СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 126 « Інформаційні системи та технології»
(шифр і назва спеціальності)

на тему «Інформаційна технологія прогнозування спрацювання подушок безпеки для автомобільного транспорту на основі статистичного аналізу даних»

Виконав: студент групи ІСТ-21д	_____ (підпис)	П.В. Мацегора (ініціали і прізвище)
Керівник	_____ (підпис)	В.О. Лифар (ініціали і прізвище)
Завідувач кафедри	_____ (підпис)	О.І. Захожай (ініціали і прізвище)

Рецензент _____ Д.В. Ратов_____

Київ 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування
Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Захожай О.І.
“___” _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Мацегора Петро Васильович
(прізвище, ім’я, по батькові)

1. Тема роботи: Інформаційна технологія прогнозування спрацювання подушок безпеки для автомобільного транспорту на основі статистичного аналізу даних
керівник роботи Володимир Олексійович Лифар доцент, д.т.н.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “_27_” _травня_ 2025 року №67/15.15-С

2. Строк подання студентом роботи 16.06.2025р.
3. Вихідні дані до роботи: матеріали переддипломної практики
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): огляд та аналіз ключових блоку управління подушками безпеки, різних методів передачі даних між сервером та блоком управління, і оцінку їх безпеки, надійності та швидкості передачі даних. Розглянути математичні алгоритми для збору та обробки даних для статистичного аналізу даних, процеси збору, обробки та збереження інформації про погодні та дорожні умови з відкритих джерел. Також збір інформації про дорожні аварії, які вже відбулися. Висвітлено архітектуру розробленої системи та тестування її компонентів. Описано заходи з охорони праці, що включають безпечну організацію робочого місця та вимоги до умов праці, пожежну безпеку та електробезпеку. В розділі про охорону праці розглядаються нормативи щодо мікроклімату, освітленості, шуму та вібрації.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників) Електронні плакати

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина			
Охорона праці			

7. Дата видачі завдання 01.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Отримання завдання та збір матеріалів	01.04.25	виконано
2	Огляд предметної області	18.04.25	виконано
3	Формулювання вимог до системи та розробка основних алгоритмів	04.05.25	виконано
4	Розробка практичної частини	01.06.25	виконано
5	Оформлення пояснювальної записки	12.06.25	виконано
6	Підготовка та подання роботи до захисту	16.06.25	виконано

Студент _____ П.В. Мацегора
підпис (ініціали і прізвище)

Керівник роботи _____ В.О. Лифар
підпис (ініціали і прізвище)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту бакалавра: 58 сторінки, 24 рис., 11 табл., 21 джерело посилань, 2 додатки на 43 сторінках.

У дипломному проекті розроблено програмне забезпечення для прогнозування спрацювання подушок безпеки для автомобільного транспорту на основі статистичного аналізу даних, що забезпечує підвищену безпеку і точність спрацювання подушок безпеки. Проведено аналіз потенційних загроз, помилок і недоліків у отримання та передачі даних, та їх використанні у автомобільному засобі. Особливу увагу приділено статистичному аналізу даних, їх швидкому та надійному опрацюванні та розробки додаткових рівнів безпеки для системи в цілому.

Розроблений системний підхід включає алгоритми формування інструкції на основі даних, зібраних з попередніх аварії, погодних та дорожніх умов, та їх подальшою передачею до блоку управління подушками безпеки. Здійснено математичне моделювання реакції автомобіля на різні типи зіткнення, а також проведено тестування розробленого програмного рішення, підтверджуючи його ефективність і відповідність сучасним вимогам безпеки.

Дипломний проєкт демонструє впровадження інноваційних технологій у сфері подушок безпеки, забезпечуючи підвищений рівень безпеки та захисту пасажирів під час аварії.

СИСТЕМНИЙ АНАЛІЗ ДАНИХ, АВАРІЇ, ПОДУШКИ БЕЗПЕКИ, БЕЗПЕКА ПАСАЖИРІВ, ОХОРОНА ПРАЦІ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Умови отримання дипломного проекту:

СНУ ім. Володимира Даля, м. Київ, вул. Чигоріна, 18

ЗМІСТ

ВСТУП

Подушки безпеки були винайдені не так давно і пройшли багато трансформацій. У 1920-х роках розроблені та запатентовані перші прототипи подушок безпеки. В 1950-х роках розроблено і запатентовано перший прототип системи подушок безпеки для автомобіля. І тільки приблизно 30 років тому, в середині 1990-років стало стандартом мати подушки безпеки для автомобіля в країнах ЄС, а в 1998 році це стало стандартом і в США. Після цього за останні роки подушки безпеки покращувалися, вдосконалювалася безпека хімікатів та способи їх випуску. Покращувалася взаємодія між подушками безпеки та іншими пристроями в автомобілі, такими як ремені безпеки, датчики удару, та датчики швидкості обертання коліс. З часом стало стандартом, що подушки безпеки не просто спрацьовують при зіткненні з іншим автомобілем, а застосовуються алгоритми зменшення ушкодження для пасажирів.

У сучасній реальності, коли застосування «big data» стає нормою, а більшість компаній збирають та використовують великі обсяги даних для збільшення кількості продаж товару, подушки безпеки залишаються закритим простором, який покладається тільки на отриманні даних із замкнутої системи та закриті алгоритми для знаходження вирішення.

Метою цього проекту є спроба зробити новий крок вперед в можливостях подушок безпеки, підвищити швидкість та якість реагування через надання систематичного аналізу даних дорожніх аварії, які вже сталися.

Для реалізації цього проекту були поставленні наступні умови:

будь які дані, бібліотеки чи API, які будуть використанні, повинні знаходитися у відкритому доступі.

система не повинна бути залежною від даних, які будуть отримані методом систематичного аналізу, а також мати алгоритм закритого принципу дії.

система повинна забезпечити спрацювання подушки безпеки за час, менший ніж 500 мілісекунд, від моменту отримання даних про удар.

система повинна використовувати такі ж перевірки, як і на сучасних автомобілях.

Дана бакалаврська атестаційна робота присвячена питанням дослідження та впровадженню систематичного аналізу роботи подушок безпеки.

РОЗДІЛ 1 РОЗГЛЯД СИСТЕМИ РОБІТ ТА ПРИЙНЯТТЯ РІШЕНЬ У СУЧАСНИХ ПОДУШКАХ БЕЗПЕКИ

1.1 Огляд подушок безпеки в сучасності

Подушки безпеки є ключовим елементом сучасних систем пасивної безпеки автомобілів, що істотно знижують ризик важких травм і летальних наслідків під час дорожньо-транспортних пригод (ДТП). Згідно з даними Всесвітньої організації охорони здоров'я (ВООЗ), щорічно у світі гине приблизно 1,19[1] мільйона осіб у результаті ДТП, причому понад половина з них — вразливі учасники дорожнього руху, включно з пішоходами, велосипедистами та мотоциклістами. У цьому контексті ефективність систем безпеки, зокрема подушок безпеки, набуває вирішального значення в зусиллях зі зменшення травматизму.

Подушки безпеки розроблені для амортизації ударів та зниження сили впливу при зіткненні, особливо в разі фронтального або бокового удару. Вони функціонують у поєднанні з ременями безпеки, значно підвищуючи шанси пасажирів на виживання. За даними ВООЗ, виконання регламентів ООН щодо конструктивної безпеки транспортних засобів, зокрема щодо оснащення автомобілів подушками безпеки, може істотно зменшити рівень травматизму як серед пасажирів, так і пішоходів.

Наявність подушок безпеки є особливо важливою в умовах значного поширення факторів ризику зокрема, перевищення швидкості, вживання алкоголю та психоактивних речовин, використання мобільних пристроїв під час водіння. Наприклад, при бічному зіткненні ризик смерті пасажирів становить 85% при швидкості 65 км/год, що підкреслює необхідність ефективного захисту від бічних ударів, де особливо критичною є функція бокових подушок безпеки.

Згідно з принципами підходу «безпечної системи», технічні засоби безпеки мають компенсувати вразливість людини до серйозних травм і передбачати помилки водіїв. Подушки безпеки є втіленням цього принципу, оскільки вони захищають життєво важливі органи під час зіткнення, коли людська помилка призводить до аварії. Водночас їх ефективність є максимальною лише за умов належного технічного обслуговування та сумісного використання з ременями безпеки.

1.2 Огляд роботи подушок безпеки як системи

Система подушок безпеки автомобіля це складний комплекс технічних засобів, призначених для мінімізації ризику отримання травм пасажирами під час

дорожньо-транспортних пригод. Вона включає кілька ключових компонентів, зокрема датчики удару, блок керування, газогенератори, власне подушки безпеки та допоміжні системи, зокрема ремені безпеки з преднатягувачами.

Датчики удару розташовуються в передній, боковій, задній частинах транспортного засобу, а також у днищі для реєстрації параметрів зіткнення, таких як сила, напрямок та тривалість удару. У разі виявлення зіткнення ці сенсори передають сигнал до блоку керування подушками безпеки, який виконує аналіз отриманих даних та визначає необхідність активації подушок безпеки. Процес ухвалення рішення ґрунтується на інтенсивності удару, положенні пасажирів та необхідності захисту певних ділянок тіла.

При перевищенні встановленого порогу удару блок керування ініціює запуск газогенераторів. Газогенератори містять піротехнічні складові, які під час ініціювання вступають у хімічну реакцію з утворенням інертного газу (найчастіше азоту або аргону). Цей газ швидко заповнює камеру подушки безпеки, приводячи її до розгортання. Весь процес займає кілька мілісекунд, що дозволяє ефективно захистити пасажирів від вторинного удару об елементи салону автомобіля.

Залежно від призначення подушки безпеки класифікуються на кілька типів. Фронтальні подушки встановлюються в передній частині салону та захищають водія і переднього пасажира при лобовому зіткненні. Бічні подушки, розміщені в дверних панелях або сидіннях, забезпечують захист під час бокових ударів. Шторкові подушки, які монтуються у верхній частині салону вздовж вікон, запобігають травмуванню голови при бічних зіткненнях або перекиданні автомобіля. Колінні подушки, розташовані в нижній частині передньої панелі, сприяють зниженню ризику травмування нижніх кінцівок. Крім того, у сучасних автомобілях можуть бути встановлені центральні подушки безпеки, що запобігають зіткненню пасажирів один з одним під час аварії.

Істотну роль у забезпеченні пасивної безпеки відіграють ремені безпеки з преднатягувачами. У разі зіткнення спеціальні механізми автоматично натягують ремінь, що дозволяє мінімізувати переміщення тіла пасажира та забезпечити його оптимальне положення для взаємодії з подушками безпеки.

Сучасні системи подушок безпеки використовують адаптивні технології, що дозволяють регулювати жорсткість та швидкість розкриття подушок залежно від таких параметрів, як маса та положення пасажира, швидкість зіткнення тощо. Це особливо важливо для захисту дітей і літніх осіб, адже знижує ризик травмування через надмірну силу розгортання подушки.

1.3 Пасивна та Активна безпека в автомобілі

Пасивна безпека спрямована на мінімізацію наслідків аварій, тоді як активна безпека націлена на запобігання виникненню дорожньо-транспортних пригод (ДТП) [2]

Пасивна безпека охоплює конструктивні елементи автомобіля, які забезпечують захист пасажирів під час аварії. Основні компоненти пасивної безпеки включають ремені безпеки, подушки безпеки, конструкцію кузова з зонами деформації та підголівники. Ці елементи працюють без активних дій з боку водія чи пасажирів, забезпечуючи захист у разі аварії [2].

Активна безпека включає системи та технології, які допомагають запобігти виникненню аварійних ситуацій. До основних систем активної безпеки належать антиблокувальна система гальм (ABS), система електронного контролю стійкості (ESC), система допомоги при екстреному гальмуванні (EBA), адаптивний круїз-контроль та система попередження про виїзд зі смуги руху. Ці системи активно втручаються в процес керування, допомагаючи водієві уникнути небезпечних ситуацій на дорозі [2].

Оптимальний рівень безпеки досягається завдяки інтеграції пасивних та активних систем. Наприклад, система ESC може запобігти заносу автомобіля, тоді як ремені та подушки безпеки захистять пасажирів у разі неминучого зіткнення. Такий комплексний підхід забезпечує всебічний захист учасників дорожнього руху [2].

1.4 Дослідження роботи ACU

Дослідження роботи блоку керування подушками безпеки (ACU) є ключовим аспектом аналізу ефективності пасивних систем безпеки транспортних засобів. Блок керування ACU (Airbag Control Unit) виконує критично важливу функцію в процесі активації подушок безпеки та їхньої взаємодії з іншими системами автомобіля.

Основним завданням ACU є отримання, обробка та аналіз даних від датчиків удару, акселерометрів та інших вхідних сигналів, що надходять у режимі реального часу. На основі цих даних блок ухвалює рішення щодо розгортання подушок безпеки, визначаючи необхідність активації, а також її параметри, такі як час відкриття та сила наповнення газом. Дослідження роботи ACU передбачає оцінку точності та швидкості його реакції в різних сценаріях дорожньо-транспортних пригод (ДТП).

Одним із ключових аспектів дослідження є аналіз алгоритмів прийняття рішень, що реалізовані в мікроконтролері ACU. Сучасні блоки керування використовують адаптивні алгоритми, що дозволяють враховувати силу зіткнення, тип дорожнього покриття, масу пасажирів та їхнє положення в салоні. Це дозволяє мінімізувати ризик травматизму, адаптуючи інтенсивність спрацьовування системи до конкретних умов аварії.

Важливим напрямом досліджень є також випробування надійності апаратного забезпечення ACU. Оскільки блок керування працює у складних умовах, таких як різкі температурні коливання, вібрації та удари, його компоненти повинні відповідати високим стандартам стійкості до навантажень. Дослідження зосереджуються на тестуванні електронних схем, стійкості програмного забезпечення до збоїв та можливості резервного дублювання критичних функцій.

Крім того, у сучасних автомобілях ACU часто взаємодіє з іншими системами активної безпеки, такими як адаптивний круїз-контроль, система екстреного гальмування та моніторинг сліпих зон. Дослідження інтеграції ACU з такими

системами дозволяє покращити загальну ефективність безпеки транспортного засобу.

1.5 Огляд основних компонентів ACU

Автоматизовані керуючі установки (ACU) складаються з декількох основних компонентів, кожен з яких виконує певні функції для забезпечення ефективної роботи системи. Центральним елементом є обчислювальний модуль, що містить процесорні блоки та пам’ять для обробки даних і зберігання необхідних алгоритмів керування. Він координує роботу всіх інших частин системи та реалізує алгоритми управління у відповідності до заданих параметрів.

Система датчиків і сенсорів забезпечує збирання інформації про стан об'єкта керування та зовнішнього середовища. Вона може включати температурні, тискові, оптичні, акустичні, магнітні та інші сенсори, що реєструють зміни параметрів у реальному часі. Отримані дані передаються до обчислювального модуля для подальшої обробки.

Комунікаційний модуль виконує функцію обміну інформацією між компонентами ACU, а також забезпечує зв’язок із зовнішніми системами. Він може використовувати дротові або бездротові технології передачі даних, такі як Ethernet, CAN-шина, Wi-Fi або інші спеціалізовані протоколи зв’язку.

Виконавчі механізми є складовою, що перетворює керуючі сигнали на фізичні дії. Вони можуть бути представлені електродвигунами, соленоїдами, гідравлічними або пневматичними приводами, залежно від специфіки керованого об'єкта. Керування цими механізмами здійснюється відповідно до команд, отриманих від обчислювального модуля.

Модуль енерозабезпення відповідає за постачання електроенергії до всіх компонентів ACU. Він включає блоки живлення, акумуляторні батареї або системи резервного енергозабезпечення для забезпечення безперебійної роботи.

Програмне забезпечення ACU складається з алгоритмів керування, системної логіки та інтерфейсів взаємодії з користувачем. Воно забезпечує обробку вхідних

даних, ухвалення рішень та формування вихідних сигналів. Інтерфейс користувача може бути реалізований у вигляді графічної панелі або віддаленого програмного середовища для моніторингу та налаштування системи.

Система діагностики та самотестування забезпечує контроль працездатності ACU та її окремих компонентів. Вона аналізує робочі параметри, виявляє можливі відхилення від нормальної роботи та генерує сповіщення у разі виникнення дефекту.

1.6 Дослідження системи прийняття рішень для активації подушок безпеки

Система прийняття рішень для активації подушок безпеки є складним багаторівневим комплексом, що поєднує сенсорні технології, алгоритми аналізу подій та виконавчі механізми. Основна її мета полягає у визначенні моменту, коли активація подушок безпеки є необхідною для мінімізації травм водія та пасажирів[2].

Центральним компонентом системи є блок управління подушками безпеки (Airbag Control Unit, ACU), який здійснює аналіз даних, отриманих від різних датчиків. До таких датчиків належать акселерометри, що вимірюють зміни прискорення транспортного засобу, датчики удару, що розташовані в різних частинах кузова автомобіля, а також гіроскопи, що визначають кутову швидкість [6]. Крім того, система може отримувати дані від бортового комп'ютера автомобіля, зокрема про швидкість руху, положення ременів безпеки та стан сидінь.

Прийняття рішення про активацію подушок безпеки здійснюється на основі аналізу характеру зіткнення, його інтенсивності та напрямку. Для цього використовуються вбудовані алгоритми, що включають фільтрацію сигналів датчиків, порогові значення спрацьовування та математичні моделі аварійних ситуацій. Зокрема, система оцінює параметри прискорення та уповільнення, визначаючи, чи перевищують вони критичні значення, що вказують на серйозну аварію [7].

Додатковим рівнем контролю є механізми перевірки достовірності отриманих даних. Система аналізує кореляцію між сигналами різних датчиків для виключення хибних спрацьовувань через механічні пошкодження, нерівності дороги або вторинні удари. У сучасних автомобілях алгоритми машинного навчання та нейронні мережі використовуються для підвищення точності прогнозування та адаптації до нових типів зіткнень на основі накопичених даних.

Активація подушок безпеки відбувається через модуль керування піротехнічними зарядними пристроями, які ініціюють миттєве наповнення подушок газом. Процес має бути виконаний за мілісекунди, тому система використовує високоенергетичні електричні імпульси для приведення в дію газогенераторів.

1.7 Огляд основних законодавчих правил щодо подушок безпеки

Законодавче регулювання вимог до подушок безпеки в Україні та Європейському Союзі (ЄС) базується на низці нормативно-правових актів, які встановлюють стандарти щодо їхньої наявності, функціональності та перевірки технічного стану.

В ЄС основним документом, що регулює вимоги до технічного стану транспортних засобів, є Директива 2014/45/ЄС Європейського Парламенту та Ради від 3 квітня 2014 року. Ця директива встановлює мінімальні вимоги до періодичних технічних оглядів транспортних засобів, включаючи перевірку системи подушок безпеки. Зокрема, у додатку I до Директиви зазначено, що під час технічного огляду необхідно перевіряти наявність та справність подушок безпеки, а також відсутність індикації несправностей через електронний інтерфейс транспортного засобу. Виявлення таких несправностей може призвести до визнання транспортного засобу непридатним для експлуатації [3].

В Україні вимоги до технічного стану транспортних засобів та їхніх складових частин, включаючи подушки безпеки, регулюються низкою нормативно-правових актів, які гармонізуються з європейськими стандартами.

Зокрема, наказ Міністерства інфраструктури України № 710 від 26 листопада 2012 року "Про затвердження Вимог до конструкції та технічного стану колісних транспортних засобів" встановлює вимоги до систем пасивної безпеки транспортних засобів, включаючи подушки безпеки. Цей наказ передбачає, що транспортні засоби повинні відповідати вимогам відповідних Правил Європейської економічної комісії ООН та актів законодавства ЄС, які містять дані про маркування стосовно перевірки технічного стану транспортних засобів [4].

Крім того, проект Закону України "Про внесення змін до деяких законодавчих актів України щодо приведення у відповідність із актами Європейського Союзу у сфері перевірки придатності транспортних засобів до експлуатації" передбачає обов'язкову перевірку технічного стану транспортних засобів після дорожньо-транспортних пригод, що спричинили зміни в основних елементах, які впливають на безпечність транспортного засобу, зокрема системи подушок безпеки [5].

1.8 Висновки до розділу 1

У межах першого розділу було розглянуто принцип роботи подушок безпеки як ключового елемента пасивної безпеки автомобіля. Описано складові системи — датчики удару, блок керування (ACU), газогенератори та типи подушок — а також алгоритм їхньої взаємодії під час аварії. Основну увагу приділено функціонуванню ACU, який аналізує сигнали сенсорів у реальному часі та ухвалює рішення про активацію подушок залежно від сили, напрямку та умов зіткнення.

Визначено різницю між пасивною та активною безпекою: перша знижує наслідки ДТП, друга — запобігає їх виникненню. Показано важливість інтеграції ACU з іншими системами безпеки та роль адаптивних алгоритмів у підвищенні ефективності захисту. Також проаналізовано архітектуру ACU, включаючи обчислювальні, сенсорні та комунікаційні модулі, і розглянуто законодавчі вимоги ЄС та України щодо контролю технічного стану подушок безпеки. Розділ

підтверджує важливість ACU як центрального елементу в системі захисту пасажирів.

РОЗДІЛ 2 АНАЛІЗ АПАРАТНИХ ЗАСОБІВ РЕЄСТРАЦІЇ КІНЕМАТИЧНИХ ПЕРЕМІЩЕНЬ ТА ПРИСКОРЕНЬ

2.1 Arduino

Arduino є однією з найпопулярніших платформ для розробки вбудованих систем, що пояснюється її гнучкістю, простотою використання та широкою підтримкою з боку спільноти. Вибір саме цієї платформи для реалізації проекту обґрунтовується кількома ключовими аспектами.

По-перше, Arduino є відкритою платформою, що забезпечує доступ як до апаратних, так і до програмних ресурсів. Це дозволяє розробникам адаптувати пристрій під конкретні потреби проекту, використовуючи широкий спектр сумісних модулів та сенсорів. Крім того, простота мови програмування, що базується на C/C++, робить платформу доступною навіть для новачків у сфері електроніки та програмування.

По-друге, важливою перевагою є значний вибір плат та модулів, що дозволяють реалізовувати проекти різного рівня складності. Наприклад, для простих задач можна використовувати Arduino Uno, а для більш складних систем – Arduino Mega або навіть платформи з вбудованими засобами зв’язку, такі як Arduino MKR або Arduino Nano 33 IoT. Це дає можливість масштабувати проект без необхідності змінювати основну платформу.

Ще одним важливим фактором є підтримка великої спільноти користувачів та розробників. Величезна кількість відкритих бібліотек, документації та прикладів значно скорочує час на розробку та налагодження проекту. Крім того, активні форуми та онлайн-ресурси дозволяють швидко знаходити відповіді на технічні питання та отримувати допомогу від досвідчених розробників.

Крім того, вартість компонентів Arduino є порівняно низькою, що робить її економічно вигідним рішенням для реалізації проектів з обмеженим бюджетом.

Багато аналогів можуть бути дорожчими або вимагати додаткових витрат на ліцензування програмного забезпечення, тоді як платформа Arduino є відкритою та безкоштовною у використанні.

Не менш важливим є фактор енергоефективності. Arduino споживає відносно мало енергії, що робить її придатною для автономних систем, які працюють на батарейному живленні. Це особливо актуально для IoT-проектів, де мінімальне енергоспоживання є критично важливим.

Таким чином, Arduino є ідеальним вибором для реалізації проекту, оскільки поєднує у собі простоту використання, широкі можливості розширення, підтримку великої спільноти, економічну та енергетичну ефективність. Саме ці фактори роблять її оптимальним рішенням для розробки вбудованих систем будь-якого рівня складності.

2.2 LSM6DS3 - MEMS

LSM6DS3 є високопродуктивним інерціальним вимірювальним модулем, який поєднує в собі тривісний акселерометр і тривісний гіроскоп. Цей чип, розроблений компанією STMicroelectronics, широко використовується в мобільних пристроях, носимих технологіях, доповненій реальності та промислових сенсорних системах. Його перевагами є низьке енергоспоживання, висока точність і широкий діапазон вимірювань, що робить його ефективним рішенням для багатьох застосувань.

Рисунок 2.2.1 - LSM6DS3

Основні характеристики LSM6DS3 включають широкий діапазон вимірювань акселерометра (±2g, ±4g, ±8g, ±16g) і гіроскопа (±125°/c, ±250°/c, ±500°/c, ±1000°/c, ±2000°/c), що дозволяє налаштовувати модуль відповідно до потреб конкретного застосування. Вбудована пам’ять FIFO на 8 КБ сприяє ефективному збору даних, знижуючи навантаження на основний процесор пристрою. Чип підтримує інтерфейси I2C і SPI, що забезпечує його гнучку інтеграцію в різні електронні системи. Важливою особливістю є функція інтелектуального пробудження, яка дозволяє пристрою залишатися в енергоефективному режимі до виявлення значних змін у русі.

Порівнюючи LSM6DS3 із конкурентами, слід звернути увагу на такі альтернативи, як Bosch BMI270, InvenSense ICM-42688 і Analog Devices ADXL375. BMI270 є подібним рішенням з аналогічним діапазоном акселерометра та гіроскопа, але його FIFO складає лише 6 КБ, а енергоспоживання трохи вище (1,3 мА проти 0,9 мА у LSM6DS3). ICM-42688 пропонує значно більший гіроскопічний діапазон (до ±15 000°/c) і більший FIFO (16 КБ), але водночас має суттєво більше енергоспоживання (2,0 мА). ADXL375 є менш традиційним конкурентом, оскільки орієнтований на вимірювання ударних навантажень і має надзвичайно широкий

діапазон акселерометра (до ±200g), що робить його придатним для промислових і спортивних застосувань, де необхідний моніторинг високих прискорень.

Параметр	LSM6DS3	BMI270	ICM-42688	ADXL375
Акселерометр	±2/4/8/16g	±2/4/8/16g	±2/4/8/16g	±200g
Гіроскоп	±125/250/500 /1000/2000°/c	±125/250/500 /1000/2000°/c	±15 000°/c	-
Споживана потужність	0.9 мА	1.3 мА	2.0 мА	0.1 мА
Інтерфейси	I2C, SPI	I2C, SPI	I2C, SPI	I2C, SPI
FIFO	8 КБ	6 КБ	16 КБ	-

таблиця 2.2.2 - порівняння типів Mems

Перевагами LSM6DS3 перед конкурентами є його висока енергоефективність, інтегрована пам’ять FIFO та широкий набір функціональних можливостей, включаючи підтримку різноманітних алгоритмів для обробки руху. Чип ідеально підходить для мобільних пристроїв, де критично важливе збереження енергії. Його низьке енергоспоживання дозволяє значно подовжити час роботи автономних пристроїв без необхідності частої підзарядки, що є ключовим фактором для новітніх технологій та IoT-рішень.

Серед інших особливостей слід зазначити підтримку вбудованих алгоритмів обробки руху, які дозволяють знизити навантаження на основний процесор, що особливо важливо для мобільних і енергоефективних додатків. Також, LSM6DS3 може працювати у поєднанні з магнітометром для побудови повноцінних дев’ятиосьових IMU-систем, що розширює спектр його використання в робототехніці, дронах та AR/VR-застосуваннях.

Таким чином, LSM6DS3 є одним із найкращих виборів серед сучасних 6-осьових інерціальних сенсорів, що поєднує високу точність, низьке енергоспоживання та широкий функціонал. Його застосування у смартфонах,

новітніх пристроях, промислових сенсорних системах та інших сферах забезпечує високу ефективність збору даних і аналізу руху. Вибір між LSM6DS3 та його конкурентами залежить від конкретних вимог до енергоефективності, швидкості обробки та діапазону вимірювань у конкретному проекті.

2.3 NEO-6M - GPS

NEO-6M є високопродуктивним модулем глобальної навігаційної супутникової системи (GNSS), розробленим компанією u-blox, що широко використовується у навігаційних системах, безпілотних літальних апаратах, транспортних трекерах та рішеннях Інтернету речей (IoT). Даний модуль забезпечує високу точність визначення координат, оптимізоване енергоспоживання та гнучкість інтеграції завдяки підтримці різноманітних комунікаційних інтерфейсів.

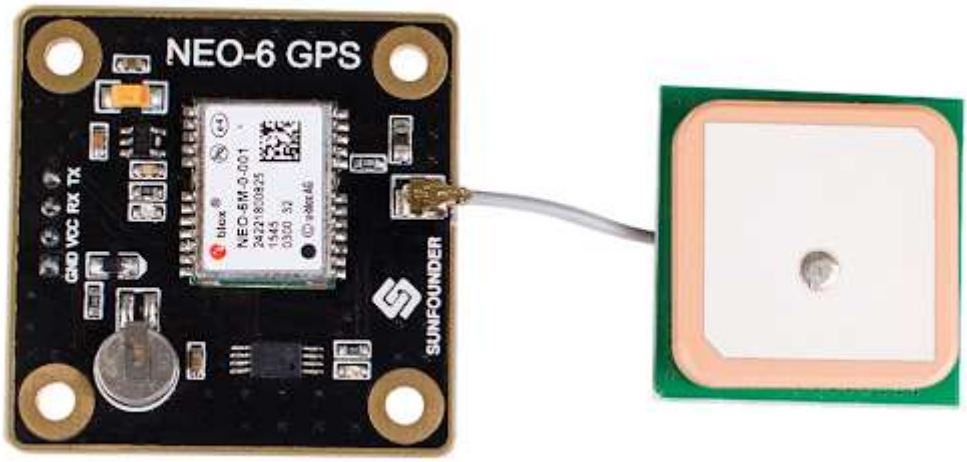

Рисунок 2.3.1 Neo-6 GPS

Основні технічні характеристики NEO-6M включають підтримку глобальної системи позиціонування GPS, що забезпечує точність визначення місцезнаходження до 2,5 метрів у стандартному режимі. Вбудована EEPROM-

пам’ять дозволяє зберігати налаштування конфігурації, що сприяє швидкому відновленню параметрів після перезапуску пристрою. Крім того, підтримка протоколів UART, I2C та SPI забезпечує зручність інтеграції модуля у різні апаратні платформи. Вбудований генератор та низьке енергоспоживання (приблизно 45 мА) роблять NEO-6M ефективним рішенням для автономних систем.

У порівнянні з аналогами, такими як u-blox M8N, Quectel L86 і MediaTek MT3339, модуль NEO-6M має певні переваги та обмеження. Модель M8N забезпечує вищу точність позиціонування (до 1,5 метрів) та підтримує додаткові супутникові системи, зокрема GLONASS і Galileo, що підвищує його універсальність, проте супроводжується дещо більшим енергоспоживанням (50 мА). Модуль Quectel L86 підтримує мультисистемне позиціонування та має компактніші розміри, що робить його оптимальним для мініатюрних пристроїв. MediaTek MT3339 вирізняється низьким енергоспоживанням (до 25 мА) та інтегрованою антеною, що спрощує його використання у малогабаритних пристроях IoT.

Параметр	NEO-6M	u-blox M8N	Quectel L86	MediaTek MT3339
Підтримувані супутникові системи	GPS	GPS, GLONASS, Galileo	GPS, GLONASS	GPS
Точність визначення координат	2,5 м	1,5 м	2 м	3 м
Споживана потужність	45 мА	50 мА	40 мА	25 мА
Інтерфейси	UART, I2C, SPI	UART, I2C, SPI	UART	UART
Вбудована пам’ять	EEPROM	Flash	Flash	EEPROM

таблиця 2.3.2 - порівняння GPS чипів

Основними перевагами NEO-6M є оптимальне співвідношення між точністю визначення місцезнаходження, енергоефективністю та доступністю. Простота

інтеграції модуля забезпечує його ефективне використання у навігаційних рішеннях, що потребують стабільного прийому GPS-сигналу. Наявність EEPROM-пам’яті дає змогу зберігати налаштування без необхідності повторної конфігурації після кожного перезапуску пристрою.

Додатковими характеристиками модуля є підтримка функції енергозбереження, що дозволяє знижувати енергоспоживання у режимах очікування. Крім того, NEO-6M може працювати із зовнішньою активною антеною, що покращує якість прийому сигналу та забезпечує стабільність роботи у складних умовах, таких як урбанізовані території чи густі лісові масиви.

Таким чином, NEO-6M є одним із найефективніших рішень серед сучасних GPS-модулів завдяки високій точності визначення координат, низькому енергоспоживанню та гнучкості у процесі інтеграції. Його застосування у транспортних трекерах, безпілотних літальних апаратах, пристроях IoT та персональних навігаційних системах забезпечує стабільний та надійний збір даних щодо місцезнаходження. Вибір між NEO-6M та його альтернативами залежить від конкретних вимог проєкту щодо точності позиціонування, енергоефективності та підтримки додаткових супутникових систем.

2.4 ADXL377 - Датчик удару

Акселерометр ADXL377 є трьохосьовим аналоговим сенсором від компанії Analog Devices, розробленим для вимірювання прискорень у широкому діапазоні до ±200 g. Цей датчик орієнтований на застосування в умовах, що вимагають високої стійкості до екстремальних навантажень, таких як спортивна біомеханіка, автомобільна безпека та військові дослідження.

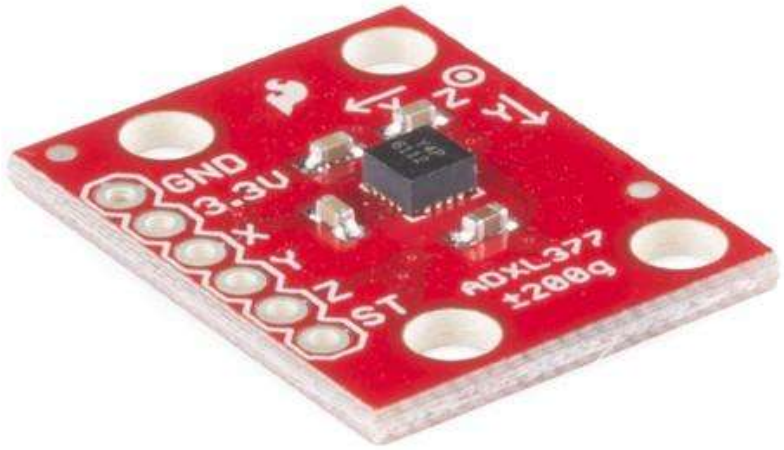

Рисунок 2.4.1 - ADXL377

Основні характеристики ADXL377 включають аналоговий вихідний сигнал, що забезпечує високу швидкість відгуку без затримок, типовий рівень шуму 550 μg/√Hz та низьке енергоспоживання — приблизно 300 мкА при напрузі живлення 3.3 В. Діапазон робочих температур становить від -40°C до +85°C, що робить його придатним для використання у складних середовищах. Його чутливість складає приблизно 6.5 мВ/g при живленні 3.3 В.

Параметр	ADXL377	MPU-6050	LIS331HH
Діапазон (g)	±200	±16	±200
Тип виходу	Аналоговий	Цифровий (I2C, SPI)	Аналоговий
Чутливість (мВ/g)	6.5	2048 (при ±16g)	12
Споживана потужність (мкА)	300	500	220
Рівень шуму (μg/√Hz)	550	400	500

таблиця 2.4.2 - порівняння датчиків удару

У порівнянні з іншими моделями акселерометрів, ADXL377 має суттєву перевагу у вигляді надзвичайно високого діапазону вимірювань (до ±200 g), що дозволяє йому реєструвати дуже різкі зміни прискорення. Це робить його незамінним у застосуваннях, пов’язаних із силовими ударами та високошвидкісною динамікою. MPU-6050, хоча і має нижчий діапазон вимірювань, пропонує цифровий вихід, що спрощує інтеграцію з мікроконтролерами, а також має вбудований гіроскоп, що розширює його можливості. LIS331HH є найближчим аналогом до ADXL377 за характеристиками, проте має вищу чутливість, що може бути корисним для деяких застосувань, але також робить його більш вразливим до шуму.

Головною перевагою ADXL377 є його здатність працювати в умовах екстремальних прискорень без насичення, що робить його ефективним у моніторингу аварійних ситуацій, ударних навантажень і спортивної техніки. Також аналоговий вихід забезпечує мінімальні затримки у передачі сигналу, що є критичним у системах реального часу. Недоліками акселерометра є відсутність вбудованої цифрової обробки сигналу, що потребує додаткового підсилення та обробки зовнішнім контролером, а також порівняно вищий рівень шуму у порівнянні з деякими альтернативами. Крім того, його використання у споживчих електронних пристроях є обмеженим через необхідність більш складної обробки аналогового сигналу.

2.5 Висновки до розділу 2

У рамках даного розділу було проведено аналіз і обґрунтований вибір апаратних засобів для фіксації кінематичних переміщень і прискорень у складі системи активної безпеки. В якості основної платформи обрано Arduino завдяки її простоті, відкритості та широкій підтримці спільноти. Для вимірювання прискорень і обертів використано сенсор LSM6DS3, що поєднує гіроскоп і акселерометр із низьким енергоспоживанням. Для точної навігації застосовано

модуль GPS NEO-6M, який забезпечує стабільне позиціонування та легку інтеграцію.

Реєстрація сильних ударів виконується за допомогою ADXL377 з діапазоном до ±200g, що дозволяє фіксувати аварії з високим навантаженням. Порівняння з аналогами підтвердило доцільність вибору зазначених компонентів, враховуючи їх точність, швидкодію та енергоефективність. Сукупність розглянутих рішень забезпечує технічну готовність системи до роботи в реальному часі та підвищує її надійність у критичних ситуаціях.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій реалізації

Вибір технологій для реалізації програмного модуля є ключовим аспектом у процесі проектування та розробки інформаційних систем. Технологічний стек повинен включати інструменти та мови програмування, які забезпечать високу продуктивність, безпеку, масштабованість та ефективність роботи всієї системи.

Python є потужною високорівневою мовою програмування, яка широко використовується для розробки серверних додатків завдяки своїй простоті, зручному синтаксису та великій екосистемі бібліотек [9]. Однією з ключових переваг Python є його зручність для швидкої розробки та підтримки коду, що робить його ідеальним вибором для серверної частини інформаційних систем. Python підтримує багатопоточність, асинхронне програмування (наприклад, через asyncio) та інтеграцію з іншими мовами програмування [10]. Його бібліотеки та фреймворки, такі як Flask і Django, забезпечують зручну розробку API та бекенд-логіки. Окрім цього, мова має вбудовані засоби захисту, а також підтримує бібліотеки для шифрування, управління аутентифікацією та безпекою системи [11]. Завдяки кросплатформеності Python може бути розгорнутий на різних операційних системах без необхідності значних змін у коді.

SQL є стандартною мовою для роботи з реляційними базами даних. Використання SQL дозволяє забезпечити ефективне управління даними, їх збереження, пошук та обробку [12]. SQL-бази даних, такі як PostgreSQL або MySQL, дозволяють працювати з великими обсягами структурованої інформації. Крім того, використання механізмів аутентифікації, резервного копіювання та реплікації підвищує рівень безпеки даних [13]. Сучасні системи управління базами даних підтримують індексацію, кешування та партиціювання таблиць, що оптимізує швидкість виконання запитів. Важливим аспектом є також масштабованість, що дозволяє ефективно працювати з великими обсягами даних та підтримувати як горизонтальне, так і вертикальне масштабування [14].

C++ є потужною мовою програмування, що забезпечує високий рівень контролю над пам’яттю та продуктивністю. Використання C++ є доцільним для розробки низькорівневого програмного забезпечення, яке взаємодіє безпосередньо з апаратними компонентами [15]. Завдяки високій продуктивності C++ дозволяє створювати оптимізований код, що працює швидше за аналогічні рішення на інших мовах. Ця мова забезпечує прямий доступ до пам’яті за допомогою вказівників, що дає можливість розробникам ефективно керувати ресурсами [16]. Використання C++ дозволяє створювати стабільні та надійні системи з низьким рівнем затримки, що є критично важливим у високопродуктивних та вбудованих системах. Завдяки широкому спектру можливостей C++ залишається однією з основних мов програмування для роботи з апаратним забезпеченням та оптимізації ресурсів

3.2 Архітектура проекту.

Архітектура проекту складається з 3-х основних компонентів, якими є

SQL сервер, в якому зберігаються всі дані, дивіться таблицю 3.2.1, для подальшої передачі та обробки.

Name	Type
Car_crash_ID	INT
Day_of_week	INT
Hour_of_crash	FLOAT
Severity	TEXT
Speed_limit	INT
Midlock	BOOLEAN DEFAULT False
Intersection	BOOLEAN DEFAULT False
Road_position_horizontal	TEXT
Road_position_vertical	TEXT
Road_slope	BOOLEAN DEFAULT False
Road_wet	BOOLEAN DEFAULT False
Weather	TEXT
Crash_type	TEXT
Lighting	TEXT
Traffic_controls	TEXT
Fatalities	INT DEFAULT 0
Serious_injuries	INT DEFAULT 0
Minor_injuries	INT DEFAULT 0

Type_of_vehicles	TEXT
Total_of_vehicles	INT

таблиця 3.2.1 - типи даних

Ці дані можна поділити на наступні категорії:

а. Час, в який стається аварія
б. Стан та позиція дороги на місці аварії
в. Погодні умови
г. Серйозність аварії
д. Машини, які були у ДТП

Далі ці данні будуть запитуватися сервером на основі програмної мови Python. Основною задачею цього сервера є проведення статичного аналізу на основі отриманих даних. Після отримання 2 точок GPS координат, після чого він починає побудову маршруту, поділяючи один великий маршрут, на невеликі відрізки з певними позначеннями. Після чого він збирає для кожної окремого позначення погодні умови коли машина наближається на відстань у 2 метри від цієї точки. Після цього за допомогою статичного аналізу знаходиться вісім найбільш вірогідних варіантів, якщо відбудеться аварія. Після чого дані передаються без черги в ACU.

Коли ACU приймає дані, вона зберігає їх в тимчасовому місці зберігання. І після того, як один з 6 акселерометрів, які виступають у формі датчиків удару, засікають різке змінення акселерації ACU починає обрахування потрібних дії. За основу ACU бере дані по критеріях: швидкість, за якою відбулася аварія, пристебнутість ременів безпеки, сила удару у нютонах, і також ще по декількох інших критеріях. ACU також може переходити в стрес мод, якщо сервер перестав передавати дані, чи передав дані які не задовольняються для цієї ситуації. У випадку перегортання автомобіля всередині вбудований гіроскоп, який при фіксації різкого збільшення нахилу автомобіля, з силою де перегортання буде неминучим, буде приймати рішення про випуск всіх подушок безпеки.

3.3 Опис принципів роботи бази знань

У таких країнах як США, Німеччина, Велика Британія та Франція неможливо отримати інформацію, необхідну для проекта, без додаткового запиту. Тому обрано наповнювати базу знань інформацією з двох країнах: Нова Зеландія та Австралія, бо вони мають більш вільний доступ до потрібної інформації.

На даний момент, база знань використовує тільки один не оновлюваний файл. Коли ця технологія стане більш поширеною та більш застосовуємою, з'явиться можливість отримувати інформацію напряму з автомобіля (звісно прийдеться враховувати той випадок коли автомобіль був пошкоджений до стадій неможливості відправляти дані). Другий варіант це збір інформації напряму з поліцейських джерел, у вигляді електронних даних, якщо є така можливість, чи зчитування за допомогою Optimal Character Recognition (OPR). Але це означає додавання більшої кількості інформації, додатково до тієї яка є в таблиці 3.2.1. Наприклад специфікація сторони дорожнього руху, та специфікація розташування руля в автомобілі.

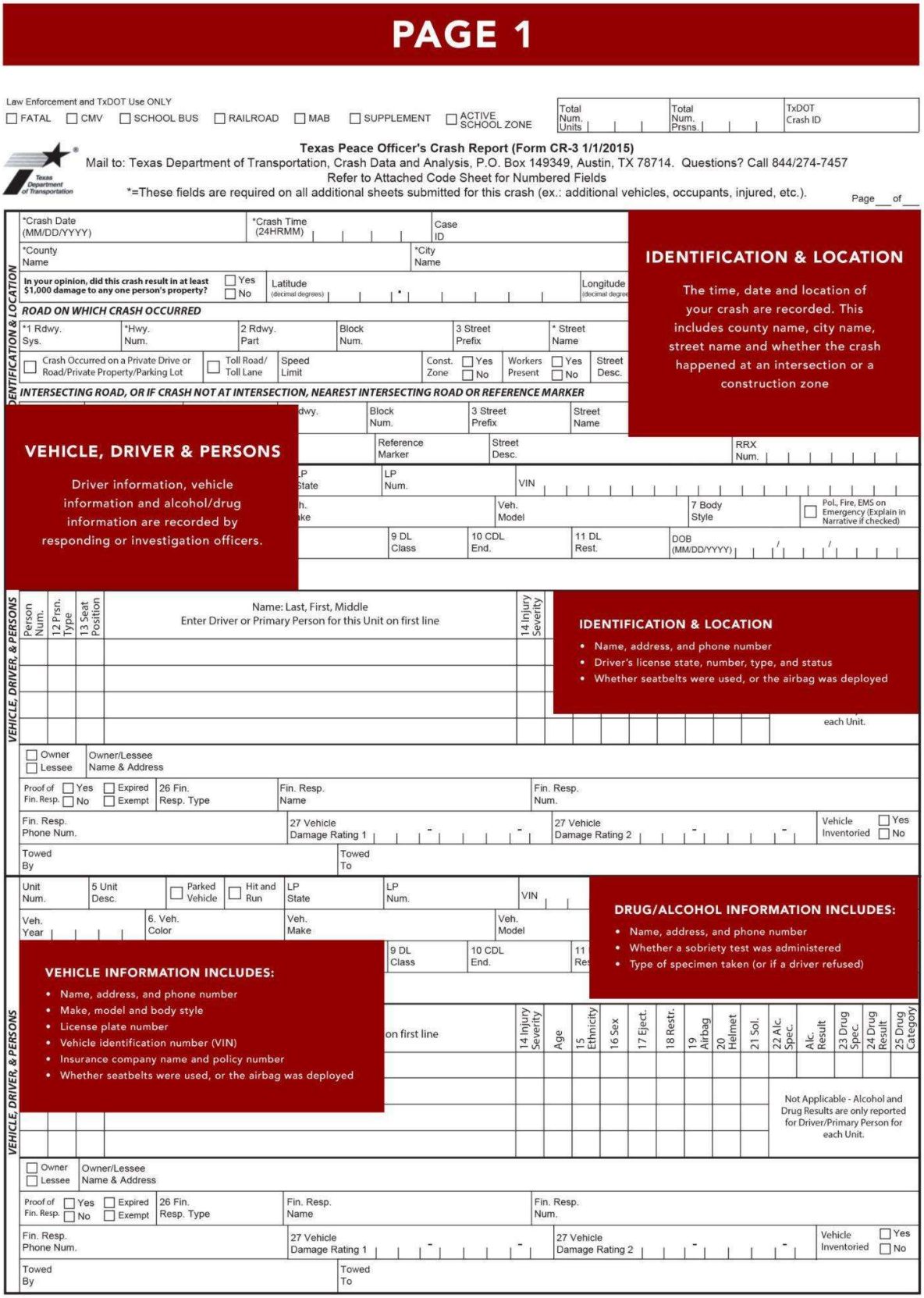

Рисунок 3.3.1 - Приклад поліцейського ДТП звітності

Було обрано виконати обробку даних за допомогою Python на хмарних серверах. Python було обрано через його гнучкість та можливість обробляти велику кількість даних. Його недолік - те, що середовище виконання є інтерпритатором,

тому він не є дуже швидким. Хмарні сервера були обрані, замість прямого вбудовування в систему ACU з наступних причин:

а. можливість збільшити потужність та кількість зберігаємих даних для серверів. Також це означає що нам не потрібно буде оновлювати кожен окремий ACU, коли ми отримуємо нові дані, а тільки отримувати декілька, чи один, сервер.
б. зменшити навантаження і вартість кожного окремого ACU, хоча через це зменшиться швидкістю та надійністю, бо тепер в нас є 2 частини які можуть не працювати так, як заплановано. Також ми не зможемо гарантувати отримання даних на чіп в будь якій ситуації.
в. можливість легко розділити сервера для отримання більш швидких результатів оброблених даних. Це також дає можливість збирати більш даних локально, з поліцейських установ.

3.4 Опис принципів роботи серверної частини

3.4.1 Отримання GPS координат та обробка даних для знаходження деталей дороги

Після отримання координат, розпочинається процес знаходження специфікації дорожніх подій, за допомогою OpenRouteService та бібліотеки osmnx й networkx. Спочатку обчислюється дорога за OpenRouteService, дані отримуються у вигляді двох списків, перший координатний, де перечислюються координати події. Другий отримується у вигляді повного опису події, звідки виокремлюється тільки тип події у вигляді номеру.

Після чого знаходяться координати для midlock, які додаються до основного списку дорожніх події, з їх відповідними координатами. Після чого типи події перетворюється в їх string еквіваленти. Таким чином події під номером таблиця №3.4.1.1.

Номер події	String еквіваленти
6,11,12,13	"straight road"
0,2,4	"turn left"
1,3,5	"turn right"

7,8	"roundabout"
9	"u-turn"

таблиця №3.4.1.1 - абревіатура події

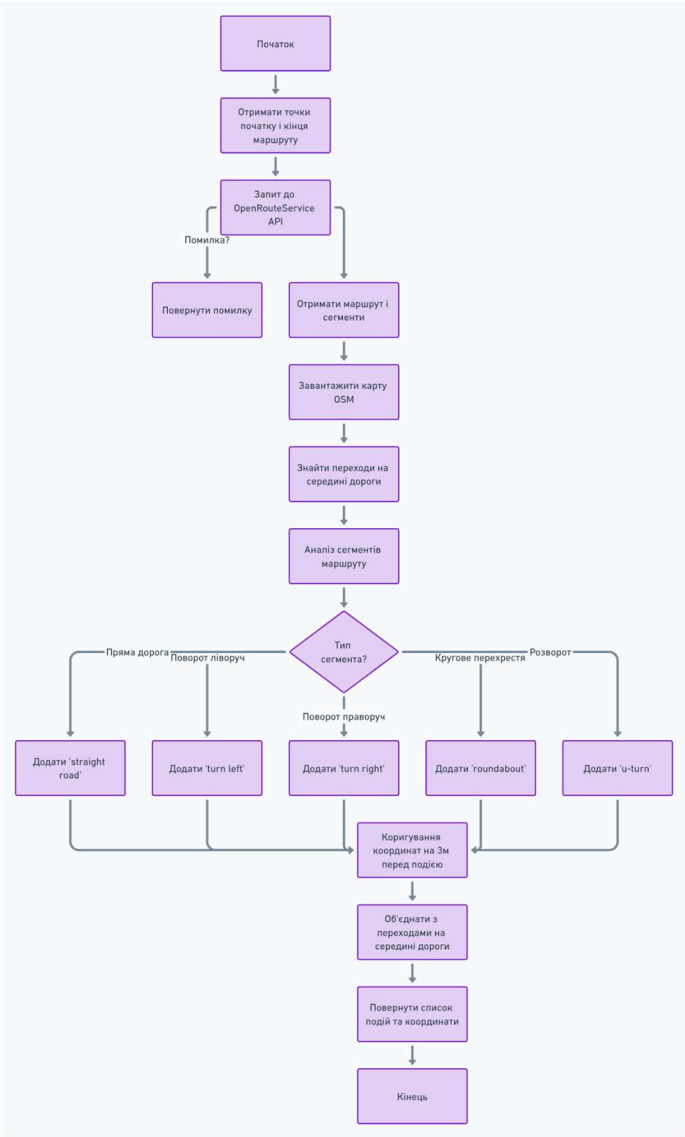

Рисунок 3.4.1.2 - Блок схема принципів роботи серверної частини

Після чого всі координати події зсуваються на 2м в сторону попередньої події, для отримання часу на обрахування події та коригування похибки GPS.

3.4.2 Збір даних про погодні умови

Для отримання погодних умов використовується відкриті API Open-Meteo. Для цього задіяні 2 процеси:

Перший повертає стан освітлення у вигляді string. Для цього він звертається до API Open-Meteo із запитом на надання часу для заходу та сходу сонця. Після чого він дивиться на те який зараз час доби, світанок, сонячне світло, чи захід сонця. І відправляє відповідний string

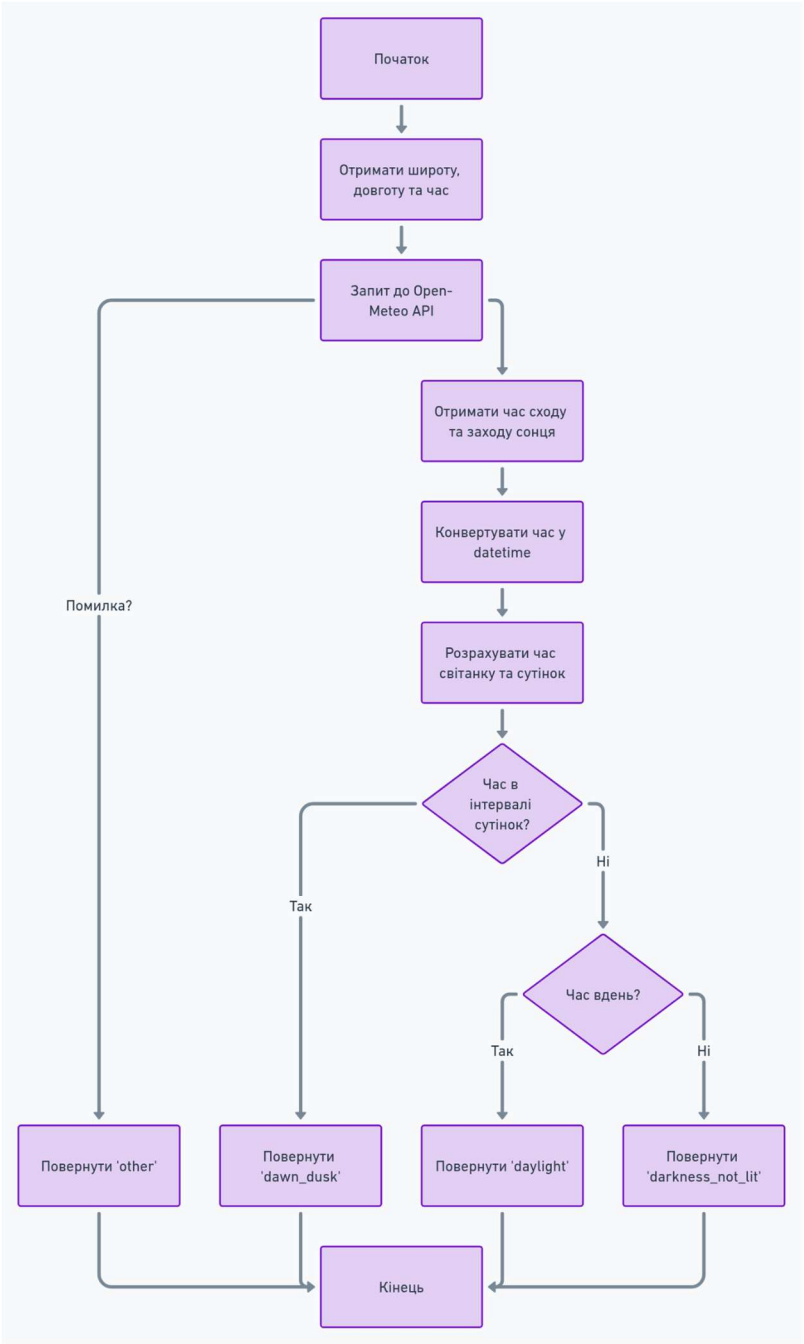

Рисунок 3.4.2.1 - Блок схема збір даних про світлові умови

Other, повертаються у випадках коли неможливо отримати відповідь від сервера, чи стається помилка при обробці даних. У цьому випадку вона слугує заміною для None в базі знань.

Другий процес відбувається для отримання безпосередніх погодних умов. Таких як погода, та мокрість дороги. Для цього робиться ще один запит до Open-Meteo. Але в даному випадку Open-Meteo використовує код погоди, такі як 0 - це чисте небо, а 45 - туман. Після отримання кодів на весь день, відбується їх розшифрування та запис в списки. Після чого відбувається перевірка на те, чи є дорога мокрою в даний момент часу.

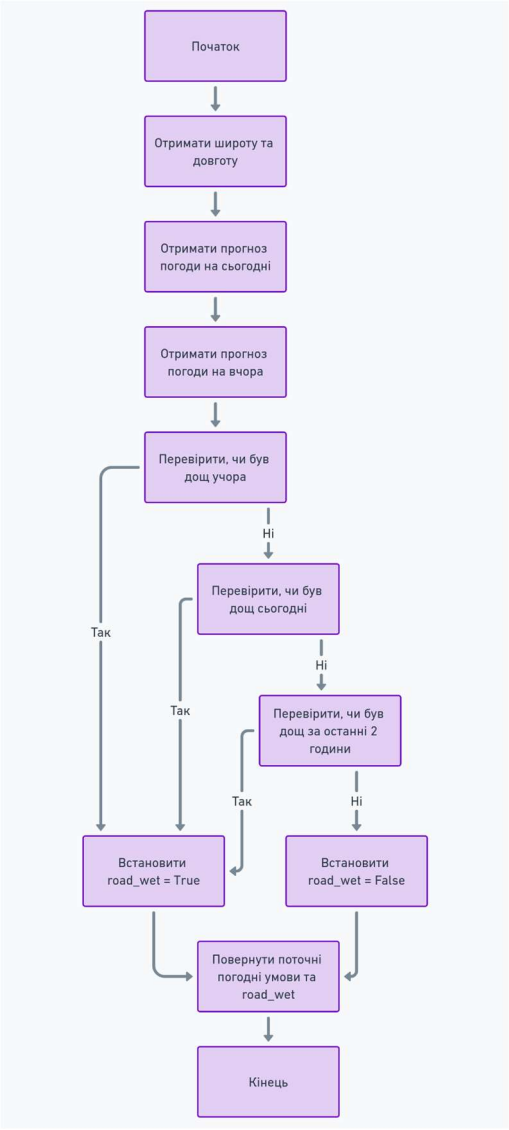

Рисунок 3.4.2.1 - Блок схема збір даних про погодні умови

Це відбувається за допомогою звичайних if-else стейтменів, які перевіряють чи на даний момент йде дощ, чи слабкий дощ відбувався протягом останніх 2-х годин, чи сьогодні був сильний дощ. І після чого повертає 2 значення у формі string та boolean. Якщо відбувається помилка зі з'єднанням з Open-Meteo, назад за повертається значення “Overcast” та False.

3.4.3 Отримання даних з бази знань

Отримання даних відбувається за наступними критеріями: день тижня та година дня, коли відбулася аварія, умови освітлення, погодні умови та мокрість дороги, та дорожня подія. Дорожня подія може бути: straight road - у такому випадку перехрестя та midlock значення буду дорівнювати нулю. turn - де тільки значення перехрестя буде дорівнювати одному, mid-lock буде дзеркальне відображення значень для turn. Та roundabout й u-turn, де ці два значення будуть дорівнювати одному. У інших випадках, ACU буде відправлений в стрес мод.

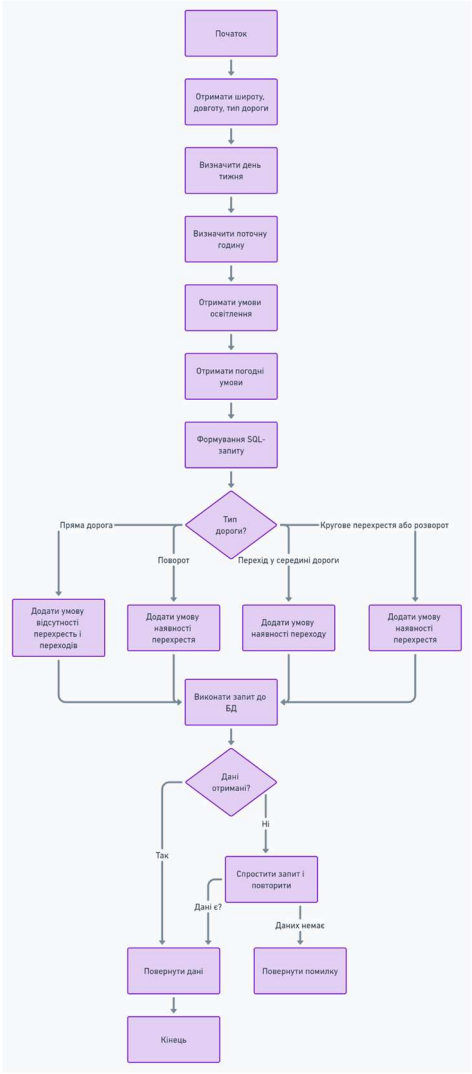

Рисунок 3.4.3.1 - Блок схема запиту до бази знань

Після чого може відбутися до 3-х викликів до бази знань, для того щоб упевнитися про достатню кількість даних. При кожному наступному виклику, буде віддаватися менш важливі компоненти виклику. В першому разі відкидається

година в якому відбулася аварія, як менш важливий компонент. В другому разі відкидається день тижня, після чого всі дані які були отримані при останньому виклику.

3.4.4 Обробка даних

Після отримання даних починається перший етап з обробки даних в якому відбувається наступне:

По перше, відбувається перевірка на кількість даних, що поступили для статистичного аналізу. Якщо за будь якої причині дані не були отримані, чи їх немає, тоді до ACU передається тільки None/Null, який приведе його в стресс мод.

Якщо ж є достатня кількість даних, то відбувається витяг інформації у вигляді словника, що дозволяє більш простому визначенню ймовірності в подальшому, через те що вони виступають як точки збору для даних які повторюються. Ці дані (кількість і тяжкість поранених, максимальна швидкість на дорозі, де відбулася аварія, важкість аварій, світлофори, та інше) розподілені по різним відповідним словникам.

Після створення словників, які будуть тримати ці дані, відбувається очистка даних він Null значень та значень швидкості які будь менше за 20 кілометрів на годину. В останню чергу відбувається сортування кожного словника, для подальшої обробки статистичним методом.

Обробка даних відбувається в два кроки, перший крок визначає кількість різних ступенів, так як це виступає як основний показник для подальшої обробки даних. Другий це проведення статистичного аналізу для отримання подальших інструкції, які буду передавати до ACU. Блок схему цього процесу можна побачити на рисунку 3.4.4.1

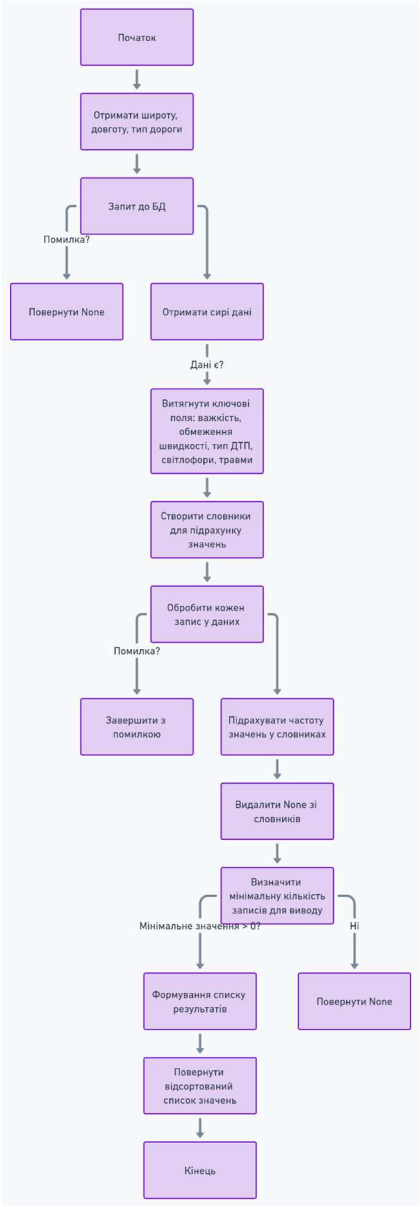

Рисунок 3.4.4.1 - блок схема статистичного аналізу

Для отримання інструкції існує окрема функція сервера, яка спочатку визначає групу зіткнення, всі групи зіткнення можна побачити на таблички 3.4.4.2. Після чого для кожного окремого удару визначається група зіткнення, призначається інструкція. Яка вже буде передається на ACU частину, коли автономно дістанеться до наступної точки.

Назва групи зіткнень	Значення в групі
right_angle_collision – зіткнення під прямим кутом	Прямий кут, Прямий кут з паркуванням, Прямий кут під час

	в'їзду/виїзду з подвір'я, Прямий кут під час обгону, Прямий кут з пішоходом
rear_end_collision – зіткнення ззаду	Зіткнення ззаду, Зіткнення ззаду з паркуванням, Зіткнення ззаду під час обгону, Зіткнення ззаду під час в'їзду/виїзду з подвір'я, Зіткнення ззаду з пішоходом, Зіткнення ззаду з твариною
side_swipe_collision – бокове зіткнення	Бокове зіткнення, Бокове зіткнення під час обгону, Бокове зіткнення з паркуванням, Бокове зіткнення під час в'їзду/виїзду з подвір'я, Бокове зіткнення з твариною, Бокове зіткнення з пішоходом
head_on_collision – лобове зіткнення	Лобове зіткнення, Лобове зіткнення під час обгону, Лобове зіткнення під час в'їзду/виїзду з подвір'я, Лобове зіткнення з твариною, Лобове зіткнення з паркуванням, Лобове зіткнення з пішоходом
right_turn_collision – правий поворот	Правий поворот, Правий поворот під час в'їзду/виїзду з подвір'я, Правий поворот з пішоходом, Правий поворот з паркуванням, Правий поворот під час обгону
left_road_out_of_control – втрата контролю на дорозі	Виїзд з дороги - втрата контролю
hit_pedestrian – наїзд на пішохода	Наїзд на пішохода, Наїзд на пішохода з участю пішохода, Наїзд на пішохода з паркуванням, Наїзд на пішохода під час обгону, Наїзд на пішохода з твариною
hit_fixed_object – наїзд на нерухомий об'єкт	Наїзд на нерухомий об'єкт, Наїзд на об'єкт з паркуванням, Наїзд на об'єкт з

	пішоходом, Наїзд на об'єкт під час в'їзду/виїзду з подвір'я, Наїзд на об'єкт з твариною
hit_object – наїзд на об'єкт на дорозі	Наїзд на об'єкт на дорозі
hit_parked_object – наїзд на припаркований транспорт	Наїзд на припаркований транспорт
hit_animal – наїзд на тварину	Наїзд на тварину, Зіткнення з твариною, Наїзд на тварину з участю тварини, Наїзд на тварину з паркуванням
rollover_collision – перекидання	Перекидання, Перекидання без зіткнення

Таблиця 3.4.4.2 - групи зіткнень

3.5 Опис принципів роботи ACU частини

3.5.1 Отримання та передача даних з серверною частиною

Було розглянуто два способи обміну даними з серверною частиною, а саме:

SIM800L — це GSM/GPRS модуль, який працює в чотирьох діапазонах частот (850/900/1800/1900 МГц), що дозволяє використовувати його в більшості країн світу. Модуль підтримує передачу голосу, SMS та даних через GPRS зі швидкістю до 85.6 кбіт/с. Для роботи SIM800L потребує microSIM-карти та живлення в діапазоні 3.8–4.2В. Модуль має компактні розміри (25 х 23 мм) та низьке енергоспоживання в режимі сну (<2 мА), але під час передачі даних споживання може досягати до 2 А. SIM800L широко використовується в проектах з Arduino та інших мікроконтролерах завдяки простому UART-інтерфейсу та підтримці AT-команд.

Рисунок 3.5.1.1 - SIM800L

Iridium 9602: Глобальний супутниковий передавач. Який представляє собою супутниковим модулем, призначений для передачі коротких повідомлень (Short Burst Data, SBD) через глобальну супутникову мережу Iridium. Це дозволяє забезпечити зв'язок у будь-якій точці земної кулі, включаючи віддалені регіони без покриття стільниковими мережами. Модуль підтримує передачу повідомлень розміром до 340 байтів (від пристрою) та до 270 байтів (до пристрою) [17]

Він не вимагає використання SIM-карти, що спрощує його інтеграцію у вбудовані системи. Модуль має компактні розміри (41 x 45 x 13 мм) та вагу 3 г, що робить його придатним для вбудованих рішень. Він також оснащений пасивним GPS-портом для спільного використання антени з GPS-приймачем

Рисунок 3.5.1.2 - Iridium 9602

Iridium 9602 забезпечує глобальне покриття, що робить його ідеальним для застосувань у віддалених або важкодоступних місцях, де відсутнє стільникове покриття. Через це можна зменшити кількість центрів обробок даних. Проте, його функціональність обмежена ціною винаймання або запуску супутникової мережі. Крім того, вартість супутникового зв'язку зазвичай вища порівняно зі стільниковими мережами.

З іншого боку, SIM800L пропонує ширший спектр функцій, включаючи голосові виклики, SMS та передачу даних через GPRS. Але, це хоч і є перевагою, більшість цього функціоналу не буде використовуватися через потребу предтечі саме цифрових даних. Його використання обмежене зонами з покриттям стільникових мереж, але вартість передачі даних зазвичай нижча. Модуль також має нижчу вартість та енергоспоживання, що робить його привабливим для масових та енергозалежних застосувань.

Зі сторони ACU з початку передається дві пари значень у вигляді double, для збільшення точності GPS. Хоч це і збільшує в два рази кількість даних які потрібно передавати, але навіть при можливості передачи 340 байтів, досі можливо передати до 21 пар GPS координат. після чого відбувається постійне передача даних з GPS до серверної частини.

Сам же сервер відправляє вісім integer-ів, які представляють собою набір інструкції, зменшених для швидкої передачі та використання даних. У випадку якщо ACU не отримує відповіді від сервера, або сервер не зможе отримати GPS координати від ACU, ACU переходить в стресовий режим.

Структура інструкції яка передається може бути побаченою на рисунку 3.5.1.1. Позначення кожного окремого елемента описано у таблиці 3.5.1.2

Рисунок 3.5.1.1 - Структура інструкції

Цифровий формат	Позначення
11	Лобове зіткнення - два найбільші значення прийшлися на перед автомобіля
21	Лобове праве зіткнення - два найбільші значення прийшлися на передній правий та правий датчики автомобіля
34	Заднє праве зіткнення - два найбільші значення прийшлися на задній правий та правий датчики автомобіля

44	Заднє зіткнення - два найбільші значення прийшлися на багажник автомобіля
43	Заднє ліве зіткнення - два найбільші значення прийшлися на задній лівий та лівий датчики автомобіля
12	Лобове ліве зіткнення - два найбільші значення прийшлися на передній лівий та лівий датчики автомобіля
99	Використовується у 2 випадках - Перший при перегортанні автомобіля. Другий - коли не вийшло визначити тип удару, тобто невідомий напрямок удару

Таблиця 3.5.1.2 - позначення сторін удару

На даний момент через обмеженості можливостей, дані передаються за допомогою .txt файлів, з семантикою, схожою на передачу у вищенаведених чипах.

3.5.2 Застосування даних, які були отримані з серверної частини

Після отримання даних з серверної сторони вони зберігаються в тимчасовій пам'яті ACU, до моменту отримання нових даних. Коли датчики стресу отримують сигнал про зміну акселерації, ці дані надходить до процесора, і він починає процес обробки інформації в наступному порядку:

Перевірка перегортання автомобіля - цей процес відбувається постійно. Для цього він використовує чип LSM6DS3 в якому є вбудований гіроскоп, який і допомагає виявити перегортання автомобіля за допомогою визначення швидкості нахилу. Поріг перегортання є унікальним і залежить від типу автомобіля, його конструкції, ваги та інших параметрів. Значення деяких порогів перегортання представлені у таблиці 3.5.2.1. Якщо буде виявлено перегортання автомобіля, то одразу будуть випущені всі подушки безпеки, для збільшення шансів виживання пасажирів в автомобілі[18]. При випуску подушок безпеки під час перегортання

автомобіля шанси критичних ушкоджень пасажирів через викид з автомобіля зменшуються приблизно на сорок п’ять відсотків [19].

Якщо ж перегортання автомобіля не виявлено, то в автомобілі відбуваються перевірки параметрів які потребуються для випуску подушок безпеки. Перший з параметрів це швидкість автомобіля, ЄС забороняє випуск подушок безпеки якщо автомобіль не перевищував швидкість 20 км/годину. Другий параметр, який потребує для активації подушок безпеки - це пристебнутість пасків безпеки, що забезпечує зменшення кількості травм при аварії. Без цієї перевірки кількість важких переломів, а саме переломів голови, шиї та грудної клітини збільшуються на п'ятдесят відсотків [20]. Тому там де пасок безпеки не був позначений як пристебнутий, не будуть випущені подушки безпеки, при будь якому випадку крім перегортання автомобіля. Бо випуск подушок збільшить шанси виживання всіх пристебнутих пасажирів. То остання, але не менш важлива перевірка. Ця перевірка була зроблена для зменшення кількості випуску подушок безпеки, де можна обійтися тільки ременями безпеки та каркасом автомобіля. І вона залежить більше від самого автомобіля, бо кожен автомобіль має різну будову рами, яка заточена під різні кути та сили удару. ACU визначає це за допомогою шести датчиків удару ADXL377. Два з яких розташовані спереду машини, по одному з кожної із сторін автомобіля, і два датчики в бампері автомобіля. Їх розташування можна побачити на рисунку 3.5.2.2, де вони буду позначені червоним кольором. Вони працюють по принципу додавання двох найбільших значень, що дасть окрім розуміння сили удару, також розуміння напрямку ударів, і порівняння його до порогу де потрібно активація подушок безпеки.

Тип транспортного засобу	Поріг швидкості крену (°/c)	Поріг швидкості висоти (°/c)
Sedan / Coupe	50 - 80(°/c)	30 - 50(°/c)
SUV / Truck	40 - 60(°/c)	20 - 40(°/c)
Off-Road / High COG	30 - 50(°/c)	15 - 30(°/c)

Таблиця 3.5.2.1 – пороги перегортання транспорту

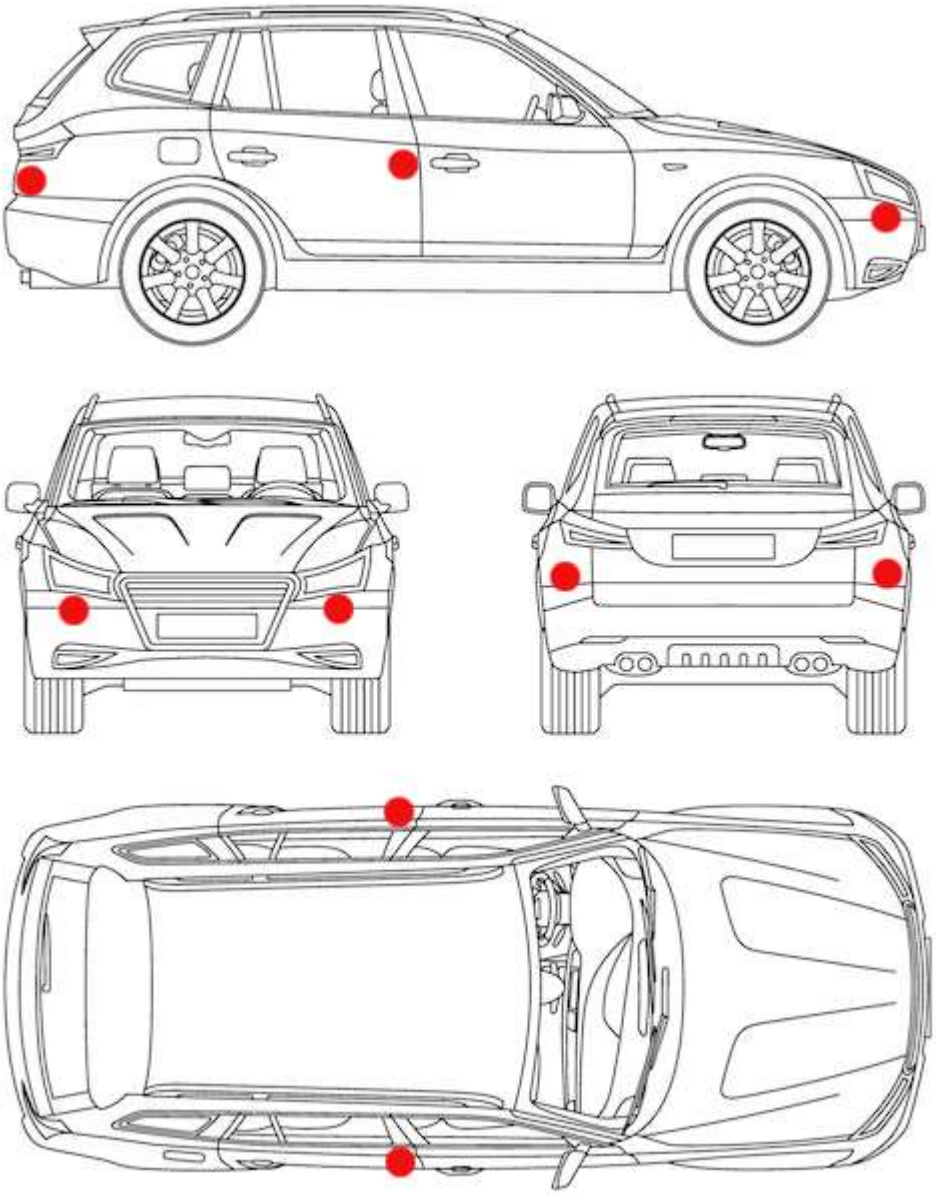

Рисунок 3.5.2.2 - розташування датчиків удару

Якщо всі ці перевірки було пройдено, то система почне передивлятися варіанти які були запропоновані сервером, одразу відкидаючи всі варіанти які не підходять по напряму удару. Це можливо зробити швидко та ефективно через структуру інструкції, яка зберігається в пам'яті чипу. Для цього він розділяє номер як було показано на рисунку 3.5.1.1. Якщо ж будь який з варіантів які були надані сервером не підходить то чип переходить в стрес мод, і починає працювати з даними які в нього є. Цей випадок буде розглянути в подальшому. Якщо ж один з варіантів підходить під ситуацію, то чип починає процес застосування інструкції.

Для цього він використовує другу частину інструкції, для поступового випуску газів в подушки безпеки, для того що вони відкрилися. Повний процес займає менше ніж 200 мілісекунд, від початку колізій до випуску подушок безпеки.

3.5.3 Огляд стрес моду

Стрес мод активується в двох випадках. Перший з яких це втрата зв’язку з сервером, тобто коли чип не можу отримати інструкції від сервера. В такому випадку він прийде одразу в стрес режим, в якому він і буде залишитися до відновлення зв'язку з сервером та отримання інструкції. Другий же випадок відбувається коли жодна з інструкції не підходить як по напряму удару так, можливо, і параметрам удару, тобто всі інструкції були відкинуті чипом під час аварії з іншим автомобілем.

Сам по собі стрес мод представляє собою стандартний протокол для зіткнення в автомобілі, які може бути знайдений в будь з якому ACU схожого типу та застосунку. Він управляється по наступним крокам:

Перший крок - це визначення з якої сторони приходиться удар по автомобілю. Ці дані вже будуть обраховані під час відкидання варіантів якщо вони є, і чип не перейшов в стрес мод через втрату зв'язку з сервером, у випадку втрати зв'язку з сервером чип обробить в першу чергу саме напрям удару. Це, як і було зазначено вище, він робить за допомогою чіпів ADXL377. Це відбувається наступним чином - Після різкої зміни напряму швидкості автомобіля, чипи засікають зміну вектора, і вираховують силу удару в джоулях за допомогою формули яка зображена на малюнку 3.5.3.1.

$$\mathrm{F} = \left(\frac{\sqrt{(\mathrm{v_x} - \mathrm{v_0})^2 + (\mathrm{v_9} - \mathrm{v_0})^2 + (\mathrm{vz} - \mathrm{v_0})^2}}{\mathrm{s}} \right) g$$

Рисунок 3.5.3.1 – формула для розрахунку сили удару

Другий крок - це перевірка необхідних значень. Тобто чи швидкість менша за 20 км/г та перевірка чи сила удару перевищує значення в 5000 Дж. Мінімальна швидкість було обрана на основі законодавства ЕС, де це встановлена мінімальна

швидкість для автомобіля що вважати аварію серйозною для випускання подушок безпеки. Мінімальна сила удару була обрана зважаючи на мінімальну середню стійкість каркасу автомобілів, для витримування аварії з будь якої сторони, без ушкоджень для пасажирів.

Третій крок - це перевірка датчиків пристебнутості пасків безпеки. Цей крок слугує як функція перевірки безпечності та, в якійсь мірі, наявності пасажирів, на місцях. Тобто кожного разу коли ми звертаємося до функції ми отримує п’ять bollian значень, які відповідають пристебнутості пасків безпеки. Там де ми отримуємо значення True, ми робимо наступне твердження - “На цьому місці точно присутня людина, і для неї безпечно випускати подушки безпеки”. Там де ми отримуємо значення False, ми припускаємо наступне - “На цьому місці не присутня людина або ця людина присутня на цьому місці, але небезпечно випускати подушки безпеки”.

Четвертий, і фінальний крок = це знаходження оптимального шляху випуску подушок безпеки, для зменшення кількості ушкоджень для пасажирів, та для зменшення травм під час аварії. Це, як і було зазначено вище, робиться на основі всіх даних, отриманих на попередніх кроках.

3.5.4 Огляд принципів прийняття рішень щодо випуску подушок

Програма може ігнорувати інструкції в декількох випадках:

Перший з яких - це швидкість автомобіля перевищує межу в 150 км/год. В такому випадку вважається, що якщо ремені безпеки дозволяють це, то випустити всі подушки безпеки одночасно. Бо під час зіткнення на такій швидкості, в більшості випадків після першого удару відбувається, як мінімум один додатковий удар. Від залишку енергії автомобіля, який змусить його продовжувати рух в тому ж, або зміненому напрямку протягом певного часу.

Другий, це випуск передньої подушки безпеки. Коли зажатий клаксон, то при не швидких аваріях, до 40 км на годину, може не бути випущена передня подушка безпеки. Це відбувається з оглядом на те, що в аварії на більш малих швидкостях,

з меншою кількістю енергії яка буде передаватися з автомобіля в автомобіль, це може призвести до більш важких ушкоджень, ніж це було б без використання передньої подушки безпеки.

У випадках коли було отримано інструкції з сервера, відбувається пряме застосування подушок безпеки за допомогою команди pop, який вибирає по одному номеру та викидає його з інструкції, залишаючи тільки наступні кроки. Одночасно з чим, посилаючи електронний сигнал до запалювача подушок безпеки, який при отриманні цього сигналу випускає гази для розкривання подушок безпеки.

Але коли немає способу отримання даних з сервера, і ACU переходить в стресовий мод, пріоритизація йде наступним чином: врятувати від якомога більшої кількості травм, і не додавати нових. Тому на даний момент система працює не на певному автомобілі, тому що це б включало додавання більш складних схем, і їх підведення під певний бренд та тип автомобіля. Через те що кожен бренд має свої стандарти, які можуть суттєво відрізнятися від іншого. А на більш усередненому понятті автомобіля, який був зібраний з регуляцій та усереднених даних з декількох різних варіантів SUV автомобіля. Позначення нумерації подушок безпеки можна побачити в таблиці 3.5.4.1. Якщо одна, чи декілька подушок безпеки відсутні в автомобілі, вони просто не будуть застосовані, тобто не буде посланий електричний сигнал для випуску подушок безпеки.

Позначення подушки безпеки	Значення позначення
0	Не випускати подушки безпеки, зазвичай використовується тільки одне позначення
1	Передні подушки безпеки
2	Бокові подушки безпеки
3	Віконні подушки безпеки
4	Подушки безпеки для колін
5	Задні подушки безпеки

6	Подушки безпеки на ременях безпеки
7	Центральні подушки безпеки
8	Подушки безпеки для пішоходів
9	Відкрити подушки безпеки, використовується тільки одне позначення

Таблиця 3.5.4.1 - позначки для випуску подушок

3.6 Функціональне та модульне тестування програми

Так як програма немає GUI (graphical user interface), показати пряма застосування неможливо. Але програма має велику кількість логів, по яким можна відслідкувати та продемонструвати тестування програми.

Так як програма складається з двох основних частин, серверна та чипова, для початку розглянемо чипову сторону. Для прикладу візьмемо випадкову початкову точку: Нова Зеландія Данідін(-45.880834,170.499600), а за кінцеву точку візьмемо: Нова Зеландія Хемпден (-45.325222, 170.815534), що знаходиться на відстані приблизно в 80 кілометрів від початкової точки, і має приблизний час поїздки на машині в одну годину.

Перший крок в роюоті сервера це отримання початкової та кінцевої точок. Це можна зробити великою кількістю можливих варіантів, від прокладення веб хука з google maps, до застосування непрямого запиту до сервера з координатами. Це буде вже залежить від типу автомобіля, його вбудованої электроники, та інших змінних. Тому на даний момент не будемо приділяти такої пильної уваги до то як ми отримали ці дані.

Після отримання даних про початкову та кінцеву точки починається процес знаходження маршруту за допомогою OpenRouteService. Це виконується в три етапи:

Перший етап це отримання сирих даних з OpenRouteService. Ці дані включаються в себе дистанцію від однієї точки до наступної, тип події та ще

додаткову інформацію, рисунок 3.6.1. Другий етап це форматування цієї всієї інформації в список для останнього етапу, рисунок 3.6.2.

Третій етап це форматуванні всіх даних в два списки. Перший список представляє собою події які відбудуться, і він застосовується для знаходження даних в базі знань про можливі типи зіткнення на дорозі. Цей список включає в себе як і стандартні дані отримані з OpenRouteService, так і mid-lock. В даному контексті mid-lock представляє собою систему контролю трафіку, по типу світлофор і дорожніх переходів, де машині потрібно буде зупинитися. Mid-lock додається після першого отримання даних, також за допомогою OpenRouteService, але іншого його endpoint-a.

Другий список представляє собою набір координат, які працюють у парі з подіями з першого списку даних. Ці координати зсунуті на 2 метри у сторону попередньої події. Це дає можливість серверу почати обробляти наступну події, поки зберігаючу набір інструкції для події яка трапиться через декілька секунд.


[{'distance': 893.1, 'duration': 164.7, 'type': 11, 'instruction': 'Head southeast on Marktstraße', 'name': 'Marktstraße', 'way_points': [0, 31]}, {'distance': 2317.8, 'duration': 264.1, 'type': 12, 'instruction': 'Keep left onto Leibnizplatz', 'name': 'Leibnizplatz', 'way_points': [31, 82]}, {'distance': 10.4, 'duration': 3.7, 'type': 5, 'instruction': 'Turn slight right onto Buntentorsteinweg', 'name': 'Buntentorsteinweg', 'way_points': [82, 83]}, {'distance': 96.9, 'duration': 16.7, 'type': 0, 'instruction': 'Turn left onto Kornstraße', 'name': 'Kornstraße', 'way_points': [83, 88]}, {'distance': 407.7, 'duration': 56.9, 'type': 1, 'instruction': 'Turn right onto Kornstraße', 'name': 'Kornstraße', 'way_points': [88, 98]}, {'distance': 25.5, 'duration': 7.5, 'type': 6, 'instruction': 'Continue straight onto Kattenturmer Heerstraße', 'name': 'Kattenturmer Heerstraße', 'way_points': [98, 100]}, {'distance': 10.5, 'duration': 3.8, 'type': 0, 'instruction': 'Turn left onto Neuenlander Straße', 'name': 'Neuenlander Straße', 'way_points': [100, 101]}, {'distance': 3542.8, 'duration': 310.5, 'type': 6, 'instruction': 'Continue straight', 'name': '-', 'way_points': [101, 139]}, {'distance': 550.5, 'duration': 58.5, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [139, 158]}, {'distance': 8798.3, 'duration': 302.8, 'type': 12, 'instruction': 'Keep left', 'name': '-', 'way_points': [158, 243]}, {'distance': 144.9, 'duration': 4.7, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [243, 246]}, {'distance': 55311.4, 'duration': 1493.4, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [246, 546]}, {'distance': 29647.5, 'duration': 886.3, 'type': 6, 'instruction': 'Continue straight onto A 27', 'name': 'A 27', 'way_points': [546, 674]}, {'distance': 12398.9, 'duration': 348.7, 'type': 13, 'instruction': 'Keep right onto A 352', 'name': 'A 352', 'way_points': [674, 759]}, {'distance': 7774.0, 'duration': 794.6, 'type': 13, 'instruction': 'Keep right onto Flughafenschnellweg', 'name': 'Flughafenschnellweg', 'way_points': [759, 875]}, {'distance': 218.6, 'duration': 29.1, 'type': 6, 'instruction': 'Continue straight onto Vahrenwalder Straße, L 190', 'name': 'Vahrenwalder Straße, L 190', 'way_points': [875, 882]}, {'distance': 14.1, 'duration': 3.4, 'type': 0, 'instruction': 'Turn left onto Weidendamm, L 380', 'name': 'Weidendamm, L 380', 'way_points': [882, 883]}, {'distance': 244.9, 'duration': 45.8, 'type': 6, 'instruction': 'Continue straight onto Herschelstraße', 'name': 'Herschelstraße', 'way_points': [883, 889]}, {'distance': 179.9, 'duration': 25.9, 'type': 1, 'instruction': 'Turn right onto Karolinenstraße', 'name': 'Karolinenstraße', 'way_points': [889, 897]}, {'distance': 245.2, 'duration': 32.4, 'type': 1, 'instruction': 'Turn right onto Celler Straße', 'name': 'Celler Straße', 'way_points': [897, 902]}, {'distance': 495.6, 'duration': 77.3, 'type': 2, 'instruction': 'Turn sharp left onto Goseriede', 'name': 'Goseriede', 'way_points': [902, 917]}, {'distance': 81.2, 'duration': 19.5, 'type': 1, 'instruction': 'Turn right onto Lützowstraße', 'name': 'Lützowstraße', 'way_points': [917, 920]}, {'distance': 0.0, 'duration': 0.0, 'type': 10, 'instruction': 'Arrive at Lützowstraße, on the right', 'name': '-', 'way_points': [920, 920]}]

Рисунок 3.6.1 - Сирі дані отримані з OpenRouteService


{'distance': 893.1, 'duration': 164.7, 'type': 11, 'instruction': 'Head southeast on Marktstraße', 'name': 'Marktstraße', 'way_points': [0, 31]}
{'distance': 2317.8, 'duration': 264.1, 'type': 12, 'instruction': 'Keep left onto Leibnizplatz', 'name': 'Leibnizplatz', 'way_points': [31, 82]}
{'distance': 10.4, 'duration': 3.7, 'type': 5, 'instruction': 'Turn slight right onto Buntentorsteinweg', 'name': 'Buntentorsteinweg', 'way_points': [82, 83]}
{'distance': 96.9, 'duration': 16.7, 'type': 0, 'instruction': 'Turn left onto Kornstraße', 'name': 'Kornstraße', 'way_points': [83, 88]}
{'distance': 407.7, 'duration': 56.9, 'type': 1, 'instruction': 'Turn right onto Kornstraße', 'name': 'Kornstraße', 'way_points': [88, 98]}
{'distance': 25.5, 'duration': 7.5, 'type': 6, 'instruction': 'Continue straight onto Kattenturmer Heerstraße', 'name': 'Kattenturmer Heerstraße', 'way_points': [98, 100]}
{'distance': 10.5, 'duration': 3.8, 'type': 0, 'instruction': 'Turn left onto Neuenlander Straße', 'name': 'Neuenlander Straße', 'way_points': [100, 101]}
{'distance': 3542.8, 'duration': 310.5, 'type': 6, 'instruction': 'Continue straight', 'name': '-', 'way_points': [101, 139]}
{'distance': 550.5, 'duration': 58.5, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [139, 158]}
{'distance': 8798.3, 'duration': 302.8, 'type': 12, 'instruction': 'Keep left', 'name': '-', 'way_points': [158, 243]}
{'distance': 144.9, 'duration': 4.7, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [243, 246]}
{'distance': 55311.4, 'duration': 1493.4, 'type': 13, 'instruction': 'Keep right', 'name': '-', 'way_points': [246, 546]}
{'distance': 29647.5, 'duration': 886.3, 'type': 6, 'instruction': 'Continue straight onto A 27', 'name': 'A 27', 'way_points': [546, 674]}
{'distance': 12398.9, 'duration': 348.7, 'type': 13, 'instruction': 'Keep right onto A 352', 'name': 'A 352', 'way_points': [674, 759]}
{'distance': 7774.0, 'duration': 794.6, 'type': 13, 'instruction': 'Keep right onto Flughafenschnellweg', 'name': 'Flughafenschnellweg', 'way_points': [759, 875]}
{'distance': 218.6, 'duration': 29.1, 'type': 6, 'instruction': 'Continue straight onto Vahrenwalder Straße, L 190', 'name': 'Vahrenwalder Straße, L 190', 'way_points': [875, 882]}
{'distance': 14.1, 'duration': 3.4, 'type': 0, 'instruction': 'Turn left onto Weidendamm, L 380', 'name': 'Weidendamm, L 380', 'way_points': [882, 883]}
{'distance': 244.9, 'duration': 45.8, 'type': 6, 'instruction': 'Continue straight onto Herschelstraße', 'name': 'Herschelstraße', 'way_points': [883, 889]}
{'distance': 179.9, 'duration': 25.9, 'type': 1, 'instruction': 'Turn right onto Karolinenstraße', 'name': 'Karolinenstraße', 'way_points': [889, 897]}
{'distance': 245.2, 'duration': 32.4, 'type': 1, 'instruction': 'Turn right onto Celler Straße', 'name': 'Celler Straße', 'way_points': [897, 902]}
{'distance': 495.6, 'duration': 77.3, 'type': 2, 'instruction': 'Turn sharp left onto Goseriede', 'name': 'Goseriede', 'way_points': [902, 917]}
{'distance': 81.2, 'duration': 19.5, 'type': 1, 'instruction': 'Turn right onto Lützowstraße', 'name': 'Lützowstraße', 'way_points': [917, 920]}
{'distance': 0.0, 'duration': 0.0, 'type': 10, 'instruction': 'Arrive at Lützowstraße, on the right', 'name': '-', 'way_points': [920, 920]}

Рисунок 3.6.2 - Дані сформовані в лист

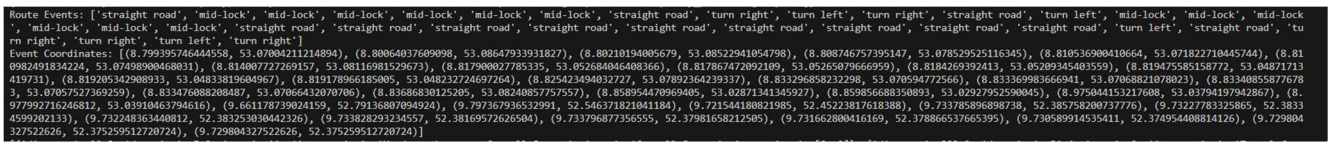
Route Events: ['straight road', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'straight road', 'turn right', 'turn left', 'turn right', 'straight road', 'turn left', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'mid-lock', 'straight road', 'straight road', 'straight road', 'straight road', 'straight road', 'straight road', 'straight road', 'straight road', 'turn left', 'straight road', 'turn right', 'turn right', 'turn left', 'turn right']
Event Coordinates: [(8.799395746444558, 53.07004211214894), (8.80064037609098, 53.08647933931827), (8.80210194005679, 53.08522941054798), (8.808746757395147, 53.078529525116345), (8.810536900410664, 53.071822710445744), (8.810982491834224, 53.07498900468031), (8.814007727269157, 53.08116981529673), (8.817900027785335, 53.052684046408366), (8.817867472092109, 53.05265079666959), (8.8184269392413, 53.05209345403559), (8.819475585158772, 53.04871713419731), (8.819205342908933, 53.04833819604967), (8.819178966185005, 53.048232724697264), (8.825423494032727, 53.07892364239337), (8.833296858232298, 53.0705947725 66), (8.833369983666941, 53.07068821078023), (8.833408558776783, 53.07057527369259), (8.833476088208487, 53.07066432070706), (8.83686830125205, 53.08240857757557), (8.858954470969405, 53.02871341345927), (8.859856688350893, 53.02927952590045), (8.975044153217608, 53.03794197942867), (8.977992716246812, 53.03910463794616), (9.661178739024159, 52.79136807094924), (9.797367936532991, 52.546371821041184), (9.721544188821985, 52.45223817618388), (9.733785896898738, 52.385758200737776), (9.732277833258 65, 52.383345599202133), (9.732248363440812, 52.383253030442326), (9.733828293234557, 52.38169572626504), (9.733796877356555, 52.37981658212505), (9.731662800416169, 52.378866537665395), (9.730589914535411, 52.374954408814126), (9.729804327522626, 52.375259512720724), (9.729804327522626, 52.375259512720724)]

Рисунок 3.6.3 - Дані готові до застосування

Коли Автомобіль пересіке координатний набір для кожної події відбувається збір інформації для запиту в базу даних. Це включає в себе:

Погодні умови які зараз присутні на цих координатах, і включають в себе хмарність та мокрість дороги. Умови освітлення які зараз на дорозі. Після отримання цих умов за допомогою Open-Meteo, ми робимо запит до бази знань.

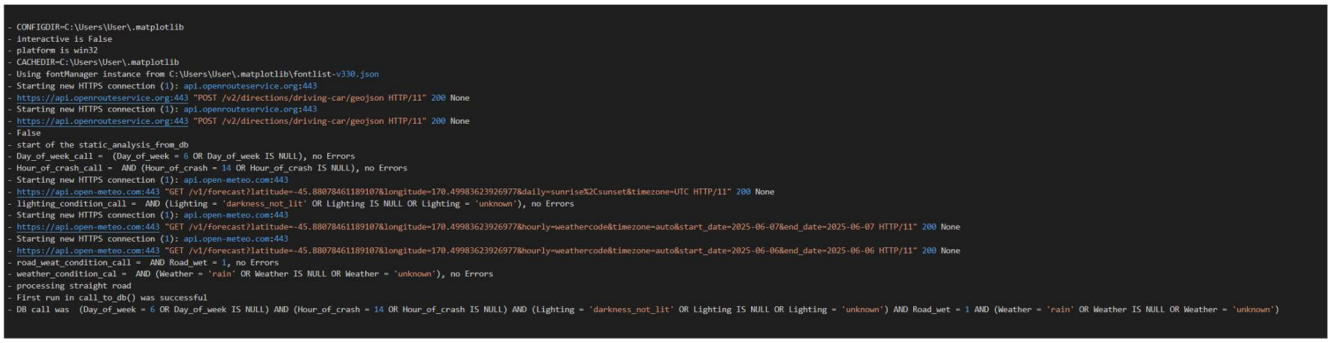

Рисунок 3.6.4 - отримання погодних умов і формування запита до БЗ

Через те що на даний момент база знань включає в себе тільки 1,590,854 даних з Нової Зеландії та Австралії з 2013 до 2020 роки, було додатково введена функція яка б нехтувала певними умовами, якщо б це означало отримання ненулевой кількості даних з бази знань.

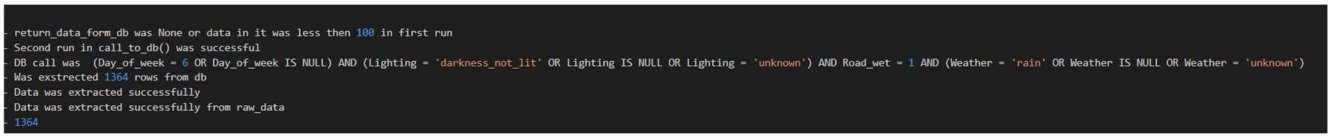

Рисунок 3.6.5 -Повторний виклик до бази знань

Після отримання даних з бази знань, вони переводяться в словники, це робиться для більш легкого знаходження кількості повторюваних варіантів. Це дає нам 5 словників: рівень серйозності інцидентів, обмеження швидкості, засоби регулювання руху, типи ДТП та інформацію про травми. Далі відбувається процес викидня None та Null значень з словників та їх сортування за значенням. Тобто найбільша кількість повторюваних значень буде спереду. Для швидкості окремо викидаються всі значення які були б менше за 20 кілометрів на годину.

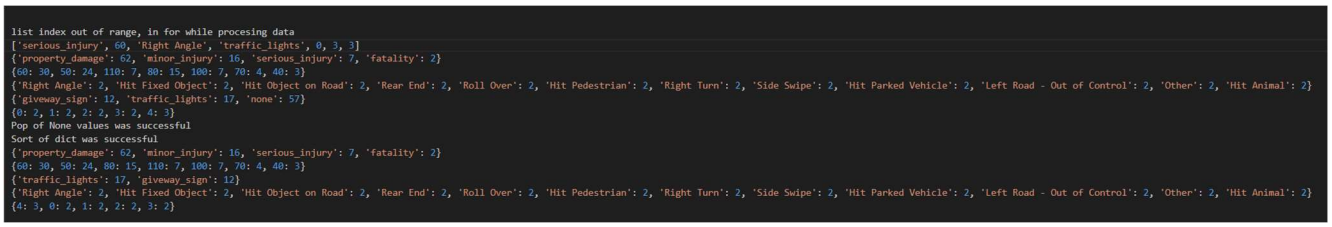

Рисунок 3.6.6 - Приклад створення словників та їх сортування

Останнім кроком в статистичному аналізі є підрахування можливостей та створення інструкції для подальшої її відправки до ACU. Всього створюються 8 інструкції, схему розподілення яких можна побачити у таблиці 3.6.7. Вони створюються на основі списків, зліва у списку буде значення яке більш за все повторювалося у словнику, праворуч яке менш за все мало повторень .Основним компонентом який визначає розподення являється кількість різних важкостей ушкоджень.

Принцип вибирання даних для формування інструкції зазначений на Рисунку 3.6.8. Коли більше ніж 1 значення важкостей ушкоджень, перше значення здвигається, на основі таблиці 3.6.7. Тобто коли ми маємо 2 різних важкості ушкодження, перше буде з'являтися 6 разів, друге всього лише 2.

Значення важкості ушкоджень	Розподілення ушкоджень
1	8 - преше
2	6 - преше, 2 - друге
3	5 - переше, 2 - друге, 1 - третье
4	4 - переше, 2 - друге, 1 - третье, 1 - четверте
5	3 - переше, 2 - друге, 1 - третье, 1 - четверте, 1- пяте
6	3 - переше, 1 - друге, 1 - третье, 1 - четверте, 1- пяте, 1 - шостке

Таблиця 3.6.7 - кількість ушкоджень, до кількості інструкцій до одного ушкодження

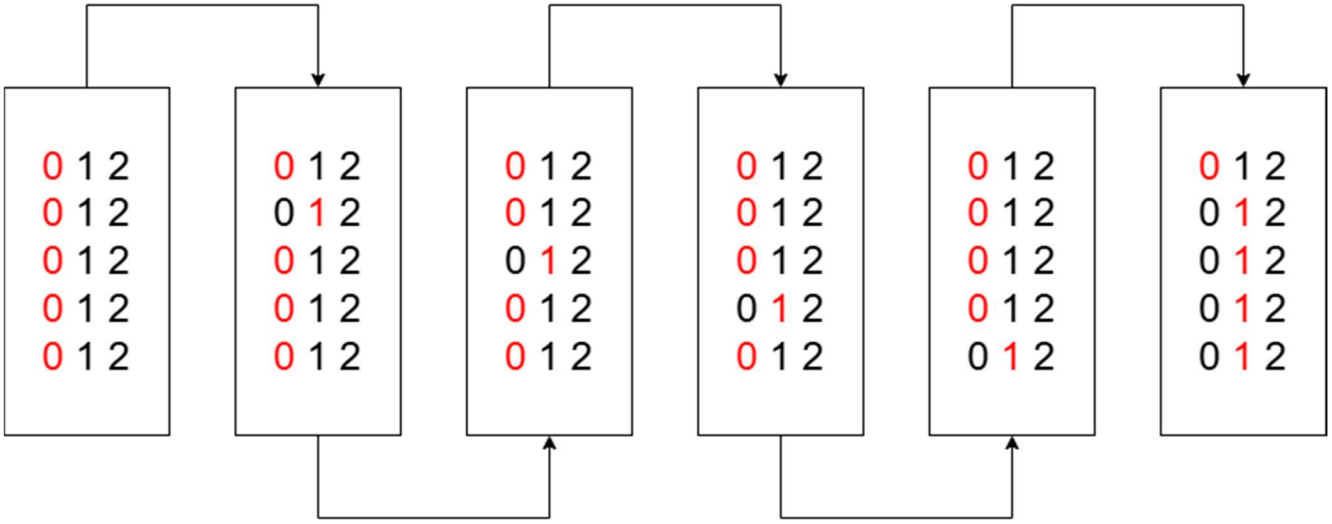

Рисунок 3.6.8 - Принци формування інструкції

Рисунок 3.6.9 - Приклад сформованого листа

Після чого дані передаються на чип, і сервер постійно перевіряє нові дані GPS, очікуючи поки координат з машини потрапляють у радіус 2 метри до наступної точки події. Після чого він починає процес розрахунку нових інструкцій.

Чипова сторона має набагато менше логової системи, через те що відкривання і записування в файл займає час, який може бути витрачений на обчислення аварії. В звичайному режимі, чип постійно перевіряє датчики ударів та гіроскоп на перегортання автомобіля. Одночасно з цим предае дані проа GPS коордити автомобіля до сервера. В випадку якщо не було виявлено ніякого перегортання або датчики удару показують значення мене ніж 2000 ньютонах на два найбільших значення логова система автомобіля буде виглядати наступним чином, як показано на рисунках 3.6.10 та 3.6.11.

```
[2025-06-07 19:34:38] ssv1: 0   ssv2: 0 ssv3: 0 ssv4: 0 ssv5: 0 ssv6: 0
sv1: 21123  sv2: 21123  sv3: 21123  sv4: 1114   sv5: 21123  sv6: 21123  sv7: 21123  sv8: 21123
[2025-06-07 19:34:40] ssv1: 0   ssv2: 0 ssv3: 0 ssv4: 0 ssv5: 0 ssv6: 0
sv1: 21123  sv2: 21123  sv3: 21123  sv4: 1114   sv5: 21123  sv6: 21123  sv7: 21123  sv8: 21123
[2025-06-07 19:34:42] ssv1: 0   ssv2: 0 ssv3: 0 ssv4: 0 ssv5: 0 ssv6: 0
sv1: 21123  sv2: 21123  sv3: 21123  sv4: 1114   sv5: 21123  sv6: 21123  sv7: 21123  sv8: 21123
[2025-06-07 19:34:44] ssv1: 0   ssv2: 0 ssv3: 0 ssv4: 0 ssv5: 0 ssv6: 0
sv1: 21123  sv2: 21123  sv3: 21123  sv4: 1114   sv5: 21123  sv6: 21123  sv7: 21123  sv8: 21123
```

Рисунок 3.6.10 - приклад основної логової системи без удару

```
All values < 2000? Yes
All values are 0? Yes
All values < 2000? Yes
All values are 0? Yes
All values < 2000? Yes
All values are 0? Yes
All values < 2000? Yes
All values are 0? Yes
All values < 2000? Yes
All values are 0? Yes
All values < 2000? Yes
All values are 0? Yes
```

Рисунок 3.6.10 - приклад логової системи перевірки удару

У інших випадках коли автомобіль фіксує перегортання або ж зіткнення на швидкості більше ніж 150 кілометрів на годину, йде перевірка па пристібнутись ременів безпеки. І випуск всіх подушок безпеки, де є пристібнуті ремені безпеки.

Якщо автомобіль потрапив у аварію, на швидкості більше ніж 20 кілометрів на годину, відбується наступні дій у заданому порядку:

а. Знаходження напрямку удару, за допомогою визначення двох чіпів з найбільшим показником удару.

б. Розділення отриманих інструкцій, на напрям удару, перші дві цифри в інструкції, і саму інструкцію випуску подушок безпеки.

в. Спроба знайти рішення на основі напряму удару, і даних отриманих з серверної сторони.

г. Якщо ж немає підходящих даних, чип переходить в стрес мод, де він буде визначати кращій хід дій локально. Спираючись на дані які вже були опрацьовані під час удару.

д. Направлення струму для випуску подушок безпеки, в зданому порядку.

Це все відбувається за час менший ніж 200 мілісекунд, для того щоб надати подушкам безпеки час для відкриття та здування. Приклад логів при такій аварій можна побачити на рисунку 3.6.12.

```
ssv2: 10000 ssv3: 2000  ssv4: 0 ssv5: 0 ssv6: 0
sv1: 21123  sv2: 21123  sv3: 21123  sv4: 1114   sv5: 21123  sv6: 21123  sv7: 21123  sv8: 21123
max value 1: 10000 max value 2: 2000
Starting the process of reviewing the instructions.
Direction of the lesion 0x2fab5ff4d0
Instruction 0x2fab5ff4f0
frontal right collision
answer was found 123
```

Рисунок 3.6.12 - приклад логів під час колізій

3.7 Перспективи застосування та впровадження

В цьому завершаючому підрозділі я хотів би розглянути можливі перспективи застосування та впровадження цієї програми.

З моєї особистої точки зору цей тип додаткового захисту не буде розроблено та застосовано приватними компаніями, через велику вартість процесу розробки і пыдтримки такої програми. Існуючі засоби захисту в автомобілях є ефективними, та проходять всі інспекції встановлені державами. Тому зменшення на декілька відсотків кількості травм, при збільшенні кількості персоналу та інфраструктури для підтримки такого проекту не виглядає реалістичним.

Але цей проект може бути реалізованим самими державами. Бо соціальна ціна в Новій зеландій, звідки було взято частично дані для проекту, вартує 505,600 доларів США для важких ушкоджень, та 25300 для легких травм[21], в 2020 році.

Тотальні соціальні витрати, від автомобільних аварій в 2019 році, становило 4.6 мільярдів доларів США. Як це розраховується можна побачити у таблиці 3.7.1

	Важкість травм		
Складові вартості	Фатальні	Серйозні	Легкі
Ціни червня 2020 року ($)			
Втрата	5,240,300	505,600	21,400

життя/постійна інвалідність			
Втрата працездатності (тимчасова непрацездатність)	800	1,800	400
Медичні	15,200	17,600	1,100
-Лікарня/медична	8,700	10,800	200
- Невідкладна/догоспітальна допомога	4,400	1,500	800
-Продовження лікування	2,200	1,500	800
Юридичний і судовий	32,700	3,900	1,300
Пошкодження транспортного засобу	12,800	8,000	6,500
Всього	**5,301,800**	**537,000**	**30,600**

таблиця 3.7.1 Середні соціальні витрати на одне ДТП за складовими витрат

Хоч ми і бачимо тенденцію зменшення кількості аварій на душу населення. Нова зеландія досі втрачає щонайменше 2 мільярди доларів США з 2000 року, через дорожні аварії.

Тому така система для держави це вигідно. Це надає змогу мати інфраструктуру та інформацію для знаходження точок де відбувається найбільше кількість аварії, через збір даних. Які держава може збирати напряму зі звої серверів. І також за допомогою цих даних зменшувати кількість травм і ушкоджень на дорогах.

3.8 Висновки до розділу 3

У рамках даного розділу було здійснено аналіз і вибір технологій для реалізації програмного забезпечення системи активної безпеки автомобіля. Серед розглянутих мов програмування для різних рівнів архітектури обрано Python — як основний інструмент для обробки даних на сервері, SQL — для роботи з базою

знань, а також C++ — для реалізації компонентів, що потребують високої продуктивності та прямої взаємодії з апаратним забезпеченням. Такий підхід забезпечує оптимальний баланс між гнучкістю, швидкодією та стабільністю роботи системи.

Сформована архітектура включає три взаємопов’язані складові: серверну частину, базу знань і ACU (автомобільний керуючий пристрій). Сервер відповідає за обробку GPS-координат, збір метеоданих, пошук релевантних випадків з бази знань та формування набору інструкцій, які в режимі реального часу передаються до ACU. Останній, у свою чергу, аналізує ці інструкції та приймає рішення щодо активації подушок безпеки, навіть у випадках втрати зв’язку із сервером.

База знань сформована на основі відкритих даних з Нової Зеландії та Австралії, що дозволило розробити алгоритми статистичного аналізу для прогнозування можливих сценаріїв аварій. Значну увагу приділено забезпеченню надійної комунікації, зокрема шляхом використання модулів SIM800L та Iridium 9602, що дозволяють адаптувати систему до різних умов покриття.

Проведене тестування підтвердило працездатність усіх компонентів, включно з маршрутизацією, обробкою подій, реагуванням на аварії та стресовими сценаріями. Отримані результати свідчать про стабільну роботу системи в реальному часі, її масштабованість і можливість практичного впровадження в сучасні транспортні засоби.

ВИСНОВКИ

Дипломний проєкт присвячено створенню інформаційної технології прогнозування спрацювання подушок безпеки в автомобільному транспорті з використанням методів статистичного аналізу даних. У ході роботи було розглянуто архітектуру сучасних систем безпеки автомобіля, зокрема алгоритми, що лежать в основі прийняття рішень про активацію подушок безпеки.

У першому розділі було проаналізовано структуру подушок безпеки та принципи їх функціонування, а також досліджено роль ACU-блоків у процесі прийняття рішень, описано ключові сенсорні елементи системи.

Другий розділ містив огляд та вибір апаратних компонентів для збору кінематичних даних, серед яких було використано Arduino, сенсори LSM6DS3, ADXL377 та GPS-модуль NEO-6M. Порівняльний аналіз підтвердив їхню доцільність для проєкту з точки зору точності, енергоефективності та швидкодії.

У третьому розділі було розроблено програмне забезпечення, побудовано архітектуру системи та реалізовано обмін даними між серверною частиною та ACU. Було здійснено тестування роботи системи на основі змодельованих сценаріїв ДТП, що підтвердило її ефективність та відповідність поставленим технічним вимогам. Особлива увага приділена використанню відкритих джерел даних і модулів статистичного аналізу для адаптації системи до реальних умов.

Розроблена технологія може знайти застосування у сфері автомобільного виробництва, страхування, а також у транспортній логістиці та аналітиці дорожньої безпеки. Вона забезпечує підвищення точності та надійності спрацювання подушок безпеки, що сприяє зменшенню травматизму внаслідок дорожньо-транспортних пригод.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] World Health Organization (WHO) Road traffic injuries - https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] Warner, Kenneth E. (1983). "Bags, Buckles, and Belts: The Debate over Mandatory Passive Restraints in Automobiles". Journal of Health Politics, Policy and Law. 8 (1): 44–75. doi:10.1215/03616878-8-1-44. PMID 6863874.

[3] ДИРЕКТИВА ЄВРОПЕЙСЬКОГО ПАРЛАМЕНТУ І РАДИ 2014/45/ЄС - https://zakon.rada.gov.ua/go/984_034-14

[4] https://zakononline.com.ua/documents/show/336591___336656

[5] https://mtu.gov.ua/news/448.html

[6] "Інтелектуальні системи автоматизації", Open Archive NURE, 2022.

[7] "Що таке Airbag?", VV24, 2023.

[8] "Системи безпеки сучасних автомобілів", АвтоЕксперт, 2021.

[9] Guido van Rossum et al., "Python Programming Language Overview"

[10] Asynchronous Programming in Python – Official Documentation

[11] OWASP Security Practices for Python Web Applications

[12] SQL Standards and Performance Optimization – PostgreSQL Docs

[13] Security Features in MySQL – MySQL Documentation

[14] Database Scaling Strategies – IEEE Journal

[15] Bjarne Stroustrup, "The C++ Programming Language"

[16] Low-Level Memory Management in C++ – ISO C++ Standards

[17] Iridium 9602_9602N SBD Transceiver Developers Guide V1.2 - DRAFT2

[18]Side curtain air bags offer protection from partial ejections during car crashes, new study finds - Harborview Injury Prevention & Research Center

[19] Evaluation of FMVSS No. 226: Curtain Air Bags and Ejection Mitigation in Rollover Events - NHTSA

[20] Seat Belts - NHTSA

[21] NZ Transport Agency Waka Kotahi - Social cost ofroad crashes and injuries 2020 update

ДОДАТОК А ЛІСТИНГИ ПРОГРАМИ

Серверна частина програми

Серверна частина програми 1 Функція для умов освітлення

```
import requests

from datetime import datetime, timedelta, timezone


def get_lighting_condition_on_road(latitude, longitude, hour):
    """
    Determine lighting condition for given latitude, longitude, and hour.
    Args:
        latitude (float): Latitude of the location.
        longitude (float): Longitude of the location.
        hour (str): Hour in HH:MM format, UTC timezone.
    Returns:
        tuple: One of ('daylight',), ('darkness_not_lit',), ('darkness_lit',), ('dawn_dusk',), or ('other',).
    """
    try:
        # Query Open-Meteo API for sunrise and sunset times
        response = requests.get(
            "https://api.open-meteo.com/v1/forecast",
            params={
                "latitude": latitude,
                "longitude": longitude,
                "daily": "sunrise,sunset",
```

```
        "timezone": "UTC",
    },
)

if response.status_code != 200:
    return ('other',)

data = response.json()
sunrise = data['daily']['sunrise'][0]  # Sunrise time (UTC)
sunset = data['daily']['sunset'][0]    # Sunset time (UTC)

# Convert hour to a datetime object
current_time = datetime.strptime(hour, "%H:%M").replace(tzinfo=timezone.utc)
sunrise_time = datetime.fromisoformat(sunrise).replace(tzinfo=timezone.utc)
sunset_time = datetime.fromisoformat(sunset).replace(tzinfo=timezone.utc)

# Define dawn and dusk times (1 hour before/after sunrise/sunset)
dawn_start = sunrise_time - timedelta(hours=1)
dawn_end = sunrise_time
dusk_start = sunset_time
dusk_end = sunset_time + timedelta(hours=1)

# Determine lighting condition
```

```
        if dawn_start <= current_time < dawn_end or dusk_start <= current_time <
dusk_end:
            return 'dawn_dusk'
        elif sunrise_time <= current_time < sunset_time:
            return 'daylight'
        elif current_time < dawn_start or current_time >= dusk_end:
            return 'darkness_not_lit' # Assume not lit by default
        else:
            return 'other'

    except Exception as e:
        print(f"Error: {e}")
        return 'other'
```

Серверна частина програми 2 Функція для отримання погодних умов

```
import requests
from datetime import datetime,timedelta

def get_decoded_weather(lat, lon,date):
    # Define weather code mappings
    weather_code_map = {
        0: "Clear sky",
        1: "Mainly clear",
        2: "Partly cloudy",
        3: "Overcast",
        45: "Fog",
        48: "Depositing rime fog",
        51: "Drizzle: Light intensity",
        53: "Drizzle: Moderate intensity",
        55: "Drizzle: Dense intensity",
        56: "Freezing drizzle: Light",
```

```
    57: "Freezing drizzle: Dense",
    61: "Rain: Slight intensity",
    63: "Rain: Moderate intensity",
    65: "Rain: Heavy intensity",
    66: "Freezing rain: Light",
    67: "Freezing rain: Heavy",
    71: "Snow fall: Slight intensity",
    73: "Snow fall: Moderate intensity",
    75: "Snow fall: Heavy intensity",
    77: "Snow grains",
    80: "Rain showers: Slight",
    81: "Rain showers: Moderate",
    82: "Rain showers: Violent",
    85: "Snow showers: Slight",
    86: "Snow showers: Heavy",
    95: "Thunderstorm: Slight or moderate",
    96: "Thunderstorm with hail: Slight",
    99: "Thunderstorm with hail: Heavy",
}

# Open-Meteo API endpoint and parameters
url = "https://api.open-meteo.com/v1/forecast"
params = {
    "latitude": lat,
    "longitude": lon,
    "hourly": "weathercode",
    "timezone": "auto",
    "start_date": date,
    "end_date": date,
}

# Make the API call
response = requests.get(url, params=params)
if response.status_code != 200:
    return {"error": f"Failed to fetch weather data: {response.text}"}

# Parse the API response
data = response.json()
hourly_times = data.get("hourly", {}).get("time", [])
```

```
    hourly_codes = data.get("hourly", {}).get("weathercode", [])

    # Decode weather codes into descriptions
    weather_dict = {}
    for i in range(len(hourly_times)):
        time = datetime.fromisoformat(hourly_times[i]).strftime("%H:%M")
        code = hourly_codes[i]
        weather_dict[time] = weather_code_map.get(code, "Unknown weather condition")

    return weather_dict

def get_decoded_weather_on_road_rn(lat, lon):
    """
    Take latitude and longitude

    returns two values:
        1. Weather condition, if form of string
        2. Boolean for - if road is wet? True - road is wet, False - road is not wet

    """
    try:
        road_wet = False

        LIST_FOR_ALL_DAY = ["Thunderstorm: Slight or moderate","Thunderstorm
with hail: Heavy",
                            "Drizzle: Moderate intensity","Drizzle: Dense intensity","Freezing
drizzle: Dense","Rain: Moderate intensity"
                            "Rain: Heavy intensity","Freezing rain: Heavy","Rain showers:
Moderate","Rain showers: Violent"]

        weather_today = get_decoded_weather(lat, lon,datetime.now().strftime("%Y-%m-
%d"))
        yesterday = datetime.now() - timedelta(days=1)
        weather_yesterday = get_decoded_weather(lat, lon,yesterday.strftime("%Y-%m-
%d"))

        Rain_yesterday = any(any(substring in value for substring in
LIST_FOR_ALL_DAY) for value in weather_yesterday.values())
```

```
        Rain_today = any(any(substring in value for substring in LIST_FOR_ALL_DAY)
for value in weather_today.values())
        hour_now = datetime.now().hour

        Rain_not_so_long_ago = False

        for x in range(hour_now-2,hour_now+2):
            value = weather_today.get(str(x)+":00")
            if "rain" in value.lower():
                Rain_not_so_long_ago = True
            elif "Thunderstorm" in value:
                Rain_not_so_long_ago = True
            elif "drizzle" in value.lower():
                Rain_not_so_long_ago = True

        if Rain_yesterday or Rain_today or Rain_not_so_long_ago:
            road_wet = True

        return weather_today.get(str(hour_now)+":00"),road_wet
    except Exception as e:
        print("EROOR",e)
        return "Overcast",False
```

Серверна частина програми 3 Функція для отримання дорожніх подій

```
import openrouteservice
import osmnx as ox
import networkx as nx
import math
from dotenv import load_dotenv
import os
from shapely.geometry import LineString, Point
from math import isclose


def get_route_details_with_coords(start_point, end_point, ors_api_key):
    """
    Get route details with event descriptions and coordinates adjusted 5m before the
event.
```

```
    Includes specific turn and intersection directions.

    Args:
        start_point (tuple): Coordinates of the starting point (longitude, latitude).
        end_point (tuple): Coordinates of the ending point (longitude, latitude).
        ors_api_key (str): OpenRouteService API key.

    Returns:
        list: Description of each route segment (e.g., "straight road", "midblock",
"intersection", "turn").
        list: Coordinates (longitude, latitude) of when events happen, adjusted by 5m
before the event.
    """
    def adjust_coordinates(coord1, coord2, distance_m):

        """
        Adjust coordinates `coord1` towards `coord2` by a given distance in meters.

        Args:
            coord1 (tuple): Starting coordinate (longitude, latitude).
            coord2 (tuple): Target coordinate (longitude, latitude).
            distance_m (float): Distance in meters to adjust.

        Returns:
            tuple: Adjusted coordinate (longitude, latitude).
        """
        # Approximate Earth's radius in meters
        R = 6371000

        # Convert degrees to radians
        lon1, lat1 = math.radians(coord1[0]), math.radians(coord1[1])
        lon2, lat2 = math.radians(coord2[0]), math.radians(coord2[1])

        # Compute bearing from coord1 to coord2
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        bearing = math.atan2(
            math.sin(dlon) * math.cos(lat2),
```

```
            math.cos(lat1) * math.sin(lat2) - math.sin(lat1) * math.cos(lat2) *
math.cos(dlon)
        )

        # Adjust distance to radians
        adjusted_distance = distance_m / R

        # Calculate the adjusted coordinates
        lat_adjusted = math.asin(
            math.sin(lat1) * math.cos(adjusted_distance) +
            math.cos(lat1) * math.sin(adjusted_distance) * math.cos(bearing)
        )
        lon_adjusted = lon1 + math.atan2(
            math.sin(bearing) * math.sin(adjusted_distance) * math.cos(lat1),
            math.cos(adjusted_distance) - math.sin(lat1) * math.sin(lat_adjusted)
        )

        # Convert radians back to degrees
        return (math.degrees(lon_adjusted), math.degrees(lat_adjusted))

    # Функція для перевірки, чи координати підходять між двома іншими
    def is_between(coord, coord1, coord2):
        return coord1 <= coord <= coord2 or coord2 <= coord <= coord1

    # Вставка верхнього списку в нижній
    def merge_coordinates(upper_list, lower_list):
        merged_list = lower_list[:]  # Копіюємо нижній список
        for coord in upper_list:
            inserted = False
            for i in range(len(merged_list) - 1):
                # Якщо координата підходить між двома іншими, вставляємо її
                if is_between(coord, merged_list[i], merged_list[i + 1]):
                    merged_list.insert(i + 1, coord)
                    events.insert(i + 1, "mid-lock")
                    inserted = True
                    break
            '''if not inserted:
                # Якщо координата не підходить ніде, додаємо її в кінець
                merged_list.append(coord)'''
```

```
    return merged_list

# Initialize ORS client
client = openrouteservice.Client(key=ors_api_key)

# Request directions from ORS
route = client.directions(
    coordinates=[start_point, end_point],
    profile='driving-car',
    format='geojson'
)

# Extract route geometry and steps
geometry = route['features'][0]['geometry']['coordinates']
steps = route['features'][0]['properties']['segments'][0]['steps']
print(steps)

# Load OpenStreetMap graph for road data
G = ox.graph_from_point(start_point[::-1], dist=2000, network_type='drive')  # Buffer around the start point
route_nodes = ox.nearest_nodes(G, [coord[0] for coord in geometry], [coord[1] for coord in geometry])



route_line = LineString([(coord[0], coord[1]) for coord in geometry])

# Find mid-block crossings
mid_block_crossings = []

# Extract all crossings (nodes with `highway=crossing`)
for node, data in G.nodes(data=True):
    if data.get('highway') == 'crossing':  # Check if node is a pedestrian crossing
        crossing_coords = (data['x'], data['y'])
        crossing_point = Point(crossing_coords)

        # Check if crossing is along the route (within buffer distance)
        if route_line.buffer(2).contains(crossing_point):  # 10 meters buffer
            # Check if the crossing is mid-block (not at an intersection)
```

```
            neighbors = list(G.neighbors(node))
            if len(neighbors) <= 2:  # Not an intersection if 2 or fewer connected edges
                mid_block_crossings.append(crossing_coords)

# Analyze route
events = []  # List to store segment types
event_coords = []  # List to store adjusted coordinates of events


for step in steps:

    print(step)
    #instruction = step['instruction'].lower()
    type_in_steps = step['type']

    if type_in_steps in [6,11,12,13]:
        events.append("straight road")
    elif type_in_steps in [0,2,4]:
         events.append("turn left")
    elif type_in_steps in [1,3,5]:
        events.append("turn right")
    elif type_in_steps in [7,8]:
        events.append("roundabout")
    elif type_in_steps == 9:
        events.append("u-turn")


    # Adjust the event coordinate 5m before the event
    coord_before_event = adjust_coordinates(
        geometry[step['way_points'][1] - 1],
        geometry[step['way_points'][1]],
        3
    )


    event_coords.append(coord_before_event)

adjusted_mid_block_crossings = []
```

```
    for crossing_coords in mid_block_crossings:
        # Reference point for direction (e.g., a nearby route node or any other point)
        # Here, we use the first route node as an example
        reference_point = geometry[0]  # First coordinate from the route geometry

        # Adjust crossing by 3 meter towards the reference point
        adjusted_coords = adjust_coordinates(crossing_coords, reference_point,
distance_m=3)

        # Store the adjusted crossing coordinates
        adjusted_mid_block_crossings.append(adjusted_coords)

    final_coords_list = merge_coordinates(adjusted_mid_block_crossings, event_coords)

    return events, final_coords_list
```

Серверна частина програми 4 Статистичний аналіз

```
from str_value_to_data_from_db import str_to_db_return_all_values
import logging
from static_annalysis_sub_function import analyze_values

for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)

logging.basicConfig(
    filename="static_analysis.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

def static_analysis_from_db(latitude,longitude,str_for_db_call:str):
    """
    Use latitude and longitude for get data from get_lighting_condition and
get_the_weather

    str call is "straight road"
                "turn left"
```

```
            "turn right"
            "roundabout"
            "u-turn"
    from path_to_list.py, value can also be None

    """


    logging.debug("start of the static_analysis_from_db")
    try:
        raw_data_from_db =
str_to_db_return_all_values(latitude,longitude,str_for_db_call)
        logging.debug("Data was extracted successfully")
    except Exception as e:
        logging.error(f"Error happened while trying to pull data from db: {e}")
        logging.warning("Sending back None, program will run some time in emergency
state")
        return None

    if (raw_data_from_db is not None) and (0<len(raw_data_from_db)):
        # extracting data from raw data from_db
        #and writing in data only Severity, Speed_limit,Crash_type,Traffic_controls,
Fatalities,Serious_injuries,Minor_injuries
        data = [[ds[3], ds[4], ds[12], ds[14], ds[15], ds[16],ds[17]] for ds in
raw_data_from_db]
        logging.debug("Data was extracted successfully from raw_data")

        Severity_dict = dict()
        Speed_limit_dict = dict()
        Traffic_controls_dict = dict()
        Crash_typ_dict = dict()
        injuries = dict()

        logging.debug(len(data))

        for x in data:
            try:
                Severity_dict[x[0]] = Severity_dict.get(x[0], 1) + 1
                Speed_limit_dict[x[1]] = Speed_limit_dict.get(x[1], 1) + 1
```

```
            Crash_typ_dict[x[2]] = Traffic_controls_dict.get(x[2], 1) + 1
            Traffic_controls_dict[x[3]] = Traffic_controls_dict.get(x[3], 1) + 1

            if all(x[i] is not None for i in [4, 6]):
                inj_sum = ((x[4]*3) + (x[5]*2) + x[6] )
                injuries[inj_sum]  = injuries.get(x[inj_sum], 1) + 1
            else:
                injuries[None]  = injuries.get(None, 1) + 1
        except Exception as e:
            logging.error(f"{e}, in for while procesing data")
            logging.error(x)
            break

    logging.debug(Severity_dict)
    logging.debug(Speed_limit_dict)
    logging.debug(Crash_typ_dict)
    logging.debug(Traffic_controls_dict)
    logging.debug(injuries)


    try:
        for x in [None,'none','None']:
            Severity_dict.pop(x,False)
            Speed_limit_dict.pop(x,False)
            Traffic_controls_dict.pop(x,False)
            Crash_typ_dict.pop(x,False)
            injuries.pop(x,False)
        logging.debug("Pop of None values was successful")

        sorted_Severity_dict = dict(sorted(Severity_dict.items(), key=lambda item:
item[1], reverse=True))
        sorted_Speed_limit_dict = dict(sorted(Speed_limit_dict.items(), key=lambda
item: item[1], reverse=True))
        sorted_Traffic_controls_dict = dict(sorted(Traffic_controls_dict.items(),
key=lambda item: item[1], reverse=True))
        sorted_Crash_typ_dict = dict(sorted(Crash_typ_dict.items(), key=lambda item:
item[1], reverse=True))
        sorted_injuries = dict(sorted(injuries.items(), key=lambda item: item[1],
reverse=True))
```

```
            logging.debug("Sort of dict was successful")

            logging.debug(sorted_Severity_dict)
            logging.debug(sorted_Speed_limit_dict)
            logging.debug(sorted_Traffic_controls_dict)
            logging.debug(sorted_Crash_typ_dict)
            logging.debug(sorted_injuries)

            #decide what to send to the chip
            list_to_return = analyze_values(
            list(sorted_Severity_dict),
            list(sorted_Speed_limit_dict),
            list(sorted_Traffic_controls_dict),
            list(sorted_Crash_typ_dict),
            list(sorted_injuries)
            )

            logging.debug("List was successfuly formd")
            logging.debug(list_to_return)

            return list_to_return

        except KeyError as ke:
            logging.error(f"Error happened while trying to pop None from dict: {ke}")
            logging.warning("Sending back None, program will run some time in
emergency state")
            return None
        except Exception as e:
            logging.error(f"Error happened while sorting data in dict: {e}")
            logging.warning("Sending back None, program will run some time in
emergency state")
            return None

    else:
        logging.error("Data was None")
        logging.warning("Sending back None, program will run some time in emergency
state")
        return None
```

Серверна частина програми 5 допоміжна функція для статистичного аналізу

```
import logging

logger_static_annalysis_sub_function =
logging.getLogger("Logger_static_annalysis_sub_function")
logger_static_annalysis_sub_function.setLevel(logging.DEBUG)
file_handler1 = logging.FileHandler("static_annalysis_sub_function.log")
formatter1 = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s -
%(message)s")
file_handler1.setFormatter(formatter1)
logger_static_annalysis_sub_function.addHandler(file_handler1)

def
sub_function_for_one_severity(sorted_severity_dict:list,sorted_speed_limit_dict:list,sor
ted_traffic_controls_dict:list,sorted_crash_typ_dict:list,sorted_injuries:list):
    list_to_return_one_severity = []
    x = 0
    y = 0
    z = 0
    i = 0
    for _ in range(8):

list_to_return_one_severity.append([sorted_severity_dict[0],sorted_speed_limit_dict[x],
sorted_traffic_controls_dict[y],sorted_crash_typ_dict[z],sorted_injuries[i],])
        x, y, z, i = (
        (x+1, y, z, i) if x==y==z==i else
        (x-1, y+1, z, i) if x>y else
        (x, y-1, z+1, i) if y>z else
        (x, y, z-1, i+1) if z>i else
        (i, i, i, i))

    logger_static_annalysis_sub_function.debug(f'List of the values from static
analysis {list_to_return_one_severity}')
    return list_to_return_one_severity
```

```
def
sub_function_for_more_than_one_severity(sorted_severity_dict:list,sorted_speed_limit
_dict:list,sorted_traffic_controls_dict:list,sorted_crash_typ_dict:list,sorted_injuries:list):
    list_to_return_severity = []
    lenth_values =[[6,2],
                   [5,3,1],
                   [4,2,1,1],
                   [3,2,1,1,1],
                   [3,1,1,1,1,1]]

    value_to_take = len(sorted_severity_dict) - 2
    logger_static_annalysis_sub_function.debug(f"List of values that
expexted{lenth_values[value_to_take]}")
    itter_severity = 0
    for itter in lenth_values[value_to_take]:
        x = 0
        y = 0
        z = 0
        i = 0
        for _ in range(itter):

list_to_return_severity.append([sorted_severity_dict[itter_severity],sorted_speed_limit_
dict[x],sorted_traffic_controls_dict[y],sorted_crash_typ_dict[z],sorted_injuries[i],])
            x, y, z, i = (
            (x+1, y, z, i) if x==y==z==i else
            (x-1, y+1, z, i) if x>y else
            (x, y-1, z+1, i) if y>z else
            (x, y, z-1, i+1) if z>i else
            (i, i, i, i))
        itter_severity += 1

    logger_static_annalysis_sub_function.debug(f'List of the values from static analysis
{list_to_return_severity}')
    return list_to_return_severity



def
analyze_values(sorted_Severity_dict:list,sorted_Speed_limit_dict:list,sorted_Traffic_co
ntrols_dict:list,sorted_Crash_typ_dict:list,sorted_injuries:list):
```

```
    print("analizis")
    try:
        if
min([len(sorted_Severity_dict),len(sorted_Speed_limit_dict),len(sorted_Traffic_control
s_dict),len(sorted_Crash_typ_dict),len(sorted_injuries)]) != 0:
            if len(sorted_Severity_dict) == 1:
                logger_static_annalysis_sub_function.debug("sorted_Severity_dict was 1,
proceed to the sub_function_for_one_severity")
                return
sub_function_for_one_severity(sorted_Severity_dict,sorted_Speed_limit_dict,sorted_Tr
affic_controls_dict,sorted_Crash_typ_dict,sorted_injuries)
            else:
                logger_static_annalysis_sub_function.debug("sorted_Severity_dict was more
than 1, proceed to the sub_function_for_more_than_one_severity")
                return
sub_function_for_more_than_one_severity(sorted_Severity_dict,sorted_Speed_limit_di
ct,sorted_Traffic_controls_dict,sorted_Crash_typ_dict,sorted_injuries)
        else:
            return None
    except Exception as e:
        logger_static_annalysis_sub_function.error(e)
        return None
```

Серверна частина програми 6 функція для отримання даних для статистичного аналізу

```
import datetime
from Data_from_db_using_condition import get_data_from_db
from get_lighting_condition import get_lighting_condition_on_road
from get_the_weather import get_decoded_weather_on_road_rn
import logging

for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)

# Configure logging
logging.basicConfig(
    filename="db_call_processing_code.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s"
```

```
)

def str_to_db_return_all_values(latitude, longitude,value:str=None):
    """
   Function for getting data from database on parametres latitude, longitude
   value is presenting value direction that car will need to take

    """


    def
call_to_db(day_of_week_call:str,hour_of_crash_call:str,Lighting_Condition_call:str,Ro
ad_Weat_Condition_call,Weather_Condition_cal,Additional_Values_for_call = '):
        try:
            ultimate_call_to_omnissiah = day_of_week_call + hour_of_crash_call +
Lighting_Condition_call + Road_Weat_Condition_call + Weather_Condition_cal +
Additional_Values_for_call
            print(ultimate_call_to_omnissiah,"ultimate_call_to_omnissiah")
            return_data_form_db = get_data_from_db(ultimate_call_to_omnissiah)
            logging.debug("First run in call_to_db() was successful")
            logging.debug(f"DB call was {ultimate_call_to_omnissiah}")

            if (return_data_form_db is None) or (len(return_data_form_db) < 100):
                logging.debug("return_data_form_db was None or data in it was less then 100
in first run")
                ultimate_call_to_omnissiah = day_of_week_call + Lighting_Condition_call +
Road_Weat_Condition_call + Weather_Condition_cal + Additional_Values_for_call
                return_data_form_db =  get_data_from_db(ultimate_call_to_omnissiah)
                logging.debug("Second run in call_to_db() was successful")
                logging.debug(f"DB call was {ultimate_call_to_omnissiah}")

                if (return_data_form_db is None) or (len(return_data_form_db) < 100):
                    ("return_data_form_db was None or data in it was less then 100 in second
run")
                    ultimate_call_to_omnissiah = Lighting_Condition_call[4:] +
Road_Weat_Condition_call + Weather_Condition_cal + Additional_Values_for_call
                    print(ultimate_call_to_omnissiah,"ultimate_call_to_omnissiah")
                    return_data_form_db =  get_data_from_db(ultimate_call_to_omnissiah)
                    logging.debug("Third run in call_to_db() was successful")
                    logging.debug(f"DB call was {ultimate_call_to_omnissiah}")
```

```
            logging.debug(f"Was exstrected {len(return_data_form_db)} rows from db")
            return return_data_form_db
        except Exception as e:
            logging.error(f"Error in call_to_db(): {e}")
            return("Database request error")

    try:
        day_of_the_week = datetime.datetime.today().weekday()+1
        Day_of_week_call = " (Day_of_week = {x1} OR Day_of_week IS
NULL)".format(x1=day_of_the_week)
        logging.debug(f"Day_of_week_call = {Day_of_week_call}, no Errors")
    except Exception as e:
        logging.error(f"Error in Day_of_week_call: {e}")
        Day_of_week_call = ""

    try:
        now = datetime.datetime.now()
        Hour_of_crash_call = " AND (Hour_of_crash = {x2} OR Hour_of_crash IS
NULL)".format(x2=now.hour)
        logging.debug(f"Hour_of_crash_call = {Hour_of_crash_call}, no Errors")
    except Exception as e:
        logging.error(f"Error in Hour_of_crash_call: {e}")
        Hour_of_crash_call = ""

    try:
        lighting_condition =
get_lighting_condition_on_road(latitude,longitude,str(now.hour)+":00")
        lighting_condition_call = " AND (Lighting = \'{x1}\' OR Lighting IS NULL OR
Lighting = \'unknown\')"

        if lighting_condition == "dawn_dusk":
            lighting_condition_call = lighting_condition_call.format(x1="dawn_dusk")
        elif lighting_condition == "daylight":
            lighting_condition_call = lighting_condition_call.format(x1="daylight")
        elif lighting_condition == "darkness_not_lit":
            lighting_condition_call = lighting_condition_call.format(x1="darkness_not_lit")
        else:
            lighting_condition_call = lighting_condition_call.format(x1="other")
```

```
        logging.debug(f"lighting_condition_call = {lighting_condition_call}, no Errors")
    except Exception as e:
        logging.error(f"Error in lighting_condition_call: {e}")
        lighting_condition_call = ""

    try:
        weather, road_weat = get_decoded_weather_on_road_rn(latitude,longitude)
        weather = weather.lower()

        road_weat_condition_call = " AND Road_wet = {x1}".format(x1 =
int(road_weat))
        weather_condition_cal = " AND (Weather = \'{x1}\' OR Weather IS NULL OR
Weather = \'unknown\')"

        if "rain" in weather:
            weather_condition_cal = weather_condition_cal.format(x1 = "rain")
        elif "snow" in weather:
            weather_condition_cal = weather_condition_cal.format(x1 = "snow")
        elif "fog" in weather:
            weather_condition_cal = weather_condition_cal.format(x1 = "fog OR Weather
IS mist")
        elif "overcast" in weather:
            weather_condition_cal = weather_condition_cal.format(x1 = "overcast")
        elif ("clear sky" == weather) or ("mainly clear" == weather) or ("partly cloudy" ==
weather):
            weather_condition_cal = weather_condition_cal.format(x1 = "fine")
        else:
            weather_condition_cal = weather_condition_cal.format(x1 = "other")

        logging.debug(f"road_weat_condition_call = {road_weat_condition_call}, no
Errors")
        logging.debug(f"weather_condition_cal = {weather_condition_cal}, no Errors")
    except Exception as e:
        logging.error(f"Error in weather_condition_cal or road_weat_condition_call: {e}")
        road_weat_condition_call = ""
        weather_condition_cal = ""
```

```
    additional_values_for_call = " AND Intersection = {intr_value} AND Midlock IS
{mid_lock_value}"


    if (value is None) or (value == ""):
        logging.warning("value was None")
        return
call_to_db(Day_of_week_call,Hour_of_crash_call,lighting_condition_call,road_weat_c
ondition_call,weather_condition_cal)

    elif value == "straight road":
        logging.debug("processing straight road")
        '''additional_values_for_call = additional_values_for_call.format(intr_value = 0,
mid_lock_value = 0)
        logging.debug(f"additional_values_for_call = {additional_values_for_call}")'''
        return
call_to_db(Day_of_week_call,Hour_of_crash_call,lighting_condition_call,road_weat_c
ondition_call,weather_condition_cal,")

    elif "turn" in value:
        logging.debug("processing turn")
        ultimate_call_to_omnissiah += additional_values_for_call.format(intr_value = 1,
mid_lock_value = 0)
        logging.debug(f"additional_values_for_call = {additional_values_for_call}")
        return
call_to_db(Day_of_week_call,Hour_of_crash_call,lighting_condition_call,road_weat_c
ondition_call,weather_condition_cal,additional_values_for_call)

    elif "mid-lock" == value:
        logging.debug("processing mid-lock")
        ultimate_call_to_omnissiah += additional_values_for_call.format(intr_value = 0,
mid_lock_value = 1)
        logging.debug(f"additional_values_for_call = {additional_values_for_call}")
        return
call_to_db(Day_of_week_call,Hour_of_crash_call,lighting_condition_call,road_weat_c
ondition_call,weather_condition_cal,additional_values_for_call)

    elif value in ["roundabout","u-turn"]:
        logging.debug("processing roundabout/u-turn")
```

```
        ultimate_call_to_omnissiah += additional_values_for_call.format(intr_value = 1,
mid_lock_value = 0)
        logging.debug(f"additional_values_for_call = {additional_values_for_call}")
        return
call_to_db(Day_of_week_call,Hour_of_crash_call,lighting_condition_call,road_weat_c
ondition_call,weather_condition_cal,additional_values_for_call)
```

Серверна частина програми 6 Формування інструкції на базі статистичного аналізу

```
def what_to_do_ret_long(list_of_data):
    """
    returns list of 3 set of long -> 0 1 2 3 4 5 6 7 8 9

    first 0 1 always reserved for the calishion type

    11 - frontal
    21 - frontal right
    34 - rear right
    44 - rear
    43 - rear left
    12 - frontal left

    all other are for Airbags operating sequence

    0 - Not open
    1 - Front Airbags
    2 - Side Airbags
    3 - Curtain Airbags
    4- Knee Airbags
    5 - Rear Airbags
    6- Seatbelt Airbags
    7 -Center Airbags
    8 - Pedestrian Airbags
    9 - open all at the same time

    example
    449 - rear end collision open all airbags
    111456 - front end coloshion, opne airbegs in sequence Front Airbags -> Knee
Airbags -> Rear Airbags -> Seatbelt Airbags
```

```
    0 - property damage

    """

    list_for_return_thre_long = []
    for x in list_of_data:
        list_for_return_thre_long.append(evaluate_what_to_do(x))

    return list_for_return_thre_long


def check_for_injuris(str_value:str):
    if str_value == "property_damage":
        return 0
    elif str_value == "fatality":
        return 999
    else: return None



def evaluate_what_to_do(slise_of_data:list):
    right_angle_collision = ["Right Angle", "Right Angle Involving Parking", "Right
Angle Entering / Leaving Driveway", "Right Angle Involving Overtaking", "Right
Angle Involving Pedestrian"]
    rear_end_collision = ["Rear End", "Rear End Involving Parking", "Rear End
Involving Overtaking", "Rear End Entering / Leaving Driveway", "Rear End Involving
Pedestrian", "Rear End Involving Animal"]
    side_swipe_collision = ["Side Swipe", "Sideswipe Same Dirn Involving Overtaking",
"Sideswipe Same Dirn Involving Parking", "Sideswipe Same Dirn Entering / Leaving
Driveway", "Sideswipe Same Dirn Involving Animal", "Sideswipe Same Dirn Involving
Pedestrian"]
    head_on_collision = ["Head On", "Head On Involving Overtaking", "Head On
Entering / Leaving Driveway", "Head On Involving Animal", "Head On Involving
Parking", "Head On Involving Pedestrian"]
    right_turn_collision = ["Right Turn", "Right Turn Thru Entering / Leaving
Driveway", "Right Turn Thru Involving Pedestrian", "Right Turn Thru Involving
Parking", "Right Turn Thru Involving Overtaking"]
    left_road_out_of_control = ["Left Road - Out of Control"]
```

```
    hit_pedestrian = ["Hit Pedestrian", "Hit Pedestrian Involving Pedestrian", "Hit
Pedestrian Involving Parking", "Hit Pedestrian Involving Overtaking", "Hit Pedestrian
Involving Animal"]
    hit_fixed_object = ["Hit Fixed Object", "Hit Object Involving Parking", "Hit Object
Involving Pedestrian", "Hit Object Entering / Leaving Driveway", "Hit Object Involving
Animal"]
    hit_object= ["Hit Object on Road"]
    hit_parked_object = ["Hit Parked Vehicle"]
    hit_animal = ["Hit Animal", "Struck Animal", "Hit Animal Involving Animal", "Hit
Animal Involving Parking"]
    rollover_collision = ["Roll Over", "Vehicle overturned (no collision)"]
    collision_with = ["Collision with vehicle","Collision with a fixed object","collision
with some other object"]



    data_3 = slise_of_data[3]
    if data_3 in [None, "Other", "Other accident"]:
        return None

    if data_3 in right_angle_collision:
        return 21123  # Frontal Right Collision -> Front Airbags -> Side Airbags ->
Curtain Airbags
    elif data_3 in rear_end_collision:
        return 446137  # Rear Collision -> Seatbelt Airbags -> Rear Airbags -> Curtain
Airbags -> Center Airbags
    elif data_3 in side_swipe_collision:
        return 2123  # Right Side Collision -> Side Airbags -> Curtain Airbags
    elif data_3 in head_on_collision:
        return 111456  # Front Collision -> Front Airbags -> Knee Airbags -> Rear
Airbags -> Seatbelt Airbags
    elif data_3 in right_turn_collision:
        return 2112  # Frontal Right -> Front Airbags -> Side Airbags
    elif data_3 in left_road_out_of_control:
        return 211236  # Frontal Right Collision -> Front -> Side -> Curtain -> Seatbelt
Airbags
    elif data_3 in hit_pedestrian:
        return 1118  # Front Collision -> Front Airbags -> Pedestrian Airbags
    elif data_3 in hit_fixed_object:
```

```
        return 1114  # Front Collision -> Front Airbags -> Knee Airbags
    elif data_3 in hit_object:
        return 1114  # Front Collision -> Front Airbags -> Knee Airbags
    elif data_3 in hit_parked_object:
        return 446137  # Rear Collision -> Seatbelt Airbags -> Rear Airbags -> Curtain
Airbags -> Center Airbags
    elif data_3 in hit_animal:
        return 1114  # Front Collision -> Front Airbags -> Knee Airbags
    elif data_3 in rollover_collision:
        return 337  # Rollover -> Curtain Airbags -> Center Airbags
    elif data_3 in collision_with:
        return 1114  # Generic Collision -> Front Airbags -> Knee Airbags
    else:
        return 999  # Unknown Case
```

Серверна частина програми 6 Основна функція сервера

```
from static_analysis import static_analysis_from_db
from path_to_list import get_route_details_with_coords
from dotenv import load_dotenv
import os
from shared_memory import send_eight_longs_to_cpp, receive_floats_from_cpp
from proximity_check import is_within_2m
from what_to_do_based_on_static_analysis import what_to_do_ret_long
import logging



def reset_program():
    first_run = True
    route_events, event_coords = None
    pointer = 0

load_dotenv("API_keys.env")
api_key = os.getenv("openrouteservice_api_key")

logger1 = logging.getLogger("Logger1")
logger1.setLevel(logging.DEBUG)
file_handler1 = logging.FileHandler("GPS_DATA_help.log")
```

```
formatter1 = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s -
%(message)s")
file_handler1.setFormatter(formatter1)

logger1.addHandler(file_handler1)



first_run = True
pointer = 0
error_counter = 0
while True:
    try:

        if (first_run):
            start_point = (170.499600,-45.880834)#Нова зеланлія
            end_point = (170.815534,-45.325222)
            route_events, event_coords = get_route_details_with_coords(start_point ,
end_point, api_key)
            first_run = False
            logger1.debug(first_run)
    except Exception as e:
        logger1.error(f"{e}, Error acupaid while trying to process rout")
        send_eight_longs_to_cpp(999,999,999)
        error_counter += 1
        pointer += 1



    try:
        if pointer == 0:

            st_from_db =
static_analysis_from_db(event_coords[pointer][1],event_coords[pointer][0],route_event
s[pointer])
            if st_from_db is not None:
                logger1.debug(f"st_from_db: {st_from_db}, Length: {len(st_from_db)}")
                logger1.debug("data are fine")
                long_list = what_to_do_ret_long(st_from_db)
```

```
            logger1.debug(f"long_list: {long_list.copy()}")
            send_eight_longs_to_cpp(
               long_list[0],long_list[1],long_list[2],long_list[3],
               long_list[4],long_list[5],long_list[6],long_list[7],
                              )
            logger1.debug(f'data was successfully sent to clip part')
         else:
            logger1.error(f"data was null")
            send_eight_longs_to_cpp(999,999,999)
         pointer += 1
   except Exception as e:
      logger1.error(f"{e}, Error acupaid while trying to process path")
      send_eight_longs_to_cpp(999,999,999)
      error_counter += 1


   try:
      car_position_rn = receive_floats_from_cpp()
      if (is_within_2m(event_coords[pointer],car_position_rn)):
         st_from_db =
static_analysis_from_db(event_coords[pointer][1],event_coords[pointer][0],route_event
s[pointer])
         long_list = what_to_do_ret_long(static_analysis_from_db)
         send_eight_longs_to_cpp(
               long_list[0],long_list[1],long_list[2],long_list[3],
               long_list[4],long_list[5],long_list[6],long_list[7],
                              )
         pointer += 1
   except Exception as e:
      logger1.error(f"{e}, Error acupaid while trying to process car cordinats")
      send_eight_longs_to_cpp(999,999,999)
      error_counter += 1
   if (pointer == (len(event_coords)-1)) or (error_counter == (len(event_coords)/2)):
      reset_program()
```

Додаток А
Чип частина

Чип частина 1 Стресс мод

```
#include <stdio.h>
#include <iostream>
#include <tuple>
#include <cstdlib>  // For strtol
#include <algorithm>
#include <limits> // Include limits for std::numeric_limits
#include "check_seat_belts.cpp"

using namespace std;

long stress_mode(double v1, double v2, double v3, double v4, double v5, double
v6,double max1,double max2) {

   long what_to_do;

   std::cout << "Max1: " << max1 << ", Max2: " << max2 << "\n";
   // 5000 its a constant for a most car to strar cosing a harm to a human if nit activate
airbegs

   if (5000<(max1+max2))
   {

      auto [driver, front_passenger, left_passenger, mid_and_right_passenger] =
seat_belts_buckled();
      if ((max1 == v1 || max2 == v1) && (max1 == v2 || max2 == v2)) {
         // frontal (11)
         if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 111456;  // Front + Knee + Rear + Seatbelt Airbags
         } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 11145;  // Front + Knee + Seatbelt Airbags
         } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 1156;  // Rear + Seatbelt Airbags
```

```
        } else {
            what_to_do = 110;  // No airbags deploy
        }
    } else if ((max1 == v2 || max2 == v2) && (max1 == v3 || max2 == v3)) {
        // frontal right (21)
        if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 211456;  // Front + Knee + Rear + Seatbelt Airbags
        } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 21145;  // Front + Knee + Seatbelt Airbags
        } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 2156;  // Rear + Seatbelt Airbags
        } else {
            what_to_do = 210;  // No airbags deploy
        }
    } else if ((max1 == v3 || max2 == v3) && (max1 == v4 || max2 == v4)) {
        // rear right (34)
        if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 34156;  // Rear + Side + Curtain + Seatbelt Airbags
        } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 3415;  // Rear + Seatbelt Airbags
        } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 3456;  // Rear + Seatbelt Airbags
        } else {
            what_to_do = 340;  // No airbags deploy
        }
    } else if ((max1 == v4 || max2 == v4) && (max1 == v5 || max2 == v5)) {
        // rear (44)
        if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 449;  // Open all airbags
        } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 4456;  // Rear + Seatbelt Airbags
        } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 446;  // Rear Airbags
```

```
        } else {
            what_to_do = 440;  // No airbags deploy
        }
    } else if ((max1 == v5 || max2 == v5) && (max1 == v6 || max2 == v6)) {
        // rear left (43)
        if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 43156;  // Rear + Side + Curtain + Seatbelt Airbags
        } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 4315;  // Rear + Seatbelt Airbags
        } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 4356;  // Rear + Seatbelt Airbags
        } else {
            what_to_do = 430;  // No airbags deploy
        }
    } else if ((max1 == v1 || max2 == v1) && (max1 == v6 || max2 == v6)) {
        // frontal left (12)
        if ((driver || front_passenger) && (left_passenger || mid_and_right_passenger)) {
            what_to_do = 121456;  // Front + Knee + Rear + Seatbelt Airbags
        } else if ((driver || front_passenger) && (!left_passenger &&
!mid_and_right_passenger)) {
            what_to_do = 12145;  // Front + Knee + Seatbelt Airbags
        } else if ((!driver && !front_passenger) && (left_passenger ||
mid_and_right_passenger)) {
            what_to_do = 1256;  // Rear + Seatbelt Airbags
        } else {
            what_to_do = 120;  // No airbags deploy
        }
    } else {
        what_to_do = 999;  // Unknown case
    }
  }
  else what_to_do = 0;

  return what_to_do;
}
```

Чип частина 2 перевірка датчиків удару

```
#include <stdio.h>
#include <iostream>
#include <tuple>
#include <fstream>

bool HasTheCarBeenHit(double v1, double v2, double v3, double v4, double v5, double
v6) {
   std::ofstream file("CarHitSensor.txt", std::ios::app);

   double total = v1 + v2 + v3 + v4 + v5 + v6;
   bool allZero = (v1 == 0 && v2 == 0 && v3 == 0 &&
                   v4 == 0 && v5 == 0 && v6 == 0);

   bool result = (!allZero && total > 2000);

   file << "Total value: " << total << std::endl;
   file << "All values are 0? " << (allZero ? "Yes" : "No") << std::endl;
   file << "Hit detected? " << (result ? "Yes" : "No") << std::endl;

   file.close();
   return result;
}
```

Чип частина 3 випуск подушок безпеки

```
#include <iostream>
#include <chrono>//time to run the program
using namespace std;

void deploy(std::string value){
   cout << value << "was deployd"<< endl;
}

void deployAirbegs(long what_to_do){
   auto start = std::chrono::high_resolution_clock::now(); //time to run the program,
start
   int temp = what_to_do, divisor = 1;
   while (temp >= 10) {
```

```
    temp /= 10;
    divisor *= 10;
}

// Extract digits one by one from left to right
while (divisor > 0) {
    int digit = what_to_do / divisor; // Get the leftmost digit

    switch (digit) {
        case 1:
            deploy("Front Airbags");
            break;
        case 2:
            deploy("Side Airbags");
            break;
        case 3:
            deploy("Curtain Airbags");
            break;
        case 4:
            deploy("Knee Airbags ");
            break;
        case 5:
            deploy("Rear Airbags ");
            break;
        case 6:
            deploy("Seatbelt Airbags ");
            break;
        case 7:
            deploy("Center Airbags");
            break;
        case 8:
            deploy("Pedestrian Airbags");
            break;
        case 9:
            deploy("All");
            break;
    }
```

```
        what_to_do %= divisor; // Remove the leftmost digit
        divisor /= 10; // Move to the next digit
    }

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> duration = end - start; //time to run the program, end
    std::cout << "Program ran for " << duration.count() << " seconds." << std::endl;
}
```

Чип частина 4 Основной процесс чипа

```
#include "cpp_shared_memory.cpp"
#include "get_chip_gps_coordinates.cpp"
#include "stres_code.cpp"
#include "check_if_car_get_hit.cpp"
#include "get_shock_sensors_values.cpp"
#include <tuple>
#include "car_rollover.cpp"
#include "OutputSimulatorForTheModules/MEMS.cpp"
#include "AirbagsDeployment.cpp"
#include "find_speed_of_the_car.cpp"
#include <fstream>
#include <chrono>
#include <ctime>
#include <iomanip>
#include <iostream>

bool check_if_cases_mutch(int firstTwoDigitsFromMain[8],int value_of_the_case, int
AllExceptFirstTwoDigits[8]){
    std::ofstream file_CICM("CICM.txt", std::ios::app);
    auto now = std::chrono::system_clock::now();
    std::time_t now_time = std::chrono::system_clock::to_time_t(now);
    file_CICM << std::put_time(std::localtime(&now_time), "[%Y-%m-%d
%H:%M:%S] ");
    for (int i = 0; i < 8; i++ ){
        if( firstTwoDigitsFromMain[i] == value_of_the_case){
            file_CICM << "answer was found " << AllExceptFirstTwoDigits[i];
            deployAirbegs(AllExceptFirstTwoDigits[i]);
            return true;
        }
```

```
    }
    return false;
}


int main(int argc, char const *argv[])
{
    std::ofstream file_ACU("ACU_main_chip.txt", std::ios::app);

    while (true)
    {
        auto now = std::chrono::system_clock::now();
        std::time_t now_time = std::chrono::system_clock::to_time_t(now);
        file_ACU << std::put_time(std::localtime(&now_time), "[%Y-%m-%d
%H:%M:%S] ");
        auto [SSV1,SSV2,SSV3,SSV4,SSV5,SSV6] =
chip_shock_sensors(0,10000,2000,0,0,0);
        file_ACU << "ssv1: " << SSV1 << "\t"
            << "ssv2: " << SSV2 << "\t"
            << "ssv3: " << SSV3 << "\t"
            << "ssv4: " << SSV4 << "\t"
            << "ssv5: " << SSV5 << "\t"
            << "ssv6: " << SSV6 << "\t" << "\n";
        auto [sv1, sv2, sv3, sv4, sv5, sv6, sv7, sv8] = receive_eight_longs_from_python();
        file_ACU << "sv1: " << sv1 << "\t"
            << "sv2: " << sv2 << "\t"
            << "sv3: " << sv3 << "\t"
            << "sv4: " << sv4 << "\t"
            << "sv5: " << sv5 << "\t"
            << "sv6: " << sv6 << "\t"
            << "sv7: " << sv7 << "\t"
            << "sv8: " << sv8 << "\n";
        if (detectRollover(readFloatGyroX(0),readFloatGyroY(0))){
            deployAirbegs(9); //all
            file_ACU << "detected Rollover" << "\n";
            file_ACU << "deploying all airbags" << "\n";
        }
        else if (HasTheCarBeenHit(SSV1,SSV2,SSV3,SSV4,SSV5,SSV6)){
            if (150 < get_speed_of_the_car(40,readFloatAccelX(0))){
```

```
            deployAirbegs(9); //all
            file_ACU << "speed over 150" << "\n";
            file_ACU << "deploying all airbags" << "\n";
        }
        else{
            double max1 = numeric_limits<double>::lowest();
            double max2 = numeric_limits<double>::lowest();
            double values[] = {SSV1,SSV2,SSV3,SSV4,SSV5,SSV6};
            if ((sv1 != NULL || sv2 != NULL || sv3 != NULL || sv4 != NULL || sv5 !=
NULL || sv6 != NULL || sv7 != NULL || sv8 != NULL))
            {
                // Find 2 max values
                for (double v : values) {
                    if (v > max1) {
                        max2 = max1;
                        max1 = v;
                    } else if (v > max2) {
                        max2 = v;
                    }
                }

                file_ACU << "max value 1: " << max1 << " max value 2: " << max2 <<
"\n";
                int numbers[] = {sv1,sv2,sv3,sv4,sv5,sv6,sv7,sv8}; // Input numbers
                int firstTwoDigits[8];
                int AllExceptFirstTwoDigits[8];

                file_ACU << "Starting the process of reviewing the instructions." << "\n";
                // reducing values to 2 digits, and writing them in first Two Digits
                for (int i = 0; i < 8; i++) {
                    int num = numbers[i];

                    // Find the number of digits in the number
                    int numDigits = 0;
                    int temp = num;
                    while (temp > 0) {
                        numDigits++;
                        temp /= 10;
                    }
```

```
                // Extract the first two digits
                int firstTwo = num / pow(10, numDigits - 2);

                // Extract the remaining digits
                int rest = num % static_cast<int>(pow(10, numDigits - 2));

                firstTwoDigits[i] = firstTwo;
                AllExceptFirstTwoDigits[i] = rest;
            }
            file_ACU << "Direction of the lesion " << firstTwoDigits << "\n";
            file_ACU << "Instruction " << AllExceptFirstTwoDigits << "\n";

            bool DesichionFound;
            if ((max1 == SSV1 || max2 == SSV1) && (max1 == SSV2 || max2 ==
SSV2)) {
                // frontal (11)
                file_ACU << "frontal collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,11,AllExceptFirstTwoDigits);
            } else if ((max1 == SSV2 || max2 == SSV2) && (max1 == SSV3 || max2
== SSV3)) {
                // frontal right (21)
                file_ACU << "frontal right collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,21,AllExceptFirstTwoDigits);
            } else if ((max1 == SSV3 || max2 == SSV3) && (max1 == SSV4 || max2
== SSV4)) {
                // rear right (34)
                file_ACU << "rear right collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,34,AllExceptFirstTwoDigits);
            } else if ((max1 == SSV4 || max2 == SSV4) && (max1 == SSV5 || max2
== SSV5)) {
                // rear (44)
                file_ACU << "rear collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,44,AllExceptFirstTwoDigits);
```

```
            } else if ((max1 == SSV5 || max2 == SSV5) && (max1 == SSV6 || max2
== SSV6)) {
                // rear left (43)
                file_ACU << "rear left collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,43,AllExceptFirstTwoDigits);
            } else if ((max1 == SSV1 || max2 == SSV1) && (max1 == SSV6 || max2
== SSV6)) {
                // frontal left (12)
                file_ACU << "frontal left collision";
                DesichionFound =
check_if_cases_mutch(firstTwoDigits,12,AllExceptFirstTwoDigits);
            }
            else{
                stress_mode(SSV1,SSV2,SSV3,SSV4,SSV5,SSV6,max1,max2);
            }

            if (DesichionFound == false){
                stress_mode(SSV1,SSV2,SSV3,SSV4,SSV5,SSV6,max1,max2);
            }
        }
        else{
            long what_to_do =
stress_mode(SSV1,SSV2,SSV3,SSV4,SSV5,SSV6,max1,max2);
            deployAirbegs(what_to_do);
        }

        auto [lon,lan] = chip_gps_coord_rn(53.0758,8.8078);
        send_floats_to_python(lan,lon);
      }
    }
    //_sleep(2000); //WARNING : USE ONLY FOR TEST CASES
  }
  file_ACU.close();
  return 0;
}
```

Чип частина 5 знаходження швидкості автомобіля

```
#include "OutputSimulatorForTheModules/GPS.cpp"
#include <cmath>  // For fabs()

const float ACCEL_THRESHOLD = 0.1;  // Ignore small accelerations (in g)
const float TIME_STEP = 0.01;       // Time step in seconds (10ms loop)
const float G_TO_MS2 = 9.81;        // Convert g to m/s²


// Function to calculate speed in km/h
float getSpeedUsingAcceleration(float accelX) {
   float velocity = 0.0; // Speed in m/s
   float acceleration_m_s2 = accelX * G_TO_MS2;  // Convert g to m/s²

   // Ignore small accelerations to reduce drift
   if (fabs(acceleration_m_s2) < ACCEL_THRESHOLD) {
      acceleration_m_s2 = 0;
   }

   // Integrate acceleration to get velocity
   velocity += acceleration_m_s2 * TIME_STEP;

   // Convert m/s to km/h
   return velocity * 3.6;
}


float get_speed_of_the_car(float speed_for_the_gps_moduel,float accelX){
   if (SpeedSsValid(kmh(speed_for_the_gps_moduel))){
      return kmh(speed_for_the_gps_moduel);
   }
   else{
      return getSpeedUsingAcceleration(accelX);
   }

}
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

Інформаційна технологія прогнозування спрацювання подушок безпеки для автомобільного транспорту на основі статистичного аналізу даних

Мацегора Петро Васильович ІСТ-21д

Керівник проєкту зав. каф. В.О.Лифар

1

Рисунок Б.1 - Слайд 1

Актуальність теми

Щороку внаслідок дорожньо-транспортнх травм передчасно втрачають життя приблизно 1,19 млн осіб. Ще від 20 до 50 млн осіб зазнають травм. Це не завжди призводять до летальних наслідків, проте часто супроводжується інвалідністю.

Дорожньо-транспортні травми спричиняють значні економічні втрати як для окремих осіб і їхніх родин, так і для держав загалом. Ці втрати обумовлені витратами на медичне лікування, втратою продуктивності праці осіб, які загинули або отримали інвалідність у наслідок травм, і членів їхніх сімей, які змушені переривати роботу чи навчання для догляду за постраждалими. У більшості країн втрати від ДТП сягають 3% валового внутрішнього продукту.

Мета цієї дипломної роботи - розробити статистичний аналіз, який допоможе заздалегідь передбачити дорожні аварій для автомобільних засобів. Надавати системі подушок безпеки більше часу на відкриття, і знаходження шляху мінімізації ушкоджень для пасажирів.

2

Рисунок Б.2 - Слайд 2

Для виконання поставленої роботи у дипломному проекті необхідно провести та аналізувати

- Розглянути методи спрацювання подушок безпеки
- Створити базу знань, зі швидким але детальми доступом до даних
- Створити серверну части для аналізу даних, і їх подальший відправці до автомобіля.
- Розробити та провести тестування ПЗ для подушок безпеки, з урахуванням її срацювання у межах 200 мілісекунд

3

Рисунок Б.3 - Слайд 3

Алгоритм системи

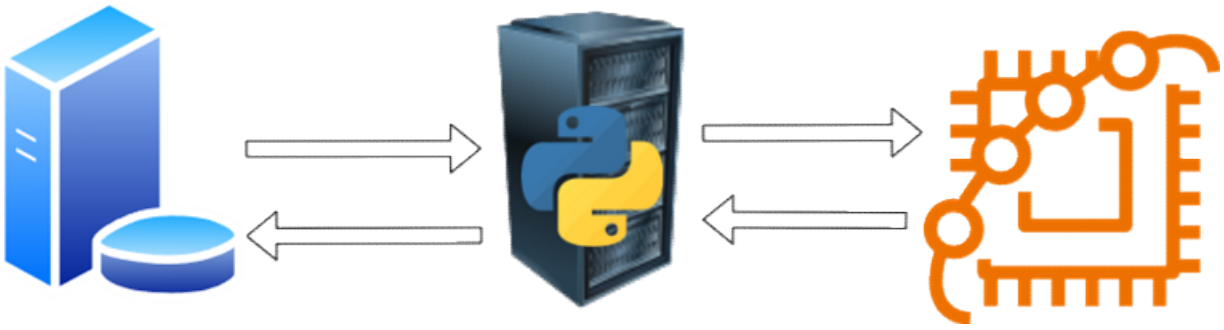

Рисунок -Алгоритм передачі даних

4

Рисунок Б.4 - Слайд 4

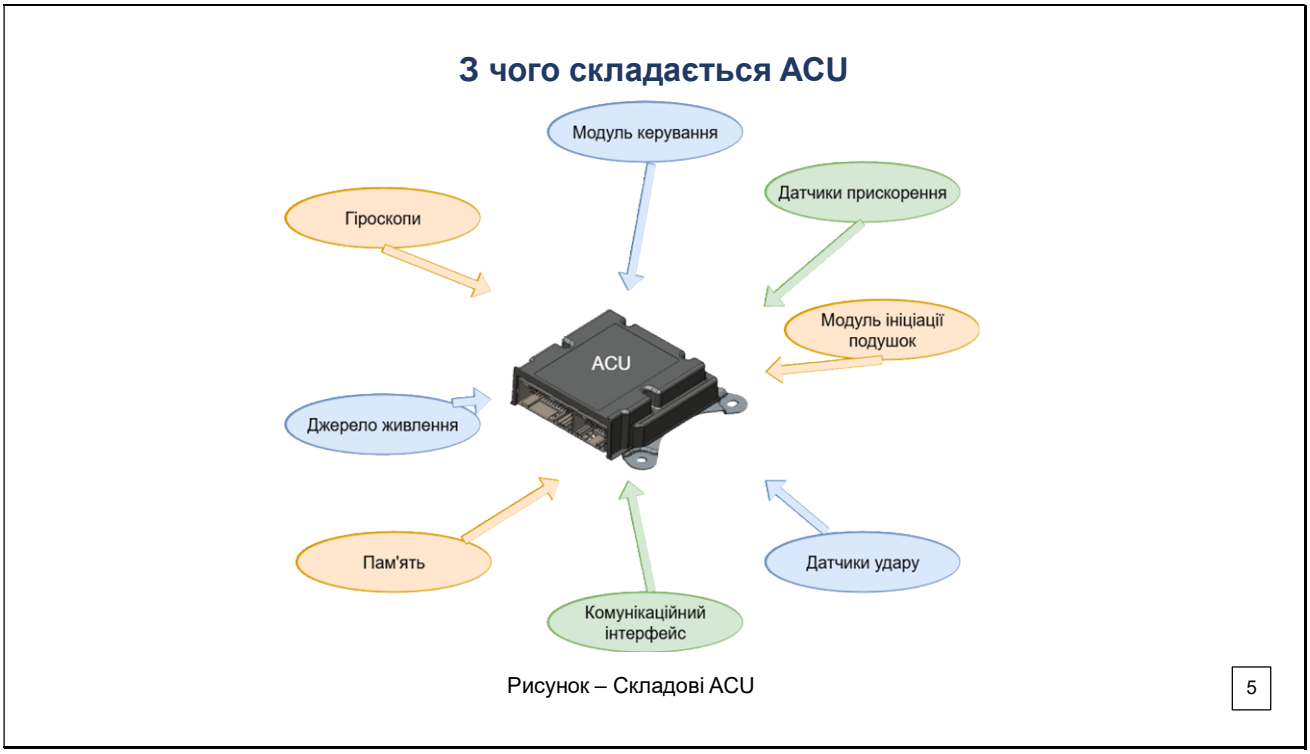

Рисунок Б.5 - Слайд 5

Рисунок Б.6 - Слайд 6

Структура бази знань

1. Ідентифікаційна інформація (Car_crash_ID)
2. Час і дата події (Day_of_week, Hour_of_crash)
3. Місце аварії (Midlock, Intersection)
4. Характер аварії (Severity, Crash_type, Type_of_vehicles, Total_of_vehicles)
5. Наслідки аварії (Fatalities, Serious_injuries, Minor_injuries)
6. Умови дороги та навколишнього середовища (Speed_limit, Road_position_horizontal, Road_position_vertical, Road_slope, Road_wet)
7. Місце аварії (Weather, Lighting, Traffic_controls)

7

Рисунок Б.7 - Слайд 7

Алгоритм роботи сервера

Рисунок – Блок-схема роботи сервера

8

Рисунок Б.8 - Слайд 8

Алгоритм роботи ACU

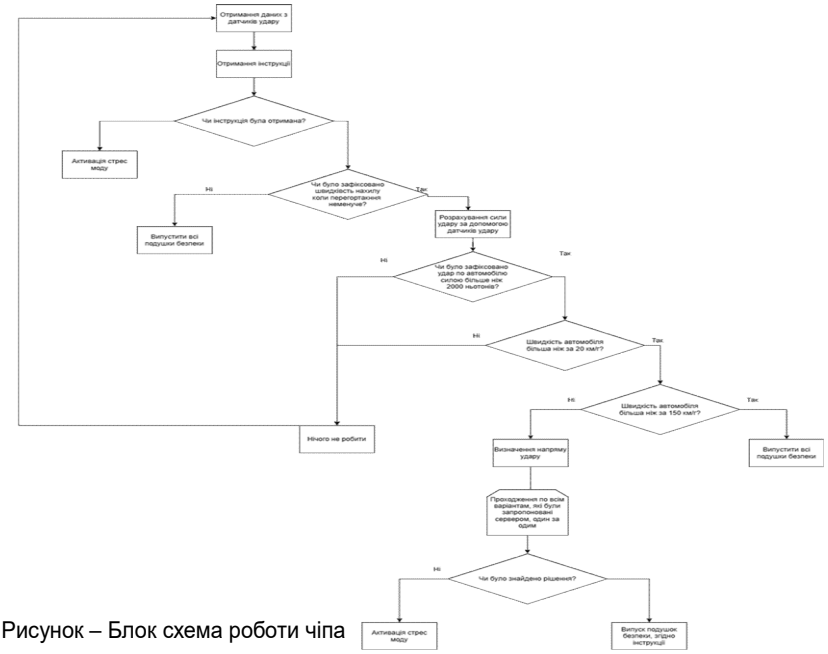

Рисунок – Блок схема роботи чіпа

9

Рисунок Б.9 - Слайд 9

Висновки

- Були розглянуті основні методи праці та компоненти системи подушок безпеки
- Була створення база знань, яка надає можливість легко збільшувати і зменшувати деапазон збору та застосувань бази знань.
- Було розроблено серверне програмне забезпечення, на основі програмної мови Python. Яке збирає дані з відкритих джерел, на їх основі аналізу статистичну ймовірність аварії. І створює інструкцію яка в подальшому відпраляється в систему управління подушками безпеки.
- Було розроблено симуляцію для системи управління подушками безпеки. Яка включає в себе як і можливість полагаться на інструкції надані сервером. Так і діяти самостійно
- Результати дослідження та розроблення, можуть застосовуватися в автомобілях безпеки, для зменшення кількості постраждалих. Також на основі бази знань, може проводитися змінення дорожніх умов, для значного зменшення кількості аварії, на участках з великою смертельністю.

10

Рисунок Б.10 - Слайд 10