

Autor: Radosław Sajdak

**Kod źródłowy aplikacji**

https://github.com/RadoslawSajdak/Recognizing\_bike\_collision

[**Wstęp**](#_n08c282bef1c) **3**

[Cel pracy](#_xak26fi8fx99) 3

[Założenia](#_juieqowxnfc8) 3

[Analiza wymagań technicznych](#_p4gsv4go4un5) **3**

[Akcelerometr](#_6zfjpof2aml4) 4

[Analiza wymagań dla układu](#_qnz3yz15isvo) 4

[MPU-6050](#_upes4vck7lkz) 4

[LSM9DS1](#_hny6jyu59ynd) 5

[LSM6DSOX](#_6jeq1kscn5hy) 5

[Porównanie wybranych układów](#_d19bgepqo9l1) 5

[Mikrokontroler](#_696f1ausorxh) 6

[Analiza wymagań dla układu](#_fg8etkukbacb) 6

[Porównanie popularnych układów](#_ykm5kryr8kav) 6

[GSM](#_2xfsypljxuc8) 6

[Analiza wymagań dla układu](#_hbanaez0xjby) 6

[Porównanie popularnych układów](#_6gibtsvzn82n) 6

[GPS](#_r46av9gcxruc) 6

[Analiza wymagań dla układu](#_todqy08xj03n) 6

[Porównanie popularnych układów](#_fgeu8648qggb) 6

[**Zaplanowanie prototypu urządzenia wykrywającego upadek na rowerze**](#_xim1cvqbvjn4) **7**

[Problematyka](#_tv7ceyt2n0h) 7

[Algorytm wykrywania kolizji](#_xhzemvi2rp3i) 7

[Algorytm działania urządzenia](#_funytfd7xfp) 7

[**Wykonanie prototypu urządzenia**](#_xlbn5zq825xr) **7**

[Akcelerometr](#_96bgejpqwx06) 7

[Implementacja algorytmu wykrywania zdarzenia](#_8ieu45txs6p) 7

[Analiza zużycia energii układu w różnych trybach](#_um6nodhcwgfn) 7

[GSM](#_fl1m20vix1tf) 7

[Implementacja powiadamiania o zdarzeniu](#_40u7xsuciili) 7

[Analiza zużycia energii układu](#_elayiel4p4ae) 8

[GPS](#_t3ayka71phaj) 8

[Implementacja znajdowania lokalizacji](#_b4aebt6uoold) 8

[Analiza zużycia energii układu](#_85461jjpvvmr) 8

[**Testowanie urządzenia**](#_a7dsvlsnbpo9) **8**

[Metody testowania](#_czxyxwipkn0z) 8

[**Możliwości rozwoju urządzenia**](#_peg3lr3emqyd) **8**

[**Podsumowanie pracy**](#_kssrt8euekdg) **8**

[**Zakończenie i oklaski**](#_lu8mcl4bf4ma) **8**

# Wstęp

Obecny rozwój mikroprocesorów, pozwala na tworzenie coraz bardziej złożonych urządzeń. Rozwój układów o niskim zużyciu energii, popycha elektronikę w kierunku małych, wielofunkcyjnych urządzeń. Połączenie tych dwóch procesów pozwala na stworzenie elastycznych urządzeń, których zastosowanie może zmieniać się jedynie dzięki oprogramowaniu.

Jako rowerzysta górski, podczas samotnych wypraw rowerowych, wielokrotnie zastanawiałem się… Co w sytuacji, gdy ulegnę wypadkowi podczas samotnej wyprawy? Jak długo nikt nie wezwie pomocy? Tym samym, narodził się pomysł stworzenia urządzenia asystującego.

Prawdopodobnie, podobne pytania, zadali sobie inżynierowie firmy Specialized. Stworzyli oni bowiem urządzenie, o nazwie ANGi[[1]](#footnote-0). Jest to małe urządzenie, wyposażone w akcelerometr i żyroskop. Komunikuje się ono przy użyciu Bluetooth z ich autorską aplikacją na telefon z iOS lub Androidem. Urządzenie, zamontowane na kask, wysyła powiadomienie do aplikacji, gdy rowerzysta uderzy kaskiem w przeszkodę. Rozwiązanie to, ma szereg zalet, takich jak prostota budowy i bardzo niskie zużycie energii. Jest ono jednak uzależnione od telefonu, który w przypadku wycieczek górskich, czasem wygodniej zostawić w domu.

## Cel pracy

Celem pracy jest stworzenie innowacyjnego urządzenia wykrywającego upadek na rowerze. Urządzenie powinno być mocowane do ramy roweru. Z uwagi na różnorodność geometrii dostępnych na rynku ram rowerowych, powinno posiadać możliwość kalibracji. Urządzenie, informacje o wypadku wysyłać będzie przy użyciu modułu LTE-M opartego o interfejs szeregowy UART. Za lokalizowanie urządzenia, odpowiadać będzie popularny moduł GPS oparty o magistralę I2C. Całość, sterowana będzie przy użyciu mikrokontrolera firmy NordicSemiconductor, z wykorzystaniem systemu operacyjnego czasu rzeczywistego Zephyr.

## Założenia

Ponieważ docelowo, urządzenie ma zwiększać bezpieczeństwo podczas wypraw rowerowych, musi być bardzo energooszczędne. Minimalne wymaganie, to 24 godziny na jednym ładowaniu baterii. Jednocześnie, nie może być duże, aby w łatwy sposób można było zamontować je na rowerze. Pobierana lokalizacja, powinna mieć dokładność około 100m. Jest to dokładność wystarczająca, aby zobaczyć ranną osobę, leżący rower, lub usłyszeć wołanie o pomoc. Ważnym jest, aby moje urządzenie było w pełni niezależne od innych urządzeń, jak np. telefon.

# Analiza wymagań technicznych

Planując pracę, zdecydowałem się na wykorzystanie trzech modułów:

* Akcelerometru
* GPS
* LTE

Dodatkowo, ze względu na łatwy w użyciu stos bluetooth, wykorzystałem również Bluetooth Low Energy celem stworzenia bezprzewodowego interfejsu sterowania urządzeniem. Ponieważ na rynku dostępne jest wiele różnych modułów, w niniejszym punkcie chciałbym dokonać ich porównania, a następnie wybrać układy, najlepsze dla mojego rozwiązania.

## Akcelerometr

Akcelerometry to układy, mierzące przyspieszenie. Mogą dokonywać pomiaru przyspieszenia statycznego (np. Ziemskiego), lub dynamicznego, wynikającego z sił działających na układ. W przypadku dostępnych na rynku akcelerometrów, należy pamiętać, że w stanie spoczynku wskazują one przyspieszenie około 9.81m/s2, a więc w moim rozwiązaniu będzie trzeba zastosować offset przyspieszenia.

Obecnie, większość układów to układy integrujące akcelerometr i żyroskop w jednym układzie scalonym. Coraz częściej, można spotkać też magnetometr. Dla obecnych na rynku układów wyróżniamy dwa najważniejsze parametry:

* Zakres pracy akcelerometru - określany jako +-Xg, a więc przyspieszenie w trzech kierunkach, podane jako wielokrotność przyspieszenia ziemskiego. Zazwyczaj, wartość ta, mieści się w przedziale kilku do kilkunastu g.
* Zakres pomiarów żyroskopu - określany jako dps (degrees per second). Jeśli prędkość kątowa będzie więc większa, niż wybrany zakres, układ ulegnie nasyceniu.

### Analiza wymagań dla układu

Głównym wymaganiem dotyczącym akcelerometru, jest jego energooszczędność. Jest to jedyny układ, którego działanie, wymagane jest przez cały czas. Z tego powodu, akcelerometr powinien nie tylko zużywać mało prądu, ale również posiadać różne tryby pracy. Dodatkowym atutem akcelerometru, będzie wbudowana pamięć, pozwalająca na buforowanie danych. Dzięki temu, mikrokontroler będzie mógł być uśpiony dłużej.

### MPU-6050

Wybrany układ, jest 3-osiowym akcelerometrem i żyroskopem. Korzysta on z magistrali I2C. Zgodnie z dokumentacją układu[[2]](#footnote-1), w normalnym trybie pracy, możemy spodziewać się ok. 3.8mA prądu pobieranego przez układ. Wartość tę, możemy zredukować do nawet 10uA, ograniczając częstotliwość próbkowania akcelerometru do 1.25Hz i wyłączając żyroskop. Ponadto, układ posiada tryb niskiego zużycia energii, pozwalające uśpić nieaktywny układ. Sam akcelerometr, pracuj w zakresie ±2g, ±4g, ±8g oraz ±16g. Dodatkowo, układ posiada tzw. Digital Motion Processor (DMP), czyli układ wspomagający przetwarzanie danych w kierunku wykrywania gestów. Wbudowane FIFO, pozwala na buforowanie danych. Zaletą akcelerometru, są programowalne przerwania oraz przerwanie “High-G”, wyzwalane w momencie przekroczenia zdefiniowanego przyspieszenia.

### LSM9DS1

Układ ten, nie różni się znacząco od MPU-6050. Zgodnie z dokumentacją[[3]](#footnote-2), jest on dodatkowo wyposażony w magnetometr. Znaczącą różnicą między tymi dwoma układami, jest pobierany przez nie prąd. W przypadku LSM9DS1, akcelerometr w trybie normalnym, pobiera około 600 uA. Niestety, wykorzystanie żyroskopu, dodaje kolejne 4mA. Żyroskop, posiada tryb niskiego zużycia energii, pozwalający ograniczyć zużycie do 1.9mA. Tym samym, chcąc użyć akcelerometru i żyroskopu w tym układzie, nie osiągnę mniej, niż 1.96 mA. Jest to zdecydowanie zbyt duża wartość dla mojego rozwiązania.

### LSM6DSOX

Ostatni z wybranych przeze mnie układów, to układ producenta ST. Jest to układ, dedykowany do rozwiązań o małym zużyciu energii. Zgodnie z dokumentacją[[4]](#footnote-3), jego zużycie energii to 0.55mA. Wartość ta, jest kilkukrotnie mniejsza od np. MPU-6050. Co więcej, układ posiada tryb Low Power, w którym zużycie energii można ograniczyć do nawet 4uA. LSM6DSOX, posiada również 9kB FIFO, po którego napełnieniu, wystawiane jest przerwanie. Dodatkową zaletą jest programowalna maszyna stanów. Potencjalnie, pozwoli mi ona na maksymalne ograniczenie zużycia energii, ponieważ mógłbym uruchamiać układ jedynie w przypadku wykrycia kolizji, a nie analizować dane na mikroprocesorze. LSM6DSOX, do komunikacji wykorzystuje SPI oraz I2C. Pozostałe jego parametry, są zbliżone do wcześniej opisywanych przeze mnie układów.

### Porównanie wybranych układów

|  | MPU-6050 | LSM9DS1 | LSM6DSOX |
| --- | --- | --- | --- |
| FIFO | 1kB | 128B | 9kB |
| Prąd pracy | 3.8 mA | 600 uA/ 4.6 mA | 0.55 mA |
| Low Power Mode | 10 - 110 uA | 1.9 - 3.1 mA | 4 - 20 uA |
| Zakres pracy akcelerometru | 2g, 4g, 8g, 16g | 2g, 4g, 8g, 16g | 2g, 4g, 8g, 16g |
| Zakres pracy żyroskopu | 250°/s, 500°/s, 1000°/s, 2000°/s | 245°/s, 500°/s, 2000°/s | 125°/s, 250°/s, 500°/s, 1000°/s, 2000°/s |
| Cena | ~27zł | ~82zł | ~20zł |

Po dokonaniu analizy, do wykonania mojego projektu wybrałem układ firmy ST, LSM6DSOX. W moim porównaniu, wypada on bezdyskusyjnie najlepiej. Bardzo niskie zużycie energii, duża ilość pamięci oraz niska cena sprawiają, że dla mojego przypadku, układ powinien sprawdzić się idealnie. Dodatkowo, duże nadzieje pokładam w funkcji programowalnej maszyny stanów.

## Mikrokontroler

### Analiza wymagań dla układu

Tak, jak w przypadku akcelerometru, układ powinien charakteryzować się przede wszystkim niskim zużyciem energii. Z oczywistych względów, musi on wspierać I2C oraz UART. Powinien też posiadać przynajmniej jeden pin(**lub więcej?**) obsługujący przerwanie zewnętrzne. Dodatkowym atutem, byłoby wsparcie bluetooth bez zewnętrznego modułu. Ponieważ projektowany przeze mnie układ będzie układem bardziej złożonym, przydatną może okazać się możliwość uruchomienia systemu operacyjnego czasu rzeczywistego. Nie jest to jednak konieczność.

### ATmega328p

Mikrokontroler ATmega328p, to jeden z najpopularniejszych układów na rynku. Jego niska cena i łatwość użycia, pozwala budować najróżniejsze aplikacje. Układ wspiera zewnętrzny zegad do 16MHz. Posiada on UART, I2C oraz SPI, co pozwala rozważać go w mojej pracy. Dodatkowo, w dokumentacji można znaleźć informację o dwóch przerwaniach zewnętrznych, obsługiwanych przez układ.[[5]](#footnote-4) Ponieważ jak wspomniałem, oczekuję od układu szybkiego zegara, do porównania poboru prądu wybrałem maksymalne wartości taktowania, czyli 16MHz. W tym przypadku, podczas pracy, układ pobiera 9-14mA które możemy zredukować do 2.8mA w trybie czuwania. W trybie zupełnego uśpienia mikrokontroler pobiera między 44, a 66uA.

### STM32F303K8

Układ STM32F303K8 oparty jest na rdzeniu ARM Cortex M4. Posiada on wbudowany układ RTC z funkcją wybudzania z kalendarzem. 11 timerów pozwala na budowanie złożonych aplikacji. Układ wyposażone jest w interfejs CAN, SPI, I2C oraz 3 UARTy. W przypadku rdzeni Cortex, każdy z pinów może być skonfigurowany jako przerwanie, co daje nam aż 60 przerwań.[[6]](#footnote-5) Mikroprocesor ten, może być taktowany zewnętrznym zegarem o częstotliwości do 32MHz, a więc dwukrotnie większym, niż w przypadku kontrolera ATmega. Jeśli chodzi o zużycie energii, to STM32F303 pobiera (w zależności od konfiguracji) 12.9 - 34 mA w trybie pracy oraz 0.93 - 18.57uA w trybie głębokiego uśpienia. Warto tutaj zaznaczyć, że wartości maksymalne dla trybu pracy, mikrokontroler osiąga, gdy wszystkie jego peryferia są uruchomione. W przypadku mojej pracy, dążyć będę do zminimalizowania ilości działających jednocześnie układów, a więc w porównaniu przyjmuję dolną granicę osiąganą przez układ.

### nRF52840

Układ nRF52840 jest najmocniejszym, spośród wybranych przeze mnie do porównania. Oparty na rdzeniu ARM Cortex M4, taktowany zegarem wewnętrznym 64MHz, posiada 1MB pamięci flash oraz 256kB RAM. Wspiera on I2C, 4 x SPI, 2 x UART i 3 x RTC. Obsługuje też Bluetooth, co wyróżnia go na tle dwóch pozostałych układów. Mimo, że tak posiada on tak dużo różnych peryferiów, jego zużycie energii w trybach głębokiego uśpienia, mieści się pomiędzy 0.4 uA, a 17.37 uA. Są to wartości zbliżone do mikrokontrolera STM32F303, opartego na tym samym rdzeniu. Pracujący układ, w zależności od konfiguracji, zużywa 2.8 - 6.3 mA. Dodatkowo, nRF52840 posiada wbudowane PMU (Power management unit), które automatycznie zarządza zużywaną energią, bez ingerencji użytkownika. Zaletą jest dla mnie również obszerna dokumentacja, dedykowane fora wsparcia oraz powszechność stosowania tego układu w urządzeniach.

### Porównanie wybranych układów

|  | ATmega328p | STM32F303K8 | nRF52840 |
| --- | --- | --- | --- |
| Flash | 32 kB | 64 kB | 1 MB |
| Prąd pracy | 9.2 - 14 mA (16MHz) | 12.9 - 34 mA | 2.8 - 6.3 mA |
| Prąd uśpienia | 40-66 μA | 0.93 - 18.57 μA | 0.4 - 17.37 μA |
| Liczniki | 2x8bit, 1x16bit | 1x32bit, 2x24bit, 8x16bit | 5x32bit |
| I2C | ✓ | ✓ | ✓ |
| UART | 1 | 3 | 2 |
| Przerwania zewnętrzne | 2 | 16 | 48 |
| System operacyjny | Brak | Free RTOS | Zephyr RTOS |

Z powyższego porównania widać, że układ ATmega328p w sposób znaczący odstaje od pozostałych dwóch. Mimo, że jest on układem najprostszym w obsłudze i posiadającym wiele przyjaznych bibliotek, nie może zostać wykorzystany w mojej pracy ze względu na duże zużycie energii. Problematyczna może okazać się też mała liczba liczników. W tym wypadku, najlepiej wypada układ firmy ST, jednak jego zużycie energii w trakcie pracy jest bardzo wysokie. Z tego powodu, zdecydowałem się wykorzystać układ nRF58240

## LTE/GSM

### Analiza wymagań dla układu

LTE/GSM, a właściwie modem LTE/GSM to układ pozwalający na komunikację poprzez sieci komórkowe 4G lub GSM. Zastosowanie tego typu komunikacji, powinno zapewnić mi dostęp do sieci niemal niezależnie od miejsca w którym znajdować będzie się urządzenie. Dodatkowo, zastosowanie Polte, czyli określania przybliżonej pozycji na podstawie LTE, pozwoli mi zwiększyć skuteczność urządzenia. Ponieważ procedura autentykacji do sieci jest czasochłonna i wymaga radia działającego z dużą mocą, spodziewałem się, że będzie to układ pochłaniający najwięcej energii. Dodatkowo, minimalne wymaganie to możliwość obsługi SMS, jednak jeśli to możliwe, chciałbym również, aby układ miał możliwość wysyłania zapytań HTTP lub MQTT. Pozwoli mi to na porównanie tych technologii pod kątem zużycia energii.

### Porównanie popularnych układów

SIM800L

Jest to jeden z najpopularniejszych układów do amatorskich rozwiązań. Jego największymi zaletami jest cena i mnogość dostępnych bibliotek, głównie na układy ATmega328p. Zastosowanie go z innymi układami nie jest jednak problemem, ponieważ działa on w oparciu o tak zwane komendy “AT”, wysyłane przez UART. Pobór prądu tego układu, mieści się pomiędzy 150, a 450mA podczas transmisji danych oraz 50-60uA w trybie Power Down (wyłączony). Co ważne, podczas uruchamiania, układ pobiera do 2A prądu, co jest wartością bardzo dużą, w przypadku stosowania układów zasilanych bateryjnie, ponieważ do tak dużego prądu, musi być dostosowany całe urządzenie. Niestety, układ ten działa z przestarzałymi już sieciami GSM i nie obsługuje zapytań HTTP lub MQTT.

Quectel EC20 R2.1

Jest to jeden z łatwo dostępnych na rynku układów, działających w oparciu o technologię LTE. Komunikacja z nim, przebiega przy użyciu komend AT. Układ, wspiera możliwość wysyłania zapytań HTTP, co jest jego plusem w przypadku mojej pracy. Pobór prądu tego układu osiąga 300mA, jednak w trybie Power Off osiąga nawet 11uA. Jest to więc wartość ponad pięciokrotnie mniejsza, niż w przypadku SIM800L. Jest to ważne, ponieważ moje urządzenie większość czasu, pracować będzie w trybie całkowitego uśpienia.

SKY66430

Jest to produkt firmy Skyworks, specjalizującej się w tworzeniu półprzewodników do układów radiowych i komunikacji mobilnej. Wybrany przeze mnie produkt, wspiera zarówno komunikację LTE jak i NB-IoT. Podobnie jak powyższe układy, obsługiwany jest przy użyciu komend AT. Modem ten, pozwala nie tylko na wysyłanie SMS, ale również zapytań HTTP i MQTT. Zużycie energii podczas transmisji oscyluje około 150-200mA, jednak tym co go wyróżnia, jest wyjątkowo niskie zużycie energii w trybie Power Off, wynoszące 1uA. Jest to wartość wielokrotnie mniejsza od tego, co mogą osiągnąć poprzednie układy, co czyni go najlepszym z kandydatów do mojej pracy.

### Porównanie wybranych układów

|  | SIM800L | EC20 R2.1 | SKY66430 |
| --- | --- | --- | --- |
| Prąd pracy | 150-450 mA | 300 mA | 150-200 mA |
| Power off | 50-60 μA | 11 μA | 1 μA |
| SMS | ✓ | ✓ | ✓ |
| HTTP | ✕ | ✓ | ✓ |
| MQTT | ✕ | ✕ | ✓ |

Na podstawie powyższego porównania, zdecydowałem się wybrać układ SKY66430. Dodatkowym argumentem, przemawiającym za jego użyciem jest fakt, że posiadam płytkę rozwojową tego układu, która znacznie ułatwi mi testowanie urządzenia w początkowych fazach projektu.

## GPS

GPS, czyli Global Positioning System, to system nawigacji oparty na lokalizacji satelitarnej. Jego zadaniem, jest dostarczenie użytkownikom informacji o ich położeniu. W przypadku proponowanego przeze mnie rozwiązania, zastosowanie modułu GPS, jest kluczowe dla osiągnięcia założonego celu.

### Analiza wymagań dla układu

Zastosowany przeze mnie układ GPS, powinien charakteryzować się możliwie niskim zużyciem energii. Niestety, ze względu na wymagający proces określania lokalizacji, prawdopodobnie będzie to najbardziej wymagający układ, pod kątem zużycia energii. Dobrze więc, by istniała możliwość swobodnego wyłączania modułu. Dobrze, gdyby układ komunikował się z mikrokontrolerem przy użyciu magistrali I2C. Dzięki temu, będę mógł swobodnie połączyć go równolegle do pozostałych komponentów. Wystarczająca dokładność układu, to około 50 metrów. Jest to odległość dostatecznie nieduża, aby usłyszeć wołanie, lub zobaczyć poszkodowaną osobę.

### Beitian BN-220

Jest to jeden z najtańszych układów dostępnych na rynku. Z układem komunikuje się przy użyciu UART, przesyłając wiadomości NMEA, czyli zadania, zapisane kodem ASCII. Zgodnie z dokumentacją[[7]](#footnote-6), układ zużywa około 50mA i jest zasilany napięciem 5V.

### Mediatek MT3339

Kolejny układ, to układ produkcji Mediatek. Zasilany napięciem 3.3V, pobiera około 3mA prądu[[8]](#footnote-7). Tak jak BN-220, komunikuje się on przy użyciu wiadomości NMEA, jednak wykorzystując I2C. Dodatkowo, układ ten, posiada wewnętrzny zegar czasu rzeczywistego, pozwalający na wybudzanie układu ze stanu uśpienia. Kolejnym z ciekawych rozwiązań, jest EPO™, czyli predykcja położenia satelit, na podstawie czasu. Dzięki temu rozwiązaniu, można znacząco skrócić czas określania lokalizacji. Dokładność modułu, to około 3m, a więc zdecydowanie więcej, niż zakładałem.

### Mediatek MT3333

Ostatni z układów, to układ zbliżony do poprzedniego. Od poprzednika, odróżnia go nieco większe zużycie energii, wynoszące od 5 mA do 19 mA[[9]](#footnote-8). On również, wykorzystując wiadomości NMEA, komunikuje się wykorzystując magistralę I2C.

### Wybór układu

Na podstawie powyższego porównania, zdecydowałem, że w mojej pracy wykorzystam układ Mediatek MT3333. Ma on bardzo dobrą dokładność, przy jednoczesnym zachowaniu niskiego zużycia energii. Decydujący okazał się fakt, że do wybranego układu posiadam zarówno noty aplikacyjne, jak i obszerną dokumentację. Pomimo tego, że MT3339 jest układem zbliżonym, mogło okazać się, że drobne różnice w obsłudze układu, do którego nie mam dostatecznie dobrych dokumentacji, uniemożliwiłyby mi ukończenie projektu.

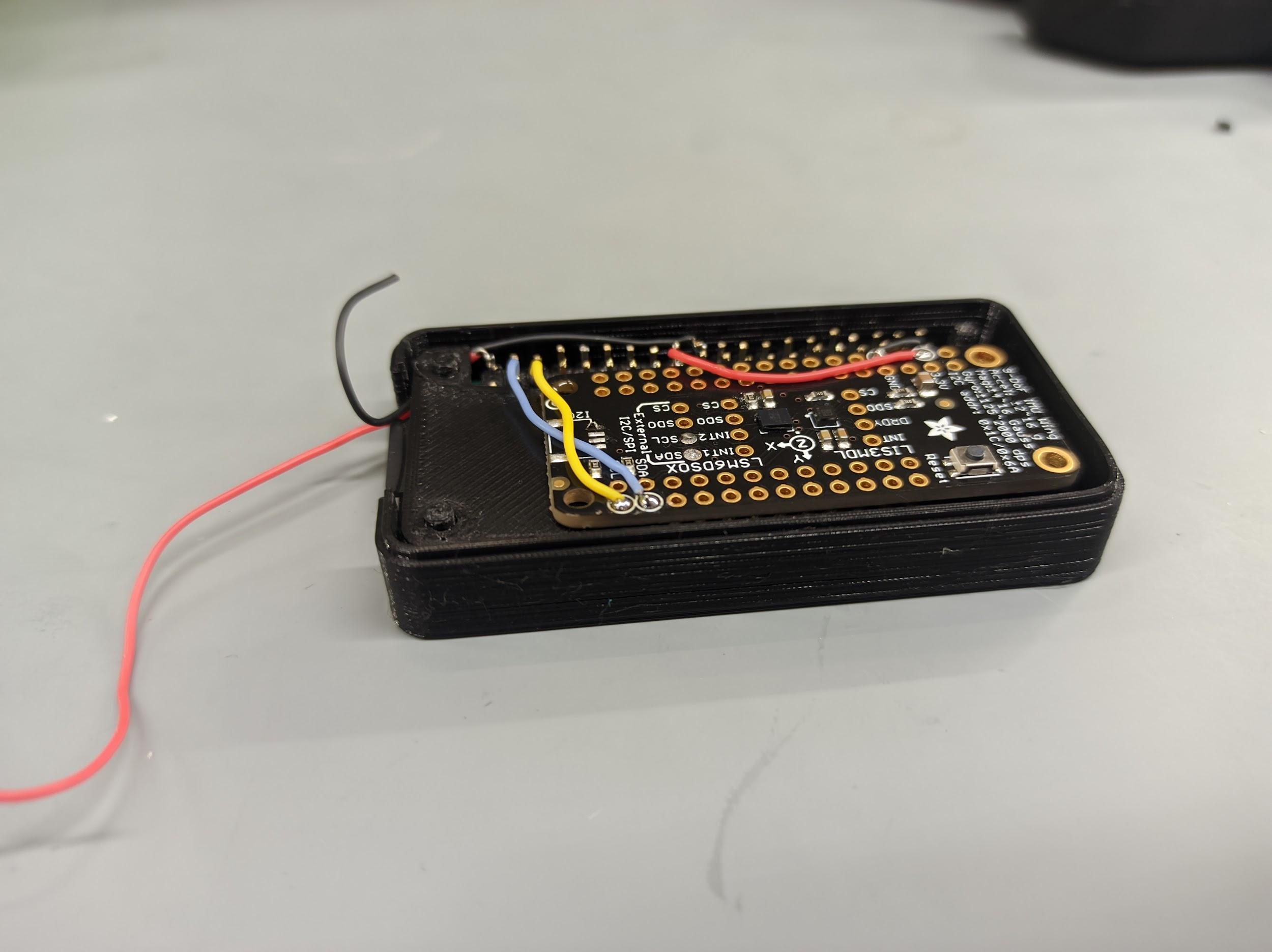
# Zaplanowanie prototypu urządzenia wykrywającego upadek na rowerze

## Problematyka

Głównym zagadnieniem mojej pracy jest stworzenie kompletnego urządzenia, które wykryje niebezpieczne zdarzenie i da o tym informację. Z tego powodu, w mojej pracy, najwięcej uwagi poświęciłem wyborowi i obsłudze akcelerometru. W miarę moich możliwości, przeprowadziłem wstępne testy mające pomóc mi stworzyć odpowiednie algorytmy. To właśnie one, są najważniejszym elementem urządzenia. Algorytmy powinny być nie tylko niezawodne i szybkie, ale jednocześnie pozwolić na jak najmniejsze zużycie energii, niezbędnej dla działania układu. W trakcie tworzenia urządzenia, rozważałem różne podejścia do każdego z komponentów, w celu znalezienia optymalnego.

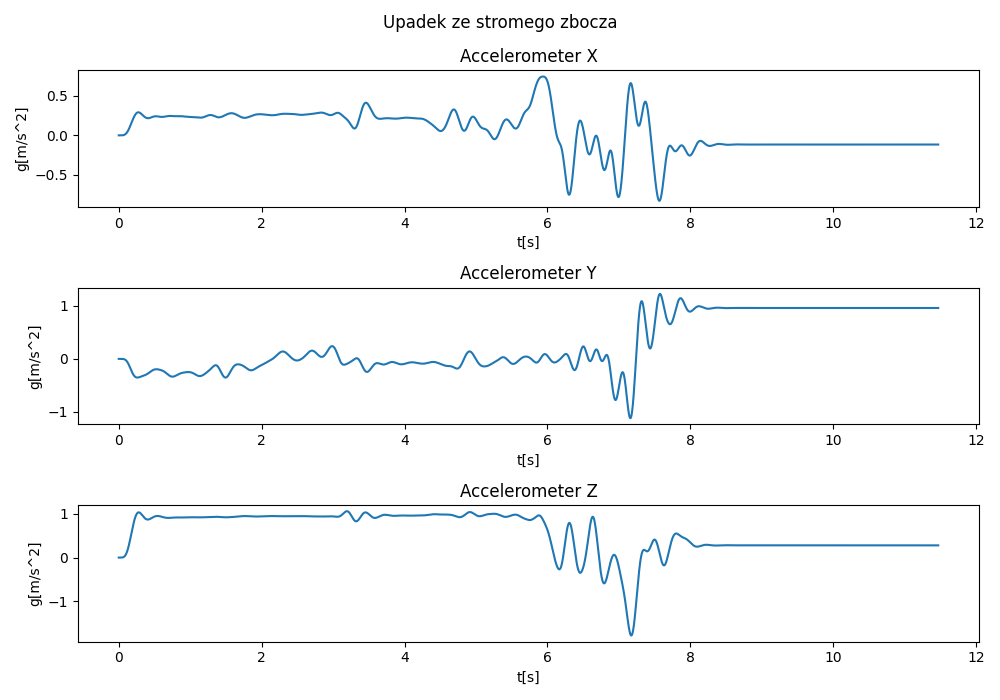
## Algorytm wykrywania kolizji

Część praktyczną, rozpocząłem od przeprowadzenia podstawowych eksperymentów, mających pokazać, jakich przyspieszeń mogę spodziewać się w przypadku rzeczywistego zderzenia na rowerze. Testy, rozpocząłem od stworzenia urządzenia, mającego zbierać surowe dane z akcelerometru. Ze względu na łatwość i szybkość implementacji, wykorzystałem platformę RaspberryPi Zero W, z systemem operacyjnym Raspbian. Korzystając z gotowych bibliotek, napisałem skrypt który po uruchomieniu tworzy plik tekstowy, do którego zapisuje surowe dane pobrane z akcelerometru. Sam akcelerometr, ustawiony był na częstotliwość próbkowania 416 Hz i skalę 8g. Ze względów czysto praktycznych tj. w celu zabezpieczenia płytek przed uszkodzeniem, wykonałem model 3D prostej obudowy, którą wydrukowałem na drukarce 3D.

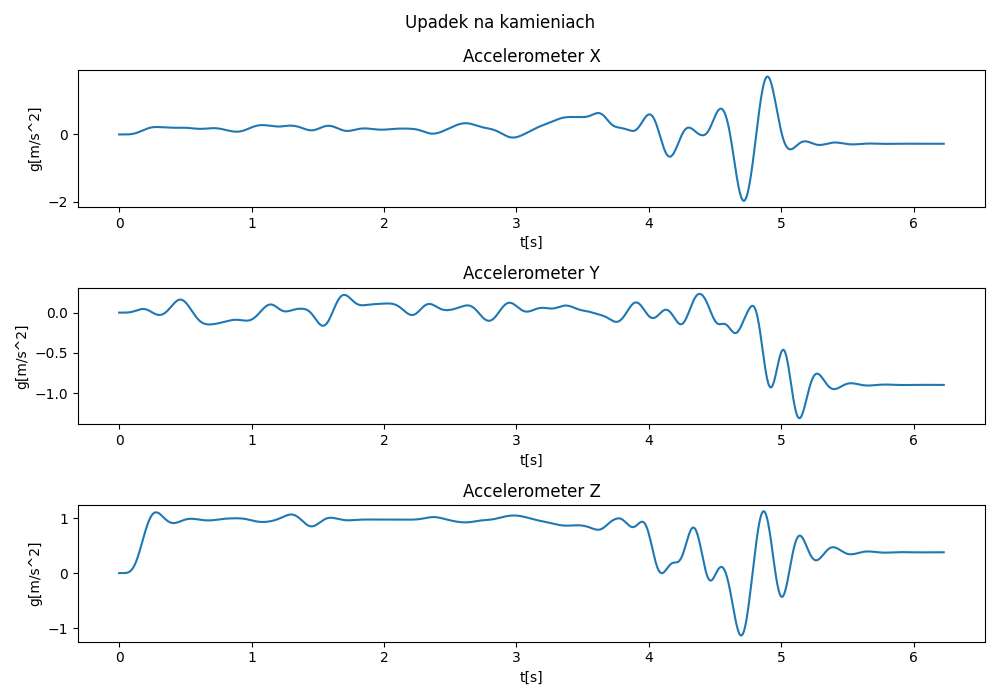


Cały układ, zasilany był przy użyciu powerbanku, zamontowanego do siodełka. Ze względu na własne bezpieczeństwo, testy przeprowadzałem na rowerze bez kierowcy. Sam układ, zamontowałem na rurze podsiodłowej. Ponieważ jest to prawie centralny punkt roweru, spodziewałem się, że przyspieszenia w przypadku silnego wyrzucenia koła w powietrze, będą mniejsze. Eksperyment przeprowadzony był przez rozpędzenie roweru do prędkości ok. 10km/h i kolejno: Zderzenie roweru z drzewem, upadek na kamieniach, upadek ze stromej góry.

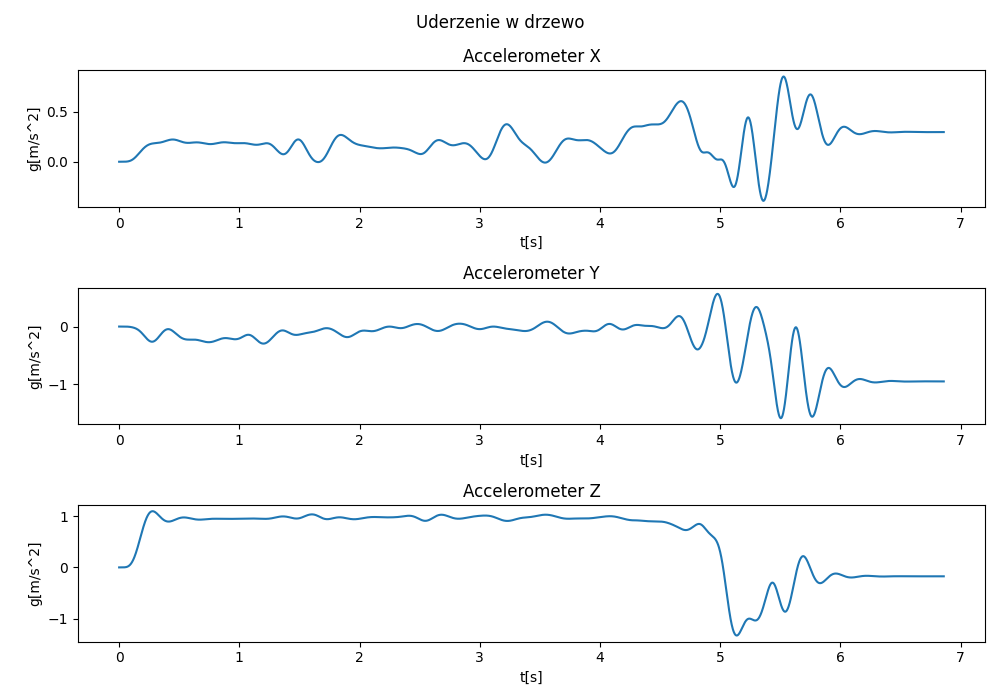
Zbierane dane przepuściłem przez dolnoprzepustowy filtr, aby usunąć szum i nierzeczywiste skoki przyspieszenia. Poniżej, prezentuję zebrane dane w postaci wykresów dla każdej z osi.



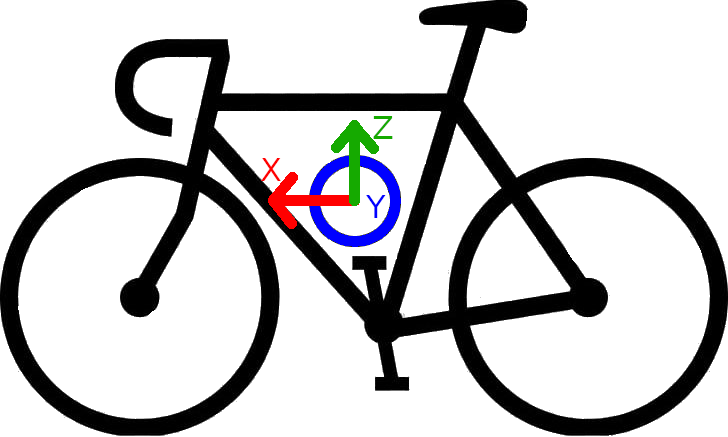
Wykres 1. Upadek ze stromej góry. V0 ~ 1m/s



Wykres 2. Upadek na kamieniach. V0 ~ 1m/s

Wykres 3. Uderzenie w drzewo. V0 ~ 1m/s

Na powyższych wykresach, bardzo dobrze widać moment uderzenia w każdym z przypadków. Bez trudu jestem również w stanie powiązać przebiegi z konkretnym momentem na nagraniu.



Rysunek 1. Orientacja urządzenia na rowerze.

Podczas każdego z testów, rower z przymocowanym urządzeniem, rozpędziłem do ok. 1m/s. Zgodnie z moimi przewidywaniami, w większości przypadków, przyspieszenia układu sięgały ok. 1g. Należy również zwrócić uwagę na fakt, że podczas upadku rower często rotował, co ma bezpośredni wpływ na zwroty przyspieszenia.

Na wykresie 1. w szóstej sekundzie widać uderzenie przedniego koła o zbocze. Jest to silny pik na osi X sięgający ok. 0.5g. W około 7 sekundzie rower po uderzeniu w ziemię obrócił się wokół osi Y, zmieniając znak osi Y. W tej pozycji zaczął wyhamowywać, co widać w postaci ujemnego przyspieszenia na osi X. Z wykresu można również odczytać, że rower po zdarzeniu, zatrzymał się przewrócony na bok. Świadczy o tym przyspieszenie 1g na osi Y.

Wykres drugi, to obraz upadku na kamieniach. Odzwierciedla sytuację, gdy rowerzysta straci kontrolę nad rowerem podczas przejazdu po korzeniach i kamieniach. Wyróżnikiem tego zdarzenia, są silne oscylacje na osi X i Z, przy stosunkowo stabilnej osi Y. Przedstawia to sytuację gdy tylne koło uderzając w korzeń, zostało przerzucone nad przednim, a następnie rower uderzył w ziemię z dużą siłą. Tak jak w poprzednim przypadku, rower zatrzymał się przechylony na bok.

Ostatni z wykresów przedstawia bezpośrednie uderzenie roweru w drzewo. W tym przypadku, to oś Z jest bardzo stabilna w porównaniu do pozostałych, ponieważ rower jechał po płaskiej powierzchni do momentu zderzenia. W około piątej sekundzie, nastąpiło uderzenie w drzewo, a rower obrócił się o 180 stopni wokół osi Y. Przyspieszenie w osi X okazało się znacznie mniejsze niż oczekiwałem. Wynika to jednak najprawdopodobniej z faktu, że rower nie zatrzymał się na drzewie, a odbił od niego mostkiem i przekoziołkował dalej.

Przeprowadzone testy, pozwoliły mi sprawdzić, jakie przeciążenia działają na rower w trakcie wypadku. Jednym z pierwszych wniosków, które nasunęły się już w momencie pierwszej analizy surowych danych, jest to, że ustawiłem zbyt małą częstotliwość próbkowania. Test, powinienem był przeprowadzić w możliwie najbardziej dokładnych ustawieniach. W przypadku użycia raspberry, magistrala I2C pozwala na transmisję 12,5kB/s. Ponieważ jeden zestaw zawiera 12 bajtów, częstotliwość próbkowania mogłaby wynosić ponad 1kHz, a więc znacznie więcej niż ustalone przeze mnie 416Hz. Podobnie w przypadku skali akcelerometru. Ustalenie jej na 16g, mogłoby dokładniej pokazać szczytowe wartości, ponieważ pojedyncze próbki były ograniczone na wartości 8g. Występowanie pojedynczych próbek, znacząco różnych od pozostałych, może też świadczyć o niedokładności układu. Tutaj nasuwa się drugi wniosek, a więc konieczność stosowania filtrów dolnoprzepustowych. Filtry te pozwolą wyciąć potencjalnie wadliwe próbki, a jednocześnie usuną szum z sygnału.

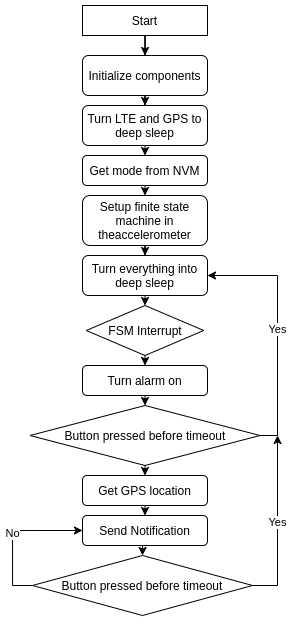
## Algorytm działania urządzenia

Głównym problemem w przypadku tak tworzenia tak złożonego urządzenia, jest optymalizacja jego działania pod kątem zużycia energii. W proponowanym przeze mnie rozwiązaniu, nie jest wymagane ciągłe śledzenie lokalizacji. Z tego powodu, już na początku możemy wyłączyć niemal wszystkie komponenty i peryferia. Daje nam to znaczne oszczędności energii.

[Tu powinno być jakieś porównanie zużycia z otii. Wszystko w deepsleep kontra część nie]

Zgodnie z poniższym diagramem, na początku inicjalizuję niezbędne komponenty, czyli I2C, GPIO przycisku oraz akcelerometr. Następnie upewniam się, że LTE i GPS są wyłączone, aby zagwarantować ich znikome zużycie energii. W kolejnym kroku, pobieram ostatnio wybrany tryb pracy z NVM. Takie podejście, pozwala mi zapewnić stan ustalony nawet po nieoczekiwanym restarcie urządzenia z powodu np. rozładowania. Znając więc stan urządzenia, ustawiam maszynę stanów układu LSM6DSOX, zgodnie z opisanym wcześniej algorytmem. Następnie, przełączam wszystkie pozostałe komponenty w tryb głębokiego uśpienia, oczekując na zdefiniowane zdarzenie.

W sytuacji wystąpienia zdarzenia, wyemitowany zostanie głośny dźwięk o około 93 dB[obrazek poniżej będzie]. W tym samym momencie, uruchomiony zostaje timer, oczekujący 30 sekund na kilkusekundowe przytrzymanie przycisku. W przypadku naciśnięcia przypadku, alarm zostanie przerwany, a urządzenie ponownie przejdzie w stan oczekiwania na zdarzenie. W przypadku, gdy przerwanie z przycisku nie zostanie wyzwolone na czas, urządzenie uruchomi moduł GPS celem określenia swojej lokalizacji, a następnie okresowo będzie wysyłać powiadomienie przy użyciu LTE. Powiadomienie będzie wysyłane co określony interwał czasu, dopóki nie zostanie wyłączone przyciskiem lub bateria nie ulegnie rozładowaniu.

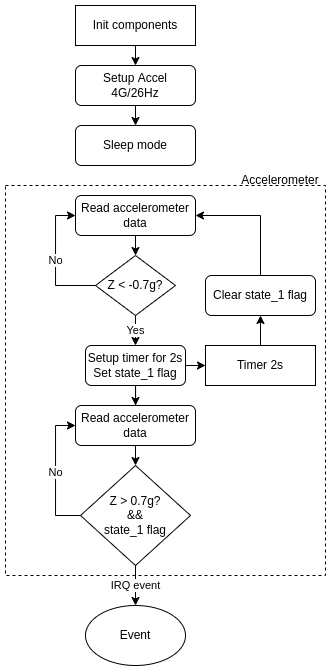
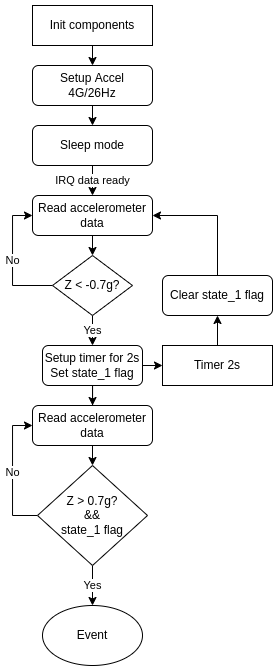


# Wykonanie prototypu urządzenia

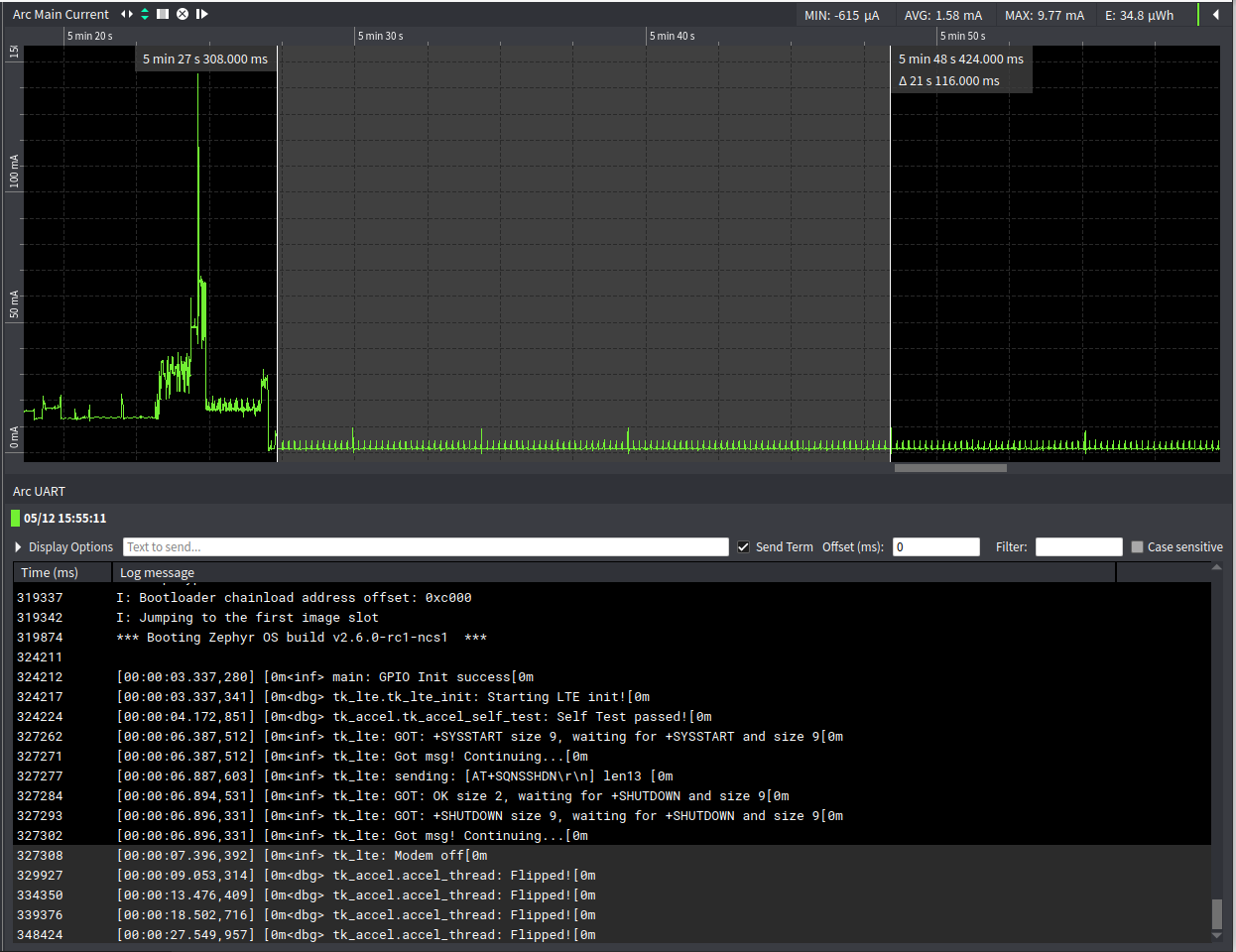
## Akcelerometr

### Wybór metody analizy danych

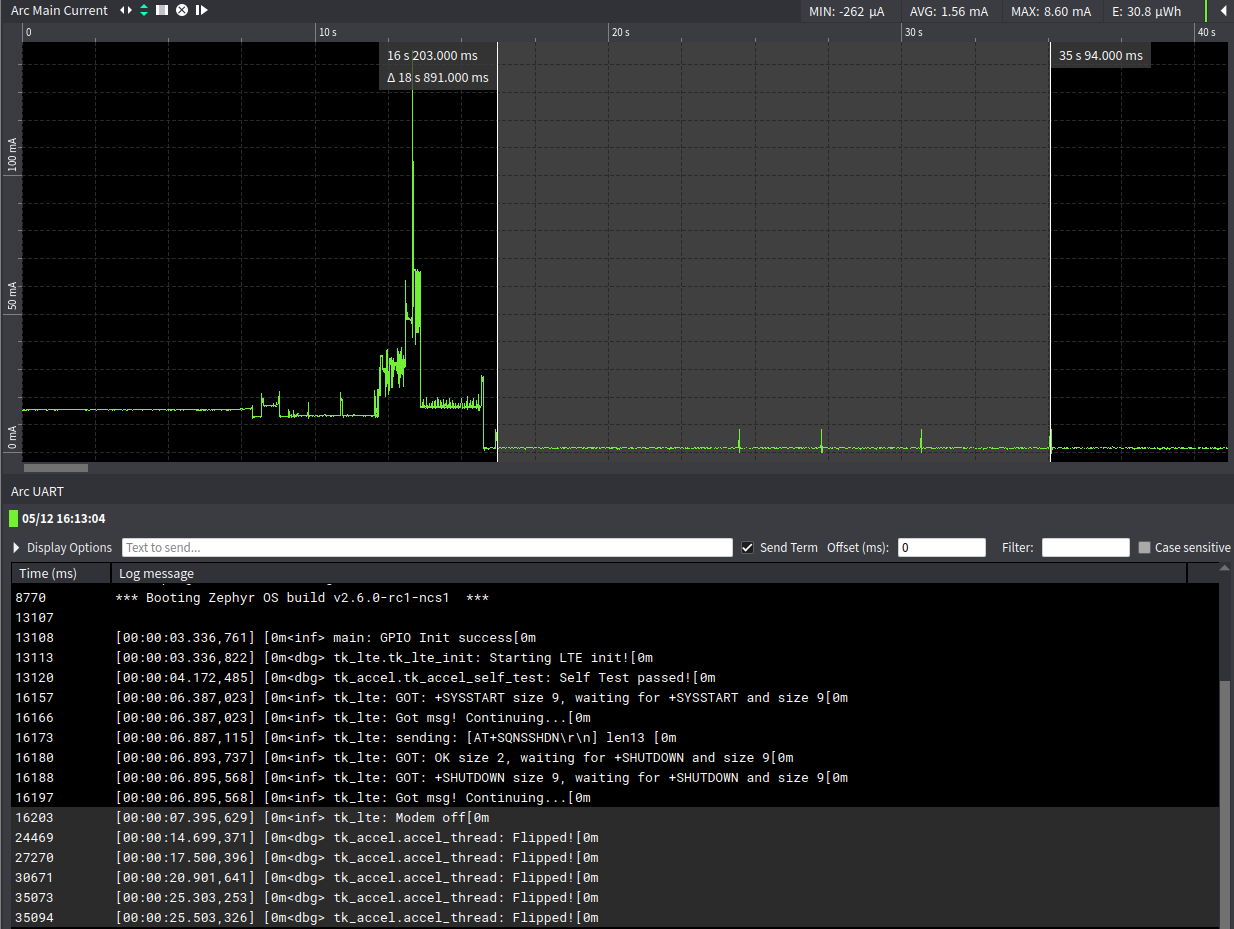
Na początku swojej pracy, postanowiłem sprawdzić zużycie energii układu dla klasycznej metody oraz wykorzystując wbudowaną w układ maszynę stanów. W tym celu, skonfigurowałem prosty algorytm, wykrywający obrót urządzenia w osi Z. Akcelerometr, skonfigurowałem w trybie niskiego zużycia energii, a następnie wprowadziłem mikroprocesor w tryb uśpienia. W tym trybie, oczekuje on na przerwanie, na pinie akcelerometru. Ten zaś, uruchamia je, gdy dane każdej z osi są gotowe. Po wyzwoleniu przerwania, ustawiamy flagę “data\_ready”, a następnie czytamy dane z akcelerometru. Porównując daną z osi Z ze stałą, próbujemy wykryć odwrócenie układu o 180 stopni. Gdy takowe nastąpi, uruchamiamy licznik, który po dwóch sekundach wyczyści flagę, świadczącą o wydarzeniu. Gdy w tym czasie odczytamy dane o przeciwnym zwrocie, uznajemy, że wykryliśmy wydarzenie i informujemy o tym przez wypisanie informacji na UART.



Na poniższym zrzucie z programu otii, widać wyraźnie inicjalizację urządzenia i pik związany z uruchomieniem modemu LTE. Widać też moment, w którym procesor jedynie odczytuje, przychodzące nieustannie dane. Widać to, w postaci drobnych pików na przebiegu. Wyraźniejsze z nich, to momenty, w których wykonałem obrót urządzenia. Skok wynika głównie z wywołania funkcji logującej, jednak wyraźnie zaznacza moment przejścia przez programową maszynę stanów. W przypadku tego podejścia, średnie zużycie wynosi 1.58mA co nie jest wynikiem złym. Warto jednak zauważyć, że powyższa maszyna stanów jest bardzo prosta, a jej znacząca rozbudowa, będzie wiązać się z szybkim wzrostem energii. Jej minusem, jest również ilość przerwań generowanych przez akcelerometr. Każdy odczyt, zajmuje linię danych I2C, uniemożliwiając komunikację z innymi komponentami. Jednak co ważniejsze, przerwanie zajmuje czas procesora, który przy dużej ilości niezależnych podzespołów, jest kluczowy.



W drugim przypadku, skonfigurowałem maszynę stanów, wbudowaną w akcelerometr. Tym razem, średnie zużycie wynosiło 1.56mA. Jest to aż 200 mikroamperów różnicy, przy tak prostej funkcjonalności. Jak pokażę w dalszej części pracy, wartość ta jest znacząca w skali osiągniętej przeze mnie wartości zużycia prądu w trybie uśpienia. Należy również zwrócić uwagę, na brak małych pików prądu, które występowały w poprzednim podejściu. Tym razem, akcelerometr działa zupełnie niezależnie, co obrazuje poniższy diagram. Widać na nim, że układ mikroprocesora cały czas znajduje się w trybie uśpienia, co ma odzwierciedlenie również na przebiegu pobieranego przez układ prądu. Wybudzany jest on jedynie przerwaniem, pochodzącym od akcelerometru.



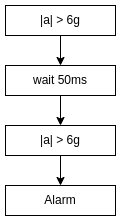
Powyższe rozważanie, potwierdza sens zastosowania tak złożonego akcelerometru. Pokazuje też, że nawet mając wyjątkowo energooszczędny układ, możemy zawyżyć zużycie energii nieumiejętnym jego wykorzystaniem. Dodatkowo, STMelectronics przygotowało zestaw narzędzi, ułatwiających budowę urządzeń w oparciu o ich układy. Przykładem jest program *Unico*, pozwalający programować akcelerometry w oparciu o płytki rozwojowe. W mojej pracy, wykorzystywać będę zestaw STEVAL-MKI109V3 + STEVAL-MKI197V1. Pierwszy z układów, to płytka rozwojowa z nakładką, pozwalająca programować oraz debugować układy ST. Drugi, to nakładka z obudowanym akcelerometrem LSM6DSOX. Program *Unico*, pozwala projektować maszyny stanów w oparciu o interfejs graficzny. Jest to znaczące ułatwienie, ponieważ pozwala on zapisać konfigurację urządzenia, a następnie wygenerować z niej tablicę struktur w języku C. Dzięki temu, pisząc funkcję obsługującą takie struktury, mogłem w łatwy i szybki sposób testować różne konfiguracje akcelerometru, unikając żmudnego, ręcznego wpisywania kolejnych rejestrów. Niestety, stosowane przez ST skróty nazw rejestrów, nie są zbyt intuicyjne. Z tego powodu, mimo użycia tak zaawansowanego programu, i tak byłem zmuszony przeszukiwać dokumentacje i fora, aby rozszyfrować niektóre skróty.

### Implementacja algorytmu wykrywania zdarzenia

Na podstawie wcześniejszych rozważań, zdecydowałem, że optymalnym podejściem będzie skorzystanie z wbudowanej w akcelerometr maszyny stanów. Pozwoli to zaoszczędzić znaczne ilości energii, bez utraty skuteczności.

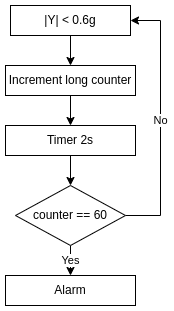
Na podstawie przeprowadzonych testów, zdecydowałem się na implementację czterech, niezależnych maszyn stanów.

**Wykrywanie przeciążenia w dowolnej osi**



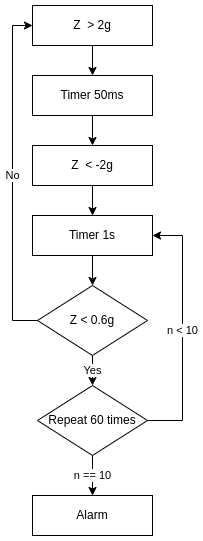
Pierwsza, najprostsza z maszyn stanów, wykrywa znaczne przyspieszenie w dowolnej osi. Odpowiada ona na występowanie krótkich skoków o bardzo dużej wartości, w trakcie zdarzenia. Gdy przyspieszenie jest większe niż 6g, ustawiam wbudowany licznik na 50ms. Po tym czasie, ponownie sprawdzam wartość przyspieszenia. Jeśli jest ona większa, niż 6g, wyzwolony zostaje alarm. Takie podejście, pozwala na usunięcie pojedynczych, fałszywych próbek. Jednocześnie, duża wartość przyspieszenia pozwoli uniknąć alarmów podczas zwyczajnej jazdy.

**Wykrywanie przewróconego roweru**

****

Dwie kolejne maszyny odpowiadają za monitorowanie nachylenia roweru. Podczas jazdy, rower skręca i poddawany jest wielu przeciążeniom również na osi prostopadłej do kierunku jazdy. Niemniej jednak, w znakomitej większości przypadków, rower po upadku pozostanie przechylony na jedną ze stron. Wybrane maszyny, wykorzystują wbudowany licznik. Co dwie sekundy, sprawdzane jest odchylenie roweru. Jeśli przez dwie minuty rower jest przechylony, wyzwolony zostaje alarm.

**Wykrywanie uderzenia z przewrotu**



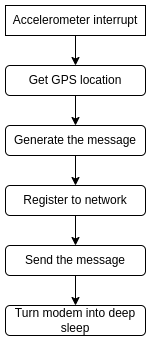
Ostatnia z przykładowych maszyn stanów, wykrywa uderzenie w osi Z, czyli np. najechanie na korzeń, a następnie przez 60 sekund sprawdza, czy rower wrócił do wyjściowej pozycji, tj. pozycji zwykłej jazdy. W przypadku, gdy przez 60 sekund rower nie zostanie przejdzie przez pozycję jazdy (nie powróci do pionu), uruchomiony zostanie alarm. Maszyna ta, okazuje się być najdokładniejszą z prezentowanych przeze mnie. Wynika to z faktu, że oś Z jest osią najbardziej stabilną w trakcie jazdy. Pozostałe maszyny, stanowią więc uzupełnienie ostatniej.

## LTE

### Implementacja powiadamiania o zdarzeniu

Do mojej pracy, zdecydowałem się na wysłanie prostego powiadomienia tekstowego, przy użyciu SMS. Za tym rozwiązaniem, przemawia stabilność i niezawodność sieci komórkowych, mających obecnie niemal całkowite pokrycie, nawet w górach. Wysłanie SMS, jest również zgodne z uniezależnieniem układu od zewnętrznych komponentów, ponieważ nie wymaga ono żadnych aplikacji. W łatwy sposób, mogę również wysłać tzw. pinezkę map google w postaci URL, co znacząco ułatwi odbiorcy interpretację lokalizacji. Nie potrzeba więc do tego żadnej aplikacji.

Dodatkowo, w swoim urządzeniu, zdecydowałem się wykorzystać również bluetooth. Na ten moment jego działanie jest bardzo proste. Podczas uruchomienia urządzenia, inicjalizuję stos oraz tworzę charakterystykę odbierającą dane. Gdy użytkownik, wpisze do niej numer telefonu, urządzenie zapisuje go i dodaje do krótkiej kolejki. Dzięki temu, w sposób asynchroniczny, modem chcąc wysłać powiadomienie, sprawdzi aktualność numeru i wyśle wiadomość do odpowiedniego odbiorcy.



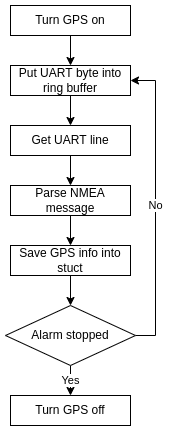
Przetestowałem również bardziej popularne podejście, wykorzystujące zapytania HTTP. Stworzyłem proste gniazdo TCP w JavaScript, a następnie wykorzystując gotowe biblioteki, stworzyłem tak zwanego bota do aplikacji Discord. Jest to platforma typu open source, na której użytkownicy mogą tworzyć zarówno kanały do komunikacji głosowej jak i tekstowej. Bot, to wirtualny użytkownik który wykonuje zdefiniowane w kodzie akcje. Mogą one obsługiwać specjalne funkcje serwera, lub monitorować zachowanie użytkowników. W moim przypadku, bot Filip (bo boty również mają imiona, aby można było je łatwo rozróżnić), mógłby wysyłać powiadomienie z lokalizacją, do wszystkich użytkowników serwera. Ponieważ platforma posiada bardzo dobrą aplikację na urządzenia mobilne, powiadomienie mogłoby szybko być odebrane przez jednego z użytkowników. Ostatecznie, na tym etapie pracy, zrezygnowałem z tego rozwiązania, aby uniezależnić urządzenie od dodatkowego kodu obsługującego bota. Jest to jednak rozwiązanie bardzo ciekawe i wydawałoby się, również przyszłościowe. Szczególnie biorąc pod uwagę szybko rosnącą popularność platformy i jej otwarty kod źródłowy.

### Analiza zużycia energii układu

<TODO: Ile prądu pobiera rejestracja i wysłanie SMS>

## GPS

Zasada działania GPS okazała się stosunkowo prosta. Zgodnie z dokumentacją, należy w odpowiedniej kolejności zasilić przetwornice, aby GPS zaczął określać swoje położenie. GPS, odsyła wiadomości zgodne z protokołem NMEA. Są to zdania, zapisane kodem ASCII. Z tego powodu, wykonałem fragmentowanie kodu, zgodnie z protokołem. Przetworzone informację, zapisywane są w strukturze, którą LTE może otrzymać stosując getter. Po zapisaniu danych, GPS zostaje wyłączony



### Analiza zużycia energii układu

<TODO: Zobaczyć ile prądu ciągnie jak łapie fixa, a ile jak go trzyma. Czy opłaca się wyłączać Giepsa>

# Testowanie urządzenia

## Metody testowania

<Nie wiem. Co, mam jechać w góry, wywalić się i zobaczyć czy zadziałało?>

< Alternatywnie, mogę jechać w kilka różnych miejsc (miasto, góry, las, dolina), spiąć to jakimś uartem i zobaczyć jak dokładnie wskazało miejsce. >

# Możliwości rozwoju urządzenia

Kierunków rozwoju urządzenia jest zdecydowanie wiele. Jednym z nich, jest ulepszenie samych algorytmów. Akcelerometr posiada rdzeń nauczania maszynowego, który można wykorzystać do stworzenia bardziej precyzyjnych sekwencji. Wymaga to jednak czasochłonnego zbierania danych i ich analizy.

Kolejnym z elementów, jest rozbudowa logiki urządzenia. Bluetooth można wykorzystać do połączenia urządzenia z telefonem. Z jego poziomu mógłbym sterować trybami pracy urządzenia i dostosowywać algorytmy do trybu jazdy. Również wspomniane przeze mnie powiadomienia przez platformę Discord, wydają się być przyszłościowym rozwiązaniem.

# Podsumowanie pracy

# Zakończenie i oklaski

< clap clap >

<Stół roboczy >

Prezka o FSM LSM, zużycia prądu, usecase itp.

https://www.stmicroelectronics.com.cn/content/dam/AME/2019/developers-conference-2019/presentations/STDevCon19\_3.1\_LSM6DSOX.pdf

1. <https://www.specialized.com/pl/pl/stories/angi> [↑](#footnote-ref-0)
2. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf [↑](#footnote-ref-1)
3. https://www.st.com/resource/en/datasheet/lsm9ds1.pdf [↑](#footnote-ref-2)
4. https://www.st.com/resource/en/datasheet/lsm6dsox.pdf [↑](#footnote-ref-3)
5. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\_Datasheet.pdf [↑](#footnote-ref-4)
6. https://www.st.com/resource/en/datasheet/stm32f303k8.pdf [↑](#footnote-ref-5)
7. https://files.banggood.com/2016/11/BN-220%20GPS+Antenna%20datasheet.pdf [↑](#footnote-ref-6)
8. https://labs.mediatek.com/en/chipset/MT3339#Features [↑](#footnote-ref-7)
9. https://labs.mediatek.com/en/chipset/MT3333 [↑](#footnote-ref-8)