Київський національний університет імені Тараса Шевченка
Механіко-математичний факультет

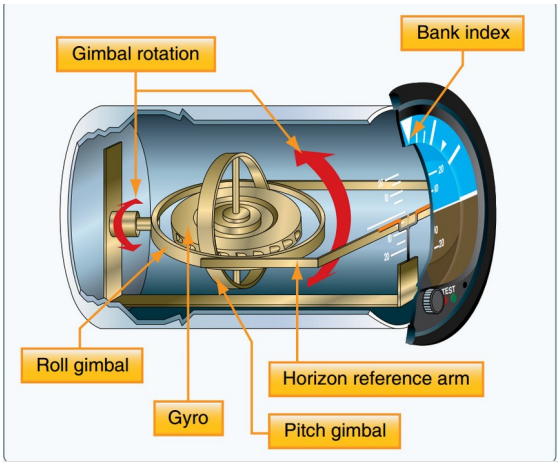
Курсова робота за темою:
“Алгоритми фільтрації зашумлених даних MEMS-акселерометрів”

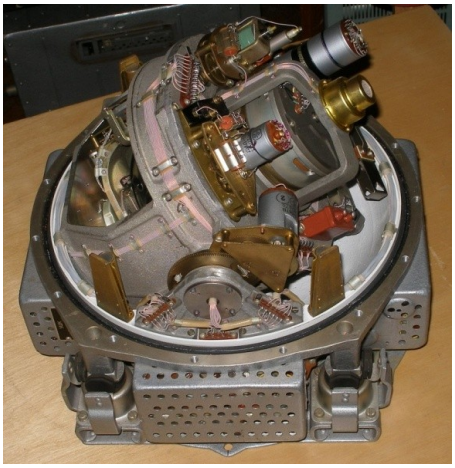
Виконав:
Студент 3-го курсу
Групи ММО спеціальності
“Математика 111”
Аніщенко Роман Сергійович

Зміст:

1 Вступ

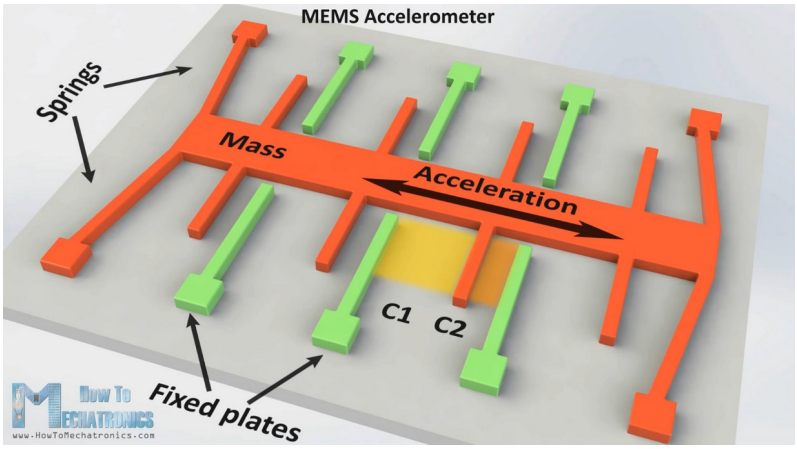
Задача відновлення орієнтації об’єкта в просторі зустрічається в широкому спектрі галузей, таких як авіація, робототехніка, віртуальна реальність, тощо. Вимоги до точності та частоти оновлення даних відрізняються, проте часто є досить високими (частоти порядку 1 кГц та похибки меньше одного градуса).





В авіації давно використовується так званий авіагоризонт, що має механічний гіроскоп задля збереження орієнтації літаючого апарату відносно землі з моменту зльоту. Проте подібні прибори достатньо громіздкі (маса порядку кількох кілограмів) та дорогі, що мало підходить для багатьох галузей, наприклад малої автономної авіації (дрони) та споживацької електроніки (телефони, пристрої VR/AR).

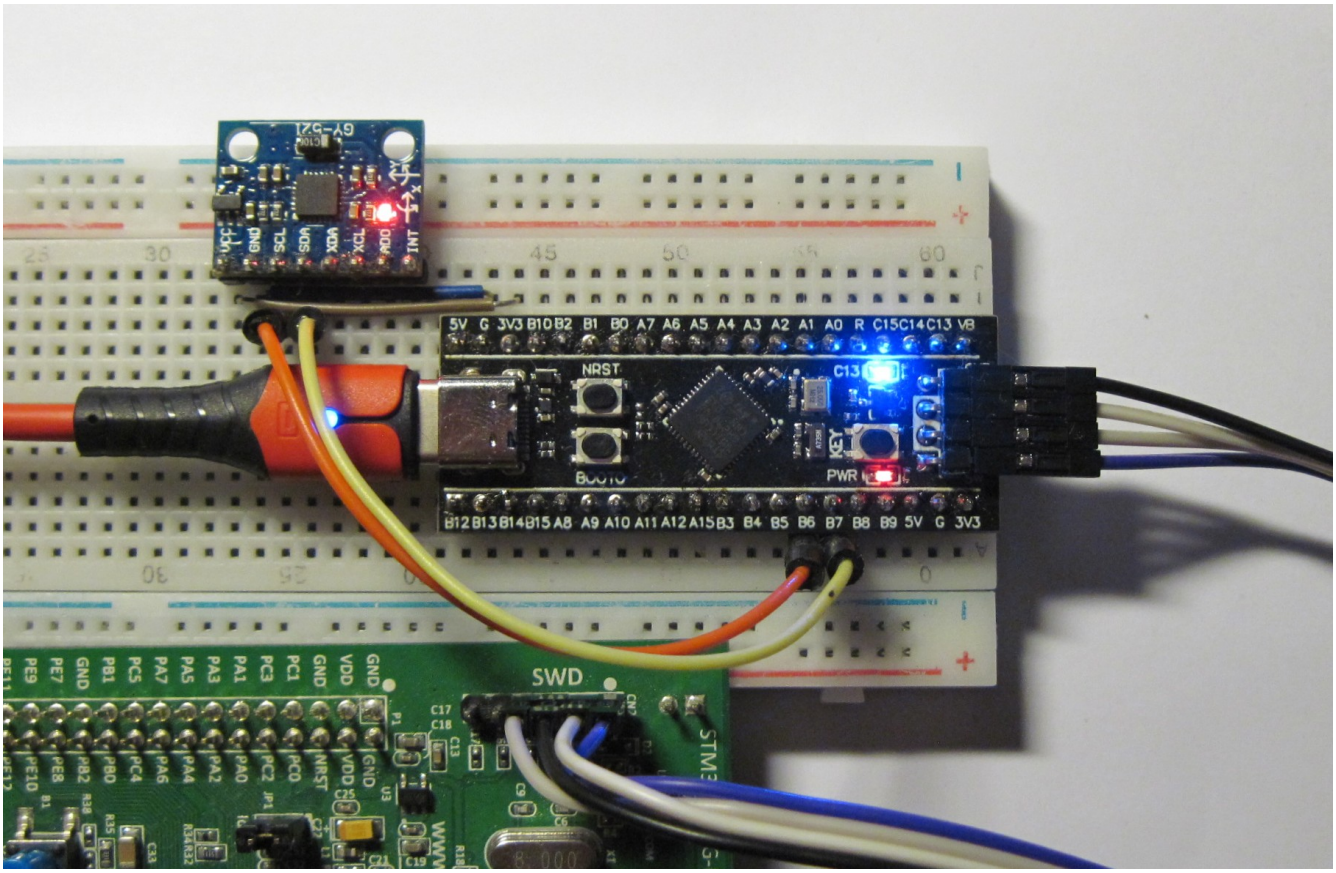
Для таких випадків застосовують набагато менші **мікромеханічні датчики**, **microelectromechanical systems** (далі **MEMS**). За допомогою технологій обробки кремнію, подібних до тих, за якими створюються інтегральні мікросхеми, можна створити датчики розміром 1-100 мікрометрів, точність яких зазвичай гірша за “класичні” аналоги, проте після належної обробки цілком достатня для багатьох застосувань, що і буде показано в даній роботі.



Принцип роботи даних пристроїв полягає у вимірюванні зміни електричних характеристик (наприклад, електричної ємності) від дії зовнішніх сил інерції, під час обертання чи прискорення. Також в таких системах часто присутній аналіз даних про напрямок магнітного поля (тривимірний компас), проте в даній роботі вони не розглядаються.

1.2 Щодо використаної апаратної частини

Алгоритми були реалізовані на мікроконтролері (далі МК) **STM32F401**, що є досить популярним сучасним МК, мовами C та C++. В якості датчика використано MPU-6050, в якому вбудовано 3-х осьові гіроскоп та акселерометр, загалом названі IMU (inertial measurement unit) в роботі. Модель зібрана на макетній платі, оскільки це все, насправді, не має бути центральною темою курсової роботи.



2 Чисельне інтегрування та чому необхідні алгоритми поєднання даних (sensor fusion)

Розглянута в роботі категорія датчиків – інерційні (IMU) не можуть вимірювати абсолютне положення чи орієнтацію об’єкта, до якого прикріплені, натомість:

- Гіроскопи вимірюють кутову швидкість, яка є першою похідною від куту положення. $\vec{\omega}=\frac{d\vec{\varphi}}{dt}$
- Акселерометри вимірюють прискорення.
  $\vec{a}=\frac{d\vec{v}}{dt}=\frac{d^2\vec{x}}{dt^2}$
  В тому числі показом такого датчика в положенні спокою (або при рівномірному русі) на землі має бути прискорення, направлене “вниз” (до землі), рівне прискоренню вільного падіння. Таким чином можно визначити орієнтацію тіла, але лише вважаючи, що тіло не рухається або рухається рівномірно, що зовсім не так для більшої частини застосувань. Однак, для розробки фільтрів часто робиться припущення, що це є правдою для частини вимірів.

Через це виникає проблема: мікромеханічні датчики мають досить високий рівень шуму (досліджений далі), до того ж розподіленого дещо несиметрично відносно істинних значень. Тому чисельне інтегрування будь-якого показу без необхідної корекції додає до результату зміщення, яке зростає з часом (в деякій літературі його називають **кумулятивною похибкою**, тобто такою, що накопичується). Подібне зміщення необхідно коригувати, для цього необхідний “допоміжний” датчик, який дає меньш точні, але абсолютні покази, які з часом дозволяють запобігти відхиленню інтегрованої величини від справжнього значення.

Алгоритми такої корекції називають алгоритмами sensor fusion, адже вони справді поєднують покази кількох приладів.

2.1 Вимірювання орієнтації тіла

В цій роботі розглядається задача відновлення орієнтації (кутів відносно координатних осей) тіла за допомогою IMU.

В якості “точного” датчика, покази якого необхідно інтегрувати, береться гіроскоп. В якості датчика, що дає абсолютні покази, береться направлення вектора прискорення, що вимірюється акселерометром, вважаючи, що прискорення вільного падіння (вниз) є головною компонентою показів.

2.2 Вимірювання положення тіла

Однак задача відновлення положення (координат тіла в деякій системі координат) є значно важчою за використання IMU, оскільки, як було сказано раніше, кумулятивна похибка накопичується квадратично (за рахунок подвійного чисельного інтегрування прискорення), а також немає способу отримати абсолютні покази положення, не використовуючи сторонні системи стеження на основі компьютерного бачення або інших технологій, всі з яких є досить складними та/або дорогими, тому ця задача не розглядається в роботі.

3 Отримання даних

Опитування датчику відбувається з частотою в 1 кГц, відповідно *dt* – різниця в часі між кроками k-1 та k – для усіх наступних формул вважається рівною 1 [мс].

Той факт, що отримані дані містять шум, виразимо у вигляді випадкових величин ξ_k та ζ_k, що додаються до справжніх кутової квидкості $\breve{\omega}$ та прискорення $\breve{a}$ відповідно (позначимо її “перевернутим дашком”).

$$\omega_k = \breve{\omega}_k + \xi_k$$
$$a_k = \breve{a}_k + \zeta_k$$

Чисельне інтегрування кутової швидкості ω для отримання куту φ з з показів гіроскопу на кроці k виражається як:

$$\varphi_k = \varphi_{k-1} + \omega_k \cdot dt\,, k \geqslant 1$$

При цьому початкове положення системи φ_0 визначається або при ініціалізації алгоритму, або як 0 радіан, бо за декілька ітерацій це зміщення має зникнути, оскільки вихідні дані фільтру мають збігатися до істинних.

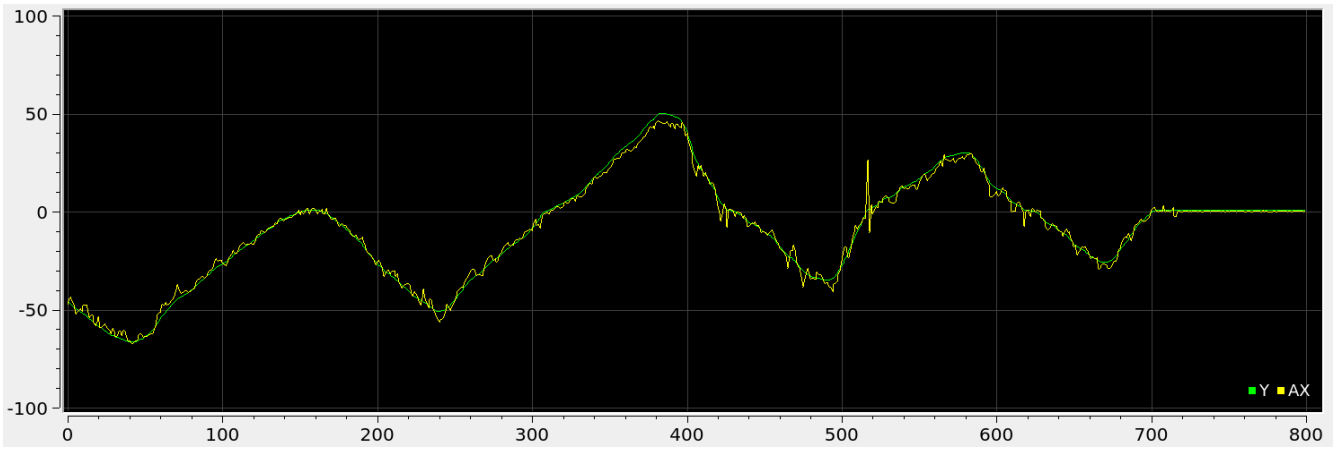
Отримання куту ψ відносно певної осі з показів акселерометра виражається як обчислення *arcsin* проекції нормованого вектору виміряного прискорення на цю вісь.

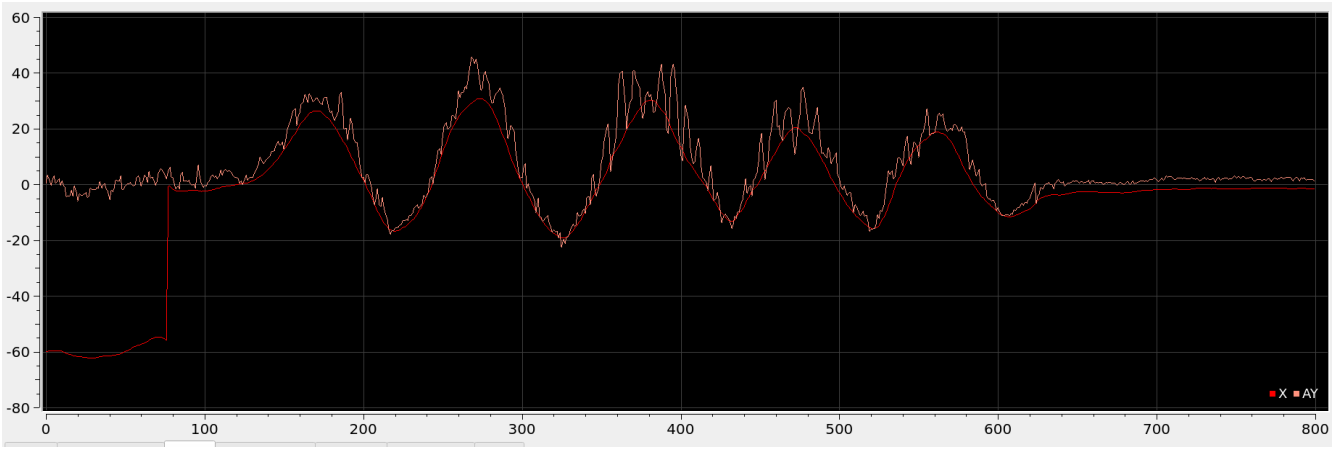
$$\psi_k = \arcsin\left(\frac{|ax_k|}{|a_k|}\right)$$

Поєднання значень φ_k та ψ_k і є метою всіх розглянутих алгоритмів.

Для отримання даних написаний клас мовою C++, який реалізує низькорівневі протоколи налаштування і комунікації з датчиком, код якого доступний на github, проте тут не розглядається.

Приклад більш точних, результатів інтегрування (зелений колір) та зашумлених даних акселерометра (жовтий колір).


100
50
0
-50
-100
0
100
200
300
400
500
600
700
800
Y
AX

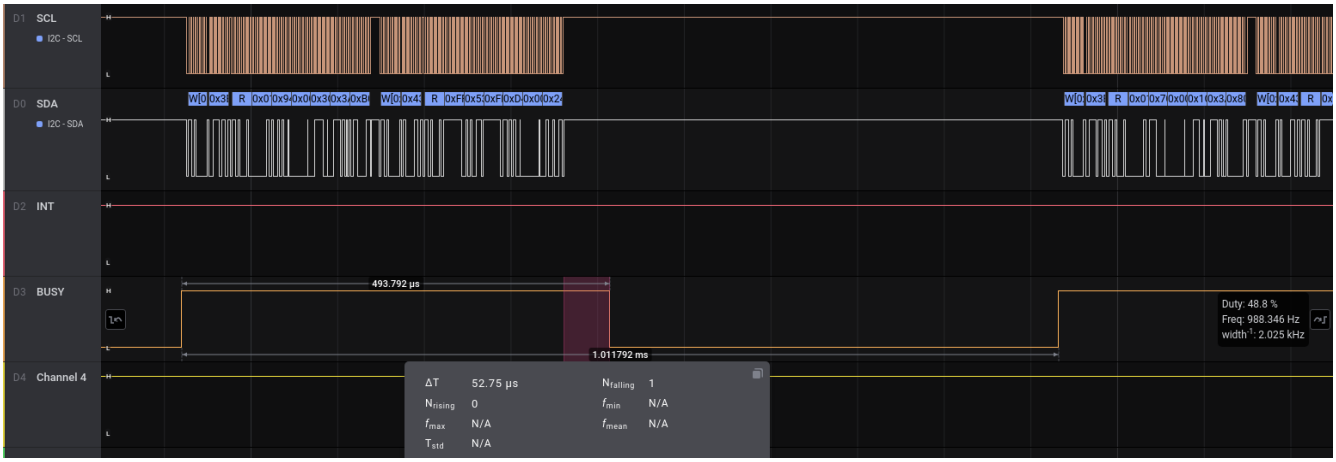

60
40
20
0
-20
-40
-60
-80
0
100
200
300
400
500
600
700
800
X
AY

Наступна функція викликається кожну мілісекунду (тобто з частотою 1 кГц) за допомогою переривання по таймеру, оновлює глобалні масиви типу float gyro_angles[3] та accel_angles[3] та викликає реалізовані функції фільтрації, які оновлюють власні глобальні масиви.

Варто зауважити, що використання глобальних змінних в подібних низькорівневих програмах майже неминуче.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);

      if (htim == &htim2) {
      mpu.get_accel(accel_data);
      mpu.get_gyro(gyro_data);

      /* atan2(accelY, accelZ) * 180/Pi; */
      accel_angles[0] = atan2f(accel_data[1], accel_data[2])/3.1415926f*180.f;

      /* atan2(-accelX, sqrt(accelY*accelY + accelZ*accelZ)) * 180/Pi; */
      accel_angles[1] = atan2f(-accel_data[0], sqrt(accel_data[1]*accel_data[1] +
                                          accel_data[2]*accel_data[2]))
                                          /3.1415926f*180.f;

      for (int i = 0; i < 3; i++) {
            /* integrate gyro, angular_speed * delta_t */
            gyro_angles[i] += gyro_data[i] * 0.001;
      }

      complementary_filter();
      kalman_filter();
      madgwick_filter();

      } else if (htim == &htim3) {

      }

      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
}
```

Для контролю того, що виконання алгоритму не займає більше, ніж 1 мс, включаючи зв’язок з датчиком, використано апаратний контроль – логічний рівень (напруга) на одному з виводів МК високий, поки відбувається виконання алгоритму.



Як видно з графіку лінії BUSY, часу достатньо. Насправді тут алгоритм виконується лише 53 мікросекунди, більшу частину займає отримання даних.

4 Аналіз розподілу похибок

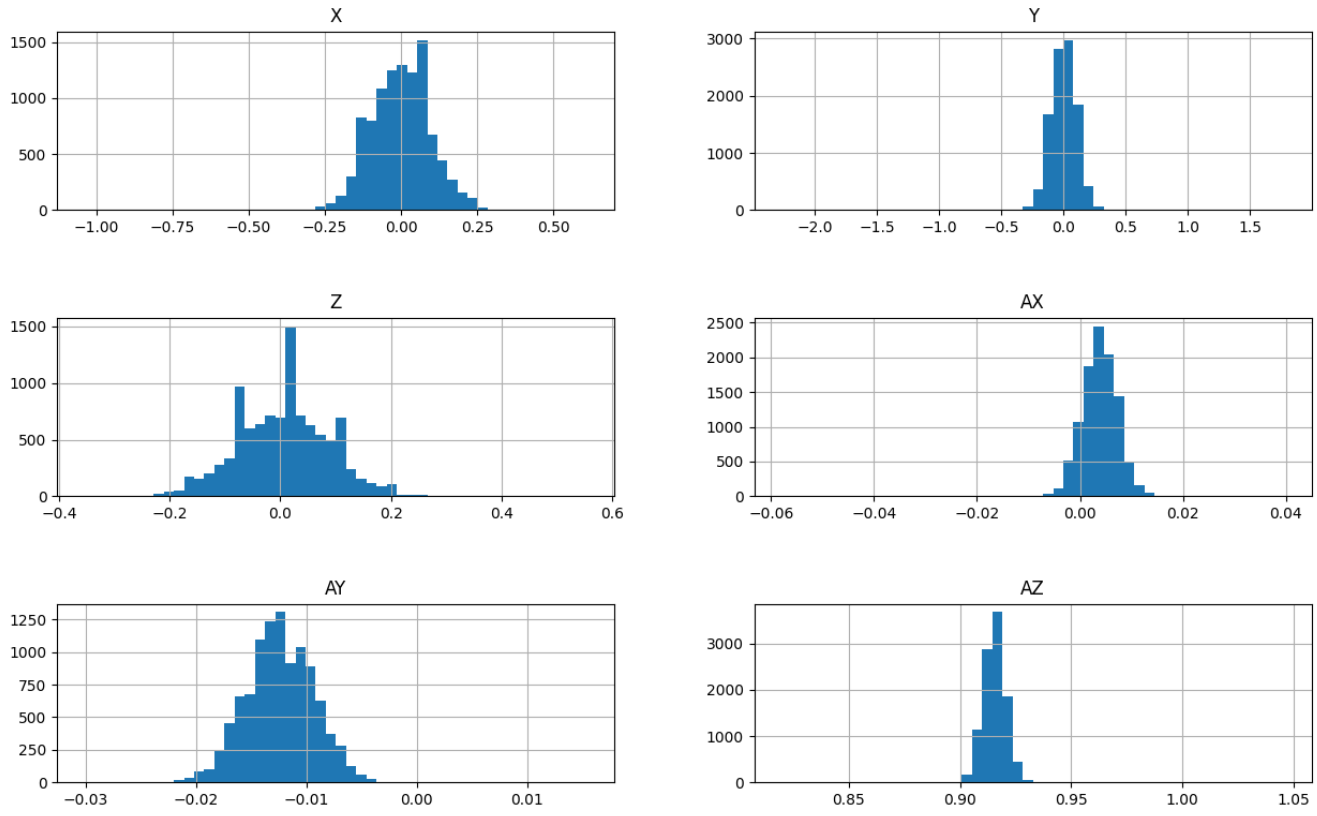
Код даного параграфу наведено у файлі zero_noize_hist.py

Зберемо достатньо велику вибірку (10253 набори) показів датчиків коли об’єкт не рухається. Для гіроскопів це буде означати нульові справжні кутові швидкості, для акселерометрів нульові прискорення за винятком прискорення вільного падіння, яке присутнє завжди.

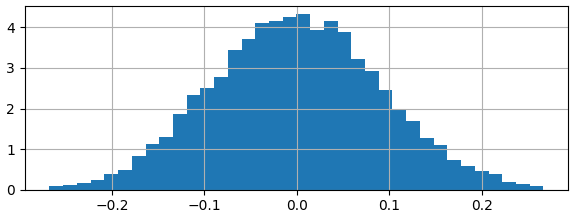
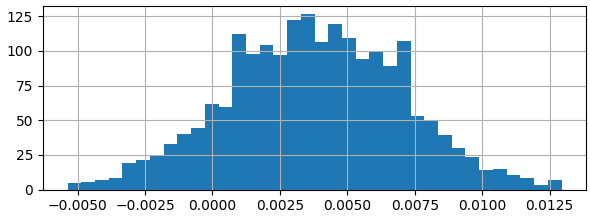
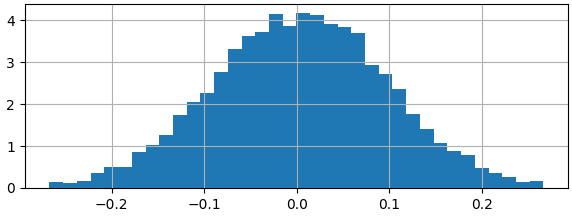
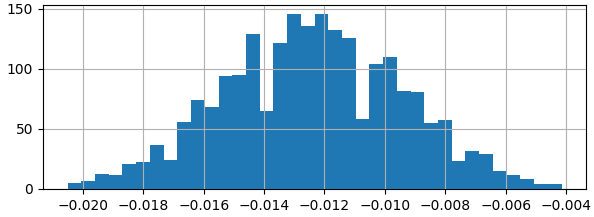
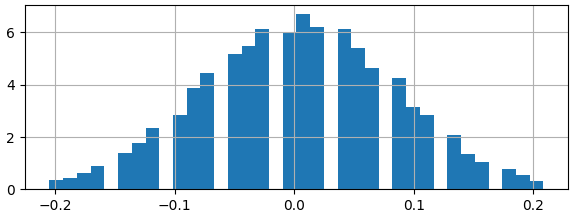
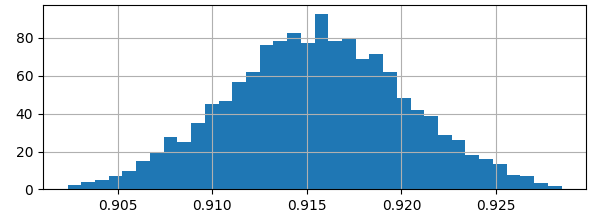
Побудуємо гістограми отриманих даних за допомогою python. Дані отримані через USB Virtual Com за допомогою програми serialplot.

На даних графіках X, Y, Z – покази гіроскопа, а AX, AY, AZ – покази акселерометра.

Для показів акселерометра одиниці – g, прискорення вільного падіння на Землі. Для показів гіроскопа – радіани в секунду.



Серед даних, насправді, наявні викиди. Вони могли утворитися від сторонніх джерел під час запису вимірів. Спробуємо позбутися їх для більшої наочності графіків. Для цього скористаємось методом 1.5 IQR. Обчислимо так званий Interquartile Range та видалимо всі дані, що менші за перший квартиль на 1.5*IQR, або більші за третій квартиль на 1.5*IQR.

З гістограм можна зробити припущення про нормальну розподіленість даних. *З дивним пропуском деяких значень, якого я не зміг позбутися за два тижні налаштування. Невідомо.*

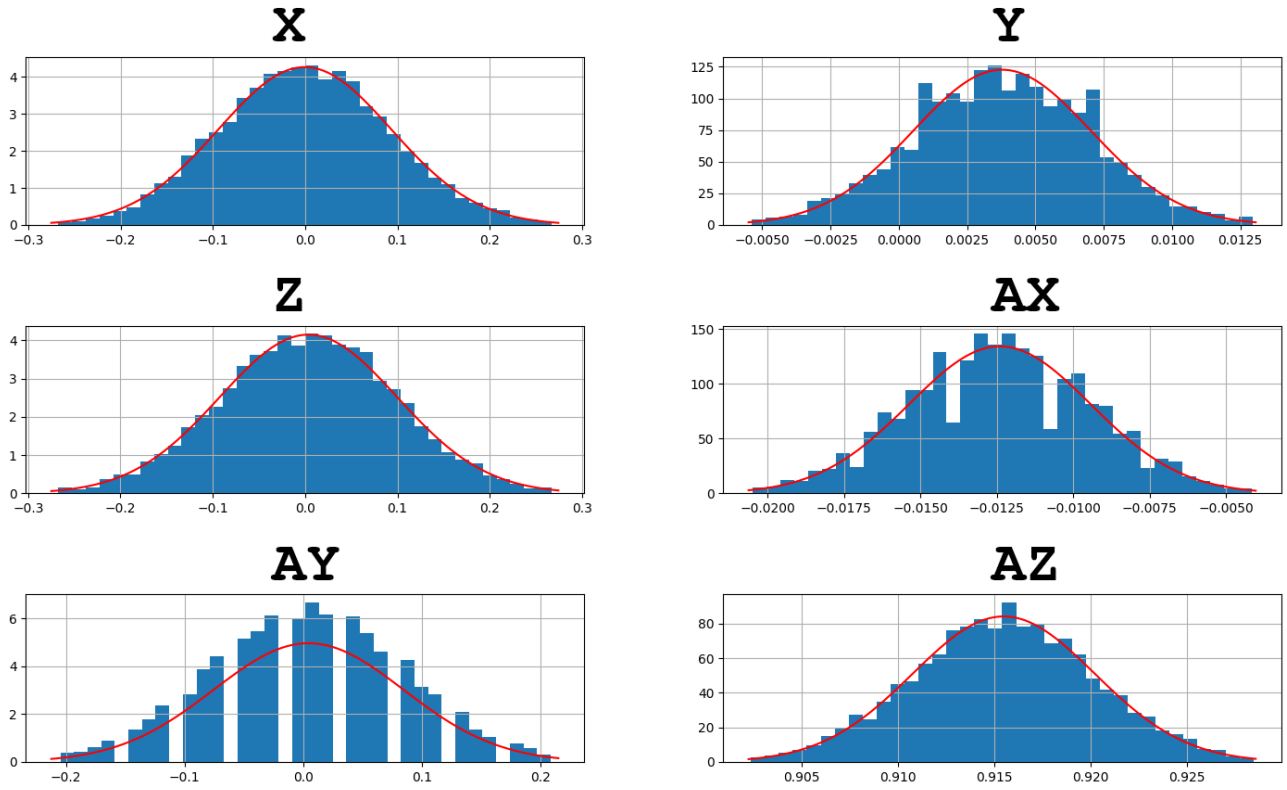
Оцінимо параметри, які б мав такий нормальний розподіл.

$$\mu = E[\overline{x}] = \frac{1}{n}\sum_{i=1}^{n} x_i = \overline{x}$$

$$\overline{\sigma}^2 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}$$

	μ	σ^2
X	-0.000331	0.008732
Y	0.003660	0.009246
Z	0.004489	0.006433
AX	0.003783	0.000011
AY	-0.012403	0.000009
AZ	0.915488	0.000022

Побудуємо графіки з отриманими параметрами зверху гістограм:



Співпадіння досить близьке, що свідчить про успішність дослідження. *Можливо, варто було б застосувати якийсь критерій, але у мене закінчується час.*

5 Комплементарний фільтр

5.1 Теорія

Ідея комплементарного фільтру (також відомого як альфа-бета фільтр), найпростішого з розглянутих, полягає у введенні деякої константи $K \in (0,1)$, яка визначає “рівень довіри” (не в сенсі математичної статистики) до результатів обох методів обчислення куту. Визначимо результат φ^* так:

$$\varphi_k^* = (K-1)\cdot\varphi_k + K\cdot\psi_k$$

Друга назва цього алгоритму – альфа-бета фільтр – виникла завдяки тому, що коефіцієнти K-1 та K часто визначають як α та β, де α = 1-β.

Чим більше ми “довіряємо” результату φ_k, тим ближчим до нуля має бути коефіцієнт K. Занадто мале значення може виявитись недостатнім для компенсації похибки, що накопичується, занадто велике призводить до залежності показів від ψ_k, підрахунок якого, як було зауважено, базується на дуже грубому припущенні.

Тож частіше за все даний коефіцієнт має підбиратися емпірично.

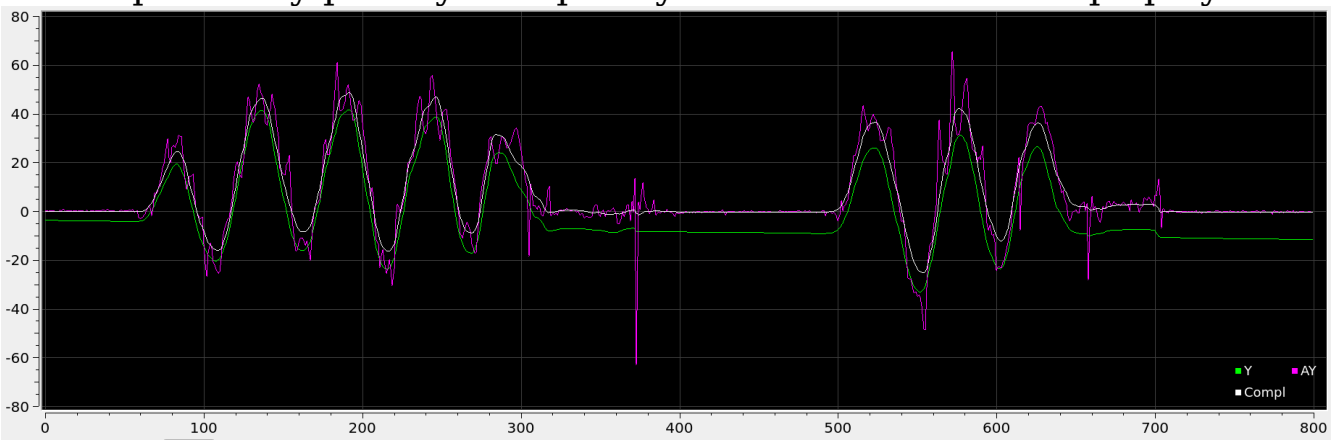
5.2 Реалізація

Вказана в параграфі 3 функція `complementary_filter()` наведена тут:

```
void complementary_filter() {
      /* Integrate while fusing accel data */
      complemenrtary_angles[0] += gyro_data[0] * 0.001;
      complemenrtary_angles[0] = (1 - complemenrtary_K) * complemenrtary_angles[0] +
                                       complemenrtary_K * accel_angles[0];

      complemenrtary_angles[1] += gyro_data[1] * 0.001;
      complemenrtary_angles[1] = (1 - complemenrtary_K) * complemenrtary_angles[1] +
                                       complemenrtary_K * accel_angles[1];

      complemenrtary_angles[2] = gyro_angles[2];
}
```

Легко помітити, що в даній реалізації обертання навколо вісі Z ніяк не компенсується. І справді, абсолютний напрям по вертикалі наявними датчиками виміряти неможливо, потрібен компас. Можливість додати його виміри є в будь-який з запропонованих алгоритмів, проте це не завжди потрібно, або реалізовано в системі сторонніми методами.

Ефективну роботу алгоритму легко побачити на графіку.



Справді, можна побачити, як вихід фільтру (біла лінія) завжди повертається в точку 0, не маючи дрейфу, наявного у гіроскопу (зелений), проте не має явного шуму, який є у даних, обчислених з напрямку прискорення.

Детальніше роботу за різних коефіцієнтів K можна побачити на відео **complementary_demo.mp4**.

5.3 Висновок

Підбором єдиного коефіцієнту можна отримати бажаний баланс між стійкістю до хибних показів акселерометра та швидкістю повернення значень кутів до істинних у випадку зашкалювання чи “дрейфу нуля” акселерометра.

Великою перевагою даного алгоритму є простота реалізації та швидкість роботи, через що його застосування може бути єдино можливим варіантом там, де обчислювальні ресурси обмежені. Так, наприклад, серед програмного забезпечення сучасних квадрокоптерів це поширений варіант.

Однак за присутності постійних та великих за модулем прискорень, відмінних від прискорення вільного падіння, або за наявності значних викидів в показаннях датчиків, оцінка ψ_k стає шкідливою, а даний алгоритм ніяк не відстежує це.

6 Фільтр Калмана

6.1 Теорія

На відміну від попереднього, фільтр Калмана передбачає створення певної **моделі руху** тіла, яке розглядається. В такому випадку розглядається похибка моделі η_k та похибка сенсора ξ_k.

Модель також називають **управляючою функцією** u_k.

Тож спостережену величину тепер варто переписати так:

$$\omega_k = \omega_{k-1}^{\check{}} + \xi_k + u_k + \eta_k$$

Далі варто ітеративно обчислити коефіцієнт фільтрації K_k, що тепер залежить від кроку.

....

Проте в загальному випадку про таку модель руху нічого невідомо, тому теорія фільтру Калмана майже незастосовна. В такому випадку рекомендується замінити ітеративне обчислення K_k деякою константою K, фактично звівши алгоритм до минулого параграфу.

6.2 Реалізація

Таким чином, реалізація вироджується до випадку комплементарного фільтру. Well...

8 Загальний висновок

В ході виконання роботи були було розглянуте прикладне застосування алгоритмів фільтрації даних, а також особливості їх реалізації в умовах обмежених обчислювальних ресурсів та вимог до часу та стабільності виконання.

Була створена модель шуму як випадкової величини, висунута теорія про характер її розподілу (нормальний розподіл) та наближено обчислені параметри такого розподілу.

Також був створерений додаток мовою Python для наочної візуалізації роботи алгоритму з тривимірним відображенням даних.