненою, що приймач може відстежувати тільки багатопроменеві сигнали.

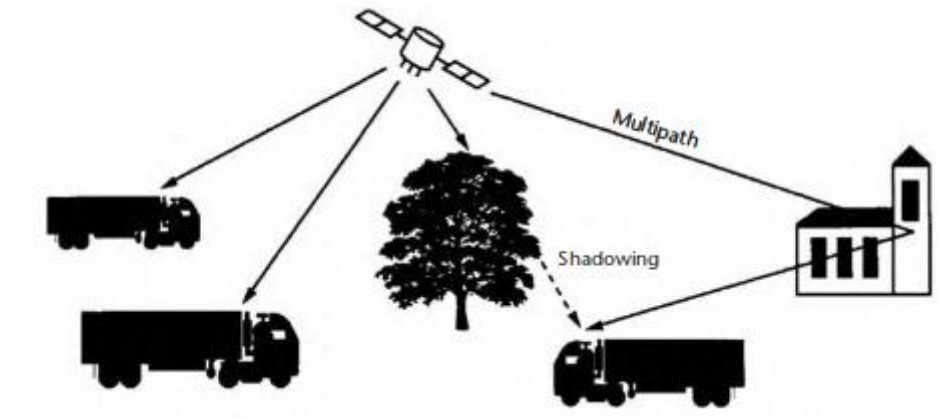


Рис. 1.14 Приклад багатопроменевого поширення та затінення

Помилка, введена через багатопроменеве поширення, залежить від їх затримок, а також від їхньої потужності та фази несучої відносно шляху прямого поширення. Сигнал із прийнятою потужністю, набагато меншим, ніж прямий шлях, створює мало спотворення отриманого сигналу і, як наслідок,

дає невелику помилку. Отримана фаза несучої багатопроменевості від прямого шляху також має прямий вплив на ступінь та характер спотворення.

1.6.5. Вплив іоносфери та тропосфери

Коли супутниковий сигнал проходить через атмосферу, його швидкість зменшується, оскільки повітря має дещо вищий індекс рефракції (близько 1.0003 [19]), що викликає зменшення швидкості сигналу в середньому близько на 0,03%. Однак неоднорідності в атмосфері, особливо в іоносфері, викликають нерівномірне сповільнення сигналу. Цей ефект найменший, коли супутник безпосередньо знаходиться над приймачем і стає більшим для супутників поблизу горизонту, оскільки шлях сигналу через атмосферу довший. Щоб пом'якшити цей ефект, після того, як відомо про приблизне положення приймача за допомогою псевдодальностей, можна використати вбудовану математичну модель для оцінки середньої кількості затримки для компенсації такого типу помилок. Однак модель може не спрогнозувати повні ефекти іоносферної затримки. Більш точний спосіб компенсувати цю помилку - це використовувати обидві частоти для вимірювання затримки. Іоносферна затримка впливає на швидкість мікрохвильових сигналів по-різному в залежності від їх частоти, це характеристика, відома як дисперсія. Затримки, виміряні на двох частотних смугах, можуть бути використані для вимірювання дисперсії, і це вимірювання потім може використовуватися для оцінки затримки на кожній частоті. Інший спосіб компенсувати іоносферну погрішність полягає в тому, щоб порівняти позицію, яка вимірюється GPS, з відомим заздалегідь положенням. В даному випадку використовується той факт, що ефекти іоносфери зазвичай змінюються повільно і можуть з часом усереднюватися; отже, корекція на іоносферну похибку може бути застосована до інших приймачів GPS у тій же загальній області. Супутникові наземні системи посилення (SBAS), такі як WAAS (доступні в Північній Америці та на Гаваях), EGNOS (Європа та Азія) або MSAS (Японія), передають дані про іоносферні корекції через супутник, а Системи посилення наземних баз (GBAS) передають дані про корекцію через наземний

радіопередавач безпосередньо до приймача GPS. Вологість у тропосфері також призводить до помилок, близьких до іоносферної затримки. Проте цей ефект більш локалізований, змінюється швидше, ніж іоносферні ефекти, і не залежить від частоти [17].Щільність іоносфери також залежить від впливу сонця. Вночі дуже мало іоносферного впливу. Вдень сонце збільшує влив на іоносферу і сигнал сповільнюється більше. Значення, на яке збільшується щільність іоносфери, коливається залежно від сонячних циклів (активність сонячних плям). Сонячна активність зростає приблизно кожні 11 років. На додаток до цього, сонячні спалахи також можуть впливати на іоносферу.

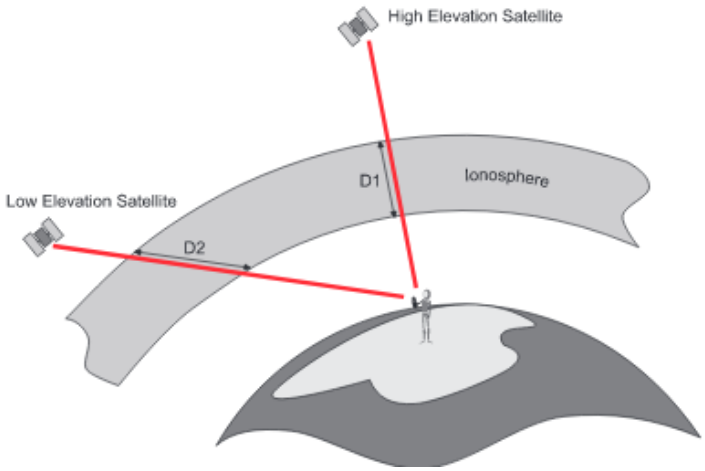


Рис. 1.15 Вплив положення супутника на шлях кріз атмосферу

Тропосфера є нижньою частиною атмосфери, яка недисперсна для частот до 15 ГГц. У цьому середовищі фазова та групова швидкості, пов'язані з несучою GPS та інформацією про сигнал (код PRN та навігаційні дані) на обох L1 і L2 однаково затримуються відносно швидкості світла в вакуумі. Ця затримка є функцією тропосферного показника заломлення, що залежить від локальної температури, тиску та відносної вологості повітря. Без компенсації ця затримка може варіюватися від приблизно 2,4 м для супутника в зеніті та користувачеві на рівні моря до приблизно 25 м для супутника під кутом нахилу приблизно 5 °.

Композиційні гази впливають на електромагнітний сигнал, що поширюється через нейтральну атмосферу. Їх комбінований показник заломлення, трохи більший за одиницю (номінально 1.0003 на рівні моря),

призводить до того, що швидкість сигналу буде меншою, ніж у вакуумі, і збільшує час отримання сигнал для досягнення антени приймача GPS, таким чином збільшуючи шлях електромагнітної хвилі. Рефракція змінює напрямок поширення хвилі і тим самим подовжує шляху хвилі, що ще більше збільшує затримку. Обидва ці ефекти часто називають затримкою. Оскільки основна частина затримки відбувається в тропосфері, вся затримка часто називається виключно тропосферною затримкою. Існує можливість адекватно компенсувати суху частину тропосферної затримки (точніше, частину, яка знаходиться в гідростатичній рівновазі, основна частина якої пояснюється сухими газами), якщо відомо поверхневий тиск з високою точністю, цю інформацію можна отримати з барометра. Нестабільний компонент — затримка через складову вологості, викликаної водяною парою в нижніх шарах тропосфери. Як і в іоносфері вміст електронів, просторовий і часовий розподіл водяної пари є значною мірою непередбачуваним і може зазнавати швидких змін, особливо за наявності погодного фронту. Характеристики поверхневої вологості повітря зазвичай не відображають вологісті тропосфери на достатньому рівні. Тому навіть з поверхневими метеорологічними даними важко, якщо не неможливо, належним чином моделювати або прогнозувати затримки викликані вологістю.

Висновок

1) У розділі описаний принцип функціонування системи GPS, описані три сегменти, а саме космічний, контролюючий та сегмент користувача.
2) Визначено поняття GPS часу, відмінностей його від часу UTC
3) Описані два основні принципи розрахунку псевдодальностей на основі C/A коду, а саме за методом загального часу прийому та методом загального часу передачі
4) Описані принципи визначення місцеположення при розрахованих значеннях псевдодальностей та відомих позиціях супутників в момент передачі.
5) Визначені основні джерела похибок GPS, а саме: вплив тропосфери та іоносфери, багатопроменеве поширення сигналів, похибки ефемерид та часу.

2. Методи диференційного позиціонування

Одночастотний приймач GPS часто може досягати точності в межах 5-9 метрів. Проте, існує безліч додатків, які вимагають більшої точності, цілісності, доступності та безперервності. Для таких додатків потрібне збільшення точності. Є два загальних класи підвищення точності: на основі опорного приймача та з використання зовнішніх датчиків/системи.

Існують джерела похибок, пов'язані з супутниками, джерела, пов'язані з сигналом, що рухається від супутника до антени приймача, та джерела, пов'язані безпосередньо з приймачем. Під час визначення псевдодальностей приймач визначає різницю в часі між тим, коли сигнал залишив супутник і коли він був отриманий. Це включає в себе фактичний час поширення сигналу, а також ефекти помилки, такі як ті, що виникають через іоносферу, тропосферу, зсув часу, завади та багатопроменеве поширення. Процес визначення позиції користувача полягає в тому, що приймач використовує передані ефемериди для визначення місцезнаходження супутника. Це значення також зазнає помилки; таким чином визначаючи одне додаткове джерело помилки. Тому, навіть якщо всі помилки при вимірюванні псевдодальності можуть бути виправлені певним чином, в результуючому положенні все одно буде елемент помилки. У програмах, що збирають дані від еталонного приймача та від інших одержувачів, які проводять вимірювання і знаходяться в точках, що є цікавими, об'єднуються для пост-обробки, а використання математичних методів дозволяє компенсувати загальні помилки. Результатом є положення, визначене в точності до 1 метра. Але це тільки після збору великої кількості спостережень і виконання складних розрахунків. Існує альтернатива, яка працює в режимі реального часу, але з менш вражаючими результатами.

2.1 Диференційна GPS поправка

DGPS - це метод для покращення позиціонування або синхронізації GPS за допомогою однієї або декількох еталонних станцій, що знаходяться у завідомо відомих місцях, кожен з яких обладнано принаймні одним

приймачем GPS. Цей приймач розміщується в місці, яке добре відоме. Це може стати еталоном, який буде використовуватися як опорна точка для розрахунків. Приймач або визначає своє положення самостійно, або отримує дані про місцезнаходження через інтерфейс вводу, оператором, і зберігає цю позицію, вважаючи, що його позиція є істинною. Далі, еталонний приймач робить звичайні спостереження та обчислює псевдодальності, як це робить будь-який інший приймач. Він також декодує навігаційне повідомлення для визначення ефемерид і поєднує в собі поточний час та передані значення ефемерид для обчислення місцезнаходження супутника. Використовуючи обчислену позицію супутника та позицію приймача, обчислюється математична або модельна дальність до супутника. Якщо не було помилок при передачі ефемерид чи розташування супутника, ця модельна дальність буде дорівнювати очікуваній псевдодальності. Якщо порівнювати спостережувані псевдодальності і модельні дальності, майже завжди буде відстежуватись, що вони відрізняються через помилки з усіх згаданих раніше джерел. Опорний приймач обчислює цю різницю для кожного супутника та повідомляє всі значення у вигляді виправленнь виправлення через певний каналу передачі даних (радіопередача, кабельний зв’язок) будь-якому іншому приймачу, який бажає їх використовувати. Інші приймачі, які називаються користувальницькими приймачами або просто користувачами, беруть ці значення та застосовують їх до власних спостережень псевдодальностей перед обчисленням власної позиції. Принцип роботи DGPS зображений на рисунку [13].

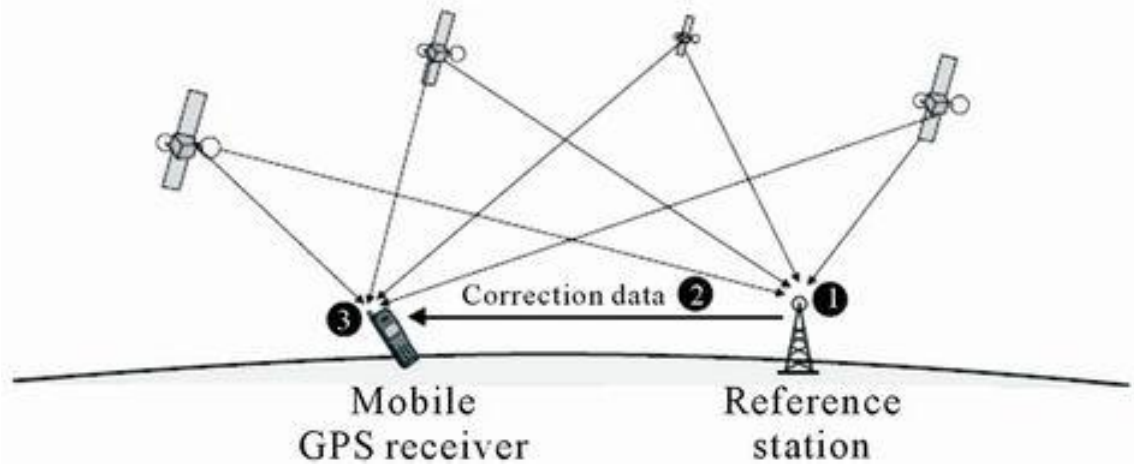


Рис. 2.1 Архітектура DGPS

Псевдодальності, що спостерігаються користувачами, містять ті самі типи помилок, що спостерігаються еталонним приймачем. Помилки, які введені супутником, однакові для обох приймачів. Ті, що виникають вздовж шляху поширення сигналу, включаючи іоносферні та тропосферні ефекти, зазвичай схожі для обох приймачів, але не точно такі ж. Помилки, викликані приймачем, звичайно, унікальні для кожного приймача.

Помилки супутників, які, як правило, складаються в основному з помилки годинника, є загальними для будь-якого спостерігача. Корекція, про яку повідомляє еталонний приймач, також містить ефекти цієї помилки. Таким чином, коли користувач застосовує виправлення, ефекти від помилок супутника видаляються з рішення. Помилки в псевдодальностях, викликані проходженням сигналу через іоносферу, знаходяться в порядку від декількох метрів до десятків метрів. Однак, оскільки супутник знаходиться на відстані приблизно 20 000 км над Землею, а іоносфера також досить висока над Землею, шлях сигналу через іоносферу, як правило, подібний для приймачів, що знаходяться на відстані кількох сотень кілометрів. Як наслідок, незважаючи на те, що похибки пов'язані з шляхом сигналу через іоносферу, дещо відрізняються для різних приймачів, коригування в більшій мірі компенсує ці помилки. Ефекти тропосфери не є спільними для двох приймачів. По-перше, тропосфера значно нижче. Таким чином, шлях сигналу через тропосферу значно відрізняється для приймачів з більшою відстанню

між собою. По-друге, вплив тропосфери може відрізнятися на досить коротких ділянках. Серед властивостей тропосфери, які впливають на сигнали, виділяють вологість. Наявність великої хмари на шляху сигналу до одного приймача та відсутність цієї хмари на іншому може призвести до значної різниці в ефектах. Загальний ефект тропосферних явищ, як правило, становить близько метрів на кілька метрів. Таким чином, виправлення, надіслані з корегуючої станції на користувацький приймач, що зазнали впливу тропосфери, скоригують цю похибку лише частково, при цьому зменшується точність визначення місцеположення, оскільки за рахунок корекції змінюється значення псевдодальності. Звичайно, ефекти від коригування можуть бути шкідливими для приймачів, що знаходяться на відстані кількох десятків кілометрів. Інша значна помилка повязана з сигналом — багатопроменеве поширенн є цілком унікальною для кожного приймача. Оскільки ця помилка складається з декількох копій одного і того ж сигналу, що з'являються в одній точці простору, але з різним часом прийому, результатом стане інтерференція. Добре відомо, що такі ефекти, як правило, сильно відрізняються на відстані менше довжини однієї хвили, в даному випадку менше 20 см для несучої GPS L1 [13].

Очевидно, що помилки багатопроменевого поширення не можуть бути виправлені надійно за допомогою зовнішніх корекцій. Виключення впливу ефекту багатопроменевості на еталонний приймач вирішується шляхом його розміщення на місці з хорошою видимістю супутника, щоб забезпечити сильний прямий сигнал, і той, який захищений від очевидних джерел багатопроменевості, таких як стіни або дах, де інші об’єкти, що викликають відбиття сигналів розташовані нижче антени. Деякі конструкції антени допомагають суттєво зменшити вплив джерел багатопроменевості поблизу горизонту та нижче, тим самим зменшуючи ефекти помилок багатопроменевого поширення сигналу на розраховані на еталонному приймачу коригування.

Оскільки виправлення є більш ефективним ближче до корегуючої станції, еталонні приймачі часто розташовуються поблизу в'їздів в гавані або у великих аеропортах, так що транспортні засоби, що рухаються в таких місцях, мають найменшу помилку у найважливішій частині своєї навігації. Похибки, що виникають в межах двох приймачів, які використовуються для диференціальної GPS корекції, включають помилки, пов'язані з годинниками приймачів, шумом прийнятих сигналів. Поправки надіслані з коригувальної станції містять ці помилки. Коли користувач застосовує виправлення, спостерігатиметься ефект передачі помилок коригуючої станції до користувача. Проте деякі з цих помилок, як правило, є загальними для всіх сигналів, що спостерігаються еталонним приймачем (наприклад, похибка зсуву годинника), і таким чином буде включена у всі виправлення. Тим самим, користувацький приймач буде виявляти таку саму помилку, яка виникає при використанні довшого або коротшого антенольного кабелю - тобто вони будуть зосереджені на зміщенні часової ділянки, а не на впливі на x, y та z координати. Саме з цієї причини, коли приймач використовує диференціальні поправки, він не повинен використовувати суміш спостережень з корекціями та спостереженнями без корекції, адже використання такої суміші призводить до того, що звичайні помилки еталонного приймача будуть розміщуватисяне лише в часовій площині, що призведе до більшої похибки в визначенні позиції.

Корисний побічний ефект цього процесу пов'язаний з помилками в позиції коригуючого приймача. Такі помилки розповсюджуються безпосередньо, оскільки вони не є загальними для всіх супутників, а скоріше впливають на кожен супутник залежно його місцезнаходження. Наприклад, якщо позиція еталонного приймача вважається північніше, ніж це є насправді, вплив на супутники північніше від приймача буде в зворотному напрямку від супутників на південь, а супутники безпосередньо на схід або захід від місця розташування не матимуть значних виправлень через таку похобку. *В*наслідок цього користувач визначить свою позицію так, що вона буде відрізнятися від

дійсності точно на стільки ж і в тому ж напрямку, що і помилка еталонного приймача. У випадку, якщо визначення місцеположення проводиться відносно точки, позиція якої відома, але зміщена по відношенню до системи GPS і позиція цієї точки була введена для еталонного приймача, то помилка, що міститься в корекціях позиції буде поширюватися на всіх користувачів. В результаті позиції всіх користувачів виявляться вірними відносно цієї точки, без необхідності перетворення. Таким чином літальні апарати в аеропорту чи кораблі, що прямують до гавані, розраховують своє положення відносно місцевих карт, незалежно від наявності будь-яких помилок на них.

Будь-яка помилка, пов'язана з переданими ефемеридами, буде розглядатися на еталонному приймачі лише для модельних дальностей, оскільки самі спостереження виконуються незалежно від переданих позицій супутників. Однак, їх ефект буде врахований в корекціях і буде враховуватися в користувачем, під час обчислення позиції, оскільки користувач буде в той час використовувати ту саму неправильну позицію супутника для коригування псевдодальностей і визначення позиції. Таким чином, всі помилки, що виникають на супутнику, будуть видалені в процесі DGPS.

На практиці існують помилки, які диференціальні поправки не можуть виправити, і навіть помилки, які вносяться самим процесом (наприклад, помилки спостережень коригуючої станції, які не є спільними для всіх супутників). Але загальний ефект досить значний. Просто використовуючи псевдодальності, отримані з коду С/А, звичайний приймач, як правило, обчислює свою позицію як точку, яка з часом рухається більш ніж на 100 метрів. Коли застосовуються коригування від коригуючої станції, цей діапазон руху, як правило, буде набагато менше, ніж 10 метрів і навіть менш ніж 2 метри, коли приймач знаходиться відносно близько до коригуючої станції. Цей процес в режимі реального часу робить визначення позиції цілком точними в більшості режимів навігації.

2.2 Кінематика в режимі реального часу

Кінематика в режимі реального часу (RTK) забезпечує інформацію про позиціювання в режимі реального часу з корекцією, що надає точність вимірювання в сантиметровому діапазоні. RTK отримує супутниковий сигнал на двох приймачах. Один з двох приймачів використовується як базова станція з відомими координатами, а інша - мобільна станція. Базова станція записує розташування своєї позиції та надсилає сигнал корекції на мобільну станцію, щоб виправити сигнал від супутника, отриманий мобільною станцією. Ця система передбачає канал зв'язку для комунікації між двома наземними приймачами. Сила сигналу зменшується, при збільшенні відстані між приймачами, а також, якщо приймачі надто далекі друг від друга, переваги методу нівелюються. Таким чином, система RTK залежна від перешкод, також можливі значні збої при роботі в міських районах, де будівлі можуть заблокувати сигнал, що перешкоджає зв'язку приймачів. Базова GPS-станція приймає вимірювання з супутників, що знаходяться в полі зору, а потім передає їх разом із своєю позицією мобільній станції. Приймач мобільної станції також збирає вимірювання з супутників обробляє їх разом з даними, отриманими з базової станції. За рахунок цієї інформації мобільна станція розраховує своє положення відносно базової станції. RTK визначає місцезнаходження відносно коригуючої станції шляхом вимірювання фази несучої хвилі, а не коду PRN, як в DGPS. Несуча сигналу має набагато коротшу довжину хвилі, ніж ширина кодованого циклу PRN - від сотні до тисячі разів менше, тому здатність вимірювати відстань збільшується пропорційно [20].

Використання фазових вимірювань несучої має як переваги, так і недоліки. Основною первагою є те, що помилки приймача зведені до мінімуму. З DGPS коригуються помилки супутника та деякі атмосферні похибки шляхом вимірювання коду PRN в контрольній точці та відправки результатів до приймача. Приймач вираховує помилки з власних вимірювань псевдодальностей, щоб отримати кращий результат. Це можна розглядати як "диференціювання". Цей процес залишає помилки приймача, що є

обмеженням GPS точності. Використовуючи фазу несучої та набагато більш детальні математичні методи, існує можливість також відрізнити помилки приймача.

Наступною перевагою є те, що атмосферні помилки зведені до мінімуму. Іоносферні похибки не дуже відрізняються залежно від місця, але їх дуже важко моделювати (передбачити) для одночастотних приймачів. Проте іоносферна затримка, що впливає на сигнал, має дуже передбачувану функцію частоти. Вимірюючи затримки на обох частотах несучої частоти L1 та L2, іоносферна похибка може визначена з високою точністю та значно зменшена. З цієї причини обладнання сантиметрової точності зазвичай вимірює як несучу L1 і L2, так і здатні працювати на великих відстанях від базової станції.

Недоліком цього методу є необхідність вирішення фазової неоднозначності методом спроб та помилок. Хоча фаза несучої дає нам можливість визначити координати з набагато більшою точністю, існує проблема визначення відстані на основі фазових вимірювань. На відміну від коду PRN, за фазою несучої неможливо визначити час, адже це лише послідовність електромагнітних хвиль. Можливо дуже точно виміряти фазу окремого періоду несучої, проте немає інформації про кількість періодів коливання між приймачем та супутником. Проте в даному випадку відстань до супутника не є необхідною; важливу роль відігає відстань від еталонного ресивера до нашого ровера. Ця відстань набагато коротша і дозволяє нам використовувати деякі математичні здогадки. Методом підбору, починаючи з одного рішення, переходячи іншого, доки не буде визначено те значення, що найкраще підходить (відповідає) всім вимірюванням. Цей процес відомий як "ініціалізація RTK", а іноді - як "пошук цілих чисел" [21].

2.2.1 Фазові вимірювання

Несуча GPS являє собою хвилю з правою круговою поляризацією. Фазові вимірювання відбуваються наступним чином:

1) Спочатку приймач вимірює часткову фазу несучої GPS.

2) Відносно отриманого значення приймач вимірює цілі довжини хвилі несучої.

3) Приймач постійно підраховує цілі довжини хвиль. Приймач, в силу неоднозначності, не знає точну кількість цілих довжин хвиль між супутником та своїм місцезнаходженням.

Різниця фаз несучих вимірюється наступним чином. Всі приймачі (опорний приймач та приймач користувача) вимірюють один і той самий сигнал із супутника. Сигнал, що надійшов до опорного приймача використовується як еталон, та в такому випадку можливо розрахувати різницю фаз між двома сигналами. Фазові відмінності розраховуються для сигналів усіх супутників в зоні видимості. Схематичне зображення фазових вимірювання зображене на рисунку [21].

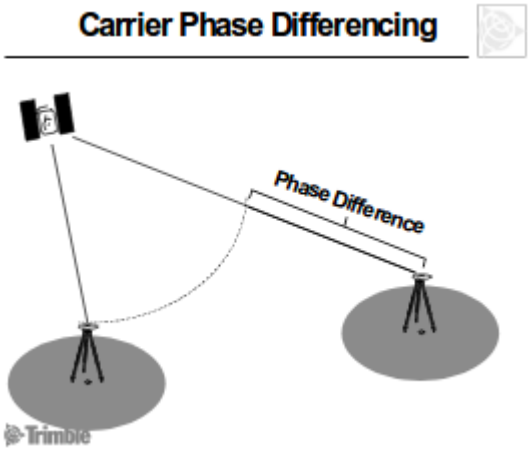


Рис. 2.2 Фазові вимірювання

2.2.2. Визначення позиції

Один автономний приймач не може достовірно визначити кількість повних коливань до супутника. Для вирішення неоднозначностей необхідні два приймачі, що спостерігають однакові сигнали. Розрізнення фаз несучих - це метод, який використовується для вирішення цих неоднозначностей. Коли сигнал надходить із супутника, він поширюється до всіх приймачів. Коли сигнал на еталонному приймач порівнюється з сигналом, отриманим на іншому приймачі, визначається різниця між фазами сигналу на двох приймачах. Після чого починається ініціалізація RTK, тобто процес підбору

цілих чисел.

Спочатку відбуваються початкові оцінки цілих чисел - їх обчислюють за допомогою однозначних псевдодальностей, отриманих на основі коду C/A.

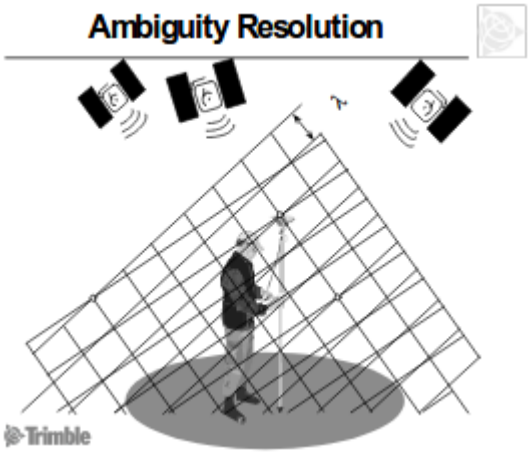


Рис. 2.3 Множина точок для рішення неоднозначності

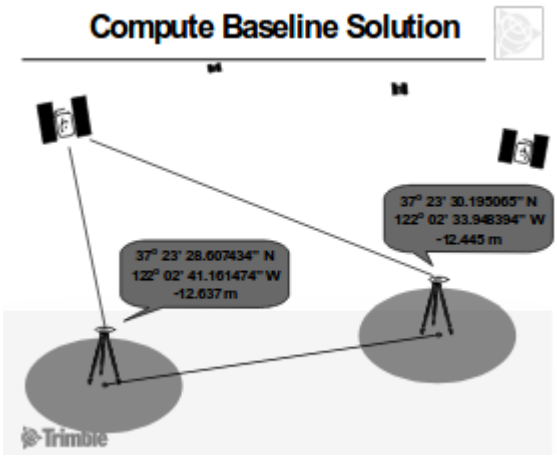
Очікувана помилка цих оцінок визначає обсяг пошуку навколо невідомої позиції користувача. Цей обсяг розглядається таким чином, щоб вмістити тривимірну матрицю точок, кожна з яких пов'язаний з комбінацією цілих чисел.

Далі відбувається пошук найкращої комбінації. У найпростішій формі цілий пошук містить ряд розрахунків позиції приймача, використовуючи кожну комбінацію цілих чисел. Правильна комбінація визначиться на основі статистичного тесту, що визначає якість рішення. Будь-який такий тест можливий лише у тому випадку, якщо доступно принаймні 5 супутників в зоні видимості [21]. Перш ніж вирішиться неоднозначність, знаходиться дробове рішення. Це означає, що значення невизначеності мають лише дробову частину. Пошук цілих чисел вирішує фазову неоднозначність, і система RTK вважається ініціалізованою. Позиція покращується до сантиметрового рівня. Після того, як невизначеність вирішена, обчислюється базова лінія для визначення позиції мобільної станції по відношенню до базової станції. Процес обчислення базової лінії починається з розрахунку фазової різниці. Вирішується та співставляється вирішена фазова неоднозначність до фазових відмінностей. Визначаються точні дальності для всіх супутників в зоні видимості, щоб визначити позицію мобільнї станції. Передаються фіксована

(відома) позиція базової станції та її спостереження. Обчислюється базова лінія та перевіряється оберненість між положеннями опорної та мобільної станцій. Рисунок відображає процес визначення базової лінії.

Рис. 2.4 Визначення базової лінії

2.2.3 Опорні джерела RTK



Джерелом інформації в RTK виступає еталонний приймач. Цим вноситься істотне обмеження у радіусі використання RTK. Чим далі користувач знаходиться від еталонного приймача, тим більшим буде шлях сигналу від супутника до приймачів проходити через різні частини атмосфери. Це збільшує вплив атмосфери (перш за все іоносфери), збільшуючи похибку в отриманій позиції. Помилка, пов'язана з відстанню, є більш значною проблемою для обробки фази носія, ніж для звичайної обробки коду PRN. Через те, що точність фазових вимірюваннь набагато вище, тому вплив атмосферних помилок буде в значно швидше виявлений, а також через те, що підвищується складність рішення неоднозначності, що призводить до додаткових помилок.

За сприятливих умов і низького впливу іоносфери приймач може знаходитись на відстані до 20 км від опорної станції. В умовах тропіків достовірність результатів гарантована при відстані не більше 5 кіломентрів [22].

Висновок

Таблиця 2.1 Порівняння DGPS та RTK

Відмінності DGPS та RTK			
Характеристика	Автономний приймач	DGPS	RTK
Кількість супутників необхідна для роботи	4		4 після ініціалізації
Кількість супутників необхідна для ініціалізації	Ініціалізація відсутня		Мінімум 5
Час необхідний для точного позиціонування	Миттєво		~ 1 хв
Приймач	Достатньо одночастотного		Двочастотний
Вимірювання	Кодові вимірювання		Фазові вимірювання
Точність	5-9 метрів	~ 1 метр	~ 1 см в горизонтальній площині

1) Точність автономного GPS приймача знаходиться в межах 5-9 метрів, чого не є достатньо для більшості випадків. На для компенсації похибок використовуються методи DGPS та RTK. Обидва методи використовують опорну станцію, з завідомо відомими координатами.

2) Обидва методи передбачають наявність GPS приймача, який буде розташований на відкритій місцевості, з визначеними наперед координатами, а також наявність каналу зв’язку з користувачами.

3) Метод DGPS дозволяє компенсувати більшу частину похибок, які впливають на визначення місцезнаходження, а саме вплив тропосфери та іоносфери, похибки пов’язані з ефемеридами та похибки супутникового годинника.

4) Метод RTK компенсує похибки майже повністю, за рахунок чого досягається точність до сантиметрів, проте вимагає дорожчого обладнання, а

також накладає обмеження на приймачі користувачів. Фазові вимірювання вимагають безперервного відстеження сигналу, що вимагаю постійної роботи GPS чіпу у смартфоні, а також складніших розрахунків. Таким чином значно зростатиме використання акумуляторної батареї смартфона.

5) Використання методу RTK в мобільних додатках на платформі Android вимагатиме підвищеного використання ресурсів, що не є сприйнятливим для сучасних смартфонів.

3. Використання необроблених вимірювань в Android

3.1 Доступ до необроблених вимірювання з використанням програмного інтерфейсу Android

Найпростішим способом визначити позицію в операційній системі Android — це використати узагальнений провайдер місцеположення, який поєднує в собі кілька джерел (GNSS, Wi-Fi або навіть мобільні мережі), щоб підвищити точність, час першого розрахунку, доступність або споживання енергії. Це доступно через API android.gsm.location. Починаючи з версії Android 7 (Nougat) разом з наступними версіями (Android O), Google представляє новий API для позиціонування Android (android.location) і надає значні нововведення для служби локації.

Взаємодія між додатками Android і різними мобільними датчиками, такими як GNSS, здійснюється за допомогою API інтерфейсу Android. Кожна нова версія платформи Android пов'язана з новою версією API, а конфігурація, взаємодія та доступ користувачів з набором мікросхем GNSS залежать від рівня API. У таблиці показано взаємозв'язок між версіями Android і рівнями API.

Таблиця 3.1 Взаємоз’язок версій Android з версіями API

Версія	**Назва**	**API**
2.3	*Gingerbread*	10
4.0	*Ice Cream Sandwich*	15
4.1	*Jelly Bean*	16
4.2	*Jelly Bean*	17
4.3	*Jelly Bean*	18
4.4	*KitKat*	19
5.0	*Lollipop*	21
5.1	*Lollipop*	22
6.0	*Marshmallow*	23

Версія	Назва	API
7.0	*Nougat*	24
7.1	*Nougat*	25
8.0	*Oreo*	26
8.1	*Oreo*	27
9.0	*Pie*	28

На малюнку 9 наведені структури Location API попередньої Android-7 (рівень API-24), в тому числі драйверів чіпсета GNSS і передачі даних між набором мікросхем і ОС.

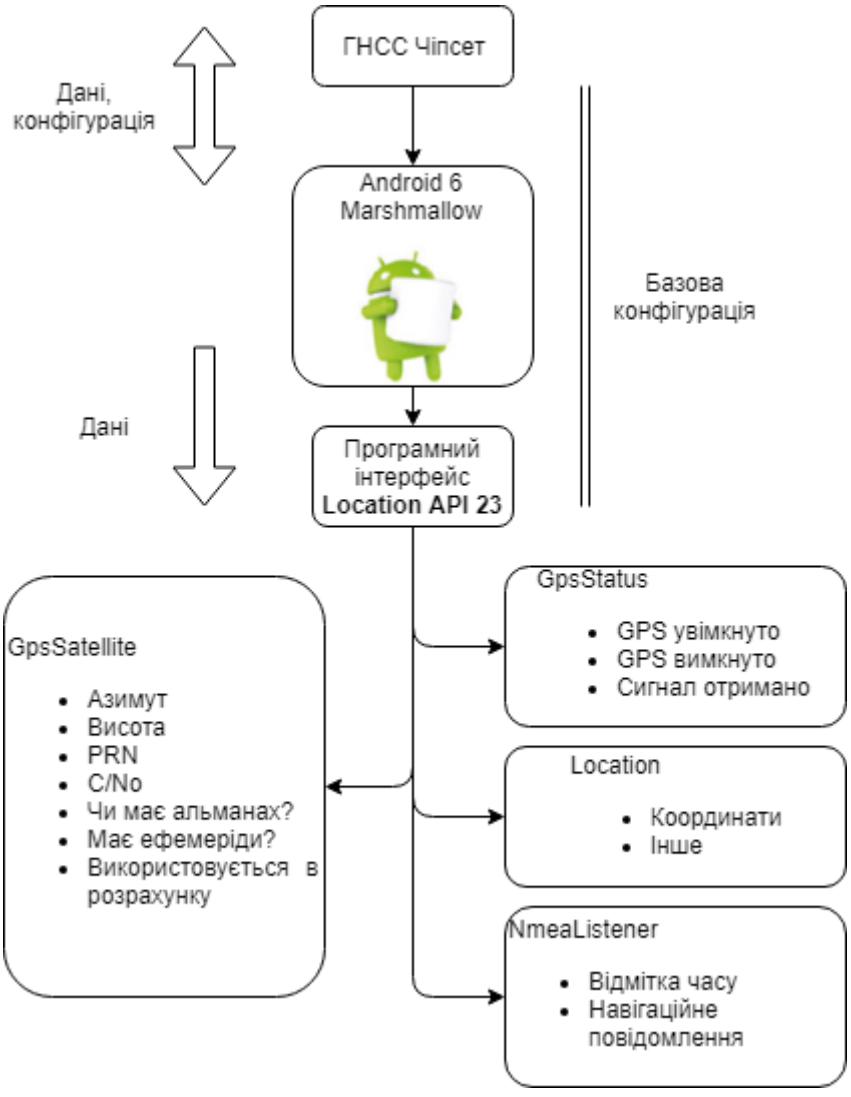


Рис. 3.1 Програмний інтерфейс Location в Android API 23

До 23-ої версії API, це було обмежено даними, наведеними знизу на рисунку , а саме: інформація про супутник (C/No, азимут, висота, якщо конкретний супутник був використаний у PVT), речення NMEA та рішення PVT з правильним значенням часу. Користувачі можуть надсилати базові команди конфігурації до чіпсета, включаючи перезапуск або запуск чіпсета GNSS чи очищення допоміжних даних. Проте всі налаштування конфігурації, які включають пріоритети вибору системи GNSS та різні алгоритми позиціонування, керуються самим набором мікросхем.

Починаючи з API 23 (Android 6), розробники отримали доступ до таких класів Android:

• GPS Satellite, що містить таку базову супутникову інформацію як азимут, висота, PRN та C/No. Він також позначає відмітки, якщо супутник використовується у вирішенні PVT та наявності альманаху та ефемеридів.

• GPS Status надає інформацію про стан та рішення набору мікросхем GNSS.

• Location, що вказує, чи передбачено позиційне та часове рішення.

• NMEA Listener, що забезпечує базові речення NMEA

Будь-який новий рівень API сумісний з попередніми версіями, тобто всі можливості з попереднього рівня API залишаються дійсними у новій версії. На малюнку 10 показано рівень розташування API-інтерфейсу 24, який реалізовано в Android 7. Починаючи з 24-ої версії API (Android 7) розробники отримали доступ до такої необробленої та обчисленої інформації GNSS через класи Android [27]:

GNSS clock, який містить:

- Час приймача (використовується для обчислення псевдодальності);
- Відхилення годинника

GNSS Navigation Message, який містить:

- Біти навігаційного повідомлення (всі сузір'я),
- Статус навігаційного повідомлення.

GNSS Measurement, яке містить:

- отриманий супутниковий час (використовується для обчислення псевдодальності)

- Кодові вимірювання

- Фазові вимірювання

Хоча ці дані надходять безпосередньо з набору мікросхем GNSS, прямий доступ до чіпсета не надається.

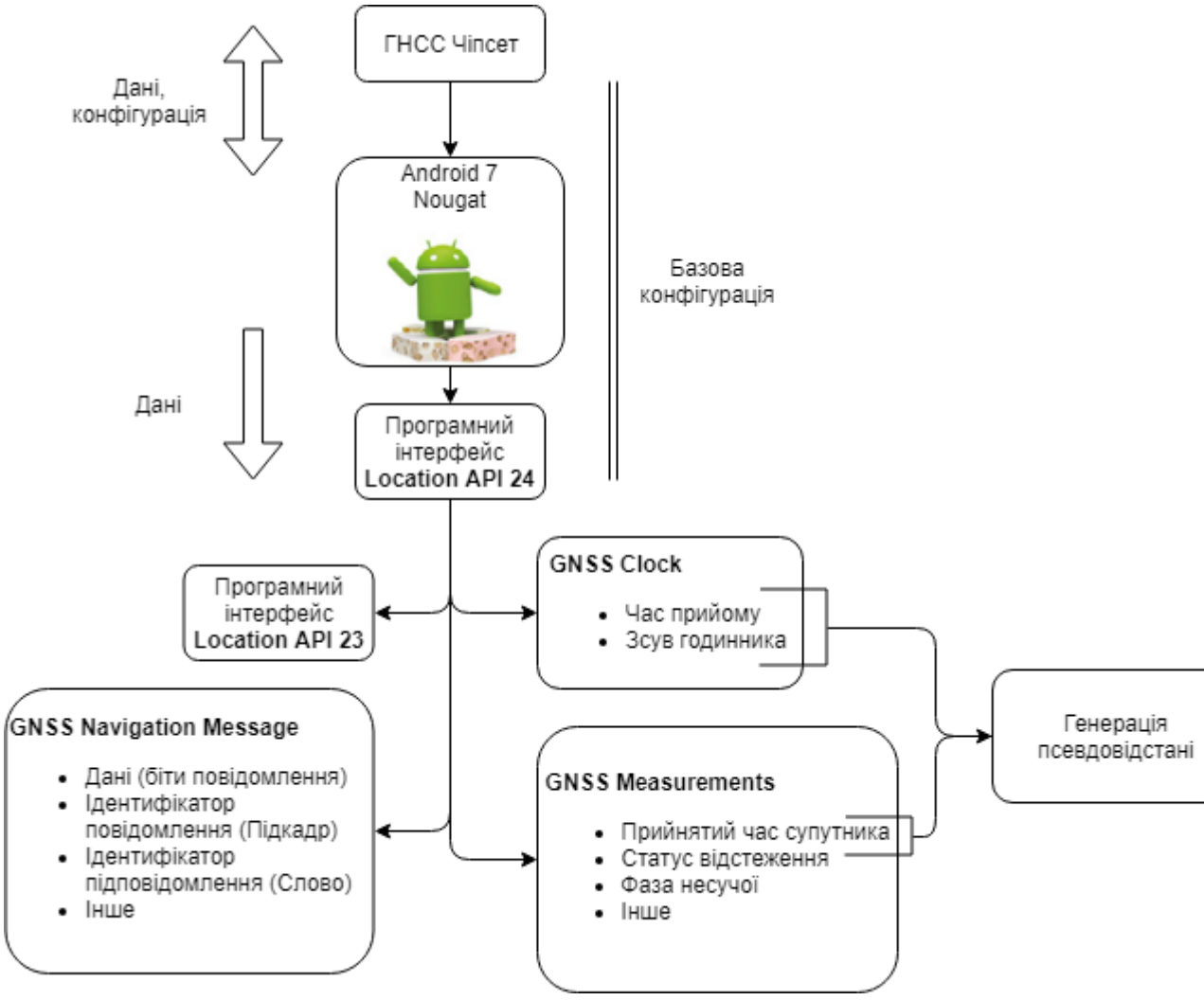


Рис. 3.2 Програмний інтерфейс Location в Android API 24 та вище

Так як API Android не надає значення псевдодальностей напряму, необхідно розраховувати це значення самостійно. Список полів класів нового API Location в Android 7 (табл.) не включає в себе час GPS або значення псевдодальності.

Таблиця 3.2 Поля та класи Location API у Android 7

Android class	Поле	Опис
GNSSClock	TimeNanos	Значення часу приймача в наносекундах
	BiasNanos	Відхилення часу
	FullBiasNanos	Різниця між TimeNanos приймача GPS та справжнім часом GPS з 00:00, 6 січня 1980 року
	DriftNanosPerSecond	Дрейф годинника
	HardwareClockDiscontinuityCount	Кількість апаратних розривів часу
	LeapSecond	Додаткова секунда
GNSSMeasurement	ConstellationType	Тип сузір’я
	ReceivedSvTimeNanos	Отриманий час GNSS-супутника в момент вимірювання
	AccumulatedDeltaRangeMeters	Накопичений дельта-діапазон з моменту останнього скидання каналу
	Svid	Унікальний номер супутника
	Cn0DbHz	Співвідношення сигнал/шум
	TimeOffsetNanos	Зсув часу, при якому зроблено вимірювання в наносекунд
	CarrierCycles	Кількість повних циклів несучої між супутником і приймачем

	CarrierFrequencyHz	Частота несучої, з якою були модульовані код та повідомлення
	PseudorangeRatemetersperSecond	Псевдошвидкість в поточний момент часу

Інформація щодо навігаційних повідомлень Android сумісна з GPS C / A, BeiDou D1 і D2, GLONASS L1 C / A. Однак точна інформація, що надається, залежить як від чіпсета GNSS, так і від виробника смартфонів

Таблиця 3.3 Дані навігаційного повідомлення

Параметр	Опис
Data	Отримує дані переданого GPS повідомлення. Для GPS L1 C / A кожний підкадр містить 10 30-бітних слів. Кожне слово (30 біт) знаходиться в останніх 30 бітах 4-ох байтового слова (байти 31 і 32 пропускаються). Загалом використовується 40 байт, що передаються протягом 6, 6 та 0.6 с відповідно.
Message ID	Для підкадрів 4 та 5 GPS L1 C/A це значення відповідає ідентифікатору кадру навігаційного повідомлення в діапазоні 1-25 (підкадри 1, 2 і 3 не містять ідентифікатора кадру, і це значення рівне -1).
Status	Повертає статус навігаційного повідомлення
Submessage Id	Для GPS L1 C / A, BeiDou D1 та BeiDou D2 це відповідає номеру підкадру навигаційного повідомлення в діапазоні 1-5.
Svid	ID супутника
Type	Отримує тип навігаційного повідомлення

3.1.1 Генерація GPS часу

Android 7 не надає значення GNSS часу, але надає час внутрішнього апаратного годинника та зміщення відносно часу GPS (у наносекундах), якщо приймач оцінює часовий інтервал GNSS. Коли приймач оцінює час GPS, його можна розрахувати як:

$$GpsTime = TimeNanos - (FullBiasNanos + BiasNanos)[\text{нс}], \quad (3.1)$$

де FullBiasNanos - зсув між годинником приймача та часом GPS у наносекундах, а BiasNano має порядок менший наносекунди.

Якщо приймач оцінює час за допомогою сузір'я, що не є GPS, приблизний час GPS може бути розрахований як:

$$GpsTime = TimeNanos - (FullBiasNanos + BiasNanos) - InterSystemBias, \quad (3.2)$$

де InterSystemBias - це зсув часу між GPS та GNSS-супутником, що використовується в оцінці часу.

3.1.2 Визначення псевдодальності

Android надає всі параметри, необхідні для обчислення псевдодальностей. Визначення псевдодальності базується на різниці часу, тобто різниці між часом, коли були отримано повідомлення (час вимірювання) і часом, коли було відправлено повідомлення:

$$\rho = \frac{(t_{Rx} - t_{Tx})}{1E9} \cdot c, \quad (3.3)$$

де t_{Tx} — отриманий час GPS-супутника в момент вимірювання, тобто час відліку GPS, коли сигнал був переданий, t_{Rx} - час вимірювання, с - швидкість світла у вакуумі.

t_{Tx} надається системою Android 7 як:

$$t_{Tx} = ReceivedSvTimeNanos[\text{нс}], \quad (3.4)$$

де ReceivedSvTimeNanos є прийнятий час GNSS супутника під час вимірювання, в наносекундах.

Дійсний діапазон t_{Tx} залежить від стану відстеження, а якщо статус відстеження не дорівнює “TOW-decoded” (GPS), то t_{Tx} стає неоднозначним, тобто псевдодальність стає неоднозначною. Щоб отримати точні значення псевдодальностей, потрібно отримати статус відстеження: “TOW decoded”

Час вимірювання у повному часі GNSS може бути відтворений таким чином:

$$t_{Rx_{GNSS}} = TimeNanos + TimeOffsetNanos - \big(FullBias(1) + BiasNanos(1)\big)[\text{нс}], \quad (3.5)$$

де TimeOffsetNanos це відхилення часу в момент вимірювання в наносекундах.

Для обчислення всіх отриманих часових величин використовуються тільки перші значення FullBiasNanos і BiasNanos. Ця операція виконується до тих пір, поки не відбудеться розрив у внутрішньому часі прийому. Це зазвичай трапляється лише при перезапуску модуля GNSS.

$t_{Rx_{GNSS}}$ надається лише в відповідній GNSS системі, і використовується для визначення часу приймача. Однак, t_{Tx} надається для кожної системи GNSS, наприклад GPST для вимірювань GPS або GLONASST для вимірювань ГЛОНАСС. Так t_{Tx} має бути приведено в ту ж систему відліку часу, що і $t_{Rx_{GNSS}}$, і навпаки. Як правило, приймачі GNSS використовують GPS як еталонний час для GNSS систем.

При обчисленні псевдодальності, коли $t_{Rx_{GNSS}}$ виражена в одиницях повного часу GNSS, з діапазоном t_{Tx}, взятим в залежності від стану стеження, обидва компоненти повинні знаходитися в одному діапазоні значень.
Наступні пункти демонструють, використання двох підходів обчислення псевдодальностей для GPS на основі вимірювань $t_{Rx_{GNSS}}$ та t_{Tx}, що вимагають двох перетворень:

• обох вимірюваннь до однієї системи часу (GPS)

• обох вимірюваннь до одного дійсного діапазону.

Для першого випадку час [24], вимірювання можна обчислити для GPS та Galileo з декодуванням TOW, як:

$$t_{Rx} = t_{Rx_{GNSS}} - weekNumberNanos[\text{нс}], \quad (3.6)$$

де weekNumberNanos - це число наносекунд, яке пройшло від початку часу GPS до поточного номеру тиждня.

Його можна розрахувати як:

$$weekNumberNanos = floor\left(\frac{-FullBiasNanos}{NumberNanoSecondsWeek}\right) \cdot NumberNanoSecondsWeek[\text{нс}],$$ (3.7)

де NumberNanoSecondsWeek - кількість наносекунд протягом одного тижня, тобто NumberNanoSecondsWeek = 604800e9.

В другому випадку, час вимірювання для GPS та Galileo з декодування TOW можна отримати як остачу від ділення (оператор mod) наступних величин:

$$t_{Rx} = mod\left(t_{Rx_{GNSS}}, NumberNanoSecondsWeek\right)[\text{нс}],$$ (3.8)

Після визначення t_{Rx} та t_{Tx}, розрахунок псевдодальності зводиться до

$$\rho = \frac{(t_{Rx} - t_{Tx})}{1E9} \cdot c[c],$$ (3.9)

3.1.3 Фазові вимірювання в Android

Вимірювання фази несучої надаються в Android як AccumulatedDeltaRangeMeters, зі значенням в метрах. Без значення часу вони являються неоднозначними, тобто приймач може лише розраховувати кількість циклів, що відбуваються між епохами. Якщо відбувається збій циклу, приймач не може розрахувати це значення. Дійсність вимірювання несучої надається за допомогою поля AccumulatedDeltaRangeState. Можливі параметри наведено в таблиці 7. Для розрахунку слід використовувати тільки дійсні вимірювання

Таблиця 3.4 Можливі значення при фазових вимірюваннях

Статус	Значення	Опис
ADR_STATE_CYCLE_SLIP	4	Виявлено збій циклу
ADR_STATE_RESET	2	Виявленно стан скидання
ADR_STATE_VALID	1	Валідний стан
ADR_STATE_UNKNOWN	0	Недійсний стан

3.1.4 Робочий цикл

Розробка мобільних телефонів зосереджена на оптимізації використання бааreї. Це впливає як на вибір апаратних компонентів так програмних алгоритмів. Чіпсет GNSS навігації використовує архітектуру A-GNSS, яка відрізняється від чіпсетів, що використовуються в інших пристроях GNSS. В телефонах розміщують дешеві компоненти, найкритичнішми для навігації є вбудовані антена та осцилятор. Осцилятор розташований за межами одиничної мікросхеми та компенсує температурні коливання (TCXO), тим самим поліпшуючи стабільність частоти. Він включається лише тоді коли GNSS чіп використовується, для того, щоб зменшити споживання енергії. Коли GNSS вимкнена, час підтримується за допомогою альтернативного кристального осцилятора з низьким енергоспоживанням, деградація точності складає 6 секунд на тиждень. Пристрій, який був вимкнений протягом кількох днів, матиме грубу апріорну оцінку часу під час увімкнення. Приклад роботи робочого циклу, зображений на малюнку 11

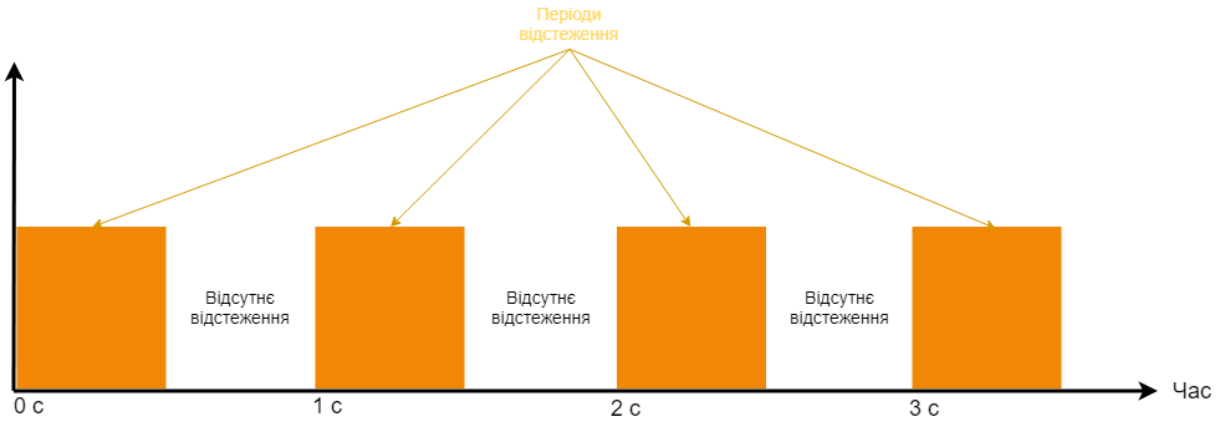


Рис. 3.3 Прицнип роботи робочого циклу

Існують дві різні реалізації робочого циклу, залежно від стану GNSS годнника:

1. TCXO апаратний годинник не є безперервним (вмикається та вимикається) протягом періодів коли не відстежується позиція. У телефоні наявні генератори: високої точності з термокомпенсацією (TCXO), які і використовується для відстеження GNSS та альтернативний, малопотужного споживання, кварцовий генератор (XO). Під час короткого періоду відстеження, коли чіпсет відстежує дані GNSS, використовується генератор з

термокомпенсацією. В інший час, коли GNSS чіп вимикається на кілька сотень мілісекунд, час забезпечується за рахунок кварцового генератора.

Тому значення поля HardwareClockDiscontinuityCount збільшується, а робочий цикл можна визначити, використовуючи необроблені вимірювання.

2. Апаратний годинник TCXO використовується безперервно, в періоди без відстеження: коли чіпсет GNSS вимкнено, годинник TCXO продовжує працювати, навіть якщо сигнал не відстежується. Користувачі не можуть визначити, коли приймачі використовують робочий цикл, так як не існує розриву в годинах GNSS, а значення HardwareClockDiscontinuityCount є незмінним.

Навіть незважаючи на те, що ця методика є прозорою для користувачів, під час обчислення нового місцезнаходження кожну секунду, це впливає на вимірювання фази несучої. Без безперервності спостережень будуть відбуватися збоїв циклів, що відбуваються між двома послідовними вимірами, що тим самим обмежує використання таких розширених технік, як кінематика в режимі реального часу (RTK) або точне точкове позиціонуванн (PPP). Збій циклу являє собою порушення безперервного супроводу супутника. Коли приймач втрачає рахунок через переривання супроводу супутника, процес підрахунку циклів або хвиль зривається. Адже під час фазових вимірювань GPS сигнали розглядаються як послідовності хвиль, переданих кожним супутником. При першому вимірюванні приймач може повідомити, яку частину хвилі він спостерігає. Після першого виміру приймач може відрахувати число спостерігаємих ним хвиль. Це вимірювання повинно бути безперервним, адже, якщо відбувається збій супроводу супутника, то відбувається скидання фази несучої. Відразу після збою, процедура вимірювання починається з самого початку [25]. Окрім робочого циклу, існує ще ряд причин, що викликають переривання прийому супутникового сигналу як при нерухомому так і рухомому обладнанні. Якщо приймач стаціонарний, то зриви можуть відбуватися внаслідок слабкого сигналу, що випромінювався низько

розташованим супутником, або можуть бути викликані перешкодами існуючими на шляху його поширення, наприклад, деревами. Якщо приймач переміщується, то є безліч перешкод, які можуть привести до зривів циклу: рух під мостом. по туннелю, під деревом або просто антена виявилася закритої рукою.

3.1.5 Зсув Допплера

Ефект доплера, що виникає в результаті руху супутника може бути отриманий з PseudorangeRateMetersPerSecond, за умови, що швидкість в момент часу буде в м/с. Це значення в Гц не включає корекції помилок частоти годинника приймача та супутника (незбалансоване значення). Позитивне «некоректоване» значення вказує, що супутник рухається від приймача. Знак «некоректованої» псевдошвидкості і його відношення до знаку зсуву Доплера визначається рівнянням:

$$pseudorangerate = -k * dopplershift, \quad (3.10)$$

де k - константа в залежності від центральної частоти сигналу (наприклад, fc для L1 = 1575.42 Мгц) та швидкість світла (c), як показано нижче, k = c / fc, тобто довжина хвилі.

3.1.6 Ідентифікатор Сузір'я

Ідентифікатор сузір'я надається безпосередньо Android 7 за допомогою поля ConstelationType. Значення для кожного сузір'я перелічені в Таблиці 3.5

Таблиця 3.5 значення ConstelationType для всіх систем навігції

Тип сузіря	
статус	значення
CONSTELLATION_BEIDOU	5
CONSTELLATION_GALILEO	6
CONSTELLATION_GLONASS	3
CONSTELLATION_GPS	1
CONSTELLATION_QZSS	4

CONSTELLATION_SBAS	2
CONSTELLATION_UNKNOWN	9

3.2 Використання необроблених навігаційних даних в ОС Android

До 2016 року API Location, був обмежений Google Play Services (android.gms.location). Цей API сфокусований на спрощеній та інкапсульованій інформації про місцезнаходження, використовуючи сукупність сенсорів (GNSS, WiFi, Bluetooth) що забезпечує оптимальне використання батареї. Всі обчислення відбуваються всередині чіпа: блоки збору та відстеження декодують навігаційне повідомлення і генерують псевдодальності, вимірювання фази та час. Вони виправлені за допомогою навігаційного повідомлення (помилки годинника, іоносфери та тропосфери тощо). Нарешті, позиція, швидкість та час (PVT) обчислюються та виводяться чіпсетом. Новий API (android.location) забезпечує прямий доступ як до необроблених спостережень GNSS, так і до рішення щодо PVT. На рисунку показана архітектура підсистеми приймача та основні відмінності між новим та старим програмним інтерфейсом. Білими блоками відображені нови параметри програмного інтерфесу. Інші параметри, такі як позиція супутника або C/No не відображені. Псевдодальності не можна отримати за допомогою нового API безпосередньо, але параметри, необхідні для їх створення, є в наявності.

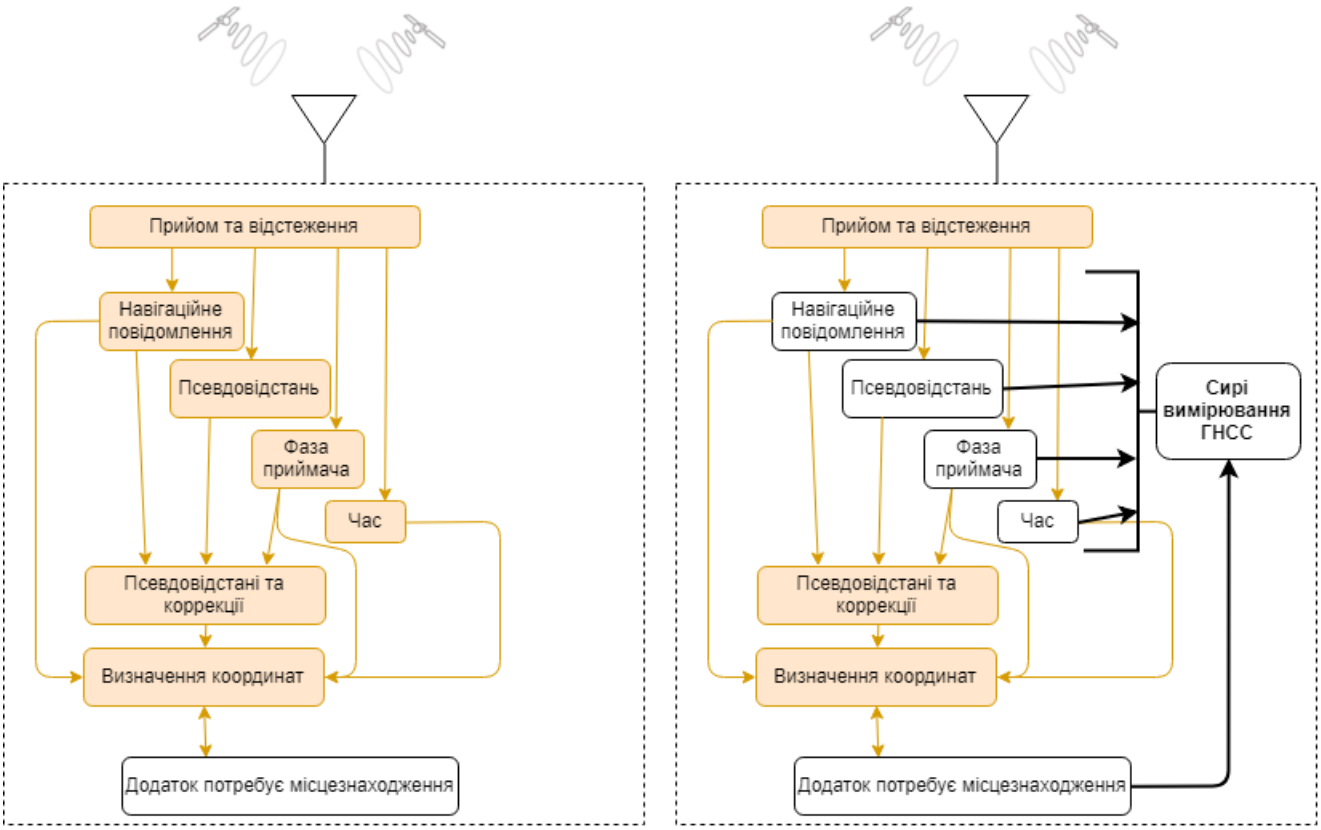


Рис. 3.4 Порівняння android.gms.location та android.location

Android додатки, що базуються на місцезнаходженні звертаються до даних через клас android.location.LocationManager, вказуючи провайдера місцеположення (мережа, Wi-Fi або GPS/GNSS) або використовуючи змішаного провайдера залежно від критеріїв точності та потужності.

Якщо використовувати необроблені дані, ця архітектура може бути розширена до власного постачальника геоданих, який може базуватися, наприклад, на згладжених псевдодальностях або диференційних даних GPS. Власний провайдер повинен обчислюватися на рівні операційної системи. Основними джерелами помилок GNSS є супутникові (орбіти і годинник), атмосферні (іоносферні та тропосферні затримки), шум та локальні ефекти (включаючи багатопроменеві). Диференціальні спостереження покращують точність позиціонування, надаючи зовнішні корекції до цих помилок, але, як правило, не корегують локальні ефекти, такі як багатопроменевість або перешкоди. Існує кілька методів для цього, причому найбільш часто використовуваним є DGNSS (або DGPS, Диференціальний GNSS / GPS) для

кодового рішення та кінематики в режимі реального часу (RTK), Network RTK і точного позиціонування точок (PPP) для рішень фаз. Ця нова архітектура зображена на рисунку 13, яка показує існуючу інфраструктуру в сірому та нову, що дозволяється за допомогою неочищених вимірів, в оранжевому кольорі.

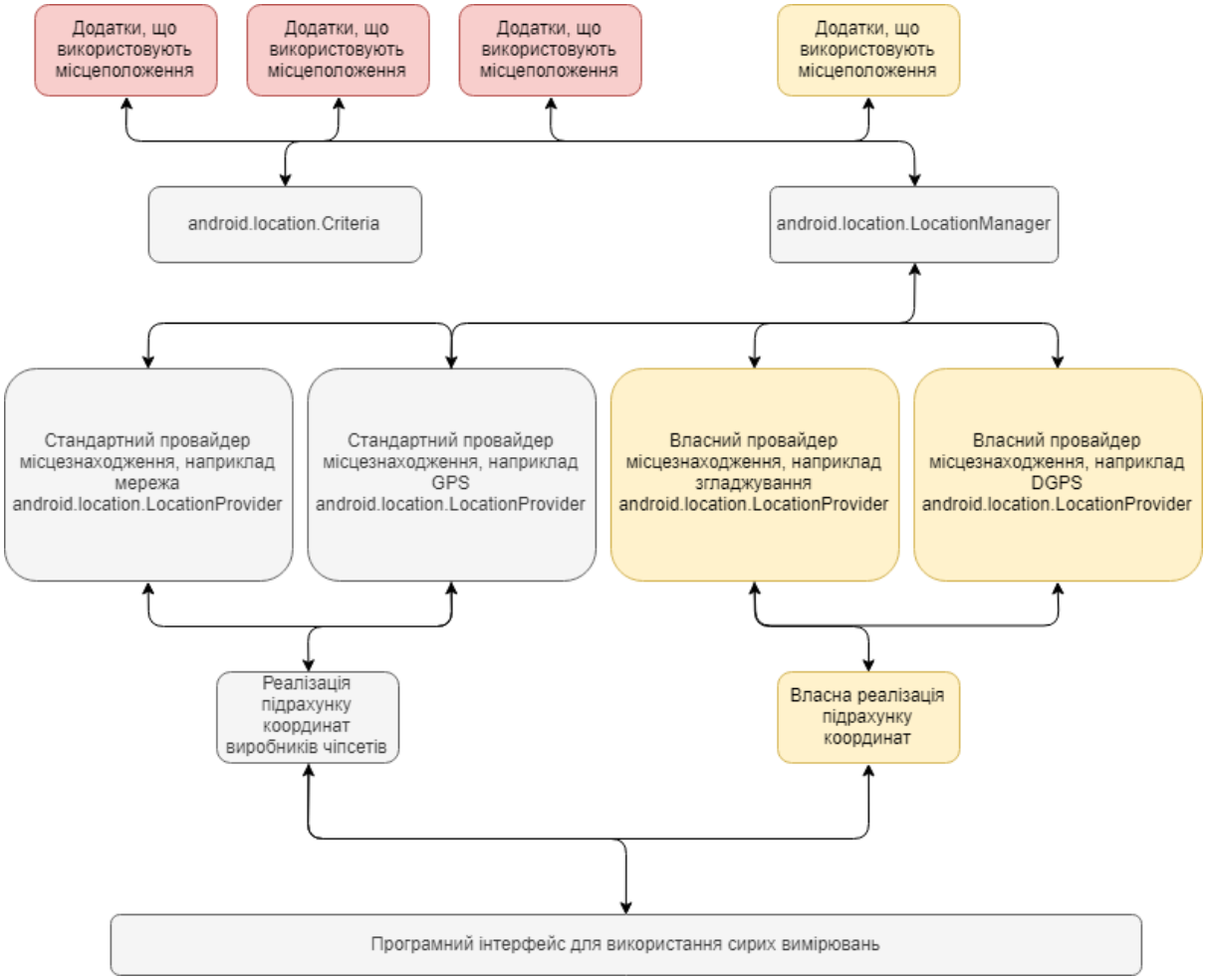


Рис. 3.5 Архітектура використання необроблених GPS даних в Android

3.2.1 Використання A-GPS

Основним пріоритетом під час проектування GNSS чіпів є зручність використання, за допомогою мінімізації TTFF (time to first fix, час першого виправлення) та покращення доступності та безперервності позиціонування. Необроблені дані, доступні через android.location API, дозволяють використовувати розширені техніки для визначення місцезнаходження. Використання додаткової інформації для зменшення помилок може підвищити точність визначення координат. Необроблені вимірювання також

підтримують інтеграцію з іншими сенсорами, що знаходяться в сматфоні, та з додатковою інформацією, такою як частота Доплера або співвідношення сигнал шум, може забезпечити вищу точність оцінки, надійності навігації та додатковий рівень безпеки. Можливе використання A-GPS у додатку для пришвидшення визначення приблизного місцезнаходження. Основною метою A-GNSS є зменшення TTFF, що досягається завдяки використанню доступу до пристрою зв'язку (такий як 4G модем або WiFi-зв'язок). Мобільний пристрій отримає:

1. Орієнтовну оцінку позиції смартфону, наприклад, через позиціювання Cell-ID та початковий час, з точністю в залежності від мережевої інфраструктури.

2. Допоміжні дані, що включають ефемериди супутників, корекцію годинника та іоносферні моделі (такі, які транслюється супутниками GNSS) [23]. Або прогнозновані GNSS ефемериди або довгострокові орбіти [26], що дозволяють отримувачу обчислювати майбутні позиції супутників на декілька днів, навіть тижнів вперед без активного з'єднання з мережею.

3. Підвищення опорної частоти шляхом калібрування локального осцилятора в телефоні. Вона відкалібрована сигналом, отриманим від базової станції, частота якої, як правило, відома в межах 5*10^-8, і частота несучої частоти мобільного телефону відома в межах 10^-7.

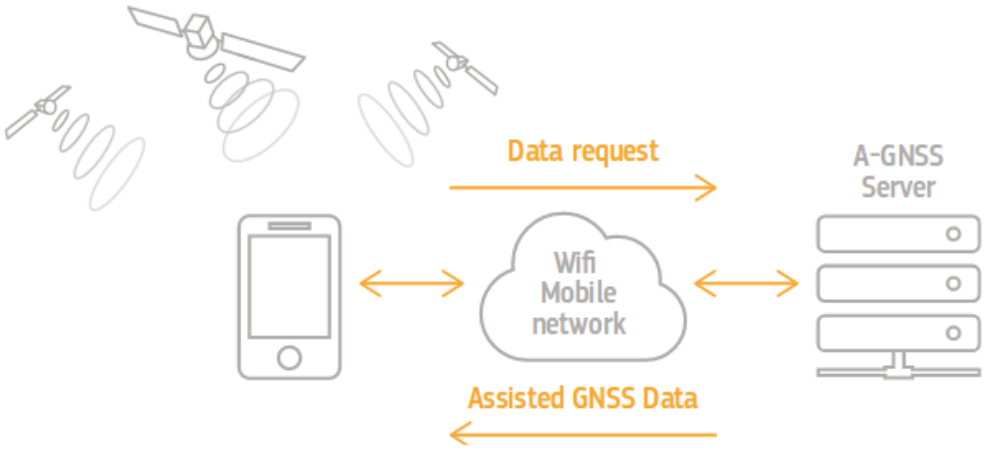


Рис. 3.6 Прицнип роботи A-GPS

Ця інформація підвищує точність приймача та забезпечує перше виправлення позиції протягом декількох секунд, навіть при складних умовах (міські або внутрішні середовища), оскільки вся інформація доступна на початку. Знання ефемеридних даних також підвищує доступність та точність позиції в суворих умовах.

Недоліком є те, що перешкоди між сусідніми радіопередавачами, які часто знаходяться всередині смартфону в невеликому просторі та розташовуються в одному чіпі, збільшують складність архітектури GNSS-чіпа і можуть погіршити його продуктивність. Типова антена, що використовується в мобільних телефонах, - це недорога PIFA (Platar Inverted-F Antenna), як правило, мікрополоскова антена, побудована з використанням PCB плати. Вона підходить для будь-якого сузір'я в смузі частот L1, такий як GPS / Galileo, GLONASS/Beidou. Проте його лінійна поляризація (замість кругової) та діаграма спрямованості призводять до втрати кількох дБ сигналу. Зокрема, спрямованість може суттєво послабити сигнали від деяких супутників, одночасно належним чином отримуючи сигнал від інших. Місце розташування антени обумовлено більше дизайном смартфону, ніж практичними причинами. Що може призвести до погіршення визначення положення, у випадках, коли користувач затуляє рукою антену смартфону, що ще більше послаблює прийом сигналів GNSS.

3.2.2 Базова продуктивність кодового позиціонування

Динамічний сценарій, розглянутий університетом Нотінгему [28], де Android пристрій був розміщений на інформаційній панелі рухомого автомобіля, слугує візуалізацією всіх апаратних обмежень смартфонів. Результати кінематичних випробувань показані на малюнку 16, де товста жовта лінія відображає істинні координати, надані тактичним класом IMU, синя лінія відображає шлях, визначений за допомогою Android планшету Nexus 9 без передачі даних та використанням супутників GPS та GLONASS. Зелена лінія відображує кодове рішення, отримане за допомогою недорогого одночастотного приймача з геодезичною антеною із стійкістю до

багатопроменевого поширення. Основний вклад у різницю між синьою та зеленою лініями вносить лінійно поляризована антена яка не підходить для видалення багатопроменевих сигналів у міському середовищі (зображення зліва). Зображення праворуч відповідає сільському середовищу без перешкод. При відсутності багатопроменевих сигналів Android може підтримувати належний рівень точності використовуючи дешевий GNSS приймач. Але точність значно погіршується при наявності відбитих сигналів (будинок на лівому зображенні).

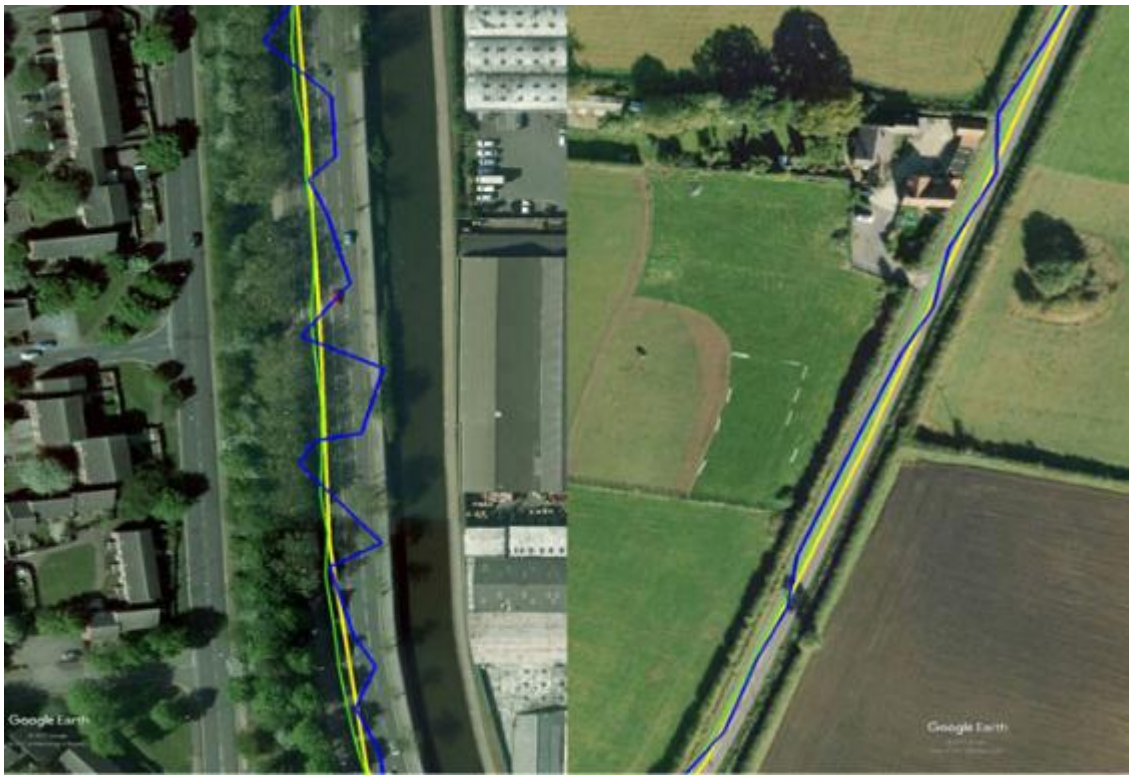


Рис. 3.7 Результати динамічного тесту з використанням стандартних алгоритмів Android

На продемонстровану продуктивність Android також впливає розташування пристрою на панелі пристроїв за вітровим склом.

На рисунку 17 показано різницю в кількості супутників, що використовуються для рішення PVT, між антеною, розташованою на даху автомобіля та всередині транспортного засобу. Дані були зібрані в окремих 6-годинних тестах, що проходили через автомагістралі та міські середовища. Кількість супутників більша для телефону розміщеного на даху автомобіля,

особливо під час проїзду на автомагістралях, де різниця майже вдвічі перевищує кількість супутників.

Це, в основному, пов'язано з перешкодою конструкції автомобіля та додатковим ослабленням (і багатопроменевістю) отриманого сигналу.

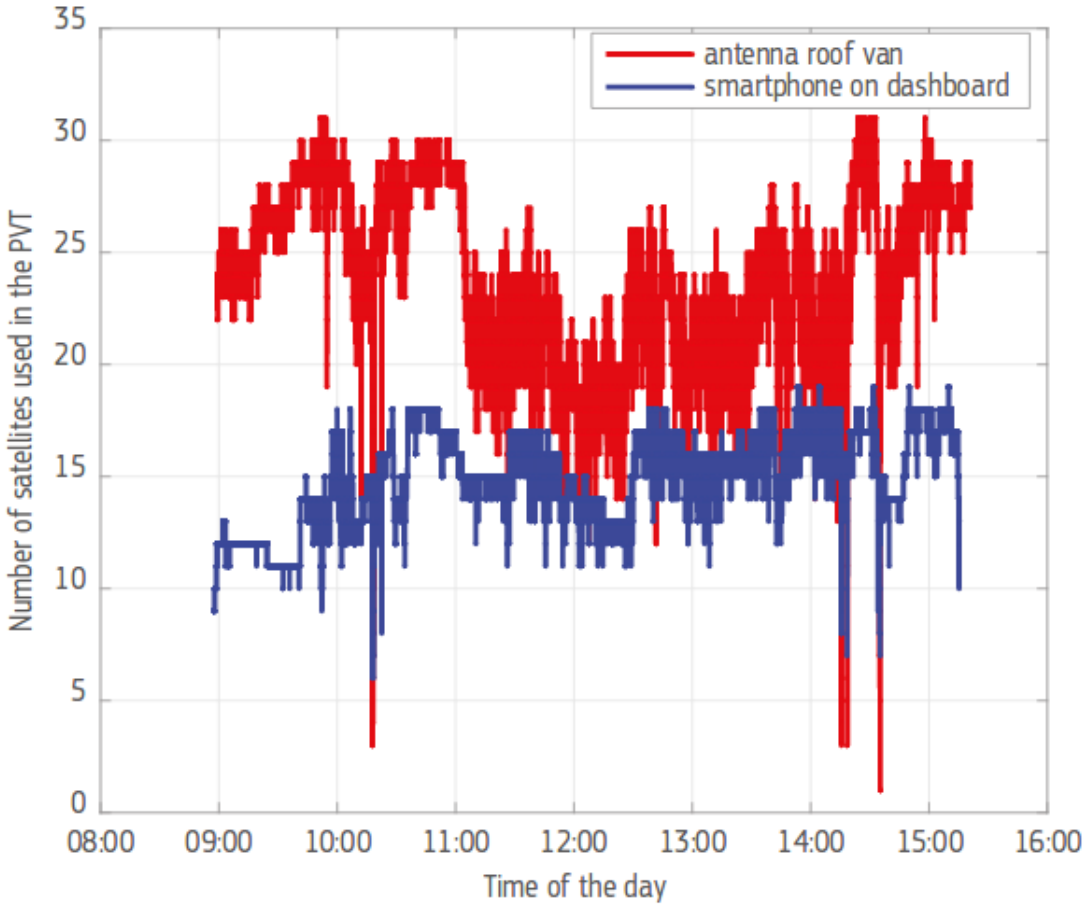


Рис. 3.8 Супутники в зоні видимості залежно від розташування смартфону

Так як Android Location API дозволяє отримати значення зсуву Доплера за допомогою поля класу PseudorangeRateMetersPerSecond, ці дані можливо використати для покращення точності позиціонування. Так, для підвищення статичної точності позиціювання, можливо використовувати фільтр Хатча для згладжування кодових рішень [29]. Так як вимірювання зсуву Доплера не зазнають значного впливу від роботи робочого циклу, його можливо використовувати для покращення точності позиціонування. На малюнку 19 наведено порівняння статичного рішення PVT, отриманого з необроблених

даних з використанням кодових вимірювань та поправки зсуву Доплера, з різним часом інтеграції. За допомогою ковзаючого вікна в 20 секунд, 3D помилка (при 68%-у довірчому рівні) знижується майже в три рази, з 19,7 метрів до 7,6 метрів [28].

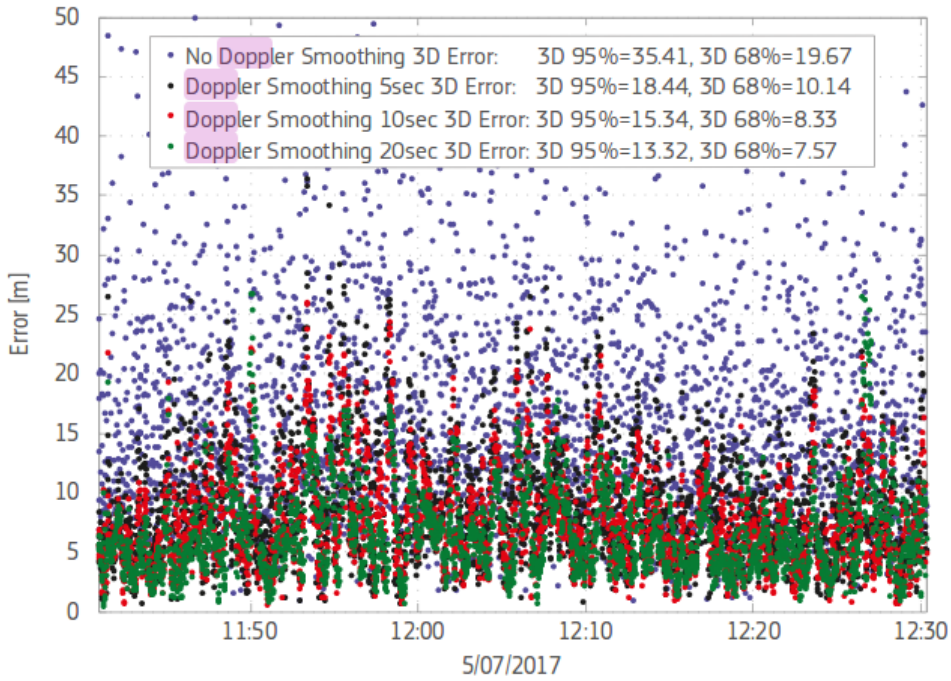


Рис. 3.9 Порівняння продуктивності залежно від розміру ковзаючого вікна

При динамічному сценарії, автомобіль, який рухався по автостраді, вікно згладжування було встановлене в 20 секунд, який було обрано на основі хороших результатів, отриманих за статичним сценарієм.

Згладжування підвищує точність, хоч і менше, ніж у статичному сценарії, але призводить до більш плавного шляху, навіть у сценарії високодинамічного шосе (Малюнок 20). Довідкова траєкторія (зелена) була розрахована з високоточним приймачем GNSS.

Таблиця 3.6 Порівняння точності PVT з використанням згладжування та без

Режим	Рівень впевненості		
	68%	95%	Доступність
Згладжування 20 с	9,38	18,87	97,79%

Без згладжування	13,36	25,51	97,79%

Рис. 3.10 PVT рішення — стандартне кодове рішення (праворуч) та з використанням згладжування

3.2.3 Відключення робочого циклу

Робочий цикл, який запускається телефоном для зменшення енергоспоживання, впливає на генератор і призводить до збою циклу фазових вимірювання. У дослідженні, проведеному Airbus Defence and Space GmbH [28] оцінювався вплив робочого циклу на збої циклу. В ході дослідження було виявлено, що відключення робочого циклу надає змогу отримати близько 92% дійсних фазових вимірювання, кількість збоїв циклу не перевищує 10%. Причому, при увімкненому робочому циклі, близько 40% вимірювань зазнали збоїв циклу, тому лише 60% вимірювань є коректними. Для першого дослідження (без робочого циклу) три супутники (5,21,27) показали надзвичайно велику кількість втрачених циклів, що може бути викликано поєднанням низьких висот і наявністю шуму. Вони були вилучені з розрахунків. Рисунок показує статичні дані, зібрані без активного циклу. Результати роботи смартфона з активним робочим циклом демонструє лише 60% придатних до використання даних (Таблиця), що робить майже неможливим використання RTK. Проте, безперервне використання GNSS

чіпсету впливає на витрати акумулятора, таким чином значно зменшуючи зацікавленість користувача в додатках з підвищеними енерговитратами.

Наступна таблиця демонструє співвідношення коректних даних до збоїв циклу при вимкненому робочому циклі.

Таблиця 3.7 Фазові вимірювання при вимкненому робочому циклі

№ супутника	Фаза коректна	Збій циклу
5	26,73	73,26
10	78,75	21,24
16	92,96	7,03
18	94,92	5,07
20	99,63	0,36
21	13,95	86,04
26	98,98	1,01
27	7,69	92,30
29	96,51	3,48
31	99,63	0,36
GPS агрегована	92,86	7,13

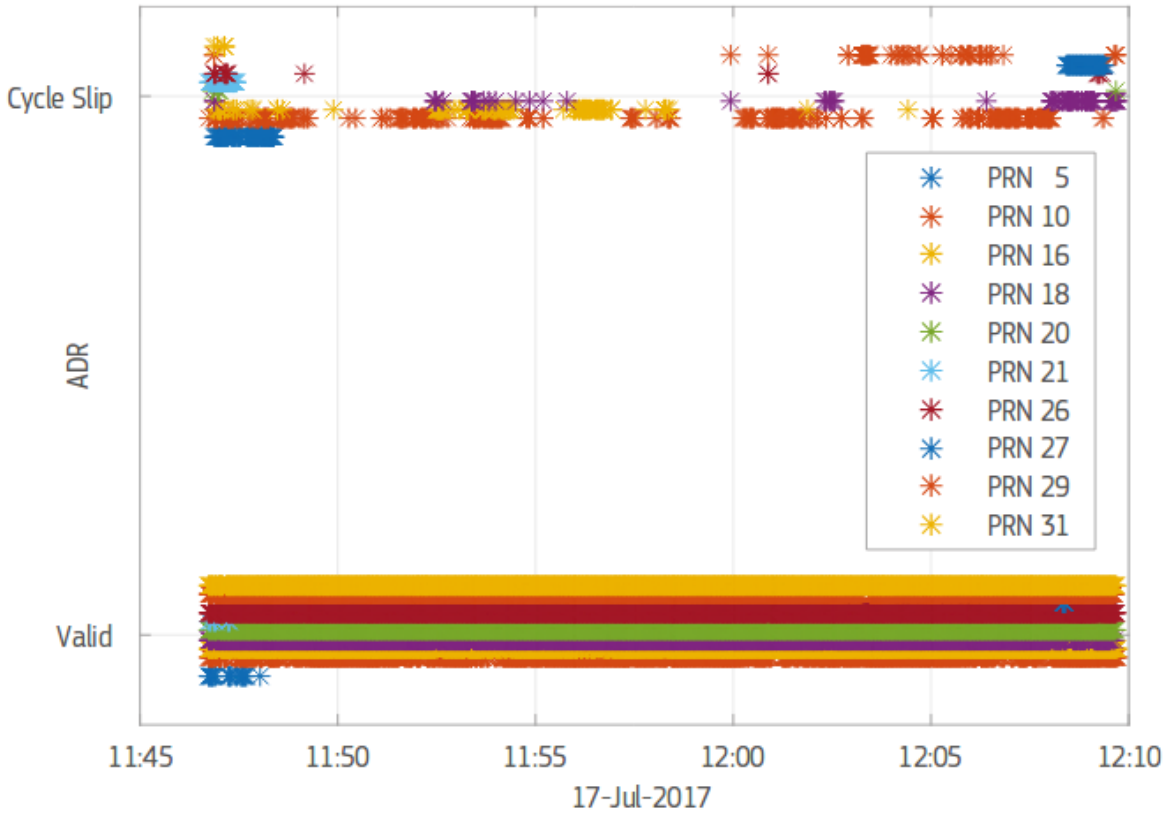


Рис 3.11 Кількість сбоїв циклу при вимкненому робочому циклі

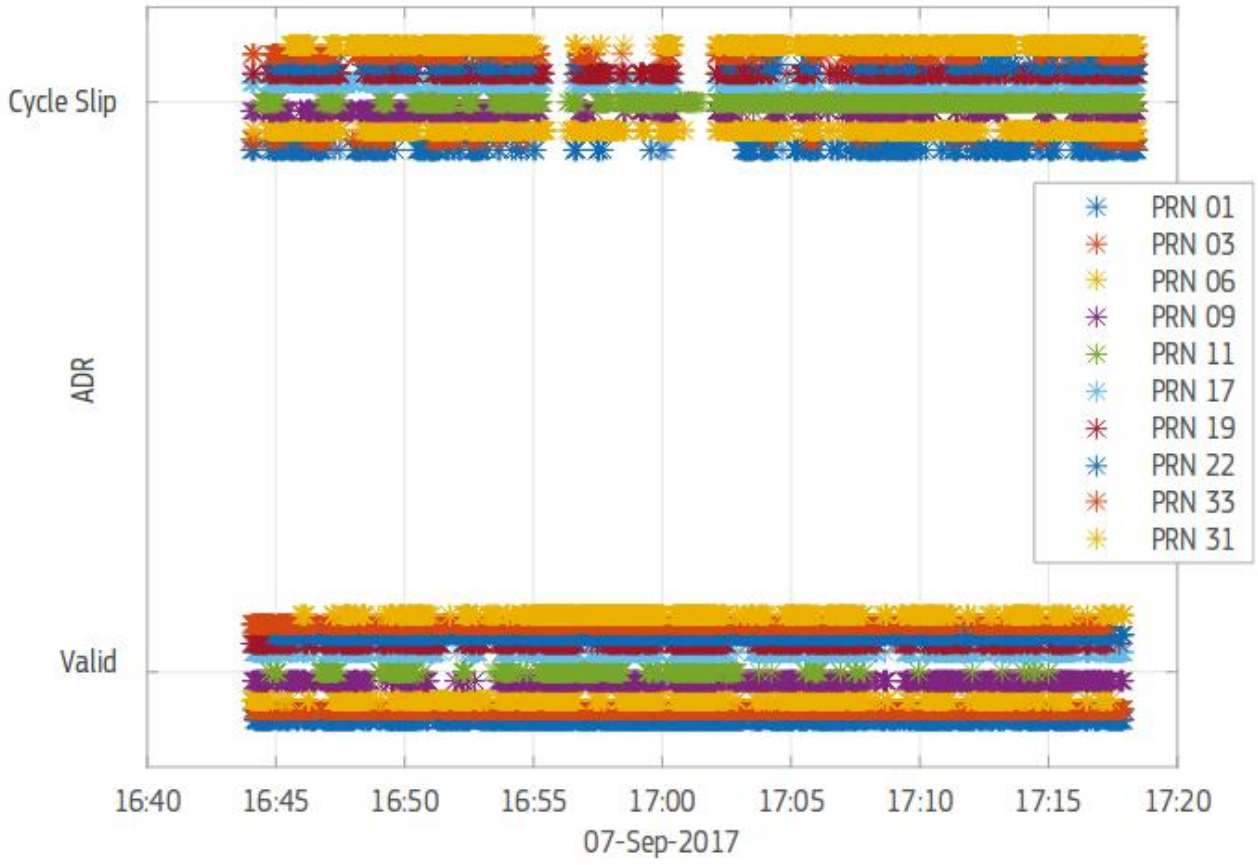


Рис. 3.12 Кількість сбоїв циклу при увімкненому робочому циклі

Таблиця 3.7 Фазові вимірювання при увімкненому робочому циклі [28].

№ супутника	Фаза коректна	Збій циклу
1	64,05	35,94
3	85,29	14,70
6	49,72	50,27
9	58,81	41,18
11	32,45	67,54
17	44,26	55,73
19	51,76	48,23
22	76,47	23,52
23	85,34	14,76
31	46,23	53,76
GPS агрегована	92,86	7,13

3.2.4 Реалізація прийому диференційної GPS поправки

DGPS поправки передаються за стандартом RTCM. Стандарт RTCM використовує 30-и бітний формат слів 24 з яких є інформаційними, а 6 біт парності, що використовуються для перевірки на наявність помилок. Слова формують кадр, довжина якого не є постійною величиною через відмінність обсягу данних, що передається у повідомленні. Перші два слова кожного кадра називаються заголовком. Повідомлення 1-17 були доступні у перших версіях RTCM, а повідомлення 18-21 були додані у версії 2.3, що дало можливість застосовувати даний стандарт до виправлень RTK. На таблиці представлені найбіль широко використовані повідомлення

Таблиця 3.8 Типи повідомлень в RTCM 2.3 [30].

Ідентифікатор типу повідомлення	**Призначення повідомлення**

1	Диференційні GPS поправки
2	Дельта DGPS поправки
3	Параметри опорної станції
5	Статус сузір’я
6	Нульовий кадр
9	Частковий набір поправок
11	Поправки C/A коду
13	Параметри наземного передавача
14	GPS TOW
15	Інформація про іоносферну затримку
16	Спеціальне GPS повідомлення
17	GPS Ефемериди
18	Некориговані вимірювання фази RTK
19	Некориговані вимірювання псевдодальності RTK
20	Фазові поправки RTK
21	Поправки псевдодальностей RTK

Повідомлення за стандартом DGNSS RTCM-104 передаються через лінії зв'язку - наприклад, через радіозв’язок або через мережу мобільного зв'язку. Для передачі RTCM поправок через мережу Internet використовується протокол Ntrip Система Ntrip заснована на стандарті потокового HTTP. Спочатку HTTP призначений для об'ємного трафіку, але він також використовується для потокових передач від додатків. Основною метою протоколу Ntrip є забепечення користувачів даними DGPS поправок в режимі реального часу. Ntrip складається з 4-ох основних компонентів, а саме:

• NtripSources, які генерують RTCM дані, це GPS приймачі на опорних станціях. Окреме джерело відноситься до окремої визначеної локації;

• NtripServers, сервер, який слугує для передачі даних до кастера;

• NtripCaster, це основний компонент системи, який являє собою HTTP сервер, що приймає запити як від серверу так і клієну по одному порту, та приймає рішення про відправку чи прийом даних;

• NtripClients, клієнт, який повинет відправити запит на NtripCaster для отримки даних [31].

Реалізований Ntrip клієнт, який при наявності логіна та пароля під'єднується до кастера та отримує DGPS поправки (Додаток А). Також реалізовано програмний модуль, що визначає координати користувача з використанням DGPS поправки та без (Додаток Б)

3.3 Сфери застосування додатків з використанням необроблених даних

Необроблені вимірювання дуже цікаві з наукової точки зору. Пристрої Android можуть бути розгорнуті як мережа датчиків, з огляду на досить низьку вартість апаратного забезпечення та відсутність необхідності розробки спеціального програмного забезпечення. Однією з наукових цілей тут передбачено детальний моніторинг атмосфери з використанням лише однієї частоти. Інші програми та використання включають в себе наявність базової станції, виявлення інтерференції і використання як частину сенсорів розумного міста.

Спостереження також можуть бути використані для порівняння рішень окремих сузір'їв (наприклад, позиціонування лише для Galileo на рисунку 25), усунення певних супутників або візуалізація найгіршої продуктивності. Такий самий підхід може також використовуватися для перевірки апаратних та програмних рішень, спостереження за продуктивністю та порівняння цієї продуктивності з базовим рішенням, наданим самим чіпсетом.

Прикладом є інструмент аналізу GNSS [32], який надає компанія Google. Він обчислює PVT методом найменших квадратів з використанням вхідних

необроблених вимірів (статичних та кінематичних), які потім можна порівняти з відомим значеннями. Програмне забезпечення забезпечує візуалізацію сигналу (Сигнал/шум, діаграми супутників), годинників (супутника, приймача і робочий цикл) та вимірювання (точність позиції, псевдодальності), надаючи користувачу можливість огляду системи.

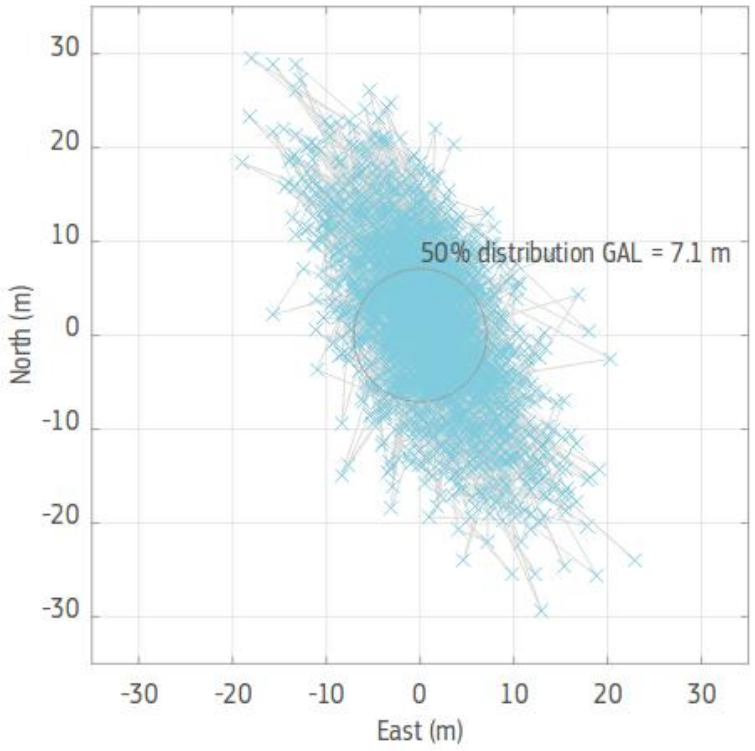


Рис 3.13 Статичне позиціонування з використанням необроблених даних для сузір’я Galileo

Також існує зростаюча зацікавленість у дешевих способах виявлення наявності інтерференції серед GNSS сигналів. Це, в основному, необхідно для виявлення та діагностики низької продуктивності GNSS та вирішення проблем з відмовою в обслуговуванні. Сітка мобільних телефонів може забезпечити загальну картину переривань в рамках частоти GNSS. Доступ до необроблених вимірювань у вигляді окремих псевдальностей та значень сигнал/шум, а також значень AGC (автоматичного регулювання підсилення) дасть змогу розробити нові способи виявлення перешкод, пов'язаних із радіочастотним сигналом, за допомогою самого пристрою. Крім того, завдяки поєднанню даних вимірювань з кількох пристроїв Android у регіоні також буде можливість знайти джерело перешкод. Доступ до таких можливостей створить

перспективи для розробки нових служб для допомоги користувачам GNSS, провайдерам послуг, операторам інфраструктури та національним частотним органам контролю.

RAIM - це алгоритмічна оцінка цілісності сигналу GNSS, яка забезпечує оцінку рівня впевненості у продуктивності позиціонування, яка може попереджати користувача, якщо продуктивність нижча за встановлений поріг і потребує іншого алгоритму для визначення координат. Вони призначені для критично важливих програм GNSS, таких, які використовуються в авіації, морському судноплавстві або у залізничній галузі. З невеликими змінами вона може бути перенесена на платформу Android, надаючи користувачам більшу впевненість у точності позиціонування, додаткову перевірку та попередження.

Дане рішення може використовуватися для мобільних пристроїв, які використовуються коли визначення місцеположення є життєвоважливим фактором (наприклад, для направлення бригади швидкої допомоги). Така перевірка може також забезпечити інший рівень безпеки для онлайнових транзакцій, надавати більшу ясність у спілкуванні та зменшити кількість людських помилок. У поєднанні з іншою інформацією, такою як позиції спутників, GPS додатки на сматфонах Android можуть бути розширене до систем, які передбачають області, в яких можливе зниження GPS продуктивності [33]. RAIM також може використовуватися у волонтерській географічній інформації (VGI), де географічні дані, створені користувачами, вимагають як достовірності положення, так і контролю за якістю зібраної інформації [34]. В даному випадку можливо використовувати той самий принцип, щоб надати візуальне попередження користувачу, інформуючи, коли точність визначення місцезнаходження в ОС Android може не відповідати певним вимогам до точності, дозволяючи користувачеві більш обізнано використовувати інформацію.

Окрім додатків для навігації можна виокремити декілька сфери для яких підвищення точності визначення місцезнаходження також має ключову роль:

- Геомаркетинг

- Доповнена реальність
- Здоров’я та безпека
- Спорт
- Інтернет речей

Геомаркетинг це реклама, що базується на основі місцеположення. Більше 70 відсотків рішень про придбання товару здійснюються на місці продажу. Саме тому маркетинг грає значну роль для успішності магазину. Показники взаємодії можуть збільшуватися до 30%, у випадках, коли данні для місцезназодження використовуються для надсилання реклами [35].

Якщо орієнтуватися на користувачів мобільних пристроїв, розташованих поблизу певного закладу, то ефективність кампанії буде знижена через неточність визначення координат. Дослідження 2014 року, проведене Асоціацією мобільного маркетингу, яка відстежувала точність даних про місцезнаходження, щоб покращити ефективність геотаргетингу, виявила, що лише 1/3 частина рекламних повідомлень була в межах 50-100 метрів від зазначеного місця.

Аналогічна ситуація стосується всіх додатків з використанням геозон, наприклад, мережа закладів, яка надає безкоштовний WiFi для всіх своїх відвідувачів, таким чином встановлюється зона в якій можливе використання WiFi точки, наприклад, виключно на території закладу, або додатки для батьків, які слідкують за переміщенням дітей. У геофізичних програмах абсолютна точність розташування не є критичною. Важливою є здатність знаходити користувача в межах або за межами певної віртуальної зони.

В той же час, навмисне втручання, таке як джемінг (придушення сигналів) або спуфінг, стане проблемою - особливо, якщо додатки з геозонами повязані з оплатою за певні послуги (наприклад, паркування на основі місця розташування). Використання необроблених вимірювань дозволяє вносити додаткові алгоритми обробки GPS сигналів, для підтримки перевірки цілісності та виявлення підробок сигналів.

Точність розташування також важлива для додатків, що використовуються у сфері доповненої реальності (AR). Доповнена реальність покращує сприйняття дійсності зображеннями, створеним комп'ютером, яке накладається на справжнє зображення. Визначення місцеположення в трьох вимірах є ключем для синхронізації фізичного та цифрового світів, з точністю, яка залежить від конкретного остаточного застосування в додатку.

Деякі програми доповненої реальності вимагають високого ступеня синхронізації, що означає необхідність у збільшенні точності визначення місцезнаходження користувача. Очікується, що до 2022 року світовий ринок додаткової реальності досягне 108 мільярдів доларів [36].

Існує три категорії інструментів додаткової реальності: 3D-окуляри з доповненою реальністю, браузери з доповненою реальністю та сматфони. Приклад першої категорії можна знайти в будівельній галузі. Промисловість рухається у напрямку використання додаткової реальності для візуалізації географічних моделей будівельних майданчиків, підземних споруд, кабелів і труб за допомогою мобільних пристроїв. Тут висока точність розташування є ключовою вимогою. Наприклад, якщо система AR дозволяє користувачеві відобразити трубу, або кабельні лінії під землею, а місце розташування цієї труби недостатньо точне, підрядчики можуть пошкодити підземний об’єкт, під час земельниї робіт.

У 2016-ому році на сматфони була випущена гра Pokemon Go, яка викликала справжній резонанс серед населення Америки та Европи. Основний принцип гри базувався на GPS координатах користувача, створюючи навколо користувача ігрових персонажів, яких потрібно було зловити. Гра захопила одразу велику кількість людей. Заринувшись у процес гри люди не зважали на реальні ризики таких ігор. Так нерідко траплялись випадки, коли люди в процесі гри виходили на проїзджі частини або залізнодорожні шляхи. Окрім цього нерідкими були випадки гри під час керування велосипедом, автомобілем та мотоциклом, що пизводило до аварій та травм. Дослідження проведене школою керування Краннерта, оцінювало кількість аварій та травм

отриманих через участь у грі Pokemon GO. У доповіді зробили висновок: "Зростання аварій, пов'язаних з введенням Pokemon GO, становить 145 632, з відповідним збільшенням кількості травм до 29,370 і пов'язаним із цим збільшенням кількості загиблих, рівним 256 за період з 6 липня 2016 р. по 30 листопада 2016 р." [37]. Автори оцінили збитки завдані цими аваріями у межах від 2 до 7 млрд. доларів США за той же період. У документі також зазначено, що частота дорожньо-транспортних пригод збільшилась, тим самим перервавши довготривалу тенденцію до зниження, з моменту появи смартфонів. Таким чином, підвищена точність рішень PVT дозволила б ввести обмеження на гру. Наприклад, при переміщенні користувача зі швидкістю, вищою ніж 10 км/год вимикати усі ігрові процеси, або використовуючи координати вираховувати прийнятні місця для появи ігрових об’єктів, тобто уникати їх появни у місцях з підвищеною небезпекою.

Приклад із сфери застосуван**ня, пов'язаного з охорони здоров’я,** де використовуються пристрої споживачів, - це мобільна охорона здоров’я (mHealth).

mHealth додатки охоплюють медичні практики та практики з охорони здоров’я, що підтримуються мобільними пристроями, включаючи зв'язок та збір даних. Сюди можно віднести всі програми для підтримки здорового способу життя та фітнесу, що використовують портативні пристрої з підтримкою GNSS. Підвищена точність місцезнаходження буде корисною, наприклад, у програмах для спостереження за спортом та фізкультурою, навігацією людей із вадами зору та людей з обмеженими фізичними можливостями, а також геозон для пацієнтів з хворобою Альцгеймера. Область застосування в цій категорії дуже широка. Кількість додатків також швидко зростає, особливо за рахунок наявності фінансування, яке доступне для інновацій навколо соціальної інтеграції. Підвищена точність визначення місцезнаходження покращує оцінку зусиль, дистанції та швидкості руху у таких видах спорту, як веслування, гольф, легка атлетика, футбол, катання на

лижах тощо, дозволяючи визначити індивідуальну продуктивність та продумувати шляхи для вдосконалення [38].

Покращена точність позиціонування в Android може призвести до зростання недорогих пристроїв, які використовуються як рентабельну альтернативу для аналізу продуктивності спортсменів.

Технологія GNSS через слухові та тактильні інтерфейси дозволяють сліпим і людям з вадами зору краще зрозуміти навколишнє середовище та підтримувати їхню мобільність. В даному випадку точність необхідна, щоб користувач міг визначити конкретний пункт призначення або точки інтересу, наприклад магазин або медичний заклад, перед яким вони перебувають у конкретний момент часу.

Екстренний виклик 112 з підтримкою GNSS - це версія універсального європейського екстреного виклику 112, де інформація про місцеположення автоматично надсилається в екстрений центр із самим викликом. За даними EENA [39], 70-80% екстрених дзвінків в Європі походять з мобільного телефону, однак інформація про місцезнаходження місцезнаходження абонента, яка надається аварійним службам, часто є неточною та надається з затримкою. Адже людина в стресовій ситуації не завжди може надати всю необхідну інформацію, крім того, у певних випадках, просто неможливо описати точне місцезнаходження адже немає ніяких орієнтирів. В даний час використання Cell-ID є найпоширенішим рішенням, тобто приблизне розташування визначається за рахунок інформації про сектор конкретної базової станції, до якої підключений абонент. Використання GPS забезпечує точне позиціонування та забезпечує кращий доступ та оцінку на розподіл ресурсів, тим самим зменшуючи час реагування служб порятунку. Медичні дослідження показують, що зменшення часу відповіді на одну хвилину збільшує шанс виживання на 24% [39]. Що безумовно необхідно брати до уваги в додатках пов’язаних з охороною здоров’я.

Наступне застосування позиціонування підвищеної точності знаходить в сфері інтернету речей (IoT), що дозволяє фізичним пристроям, транспортним

засобам, будинкам та іншим об'єктам бути взаємопов'язаними та дистанційно керованими, створюючи мережу сенсорів. Очікується, стрімкий розвиток IoT через мережеві інфраструктури, створюючи мережу з взаємоповязаними датчиками та технологіями, в яких позиціонування та синхронізація здійснюється через GPS. Це буде важливим механізмом реалізації LBS сервісів, з даними з смартфонів. Що дозволяє створювати дуже персональні комп'ютерні служби та покращувати точність позиціонування шляхом створення моделей на основі даних.

Висновок

1) Для дослідження можливості використання диференційних поправок в ОС Android проаналізовано, можливості отримання даних, необхідних для розрахунку місцезнаходження користувача.
2) Наведено основні відмінності Android Location API 23 та 24, які включають в себе наявність необроблених даних
3) Засобами Android не надаються безпосередньо значення псевдодальностей до супутників, проте надаються всі необхідні дані для їх розрахунку
4) Наведені порівняння базової продуктивності GPS приймача смартфону з спецалізованим приймачем, а також наведені приклади використання розширених технік в Android смартфонах
5) Реалізований Ntrip клієнт для Android додатку, який дозволяє отримувати диференціальні поправки через мережу Інтернет. А також, модуль розрахунку місцеположення з використанням диференційної GPS поправки

ВИСНОВОК

У першому розділі описаний основний принцип функціонування системи GPS, описані три сегменти, а саме космічний, контролюючий та сегмент користувача. Визначено поняття GPS часу, відмінностей його від часу UTC

Описані два основні принципи розрахунку псевдодальностей на основі C/A коду, а саме за методом загального часу прийому та методом загального часу передачі. Також описані принципи визначення місцеположення при розрахованих значеннях псевдодальностей та відомих позиціях супутників в момент передачі. Позиція користувача оцінюється, використовуючи вимірювання відстані (псевдовідстані) між антеною приймача користувача та позицією щонайменше чотирьох супутників. Обидва параметри визначаються приймачем, який оцінює сигнал супутника та навігаційне повідомлення, відповідно. Ця інформація вимагається рішенням PVT (Position Velocity Time), яке визначає розташування та час користувача в будь-якій точці земної кулі.

Визначені основні джерела похибок GPS, а саме: вплив тропосфери та іоносфери, багатопроменеве поширення сигналів, похибки ефемерид та часу і розташування супутників.

У другому розділі оцінювалися методи диференційного позиціонування, а саме DGPS та RTK. Наведені основні принципи роботи цих методів. Так точність автономного GPS приймача знаходиться в межах 5-9 метрів, то вона не є достатньою для більшості випадків. Для компенсації похибок використовуються методи DGPS та RTK. Обидва методи використовують опорну станцію, яка передбачає наявність GPS приймача, який буде розташований на відкритій місцевості, з визначеними наперед координатами, а також наявність каналу зв’язку з користувачами. Метод DGPS дозволяє компенсувати більшу частину похибок, які впливають на визначення місцезнаходження, а саме вплив тропосфери та іоносфери, похибки пов’язані з ефемеридами та похибки супутникового годинника. Тоді як метод RTK компенсує похибки майже повністю, за рахунок чого досягається точність до

сантиметрів, проте вимагає дорожчого обладнання, а також накладає обмеження на приймачі користувачів. Фазові вимірювання вимагають безперервного відстеження сигналу, що вимагаю постійної роботи GPS чіпсету у смартфоні, а також складніших розрахунків. Що значно підвищує використання акумуляторної батареї смартфона і тим самим зменшує зацікавленість користувачів в таких рішеннях. Окрім того, фазові вимірювання зазнають значного впливу в результаті багатопроменевого поширення сигналів.

У третьому розділі, для дослідження можливості використання диференційних поправок в ОС Android проаналізовано, можливості отримання даних, необхідних для розрахунку місцезнаходження користувача.. Додатки користувачів отримують доступ до даних GNSS за допомогою Location API. До API версії 23, цей доступ було обмежено, а саме надавалась лише наступна інформація: супутникова інформація GPS (C / No, азимут, висота, якщо конкретний супутник був використаний у PVT), речення NMEA та рішення з PVT правильним значенням часу. Починаючи з API 24 (Android 7) розробники мають доступ до такої «сирої» та обчисленої інформації GNSS через класи Android:

- GNSS Clock, який містить:
    - Час приймача (використовується для обчислення псевдовідстані);
    - Відхилення годиника
- Навігаційне повідомлення GNSS, яке містить:
    - Біти повідомлень навігації (усі сузір'я);
    - Статус навігаційного повідомлення.
- Вимірювання GNSS, яке містить:
    - Час прийому від супутника (використовується для обчислення псевдовідстані);
    - Код;
    - Фаза несучої.

Таким чином, в операційній системі Android існує весь необхідний функціонал для реалізації власного розрахунку місцезнаходження що робить можливим використання диференційних поправок для більш точного розрахунку

Наведені порівняння базової продуктивності GPS приймача смартфону з спецалізованим приймачем, а також наведені приклади використання розширених технік в Android смартфонах.

Реалізований Ntrip клієнт для Android додатку, який дозволяє отримувати диференціальні поправки через мережу Інтернет. А також, модуль розрахунку місцеположення з використанням диференційної GPS поправки

ПЕРЕЛІК ПОСИЛАНЬ

1. Parkinson, B., “A History of Satellite Navigation,” NAVIGATION: Journal of The Institute of Navigation, Vol. 42, No. 1, Spring 1995

2. Причины погрешностей в GPS мониторинге [Електронний ресурс] - режим доступу: http://gps-tracker.com.ua/prichiny-pogreshnostej-v-gps-monitoringe.html

3. GPS Space Segment [Електронний ресурс] - режим доступу: https://gssc.esa.int/navipedia/index.php/GPS_Space_Segment

4. U.S. Department of Defense, Global Positioning System Standard Positioning Service Performance Standard, Washington, D.C., October 2001

5. GPS Interface Specification (GPS-IS-200H) - режим доступу: https://www.gps.gov/technical/icwg/IS-GPS-200H.pdf

6. GPS System Segments, L. F. Wiederholt, E. D. Kaplan

7. Seeber, G., Satellite Geodesy: Foundations, Methods, and Applications, New York: Walter De Gruyter, 1993

8. Langley, R., “Time, Clocks, and GPS,”, GPS World Magazine, Advanstar Communications, November–December 1991

9. Interface Specification IS-GPS-200, september 2013. Режим доступу: http://www.gps.gov/technical/icwg/IS-GPS-200H.pdf

10. Van Diggelen, Frank Stephen Tromp. 2009. A-GPS: assisted GPS, GNSS, and SBAS. Boston: Artech House

11. National Imagery and Mapping Agency, Department of Defense, World Geodetic System 1984 (WGS 84): Its Definition and Relationships with Local Geodetic Systems, NIMA TR8350.2, 3rd ed., Bethesda, January 2000

12. Code Tracking and Pseudoranges, Inside GNSS, January/February 2012. Режим доступу: http://www.insidegnss.com/auto/IGM_janfeb12-Solutions.pdf

13. Principles of GPS, A Brief Primer on the Operation of the Global Positioning System, Carl Carter, Revision 1, February, 1997

14. ARINC Research Corporation, NAVSTAR GPS Space Segment/Navigation User Interfaces, Interface Specification, IS-GPS-200D (Public Release Version), ARINC Research Corporation, Fountain Valley, CA, 2004
15. Warren, D. L. M., and J. F. Raquet, “Broadcast vs. Precise GPS Ephemerides: A Historical Perspective,” Proc. of ION National Technical Meeting, San Diego, CA, January 28–30, 2002.
16. Taylor, J., and E. Barnes, “GPS Current Signal-in-Space Navigation Performance,” Proc. Of The Institute of Navigation National Technical Meeting, San Diego, CA, January 2005.
17. DILUTION OF PRECISION (DOP) CALCULATION FOR MISSION PLANNING PURPOSES, Ming Fatt Yuen March 2009
18. Interference, Multipath, and Scintillation, Phillip W. Ward, John W. Betz and Christopher J. Hegarty
19. Wikipedia, "List of Refractive Indices," [Електрониий ресурс] - режим доступу: http://en.wikipedia.org/wiki/List_of_indices_of_refraction
20. RTK GPS, Richard B. Langley, University of New Brunswick
21. Real-Time Kinematic Surveying Training Guide, Trimbe, Revision D September 2003
22. Nawzad Kameran Al-Salihi, Precise Positioning in Real Time using GPS RTK Signal for Visually Impaired People Navigation System, September 2010
23. Van Diggelen, Frank Stephen Tromp. 2009. A-GPS: assisted GPS, GNSS, and SBAS. Boston: Artech House.
24. GPS Measurement Tools, Google. Режим доступу: https://github.com/google/gps-measurement-tools
25. Cycle Slip [Електронний ресурс] — режим доступу: https://www.e-education.psu.edu/geog862/node/1728
26. Predicted GNSS Ephemeris [Електронний ресурс] — режим доступу: http://rxnetworks.com/location-io/predicted--gnss-ephemeris/
27. android.location [Електронний ресурс] — режим доступу: https://developer.android.com/reference/android/location/package-summary

28. European GNSS Agency, USING GNSS RAW MEASUREMENTS ON ANDROID DEVICES, 2017
29. Humphreys, T. E.; Murrian, M.; van Diggelen, F.; Podshivalov, S. & Kenneth M. Pesyna, J. On the Feasibility of cm-Accurate Positioning via a Smartphone's Antenna and GNSS Chip Proceedings of the 2016 IEEE/ION PLANS Conference, 2016
30. DGNSS Standards [Електронний ресурс] – режим доступу: https://gssc.esa.int/navipedia/index.php/DGNSS_Standards
31. Harald Gebhard, Georg Weber Networked Transport of RTCM via Internet Protocol, Ntrip Version 1.0
30. GPS Measurement Tools, Google [Електронний ресурс] — режим доступу: https://github.com/google/gps-measurement-tools
31. Roberts et al (2017) Predictive Intelligence for a Rail Traffic Management System, ION GNSS+ 2017
32. Leibovici et al (2017) Earth Observation for Citizen Science validation, or, Citizen Science for Earth Observation validation? Role of the Quality Assurance of Volunteered Observations
33. Location Score Index: Q2 2016, Mobile Advertising's Guide to Location Accuracy, Thinknear.
34. GNSS Market Report, Issue 5, GSA. [Електронний ресурс] — режим доступу: https://www.gsa.europa.eu/system/files/reports/gnss_mr_2017.pd
35. Faccio, Mara and McConnell, John J., Death by Pokémon GO: The Economic and Human Cost of Using Apps While Driving (February 2, 2018)
36. Sports Equipment: is GPS the best route to performance analysis? [Електронний ресурс] — режим доступу: https://www.peakendurancesport.com/endurance-products-and-technology/electronics-and-software/sports-equipment-gps-best-route-performance-analysis/
37. Colin O'Keeffe, Jon Nicholl, Janette Turner, Steve Goodacre, Role of ambulance response times in the survival of patients with out-of-hospital cardiac arrest

ДОДАТОК А. Обробка диференційної коригувальної інформації

```
Клас, що оброблює потік даних від кастеру
class RTCM23Client(
        private val connectionSettings: ConnectionSettings,
        private val decodeMap: HashMap<Int, Decode>
) : Runnable, RTCMClient {

    private val subject = PublishSubject.create<RTCMMessage>()
    private var nos: OutputStream? = null
    private var nis: InputStream? = null
    private lateinit var socket: Socket
    private val header = IntArray(11)
    private val correctHeader = intArrayOf(73, 67, 89, 32, 50, 48, 48, 32, 79, 75, 13)
    private var isRunning: Boolean = false
    private val allowedMessages = listOf(1)
    private val flip = intArrayOf(0, 32, 16, 48, 8, 40, 24, 56, 4, 36, 20, 52, 12, 44, 28, 60, 2, 34, 18,
50, 10, 42, 26, 58, 6, 38, 22, 54, 14, 46, 30, 62, 1, 33, 17, 49, 9, 41, 25, 57, 5, 37, 21, 53, 13, 45, 29,
61, 3, 35, 19, 51, 11, 43, 27, 59, 7, 39, 23, 55, 15, 47, 31, 63)
    private var buf = StringBuilder()
    private var scan = true
    private var lp = 0
    private var parseHeader = false
    private var headerIteration = 0
    private var rtcm23Header: BooleanArray = BooleanArray(60)
    private var messageType23 = 0
    private var messageLength23 = 0
    private var index = 0
    private var parseBody = false
    private lateinit var rtcm23Body: BooleanArray

    override fun run() {
        try {
            socket = Socket()
            Timber.i("Starting the job!")
            isRunning = connect(connectionSettings)
            nis?.let {
                checkHeader(it)
                readLoop(it)
            }

            if (nis == null) {
                subject.onError(Exception("Something went wrong"))
            } else if (!isRunning) {
//                subject.onComplete()
                finally()
            }

        }  catch (ex: SocketTimeoutException) {
            //TODO: fix this shit
            //subject.onError(ex)
        } catch (e: Exception) {
```

```
        //subject.onError(e)
    }
}

override fun connect(connectionSettings: ConnectionSettings) : Boolean {
    val address = InetSocketAddress(connectionSettings.server, connectionSettings.port)
    if (!socket.isConnected) {
        socket.connect(address)
    }
    nis = socket.getInputStream()
    nos = socket.getOutputStream()

    socket.soTimeout = 20 * 1000
    val requestmsg = """
                        >GET /${connectionSettings.mountpoint} HTTP/1.0
                        >User-Agent: NTRIP Android Client/20180729
                        >Accept: */*
                        >Connection: close
                        >Authorization: ${connectionSettings.authorization}
                        >
                    """.trimMargin(">")
    nos?.write(requestmsg.toByteArray())
    return true
}

override fun checkHeader(inputStream: InputStream) {
    // /First we read the HTTP header using a small state machine
    // The end of the header is received when a double end line
    // consisting
    // of a "new line" and a "carriage return" character has been received
    var state = 0
    // First the HTTP header type is read. It should be "ICY 200 OK"
    // But Since we receive integers not characters the correct header is
    // numeric: 73 = 'I', 67 = 'C' and so on.
    var hindex = 0
    // when 'running' is changed to false the loop is stopped

    while ((state == 0) and isRunning) {
        val c = inputStream.read()
        if (c < 0) {
            break
        }
        // break;
        // tester.write(c);
        state = transition(state, c)
        if (hindex > 10) {
            // The header should only be 11 characters long
            isRunning = false
        } else {
            header[hindex] = c
            hindex++
        }
```

```
    }

    var i = 0
    while ((i < 11) and isRunning) {
        if (header[i] != correctHeader[i]) {
            isRunning = false
        }
        i++
    }
}

override fun readLoop(inputStream: InputStream) {
    var c: Int

    while (isRunning) {
        c = inputStream.read()
        if (c < 0) {
            isRunning = false
        } else {
            parseMessage(c)
        }
    }
}

private fun parseMessage(inputByte: Int) {
    buf.append(inputByte.toChar())
    if (buf.length >= 5) {
        val data = buf.toString().toByteArray(charset("UTF8"))
        var idx = 0
        if (scan) {
            for (tmp in 0..3) {
                val word = getWord(tmp, data, idx)
                if (word and 0x40000000 != 0) {
                    get23Message(word)
                    idx += 5
                    lp = word and 0x03
                    scan = false
                    break
                }
            }
            if (scan) {
                idx++
            }
        } else {
            val word = getWord(lp, data, idx)
            if (word and 0x40000000 != 0) {
                get23Message(word)
                idx += 5
                lp = word and 0x03
            } else {
                scan = true
            }
```

```
        }
        buf = StringBuilder(buf.substring(idx))
    }
}

private fun b(word: Int, b: Int): Int {
    return word shr 29 - b and 1
}

fun getWord(lp: Int, data: ByteArray, idx: Int): Int {
    val p29 = lp shr 1 and 1
    val p30 = lp and 1
    var w = 0
    for (ii in 0..4) {
        w = (w shl 6) + flip[(data[idx + ii] and 0x3F).toInt()]
    }
    w = if (p30 != 0) w xor 0x3FFFFFC0 else w
    val d25 = p29 xor b(w, 0) xor b(w, 1) xor b(w, 2) xor
            b(w, 4) xor b(w, 5) xor b(w, 9) xor b(w, 10) xor
            b(w, 11) xor b(w, 12) xor b(w, 13) xor b(w, 16) xor
            b(w, 17) xor b(w, 19) xor b(w, 22)
    val d26 = p30 xor b(w, 1) xor b(w, 2) xor b(w, 3) xor
            b(w, 5) xor b(w, 6) xor b(w, 10) xor b(w, 11) xor
            b(w, 12) xor b(w, 13) xor b(w, 14) xor b(w, 17) xor
            b(w, 18) xor b(w, 20) xor b(w, 23)
    val d27 = p29 xor b(w, 0) xor b(w, 2) xor b(w, 3) xor
            b(w, 4) xor b(w, 6) xor b(w, 7) xor b(w, 11) xor
            b(w, 12) xor b(w, 13) xor b(w, 14) xor b(w, 15) xor
            b(w, 18) xor b(w, 19) xor b(w, 21)
    val d28 = p30 xor b(w, 1) xor b(w, 3) xor b(w, 4) xor
            b(w, 5) xor b(w, 7) xor b(w, 8) xor b(w, 12) xor
            b(w, 13) xor b(w, 14) xor b(w, 15) xor b(w, 16) xor
            b(w, 19) xor b(w, 20) xor b(w, 22)
    val d29 = p30 xor b(w, 0) xor b(w, 2) xor b(w, 4) xor
            b(w, 5) xor b(w, 6) xor b(w, 8) xor b(w, 9) xor
            b(w, 13) xor b(w, 14) xor b(w, 15) xor b(w, 16) xor
            b(w, 17) xor b(w, 20) xor b(w, 21) xor b(w, 23)
    val d30 = p29 xor b(w, 2) xor b(w, 4) xor b(w, 5) xor
            b(w, 7) xor b(w, 8) xor b(w, 9) xor b(w, 10) xor
            b(w, 12) xor b(w, 14) xor b(w, 18) xor b(w, 21) xor
            b(w, 22) xor b(w, 23)
    val parity = (((((d25 shl 1) + d26 shl 1) + d27 shl 1) + d28 shl 1) + d29 shl 1) + d30
    if (w and 0x3F == parity)
        w = w or 0x40000000
    return w
}

fun get23Message(word: Int) {
    val bits = Bits.rollIntToBits(word)
    val preamble = Bits.bitsToUInt(
            Bits.subset(
                    bits,
```

```
                0,
                8
        )
).toInt()
if (preamble == 102) {
    parseHeader = true
    headerIteration = 0
    rtcm23Header = BooleanArray(60)
}

if (parseHeader) {
    if (headerIteration < 2) {
        for (i in 0..29) {
            rtcm23Header[30 * headerIteration + i] = bits[i]
        }
        headerIteration++
    } else {
        messageType23 = Bits.bitsToUInt(
                Bits.subset(
                        rtcm23Header,
                        8,
                        6
                )
        ).toInt()

        messageLength23 = Bits.bitsToUInt(
                Bits.subset(
                        rtcm23Header,
                        45,
                        5
                )
        ).toInt()
        parseHeader = false
        parseBody = true
        rtcm23Body = BooleanArray(messageLength23 * 30)
        index = 0
    }

} else if (parseBody) {
    if (index < messageLength23) {
        for (i in 0..29) {
            rtcm23Body[30 * index + i] = bits[i]
        }
        index++
    } else {
        val decoder = decodeMap[messageType23]
        val msg = decoder?.decode(booleanArrayOf(*rtcm23Header, *rtcm23Body), 0)
        msg?.let {
            subject.onNext(it)
        }
        parseBody = false
    }
```

```
    }
}

private fun transition(oldState: Int, input: Int): Int {
    var state = oldState
    when (state) {
        0 -> {
            if (input == 13)
                state = 1
        }

        1 -> {
            if (input == 13)
                state = 2
        }
        2 -> {
            if (input == 10)
                state = 5
            else
                state = 1
        }
        3 -> {
            if (input == 13)
                state = 4
            else
                state = 1
        }
        4 -> {
            if (input == 10)
                state = 5
            else
                state = 1
        }
    }

    return state
}

override fun getObservable(): Observable<RTCMMessage> {
    return subject.hide()
}

@Synchronized
override fun stop() {
    isRunning = false
}

override fun finally() {
    try {
        nis?.close()
        nos?.close()
        socket.close()
```

```
        } catch (e: IOException) {
            e.printStackTrace()
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    @Synchronized
    override fun isActive(): Boolean = isRunning
}

Клас, що декодує навігаційне повідомлення
class Decode1Msg : Decode {

    override fun decode(bits: BooleanArray, week: Int): RTCMMessage {
        var i = 48
        var fact: Int
        var udre: Int
        var prn: Int
        var iod: Int
        var prc: Double
        var rrc: Double
        var sat = 0
        val currentTime = Time(
                System.currentTimeMillis()
        )
        val observations = arrayListOf<Dgps>()

        while (i+40 <= bits.size) {
            fact = Bits.bitsToUInt(
                    Bits.subset(
                            bits,
                            i,
                            1
                    )
            ).toInt()
            i += 1
            udre = Bits.bitsToUInt(
                    Bits.subset(
                            bits,
                            i,
                            2
                    )
            ).toInt()
            i += 2
            prn = Bits.bitsToUInt(
                    Bits.subset(
                            bits,
                            i,
                            5
                    )
            ).toInt()
```

```
i += 5
prc = Bits.byteToIEEE754Double(
        Bits.tobytes(
                Bits.subset(
                        bits,
                        i,
                        16
                )
        )
)
i += 16
rrc = Bits.byteToIEEE754Double(
        Bits.tobytes(
                Bits.subset(
                        bits,
                        i,
                        8
                )
        )
)
i += 8
iod = Bits.byteToInt(
        Bits.tobytes(
                Bits.subset(
                        bits,
                        i,
                        8
                )
        )
)
i += 8

if (prn == 0) prn = 32
if ((prc.toLong()==0x80000000) or (rrc.toLong()==0xFFFF8000)) {
    continue
}

if (fact == 1) {
    prc = prc * 0.32
    rrc = rrc * 0.032
} else {
    prc = prc * 0.02
    rrc = rrc * 0.002
}
val dgps = Dgps(
        satId = satNo(prn),
        time = currentTime,
        prc = prc,
        rrc = rrc,
        iod = iod,
        udre = udre
        )
```

```
            observations.add(dgps)
            sat++
        }
        return RTCMMessage(MsgType.M1, dgps = observations)
    }

    fun satNo(prn: Int): Int {
        if ((prn < Constants.MINPRNGPS) or (prn > Constants.MAXPRNGPS)) {
            return 0
        } else {
            return prn - Constants.MINPRNGPS + 1
        }
    }

}Клас, що виконує підключення до кастеру
class NetworkClient(
        val server: String,
        val port: Int
) : NTRIPClient {
    val socket = Socket()

    override fun getNTRIPData(
            user: LoginDTO,
            mountpoint: String
    ): Observable<List<ControlStationDTO>> {
        return Observable
                .just(InetSocketAddress(server, port))
                .map {
                    if (!socket.isConnected) {
                        socket.connect(it)
                    }
                    socket.isConnected
                }
                .map {
                    if (it) {
                        socket.soTimeout = 20 * 1000
                        val requestmsg = """
                            >GET /$mountpoint HTTP/1.0
                            >User-Agent: NTRIP Android Client/20180729
                            >Accept: */*
                            >Connection: close
                            >Authorization: ${ToBase64("${user.user}:${user.password}")}
                            >
                        """.trimMargin(">")

                        socket.getOutputStream().write(requestmsg.toByteArray())
                        socket.getInputStream()
                    } else {
                        throw CasterConnectionException("Socket not connected")
                    }
                }
                .flatMap {
```

```
                Utils.observableFromStream(it, 4096)
            }.compose(
                    SourceTableTransformer()
            )
            .doOnDispose {
                try {
                    socket.getOutputStream().close()
                    socket.close()
                } catch (ex: IOException) {
                    throw ex
                }
            }
}

private class SourceTableTransformer : ObservableTransformer<ByteArray, List<ControlStationDTO>> {
    override fun apply(upstream: Observable<ByteArray>): ObservableSource<List<ControlStationDTO>> {
        return upstream.reduce("") { accum, arr ->
            val str = accum + String(arr)
            if (str.indexOf("401 Unauthorized") > 1) {
                throw CasterConnectionException("Bad username or password.")
            } else if (str.startsWith("SOURCETABLE 200 OK")){
                str
            } else {
                throw CasterConnectionException("Unrecognized server response:")
            }

        }
                .observeOn(Schedulers.computation())
                .map { s ->
                    s.split("\r\n").map { it.trim() }
                }
                .flatMapObservable {
                    Observable.fromIterable(it)
                }
                .filter{
                    it.startsWith("STR") and (it.indexOf("Zone_") < 0)
                }
                .map {
                    it.split(";").map { it.trim() }
                }
                .reduce(mutableListOf<ControlStationDTO>()
                ) { accum, strList ->
                    accum.add(
                            ControlStationDTO(
                                    name = strList[1],
                                    zone = strList[2],
                                    protocol = strList[3],
                                    navType = strList[6],
                                    country = strList[8],
                                    latitude = strList[9].toDouble(),
                                    longitude = strList[10].toDouble(),
                                    device = strList[13],
```

```
                                    accuracy = strList[18].substring(9)
                                )
                            )
                            accum
                        }
                        .map {
                            it.toList()
                        }.toObservable()
            }
        }
    }
```

ДОДАТОК Б. Модуль підрахунку місцезнаходження

```
Клас, що описує розрахунок позиції користувача за методом найменших квадратів
class UserPositionVelocityWeightedLeastSquare {
  private static final double SPEED_OF_LIGHT_MPS = 299792458.0;
  private static final int SECONDS_IN_WEEK = 604800;
  private static final double LEAST_SQUARE_TOLERANCE_METERS = 4.0e-8;
  /** Position correction threshold below which atmospheric correction will be applied */
  private static final double ATMPOSPHERIC_CORRECTIONS_THRESHOLD_METERS = 1000.0;
  private static final int MINIMUM_NUMER_OF_SATELLITES = 4;
  private static final double RESIDUAL_TO_REPEAT_LEAST_SQUARE_METERS = 20.0;
  private static final int MAXIMUM_NUMBER_OF_LEAST_SQUARE_ITERATIONS = 100;
  /** GPS C/A code chip width Tc = 1 microseconds */
  private static final double GPS_CHIP_WIDTH_T_C_SEC = 1.0e-6;
  /** Narrow correlator with spacing d = 0.1 chip */
  private static final double GPS_CORRELATOR_SPACING_IN_CHIPS = 0.1;
  /** Average time of DLL correlator T of 20 milliseconds */
  private static final double GPS_DLL_AVERAGING_TIME_SEC = 20.0e-3;
  /** Average signal travel time from GPS satellite and earth */
  private static final double AVERAGE_TRAVEL_TIME_SECONDS = 70.0e-3;
  private static final double SECONDS_PER_NANO = 1.0e-9;
  private static final double DOUBLE_ROUND_OFF_TOLERANCE = 0.0000000001;

  private final PseudorangeSmoother pseudorangeSmoother;
  private double geoidHeightMeters;
  private ElevationApiHelper elevationApiHelper;
  private boolean calculateGeoidMeters = true;
  private RealMatrix geometryMatrix;
  private double[] truthLocationForCorrectedResidualComputationEcef = null;

  /** Constructor */
  public UserPositionVelocityWeightedLeastSquare(PseudorangeSmoother pseudorangeSmoother) {
    this.pseudorangeSmoother = pseudorangeSmoother;
  }

  /** Constructor with Google Elevation API Key */
  public UserPositionVelocityWeightedLeastSquare(PseudorangeSmoother pseudorangeSmoother,
      String elevationApiKey){
    this.pseudorangeSmoother = pseudorangeSmoother;
    this.elevationApiHelper = new ElevationApiHelper();
  }

  /**
   * Sets the reference ground truth for pseudornage residual correction calculation. If no ground
   * truth is set, no corrected pesudorange residual will be calculated.
   */
  public void setTruthLocationForCorrectedResidualComputationEcef
  (double[] groundTruthForResidualCorrectionEcef) {
    this.truthLocationForCorrectedResidualComputationEcef = groundTruthForResidualCorrectionEcef;
  }

  /**
```

```
 * Least square solution to calculate the user position given the navigation message, pseudorange
 * and accumulated delta range measurements. Also calculates user velocity non-iteratively from
 * Least square position solution.
 *
 * <p>The method fills the user position and velocity in ECEF coordinates and receiver clock
 * offset in meters and clock offset rate in meters per second.
 *
 * <p>One can choose between no smoothing, using the carrier phase measurements (accumulated delta
 * range) or the doppler measurements (pseudorange rate) for smoothing the pseudorange. The
 * smoothing is applied only if time has changed below a specific threshold since last invocation.
 *
 * <p>Source for least squares:
 *
 * <ul>
 *   <li>http://www.u-blox.com/images/downloads/Product_Docs/GPS_Compendium%28GPS-X-02007%29.pdf
 *       page 81 - 85
 *   <li>Parkinson, B.W., Spilker Jr., J.J.: 'Global positioning system: theory and applications'
 *       page 412 - 414
 * </ul>
 *
 * <p>Sources for smoothing pseudorange with carrier phase measurements:
 *
 * <ul>
 *   <li>Satellite Communications and Navigation Systems book, page 424,
 *   <li>Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, page 388,
 *       389.
 * </ul>
 *
 * <p>The function does not modify the smoothed measurement list {@code
 * immutableSmoothedSatellitesToReceiverMeasurements}
 *
 * @param navMessageProto parameters of the navigation message
 * @param usefulSatellitesToReceiverMeasurements Map of useful satellite PRN to {@link
 *     GpsMeasurementWithRangeAndUncertainty} containing receiver measurements for computing the
 *     position solution.
 * @param receiverGPSTowAtReceptionSeconds Receiver estimate of GPS time of week (seconds)
 * @param receiverGPSWeek Receiver estimate of GPS week (0-1024+)
 * @param dayOfYear1To366 The day of the year between 1 and 366
 * @param positionVelocitySolutionECEF Solution array of the following format:
 *        [0-2] xyz solution of user.
 *        [3] clock bias of user.
 *        [4-6] velocity of user.
 *        [7] clock bias rate of user.
 * @param positionVelocityUncertaintyEnu Uncertainty of calculated position and velocity solution
 *     in meters and mps local ENU system. Array has the following format:
 *     [0-2] Enu uncertainty of position solution in meters
 *     [3-5] Enu uncertainty of velocity solution in meters per second.
 * @param pseudorangeResidualMeters The pseudorange residual corrected by subtracting expected
 *     psudorange calculated with the use clock bias of the highest elevation satellites.
 */
public void calculateUserPositionVelocityLeastSquare(
    GpsNavMessageProto navMessageProto,
```

```
  List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements,
  double receiverGPSTowAtReceptionSeconds,
  int receiverGPSWeek,
  int dayOfYear1To366,
  double[] positionVelocitySolutionECEF,
  double[] positionVelocityUncertaintyEnu,
  double[] pseudorangeResidualMeters,
  List<RawDgps> dgps)
  throws Exception {

// Use PseudorangeSmoother to smooth the pseudorange according to: Satellite Communications and
// Navigation Systems book, page 424 and Principles of GNSS, Inertial, and Multisensor
// Integrated Navigation Systems, page 388, 389.
double[] deltaPositionMeters;
pseudorangeSmoother.setSmoothing(dgps);
List<GpsMeasurementWithRangeAndUncertainty> immutableSmoothedSatellitesToReceiverMeasurements =
    pseudorangeSmoother.updatePseudorangeSmoothingResult(
        Collections.unmodifiableList(usefulSatellitesToReceiverMeasurements));
List<GpsMeasurementWithRangeAndUncertainty> mutableSmoothedSatellitesToReceiverMeasurements =
    Lists.newArrayList(immutableSmoothedSatellitesToReceiverMeasurements);
int numberOfUsefulSatellites =
    getNumberOfUsefulSatellites(mutableSmoothedSatellitesToReceiverMeasurements);
// Least square position solution is supported only if 4 or more satellites visible
Preconditions.checkArgument(numberOfUsefulSatellites >= MINIMUM_NUMER_OF_SATELLITES,
    "At least 4 satellites have to be visible... Only 3D mode is supported...");
boolean repeatLeastSquare = false;
SatellitesPositionPseudorangesResidualAndCovarianceMatrix satPosPseudorangeResidualAndWeight;

boolean isFirstWLS = true;

do {
  // Calculate satellites' positions, measurement residuals per visible satellite and
  // weight matrix for the iterative least square
  boolean doAtmosphericCorrections = false;
  satPosPseudorangeResidualAndWeight =
      calculateSatPosAndPseudorangeResidual(
          navMessageProto,
          mutableSmoothedSatellitesToReceiverMeasurements,
          receiverGPSTowAtReceptionSeconds,
          receiverGPSWeek,
          dayOfYear1To366,
          positionVelocitySolutionECEF,
          doAtmosphericCorrections);

  // Calculate the geometry matrix according to "Global Positioning System: Theory and
  // Applications", Parkinson and Spilker page 413
  RealMatrix covarianceMatrixM2 =
      new Array2DRowRealMatrix(satPosPseudorangeResidualAndWeight.covarianceMatrixMetersSquare);
  geometryMatrix = new Array2DRowRealMatrix(calculateGeometryMatrix(
      satPosPseudorangeResidualAndWeight.satellitesPositionsMeters,
      positionVelocitySolutionECEF));
  RealMatrix weightedGeometryMatrix;
```

```
RealMatrix weightMatrixMetersMinus2 = null;
// Apply weighted least square only if the covariance matrix is not singular (has a non-zero
// determinant), otherwise apply ordinary least square. The reason is to ignore reported
// signal to noise ratios by the receiver that can lead to such singularities
LUDecomposition ludCovMatrixM2 = new LUDecomposition(covarianceMatrixM2);
double det = ludCovMatrixM2.getDeterminant();

if (det <= DOUBLE_ROUND_OFF_TOLERANCE) {
  // Do not weight the geometry matrix if covariance matrix is singular.
  weightedGeometryMatrix = geometryMatrix;
} else {
  weightMatrixMetersMinus2 = ludCovMatrixM2.getSolver().getInverse();
  RealMatrix hMatrix =
      calculateHMatrix(weightMatrixMetersMinus2, geometryMatrix);
  weightedGeometryMatrix = hMatrix.multiply(geometryMatrix.transpose())
      .multiply(weightMatrixMetersMinus2);
}

// Equation 9 page 413 from "Global Positioning System: Theory and Applicaitons", Parkinson
// and Spilker
deltaPositionMeters =
    GpsMathOperations.matrixByColVectMultiplication(weightedGeometryMatrix.getData(),
    satPosPseudorangeResidualAndWeight.pseudorangeResidualsMeters);

// Apply corrections to the position estimate
positionVelocitySolutionECEF[0] += deltaPositionMeters[0];
positionVelocitySolutionECEF[1] += deltaPositionMeters[1];
positionVelocitySolutionECEF[2] += deltaPositionMeters[2];
positionVelocitySolutionECEF[3] += deltaPositionMeters[3];
// Iterate applying corrections to the position solution until correction is below threshold
satPosPseudorangeResidualAndWeight =
    applyWeightedLeastSquare(
        navMessageProto,
        mutableSmoothedSatellitesToReceiverMeasurements,
        receiverGPSTowAtReceptionSeconds,
        receiverGPSWeek,
        dayOfYear1To366,
        positionVelocitySolutionECEF,
        deltaPositionMeters,
        doAtmosphericCorrections,
        satPosPseudorangeResidualAndWeight,
        weightMatrixMetersMinus2);

// We use the first WLS iteration results and correct them based on the ground truth position
// and using a clock error computed from high elevation satellites. The first iteration is
// used before satellite with high residuals being removed.
if (isFirstWLS && truthLocationForCorrectedResidualComputationEcef != null) {
  // Snapshot the information needed before high residual satellites are removed
  System.arraycopy(
      ResidualCorrectionCalculator.calculateCorrectedResiduals(
          satPosPseudorangeResidualAndWeight,
          positionVelocitySolutionECEF.clone(),
```

```
            truthLocationForCorrectedResidualComputationEcef),
        0 /*source starting pos*/,
        pseudorangeResidualMeters,
        0 /*destination starting pos*/,
        GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES /*length of elements*/);
    isFirstWLS = false;
  }
  repeatLeastSquare = false;
  int satsWithResidualBelowThreshold =
      satPosPseudorangeResidualAndWeight.pseudorangeResidualsMeters.length;
  // remove satellites that have residuals above RESIDUAL_TO_REPEAT_LEAST_SQUARE_METERS as they
  // worsen the position solution accuracy. If any satellite is removed, repeat the least square
  repeatLeastSquare =
      removeHighResidualSats(
          mutableSmoothedSatellitesToReceiverMeasurements,
          repeatLeastSquare,
          satPosPseudorangeResidualAndWeight,
          satsWithResidualBelowThreshold);

} while (repeatLeastSquare);
calculateGeoidMeters = false;

// The computed ECEF position will be used next to compute the user velocity.
// we calculate and fill in the user velocity solutions based on following equation:
// Weight Matrix * GeometryMatrix * User Velocity Vector
// = Weight Matrix * deltaPseudoRangeRateWeightedMps
// Reference: Pratap Misra and Per Enge
// "Global Positioning System: Signals, Measurements, and Performance" Page 218.

// Get the number of satellite used in Geometry Matrix
numberOfUsefulSatellites = geometryMatrix.getRowDimension();

RealMatrix rangeRateMps = new Array2DRowRealMatrix(numberOfUsefulSatellites, 1);
RealMatrix deltaPseudoRangeRateMps =
    new Array2DRowRealMatrix(numberOfUsefulSatellites, 1);
RealMatrix pseudorangeRateWeight
    = new Array2DRowRealMatrix(numberOfUsefulSatellites, numberOfUsefulSatellites);

// Correct the receiver time of week with the estimated receiver clock bias
receiverGPSTowAtReceptionSeconds =
    receiverGPSTowAtReceptionSeconds - positionVelocitySolutionECEF[3] / SPEED_OF_LIGHT_MPS;

int measurementCount = 0;

// Calculate range rates
for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
  if (mutableSmoothedSatellitesToReceiverMeasurements.get(i) != null) {
    GpsEphemerisProto ephemeridesProto = getEphemerisForSatellite(navMessageProto, i + 1);

    double pseudorangeMeasurementMeters =
        mutableSmoothedSatellitesToReceiverMeasurements.get(i).pseudorangeMeters;
    GpsTimeOfWeekAndWeekNumber correctedTowAndWeek =
```

```
        calculateCorrectedTransmitTowAndWeek(ephemeridesProto, receiverGPSTowAtReceptionSeconds,
            receiverGPSWeek, pseudorangeMeasurementMeters);

    // Calculate satellite velocity
    PositionAndVelocity satPosECEFMetersVelocityMPS = SatellitePositionCalculator
        .calculateSatellitePositionAndVelocityFromEphemeris(
            ephemeridesProto,
            correctedTowAndWeek.gpsTimeOfWeekSeconds,
            correctedTowAndWeek.weekNumber,
            positionVelocitySolutionECEF[0],
            positionVelocitySolutionECEF[1],
            positionVelocitySolutionECEF[2]);

    // Calculate satellite clock error rate
    double satelliteClockErrorRateMps = SatelliteClockCorrectionCalculator.
        calculateSatClockCorrErrorRate(
            ephemeridesProto,
            correctedTowAndWeek.gpsTimeOfWeekSeconds,
            correctedTowAndWeek.weekNumber);

    // Fill in range rates. range rate = satellite velocity (dot product) line-of-sight vector
    rangeRateMps.setEntry(measurementCount, 0,  -1 * (
        satPosECEFMetersVelocityMPS.velocityXMetersPerSec
            * geometryMatrix.getEntry(measurementCount, 0)
            + satPosECEFMetersVelocityMPS.velocityYMetersPerSec
            * geometryMatrix.getEntry(measurementCount, 1)
            + satPosECEFMetersVelocityMPS.velocityZMetersPerSec
            * geometryMatrix.getEntry(measurementCount, 2)));

    deltaPseudoRangeRateMps.setEntry(measurementCount, 0,
        mutableSmoothedSatellitesToReceiverMeasurements.get(i).pseudorangeRateMps
            - rangeRateMps.getEntry(measurementCount, 0) + satelliteClockErrorRateMps
            - positionVelocitySolutionECEF[7]);

    // Calculate the velocity weight matrix by using 1 / square(Pseudorangerate Uncertainty)
    // along the diagonal
    pseudorangeRateWeight.setEntry(measurementCount, measurementCount,
        1 / (mutableSmoothedSatellitesToReceiverMeasurements
            .get(i).pseudorangeRateUncertaintyMps
            * mutableSmoothedSatellitesToReceiverMeasurements
            .get(i).pseudorangeRateUncertaintyMps));
    measurementCount++;
  }
}

RealMatrix weightedGeoMatrix = pseudorangeRateWeight.multiply(geometryMatrix);
RealMatrix deltaPseudoRangeRateWeightedMps =
    pseudorangeRateWeight.multiply(deltaPseudoRangeRateMps);
QRDecomposition qrdWeightedGeoMatrix = new QRDecomposition(weightedGeoMatrix);
RealMatrix velocityMps
    = qrdWeightedGeoMatrix.getSolver().solve(deltaPseudoRangeRateWeightedMps);
positionVelocitySolutionECEF[4] = velocityMps.getEntry(0, 0);
```

```
  positionVelocitySolutionECEF[5] = velocityMps.getEntry(1, 0);
  positionVelocitySolutionECEF[6] = velocityMps.getEntry(2, 0);
  positionVelocitySolutionECEF[7] = velocityMps.getEntry(3, 0);

  RealMatrix pseudorangeWeight
      = new LUDecomposition(
          new Array2DRowRealMatrix(satPosPseudorangeResidualAndWeight.covarianceMatrixMetersSquare
          )
  ).getSolver().getInverse();

  // Calculate and store the uncertainties of position and velocity in local ENU system in meters
  // and meters per second.
  double[] pvUncertainty =
      calculatePositionVelocityUncertaintyEnu(pseudorangeRateWeight, pseudorangeWeight,
          positionVelocitySolutionECEF);
  System.arraycopy(pvUncertainty,
      0 /*source starting pos*/,
      positionVelocityUncertaintyEnu,
      0 /*destination starting pos*/,
      6 /*length of elements*/);
}

/**
 * Calculates the position uncertainty in meters and the velocity uncertainty
 * in meters per second solution in local ENU system.
 *
 * <p> Reference: Global Positioning System: Signals, Measurements, and Performance
 * by Pratap Misra, Per Enge, Page 206 - 209.
 *
 * @param velocityWeightMatrix the velocity weight matrix
 * @param positionWeightMatrix the position weight matrix
 * @param positionVelocitySolution the position and velocity solution in ECEF
 * @return an array containing the position and velocity uncertainties in ENU coordinate system.
 *         [0-2] Enu uncertainty of position solution in meters.
 *         [3-5] Enu uncertainty of velocity solution in meters per second.
 */
public double[] calculatePositionVelocityUncertaintyEnu(
    RealMatrix velocityWeightMatrix, RealMatrix positionWeightMatrix,
    double[] positionVelocitySolution){

  if (geometryMatrix == null){
    return null;
  }

  RealMatrix velocityH = calculateHMatrix(velocityWeightMatrix, geometryMatrix);
  RealMatrix positionH = calculateHMatrix(positionWeightMatrix, geometryMatrix);

  // Calculate the rotation Matrix to convert to local ENU system.
  RealMatrix rotationMatrix = new Array2DRowRealMatrix(4, 4);
  GeodeticLlaValues llaValues = Ecef2LlaConverter.convertECEFToLLACloseForm
      (positionVelocitySolution[0], positionVelocitySolution[1], positionVelocitySolution[2]);
  rotationMatrix.setSubMatrix(
```

```
      Ecef2EnuConverter.getRotationMatrix(llaValues.longitudeRadians,
          llaValues.latitudeRadians).getData(), 0, 0);
  rotationMatrix.setEntry(3, 3, 1);

  // Convert to local ENU by pre-multiply rotation matrix and multiply rotation matrix transposed
  velocityH = rotationMatrix.multiply(velocityH).multiply(rotationMatrix.transpose());
  positionH = rotationMatrix.multiply(positionH).multiply(rotationMatrix.transpose());

  // Return the square root of diagonal entries
  return new double[] {
      Math.sqrt(positionH.getEntry(0, 0)), Math.sqrt(positionH.getEntry(1, 1)),
      Math.sqrt(positionH.getEntry(2, 2)), Math.sqrt(velocityH.getEntry(0, 0)),
      Math.sqrt(velocityH.getEntry(1, 1)), Math.sqrt(velocityH.getEntry(2, 2))};
}

/**
 * Calculates the measurement connection matrix H as a function of weightMatrix and
 * geometryMatrix.
 *
 * <p> H = (geometryMatrixTransposed * Weight * geometryMatrix) ^ -1
 *
 * <p> Reference: Global Positioning System: Signals, Measurements, and Performance, P207
 * @param weightMatrix Weights for computing H Matrix
 * @return H Matrix
 */
private RealMatrix calculateHMatrix
    (RealMatrix weightMatrix, RealMatrix geometryMatrix){

  RealMatrix tempH = geometryMatrix.transpose().multiply(weightMatrix).multiply(geometryMatrix);
  return new LUDecomposition(tempH).getSolver().getInverse();
}

/**
 * Applies weighted least square iterations and corrects to the position solution until correction
 * is below threshold. An exception is thrown if the maximum number of iterations:
 * {@value #MAXIMUM_NUMBER_OF_LEAST_SQUARE_ITERATIONS} is reached without convergence.
 */
private SatellitesPositionPseudorangesResidualAndCovarianceMatrix applyWeightedLeastSquare(
    GpsNavMessageProto navMessageProto,
    List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements,
    double receiverGPSTowAtReceptionSeconds,
    int receiverGPSWeek,
    int dayOfYear1To366,
    double[] positionSolutionECEF,
    double[] deltaPositionMeters,
    boolean doAtmosphericCorrections,
    SatellitesPositionPseudorangesResidualAndCovarianceMatrix satPosPseudorangeResidualAndWeight,
    RealMatrix weightMatrixMetersMinus2)
    throws Exception {
  RealMatrix weightedGeometryMatrix;
  int numberOfIterations = 0;
```

```
while ((Math.abs(deltaPositionMeters[0]) + Math.abs(deltaPositionMeters[1])
    + Math.abs(deltaPositionMeters[2])) >= LEAST_SQUARE_TOLERANCE_METERS) {
  // Apply ionospheric and tropospheric corrections only if the applied correction to
  // position is below a specific threshold
  if ((Math.abs(deltaPositionMeters[0]) + Math.abs(deltaPositionMeters[1])
      + Math.abs(deltaPositionMeters[2])) < ATMPOSPHERIC_CORRECTIONS_THRESHOLD_METERS) {
    doAtmosphericCorrections = true;
  }
  // Calculate satellites' positions, measurement residual per visible satellite and
  // weight matrix for the iterative least square
  satPosPseudorangeResidualAndWeight =
      calculateSatPosAndPseudorangeResidual(
          navMessageProto,
          usefulSatellitesToReceiverMeasurements,
          receiverGPSTowAtReceptionSeconds,
          receiverGPSWeek,
          dayOfYear1To366,
          positionSolutionECEF,
          doAtmosphericCorrections);

  // Calculate the geometry matrix according to "Global Positioning System: Theory and
  // Applications", Parkinson and Spilker page 413
  geometryMatrix = new Array2DRowRealMatrix(calculateGeometryMatrix(
      satPosPseudorangeResidualAndWeight.satellitesPositionsMeters, positionSolutionECEF));
  // Apply weighted least square only if the covariance matrix is
  // not singular (has a non-zero determinant), otherwise apply ordinary least square.
  // The reason is to ignore reported signal to noise ratios by the receiver that can
  // lead to such singularities
  if (weightMatrixMetersMinus2 == null) {
    weightedGeometryMatrix = geometryMatrix;
  } else {
    RealMatrix hMatrix =
        calculateHMatrix(weightMatrixMetersMinus2, geometryMatrix);
    weightedGeometryMatrix = hMatrix.multiply(geometryMatrix.transpose())
        .multiply(weightMatrixMetersMinus2);
  }

  // Equation 9 page 413 from "Global Positioning System: Theory and Applicaitons",
  // Parkinson and Spilker
  deltaPositionMeters =
      GpsMathOperations.matrixByColVectMultiplication(
          weightedGeometryMatrix.getData(),
          satPosPseudorangeResidualAndWeight.pseudorangeResidualsMeters);

  // Apply corrections to the position estimate
  positionSolutionECEF[0] += deltaPositionMeters[0];
  positionSolutionECEF[1] += deltaPositionMeters[1];
  positionSolutionECEF[2] += deltaPositionMeters[2];
  positionSolutionECEF[3] += deltaPositionMeters[3];
  numberOfIterations++;
  Preconditions.checkArgument(numberOfIterations <= MAXIMUM_NUMBER_OF_LEAST_SQUARE_ITERATIONS,
      "Maximum number of least square iterations reached without convergance...");
```

```
  }
  return satPosPseudorangeResidualAndWeight;
}

/**
 * Removes satellites that have residuals above {@value #RESIDUAL_TO_REPEAT_LEAST_SQUARE_METERS}
 * from the {@code usefulSatellitesToReceiverMeasurements} list. Returns true if any satellite is
 * removed.
 */
private boolean removeHighResidualSats(
    List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements,
    boolean repeatLeastSquare,
    SatellitesPositionPseudorangesResidualAndCovarianceMatrix satPosPseudorangeResidualAndWeight,
    int satsWithResidualBelowThreshold) {

  for (int i = 0; i < satPosPseudorangeResidualAndWeight.pseudorangeResidualsMeters.length; i++) {
    if (satsWithResidualBelowThreshold > MINIMUM_NUMER_OF_SATELLITES) {
      if (Math.abs(satPosPseudorangeResidualAndWeight.pseudorangeResidualsMeters[i])
          > RESIDUAL_TO_REPEAT_LEAST_SQUARE_METERS) {
        int prn = satPosPseudorangeResidualAndWeight.satellitePRNs[i];
        usefulSatellitesToReceiverMeasurements.set(prn - 1, null);
        satsWithResidualBelowThreshold--;
        repeatLeastSquare = true;
      }
    }
  }
  return repeatLeastSquare;
}

/**
 * Calculates position of all visible satellites and pseudorange measurement residual
 * (difference of measured to predicted pseudoranges) needed for the least square computation. The
 * result is stored in an instance of {@link
 * SatellitesPositionPseudorangesResidualAndCovarianceMatrix}
 *
 * @param navMeassageProto parameters of the navigation message
 * @param usefulSatellitesToReceiverMeasurements Map of useful satellite PRN to {@link
 *     GpsMeasurementWithRangeAndUncertainty} containing receiver measurements for computing the
 *     position solution
 * @param receiverGPSTowAtReceptionSeconds Receiver estimate of GPS time of week (seconds)
 * @param receiverGpsWeek Receiver estimate of GPS week (0-1024+)
 * @param dayOfYear1To366 The day of the year between 1 and 366
 * @param userPositionECEFMeters receiver ECEF position in meters
 * @param doAtmosphericCorrections boolean indicating if atmospheric range corrections should be
 *     applied
 * @return SatellitesPositionPseudorangesResidualAndCovarianceMatrix Object containing satellite
 *     prns, satellite positions in ECEF, pseudorange residuals and covariance matrix.
 */
public SatellitesPositionPseudorangesResidualAndCovarianceMatrix
    calculateSatPosAndPseudorangeResidual(
        GpsNavMessageProto navMeassageProto,
        List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements,
```

```
        double receiverGPSTowAtReceptionSeconds,
        int receiverGpsWeek,
        int dayOfYear1To366,
        double[] userPositionECEFMeters,
        boolean doAtmosphericCorrections)
        throws Exception {
  int numberOfUsefulSatellites =
      getNumberOfUsefulSatellites(usefulSatellitesToReceiverMeasurements);
  // deltaPseudorange is the pseudorange measurement residual
  double[] deltaPseudorangesMeters = new double[numberOfUsefulSatellites];
  double[][] satellitesPositionsECEFMeters = new double[numberOfUsefulSatellites][3];

  // satellite PRNs
  int[] satellitePRNs = new int[numberOfUsefulSatellites];

  // Ionospheric model parameters
  double[] alpha =
          {navMeassageProto.iono.alpha[0], navMeassageProto.iono.alpha[1],
                  navMeassageProto.iono.alpha[2], navMeassageProto.iono.alpha[3]};
  double[] beta = {navMeassageProto.iono.beta[0], navMeassageProto.iono.beta[1],
          navMeassageProto.iono.beta[2], navMeassageProto.iono.beta[3]};
  // Weight matrix for the weighted least square
  RealMatrix covarianceMatrixMetersSquare =
      new Array2DRowRealMatrix(numberOfUsefulSatellites, numberOfUsefulSatellites);
  calculateSatPosAndResiduals(
      navMeassageProto,
      usefulSatellitesToReceiverMeasurements,
      receiverGPSTowAtReceptionSeconds,
      receiverGpsWeek,
      dayOfYear1To366,
      userPositionECEFMeters,
      doAtmosphericCorrections,
      deltaPseudorangesMeters,
      satellitesPositionsECEFMeters,
      satellitePRNs,
      alpha,
      beta,
      covarianceMatrixMetersSquare);

  return new SatellitesPositionPseudorangesResidualAndCovarianceMatrix(satellitePRNs,
      satellitesPositionsECEFMeters, deltaPseudorangesMeters,
      covarianceMatrixMetersSquare.getData());
}

/**
 * Calculates and fill the position of all visible satellites:
 * {@code satellitesPositionsECEFMeters}, pseudorange measurement residual (difference of
 * measured to predicted pseudoranges): {@code deltaPseudorangesMeters} and covariance matrix from
 * the weighted least square: {@code covarianceMatrixMetersSquare}. An array of the satellite PRNs
 * {@code satellitePRNs} is as well filled.
 */
private void calculateSatPosAndResiduals(
```

```
  GpsNavMessageProto navMeassageProto,
  List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements,
  double receiverGPSTowAtReceptionSeconds,
  int receiverGpsWeek,
  int dayOfYear1To366,
  double[] userPositionECEFMeters,
  boolean doAtmosphericCorrections,
  double[] deltaPseudorangesMeters,
  double[][] satellitesPositionsECEFMeters,
  int[] satellitePRNs,
  double[] alpha,
  double[] beta,
  RealMatrix covarianceMatrixMetersSquare)
  throws Exception {
// user position without the clock estimate
double[] userPositionTempECEFMeters =
    {userPositionECEFMeters[0], userPositionECEFMeters[1], userPositionECEFMeters[2]};
int satsCounter = 0;
for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
  if (usefulSatellitesToReceiverMeasurements.get(i) != null) {
    GpsEphemerisProto ephemeridesProto = getEphemerisForSatellite(navMeassageProto, i + 1);
    // Correct the receiver time of week with the estimated receiver clock bias
    receiverGPSTowAtReceptionSeconds =
        receiverGPSTowAtReceptionSeconds - userPositionECEFMeters[3] / SPEED_OF_LIGHT_MPS;

    double pseudorangeMeasurementMeters =
        usefulSatellitesToReceiverMeasurements.get(i).pseudorangeMeters;
    double pseudorangeUncertaintyMeters =
        usefulSatellitesToReceiverMeasurements.get(i).pseudorangeUncertaintyMeters;

    // Assuming uncorrelated pseudorange measurements, the covariance matrix will be diagonal as
    // follows
    covarianceMatrixMetersSquare.setEntry(satsCounter, satsCounter,
        pseudorangeUncertaintyMeters * pseudorangeUncertaintyMeters);

    // Calculate time of week at transmission time corrected with the satellite clock drift
    GpsTimeOfWeekAndWeekNumber correctedTowAndWeek =
        calculateCorrectedTransmitTowAndWeek(ephemeridesProto, receiverGPSTowAtReceptionSeconds,
            receiverGpsWeek, pseudorangeMeasurementMeters);

    // calculate satellite position and velocity
    PositionAndVelocity satPosECEFMetersVelocityMPS = SatellitePositionCalculator
        .calculateSatellitePositionAndVelocityFromEphemeris(ephemeridesProto,
            correctedTowAndWeek.gpsTimeOfWeekSeconds, correctedTowAndWeek.weekNumber,
            userPositionECEFMeters[0], userPositionECEFMeters[1], userPositionECEFMeters[2]);

    satellitesPositionsECEFMeters[satsCounter][0] = satPosECEFMetersVelocityMPS.positionXMeters;
    satellitesPositionsECEFMeters[satsCounter][1] = satPosECEFMetersVelocityMPS.positionYMeters;
    satellitesPositionsECEFMeters[satsCounter][2] = satPosECEFMetersVelocityMPS.positionZMeters;

    // Calculate ionospheric and tropospheric corrections
    double ionosphericCorrectionMeters;
```

```
      double troposphericCorrectionMeters;
      if (doAtmosphericCorrections) {
        ionosphericCorrectionMeters =
            IonosphericModel.ionoKloboucharCorrectionSeconds(
                    userPositionTempECEFMeters,
                    satellitesPositionsECEFMeters[satsCounter],
                    correctedTowAndWeek.gpsTimeOfWeekSeconds,
                    alpha,
                    beta,
                    IonosphericModel.L1_FREQ_HZ)
                * SPEED_OF_LIGHT_MPS;

        troposphericCorrectionMeters =
            calculateTroposphericCorrectionMeters(
                dayOfYear1To366,
                satellitesPositionsECEFMeters,
                userPositionTempECEFMeters,
                satsCounter);
      } else {
        troposphericCorrectionMeters = 0.0;
        ionosphericCorrectionMeters = 0.0;
      }
      double predictedPseudorangeMeters =
          calculatePredictedPseudorange(userPositionECEFMeters, satellitesPositionsECEFMeters,
              userPositionTempECEFMeters, satsCounter, ephemeridesProto, correctedTowAndWeek,
              ionosphericCorrectionMeters, troposphericCorrectionMeters);

      // Pseudorange residual (difference of measured to predicted pseudoranges)
      deltaPseudorangesMeters[satsCounter] =
          pseudorangeMeasurementMeters - predictedPseudorangeMeters;

      // Satellite PRNs
      satellitePRNs[satsCounter] = i + 1;
      satsCounter++;
    }
  }
}

/** Searches ephemerides list for the ephemeris associated with current satellite in process */
private GpsEphemerisProto getEphemerisForSatellite(GpsNavMessageProto navMeassageProto,
                                                   int satPrn) {
  List<GpsEphemerisProto> ephemeridesList
          = new ArrayList<GpsEphemerisProto>(Arrays.asList(navMeassageProto.ephemerids));
  GpsEphemerisProto ephemeridesProto = null;
  int ephemerisPrn = 0;
  for (GpsEphemerisProto ephProtoFromList : ephemeridesList) {
    ephemerisPrn = ephProtoFromList.prn;
    if (ephemerisPrn == satPrn) {
      ephemeridesProto = ephProtoFromList;
      break;
    }
  }
```

```
  return ephemeridesProto;
}

/** Calculates predicted pseudorange in meters */
private double calculatePredictedPseudorange(
    double[] userPositionECEFMeters,
    double[][] satellitesPositionsECEFMeters,
    double[] userPositionNoClockECEFMeters,
    int satsCounter,
    GpsEphemerisProto ephemeridesProto,
    GpsTimeOfWeekAndWeekNumber correctedTowAndWeek,
    double ionosphericCorrectionMeters,
    double troposphericCorrectionMeters)
    throws Exception {
  // Calcualte the satellite clock drift
  double satelliteClockCorrectionMeters =
      SatelliteClockCorrectionCalculator.calculateSatClockCorrAndEccAnomAndTkIteratively(
              ephemeridesProto,
              correctedTowAndWeek.gpsTimeOfWeekSeconds,
              correctedTowAndWeek.weekNumber)
          .satelliteClockCorrectionMeters;

  double satelliteToUserDistanceMeters =
      GpsMathOperations.vectorNorm(GpsMathOperations.subtractTwoVectors(
          satellitesPositionsECEFMeters[satsCounter], userPositionNoClockECEFMeters));
  // Predicted pseudorange
  double predictedPseudorangeMeters =
      satelliteToUserDistanceMeters - satelliteClockCorrectionMeters + ionosphericCorrectionMeters
          + troposphericCorrectionMeters + userPositionECEFMeters[3];
  return predictedPseudorangeMeters;
}

/** Calculates the Gps tropospheric correction in meters */
private double calculateTroposphericCorrectionMeters(int dayOfYear1To366,
    double[][] satellitesPositionsECEFMeters, double[] userPositionTempECEFMeters,
    int satsCounter) {
  double troposphericCorrectionMeters;
  TopocentricAEDValues elevationAzimuthDist =
      EcefToTopocentricConverter.convertCartesianToTopocentericRadMeters(
          userPositionTempECEFMeters, GpsMathOperations.subtractTwoVectors(
              satellitesPositionsECEFMeters[satsCounter], userPositionTempECEFMeters));

  GeodeticLlaValues lla =
      Ecef2LlaConverter.convertECEFToLLACloseForm(userPositionTempECEFMeters[0],
          userPositionTempECEFMeters[1], userPositionTempECEFMeters[2]);

  // Geoid of the area where the receiver is located is calculated once and used for the
  // rest of the dataset as it change very slowly over wide area. This to save the delay
  // associated with accessing Google Elevation API. We assume this very first iteration of WLS
  // will compute the correct altitude above the ellipsoid of the ground at the latitude and
  // longitude
  if (calculateGeoidMeters) {
```

```
    double elevationAboveSeaLevelMeters = 0;
    if (elevationApiHelper == null){
      System.out.println("Elevation above sea level is set to "
          + "default 0 meters. This may cause inaccuracy in tropospheric correction.");
    } else {
      try {
        elevationAboveSeaLevelMeters = elevationApiHelper
            .getElevationAboveSeaLevelMeters(
                Math.toDegrees(lla.latitudeRadians), Math.toDegrees(lla.longitudeRadians)
            );
      } catch (Exception e){
        e.printStackTrace();
        System.out.println("Error when getting elevation from Server. "
            + "Elevation above sea level is set to "
            + "default 0 meters. This may cause inaccuracy in tropospheric correction.");
      }
    }

    geoidHeightMeters = ElevationApiHelper.calculateGeoidHeightMeters(
            lla.altitudeMeters,
            elevationAboveSeaLevelMeters
    );
    troposphericCorrectionMeters = TroposphericModelEgnos.calculateTropoCorrectionMeters(
        elevationAzimuthDist.elevationRadians, lla.latitudeRadians, elevationAboveSeaLevelMeters,
        dayOfYear1To366);
  } else {
    troposphericCorrectionMeters = TroposphericModelEgnos.calculateTropoCorrectionMeters(
        elevationAzimuthDist.elevationRadians, lla.latitudeRadians,
        lla.altitudeMeters - geoidHeightMeters, dayOfYear1To366);
  }
  return troposphericCorrectionMeters;
}

/**
 * Gets the number of useful satellites from a list of
 * {@link GpsMeasurementWithRangeAndUncertainty}.
 */
private int getNumberOfUsefulSatellites(
    List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToReceiverMeasurements) {
  // calculate the number of useful satellites
  int numberOfUsefulSatellites = 0;
  for (int i = 0; i < usefulSatellitesToReceiverMeasurements.size(); i++) {
    if (usefulSatellitesToReceiverMeasurements.get(i) != null) {
      numberOfUsefulSatellites++;
    }
  }
  return numberOfUsefulSatellites;
}

/**
 * Computes the GPS time of week at the time of transmission and as well the corrected GPS week
 * taking into consideration week rollover. The returned GPS time of week is corrected by the
```

```
 * computed satellite clock drift. The result is stored in an instance of
 * {@link GpsTimeOfWeekAndWeekNumber}
 *
 * @param ephemerisProto parameters of the navigation message
 * @param receiverGpsTowAtReceptionSeconds Receiver estimate of GPS time of week when signal was
 *        received (seconds)
 * @param receiverGpsWeek Receiver estimate of GPS week (0-1024+)
 * @param pseudorangeMeters Measured pseudorange in meters
 * @return GpsTimeOfWeekAndWeekNumber Object containing Gps time of week and week number.
 */
private static GpsTimeOfWeekAndWeekNumber calculateCorrectedTransmitTowAndWeek(
    GpsEphemerisProto ephemerisProto, double receiverGpsTowAtReceptionSeconds,
    int receiverGpsWeek, double pseudorangeMeters) throws Exception {
  // GPS time of week at time of transmission: Gps time corrected for transit time (page 98 ICD
  // GPS 200)
  double receiverGpsTowAtTimeOfTransmission =
      receiverGpsTowAtReceptionSeconds - pseudorangeMeters / SPEED_OF_LIGHT_MPS;

  // Adjust for week rollover
  if (receiverGpsTowAtTimeOfTransmission < 0) {
    receiverGpsTowAtTimeOfTransmission += SECONDS_IN_WEEK;
    receiverGpsWeek -= 1;
  } else if (receiverGpsTowAtTimeOfTransmission > SECONDS_IN_WEEK) {
    receiverGpsTowAtTimeOfTransmission -= SECONDS_IN_WEEK;
    receiverGpsWeek += 1;
  }

  // Compute the satellite clock correction term (Seconds)
  double clockCorrectionSeconds =
      SatelliteClockCorrectionCalculator.calculateSatClockCorrAndEccAnomAndTkIteratively(
          ephemerisProto, receiverGpsTowAtTimeOfTransmission,
          receiverGpsWeek).satelliteClockCorrectionMeters / SPEED_OF_LIGHT_MPS;

  // Correct with the satellite clock correction term
  double receiverGpsTowAtTimeOfTransmissionCorrectedSec =
      receiverGpsTowAtTimeOfTransmission + clockCorrectionSeconds;

  // Adjust for week rollover due to satellite clock correction
  if (receiverGpsTowAtTimeOfTransmissionCorrectedSec < 0.0) {
    receiverGpsTowAtTimeOfTransmissionCorrectedSec += SECONDS_IN_WEEK;
    receiverGpsWeek -= 1;
  }
  if (receiverGpsTowAtTimeOfTransmissionCorrectedSec > SECONDS_IN_WEEK) {
    receiverGpsTowAtTimeOfTransmissionCorrectedSec -= SECONDS_IN_WEEK;
    receiverGpsWeek += 1;
  }
  return new GpsTimeOfWeekAndWeekNumber(receiverGpsTowAtTimeOfTransmissionCorrectedSec,
      receiverGpsWeek);
}

/**
 * Calculates the Geometry matrix (describing user to satellite geometry) given a list of
```

```
 * satellite positions in ECEF coordinates in meters and the user position in ECEF in meters.
 *
 * <p>The geometry matrix has four columns, and rows equal to the number of satellites. For each
 * of the rows (i.e. for each of the satellites used), the columns are filled with the normalized
 * line-of-sight vectors and 1 s for the fourth column.
 *
 * <p>Source: Parkinson, B.W., Spilker Jr., J.J.: 'Global positioning system: theory and
 * applications' page 413
 */
private static double[][] calculateGeometryMatrix(double[][] satellitePositionsECEFMeters,
    double[] userPositionECEFMeters) {

  double[][] geometeryMatrix = new double[satellitePositionsECEFMeters.length][4];
  for (int i = 0; i < satellitePositionsECEFMeters.length; i++) {
    geometeryMatrix[i][3] = 1;
  }
  // iterate over all satellites
  for (int i = 0; i < satellitePositionsECEFMeters.length; i++) {
    double[] r = {satellitePositionsECEFMeters[i][0] - userPositionECEFMeters[0],
        satellitePositionsECEFMeters[i][1] - userPositionECEFMeters[1],
        satellitePositionsECEFMeters[i][2] - userPositionECEFMeters[2]};
    double norm = Math.sqrt(Math.pow(r[0], 2) + Math.pow(r[1], 2) + Math.pow(r[2], 2));
    for (int j = 0; j < 3; j++) {
      geometeryMatrix[i][j] =
          (userPositionECEFMeters[j] - satellitePositionsECEFMeters[i][j]) / norm;
    }
  }
  return geometeryMatrix;
}

/**
 * Class containing satellites' PRNs, satellites' positions in ECEF meters, the pseudorange
 * residual per visible satellite in meters and the covariance matrix of the
 * pseudoranges in meters square
 */
protected static class SatellitesPositionPseudorangesResidualAndCovarianceMatrix {

  /** Satellites' PRNs */
  protected final int[] satellitePRNs;

  /** ECEF positions (meters) of useful satellites */
  protected final double[][] satellitesPositionsMeters;

  /** Pseudorange measurement residuals (difference of measured to predicted pseudoranges) */
  protected final double[] pseudorangeResidualsMeters;

  /** Pseudorange covariance Matrix for the weighted least squares (meters square) */
  protected final double[][] covarianceMatrixMetersSquare;

  /** Constructor */
  private SatellitesPositionPseudorangesResidualAndCovarianceMatrix(int[] satellitePRNs,
      double[][] satellitesPositionsMeters, double[] pseudorangeResidualsMeters,
```

```
      double[][] covarianceMatrixMetersSquare) {
    this.satellitePRNs = satellitePRNs;
    this.satellitesPositionsMeters = satellitesPositionsMeters;
    this.pseudorangeResidualsMeters = pseudorangeResidualsMeters;
    this.covarianceMatrixMetersSquare = covarianceMatrixMetersSquare;
  }

}

/**
 * Class containing GPS time of week in seconds and GPS week number
 */
private static class GpsTimeOfWeekAndWeekNumber {
  /** GPS time of week in seconds */
  private final double gpsTimeOfWeekSeconds;

  /** GPS week number */
  private final int weekNumber;

  /** Constructor */
  private GpsTimeOfWeekAndWeekNumber(double gpsTimeOfWeekSeconds, int weekNumber) {
    this.gpsTimeOfWeekSeconds = gpsTimeOfWeekSeconds;
    this.weekNumber = weekNumber;
  }
}

/**
 * Uses the common reception time approach to calculate pseudoranges from the time of week
 * measurements reported by the receiver according to http://cdn.intechopen.com/pdfs-wm/27712.pdf.
 * As well computes the pseudoranges uncertainties for each input satellite
 */
@VisibleForTesting
static List<GpsMeasurementWithRangeAndUncertainty> computePseudorangeAndUncertainties(
    List<GpsMeasurement> usefulSatellitesToReceiverMeasurements,
    Long[] usefulSatellitesToTOWNs,
    long largestTowNs) {

  List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToPseudorangeMeasurements =
      Arrays.asList(
          new GpsMeasurementWithRangeAndUncertainty
              [GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES]);
  for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
    if (usefulSatellitesToTOWNs[i] != null) {
      double deltai = largestTowNs - usefulSatellitesToTOWNs[i];
      double pseudorangeMeters =
          (AVERAGE_TRAVEL_TIME_SECONDS + deltai * SECONDS_PER_NANO) * SPEED_OF_LIGHT_MPS;

      double signalToNoiseRatioLinear =
          Math.pow(10, usefulSatellitesToReceiverMeasurements.get(i).signalToNoiseRatioDb / 10.0);
      // From Global Positoning System book, Misra and Enge, page 416, the uncertainty of the
      // pseudorange measurement is calculated next.
      // For GPS C/A code chip width Tc = 1 microseconds. Narrow correlator with spacing d = 0.1
```

```
        // chip and an average time of DLL correlator T of 20 milliseconds are used.
        double sigmaMeters =
            SPEED_OF_LIGHT_MPS
                * GPS_CHIP_WIDTH_T_C_SEC
                * Math.sqrt(
                    GPS_CORRELATOR_SPACING_IN_CHIPS
                        / (4 * GPS_DLL_AVERAGING_TIME_SEC * signalToNoiseRatioLinear));
        usefulSatellitesToPseudorangeMeasurements.set(
            i,
            new GpsMeasurementWithRangeAndUncertainty(
                usefulSatellitesToReceiverMeasurements.get(i), pseudorangeMeters, sigmaMeters));
      }
    }
    return usefulSatellitesToPseudorangeMeasurements;
  }

}
```

Клас, що описує підрахунок псевдовідстаней з «сирих» навігаційних даних

```
public class PseudorangePositionVelocityFromRealTimeEvents {

  private static final double SECONDS_PER_NANO = 1.0e-9;
  private static final int TOW_DECODED_MEASUREMENT_STATE_BIT = 3;
  /** Average signal travel time from GPS satellite and earth */
  private static final int VALID_ACCUMULATED_DELTA_RANGE_STATE = 1;
  private static final int MINIMUM_NUMBER_OF_USEFUL_SATELLITES = 4;
  private static final int C_TO_N0_THRESHOLD_DB_HZ = 18;

  private static final String SUPL_SERVER_NAME = "supl.google.com";
  private static final int SUPL_SERVER_PORT = 7276;

  private GpsNavMessageProto mHardwareGpsNavMessageProto = null;

  // navigation message parser
  private GpsNavigationMessageStore mGpsNavigationMessageStore = new GpsNavigationMessageStore();
  private double[] mPositionSolutionLatLngDeg = GpsMathOperations.createAndFillArray(3, Double.NaN);
  private double[] mVelocitySolutionEnuMps = GpsMathOperations.createAndFillArray(3, Double.NaN);
  private final double[] mPositionVelocityUncertaintyEnu
      = GpsMathOperations.createAndFillArray(6, Double.NaN);
  private double[] mPseudorangeResidualsMeters =
      GpsMathOperations.createAndFillArray(
          GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES, Double.NaN
      );
  private boolean mFirstUsefulMeasurementSet = true;
  private int[] mReferenceLocation = null;
  private long mLastReceivedSuplMessageTimeMillis = 0;
  private long mDeltaTimeMillisToMakeSuplRequest = TimeUnit.MINUTES.toMillis(30);
  private boolean mFirstSuplRequestNeeded = true;
  private GpsNavMessageProto mGpsNavMessageProtoUsed = null;

  // Only the interface of pseudorange smoother is provided. Please implement customized smoother.
  PseudorangeSmoother mPseudorangeSmoother = new PeudorangeDgpsSmoother();
```

```
private final UserPositionVelocityWeightedLeastSquare mUserPositionVelocityLeastSquareCalculator =
    new UserPositionVelocityWeightedLeastSquare(mPseudorangeSmoother);
private GpsMeasurement[] mUsefulSatellitesToReceiverMeasurements =
    new GpsMeasurement[GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES];
private Long[] mUsefulSatellitesToTowNs =
    new Long[GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES];
private long mLargestTowNs = Long.MIN_VALUE;
private double mArrivalTimeSinceGPSWeekNs = 0.0;
private int mDayOfYear1To366 = 0;
private int mGpsWeekNumber = 0;
private long mArrivalTimeSinceGpsEpochNs = 0;

/**
 * Computes Weighted least square position and velocity solutions from a received {@link
 * GnssMeasurementsEvent} and store the result in {@link
 * PseudorangePositionVelocityFromRealTimeEvents#mPositionSolutionLatLngDeg} and {@link
 * PseudorangePositionVelocityFromRealTimeEvents#mVelocitySolutionEnuMps}
 */
public void computePositionVelocitySolutionsFromRawMeas(GnssMeasurementsEvent event, List<RawDgps> dgps)
    throws Exception {
  if (mReferenceLocation == null) {
    // If no reference location is received, we can not get navigation message from SUPL and hence
    // we will not try to compute location.
      Timber.d(" No reference Location ..... no position is calculated");
    return;
  }
  for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
    mUsefulSatellitesToReceiverMeasurements[i] = null;
    mUsefulSatellitesToTowNs[i] = null;
  }

  GnssClock gnssClock = event.getClock();
  mArrivalTimeSinceGpsEpochNs = gnssClock.getTimeNanos() - gnssClock.getFullBiasNanos();

  for (GnssMeasurement measurement : event.getMeasurements()) {
    // ignore any measurement if it is not from GPS constellation
    if (measurement.getConstellationType() != GnssStatus.CONSTELLATION_GPS) {
      continue;
    }
    // ignore raw data if time is zero, if signal to noise ratio is below threshold or if
    // TOW is not yet decoded
    if (measurement.getCn0DbHz() >= C_TO_N0_THRESHOLD_DB_HZ
        && (measurement.getState() & (1L << TOW_DECODED_MEASUREMENT_STATE_BIT)) != 0) {

      // calculate day of year and Gps week number needed for the least square
      GpsTime gpsTime = new GpsTime(mArrivalTimeSinceGpsEpochNs);
      // Gps weekly epoch in Nanoseconds: defined as of every Sunday night at 00:00:000
      long gpsWeekEpochNs = GpsTime.getGpsWeekEpochNano(gpsTime);
      mArrivalTimeSinceGPSWeekNs = mArrivalTimeSinceGpsEpochNs - gpsWeekEpochNs;
      mGpsWeekNumber = gpsTime.getGpsWeekSecond().first;
      // calculate day of the year between 1 and 366
      Calendar cal = gpsTime.getTimeInCalendar();
```

```
    mDayOfYear1To366 = cal.get(Calendar.DAY_OF_YEAR);

    long receivedGPSTowNs = measurement.getReceivedSvTimeNanos();
    if (receivedGPSTowNs > mLargestTowNs) {
      mLargestTowNs = receivedGPSTowNs;
    }
    mUsefulSatellitesToTowNs[measurement.getSvid() - 1] = receivedGPSTowNs;
    GpsMeasurement gpsReceiverMeasurement =
        new GpsMeasurement(
            (long) mArrivalTimeSinceGPSWeekNs,
            measurement.getAccumulatedDeltaRangeMeters(),
            measurement.getAccumulatedDeltaRangeState() == VALID_ACCUMULATED_DELTA_RANGE_STATE,
            measurement.getPseudorangeRateMetersPerSecond(),
            measurement.getCn0DbHz(),
            measurement.getAccumulatedDeltaRangeUncertaintyMeters(),
            measurement.getPseudorangeRateUncertaintyMetersPerSecond());
    mUsefulSatellitesToReceiverMeasurements[measurement.getSvid() - 1] = gpsReceiverMeasurement;
  }
}

// check if we should continue using the navigation message from the SUPL server, or use the
// navigation message from the device if we fully received it
boolean useNavMessageFromSupl =
    continueUsingNavMessageFromSupl(
        mUsefulSatellitesToReceiverMeasurements, mHardwareGpsNavMessageProto);
if (useNavMessageFromSupl) {
    Timber.d("Using navigation message from SUPL server");

  if (mFirstSuplRequestNeeded
      || (System.currentTimeMillis() - mLastReceivedSuplMessageTimeMillis)
          > mDeltaTimeMillisToMakeSuplRequest) {
    // The following line is blocking call for SUPL connection and back. But it is fast enough
    mGpsNavMessageProtoUsed = getSuplNavMessage(mReferenceLocation[0], mReferenceLocation[1]);
    if (!isEmptyNavMessage(mGpsNavMessageProtoUsed)) {
      mFirstSuplRequestNeeded = false;
      mLastReceivedSuplMessageTimeMillis = System.currentTimeMillis();
    } else {
      return;
    }
  }

} else {
    Timber.d("Using navigation message from the GPS receiver");
  mGpsNavMessageProtoUsed = mHardwareGpsNavMessageProto;
}

// some times the SUPL server returns less satellites than the visible ones, so remove those
// visible satellites that are not returned by SUPL
for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
  if (mUsefulSatellitesToReceiverMeasurements[i] != null
      && !navMessageProtoContainsSvid(mGpsNavMessageProtoUsed, i + 1)) {
    mUsefulSatellitesToReceiverMeasurements[i] = null;
```

```
    mUsefulSatellitesToTowNs[i] = null;
  }
}

// calculate the number of useful satellites
int numberOfUsefulSatellites = 0;
for (GpsMeasurement element : mUsefulSatellitesToReceiverMeasurements) {
  if (element != null) {
    numberOfUsefulSatellites++;
  }
}
if (numberOfUsefulSatellites >= MINIMUM_NUMBER_OF_USEFUL_SATELLITES) {
  // ignore first set of > 4 satellites as they often result in erroneous position
  if (!mFirstUsefulMeasurementSet) {
    // start with last known position and velocity of zero. Following the structure:
    // [X position, Y position, Z position, clock bias,
    //  X Velocity, Y Velocity, Z Velocity, clock bias rate]
    double[] positionVeloctySolutionEcef = GpsMathOperations.createAndFillArray(8, 0);
    double[] positionVelocityUncertaintyEnu = GpsMathOperations.createAndFillArray(6, 0);
    double[] pseudorangeResidualMeters
        = GpsMathOperations.createAndFillArray(
            GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES, Double.NaN
    );
    performPositionVelocityComputationEcef(
        mUserPositionVelocityLeastSquareCalculator,
        mUsefulSatellitesToReceiverMeasurements,
        mUsefulSatellitesToTowNs,
        mLargestTowNs,
        mArrivalTimeSinceGPSWeekNs,
        mDayOfYear1To366,
        mGpsWeekNumber,
        positionVeloctySolutionEcef,
        positionVelocityUncertaintyEnu,
        pseudorangeResidualMeters,
        dgps);
    // convert the position solution from ECEF to latitude, longitude and altitude
    GeodeticLlaValues latLngAlt =
        Ecef2LlaConverter.convertECEFToLLACloseForm(
            positionVeloctySolutionEcef[0],
            positionVeloctySolutionEcef[1],
            positionVeloctySolutionEcef[2]);
    mPositionSolutionLatLngDeg[0] = Math.toDegrees(latLngAlt.latitudeRadians);
    mPositionSolutionLatLngDeg[1] = Math.toDegrees(latLngAlt.longitudeRadians);
    mPositionSolutionLatLngDeg[2] = latLngAlt.altitudeMeters;
    mPositionVelocityUncertaintyEnu[0] = positionVelocityUncertaintyEnu[0];
    mPositionVelocityUncertaintyEnu[1] = positionVelocityUncertaintyEnu[1];
    mPositionVelocityUncertaintyEnu[2] = positionVelocityUncertaintyEnu[2];
    System.arraycopy(
        pseudorangeResidualMeters,
        0 /*source starting pos*/,
        mPseudorangeResidualsMeters,
        0 /*destination starting pos*/,
```

```
        GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES /*length of elements*/
        );
    Timber.d("Position Uncertainty ENU Meters :"
            + mPositionVelocityUncertaintyEnu[0]
            + " "
            + mPositionVelocityUncertaintyEnu[1]
            + " "
            + mPositionVelocityUncertaintyEnu[2]);
      Timber.d("Latitude, Longitude, Altitude: "
              + mPositionSolutionLatLngDeg[0]
              + " "
              + mPositionSolutionLatLngDeg[1]
              + " "
              + mPositionSolutionLatLngDeg[2]);
    EnuValues velocityEnu = Ecef2EnuConverter.convertEcefToEnu(
        positionVeloctySolutionEcef[4],
        positionVeloctySolutionEcef[5],
        positionVeloctySolutionEcef[6],
        latLngAlt.latitudeRadians,
        latLngAlt.longitudeRadians
    );

    mVelocitySolutionEnuMps[0] = velocityEnu.enuEast;
    mVelocitySolutionEnuMps[1] = velocityEnu.enuNorth;
    mVelocitySolutionEnuMps[2] = velocityEnu.enuUP;
      Timber.d("Velocity ENU Mps: "
              + mVelocitySolutionEnuMps[0]
              + " "
              + mVelocitySolutionEnuMps[1]
              + " "
              + mVelocitySolutionEnuMps[2]);
    mPositionVelocityUncertaintyEnu[3] = positionVelocityUncertaintyEnu[3];
    mPositionVelocityUncertaintyEnu[4] = positionVelocityUncertaintyEnu[4];
    mPositionVelocityUncertaintyEnu[5] = positionVelocityUncertaintyEnu[5];
      Timber.d("Velocity Uncertainty ENU Mps :"
              + mPositionVelocityUncertaintyEnu[3]
              + " "
              + mPositionVelocityUncertaintyEnu[4]
              + " "
              + mPositionVelocityUncertaintyEnu[5]);
  }
  mFirstUsefulMeasurementSet = false;
} else {
    Timber.d("Less than four satellites with SNR above threshold visible ... "
            + "no position is calculated!");

  mPositionSolutionLatLngDeg = GpsMathOperations.createAndFillArray(3, Double.NaN);
  mVelocitySolutionEnuMps = GpsMathOperations.createAndFillArray(3, Double.NaN);
  mPseudorangeResidualsMeters =
      GpsMathOperations.createAndFillArray(
          GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES, Double.NaN
      );
```

```
  }
}

private boolean isEmptyNavMessage(GpsNavMessageProto navMessageProto) {
  if(navMessageProto.iono == null)return true;
  if(navMessageProto.ephemerids.length ==0)return true;
  return  false;
}

private boolean navMessageProtoContainsSvid(GpsNavMessageProto navMessageProto, int svid) {
  List<GpsEphemerisProto> ephemeridesList =
          new ArrayList<GpsEphemerisProto>(Arrays.asList(navMessageProto.ephemerids));
  for (GpsEphemerisProto ephProtoFromList : ephemeridesList) {
    if (ephProtoFromList.prn == svid) {
      return true;
    }
  }
  return false;
}

/**
 * Calculates ECEF least square position and velocity solutions from an array of {@link
 * GpsMeasurement} in meters and meters per second and store the result in {@code
 * positionVelocitySolutionEcef}
 */
private void performPositionVelocityComputationEcef(
    UserPositionVelocityWeightedLeastSquare userPositionVelocityLeastSquare,
    GpsMeasurement[] usefulSatellitesToReceiverMeasurements,
    Long[] usefulSatellitesToTOWNs,
    long largestTowNs,
    double arrivalTimeSinceGPSWeekNs,
    int dayOfYear1To366,
    int gpsWeekNumber,
    double[] positionVelocitySolutionEcef,
    double[] positionVelocityUncertaintyEnu,
    double[] pseudorangeResidualMeters,
    List<RawDgps> dgps)
    throws Exception {

  List<GpsMeasurementWithRangeAndUncertainty> usefulSatellitesToPseudorangeMeasurements =
      UserPositionVelocityWeightedLeastSquare.computePseudorangeAndUncertainties(
          Arrays.asList(usefulSatellitesToReceiverMeasurements),
          usefulSatellitesToTOWNs,
          largestTowNs);

  // calculate iterative least square position solution and velocity solutions
  userPositionVelocityLeastSquare.calculateUserPositionVelocityLeastSquare(
      mGpsNavMessageProtoUsed,
      usefulSatellitesToPseudorangeMeasurements,
      arrivalTimeSinceGPSWeekNs * SECONDS_PER_NANO,
      gpsWeekNumber,
      dayOfYear1To366,
```

```
      positionVelocitySolutionEcef,
      positionVelocityUncertaintyEnu,
      pseudorangeResidualMeters,
      dgps);

    Timber.d("Least Square Position Solution in ECEF meters: "
            + positionVelocitySolutionEcef[0]
            + " "
            + positionVelocitySolutionEcef[1]
            + " "
            + positionVelocitySolutionEcef[2]);
    Timber.d("Estimated Receiver clock offset in meters: " + positionVelocitySolutionEcef[3]);

    Timber.d("Velocity Solution in ECEF Mps: "
            + positionVelocitySolutionEcef[4]
            + " "
            + positionVelocitySolutionEcef[5]
            + " "
            + positionVelocitySolutionEcef[6]);
    Timber.d("Estimated Reciever clock offset rate in mps: " + positionVelocitySolutionEcef[7]);
}

/**
 * Reads the navigation message from the SUPL server by creating a Stubby client to Stubby server
 * that wraps the SUPL server. The input is the time in nanoseconds since the GPS epoch at which
 * the navigation message is required and the output is a {@link GpsNavMessageProto}
 *
 * @throws IOException
 * @throws UnknownHostException
 */
private GpsNavMessageProto getSuplNavMessage(long latE7, long lngE7)
    throws UnknownHostException, IOException {
  SuplRrlpController suplRrlpController =
      new SuplRrlpController(SUPL_SERVER_NAME, SUPL_SERVER_PORT);
  GpsNavMessageProto navMessageProto = suplRrlpController.generateNavMessage(latE7, lngE7);

  return navMessageProto;
}

/**
 * Checks if we should continue using the navigation message from the SUPL server, or use the
 * navigation message from the device if we fully received it. If the navigation message read from
 * the receiver has all the visible satellite ephemerides, return false, otherwise, return true.
 */
private static boolean continueUsingNavMessageFromSupl(
        GpsMeasurement[] usefulSatellitesToReceiverMeasurements,
        GpsNavMessageProto hardwareGpsNavMessageProto) {
  boolean useNavMessageFromSupl = true;
  if (hardwareGpsNavMessageProto != null) {
    ArrayList<GpsEphemerisProto> hardwareEphemeridesList=
            new ArrayList<GpsEphemerisProto>(Arrays.asList(hardwareGpsNavMessageProto.ephemerids));
    if (hardwareGpsNavMessageProto.iono != null) {
```

```
      for (int i = 0; i < GpsNavigationMessageStore.MAX_NUMBER_OF_SATELLITES; i++) {
        if (usefulSatellitesToReceiverMeasurements[i] != null) {
          int prn = i + 1;
          for (GpsEphemerisProto hardwareEphProtoFromList : hardwareEphemeridesList) {
            if (hardwareEphProtoFromList.prn == prn) {
              useNavMessageFromSupl = false;
              break;
            }
            useNavMessageFromSupl = true;
          }
          if (useNavMessageFromSupl == true) {
            break;
          }
        }
      }
    }
  }
  return useNavMessageFromSupl;
}

/**
 * Parses a string array containing an updates to the navigation message and return the most
 * recent {@link GpsNavMessageProto}.
 */
public void parseHwNavigationMessageUpdates(GnssNavigationMessage navigationMessage) {
  byte messagePrn = (byte) navigationMessage.getSvid();
  byte messageType = (byte) (navigationMessage.getType() >> 8);
  int subMessageId = navigationMessage.getSubmessageId();

  byte[] messageRawData = navigationMessage.getData();
  // parse only GPS navigation messages for now
  if (messageType == 1) {
    mGpsNavigationMessageStore.onNavMessageReported(
        messagePrn, messageType, (short) subMessageId, messageRawData);
    mHardwareGpsNavMessageProto = mGpsNavigationMessageStore.createDecodedNavMessage();
  }

}

/** Sets a rough location of the receiver that can be used to request SUPL assistance data */
public void setReferencePosition(int latE7, int lngE7, int altE7) {
  if (mReferenceLocation == null) {
    mReferenceLocation = new int[3];
  }
  mReferenceLocation[0] = latE7;
  mReferenceLocation[1] = lngE7;
  mReferenceLocation[2] = altE7;
}

/**
 * Converts the input from LLA coordinates to ECEF and set up the reference position of
 * {@code mUserPositionVelocityLeastSquareCalculator} to calculate a corrected residual.
```

```
   *
   * <p> Based on this input ground truth, true residuals can be computed. This is done by using
   * the high elevation satellites to compute the true user clock error and with the knowledge of
   * the satellite positions.
   *
   * <p> If no ground truth is set, no residual analysis will be performed.
   */
  public void setCorrectedResidualComputationTruthLocationLla
  (double[] groundTruthLocationLla) {
    if (groundTruthLocationLla == null) {
      mUserPositionVelocityLeastSquareCalculator
          .setTruthLocationForCorrectedResidualComputationEcef(null);
      return;
    }
    GeodeticLlaValues llaValues =
        new GeodeticLlaValues(
            Math.toRadians(groundTruthLocationLla[0]),
            Math.toRadians(groundTruthLocationLla[1]),
            Math.toRadians(groundTruthLocationLla[2]));
    mUserPositionVelocityLeastSquareCalculator.setTruthLocationForCorrectedResidualComputationEcef(
        Lla2EcefConverter.convertFromLlaToEcefMeters(llaValues));
  }

  /** Returns the last computed weighted least square position solution */
  public double[] getPositionSolutionLatLngDeg() {
    return mPositionSolutionLatLngDeg;
  }

  /** Returns the last computed Velocity solution */
  public double[] getVelocitySolutionEnuMps() {
    return mVelocitySolutionEnuMps;
  }

  /**
   * Returns the last computed position and velocity uncertainties in meters and meter per seconds,
   * respectively.
   */
  public double[] getPositionVelocityUncertaintyEnu() {
    return mPositionVelocityUncertaintyEnu;
  }

  /**
   * Returns the pseudorange residuals corrected by using clock bias computed from highest
   * elevationDegree satellites.
   */
  public double[] getPseudorangeResidualsMeters() {
    return mPseudorangeResidualsMeters;
  }
}
```