# Лабораторна робота № 4

# Дослідження алгоритмів обробки даних супутникових вимірів

**Мета:** вироблення навичок роботи з навігаційними даними, використання сучасних GPS-технологій

**1 Теоретичне введення**

Для зв'язку з COM-порту комп'ютера доцільно використовувати драйвер інтерфейсу RS232 / RS485.

Опис

Драйвер інтерфейсу RS232 / RS485 дозволяє програмі здійснювати комунікації з відповідного послідовному каналу зв'язку. Драйвер відповідає за прийом і передачу блоків даних різної довжини. Обробляє апаратні і програмні помилки передачі. Драйвер дозволяє вести журнал подій протоколу RS232 / RS485. Основні налаштування винесені в конфігураційний файл net\_rs232.ini, який повинен знаходитися в каталозі виконуваного файлу НУ. Якщо такий файл відсутній під час запуску програми НУ, він автоматично створюється з настройками за замовчуванням.

## Налаштування

Файл налаштування драйвера RS232 / RS485 має наступну структуру:

[ComPort]

name = COM2

init = BAUD = 38400 PARITY = E DATA = 7 STOP = 1 TO = ON DTR = ON RTS = ON OCTS = OFF ODSR = OFF

receive\_timeout\_interval = 1000

receive\_timeout\_constant = 1000

logging = 1

logging\_rxtx = 1

В поле name має бути зазначено ім'я комунікаційного порту, до якого підключається драйвер, як правило, це ім'я починається на 'COM'.

В поле init вказується рядок ініціалізації для комунікаційного порту. Цей рядок повинна відповідати формату команди mode операційної системи (для довідки виконайте в консолі help mode). Поля рядки ініціалізації означають: BAUD - швидкість передачі даних в каналі, бод, як правило, може приймати такі значення: 300, 600, 1200, 2400, 4800, 9600, 19200, 34800, 57600, 115200 і більше, однак може мати і інші значення; PARITY - тип контролю парності: N - немає контролю, E - чет, O - непарне, M - маркер; DATA - вказує кількість переданих біт в байті даних: 5, 6, 7 або 8; STOP - вказує на кількість степових біт при передачі даних: 1, 1.5 або 2; інші параметри вказують на стан допоміжних ліній комунікаційного порту і реакцію драйвера операційної системи на зміну їх стану.

В поле receive\_timeout\_interval вказується максимальний інтервал часу (мс) між прийомом одного байта і прийомом наступного байта, після закінчення якого драйвер фіксує відмову.

В поле receive\_timeout\_constant вказується інтервал часу (мс), після закінчення якого операція читання з комунікаційного порту переривається з таймаут прийому. Іншими словами, це максимальний час очікування відповіді від віддаленого пристрою. Якщо після закінчення цього часу пакет даних не буде прийнятий - драйвер фіксує відмову.

В поле logging вказується, чи повинен драйвер вести журнал протоколу протокол. Якщо logging = 1, то драйвер створює файл протоколу і записує в нього події про помилки передачі. Якщо при цьому також logging\_rxtx = 1, драйвер протоколює в журналі все передачі даних по комунікаційному каналу.

1. **Порядок виконання роботи:**

1. Грунтуючись на вихідних даних виконати наступні дії:

- Підключити GPS-приймач, (або віртуальний GPS) до одного з портів робочої станції. Перевірити надходження сигналу з відповідного порту.

- Розробити програмний проект в середовище , який забезпечить прийом даних з апаратури споживача супутникової навігації (або віртуального GPS), їх обробку і відображення даних вимірювань за допомогою растрових даних, ґрунтуючись на результатах виконання роботи 2, попередньо визначивши номенклатуру картографічного матеріалу.

1. Варіант на «20» - розроблений програмний проект повинен дозволити проводити статистичну обробку результатів вимірювань.
2. Оформити звіт про лабораторну роботу, що містить
* завдання;
* лістинг програми;
* екранні форми;
* результати у вигляді зображення карти і траєкторії польоту
* висновки.

**Методичні вказівки**

* 1. Необхідно створити проект, що дозволяє відобразити місце знаходження приймача GPS, або даних створених за допомогою віртуального GPS приймача.
	2. Для створення віртуального приймача GPS необхідно
* Встановити програму Virtual Serial Port Driver та зв’язати Com порти, як показано на рисунку



Рисунок 1 – Зв’язка Сom портів 2 та 4

Запустити програму NMEAGEN.exe та зробити необхідні налаштування для запису повідомлень NMEA у Сom порт 2.



Рисунок 2 – Налаштування Сom порту 2

* Запустити програму NmeaMon, та перевірити наявність повідомлень NMEA у Сom порту 4



Рисунок 3 –Контроль повідомлень у Сom порту 4

* зупинити програму NmeaMon.

2. Для зчитування даних з COM-порту використовуйте бібліотеку PySeries:



* 1. Для відображення даних використовуйте модуль folium



**. Звіт по лабораторній роботі повинен містити:**

1. мету й короткий теоретичний опис лабораторної роботи;
2. вихідні (вхідні) дані для розрахунків;
3. для пунктів індивідуального завдання: експериментально отримані результати розрахунків, графіки й лістинг операцій у використовуваному програмному середовищі;
4. аналіз отриманих результатів і висновки.
5. відповіді на контрольні питання.

Контрольні питання і завдання

1. Переваги та недоліки растрових карт.

2. Особливості векторних карт.

3. Які ви знаєте формати векторних карт?

4. Що таке географічна інформаційна система (ГІС)?

5. Як організовано отримання даних з GPS- приймача за допомогою COM-порту?

6. Як виконується прив'язка листакарти?

**Література**

1. Гофман-Веленгоф Б. Лихтенегер Г. Глобальная система супутникової навігації. Теорія і практика. Київ: Наукова думка, 1996

2. Яценков В.С. Основы спутниковой навигации. М: Горячая линия – Телеком, 2005, 272с.

3. Шебшевич В.С. Сетевые спутниковые радионавигационные системы. М.: Радио и связь. 1986

Додаток

*Приклад автоматичної обробки повідомлень*

*Лістинг модуля message.py:*

*"""This file contain main message for work"""*

*MESSAGE = (*

*"0GPGGA,060355.00,4950.4828,N,03638.5901,E,1,05,4.3,144.5,M,16.2,M,,*

*\*67",*

*"0GPGGA,060356.00,4950.4837,N,03638.5901,E,1,05,4.3,144.6,M,16.2,M,,*

*\*69",*

*"0GPGGA,060357.00,4950.4845,N,03638.5899,E,1,05,4.3,144.3,M,16.2,M,,*

*\*68",*

*"0GPGGA,060358.00,4950.4838,N,03638.5906,E,1,05,4.3,143.7,M,16.2,M,,*

*\*69",*

*"0GPGGA,060359.00,4950.4838,N,03638.5910,E,1,05,4.3,143.9,M,16.2,M,,*

*\*61",*

*"0GPGGA,060400.00,4950.4835,N,03638.5911,E,1,05,4.3,144.6,M,16.2,M,,*

*\*6E",*

*"0GPGGA,060401.00,4950.4831,N,03638.5912,E,1,04,14.7,145.4,M,16.2,M,*

*,\*5F",*

*"0GPGGA,060402.00,4950.4825,N,03638.5913,E,1,04,14.7,146.4,M,16.2,M,*

*,\*5B",*

*"0GPGGA,060403.00,4950.4824,N,03638.5912,E,1,04,14.7,146.1,M,16.2,M,*

*,\*5F",*

*"0GPGGA,060404.00,4950.4822,N,03638.5911,E,1,04,14.7,145.6,M,16.2,M,*

*,\*59",*

*"0GPGGA,060405.00,4950.4820,N,03638.5911,E,1,04,14.7,145.2,M,16.2,M,*

*,\*5E",*

*"0GPGGA,060406.00,4950.4818,N,03638.5910,E,1,04,14.7,144.8,M,16.2,M,*

*,\*5C",*

*"0GPGGA,060407.00,4950.4818,N,03638.5910,E,1,04,14.7,144.4,M,16.2,M,*

*,\*51",*

*"0GPGGA,060408.00,4950.4817,N,03638.5909,E,1,04,14.7,144.0,M,16.2,M,*

*,\*5D",*

*"0GPGGA,060409.00,4950.4817,N,03638.5909,E,1,04,14.7,143.6,M,16.2,M,*

*,\*5D",*

*"0GPGGA,060410.00,4950.4817,N,03638.5909,E,1,04,14.7,143.2,M,16.2,M,*

*,\*51",*

*"0GPGGA,060411.00,4950.4817,N,03638.5909,E,1,04,14.7,142.8,M,16.2,M,*

*,\*5B",*

*"0GPGGA,060412.00,4950.4817,N,03638.5909,E,1,04,14.7,142.5,M,16.2,M,*

*,\*55",*

*"0GPGGA,060413.00,4950.4817,N,03638.5909,E,1,04,14.6,142.2,M,16.2,M,*

*,\*52",*

*"0GPGGA,060414.00,4950.4816,N,03638.5909,E,1,04,14.6,141.8,M,16.2,M,*

*,\*5D",*

*)*

*Лістинг коду модуля base.py:*

*"""This file created for working with navigation data"""*

*from datetime import timedelta*

*from math import sin, cos, atan, sqrt*

*# For example*

*#*

*0GPGGA,060355.00,4950.4828,N,03638.5901,E,1,05,4.3,144.5,M,16.2,M,,\*67*

*# Where next elements*

*# 0GPGGA - command about write current message*

*# 060355.00 - 06 hours 03 minutes 55.00 seconds*

*# 4950.4828 - latitude: 49 degrees 50.4828 minutes*

*# N - Nord*

*# 03638.5901 - longitude: 036 degrees 38.5901 minutes*

*# E - East*

*# 1 - trash*

*# 05 - trash*

*# 4.3 - trash*

*# 144.5 - height*

*# M - meters*

*# 16.2 - trash*

*# \*67 - control summ message in 0x16 by module 16*

*from message import MESSAGE*

*EARTH\_RADIUS\_IN\_KM = 6372.795*

*EARTH\_RADIUS\_IN\_M = 6372795*

*class Aircraft:*

*"""*

*Main class aircraft with data on fly*

*"""*

*def \_\_init\_\_(self, message) -> None:*

*self.\_message = message*

*self.\_list\_commands: list = []*

*self.\_list\_time: list = []*

*self.\_list\_latitude: list = []*

*self.\_latitude: str = ""*

*self.\_list\_longtitude: list = []*

*self.\_longtitude: str = ""*

*self.\_list\_height: list = []*

*self.search\_data\_for\_lists()*

*def search\_data\_for\_lists(self) -> None:*

*"""*

*Get all elements for data flight*

*"""*

*for element in self.\_message:*

*info = element.split(',')*

*self.local\_append(self.\_list\_commands, info[0])*

*self.local\_append(self.\_list\_time, info[1])*

*self.local\_append(self.\_list\_latitude, info[2])*

*self.\_latitude = info[3]*

*self.local\_append(self.\_list\_longtitude, info[4])*

*self.\_longtitude = info[5]*

*self.local\_append(self.\_list\_height, info[9])*

*return None*

*@staticmethod*

*def local\_append(local\_attr: list, value\_to\_append: str) -> None:*

*"""*

*Append value to current list*

*"""*

*local\_attr.append(value\_to\_append)*

*return None*

*@staticmethod*

*def get\_list\_with\_float(list\_with\_str: list) -> list:*

*"""*

*Get list with float elements from list with str's*

*"""*

*return [float(element) for element in list\_with\_str]*

*def get\_max\_hight\_value(self) -> float:*

*"""*

*Get max value of height fly by list of message heights*

*"""*

*return max(self.get\_list\_with\_float(self.\_list\_height))*

*def get\_min\_hight\_value(self) -> float:*

*"""*

*Get min value of height fly by list of message heights*

*"""*

*return min(self.get\_list\_with\_float(self.\_list\_height))*

*@staticmethod*

*def parser\_time(str\_with\_time: str) -> tuple:*

*"""*

*Get time info from string*

*"""*

*hours = int(str\_with\_time[:2])*

*minutes = int(str\_with\_time[2:4])*

*seconds = float(str\_with\_time[4:])*

*return hours, minutes, seconds*

*def get\_all\_time\_fly(self):*

*"""*

*Get summ of time in fly*

*"""*

*time\_start = self.\_list\_time[0]*

*time\_finish = self.\_list\_time[-1]*

*tuple\_start = self.parser\_time(time\_start)*

*tuple\_finish = self.parser\_time(time\_finish)*

*start = timedelta(hours=tuple\_start[0],*

*minutes=tuple\_start[1], seconds=tuple\_start[2])*

*finish = timedelta(hours=tuple\_finish[0],*

*minutes=tuple\_finish[1], seconds=tuple\_finish[2])*

*time\_in\_fly = finish - start*

*return time\_in\_fly*

*def get\_sum\_flight\_distance(self):*

*"""*

*Get current summa all flight distance*

*"""*

*sum\_flight\_distance = 0*

*for distance in self.get\_list\_distances\_per\_signal():*

*sum\_flight\_distance += distance*

*return sum\_flight\_distance*

*def get\_list\_distances\_per\_signal(self):*

*"""*

*Get list with elements, where on one signal -> one distance*

*"""*

*list\_of\_distances = []*

*fo\_1\_rad = 0*

*la\_1\_rad = 0*

*fo\_2\_rad = 0*

*la\_2\_rad = 0*

*for index in range(len(self.\_list\_latitude)):*

*if index >= 19:*

*break*

*fo\_1\_rad = self.get\_radians(self.\_list\_latitude[index])*

*la\_1\_rad =*

*self.get\_radians(self.\_list\_longtitude[index][1:])*

*fo\_2\_rad = self.get\_radians(self.\_list\_latitude[index +*

*1])*

*la\_2\_rad = self.get\_radians(self.\_list\_longtitude[index+*

*1][1:])*

*list\_of\_distances.append(self.get\_distance\_in\_meters(fo\_1\_rad, la\_1\_rad,*

*fo\_2\_rad, la\_2\_rad))*

*return list\_of\_distances*

*@staticmethod*

*def get\_radians(value):*

*"""*

*Get radians value from degrees and minutes*

*"""*

*degrees: float = float(value[:2])*

*minutes: float = float(value[2:])*

*radians = degrees + (minutes / 60)*

*return radians*

*@staticmethod*

*def get\_distance\_in\_meters(rad\_fo\_1, rad\_la\_1, rad\_fo\_2,*

*rad\_la\_2):*

*"""*

*Get value of distance by formula*

*"""*

*delta\_fo = rad\_fo\_2 - rad\_fo\_1*

*delta\_la = rad\_la\_2 - rad\_la\_1*

*delta\_ce\_de = atan(sqrt((cos(rad\_fo\_2)\*sin(delta\_la))\*\*2 +*

*(cos(rad\_fo\_1)\*sin(rad\_fo\_2) -*

*sin(rad\_fo\_1)\*cos(rad\_fo\_2)\*cos(delta\_la))\*\*2)/((sin(rad\_fo\_1)\*sin(rad\_fo\_*

*2) + cos(rad\_fo\_1)\*cos(rad\_fo\_2)\*cos(delta\_la))))*

*distance = EARTH\_RADIUS\_IN\_KM \* delta\_ce\_de*

*return distance \* 1000*

*def get\_middle\_speed(self):*

*"""*

*Get flight middle speed by all time in fly*

*"""*

*all\_distance = int(self.get\_sum\_flight\_distance())*

*time\_in\_second = int(self.get\_all\_time\_fly().seconds)*

*return all\_distance / time\_in\_second*

*def get\_delta\_in\_signals(self):*

*"""*

*Get delta in seconds by all distans for seconds*

*"""*

*all\_time\_in\_sec = int(self.get\_all\_time\_fly().seconds)*

*count\_signals = int(len(self.\_list\_commands))*

*return count\_signals / all\_time\_in\_sec*

*def get\_min\_max\_speed\_fly(self):*

*"""*

*Get dict with minimal and maximum speed on fly*

*"""*

*delta\_signal\_in\_second = self.get\_delta\_in\_signals()*

*speed\_info = {*

*"min": 1000.0,*

*"max": 0.0,*

*}*

*for distance in self.get\_list\_distances\_per\_signal():*

*current\_speed = distance / delta\_signal\_in\_second*

*if current\_speed > speed\_info["max"]:*

*speed\_info["max"] = current\_speed*

*if current\_speed < speed\_info["min"]:*

*speed\_info["min"] = current\_speed*

*return speed\_info*

*def base():*

*"""*

*Base method for show all data info*

*"""*

*aircraft = Aircraft(message=MESSAGE)*

*print(f"Сумарна довжина маршруту польоту:*

*{round(aircraft.get\_sum\_flight\_distance(), 2)} метрів.")*

*print(f"Середня швидкість польоту:*

*{round(aircraft.get\_middle\_speed(), 2)} м/c.")*

*print(f"Мінімальна швидкість польоту:*

*{round(aircraft.get\_min\_max\_speed\_fly()['min'], 2)} м/c.")*

*print(f"Максимальна швидкість польоту:*

*{round(aircraft.get\_min\_max\_speed\_fly()['max'], 2)} м/c.")*

*print(f"Мінімальна висота польоту:*

*{round(aircraft.get\_min\_hight\_value(), 2)} метрів.")*

*print(f"Максимальна висота польоту:*

*{round(aircraft.get\_max\_hight\_value(), 2)} метрів.")*

*print(f"Загальний час усього польоту:*

*{aircraft.get\_all\_time\_fly()}.")*

*if \_\_name\_\_ == "\_\_main\_\_":*

*base()*