Imię i nazwisko studenta: Stanisław Rachwał
Nr albumu: 180504
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Międzywydziałowy kierunek studiów: Inżynieria biomedyczna
prowadzony przez: Wydział Fizyki Technicznej i Matematyki Stosowanej, Wydział Chemiczny, Wydział Elektroniki, Telekomunikacji i Informatyki
Specjalność/profil: Fizyka Medyczna

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Automatyzacja i optymalizacja obsługi laserowego tomografu komputerowego

Tytuł pracy w języku angielskim: Automating and optimizing of the operation of a laser computed tomography

Opiekun pracy: dr Brygida Mielewska

POLITECHNIKA GDAŃSKA | WYDZIAŁ FIZYKI TECHNICZNEJ I MATEMATYKI STOSOWANEJ

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Automatyzacja i optymalizacja obsługi laserowego tomografu komputerowego

Imię i nazwisko studenta: Stanisław Rachwał
Data i miejsce urodzenia: 02.06.2000, Gdynia
Nr albumu: 180504

Międzywydziałowy kierunek studiów: inżynieria biomedyczna

prowadzony przez: Wydział Fizyki Technicznej i Matematyki Stosowanej, Wydział Chemiczny, Wydział Elektroniki, Telekomunikacji i Informatyki

Poziom kształcenia: pierwszy
Forma studiów: stacjonarne

Typ pracy: projekt dyplomowy inżynierski

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2019 r. poz. 1231, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz. U. z 2020 r. poz. 85, z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

06.01.2023, Stanisław Rachwał

Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

**) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla.*

[1] Ustawa z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce:

Art. 312. ust. 3. W przypadku podejrzenia popełnienia przez studenta czynu, o którym mowa w art. 287 ust. 2 pkt 1–5, rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 312. ust. 4. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 5, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o podejrzeniu popełnienia przestępstwa.

STRESZCZENIE

Praca skupia się na aspektach automatyzacji diagnostyki działania laserowego tomografu komputerowego. Wraz z opisem budowy i zasady działania LCT przedstawiono w niej fizyczne aspekty działania akcelerometrów i żyroskopów, na których ma bazować opracowywany prototyp czujnika ze sterownikiem. Opisano również zagadnienie filtru Kalmana oraz zasadność jego wykorzystania w proponowanym rozwiązaniu. Następnie przedstawiono założenie i działanie platformy Arduino oraz modułu SEN0142, a następnie opisano tryb komunikacji pomiędzy poszczególnymi elementami opracowywanego urządzenia, tj.: komputerem z systemem *Microsoft Windows*, Arduino, a modułem z akcelerometrem i żyroskopem.

W dalszej części pracy przedstawiona została stworzona aplikacja, jej wygląd, funkcjonalności i ograniczenia, wraz z kluczowymi fragmentami kodu. Przytoczony został również skrypt samego sterownika (Arduino), który implementuje uprzednio opisane wzory.

Następnie w pracy przedstawiono pomiary wykonane czujnikiem dla różnych położeń, na podstawie których możliwe jest określenie niepewności pomiarów. Wykazano również, że opracowane rozwiązanie nie sprawdza się w przypadku badań dynamicznych. Fakt ten zostaje poddany dalszej analizie.

W podsumowaniu przedstawiono przykładowe propozycje rozwiązań mające na celu zarówno ulepszenie, jak i naprawę działania urządzenia.

Słowa kluczowe: wzorcowanie, akcelerometr, kąty RPY

Dziedzina nauki i techniki, zgodnie z wymogami OECD: nauki inżynieryjne i techniczne, inżynieria biomedyczna

ABSTRACT

The project focuses on aspects of automating the diagnostic performance of a laser-based tomograph. After a description of the LCT, the physical aspects of accelerometers and gyroscopes on which the prototype sensor with controller is based are presented. The Kalman filter is also addressed, and why its use is needed. Next, the premise and operation of the Arduino platform and the SEN0142 module are introduced. The next section presents the appearance of the sensor, and the communication between the various components of the device under development is also described, i.e.: the computer with the *Microsoft Widnows* system, Arduino, and the module with the accelerometer and gyroscope.

The next chapter is devoted to the created application, its appearance, functionalities and limitations, along with key code snippets. The script of the controller itself (Arduino), which implements the previously described designs, is also cited.

The paper then presents measurements made with the sensor for various positions, from which it is possible to determine the uncertainty of the measurements. It is also shown that the developed solution does not work for dynamic tests. This fact is further analyzed.

In conclusion, sample ideas for both improving and repairing the device's performance are presented.

Keywords: calibration, accelerometer, RPY angles

SPIS TREŚCI

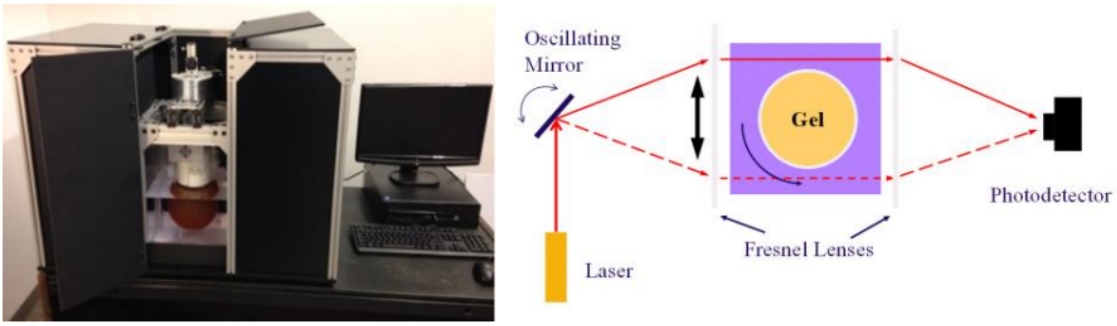
WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

LCT - laserowa tomografia komputerowa (laser computed tomography),
MEMS - mikroukład elektromechaniczny,
ADC - przetwornik analogowo-cyfrowy (analog to digital converter),
RPY - roll ϕ, pitch θ, yaw ψ: kąty eulerowskie względem osi (OY, OX, OZ)

1. WSTĘP I CEL PRACY

Laserowy tomograf komputerowy to innowacyjne urządzenie opracowywane i udoskonalane w ramach szeregu projektów naukowych poświęconych dozymetrii wielowymiarowej wysokiej rozdzielczości na Wydziale Fizyki Technicznej i Matematyki Stosowanej PG. W [1] przedstawiono w postaci schematu blokowego proces pozyskiwania oraz przetwarzania obrazów w celach radioterapeutycznych. Dodanie do niego symulacji LCT jeszcze bardziej rozbuduje istniejący algorytm postępowania, ale umożliwi porównanie rozkładu dawki w przestrzeni trójwymiarowej otrzymanej w ramach symulacji z teoretycznymi oczekiwaniami wyliczonymi przez odpowiedni algorytm (bazujące na wokselach, jeżeli do diagnostyki zastosowano metodę SPECT/CT, to metody niesztywne rejestracji obrazów przyczyniają się do poprawy rozdzielczości [2]). Docelowo rozwiązanie to ma zostać wykorzystane m.in.: przez oddziały radiologiczne w szpitalach w celach dozymetrycznych, konkretnie do symulacji przygotowanych przez fizyków medycznych planów leczenia nowotworów.

Do najważniejszych elementów LCT można zaliczyć: laser, zwierciadło zamontowane na silniku krokowym (lub też silniku rezonansowym), akwarium (w którym umieszcza się fantom polimerowo-żelowy) i detektor promieniowania optycznego. Sama wiązka po odbiciu od zwierciadła jest modyfikowana specjalnymi soczewkami. Wykorzystywany laserowy tomograf komputerowy został bardziej szczegółowo opisany w [3].

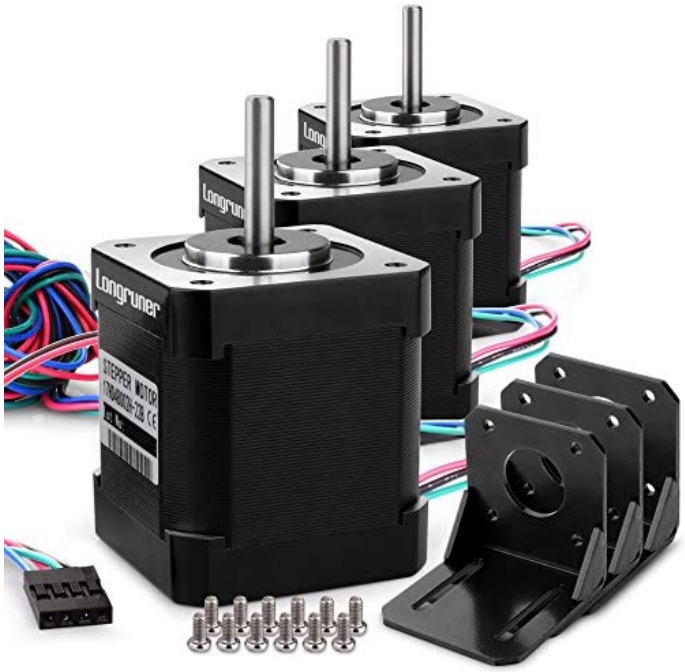


Rys. 1.1. Wygląd oraz schemat tomografu [3]

Omawiane urządzenie ma przyczynić się do wzrostu bezpieczeństwa i skuteczności radioterapii, jak również przyspieszyć i zoptymalizować pracę samego fizyka medycznego. Z tego powodu, pożądana jest możliwie jak największa automatyzacja działania tomografu. Jednym z aspektów do usamodzielnienia jest diagnostyka działania LCT. Opracowany aparat jest prototypem zbudowanym ręcznie. Trzeba więc w analizie błędów uwzględnić potencjalny czynnik ludzki.

Przy leczeniu wykorzystującym teleradioterapię, niezwykle istotna jest dokładność, jak i precyzja, z jaką deponowana jest dawka promieniowania w guzie nowotworowym oraz jego otoczeniu. W miarę możliwości należy unikać niepotrzebnego narażania tkanek zdrowych oraz narządów krytycznych na uszkodzenia wywołane promieniowaniem. Dlatego też przy symulacjach należy zadbać o możliwie najwierniejsze odzwierciedlenie docelowej operacji, niwelując przy tym błędy mogące wynikać z samego działania urządzenia. W tym celu potrzebna jest dodatkowa aparatura, która umożliwi szybką i sprawną analizę działania samego LCT.

Szczególną uwagę trzeba poświęcić silnikowi krokowemu i przymocowanemu do niego zwierciadełku. Kształt zwierciadła (kołowy) pozwala na łatwiejsze zachowanie symetrii. Jednakże, przez potencjalne nieidealne umieszczenie zwierciadła na elemencie obrotowym mogą powstać asymetryczne momenty bezwładności, które będą bezpośrednio wpływać na prędkość obrotową samego elementu (na charakterystyce zmiany położenia objawiłoby się to w postaci niesymetrycznej sinusoidy). Przykładowy silnik krokowy z Amazona przedstawia rysunek 1.2. Kątownik jest przymocowany do silnika, po jednej stronie ma przymocowane zwierciadło, a po drugiej moduł SEN0142 (rozdział 3.3).



Rys. 1.2. Przykładowy silnik krokowy

Celem pracy jest zaprojektowanie i zbudowanie czujnika wraz z oprogramowaniem, umożliwiającego zautomatyzowany pomiar zmian położenia elementu obrotowego silnika krokowego. Pozwoli to na zaobserwowanie potencjalnych nieprawidłowości w zmianach kątowych poruszającego się zwierciadła, dzięki czemu możliwa będzie szybka reakcja i korekta elementu. Praca ma skupić się na:

1. analizie nowych rozwiązań zagadnienia,
2. zaimplementowaniu wybranego rozwiązania,
3. testowaniu i udoskonaleniu stworzonego programu.

2. ROZWIĄZANIA DOSTĘPNE NA RYNKU

2.1. Laserowe inklinometry cyfrowe

Podstawowe cechy:

- Cena: ≈ 200 zł,
- niepewność pomiaru: $\pm 0.2^o$,
- wysoka precyzja,
- bezprzewodowe,
- przenośne,
- zazwyczaj jednoosiowe.

Ładowane za pomocą kabla USB (zarówno połączonego do komputera, jak i bezpośrednio do sieci elektrycznej. Mają wbudowaną pamięć, pozwalającą na zapis pomiarów (te mogą być bezwzględne, jak i względne).

Ich wykorzystanie wprowadziłoby ryzyko błędów symulacji LCT wywołanych wiązką laserową z inklinometru.

Przykładowe rozwiązania:

- Laserowy kątomierz cyfrowy - inklinometr

2.2. Inklinometry analogowe

Podstawowe cechy:

- Cena: ≈ 2000 zł,
- niepewność pomiaru: $\pm 0.14^o$,
- zakres poprawnej temperatury: $-40 - 70^o$C
- częstotliwość próbkowania: 500 S/s
- wysoka precyzja,
- przewodowe,
- najczęściej jednoosiowe lub dwuosiowe.

Ilość osi, na których czujnik pracuje, jak i precyzja pomiaru są programowalne (drugi parametr jest dodatkowo zależny od zakresu mierzonego kąta). Posiadają również wbudowane filtry antywibracyjne.

Przykładowe rozwiązania:

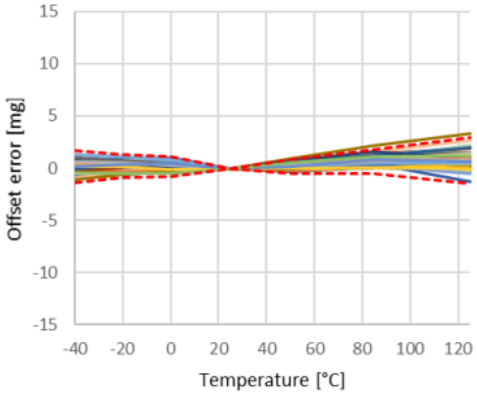
- Inklinometr analogowy seria IS40, 8.IS40.14121
- CANopen Inklinometr seria IN360TC

2.3. *Inklinometry cyfrowe*

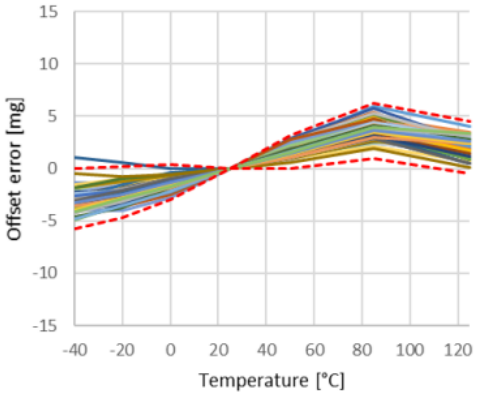
Podstawowe cechy:

- Cena: ≈ 250 zł,
- niepewność pomiaru: $\pm 1.15^o$
- zakres poprawnej temperatury: $-20 - 125^o$C,
- przewodowe,
- mogą być 3-osiowe,
- wrażliwe na wilgoć,
- wbudowany interfejs SPI.

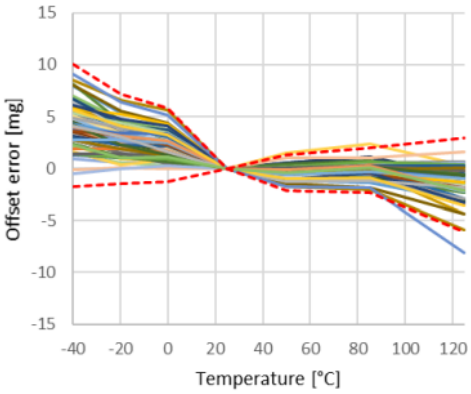
Są podatne na działanie temperatury - odchylenie jest zaniedbywalne dla temperatury wynoszącej 23^oC [4]. Charakterystyki odchyleń w funkcji temperatury prezentuje rys. 2.1.



(a) Oś OX



(b) Oś OY



(c) Oś OZ

Rys. 2.1. Odchylenia akcelerometru w funkcji temperatury [4]

Pracuje on na napięciach z zakresu $3.3 - 3.6$V, możliwe jest więc włączenie go do układu razem z Arduino/Raspberry Pi. Interfejs SPI jest dodatkowym ułatwieniem w komunikacji pomiędzy komponentami.

Przykładowe rozwiązanie:

- MuRata SCL3300-D01-10

3. ANALIZA ZAGADNIENIA

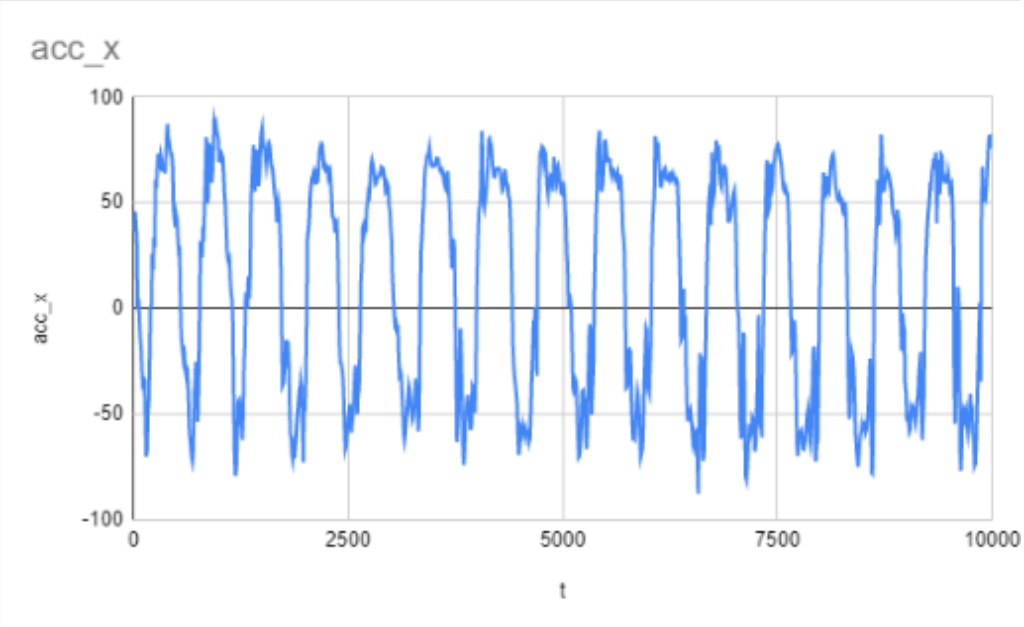
3.1. Fizyczne zasady działania akcelerometru i żyroskopu

3.1.1. Akcelerometr

Zadaniem tego podzespołu jest pomiar i analiza przyspieszenia liniowego/kątowego. Wewnątrz urządzenia znajduje się czujnik przyspieszenia, który bada siłę wynikającą ze zmian ruchu. Dzięki dużej wrażliwości na zmianę położenia jest on w stanie określać z dużą precyzją i dokładnością ruch własny, jak i całego akcelerometru. Najczęściej wykorzystuje się 3-osiowe podzespoły (wykonujące pomiar względem osi OX, OY i OZ). Możemy podzielić je na:

- piezoelektryczne - mają duże pasmo pomiarowe. Mogą mierzyć drgania o wysokich częstotliwościach i amplitudach.
- piezorezystancyjne - wykorzystywane do pomiarów wibracji. Stosowane powszechnie w przemyśle.
- pojemnościowe MEMS - najmniejsze i najtańsze. Zostały zastosowane w opracowywanym prototypie.

W przypadku akcelerometrów MEMS pomiar polega na badaniu zmian pojemności pomiędzy ruchomą masą a nieruchomymi okładkami kondensatora. Podczas ruchu z przyspieszeniem zmienia się pojemność, którą następnie można przekonwertować na potrzebną wartość. Wadą tych elementów (korygowaną stopniowo przez producenta) jest stosunkowo mała czułość i podatność na szumy. Rysunek 3.1 przedstawia przykładową charakterystykę niezamocowanego modułu wprawionego w ruch symulujący ten z tomografu.



Rys. 3.1. Przykładowa charakterystyka z akcelerometru A [ref] = f(t [ms])

Dla wartości względem osi OX i OY przesyłanych przez urządzenie stosuje się przekształcenia, umożliwiające otrzymanie odpowiednio: kąta pochylenia (ang. pitch) θ, jak i przechylania (ang. roll) ϕ :

$$\theta = \arctan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right) \tag{3.1}$$

$$\phi = \arctan\left(\frac{A_y}{\sqrt{A_x^2 + A_z^2}}\right) \tag{3.2}$$

Powyższe wzory wynikają bezpośrednio z twierdzenia Pitagorasa zastosowanego w przestrzeni trójwymiarowej (A - wektor przyspieszenia).

3.1.2. Żyroskop

Żyroskopy to urządzenia służące do analizy, jak i utrzymywania orientacji przestrzennej. Można je podzielić na kierunkowe, prędkościowe i elektroniczne. Służą do mierzenia prędkości obrotowej obiektu. Drgania elementu zostają wymuszone dzięki efektowi piezoelektrycznemu, natomiast pomiar wykorzystuje siłę Coriolisa (złożenie dwóch ruchów pewnej masy, z czego jeden jest ruchem obrotowym) [5].

By otrzymać zmianę kąta, otrzymane dane należy scałkować po czasie, co prezentują poniższe równania [5]:

$$\theta(t_i) = \theta(0) + \sum_{j=1}^{i} \omega_X(t_j)(t_j - t_{j-1}) \tag{3.3}$$

$$\phi(t_i) = \phi(0) + \sum_{j=1}^{i} \omega_Y(t_j)(t_j - t_{j-1}), \tag{3.4}$$

$$\psi(t_i) = \psi(0) + \sum_{j=1}^{i} \omega_Z(t_j)(t_j - t_{j-1}), \tag{3.5}$$

gdzie: θ - kąt przechylania (pitch), ϕ - kąt pochylania (roll), ψ - kąt skręcenia (yaw), ω - prędkość obrotowa i jej odpowiednie składowe względem danej osi, t - czas.

3.1.3. Filtr Kalmana

Otrzymywane pomiary podatne są na zakłócenia: akcelerometr jest wyczulony na drgania otoczenia, natomiast żyroskop jest podatny na zjawisko dryfu. Dodatkowo, ze względu na dane z dwóch czujników, należy dokonać ich złożenia. Do tego celu stosuje się między innymi filtr Kalmana, który dokonuje iteracyjnej estymacji procesu w układzie ze sprzężeniem zwrotnym [5]. Możemy wyznaczyć dwie fazy działania filtru: fazę predykcyjną i fazę korekcji.

W praktyce zostanie wykorzystany cyfrowy filtr Kalmana realizujący algorytm rekurencyjny. Bazując na [6], model estymacji możemy przedstawić jako: liniowe równanie stanu i liniowe równanie obserwacji (odpowiednio (3.6) i (3.7)).

$$x(t+1) = A(t)x(t) + B(t)v(t) \tag{3.6}$$

$$y(t) = C(t)x(t) + w(t) \tag{3.7}$$

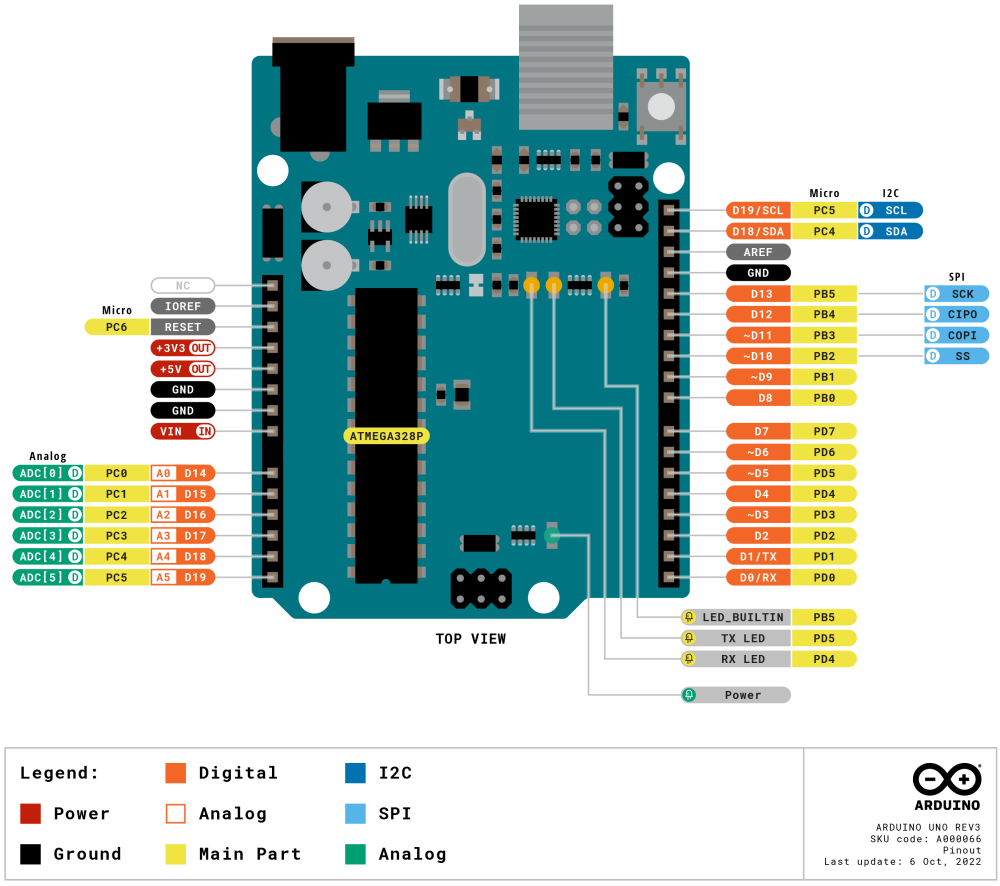
gdzie: t - wartości dyskretne na osi czasu, $x(t)$ - wektor stanu, $y(t)$ - wektor obserwacji, $v(t)$ - biały szum procesowy o zerowej wartości średniej, $w(t)$ - biały szum obserwacji o zerowej wartości średniej, $A(t)$ - macierz systemowa, $B(t)$ - macierz wejścia, $C(t)$ - macierz wyjścia.

Następnie w fazie korekcji algorytm wykorzystuje pomiary do aktualizacji i poprawy estymacji. W publikacji [6] została przedstawiona dalsza metodologia działania tych filtrów.

3.2. *Płytka Arduino Uno*

Arduino to projekt rozpoczęty w roku 2005, opierający się na licencji typu *open hardware*. Przeznaczony on jest dla mikrokontrolerów zamontowanych na obwodzie drukowanym z wbudowanymi układami wejścia/wyjścia (I/O) oraz standaryzowanym językiem programowania. Ostatni bazuje na C++, a wykorzystując wtyczkę PlatformIO możliwe jest kompletne posługiwanie się tym językiem.

W projekcie zostanie wykorzystany model Arduino Uno. Został on wybrany na podstawie dostępnych funkcjonalności, przy akceptowalnej cenie produktu. Opracowywane oprogramowanie nie będzie zapisywać danych na płytce, lecz będą one przesyłane i przechowywane w aplikacji okienkowej.



Rys. 3.2. Schemat Arduino Uno [7]

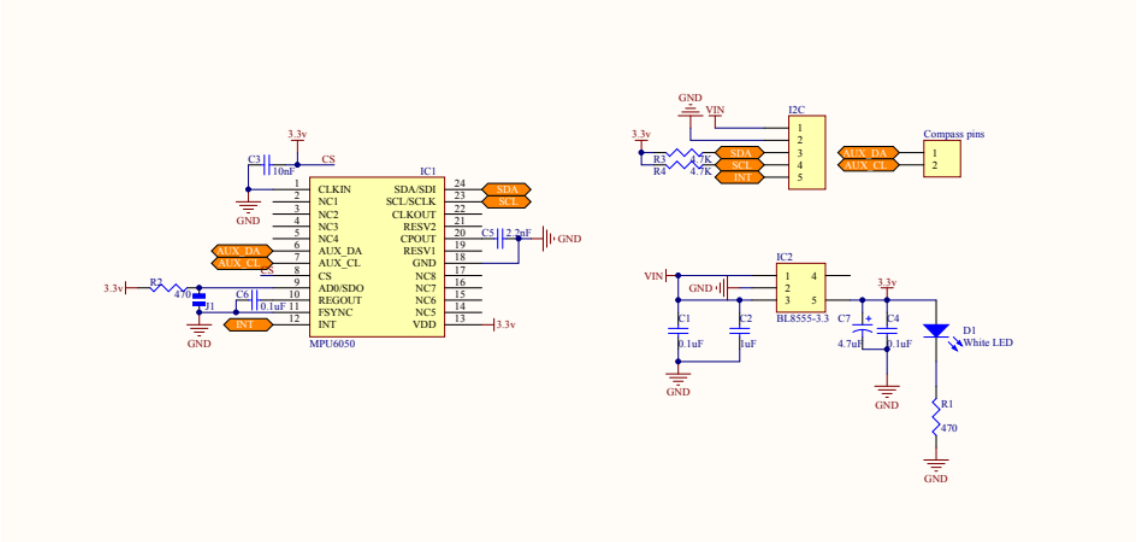
Sama płytka składa się z 8-bitowego mikrokontrolera Atmel AVR (konkretnie ATmega328P), ma wbudowane 32KB pamięci flash, 2KB SRAM, 1KB EEPROM. Posiada interfejsy: I2C, SPI, UART i USB (jest programowana przez adapter *USB to serial*). Działa na napięciach wejścia 7-12V oraz wartościach napięć operacyjnych 3.3V i 5V. Płytka posiada 20 pinów cyfrowych oraz 6 analogowych i typu PWM. Rys. 3.2 przedstawia umiejscowienie powyższych funkcji na płytce drukowanej.

Oficjalnym środowiskiem do programowania omawianych płytek jest Arduino IDE. Aczkolwiek, ze względu na stosunkowo ograniczone możliwości w pracy zostanie wykorzystane JetBrains CLion wraz z wtyczką PlatformIO. Oprogramowanie to posiada znacznie większe możliwości, jako że producent specjalizuje się w narzędziach programistycznych, zarówno na licencjach płatnych, jak i tych typu *open source*.

3.3. Moduł DFRobot SEN0142 (MPU-6050) - dodatek do Arduino

MPU-6050 opracowane przez firmę InvenSense są powszechnie używane w modułach określających położenie w przestrzeni, jak i prędkością obrotową. Charakteryzują się one małym poborem mocy oraz wysoką wydajnością. Podzespoły te zawierają w sobie dwa analogowe czujniki: akcelerometr i żyroskop. Oba pozwalają na pomiar zmiany kąta, jednak na odmiennej zasadzie. Sam moduł posiada wbudowany cyfrowy procesor ruchu, jak i 16-bitowy przetwornik analogowo-cyfrowy (ADC). Wykorzystując magistralę I2C wraz z wbudowanymi rezystorami, możliwe jest (poprzez komunikację szeregową) uzyskanie zestawów danych przy bezpośrednim połączeniu układu z Arduino.

Na rynku jest dostępnych wiele podzespołów z wbudowanym układem MPU-6050. Moduł od firmy DFRobot został wytypowany ze względu na posiadanie funkcji wystarczających do spełnienia założeń doświadczenia przy akceptowalnej cenie. Dodatkowym atutem jest rozmiar czujnika, który całkowicie mieści się na elemencie obrotowym silnika krokowego. Rys. 3.3 prezentuje zespolenie MPU-6050 z całym modułem SEN0142.



Rys. 3.3. Schemat modułu DFRobot SEN0142 [8]

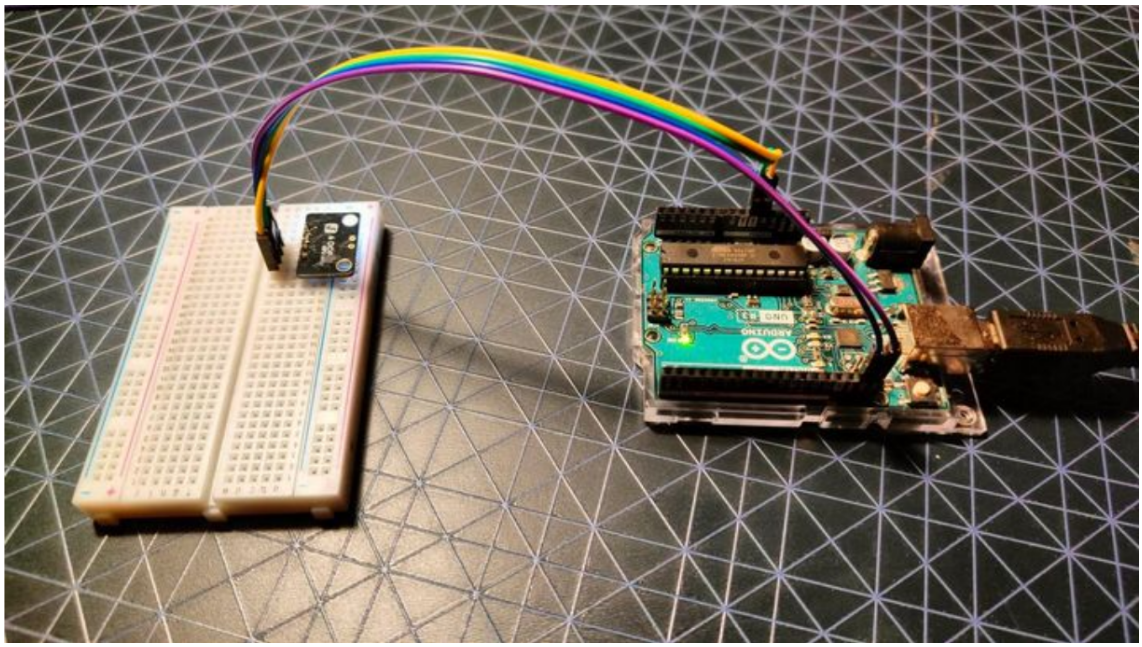
Specyfikacja na podstawie [9]

1. **Akcelerometr**
   - Trzyosiowy, z cyfrowym wyjściem, programowalny w zakresie przeciążeń: $\pm 2\ g$, $\pm 4\ g$, $\pm 8\ g$ i $\pm 16\ g$
   - Posiada zintegrowane, 16-bitowe przetworniki ADC, pozwalające na próbkowanie sygnałów bez stosowania multiplekserów
   - Natężenie prądu roboczego: $500 \mu A$
   - Natężenie prądu przy niskim poborze mocy: $10\ \mu A$ dla $1.25\ Hz$, $20\ \mu A$ dla $5\ Hz$, $60\ \mu A$ dla $20\ Hz$, $110\ \mu A$ dla $40\ Hz$
2. **Żyroskop**
   - Trzyosiowy, z cyfrowym wyjściem, programowalny w przedziale prędkości kątowych: ± 250, ± 500, ± 1000 i $\pm 2000^o/s$
   - Wbudowane 16-bitowe ADC
   - Poprawiona wydajność (względem poprzednich modeli) szumów przy niskich częstotliwościach
   - Programowalny cyfrowy filtr dolnoprzepustowy
   - Natężenie prądu roboczego: $3.6\ mA$
   - Natężenie prądu spoczynkowego: $5\ \mu A$

3.4. Budowa i założenia działania czujnika

3.4.1. Budowa

Czujnik składa się z dwóch elementów: płytki Arduino oraz modułu DFRobot SEN0142 połączonych czterema przewodami, jak pokazano na Rys. 3.4.



Rys. 3.4. Pierwszy prototyp urządzenia

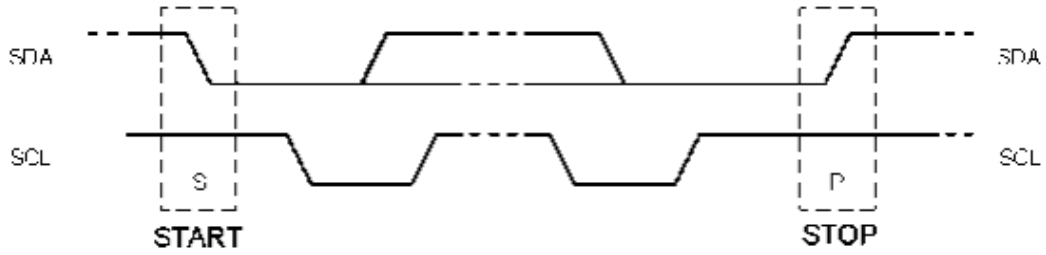
Moduł SEN0142 zasilany jest stałym napięciem 5V. SDA i SCL są pinami magistrali I2C odpowiadającymi odpowiednio za: linię danych (serial data) oraz linię zegara taktującego tranmisję (serial clock).

3.4.2. Magistrala I2C i port szeregowy

Komunikacja pomiędzy Arduino a modułem SEN0142 przebiega poprzez szynę I2C. Jak przedstawiono w [10], obie linie magistrali są dwukierunkowe, rezystor podciągający podłącza je do dodatniego napięcia. Gdy nie ma transmisji, to zarówno na SDA i SCL znajduje się stan wysoki. Standardowo dane są przesyłane z prędkością 100 kb/s, lub 400 kb/s.

Każdy układ podłączony do magistrali ma przypisany adres i może być zarówno odbiornikiem jak i nadajnikiem. Dzielimy je na tzw.: master i slave. Pierwszy inicjuje transmisję i generuje sygnał zegarowy. Drugim natomiast jest każdy zaadresowany układ. Rola mistrza najczęściej jest przypisana mikrokontrolerom, podobnie w przypadku tego projektu: płytce Arduino.

Napięcia poziomów logicznych nie są zdefiniowane, ale zależą od napięcia zasilającego V_{dd}. Jeden impuls zegarowy przypada na każdy przesłany bit. Dla wysokiego poziomu zegara wymagana jest stabilność danych na lilii SDA. Stan może się zmieniać dla niskiego zegara.



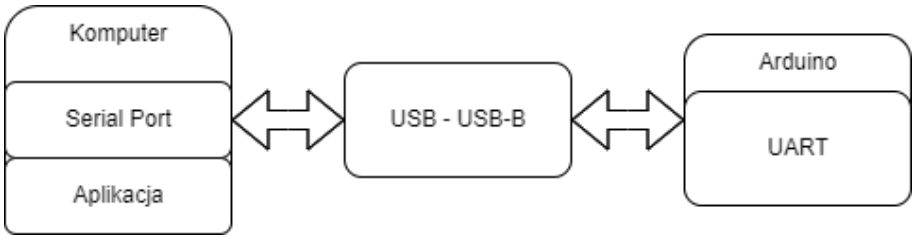
Rys. 3.5. Bity start i stop [10]

Gdy na SCL panuje stan 1 a na SDA następuje zmiana:

- $1 \rightarrow 0$, następuje bit START,
- $0 \rightarrow 1$, następuje bit STOP.

Obie sytuacje są generowane przez układ master. Po bicie START szyna jest zajęta, a po STOP się zwalnia.

Komunikacja pomiędzy Arduino a komputerem stacjonarnym odbywa się poprzez UART, kabel USB i port szeregowy.



Rys. 3.6. Schemat komunikacji

Port szeregowy jest typem portu komputerowego, gdzie przekazane dane mają formę ciągu bitów. Jest zdolny do tłumaczenia bitów na bajty i odwrotnie. Jako, że komputer posiada kilka takich portów, należy upewnić się, który z nich przypada na Arduino. Otwierając komunikację istnieje również opcja między innymi: wyboru prędkości połączenia.

4. ANALIZA OPRACOWANEGO OPROGRAMOWANIA

Opracowane oprogramowanie działa na zasadzie akcja-reakcja. W zależności od nastaw w aplikacji, ta wysyła odpowiedni ciąg znakowy do Arduino, które na tej podstawie wykonuje odpowiedni pomiar. Otrzymywane dane są na bieżąco przesyłane z powrotem do aplikacji, gdzie ta wyświetla je w oknie tekstowym. Po otrzymaniu wszystkich wyników, program może:

- przekonwertować dane na odpowiedni format, umożliwiając tym samym wizualizację otrzymanych danych,
- zapisać dane do pliku tekstowego w przygotowanym folderze, nadając mu nazwę złożoną z daty i godziny pomiaru.

Przesyłanie danych pomiarowych bezpośrednio do komputera było konieczne ze względu na małą pamięć płytki Arduino. Wiąże się to jednak z pewnym utrudnieniem, wynikającym bezpośrednio z działania portu szeregowego. Mianowicie, komenda przesyłania danych `Serial.print()` jest powolna. Z tego powodu, pomimo działania zegara magistrali I2C nawet na częstotliwości 400 kHz, w rzeczywistości osiągalna jest częstotliwość jedynie 100 Hz. Powinno jednak być to wystarczające do analizy potencjalnych różnic w momentach bezwładności poruszającego się zwierciadła, jak i potencjalnych odchyleń od położenia referencyjnego.

4.1. Skrypt do Arduino

Każdy skrypt Arduino do poprawnego działania wymaga dwóch funkcji: `void setup()` oraz `void loop()`. Pierwsza służy do konfiguracji parametrów, wykonywana jest tylko przy uruchomieniu płytki, druga natomiast jest nieskończoną pętlą wykonującą się tak długo, dopóki Arduino jest podłączone do zasilania (czy to przez USB-B, czy bezpośrednio do źródła (np. bateria 9V).

Przy definicjach funkcji został zastosowany dodatkowy człon `__attribute__((unused))`. Jest on jedynie informacją dla środowiska, by to nie wyświetlało błędu o niewykorzystaniu owych funkcji (nie są one w żadnym momencie wywoływane). Omawiane funkcje prezentują się następująco:

```
__attribute__((unused)) void loop() {
    String value = Serial.readString();

    if (value == "0") {
        filtered_angles(10000, 10);
    }
    else if (value == "1") {
        get_gyr(10000, 10);
    }
    else if (value == "9") {
        init_mpu();
    }
}
```

Listing 4.1: Funkcja loop()

```
__attribute__((unused)) void setup() {
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000L);
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
#endif

    Serial.begin(115200);
}
```

Listing 4.2: Funkcja setup()

Listing 4.1 prezentuje ciało kluczowej funkcji - właśnie ona rozporządza wywoływaniem odpowiednich funkcji w zależności od odczytanego znaku. Wartość 10 odpowiada co ile milisekund jest próbkowany sygnał z czujnika, natomiast 10000 to czas pełnego pomiaru, również w milisekundach.

Drugi listing natomiast prezentuje ustawienie parametru częstotliwości zegara, na którym będzie operować szyna I2C (400 kHz). Zostaje również ustawiona prędkość portu szeregowego.

Prócz opisanych funkcji, skrypt posiada jeszcze trzy dodatkowe, kluczowe do wykonania i obróbki pomiarów z postaci surowej do wartości docelowych.

Funkcja badająca położenie czujnika (w stopniach, względem wektora siły grawitacyjnej) opiera się na złożeniu pomiarów z akceleratora i żyroskopu przy pomocy filtra Kalmana (zaimplementowany z biblioteki).

```
void filtered_angles(int time, int t) {
    KalmanFilter kalmanX(0.001, 0.003, 0.03);
    KalmanFilter kalmanY(0.001, 0.003, 0.03);

    Serial.println("Czas \t Pitch \t Roll \t (K)Pitch \t (K)Roll");

    for (long i = 0; i <= time; i += t) {
        Vector acc = mpu.readNormalizeAccel();
        Vector gyr = mpu.readNormalizeGyro();

        timer = millis();

        double acc_pitch = -(atan2(acc.XAxis, sqrt(square(acc.YAxis) + square(acc.
    ZAxis))) * 180.0) / M_PI;
        double acc_roll = (atan2(acc.YAxis, sqrt(square(acc.XAxis) + square(acc.
    ZAxis))) * 180.0) / M_PI;

        double kal_pitch = kalmanY.update(acc_pitch, gyr.YAxis);
        double kal_roll = kalmanX.update(acc_roll, gyr.XAxis);

        Serial.print(i); Serial.print("\t");
        Serial.print(acc_pitch); Serial.print("\t"); Serial.print(acc_roll); Serial
    .print("\t");
        Serial.print(kal_pitch); Serial.print("\t"); Serial.println(kal_roll);

        blink = !blink;
        digitalWrite(LED_BUILTIN, blink);
```

Listing 4.3: Funkcja filtered_angles()

Definicje filtrów muszą być umieszczone poza pętlą, w przeciwnym wypadku byłyby one z każdą kolejną iteracją resetowane, w efekcie czego wartości byłyby złożeniem zmiany kątów, a nie położenia. Otrzymane dane z akcelerometru zostają przeliczone na kąty za pomocą wzorów (3.5) i (3.6). Wyniki po fuzji sensorycznej IMU zostają przesłane na komputer.

```
    }
}

void get_gyr(int time, int t_s) {
    Serial.println("t [ms] \t pitch \t roll \t yaw");

    double pitch = 0;
    double roll = 0;
    double yaw = 0;

    for (int i = 0; i <= time; i+= t_s) {
        Vector gyr = mpu.readNormalizeGyro();

        timer = millis();

        pitch += gyr.YAxis * t_s / 1000;
        roll += gyr.XAxis * t_s / 1000;
        yaw += gyr.ZAxis * t_s / 1000;

        show(i, pitch, roll, yaw);

        blink = !blink;
        digitalWrite(LED_BUILTIN, blink);
```

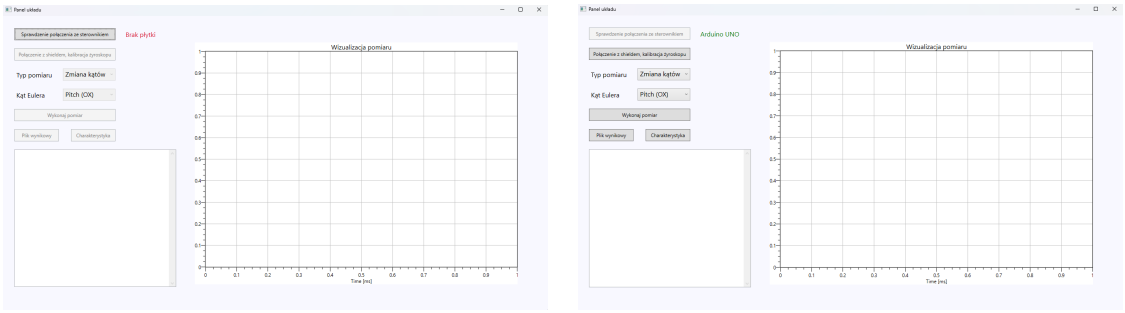
Listing 4.4: Funkcja get_gyr()

Listing 4.4 jest w rzeczywistości implementacją wzorów (3.3) i (3.4) z uwzględnieniem czasu trwania pomiaru. Dodatkowo, linie 19-20 są odpowiedzialne za włączanie/wyłączanie diody wraz z wykonywaniem kolejnych pomiarów. Funkcja `show()` służy do przesyłania wyników do komputera w sformatowanej formie, ułatwiając potem aplikacji rozdzielanie danych do odpowiednich tablic.

4.2. Aplikacja

4.2.1. Wygląd i funkcjonalności

Okno aplikacji prezentuje się następująco:



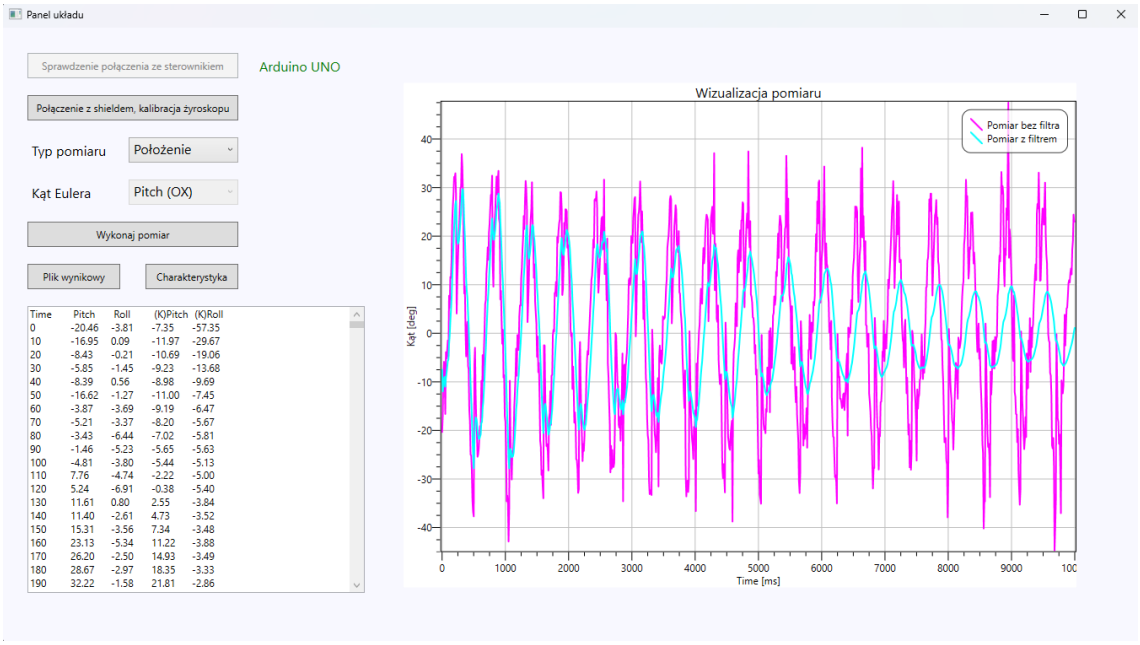
(a) Okno bez płytki (b) Okno z płytką

Rys. 4.1. Okno aplikacji

Aplikacja pozwala na pomiar:

- położenia względem wektora przyspieszenia ziemskiego względem osi OX i OY (charakterystyka przedstawia jednocześnie dane przed i po przefiltrowaniu),
- zmian jednego z kątów RPY (wybranego),
- pomiar szumów (ponownie z filtrem i bez) - położenie dla nieruchomego czujnika.

Okno aplikacji po wykonaniu pomiaru prezentuje następujące informacje (Rys. 4.2): nastawy wyboru (położenie i pitch), wartości pomiarowe w oknie tekstowym oraz wykres zależności wybranego kąta od czasu.



Rys. 4.2. Przykład działania aplikacji

4.2.2. Klasa aplikacji

```
    private void BtnStart_OnClick(object sender, RoutedEventArgs e)
    {

        switch (CmbType.Text)
        {
            case "Położenie":
                PortWrite("0");
                TxtStats.Clear();
                break;
            case "Szum":
                PortWrite("0");
                TxtStats.Clear();
                Plt.Title = "Pomiar szumu";
                break;
            case "Zmiana kątów RPY":
                PortWrite("1");
                TxtStats.Clear();
                break;
        }
        if (_port != null) _port.DataReceived += DataReceived;
    }
```

Listing 4.5: Metoda BtnStart_OnClick()

Listing 4.5 prezentuje główną metodę programu (odpowiadającą za obsługę przycisku *Wykonaj pomiar*). Wyglądem przypomina funkcję `void loop()`, z tą różnicą, że ta wysyła informację. Wariant dla szumu i położenia wysyła tę samą informację do płytki Arduino, ten pierwszy jednak dodatkowo zmienia jeszcze tytuł wykresu. Jako, że aplikacja w czasie rzeczywistym wyświetla pomiary w oknie tekstowym, należało wprowadzić wielowątkowość (linia 21), która umożliwia oprogramowaniu śledzenie zmian w zmiennej odbierającej dane (na bazie [11]).

Jednak zanim możliwe będzie wykonanie pomiarów, program musi upewnić się, że płytka Arduino jest dostępna. Służy do tego metoda `BtnCheck_OnClick()`.

```
1   private void BtnCheck_OnClick(object sender, RoutedEventArgs e)
2   {
3       try
4       {
5           _port = new SerialPort("COM3", 115200);
6           _port.Open();
7           LblName.Foreground = Brushes.Green;
8           LblName.Content = "Arduino UNO";
9
10          BtnCheck.IsEnabled = false;
11          BtnFile.IsEnabled = true;
12          BtnReset.IsEnabled = true;
13          BtnShow.IsEnabled = true;
14          BtnStart.IsEnabled = true;
15          CmbType.IsEnabled = true;
16          CmbAng.IsEnabled = true;
17      }
18      catch (IOException)
19      {
20          LblName.Foreground = Brushes.Crimson;
21          LblName.Content = "Brak płytki";
22      }
23  }
```

Listing 4.6: Metoda BtnCheck_OnClick()

Powyższy fragment kodu próbuje uruchomić połączenie szeregowe. Jeśli jest to wykonalne, kontakt zostanie nawiązany (o czym program poinformuje) i odblokowane zostaną wszystkie nastawy. W przeciwnym wypadku zostanie wystosowany komunikat o braku płytki (Rys. 4.1).

Ze względu na specyfikę kompilatora języka C# należało opracować metodę do modyfikacji danych (listing 4.7), ponieważ ten wymusza separator dziesiętny w zależności od ustawień regionalnych systemu operacyjnego. Okazało się to o tyle problematyczne, że dane generowane przez moduł SEN0142 część ułamkową miały zapisaną po kropce, gdy w Polsce oficjalnie jest to przecinek. Metoda (dla pomiarów z filtrem) dodatkowo pomija pomiary z pierwszych 200 ms, ze względu na wartości z filtra, które są obarczone błędem grubym.

```
1
2     private List<double> ModifyData(IReadOnlyList<string[]> d, int col, int len)
3     {
4         var temp = new List<double>();
5
6         var index = CmbType.Text is "Położenie" or "Szum" ? 21 : 1;
7
8         for (var i = index; i < len - 1; i++)
9         {
10            if (col != 0)
11            {
12                d[i][col] = d[i][col].Replace(".", ",");
13            }
14
15            temp.Add(double.Parse(d[i][col]));
16        }
17
18        return temp;
```

Listing 4.7: Metoda ModifyData()

Największą i najbardziej skomplikowaną metodą jest `BtnShow_OnClick()`. Odpowiada on w głównej mierze za rysowanie wykresów. Na poniższych listingach zostały zaprezentowane jedynie najważniejsze fragmenty kodu - ze względu na bardzo dużą ilość warunków logicznych, gdzie na każdą wariację przypadają podobne akcje, co najwyżej z innymi parametrami. Pokazano również fragmenty o unikalnym działaniu.

```
1     var data = TxtStats.Text;
2     var dataArray = data.Split('\n').Select(x => x.Split('\t')).ToArray();
```

Listing 4.8: BtnShow_OnClick - rozdzielanie danych)

Listing 4.8 prezentuje podział danych pobranych z okna tekstowego aplikacji za pomocą systemu zapytań kwerend LINQ. Na podstawie konkretnej sekwencji ucieczki (`\n` lub `\t`) dane są odpowiednio podzielone na dwuwymiarową tablicę.

```
1     var line = new LineGraph
2     {
3         Stroke = Brushes.Magenta,
4         StrokeThickness = 2
5     };
6     // ------------------------------------------------------
7     line.Plot(x, y3);
8     // ------------------------------------------------------
9     Plt.LegendVisibility = Visibility.Hidden;
10    Grid.Children.Clear();
11    Grid.Children.Add(line);
```

Listing 4.9: BtnShow_OnClick - kreślenie charakterystyki

Listing 4.9 prezentuje kolejne polecenia, które trzeba wykonać, by otrzymać charakterystykę. Wymagane jest każdorazowe wyczyszczenie siatki wykresu w celu zlikwidowania poprzednich wykresów.

```
1     case "Yaw (OZ)":
2         MessageBox.Show("Brak odpowiednich danych.");
3         break;
```

Listing 4.10: BtnShow_OnClick - zapobieganie złemu kreśleniu

Jako, że położenie mierzone jest względem osi OX lub OY, należy zabezpieczyć oprogramowanie przed potencjalną próbą wykonania analogicznej operacji dla osi OZ. Powyższy listing sprawia, że w przypadku takiej próby, program wyświetli komunikat z ostrzeżeniem.

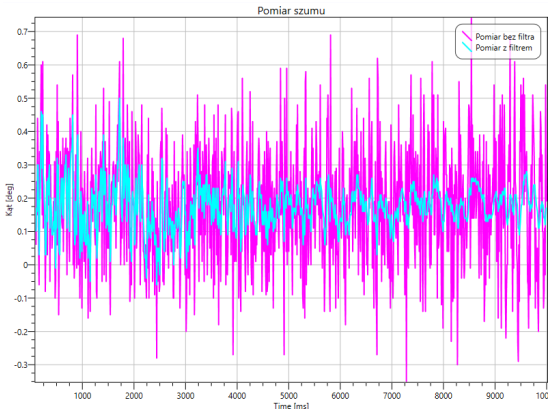
Ostatnia metoda, którą użytkownik może wywołać za pomocą przycisku, jest zapisanie pomiarów do pliku tekstowego (.txt). Funkcja jednak nie tworzy katalogu, zakłada że taki już istnieje.

```
1    private void BtnFile_OnClick(object sender, RoutedEventArgs e)
2    {
3        var appPath = Directory.GetParent(Environment.CurrentDirectory)?.Parent?.
     Parent?.FullName;
4        var path =
5            $"{appPath}\\logs\\logfile_{DateTime.Now:yyyy-MM-dd_HH-mm-ss-fff}.txt";
6        File.WriteAllText(path, TxtStats.Text);
7    }
```

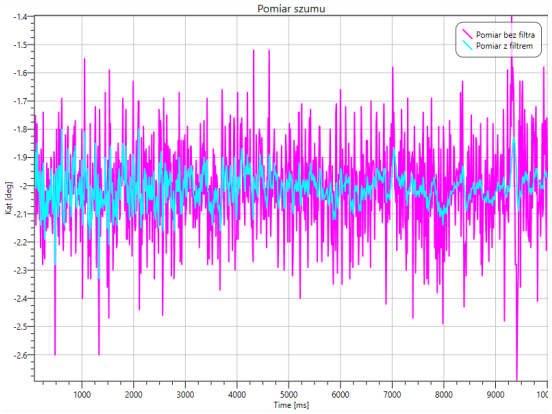
Listing 4.11: Metoda BtnFile_OnClick()

4.3. Pomiar szumów

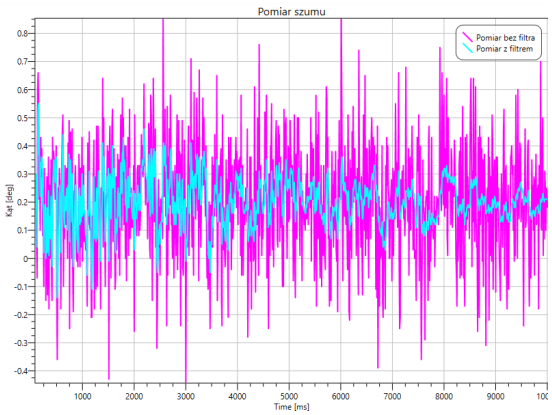
W celu oceny wykonano dwa pomiary szumów - dla czujnika leżącego i stojącego. Otrzymano następujące charakterystyki:



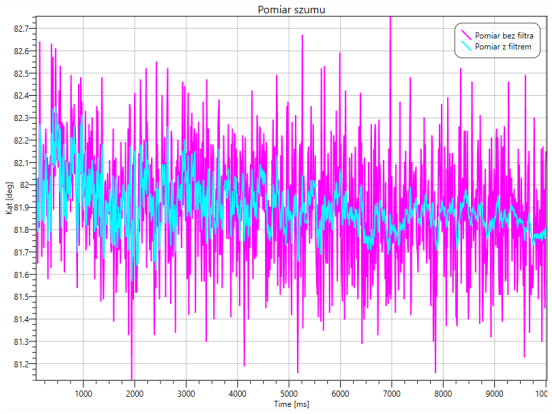
(a) Szum dla θ



(b) Szum dla ϕ



(c) Szum dla θ



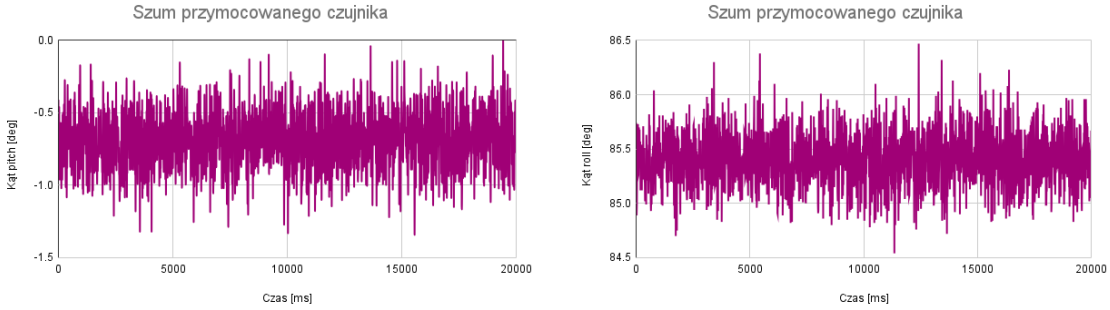
(d) Szum dla ϕ

Rys. 4.3. Charakterystyki czujnika: leżącego ((a) i (b)) i stojącego ((c) i (d))

Widać, że podane szumy w każdym z przypadków wprowadzają niepewność pomiarową wynoszącą 1^o. Filtr zmniejsza ją do poziomu 0.5^o. Jednak by ten zaczął poprawnie działać potrzebne jest kilkadziesiąt próbek by estymacja osiągała sensowne wartości.

4.4. Pomiary laboratoryjne

Moduł SEN0142 został zamocowany do elementu obrotowego silnika za pomocą kleju *Kropelka*. Został on wytypowany na podstawie tego, że wiąże materiały na sztywno, tak więc czujnik nie będzie ciągnięty przez powierzchnię, na której się znajduje, a raczej stworzy z nią integralną całość. W pierwszej kolejności po przymocowaniu czujnika wykreślono jego szum:



(a) Szum dla θ (b) Szum dla ϕ

Rys. 4.4. Wykresy dla przymocowanego czujnika

Uśredniając pomiary wykorzystane do wykresów otrzymano wartości początkowe (wraz z ich odchyleniami standardowymi) w jakich ustawiony jest czujnik. Wyniosły one:

- Kąt pitch $\theta = -0.69 \pm 0.19$ [deg]
- Kąt roll $\phi = 85.41 \pm 0.25$ [deg]

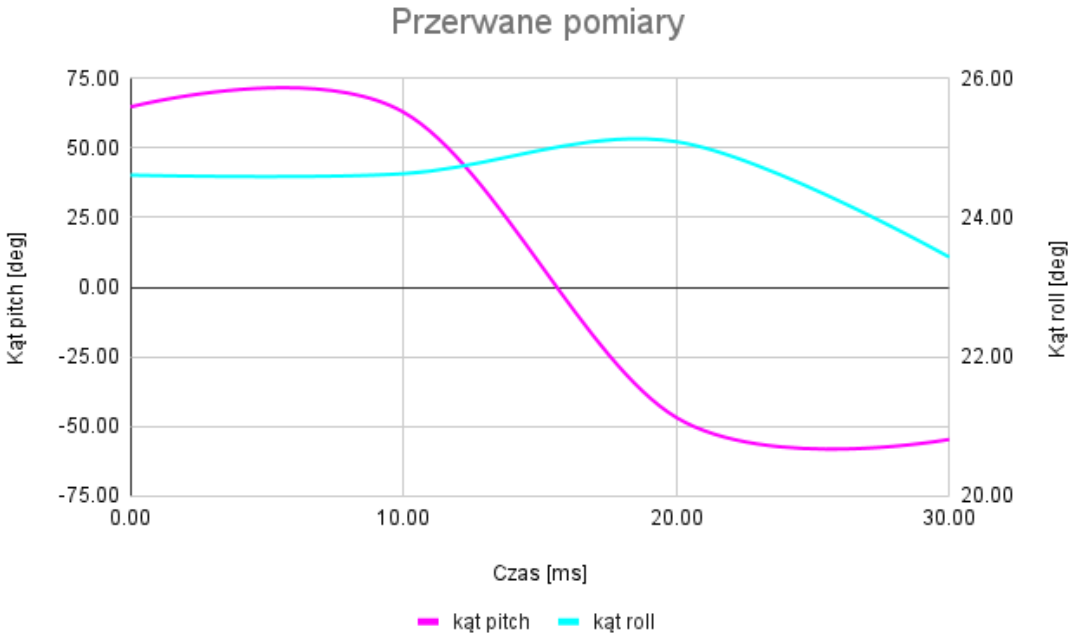
Widać, że wartości nie wynoszą idealnie 0^o i 90^o. Biorąc pod uwagę, że sam LCT znajduje się w fazie prototypowej, należy uwzględnić błąd wynikający z ręcznego składania urządzenia (np. fakt, że samo zwierciadło nie jest idealnie wycentrowane).

Po włączeniu symulacji na tomografie i załączeniu pomiaru położenia w aplikacji, czujnik po kilku pomiarach zamilkł. W przypadku pomiaru zmiany kątów żyroskopem nie otrzymano żadnej odpowiedzi.

Tabela 4.1. Otrzymane wyniki podczas ruchu zwierciadła

Czas [ms]	0	10	20	30
Pitch [deg]	64.75	62.86	-46.87	-54.72
Roll [deg]	24.61	24.63	25.09	23.43

Po wykreśleniu dane przedstawiły się następująco:



Rys. 4.5. Nieukończone pomiary

Na charakterystyce kąta pitch widać, zapoczątkowanie ruchu okresowego, jednakże nic więcej nie da się z owej wizualizacji wywnioskować.

Potencjalnym źródłem błędów mogą być asymetryczne działanie momentów bezwładności wprowadzone przez przewody podłączone do czujnika. Przewody zostały ze sobą sklejone oraz usztywnione w odpowiednich miejscach, przyjęto więc wzór na moment bezwładności pręta o długości l nachylonego do osi obrotu pod kątem α.

$$I = \frac{1}{3} m l^2 \sin^2 \alpha \tag{4.1}$$

W celu wyznaczenia niepewności wykorzystano metodę różniczki zupełnej:

$$\delta I = \left|\frac{\partial I}{\partial m}\right| \delta m + \left|\frac{\partial I}{\partial l}\right| \delta l + \left|\frac{\partial I}{\partial \alpha}\right| \delta \alpha \tag{4.2}$$

Dane:

- $m = 18 \pm 1$ [g],
- $l = 45.0 \pm 0.1$ [cm],
- $\alpha = 60^o \pm 5^o$

Moment bezwładności otrzymany wraz z niepewnością wyniósł:

$$I = \frac{1}{3} \cdot 18 \cdot 45^2 \cdot \sin^2 60 = 0.911 \pm 0.005 \; [kg \cdot m^2]$$

Widać, że moment ten występuje, może więc on bezpośrednio zaburzać ruch masy w akcelerometrze, przez co czujnik przestaje odczytywać dane. Dodatkowo, przez ręczne zamontowanie elementów (zwierciadło, czujnik) elementy te nie są umieszczone idealnie symetrycznie, co w efekcie wprowadza asymetrię do rozkładu momentów bezwładności.

5. PODSUMOWANIE I WNIOSKI KOŃCOWE

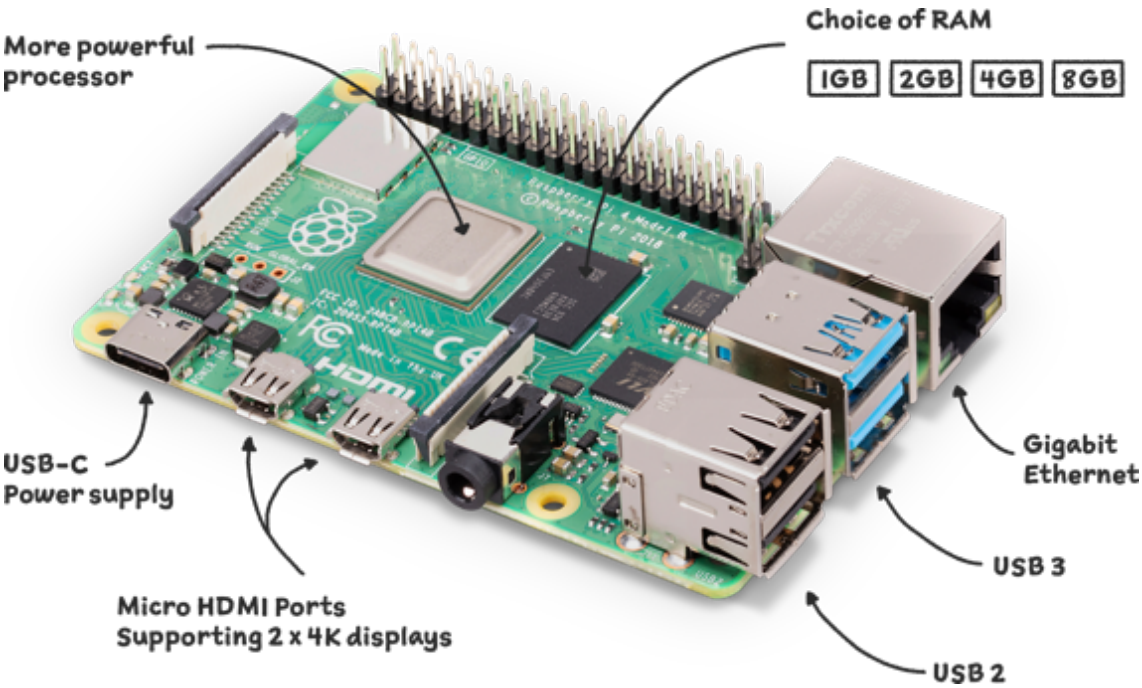
Opracowane rozwiązanie okazało się niewystarczające. Pomimo spełniania teoretycznych założeń, w praktyce opracowany czujnik nie spełnia swojej funkcji. Wpływa na to parę czynników.

- Czujnik przy dużych prędkościach przerywa pomiar, traci orientację w przestrzeni.
- Przewody łączące sensor SEN0142 z płytką Arduino wprowadzają dodatkowe zaburzenia:
  - są to elementy posiadające masę, dodają więc dodatkowe obciążenie do układu, co przekłada się na zmiany momentów bezwładności elementu obrotowego silnika krokowego,
  - przewody nie są trwale przyłączone (przylutowane, bądź zaciśnięte rurką termokurczliwą) do modułu SEN0142, nałożono jedynie złącza żeńskie na piny, może więc dochodzić do spadków napięcia, co zaburza prawidłową pracę czujnika.
- Ograniczenie częstotliwości próbkowania do 100 Hz niekorzystnie wpływa na jakość samego pomiaru.
- Silnik krokowy jest rozwiązaniem tymczasowym, ponieważ wymusza długi czas symulacji wykonywanej przez LCT. Rozważa się zastosowanie silnika rezonansowego, lub też skanera galwanometrycznego, które docelowo mają skrócić symulację. Czujnik, przez częstotliwość, na której pracuje, nie znajdzie zastosowania w owej technologii (jest zbyt ograniczony).
- Opracowane rozwiązanie miałoby zastosowanie przy badaniu obiektów powolnie zmieniających swoje położenie w przestrzeni. Przy prędkościach z jakimi porusza się silnik krokowy, czujnik obecnie nie jest przydatny.

Kolejnym ograniczającym rozwiązaniem było wykorzystanie *Microsoft WPF*. Pomimo, że umożliwia łatwe i szybkie utworzenie aplikacji z graficznym interfejsem, blokuje ono wykorzystanie programu na innych systemach operacyjnych niż *Microsoft Windows*. Jeśli aplikacja miałaby być multiplatformowa (potencjalnie z implementacją również na urządzeniach mobilnych) warto by było zastosować platformę do stworzenia aplikacji webowej (np. React.JS, Angular, Google Flutter), dzięki czemu zostałoby zniwelowane ograniczenie do jednej platformy. Co więcej, przy integracji z bazą danych, możliwe by było jednoczesne składowanie wcześniej wykonanych pomiarów.

Należałoby również wymienić moduł na taki, który posiada wbudowany przekaźnik bezprzewodowy. Zniwelowałoby to problem jakim są przewody, a raczej ich masa. Jednakże, w takim wypadku trzeba zagwarantować jak najmniejszą stratę danych.

Rozwiązaniem, które umożliwiłoby pracę czujnika na większych częstotliwościach, wymagałoby wykorzystania pamięci Arduino - skrypt zapisywałby pomiary na płytce, a dopiero potem przesyłał je do komputera. Byłoby to problematyczne o tyle, że tej jest relatywnie niewiele. Należałoby zastosować rozwiązanie technologiczne umożliwiające podłączenie dodatkowej pamięci, np. Raspberry Pi. Byłoby ono o tyle wygodne, że w odróżnieniu od Arduino, które jest jedynie płytką z mikrokontrolerem, jest to w pełni funkcjonalny komputer, z własnym systemem operacyjnym na jądrze Linuxa. Takie rozwiązanie byłoby rozsądniejsze o tyle, że samą aplikację sterującą można by zaimplementować już na Raspberry Pi. Dodatkowym atutem jest łatwiejsze programowanie - to bazuje na Bashu lub Pythonie.



Rys. 5.1. Komputer Raspberry Pi 4B [12]

Na rys. 5.1 widać, że zniknąłby problem z prędkością przesyłania danych, bo te byłyby bezpośrednio składowane na Raspberry Pi, lub też pamięci zewnętrznej podłączonej do tego komputera. Samo urządzenie ma wbudowane złącze Ethernet (dostęp do internetu w razie potrzeby jest więc zapewniony). Można je również skonfigurować z mocniejszym komputerem by działały w relacji master-slave. Wadą tego rozwiązania jest jednak cena zestawu - na przestrzeni ostatnich lat Raspberry Pi podrożało, gdyż zaczęto je wykorzystywać do kopania kryptowalut.

WYKAZ LITERATURY

[1] Kristy K. Brock, Sasa Mutic, Todd R. McNutt, Hua Li, Marc L. Kessler: *Use of image registration and fusion algorithms and techniques in radiotherapy: Report of the AAPM Radiation Therapy Committee Task Group No. 132*, (2017) Med. Phys. Volume 44, Issue 7, data dostępu: 06.01.2022 r.

[2] K. Sjogreen-Gleisner, D. Rueckert, M. Ljungberg: *Registration of serial SPECT/CT images for threedimensional dosimetry in radionuclide therapy*, (2009) Physics in Medicine & Biology, Volume 54, Number 20, data dostępu: 06.01.2022 r.

[3] M. Maryanski (2018), *High-Definition 3D Dosimetry for End-To-End Patient-Specific Treatment Delivery Verification*. W: Chan Maria F., Editor, *Recent Advancements and Applications in Dosimetry*. Wyd. Nova, ISBN: 978-1-53613-759-0.

[4] Dokumentacja 3-osiowego inklinometru SCL3300-D01, data dostępu: 04.01.2022 r.

[5] Dr inż. Marek Wnuk: *Filtracja komplementarna w inercyjnych czujnikach orientacji*, Politechnika Wrocławska, Wrocław 2014, data dostępu: 12.12.2022 r.

[6] Dr inż. Piotr Suchomski: *Liniowe filtry Kalmana*, Politechnika Gdańska, data dostępu: 12.12.2022 r.

[7] *Oficjalna dokumentacja Arduino Uno*, data dostępu: 12.12.2022 r.

[8] *Sklep producenta*, data dostępu: 29.12.2022 r.

[9] *MPU-6000 and MPU-6050 Product Specification Revision 3.4*, InvenSense, Sunnyvale 2013, data dostępu: 12.12.2022 r.

[10] Forbot, *Magistarala I2C*

[11] *Dokumentacja języka C# i platformy .NET*, data dostępu: 20.12.2022 r.

[12] *Oficjalna strona rozwiązania Raspberry Pi*, data dostępu: 29.12.2022 r.

WYKAZ RYSUNKÓW

WYKAZ LISTINGÓW

WYKAZ TABEL

DODATKI

Poniżej znajdują sią kody źródłowe opracowanego rozwiązania, pełne projekty dostępne na:

- Arduino: `https://github.com/StachRach/dyplom-arduino-final.git`
- Aplikacja: `https://github.com/StachRach/dyplom-WPF.git`

```
// ------------------------------------------------
//              IMPLEMENTACJE BIBLIOTEK
// ------------------------------------------------

#include <Arduino.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <KalmanFilter.h>

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// ------------------------------------------------
//            DEFINICJE ZMIENNYCH GLOBALNYCH
// ------------------------------------------------

MPU6050 mpu;

bool blink = false;
unsigned long timer;

// ------------------------------------------------
//                 DEFINICJE FUNKCJI
// ------------------------------------------------

void init_mpu();
void get_gyr(int time, int t_s);
void filtered_angles(int time, int t);
void show(int t, double a_x, double a_y, double a_z);

// ------------------------------------------------
//              FUNKCJA STARTOWA - SETUP
// ------------------------------------------------

__attribute__((unused)) void setup() {
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000L);
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
#endif

    Serial.begin(115200);
}

// ------------------------------------------------
//        GŁÓWNE CIAŁO PROGRAMU - FUNKCJA LOOP
```

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

__attribute__((unused)) void loop() {
    String value = Serial.readString();

    if (value == "0") {
        filtered_angles(10000, 10);
    }
    else if (value == "1") {
        get_gyr(10000, 10);
    }
    else if (value == "9") {
        init_mpu();
    }
}

// ------------------------------------------------
//            CIAŁA ZDEFINIOWANYCH FUNKCJI
// ------------------------------------------------

void init_mpu() {
    Serial.println("Sprawdzanie polaczenia z shieldem...");
    Serial.println(mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)
        ? "Polaczenie udane." : "Polaczenie nieudane.");

    Serial.println("Kalibracja zyroskopu...");
    mpu.calibrateGyro();
    Serial.println("Kalibracja ukonczona.");

    mpu.setThreshold(0);
}

void filtered_angles(int time, int t) {
    KalmanFilter kalmanX(0.001, 0.003, 0.03);
    KalmanFilter kalmanY(0.001, 0.003, 0.03);

    Serial.println("Czas \t Pitch \t Roll \t (K)Pitch \t (K)Roll");

    for (long i = 0; i <= time; i += t) {
        Vector acc = mpu.readNormalizeAccel();
        Vector gyr = mpu.readNormalizeGyro();

        timer = millis();

        double acc_pitch = -(atan2(acc.XAxis, sqrt(square(acc.YAxis) + square(acc.
    ZAxis))) * 180.0) / M_PI;
        double acc_roll = (atan2(acc.YAxis, sqrt(square(acc.XAxis) + square(acc.
    ZAxis))) * 180.0) / M_PI;

        double kal_pitch = kalmanY.update(acc_pitch, gyr.YAxis);
        double kal_roll = kalmanX.update(acc_roll, gyr.XAxis);

        Serial.print(i); Serial.print("\t");
        Serial.print(acc_pitch); Serial.print("\t"); Serial.print(acc_roll); Serial
    .print("\t");
        Serial.print(kal_pitch); Serial.print("\t"); Serial.println(kal_roll);

```

```
        blink = !blink;
        digitalWrite(LED_BUILTIN, blink);

        delay(millis() - timer);
    }
}

void get_gyr(int time, int t_s) {
    Serial.println("t [ms] \t pitch \t roll \t yaw");

    double pitch = 0;
    double roll = 0;
    double yaw = 0;

    for (int i = 0; i <= time; i+= t_s) {
        Vector gyr = mpu.readNormalizeGyro();

        timer = millis();

        pitch += gyr.YAxis * t_s / 1000;
        roll += gyr.XAxis * t_s / 1000;
        yaw += gyr.ZAxis * t_s / 1000;

        show(i, pitch, roll, yaw);

        blink = !blink;
        digitalWrite(LED_BUILTIN, blink);

        delay(millis() - timer);
    }
}

void show(int t, double a_x, double a_y, double a_z) {
    Serial.print(t); Serial.print("\t");
    Serial.print(a_x); Serial.print("\t");
    Serial.print(a_y); Serial.print("\t");
    Serial.println(a_z);
}
```

Listing 5.1: Skrypt Arduino

```
// ------------------------------------------------
//               IMPLEMENTACJE NUGETÓW
// ------------------------------------------------

using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Windows;
using System.IO.Ports;
using System.Windows.Media;
using InteractiveDataDisplay.WPF;
using DispatcherPriority = System.Windows.Threading.DispatcherPriority;

// ------------------------------------------------
//              OKREŚLENIE PRZESTRZENI NAZW
```

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

namespace dyplom_WPF;

// ------------------------------------------------
//                GŁÓWNA KLASA PROGRAMU
// ------------------------------------------------

public partial class MainWindow
{
    private static SerialPort? _port;

    public MainWindow()
    {
        InitializeComponent();
    }

// ------------------------------------------------------
// METODY OBSŁUGUJĄCE ZDARZENIA ZWIĄZANE Z ODBIOREM DANYCH
// ------------------------------------------------------

    private void DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        var sp = (SerialPort)sender;
        var data = sp.ReadExisting();
        Dispatcher.BeginInvoke(DispatcherPriority.Normal, (Action<string>)Update,
    data);
    }

    private void Update(string data)
    {
        TxtStats.Text += data;
    }

// ------------------------------------------------------
// BEZPIECZNA METODA WYSYŁANIA INFORMACJI PORTEM SZEREGOWYM
// ------------------------------------------------------

    private static void PortWrite(string message)
    {
        if (_port is { IsOpen: true })
        {
            _port.Write(message);
        }
    }

// ------------------------------------------------
//      METODA SPRAWDZAJĄCA POŁĄCZENIE Z ARDUINO
// ------------------------------------------------

    private void BtnCheck_OnClick(object sender, RoutedEventArgs e)
    {
        try
        {
            _port = new SerialPort("COM3", 115200);
            _port.Open();
            LblName.Foreground = Brushes.Green;
```

```
73              LblName.Content = "Arduino UNO";
74
75              BtnCheck.IsEnabled = false;
76              BtnFile.IsEnabled = true;
77              BtnReset.IsEnabled = true;
78              BtnShow.IsEnabled = true;
79              BtnStart.IsEnabled = true;
80              CmbType.IsEnabled = true;
81              CmbAng.IsEnabled = true;
82          }
83          catch (IOException)
84          {
85              LblName.Foreground = Brushes.Crimson;
86              LblName.Content = "Brak płytki";
87          }
88      }
89
90  // ------------------------------------------------
91  //        METODA ZARZĄDZAJĄCA NASTAWAMI POMIARÓW
92  // ------------------------------------------------
93
94      private void BtnStart_OnClick(object sender, RoutedEventArgs e)
95      {
96
97          switch (CmbType.Text)
98          {
99              case "Położenie":
100                 PortWrite("0");
101                 TxtStats.Clear();
102                 break;
103             case "Szum":
104                 PortWrite("0");
105                 TxtStats.Clear();
106                 Plt.Title = "Pomiar szumu";
107                 break;
108             case "Zmiana kątów RPY":
109                 PortWrite("1");
110                 TxtStats.Clear();
111                 break;
112         }
113         if (_port != null) _port.DataReceived += DataReceived;
114     }
115
116 // ------------------------------------------------
117 //      METODA ZAPISUJĄCA POMIARY DO PLIKU .TXT
118 // ------------------------------------------------
119
120     private void BtnFile_OnClick(object sender, RoutedEventArgs e)
121     {
122         var appPath = Directory.GetParent(Environment.CurrentDirectory)?.Parent?.
    Parent?.FullName;
123         var path =
124             $"{appPath}\\logs\\logfile_{DateTime.Now:yyyy-MM-dd_HH-mm-ss-fff}.txt";
125         File.WriteAllText(path, TxtStats.Text);
126     }
127
128 // ------------------------------------------------
```

```
129 //            METODA KREŚLĄCA CHARAKTERYSTYKI
130 // ------------------------------------------------
131
132     private void BtnShow_OnClick(object sender, RoutedEventArgs e)
133     {
134     var data = TxtStats.Text;
135     var dataArray = data.Split('\n').Select(x => x.Split('\t')).ToArray();
136
137         var x = ModifyData(dataArray, 0, dataArray.Length);
138
139         List<double> y1;
140         List<double> y2;
141
142         switch (CmbType.Text)
143         {
144             case "Zmiana kątów RPY":
145                 y1 = ModifyData(dataArray, 1, dataArray.Length);
146                 y2 = ModifyData(dataArray, 2, dataArray.Length);
147                 var y3 = ModifyData(dataArray, 3, dataArray.Length);
148
149                 var line = new LineGraph
150                 {
151                     Stroke = Brushes.Magenta,
152                     StrokeThickness = 2
153                 };
154
155                 switch (CmbAng.Text)
156                 {
157                     case "Yaw (OZ)":
158                         line.Plot(x, y3);
159                         break;
160                     case "Roll (OY)":
161                         line.Plot(x, y2);
162                         break;
163                     default:
164                         line.Plot(x, y1);
165                         break;
166                 }
167
168                 Plt.LegendVisibility = Visibility.Hidden;
169                 Grid.Children.Clear();
170                 Grid.Children.Add(line);
171                 break;
172             default:
173                 switch (CmbAng.Text)
174                 {
175                     case "Pitch (OX)":
176                         y1 = ModifyData(dataArray, 1, dataArray.Length);
177                         y2 = ModifyData(dataArray, 3, dataArray.Length);
178                         PlotData(x, y1, y2);
179                         break;
180                     case "Roll (OY)":
181                         y1 = ModifyData(dataArray, 2, dataArray.Length);
182                         y2 = ModifyData(dataArray, 4, dataArray.Length);
183                         PlotData(x, y1, y2);
184                         break;
185                     case "Yaw (OZ)":
```

```
186                         MessageBox.Show("Brak odpowiednich danych.");
187                         break;
188                 }
189                 break;
190         }
191     }
192 
193     private void PlotData(List<double> a, IEnumerable<double> b, IEnumerable<double
    > c)
194     {
195         var line1 = new LineGraph
196         {
197             Stroke = Brushes.Magenta,
198             Description = "Pomiar bez filtra",
199             StrokeThickness = 2
200         };
201         var line2 = new LineGraph
202         {
203             Stroke = Brushes.Cyan,
204             Description = "Pomiar z filtrem",
205             StrokeThickness = 2
206         };
207 
208         line1.Plot(a, b);
209         line2.Plot(a, c);
210 
211         Grid.Children.Clear();
212         Grid.Children.Add(line1);
213         Grid.Children.Add(line2);
214     }
215 
216 // ------------------------------------------------
217 //               METODA OBRABIAJĄCA DANE
218 // ------------------------------------------------
219 
220     private List<double> ModifyData(IReadOnlyList<string[]> d, int col, int len)
221     {
222         var temp = new List<double>();
223 
224         var index = CmbType.Text is "Położenie" or "Szum" ? 21 : 1;
225 
226         for (var i = index; i < len - 1; i++)
227         {
228             if (col != 0)
229             {
230                 d[i][col] = d[i][col].Replace(".", ",");
231             }
232 
233             temp.Add(double.Parse(d[i][col]));
234         }
235 
236         return temp;
237     }
238 // ------------------------------------------------
239 //               METODA REGULUJĄCA OFFSET
240 // ------------------------------------------------
241 
```

```
    private void BtnReset_OnClick(object sender, RoutedEventArgs e)
    {
        PortWrite("9");
        TxtStats.Clear();
        if (_port != null) _port.DataReceived += DataReceived;
    }
}
```

Listing 5.2: Klasa aplikacji

```
<Window x:Class="dyplom_WPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d3="clr-namespace:InteractiveDataDisplay.WPF;assembly=
    InteractiveDataDisplay.WPF"
        mc:Ignorable="d"
        Title="Panel układu" Height="768" Width="1366" d:DataContext="{
    d:DesignInstance}">

<!-- ZNACZNIKI DEFINIUJĄCE WYGLĄD APLIKACJI -->

    <Grid Background="GhostWhite">
        <Button x:Name="BtnCheck" Click="BtnCheck_OnClick" HorizontalAlignment="
    Left" VerticalAlignment="Top" Margin="30 30 0 0"
                Content="Sprawdzenie połączenia ze sterownikiem" Height="30" Width=
    "250"/>
        <Label x:Name="LblName" HorizontalAlignment="Left" VerticalAlignment="Top"
    Margin="300 30 0 0"
               Content="Nazwa układu" Height="30" Width="150" FontSize="12pt"
    Foreground="#888888"/>

        <Button x:Name="BtnReset" Click="BtnReset_OnClick" IsEnabled="False"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="30 80 0 0"
                Content="Połączenie z shieldem, kalibracja żyroskopu" Height="30"
    Width="250"/>

        <Label x:Name="LblType" HorizontalAlignment="Left" VerticalAlignment="Top"
    Margin="30 130 0 0"
               Content="Typ pomiaru" Height="30" Width="112" FontSize="12pt"/>
        <ComboBox x:Name="CmbType" IsEnabled="False" SelectedIndex="0"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="150 130 0 0"
                  Height="30" Width="130" FontSize="12pt">
            <ComboBoxItem>Zmiana kątów RPY</ComboBoxItem>
            <ComboBoxItem>Położenie</ComboBoxItem>
            <ComboBoxItem>Szum</ComboBoxItem>
        </ComboBox>

        <Label x:Name="LblAng" HorizontalAlignment="Left" VerticalAlignment="Top"
    Margin="30 180 0 0"
               Content="Kąt Eulera" Height="30" Width="112" FontSize="12pt"/>
        <ComboBox x:Name="CmbAng" IsEnabled="False" SelectedIndex="0"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="150 180 0 0"
                  Height="30" Width="130" FontSize="12pt">
            <ComboBoxItem>Pitch (OX)</ComboBoxItem>
            <ComboBoxItem>Roll (OY)</ComboBoxItem>
```

```
36              <ComboBoxItem>Yaw (OZ)</ComboBoxItem>
37          </ComboBox>
38
39          <Button x:Name="BtnStart" IsEnabled="False" Click="BtnStart_OnClick"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="30 230 0 0"
40                  Content="Wykonaj pomiar" Height="30" Width="250"/>
41
42          <Button x:Name="BtnFile" Click="BtnFile_OnClick" IsEnabled="False"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="30 280 0 0"
43                  Content="Plik wynikowy" Height="30" Width="110"/>
44          <Button x:Name="BtnShow" Click="BtnShow_OnClick" IsEnabled="False"
    HorizontalAlignment="Left" VerticalAlignment="Top" Margin="170 280 0 0"
45                  Content="Charakterystyka" Height="30" Width="110"/>
46
47          <TextBox x:Name="TxtStats" IsReadOnly="True" VerticalScrollBarVisibility="
    Visible" Margin="30 330 0 0"
48                   HorizontalAlignment="Left" VerticalAlignment="Top" Height="340"
    Width="400"/>
49
50          <d3:Chart x:Name="Plt" Title="Wizualizacja pomiaru" LeftTitle="Kąt [deg]"
    BottomTitle="Czas [ms]" Height="600" Width="800"
51                    Margin="400 0 0 0">
52                  <Grid Name="Grid"/>
53          </d3:Chart>
54      </Grid>
55  </Window>
```

Listing 5.3: Formularz XAML aplikacji