



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Mateusz Prusaczyk

Nr albumu s32144

Projekt i implementacja cyfrowej czarnej skrzynki rowerowej

Praca magisterska napisana

pod kierunkiem:

dr inż. Mariusz Trzaska, prof. PJATK

Warszawa Lipiec 2026

Streszczenie

Praca dotyczy projektu i implementacji cyfrowej czarnej skrzynki rowerowej - autonomicznego urządzenia wbudowanego, łączącego funkcje detekcji upadku, telematyki nawigacyjnej oraz alarmowania ratunkowego przez sieć GSM. Wzrastająca liczba użytkowników rowerów w przestrzeni miejskiej i pozamiejskiej generuje zapotrzebowanie na rozwiązania zwiększające bezpieczeństwo użytkownika poprzez automatyczne powiadamianie osób trzecich w sytuacjach krytycznych. Dostępne na rynku rozwiązania komercyjne, oparte głównie o aplikacje mobilne zależne od smartfona, wykazują ograniczenia w zakresie autonomii energetycznej, niezawodności wykrywania zdarzeń oraz pracy w warunkach braku dostępności sieci komórkowej telefonu. W ramach pracy zaprojektowano i wykonano kompletne urządzenie montowane bezpośrednio w ramie roweru, oparte na mikrokontrolerze ESP32-C6 oraz zestawie czujników obejmującym jednostkę inercyjną MPU-6500, czujnik środowiskowy BME280, odbiornik GPS Air530Z oraz modem GSM SIM800L. Opracowano autorską czterowarstwową płytkę drukowaną oraz obudowę wytwarzaną metodą druku 3D, dostosowaną do mocowania w głównym trójkącie ramy rowerowej. Oprogramowanie urządzenia, napisane w języku C++ z wykorzystaniem frameworka Arduino oraz środowiska PlatformIO, podzielono na dziesięć modułów funkcjonalnych realizujących akwizycję danych z czujników, trójfazowy algorytm detekcji upadku, niezależny rejestrator pików akceleracji o częstotliwości próbkowania 200 Hz, rejestrację telemetrii w formacie CSV na karcie microSD, procedurę alarmową z wysyłką wiadomości SMS oraz konfigurowalny serwer WWW dostępny w trybie serwisowym. Funkcjonalność urządzenia zweryfikowano w warunkach laboratoryjnych oraz w testach terenowych obejmujących akwizycję danych z czujników, rejestrację telemetrii, detekcję zdarzeń wstrząsowych oraz komunikację GSM. Przeprowadzone testy potwierdziły poprawność działania zaimplementowanych mechanizmów oraz wskazały kierunki dalszego rozwoju w zakresie optymalizacji algorytmu detekcji upadku oraz autonomii energetycznej urządzenia.

Słowa kluczowe: czarna skrzynka, telematyka rowerowa, detekcja upadku, system alarmowy GSM, bezpieczeństwo czynne, mikrokontroler ESP32, jednostka inercyjna MEMS, akwizycja danych

Spis treści

Spis treści	3
1. Wstęp	8
1.1. Motywacja	8
1.2. Cel Pracy	8
1.3. Rozwiązania przyjęte w pracy	8
1.4. Rezultat pracy	9
1.5. Organizacja pracy.....	9
2. Przegląd literatury i analiza stanu sztuki	11
2.1. Historia i geneza czarnych skrzynek.....	11
2.1.1. Początki czarnych skrzynek w lotnictwie	11
2.1.2. Rozszerzenie zastosowań czarnych skrzynek.....	11
2.1.3. Definicja i funkcje kluczowe czarnej skrzynki	11
2.1.4. Przykłady zastosowań czarnej skrzynki.....	12
2.2. Ewolucja platform systemów wbudowanych na przykładzie ekosystemu Arduino i układów SoC	12
2.2.1. Geneza i wpływ ekosystemu Arduino na inżynierię systemów wbudowanych	12
2.2.2. Architektura procesorowa: od AVR do RISC-V	12
2.2.3. Funkcje kluczowe nowoczesnych układów SoC w systemach telemetry	13
2.2.4. Przykłady zastosowań i kierunki rozwoju	13
2.3. Istniejące rozwiązania i ich analiza	13
2.3.1. Zintegrowane liczniki rowerowe z funkcjami bezpieczeństwa	13
2.3.2. Dedykowane sensory wypadkowe i kaski inteligentne.....	14
2.3.3. Autonomiczne rejestratory wizyjne.....	15
2.3.4. Systemy antykradzieżowe i lokalizatory GSM/LTE.....	15
3. Koncepcja i założenia projektowe	17
3.1. Wymagania funkcjonalne.....	17
3.1.1. Rejestracja danych.....	17
3.1.2. Detekcja zdarzeń krytycznych	18
3.1.3. Powiadomienie alarmowe	19
3.1.4. Interfejs i konfiguracja	19
3.1.5. Zarządzanie energią	20
3.2. Wymagania niefunkcjonalne	21
3.2.1. Charakterystyka czasowa reakcji	21
3.2.2. Niezawodność rejestracji danych	22

3.2.3. Odporność środowiskowa	23
3.2.4. Modułowość konstrukcji.....	23
3.2.5. Łatwość konserwacji i kalibracji.....	23
3.3. Architektura systemu.....	24
3.3.1. Diagram blokowy systemu	24
3.3.2. Tryby pracy i przejścia między nimi.....	25
3.3.3. Przepływ danych	26
3.3.4. Strategia sterowanego włączania modemu GSM	27
3.3.5. Tryb głębokiego uśpienia i strategia wybudzania	27
4. Wykorzystane technologie i narzędzia.....	28
4.1. Środowisko i narzędzia programistyczne warstwy firmware	28
4.1.1. Język programowania i framework.....	28
4.1.2. Systemy budowania i IDE	28
4.2. Biblioteki warstwy firmware	29
4.2.1. Biblioteki zewnętrzne	29
4.2.2. Standardowe biblioteki framework Arduino-ESP32	30
4.3. Technologie warstwy webowej	31
4.3.1. Języki znaczników i stylów	31
4.3.2. JavaScript i wykorzystane biblioteki	31
4.4. Środowisko i biblioteki analizy danych.....	32
4.4.1. Python i interfejs graficzny	32
4.4.2. Biblioteki przetwarzania i wizualizacji danych.....	32
4.5. Narzędzia wspomagające proces deweloperski.....	33
4.5.1. Kontrola wersji	33
4.5.2. Wspomaganie programistyczne AI	33
4.5.3. Narzędzia projektowania sprzętowego	33
5. Implementacja sprzętowa.....	35
5.1. Dobór komponentów.....	35
5.1.1. Jednostka centralna	35
5.1.2. Komunikacja alarmowa.....	37
5.1.3. Czujniki i akwizycja danych.....	38
5.1.4. Magistrale komunikacyjne	39
5.1.5. Przechowywanie danych	39
5.1.6. Zasilanie i konwersja napięcia	40
5.2. Schemat elektroniczny	42
5.2.1. Topologia magistral i połączeń	44

5.2.2. Układ zasilania	45
5.2.3. Interfejs użytkownika.....	46
5.2.4. Złącza zewnętrzne i punkty pomiarowe	47
5.3. Projekt płytki drukowanej.....	48
5.3.1. Wymiary i kształt płytki	49
5.3.2. Stos warstwowy	50
5.3.3. Rozdzielenie domen zasilania.....	50
5.3.4. Rozmieszczenie komponentów i prowadzenie ścieżek	52
5.4. Konstrukcja mechaniczna.....	54
5.4.1. Obudowa drukowana w technologii 3D	55
5.4.2. Mocowanie modułów i tuleje dystansowe	57
5.4.3. Mocowanie urządzenia do roweru	58
6. Implementacja oprogramowania.....	60
6.1. Architektura i pętla główna	60
6.1.1. Podział oprogramowania na moduły funkcjonalne	60
6.1.2. Sekwencja inicjalizacji i pętla główna.....	62
6.2. Akwizycja danych z czujników	63
6.2.1. Inicjalizacja czujników.....	64
6.2.2. Częstotliwości próbkowania	65
6.2.3. Akwizycja danych z czujnika inercyjnego i kalibracja	65
6.2.4. Pomiar napięcia ogniwa	66
6.2.5. Akwizycja sygnału z odbiornika GPS.....	66
6.2.6. Akwizycja parametrów środowiskowych	67
6.3. Algorytm detekcji upadku	68
6.3.1. Architektura algorytmu - maszyna stanu	68
6.3.2. Faza 1 - Monitorowanie spoczynku.....	70
6.3.3. Faza 2 - Obserwacja poudrzeniowa	70
6.3.4. Faza 3 - Klasyfikacja zdarzenia i okres odporności	71
6.3.5. Zestawienie parametrów algorytmu	72
6.3.6. Fragment implementacji	72
6.4. Procedura alarmowa i komunikacja GSM.....	74
6.4.1. Architektura sekwencji alarmowej	74
6.4.2. Procedura alarmu wstępnego	76
6.4.3. Sekwencja komend AT.....	76
6.4.4. Treść wiadomości sms	79
6.4.5. Mechanizm ponawiania prób i odporność na awarie.....	79

6.4.6. Interakcja z innymi modułami	80
6.4.7. Fragment implementacji	80
6.5. Rejestracja danych na karcie microSD.....	82
6.5.1. Organizacja danych na karcie	83
6.5.2. Format i częstotliwość zapisu danych z jazdy	83
6.5.3. Mechanizm trwałości zapisu	84
6.5.4. Metadane jazdy i eksport trasy w formacie GPX	84
6.5.5. Rejestrator wstrząsów	85
6.5.6. Obsługa błędów oraz integracja z interfejsem WWW.....	86
6.6. Serwer WWW	86
6.6.1. Architektura serwera i tryb dostępu Wi-Fi	86
6.6.2. Cykl życia serwera i aktywacja w trybie serwisowym.....	87
6.6.3. Struktura aplikacji webowej i nawigacja	87
6.6.4. Interfejs programistyczny API.....	89
6.6.5. Przepływ obsługi zapytań HTTP.....	91
6.6.6. Bezpieczeństwo i integracja z resztą systemu	92
7. Testy i walidacja prototypu	95
7.1. Metodologia testów	95
7.1.1. Tryby pracy w procedurach testowych.....	95
7.1.2. Aparatura pomiarowa.....	96
7.1.3. Metodologia pomiaru charakterystyk czujnika środowiskowego oraz modułu GPS...96	
7.1.4. Metodologia walidacji algorytmu detekcji upadku	97
7.2. Testy sprzętowe i charakterystyk energetycznych	97
7.2.1. Pomiary poboru prądu w trybach pracy	97
7.2.2. Autonomia energetyczna systemu	100
7.2.3. Pomiary charakterystyk czujnika środowiskowego oraz modułu GPS	101
7.2.4. Niezawodność komunikacji GSM.....	104
7.3. Walidacja algorytmu detekcji	105
7.3.1. Scenariusze testowe	105
7.3.2. Wyniki testów kontrolnych.....	106
7.3.3. Dostrajanie parametrów algorytmu.....	106
7.3.4. Obserwacje z testów terenowych	108
8. Analiza rozwiązania i kierunki rozwoju.....	113
8.1. Trudności napotkane podczas realizacji	113
8.2. Zalety i wady przyjętego rozwiązania.....	113
8.3. Potencjalne zastosowania	114

8.4. Kierunki dalszego rozwoju	114
9. Podsumowanie	116
Prace cytowane	117
Wykaz rysunków	120
Wykaz tabel	122
Wykaz kodu	123

1. Wstęp

Wzrost popularności turystyki rowerowej oraz sportów ekstremalnych, takich jak Enduro czy MTB, niesie ze sobą potrzebę zwiększenia bezpieczeństwa użytkowników poruszających się samotnie. Tradycyjne liczniki rowerowe skupiają się głównie na parametrach treningowych, często pomijając aspekty diagnostyki powypadkowej oraz automatycznego powiadamiania ratunkowego w miejscach o ograniczonej infrastrukturze. Niniejsza praca opisuje proces projektowania i implementacji urządzenia typu „czarna skrzynka”, które łączy funkcje rejestratora parametrów jazdy z systemem aktywnego alarmowania o zdarzeniach niebezpiecznych.

1.1. Motywacja

Motywacją autora do podjęcia niniejszego tematu była wieloletnia praktyka rekreacyjnej jazdy rowerowej, prowadzona głównie na trasach leśnych. Doświadczenia zebrane podczas tego typu aktywności pozwoliły zaobserwować szereg sytuacji potencjalnie niebezpiecznych. W głębi lasu rzadko znajdują się świadkowie zdarzenia, a sam rowerzysta po upadku może pozostać niezdolny do obsługi telefonu. Obserwacje te uświadomiły praktyczną potrzebę dysponowania urządzeniem zdolnym do automatycznego wykrycia upadku oraz wysłania powiadomienia z lokalizacją bez konieczności jakiegokolwiek interakcji ze strony użytkownika. Naturalnym dopełnieniem tej motywacji było wcześniejsze zainteresowanie autora elektroniką.

1.2. Cel Pracy

Głównym celem niniejszej pracy jest opracowanie koncepcji oraz implementacja autonomicznego systemu rejestracji i analizy zdarzeń dynamicznych dla pojazdów lekkich, na przykładzie „rowerowej czarnej skrzynki”. Praca skupia się na zwiększeniu bezpieczeństwa podczas jazdy oraz rozwiązaniu problemu ograniczonej diagnostyki zdarzeń powypadkowych w ruchu rowerowym.

1.3. Rozwiązania przyjęte w pracy

W celu realizacji założeń projektowych oraz sprostania wymaganiom stawianym przed systemami typu rejestrator danych wypadkowych (EDR, *Event Data Recorder*), w niniejszej pracy zdecydowano się na autorskie podejście integrujące przetwarzanie danych inercyjnych z niezależnym torem komunikacji alarmowej, co wynika z potrzeby stworzenia urządzenia niezależnego od zewnętrznej infrastruktury obliczeniowej (np. smartfonów) w momencie wystąpienia zdarzenia krytycznego. Architektura systemu została oparta na paradygmacie modułowości, co zapewnia skalowalność rozwiązania oraz ułatwia diagnostykę poszczególnych podsystemów.

Kluczowe paradygmaty przyjęte podczas realizacji pracy obejmują:

- **Podejście zorientowane na autonomię komunikacyjną:** Zdecydowano się na wyposażenie systemu w bezpośredni interfejs łączności bezprzewodowej dalekiego zasięgu. Rozwiązanie to zapewnia redundancję systemu alarmowego i pozwala na wysyłkę powiadomień ratunkowych nawet w przypadku zniszczenia innych urządzeń elektronicznych posiadanych przez użytkownika;
- **Wieloskalowy model akwizycji danych:** W pracy przyjęto strategię dwupoziomowego zapisu informacji. Pierwszy poziom obejmuje ciągłą rejestrację parametrów trasy z niską częstotliwością, natomiast drugi poziom aktywowany jest zdarzeniowo i pozwala na przechwytywanie danych o wysokiej rozdzielczości czasowej (mechanizm rejestratora danych wypadkowych). Pozwala to na precyzyjną analizę dynamiki zderzenia przy jednoczesnej optymalizacji wykorzystania pamięci masowej;

- **Architektura sterowana zdarzeniami:** W celu maksymalizacji czasu pracy na zasilaniu bateryjnym, system zaprojektowano w oparciu o stany głębokiego uśpienia. Logika urządzenia zakłada wybudzanie kluczowych modułów jedynie w odpowiedzi na sygnały z sensorów inercyjnych lub interakcję użytkownika, co znacząco redukuje średni pobór prądu;
- **Uniwersalny interfejs konfiguracyjny:** Interakcja z użytkownikiem została rozwiązana poprzez implementację bezprzewodowego serwera usług, co eliminuje konieczność tworzenia dedykowanych aplikacji na konkretne systemy operacyjne i pozwala na obsługę urządzenia z poziomu standardowej przeglądarki internetowej.

1.4. Rezultat pracy

Głównym rezultatem niniejszej pracy jest opracowanie i weryfikacja koncepcji autonomicznej cyfrowej czarnej skrzynki dla pojazdów jednośladowych. Kompletnego rozwiązania problemu rejestracji telemetry oraz detekcji zdarzeń krytycznych w środowisku mikromobilności. Pracę zorientowano wokół tezy o możliwości zapewnienia zaawansowanych funkcji bezpieczeństwa rowerzysty w szczególności automatycznego alarmowania ratunkowego, przy zachowaniu pełnej autonomii sprzętowej i niezależności od infrastruktury smartfonowej, na której opiera się większość dostępnych obecnie rozwiązań komercyjnych.

Do kluczowych składowych rezultatu pracy należą:

- **Architektura sprzętowa urządzenia** - integralna struktura funkcjonalna łącząca jednostkę inercyjną, lokalizację satelitarną, łączność komórkową oraz lokalny interfejs konfiguracyjny w jednym systemie wbudowanym, dostosowanym do warunków pracy na ramie roweru;
- **Algorytm wykrywania upadków** - trójfazowa metoda progowa oparta na analizie sygnału inercyjnego z weryfikacją czasową i krzyżową z danymi prędkościowymi;
- **Dwuwarstwowa rejestracja danych telemetrycznych** - równoczesna akwizycja niskoczęstotliwościowego logu jazdy oraz wysokoczęstotliwościowego zapisu pików akceleracji, umożliwiającą zarówno analizę długoterminową stylu jazdy, jak i precyzyjną rekonstrukcję dynamiki pojedynczego zdarzenia krytycznego;
- **Samodzielne alarmowanie ratunkowe** - niezależny od urządzeń mobilnych mechanizm powiadamiania osób trzecich w sytuacjach krytycznych, z odpornością na nagłe awarie zasilania oraz krótkotrwałą utratę pokrycia satelitarnego;
- **Prototyp urządzenia wraz z dokumentacją techniczną** - implementacyjny dowód wykonalności (*proof of concept*) opracowanej koncepcji, obejmujący kompletną realizację sprzętową, oprogramowanie układowe oraz lokalny serwis konfiguracyjny, mogący stanowić bazę dla dalszych prac rozwojowych w obszarze bezpieczeństwa mikromobilności.

1.5. Organizacja pracy

Struktura niniejszej pracy została zaprojektowana w sposób prowadzący od analizy teoretycznej i stanu sztuki, poprzez proces projektowy i implementacyjny, aż po weryfikację empiryczną i podsumowanie wyników. Tekst został podzielony na dziewięć rozdziałów o następującej tematyce:

- **Rozdział 1** wprowadza w tematykę pracy. Sformułowano w nim cel pracy, przedstawiono motywację, przyjęte rozwiązania oraz oczekiwane rezultaty;
- **Rozdział 2** zawiera przegląd literatury oraz analizę stanu sztuki. Przedstawiono w nim genezę systemów typu czarna skrzynka, ewolucję platformy Arduino/ESP32 oraz krytyczną analizę istniejących na rynku rozwiązań komercyjnych i hobbystycznych;
- **Rozdział 3** zawiera koncepcję i założenia projektowe urządzenia. Sformułowano w nim wymagania funkcjonalne i niefunkcjonalne stawiane systemowi, opisano architekturę na

poziomie strukturalnym, behawioralnym oraz informacyjnym, a także przedstawiono strategię zarządzania energią urządzenia;

- **Rozdział 4** przedstawia wykorzystane technologie, biblioteki oraz narzędzia programistyczne i projektowe, opisane w oderwaniu od samej implementacji. Omówiono w nim środowisko programistyczne warstwy firmware, biblioteki obsługi peryferiów, technologie warstwy webowej, narzędzia analizy danych oraz oprogramowanie wykorzystane do projektowania sprzętu;
- **Rozdział 5** dokumentuje implementację sprzętową urządzenia. Zawiera szczegółowy opis schematu elektrycznego z mapą połączeń komponentów, projekt płytki drukowanej obejmujący wybór struktury warstwowej i rozmieszczenie elementów, projekt konstrukcji mechanicznej obudowy drukowanej w technologii 3D oraz metodę montażu urządzenia do roweru;
- **Rozdział 6** opisuje implementację warstwy oprogramowania. Przedstawiono w nim architekturę kodu i sposób konfiguracji systemu przez pliki JSON, mechanizmy akwizycji i przetwarzania danych z czujników, implementację algorytmu detekcji upadku opartego na maszynie stanów, procedurę alarmową z wysyłką wiadomości SMS, strategię zarządzania trybami pracy i energią oraz konstrukcję lokalnego serwera webowego pełniącego funkcję interfejsu serwisowego;
- **Rozdział 7** prezentuje testy i walidację zrealizowanego prototypu. Opisano w nim metodykę badań oraz wykorzystaną aparaturę pomiarową, a następnie przedstawiono wyniki testów sprzętowych i charakterystyk energetycznych, pomiarów charakterystyk czujnika środowiskowego oraz modułu nawigacji satelitarnej, a także walidację algorytmu detekcji upadku w warunkach stanowiskowych i terenowych;
- **Rozdział 8** zawiera analizę zrealizowanego rozwiązania oraz kierunki jego dalszego rozwoju. Omówiono w nim trudności napotkane podczas realizacji, zalety i wady przyjętych rozwiązań projektowych, potencjalne obszary zastosowania systemu oraz planowane kierunki rozbudowy;
- **Rozdział 9** stanowi podsumowanie całości prac. Zawiera zwięzłą ocenę stopnia spełnienia założonych celów, wskazanie osiągnięć oraz niedostatków pracy, a także zasadnicze wnioski płynące z jej realizacji.

Całość uzupełniają spisy literatury, rysunków, tabel oraz listingów kodu źródłowego, stanowiące integralną część dokumentacji technicznej projektu.

2. Przegląd literatury i analiza stanu sztuki

W poniższym rozdziale dokonano krytycznej analizy ewolucji rejestratorów danych oraz technologii wspomagających ich rozwój. Rozważania rozpoczynają się od historycznego ujęcia genezy systemów typu czarna skrzynka w transporcie, przechodząc do analizy platform mikrokontrolerowych oraz przeglądu współczesnych rozwiązań komercyjnych i badawczych w obszarze bezpieczeństwa rowerzysty.

2.1. Historia i geneza czarnych skrzynek

Koncepcja rejestrowania parametrów w celu późniejszej rekonstrukcji zdarzeń, choć kojarzona głównie z nowoczesną elektroniką, ma swoje korzenie w połowie XX wieku. Systemy te, określane w literaturze rejestrator danych wypadkowych (EDR, *Event Data Recorder*), ewoluowały od prymitywnych zapisów mechanicznych do zaawansowanych systemów cyfrowych [1].

2.1.1. Początki czarnych skrzynek w lotnictwie

Za ojca technologii czarnych skrzynek uznaje się dr. Davida Warrena z australijskiego Aeronautical Research Laboratories. W 1953 roku zaproponował on urządzenie zdolne do ciągłej rejestracji głosu w kokpicie oraz parametrów lotu, co miało pomóc w analizie przyczyn katastrof odrzutowca de Havilland Comet [2]. Pierwsze modele, znane jako „Flight Memory”, wykorzystywały stalową taśmę magnetyczną jako nośnik danych. Jak zauważa literatura przedmiotu, jaskrawopomarańczowy kolor obudowy, mający ułatwić lokalizację urządzenia, stał się standardem w lotnictwie cywilnym już w latach 60. XX wieku.

Technologia ta szybko zyskała uznanie w branży lotniczej. Do lat 60. XX wieku czarne skrzynki stały się obowiązkowym wyposażeniem większości samolotów pasażerskich. Pierwsze modele, choć proste w porównaniu do współczesnych urządzeń, rejestrowały takie parametry jak wysokość, prędkość i podstawowe komunikaty załogi. Współczesne rejestratory, znane jako rejestrator parametrów lotu (FDR, *Flight Data Recorder*) i rejestrator rozmów w kokpicie (CVR, *Cockpit Voice Recorder*), zbierają setki parametrów w czasie rzeczywistym, co czyni je niezastąpionymi w analizie zdarzeń lotniczych.

2.1.2. Rozszerzenie zastosowań czarnych skrzynek

Wraz z postępem miniaturyzacji, systemy EDR zaczęły być wdrażane w innych gałęziach transportu. W motoryzacji przełom nastąpił w latach 70., kiedy to General Motors wprowadziło pierwsze moduły monitorujące pracę poduszek powietrznych [3].

Współczesne ramy prawne dla tych urządzeń definiuje m.in. amerykański standard IEEE 1616 (*Standard for Motor Vehicle Event Data Recorders*), który określa protokoły dostępu do danych oraz ich formatowanie [4]. W Unii Europejskiej, zgodnie z Rozporządzeniem 2019/2144, systemy EDR stały się obowiązkowym wyposażeniem wszystkich nowych pojazdów od 2022 roku, co podkreśla ich kluczową rolę w poprawie bezpieczeństwa ruchu drogowego.

2.1.3. Definicja i funkcje czarnej skrzynki

Według publikacji, „czarna skrzynka” (*Event Data Recorder*) to system pokładowy, którego głównym zadaniem jest rejestracja i zabezpieczenie parametrów technicznych pojazdu w fazie przed zderzeniowej oraz w samym momencie kolizji. W przeciwieństwie do rejestratorów ciągłych, EDR skupia się wyłącznie na danych o wysokim znaczeniu dla rekonstrukcji wypadku [5].

Kluczowe funkcje systemu:

- Praca w pętli (Buforowanie): Urządzenie stale monitoruje parametry (prędkość, obroty, stan hamulca), ale przechowuje je tymczasowo. Starsze dane są na bieżąco nadpisywane nowymi, aż do momentu wykrycia incydentu.

- Detekcja progu wyzwalania (Wyzwalanie): System aktywuje stały zapis danych dopiero po przekroczeniu krytycznej wartości opóźnienia/przyspieszenia
- Trwałość zapisu: Dane z bufora (np. z ostatnich 5 sekund przed zdarzeniem) są natychmiast przenoszone do pamięci nieulotnej (EEPROM/Flash). Dzięki temu pozostają one dostępne do odczytu nawet po całkowitym zniszczeniu zasilania pojazdu.
- Kategoryzacja zdarzeń: System rozróżnia zdarzenia „niewyciszające” (lekkie kolizje) od „wyciszających” (aktywacja poduszek powietrznych), co pozwala na precyzyjną ocenę powagi wypadku przez biegłych.

2.1.4. Przykłady zastosowań czarnej skrzynki

Ewolucja technologii zapisu danych doprowadziła do wyjścia systemów EDR poza obszar lotnictwa. Obecnie rejestratory o charakterystyce „czarnej skrzynki” spotykane są w wielu dziedzinach wymagających wysokiej niezawodności:

- **Transport morski (VDR, *Voyage Data Recorder*):** Zgodnie z konwencją SOLAS, statki o dużej wyporności muszą posiadać rejestrator VDR. Urządzenia te przechowują dane z radarów, radiotelefonów oraz parametry kadłuba w kapsułach zdolnych do przetrwania na dużych głębokościach [6].
- **Medycyna i inżynieria kliniczna:** Rejestratory typu Holter oraz implantowalne kardiowertery-defibrylatory (ICD, *Implantable Cardioverter-Defibrillator*) monitorują parametry życiowe w pętli kołowej. W momencie wykrycia anomalii (wyzwalacz), system trwale zapisuje fragment elektrokardiogramu (EKG), co umożliwia precyzyjną diagnostykę incydentów kardiologicznych.
- **Sporty kontaktowe i biomechanika:** Sensory wbudowane w kaski zawodników NFL (futbol amerykański) rejestrują siły uderzeń i przyspieszenia kątowe. Analiza danych z tych „osobistych czarnych skrzynek” jest kluczowa w opracowywaniu standardów ochrony przed wstrząśnieniami mózgu.
- **Logistyka i monitoring łańcucha dostaw:** W transporcie towarów wrażliwych stosuje się rejestratory uderzeń, które dokumentują przekroczenia dopuszczalnych przeciążeń, stanowiąc obiektywny dowód w procesach ubezpieczeniowych.

2.2. Ewolucja platform systemów wbudowanych na przykładzie ekosystemu Arduino i układów SoC

Współczesne projektowanie systemów elektronicznych opiera się na paradygmacie szybkiego prototypowania, który został zdefiniowany przez rozwój ruchu *Open Source Hardware*. Podrozdział ten analizuje przejście od prostych układów 8-bitowych do zaawansowanych systemów jednoukładowych (SoC, *System on a Chip*), które stanowią fundament technologiczny niniejszej pracy.

2.2.1. Geneza i wpływ ekosystemu Arduino na inżynierię systemów wbudowanych

Platforma Arduino, zapoczątkowana w 2005 roku, wprowadziła nową jakość w projektowaniu systemów wbudowanych poprzez udostępnienie ustandaryzowanych warstw abstrakcji sprzętowej (HAL - *Hardware Abstraction Layer*) [7]. Kluczowym osiągnięciem tego ekosystemu nie była sama warstwa sprzętowa, lecz stworzenie ujednoliconego interfejsu programistycznego (API), który umożliwił przenoszenie kodu pomiędzy różnymi architekturami procesorowymi. W literaturze wskazuje się, że to właśnie demokratyzacja dostępu do narzędzi inżynierskich pozwoliła na gwałtowny rozwój niszowych rozwiązań telemetrycznych, do których zalicza się projektowana czarna skrzynka.

2.2.2. Architektura procesorowa: od AVR do RISC-V

Tradycyjne systemy wbudowane oparte na architekturze 8-bitowej (np. AVR) cechowały się ograniczoną mocą obliczeniową i skromnymi zasobami pamięci operacyjnej (SRAM), co uniemożliwiało realizację złożonych zadań, takich jak wielowątkowe przetwarzanie sygnałów czy obsługa stosów sieciowych w czasie rzeczywistym.

W niniejszym projekcie zdecydowano się na wykorzystanie układu **Seeed XIAO ESP32-C6**, który reprezentuje nowoczesny kierunek rozwoju systemów SoC. Wykorzystuje on architekturę **RISC-V** - otwarty standard zestawu instrukcji (ISA), który w ostatnich latach zyskuje miano standardu w aplikacjach IoT ze względu na optymalizację pod kątem wydajności energetycznej [8]. Przejście na architekturę 32-bitową o taktowaniu 160 MHz pozwala na:

- Realizację zaawansowanych operacji na liczbach zmiennoprzecinkowych (potrzebnych do obliczania wielkości wstrząsu);
- Wykorzystanie mechanizmów DMA (*Direct Memory Access*) do bezobsługowego przesyłania danych między sensorami a pamięcią;
- Równoległą obsługę komunikacji bezprzewodowej i akwizycji danych.

2.2.3. Funkcje kluczowe nowoczesnych układów SoC w systemach telemetry

Nowoczesne platformy systemów wbudowanych, takie jak rodzina ESP32, integrują w jednej strukturze krzemowej moduły, które wcześniej wymagały osobnych układów peryferyjnych. Do najważniejszych cech, kluczowych z punktu widzenia „czarnej skrzynki”, należą:

- **Zintegrowana łączność wieloprotokołowa:** Natywne wsparcie dla Wi-Fi 6, Bluetooth 5 (LE) oraz IEEE 802.15.4 (Zigbee/Thread), co pozwala na budowę rozproszonych sieci czujników [8];
- **Zaawansowane zarządzanie energią:** Możliwość pracy w stanach głębokiego snu (*Deep Sleep*) z zachowaniem aktywności wybranych peryferiów (np. kontrolera LP-IO), co jest niezbędne w urządzeniach zasilanych bateryjnie [8];
- **Wsparcie dla systemów czasu rzeczywistego (RTOS):** Architektura ta natywnie współpracuje z systemem FreeRTOS, co umożliwia deterministyczne zarządzanie zadaniami (np. nadanie najwyższego priorytetu modułowi detekcji wypadku).

2.2.4. Przykłady zastosowań i kierunki rozwoju

Zastosowanie ekosystemu Arduino w połączeniu z mocą układów SoC znajduje odzwierciedlenie w profesjonalnych systemach monitorowania parametrów środowiskowych, inteligentnych miastach oraz w zaawansowanej teledystrybucji sportowej. Jak zauważają badacze w publikacjach dotyczących bezpieczeństwa mikromobilności, możliwość taniego i szybkiego wdrażania systemów typu EDR na platformach typu ESP32 otwiera drogę do masowej diagnostyki wypadków z udziałem niechronionych uczestników ruchu drogowego.

2.3. Istniejące rozwiązania i ich analiza

Rynek elektroniki rowerowej w ostatniej dekadzie przeszedł ewolucję od prostych liczników mierzących prędkość i dystans do złożonych systemów teledystrybucyjnych integrujących dane z wielu sensorów. Poniżej przedstawiono przegląd dominujących na rynku rozwiązań, sklasyfikowanych według ich głównej funkcjonalności i architektury systemowej.

2.3.1. Zintegrowane liczniki rowerowe z funkcjami bezpieczeństwa

Najszerszą grupę produktów stanowią zaawansowane komputery rowerowe wyposażone w odbiorniki GPS oraz sensory inercyjne.

- **Garmin Edge (seria 540/840/1040/1050):** Urządzenia te wykorzystują funkcję wykrywania zdarzeń (*Incident Detection*). W przypadku wykrycia gwałtownego hamowania lub uderzenia, licznik inicjuje procedurę alarmową. Użytkownik ma określony czas na anulowanie alertu, po czym urządzenie wysyła wiadomość do kontaktów alarmowych [9]. Garmin Edge serii 1050, został przedstawiony na Rysunek 1 [10].
- **Wahoo ELEMNT ROAM:** Podobnie jak rozwiązania marki Garmin, urządzenia Wahoo oferują funkcję *Live Track*, umożliwiającą udostępnianie pozycji w czasie rzeczywistym. Analiza dynamiki jazdy odbywa się w oparciu o zintegrowane akcelerometry, jednak zapis danych w pliku wynikowym ogranicza się zazwyczaj do interwału jednej sekundy [11].



Rysunek 1. Garmin Edge 1050 (źródło: [10]).

2.3.2. Dedykowane sensory wypadkowe i kaski inteligentne

Osobną kategorię stanowią urządzenia, których jedynym zadaniem jest monitorowanie uderzeń i wstrząsów. Są one montowane najczęściej na kasku, co pozwala na pomiar sił bezpośrednio oddziałujących na głowę rowerzysty.

- **Specialized ANGi:** Niewielki sensor montowany do kasku, integrujący akcelerometr i żyroskop. Urządzenie mierzy siły liniowe i rotacyjne. Wymaga stałej łączności Bluetooth z aplikacją mobilną *Specialized Ride* [12].
- **Tocsen:** Niemiecki system składający się z precyzyjnego sensora uderzeniowego montowanego na kasku. Charakterystyczną cechą tego rozwiązania jest budowa społeczności „ratowników” - alert o wypadku jest wysyłany nie tylko do bliskich, ale również do innych użytkowników systemu Tocsen znajdujących się w pobliżu. System Tocsen został przedstawiony na Rysunek 2 [13].
- **Kaski Livall (seria EVO21):** Przykład pełnej integracji elektroniki z wyposażeniem pasywnym. Kask posiada wbudowane oświetlenie kierunkowe, głośniki oraz sensor upadku, który automatycznie wysyła powiadomienie SOS po wykryciu gwałtownej zmiany orientacji przestrzennej kasku [14].



Rysunek 2. Czujnik upadku Tocsen do montażu na kasku (źródło: [13]).

2.3.3. Autonomiczne rejestratory wizyjne

W odpowiedzi na potrzebę dokumentowania zdarzeń drogowych (szczególnie konfliktów z kierowcami), powstały urządzenia łączące oświetlenie z kamerami pracującymi w pętli.

- **Cycliq Fly6 Sport i Fly12 Pro:** Urządzenia te są określane przez producenta jako „czarne skrzynki dla rowerzystów”. Ich kluczową funkcją jest *Incident Protection* - wbudowany sensor po wykryciu upadku (pochylenie urządzenia powyżej 60 stopni przez ponad 5 sekund) automatycznie zabezpiecza nagranie wideo sprzed i po zdarzeniu przed nadpisaniem. System ten dostarcza dowodów wizualnych, ale zazwyczaj nie oferuje zaawansowanej telemetrii inercyjnej dostępnej dla użytkownika w formie surowych logów. Systemy Cycliq Fly6 i Fly12 zostały przedstawiony na Rysunek 3 [15]



Rysunek 3. Kamera rowerowa przednia CYCLIQ FLY12 Sport i tylna FLY6 Pro (źródło: [15]).

2.3.4. Systemy antykradzieżowe i lokalizatory GSM/LTE

Ostatnią grupę stanowią urządzenia skupiające się na autonomii komunikacyjnej, montowane na stałe do ramy roweru.

- **BikeFinder:** Norweski system ukryty w kierownicy roweru. Posiada własną kartę SIM, moduł GPS oraz sensor ruchu. Urządzenie raportuje pozycję pojazdu w przypadku kradzieży, a jego konstrukcja uniemożliwia łatwy demontaż [16]. System antykradzieżowy BikeFinder został przedstawiony na Rysunek 4.
- **AlterLock:** Japoński system łączący alarm akustyczny z powiadomieniami na telefon. Wykorzystuje technologię Sigfox lub LTE-M (w zależności od wersji) do przesyłania danych o pozycji. Skupia się na detekcji ruchu spoczynkowego, a nie na dynamice jazdy kolizyjnej [17].



Rysunek 4. Lokalizator GPS/GSM BikeFinder (źródło: [16]).

3. Koncepcja i założenia projektowe

Niniejszy rozdział stanowi rdzeń koncepcyjny pracy. Przedstawia założenia, decyzje i kompromisy projektowe, które ukształtowały zaproponowane rozwiązanie cyfrowej czarnej skrzynki rowerowej. W odróżnieniu od opisu istniejących rozwiązań rynkowych, zaprezentowanego w rozdziale poprzednim oraz od warstwy implementacyjnej, omówionej w rozdziałach kolejnych, treść poniższa skupia się na uzasadnieniu obranego kierunku rozwoju projektu i na sformalizowaniu oczekiwań stawianych docelowemu urządzeniu.

Punktem wyjścia jest rozróżnienie pomiędzy wymaganiami funkcjonalnymi, opisującymi zachowanie systemu i oczekiwane od niego operacje, a wymaganiami нефunkcjonalnymi, definiującymi jakościowe i jakościowo-ilościowe cechy rozwiązania, takie jak niezawodność, autonomia energetyczna czy modułowość konstrukcji. Na bazie tak sformułowanych wymagań zaprezentowano koncepcję architektury systemu oraz uzasadniono dobór kluczowych technologii i komponentów. Osobny podrozdział poświęcono strategii zarządzania energią, której rola w urządzeniu zasilanym bateryjnie, eksploatowanym w trybie ciągłym, wykracza poza zwykłe zagadnienie inżynieryjne i stanowi jeden z czynników determinujących całość architektury sprzętowej i programowej.

Decyzje projektowe opisane w niniejszym rozdziale stanowią podstawę dla rozważań technicznych zawartych w rozdziałach kolejnych - od projektu sprzętowego, przez implementację oprogramowania, po metodologię i wyniki testów weryfikacyjnych.

3.1. Wymagania funkcjonalne

Wymagania funkcjonalne określają zbiór operacji, jakie urządzenie powinno wykonywać, oczekiwany przebieg jego interakcji z otoczeniem - rowerzystą, modułami sprzętowymi oraz infrastrukturą sieciową. Sformułowano je na poziomie abstrakcji niezależnym od warstwy implementacyjnej, koncentrując się na obserwowanym zachowaniu systemu, a nie na konkretnym sposobie realizacji poszczególnych funkcji. Szczegóły implementacyjne, w tym przyjęte algorytmy decyzyjne i wartości progowe, opisano w Rozdziale 6.

Wymagania pogrupowano w pięć obszarów tematycznych, odpowiadających głównym zakresom odpowiedzialności urządzenia. Pierwszy obszar dotyczy ciągłej rejestracji parametrów trasy i środowiska, stanowiącej rdzeń funkcjonalny rejestratora. Drugi opisuje mechanizmy detekcji zdarzeń krytycznych, których wystąpienie inicjuje procedurę powiadomienia alarmowego - tę z kolei definiuje obszar trzeci. Obszar czwarty obejmuje lokalny interfejs serwisowy, umożliwiający użytkownikowi konfigurację oraz przegląd danych historycznych. Obszar piąty grupuje wymagania związane z zarządzaniem energią, których spełnienie warunkuje praktyczną użyteczność urządzenia w warunkach eksploatacyjnych.

3.1.1. Rejestracja danych

Podstawową funkcją urządzenia jest ciągła akwizycja i utrwalanie parametrów dynamiki ruchu, geolokalizacji oraz warunków środowiskowych podczas aktywnej eksploatacji roweru. Niniejszy podrozdział definiuje wymagania dotyczące częstotliwości próbkowania, zakresu rejestrowanych wielkości oraz organizacji danych na nośniku pamięci masowej.

System rejestruje dane środowiskowe i nawigacyjne podczas aktywnej jazdy z adaptacyjną częstotliwością próbkowania uzależnioną od stanu ruchu pojazdu. Dla prędkości przekraczającej 15 km/h częstotliwość efektywnego zapisu wynosi 1 Hz, dla wolniejszej jazdy 0,5 Hz, a w stanie postoju 0,2 Hz. Adaptacja częstotliwości pozwala na ograniczenie obciążenia pamięci masowej w okresach mało dynamicznych przy jednoczesnym zachowaniu wysokiej rozdzielczości czasowej trasy podczas właściwej jazdy.

Pojedynczy rekord pamięci masowej zawiera 28 pól danych, pogrupowanych w sześć kategorii:

- znaczniki czasowe - czas lokalny, czas pracy urządzenia od ostatniego uruchomienia oraz unikatowy identyfikator sesji jazdy;
- dane geolokalizacyjne z modułu GPS - szerokość i długość geograficzna, wysokość nad poziomem morza, liczba widzianych satelitów oraz binarny status fixa pozycji;
- parametry dynamiki ruchu - prędkość chwilowa, dystans skumulowany od początku sesji oraz kurs;
- dane inercyjne z modułu inercyjnego - przyspieszenia w trzech osiach z rozdzielczością jednej tysięcznej g oraz prędkości kątowe w trzech osiach z rozdzielczością jednej tysięcznej stopnia na sekundę;
- dane środowiskowe z modułu środowiskowego - temperatura, wilgotność względna oraz ciśnienie atmosferyczne;
- flagi detekcji zdarzeń i parametry stanu systemu - flaga ruchu pojazdu, flagi wystąpienia impactu i potwierdzonego upadku, magnituda i klasyfikacja nasilenia impactu, znacznik czasowy ostatniego wykrytego impactu, czas trwania bieżącej sesji oraz aktualne napięcie ogniwa.

Dane są zapisywane na karcie microSD w formacie CSV z synchronizacją systemu plików co dziesięć rekordów, co zapewnia odporność na nagłą utratę zasilania kosztem utraty co najwyżej dziesięciu ostatnich próbek. Wartość ta uznana została za akceptowalną z uwagi na adaptacyjną częstotliwość zapisu - w trybie maksymalnej rejestracji odpowiada to dziesięciu sekundom danych, a w trybie postojowym pięćdziesięciu sekundom.

Organizacja danych na nośniku. Struktura katalogów na karcie SD odzwierciedla hierarchię czasową w układzie rok / miesiąc / dzień, co umożliwia archiwizację długoterminową i ułatwia eksport danych do narzędzi analitycznych. Każda sesja jazdy generuje trzy współpracujące pliki: surowy log w formacie CSV, podsumowanie sesji w formacie JSON oraz trasę GPS w standardzie GPX, kompatybilnym z popularnymi platformami treningowymi.

3.1.2. Detekcja zdarzeń krytycznych

Drugą fundamentalną funkcją urządzenia jest automatyczne rozpoznawanie sytuacji wymagających reakcji alarmowej. Niniejszy podrozdział definiuje wymagania dotyczące mechanizmów detekcyjnych oraz strategii ich konfigurowalności. Szczegółowe algorytmy klasyfikacji i wartości progowe opisano w Rozdziale 6, a wyniki ich testów w Rozdziale 7.

System rozpoznaje sytuacje wymagające reakcji alarmowej za pomocą zintegrowanego, wielofazowego algorytmu detekcji upadku, opartego na analizie sygnału z modułu inercyjnego i wspomaganego kontekstowo informacją o prędkości GPS. Algorytm odzwierciedla naturalną sygnaturę zdarzenia upadku, na którą składają się trzy następujące po sobie fazy:

Pierwszą fazą jest wykrycie gwałtownego piku przyspieszenia (uderzenia), sygnalizującego potencjalny moment zderzenia urządzenia z podłożem lub przeszkodą. Faza ta uruchamia procedurę dalszej analizy, ale sama w sobie nie stanowi podstawy do alarmu - w warunkach rzeczywistej eksploatacji rowerowej, w szczególności w terenie nieutwardzonym, piki o porównywalnej magnitudzie generowane są przez najazdy na krawężniki, dziury i nierówności nawierzchni, których klasyfikacja jako zdarzeń alarmowych prowadziłaby do nieakceptowalnej liczby fałszywych powiadomień.

Drugą fazą jest obserwacja stanu dynamiki ruchu w przedziale czasowym następującym po wystąpieniu piku. Długość tego okna stanowi parametr konfigurowalny umożliwiający dostrojenie zachowania algorytmu w drodze prób terenowych. W trakcie tej fazy algorytm analizuje sygnał inercyjny w poszukiwaniu jednego z dwóch wzorców rozstrzygających: utrzymującego się stanu niskiej dynamiki ruchu, interpretowanego jako unieruchomienie urządzenia po wypadku, albo wznowienia charakterystycznych dla normalnej jazdy drgań, świadczących o zachowaniu kontroli nad pojazdem przez rowerzystę. Pomocniczo wykorzystywana jest informacja o prędkości GPS - utrzymujący się stan niemal zerowej prędkości zwiększa pewność klasyfikacji zdarzenia jako wypadku.

Trzecią fazą jest klasyfikacja końcowa zdarzenia jako potwierdzonego upadku lub fałszywego alarmu. Tylko w pierwszym przypadku inicjowana jest procedura powiadomienia opisana w Podrozdziale 3.1.3. W drugim przypadku system powraca do trybu normalnej rejestracji, odnotowując w logu fakt wystąpienia piku oraz wynik klasyfikacji do późniejszej analizy.

Niezależnie od wyniku klasyfikacji, każdy wykryty pik powyżej zdefiniowanego progu zapisywany jest w rejestrze pomiarowym wraz z jego magnitudą oraz przypisanym poziomem nasilenia w czterostopniowej skali. Dane te stanowią podstawę do retrospektywnej analizy charakterystyki wibracyjnej tras rowerowych oraz do kalibracji parametrów algorytmu detekcyjnego, metodologię, której opisano w Rozdziale 7.

3.1.3. Powiadomienie alarmowe

Wykrycie zdarzenia krytycznego inicjuje procedurę powiadomienia osób kontaktowych. Niniejszy podrozdział definiuje wymagania dotyczące przebiegu tej procedury, w tym mechanizmu potwierdzenia alarmu, treści wysyłanej wiadomości oraz strategii zarządzania kontaktami alarmowymi.

Po wykryciu zdarzenia krytycznego system wchodzi w stan oczekiwania trwający 30 sekund, sygnalizowany wizualnie diodą. W trakcie tego okna alarm może zostać anulowany na dwa sposoby. Pierwszym jest manualne wciśnięcie przycisku - stanowi ono świadomą interwencję użytkownika. Drugim jest automatyczne anulowanie na podstawie rozpoznania wzorca aktywnej jazdy w sygnale inercyjnym: utrzymujące się przez co najmniej dziesięć sekund rytmiczne drgania, charakterystyczne dla pedałowania i toczenia się koła po nawierzchni, traktowane są jako dowód, że rowerzysta zachował kontrolę nad pojazdem i kontynuuje jazdę.

Świadomie zrezygnowano przy tym z anulowania alarmu na podstawie prędkości GPS, ponieważ sygnał satelitarny nie stanowi wiarygodnego źródła decyzji w tym kontekście - pozorny ruch wynikający z dryfu pozycji na postoju mógłby doprowadzić do błędnego anulowania alarmu w sytuacji, gdy uszkodzony pozostaje unieruchomiony. Kryterium oparte na rozpoznaniu wzorca jazdy z modułu inercyjnego jest znacznie trudniejsze do przypadkowego wywołania niż prosty próg wykrycia ruchu, co ogranicza ryzyko niezamierzonego anulowania przy chwilowym poruszeniu urządzenia, na przykład podczas podnoszenia przewróconego roweru przez osobę postronną.

Po wygaśnięciu okna potwierdzenia, jeśli alarm nie został anulowany, system wysyła wiadomość SMS poprzez moduł GSM do numeru zarejestrowanego w pliku konfiguracyjnym alarmów na karcie microSD. Treść wiadomości zawiera zwięzłą informację o zdarzeniu: identyfikator urządzenia, znacznik czasowy oraz link do aktualnej pozycji geograficznej w formacie zgodnym z usługą Google Maps, z dokładnością współrzędnych do sześciu miejsc po przecinku. Treść skomponowano w zestawie znaków GSM-7, co umożliwia zmieszczenie pełnego komunikatu w jednej wiadomości o długości 160 znaków. Status wysyłki każdej wiadomości - powodzenie, błąd transmisji, brak rejestracji w sieci - jest rejestrowany w dedykowanym logu na karcie SD. W przypadku braku świeżego fixa GPS w momencie wystąpienia alarmu, system wykorzystuje ostatnią znaną pozycję.

Numer kontaktowy jest przechowywany w pliku konfiguracyjnym na karcie SD i wczytywany podczas rozruchu urządzenia. Modyfikacja numeru kontaktowego odbywa się wyłącznie poprzez lokalny interfejs serwisowy opisany w podrozdziale 3.1.4, co eliminuje ryzyko przypadkowej zmiany danych alarmowych podczas normalnej eksploatacji. Wprowadzane numery podlegają walidacji pod kątem zgodności z formatem międzynarodowym przed zapisem do pliku konfiguracyjnego.

3.1.4. Interfejs i konfiguracja

Urządzenie nie posiada własnego wyświetlacza ani fizycznych elementów konfiguracyjnych, co podyktowane jest zarówno względami energetycznymi, jak i ograniczeniami przestrzennymi konstrukcji przeznaczonej do montażu na rowerze. Funkcję interfejsu użytkownika pełni aplikacja webowa hostowana lokalnie przez urządzenie, dostępna w wydzielonym trybie pracy.

System udostępnia interfejs webowy hostowany lokalnie przez mikrokontroler, dostępny po wejściu urządzenia w tryb serwisowy. Tryb ten aktywowany jest poprzez wciśnięcie dedykowanego przycisku serwisowego. Wymóg fizycznego wciśnięcia przycisku zapobiega niezamierzonej aktywacji

trybu podczas normalnej eksploatacji. Po aktywacji urządzenie uruchamia własną sieć Wi-Fi w trybie punktu dostępu (*Access Point*), do której użytkownik łączy się z dowolnego urządzenia z przeglądarką internetową, uzyskując dostęp do interfejsu pod stałym adresem lokalnym.

Aktualny stan trybu serwisowego sygnalizowany jest dedykowaną diodą LED na obudowie urządzenia, świecącą światłem ciągłym przez cały czas aktywności serwera. Krótkotrwałe zmiany stanu (uruchamianie, zatrzymywanie, ponowne uruchomienie po sekwencji alarmowej) sygnalizowane są charakterystycznym, kilkukrotnym mignięciem diody jako wizualne potwierdzenie zarejestrowania komendy użytkownika.

Interfejs udostępnia następujące funkcjonalności:

- Podgląd bieżącego stanu systemu, stanu poszczególnych czujników, aktualnego napięcia ogniwa zasilającego, informacje diagnostyczne;
- Przegląd historii zarejestrowanych przejazdów oraz wykrytych zdarzeń bezpieczeństwa, parsowanych bezpośrednio z plików na karcie SD;
- Zarządzanie kontaktem alarmowym - dodawanie, edycja i usuwanie numeru telefonu odbiorcy powiadomień SMS;
- Zakładkę deweloperską, przeznaczoną do prac kalibracyjnych i diagnostycznych, umożliwiającą edycję parametrów konfiguracyjnych algorytmów decyzyjnych oraz weryfikację działania poszczególnych endpointów aplikacji webowej.

Wybór architektury punktu dostępu Wi-Fi zamiast trybu klienckiego (podłączenie do istniejącej sieci) wynika z założenia projektowego o niezależności urządzenia od zewnętrznej infrastruktury sieciowej. Rowerzysta może uzyskać dostęp do interfejsu w dowolnej lokalizacji, w tym podczas wycieczki poza zasięgiem znanych sieci. Ograniczenie dostępu wyłącznie do trybu serwisowego, aktywowanego mechanicznie, redukuje powierzchnię ataku na urządzenie oraz eliminuje zużycie energii związane z utrzymaniem aktywnego modułu radiowego Wi-Fi podczas normalnej eksploatacji.

3.1.5. Zarządzanie energią

Urządzenie zasilane bateryjnie, przeznaczone do pracy w trybie ciągłym z okresami wielodniowego czuwania, wymaga przemyślanej strategii zarządzania energią. Niniejszy podrozdział definiuje wymagania funkcjonalne tej warstwy systemu na poziomie obserwowanego zachowania urządzenia, abstrahując od szczegółów rozwiązania sprzętowego.

System wymaga źródła zasilania o charakterystyce dopasowanej do dwóch jednoczesnych ograniczeń. Pierwszym jest pojemność energetyczna pozwalająca na nieprzerwaną pracę przez czas odpowiadający realistycznym scenariuszom użytkowania - całodniowej wycieczki rowerowej w trybie aktywnej rejestracji oraz tygodniowego okresu przechowywania urządzenia w stanie czuwania. Drugim jest zdolność do dostarczenia krótkotrwałych impulsów prądowych o znacznej amplitudzie, generowanych przez warstwę komunikacji komórkowej podczas nawiązywania połączenia z siecią i transmisji wiadomości alarmowej. Spełnienie obu wymagań jednocześnie wymaga zastosowania ogniwa o niskiej impedancji wewnętrznej, gdyż samo zwiększenie pojemności nominalnej nie rozwiązuje problemu chwilowych spadków napięcia podczas impulsów prądowych.

System musi dystrybuować energię z pojedynczego ogniwa zasilającego do podzespołów o różnych wymaganiach napięciowych, przy czym napięcie ogniwa zmienia się w znacznym zakresie w funkcji stanu naładowania. Wymaga to zastosowania warstwy konwersji napięcia, zapewniającej stabilne napięcia robocze niezależnie od stanu rozładowania ogniwa - w szczególności dla podzespołu komunikacji komórkowej, którego stabilność pracy bezpośrednio warunkuje niezawodność funkcji alarmowej.

System musi udostępniać funkcjonalność doładowywania ogniwa bez konieczności jego demontażu i bez wykorzystania specjalistycznego oprzyrządowania. Doładowanie powinno być możliwe za pośrednictwem powszechnie dostępnego interfejsu.

System musi udostępniać użytkownikowi informację o bieżącym stanie energetycznym ogniwa, prezentowaną w interfejsie serwisowym oraz utrwalaną w plikach sesji w celu retrospektywnej analizy charakterystyk poboru.

Komponenty systemu o znacznym poborze energii, których funkcjonalność wymagana jest jedynie w krótkich, ściśle określonych przedziałach czasowych - w szczególności podzespół komunikacji komórkowej, wykorzystywany wyłącznie podczas wysyłki powiadomień alarmowych - powinny pozostawać odłączone od źródła zasilania poza okresami swojej aktywności. Selekttywne włączanie tych komponentów stanowi kompromis pomiędzy dodatkowym opóźnieniem związanym z czasem ich inicjalizacji a wielokrotnym obniżeniem średniego poboru prądu w okresach normalnej eksploatacji.

W okresach długotrwałego bezruchu urządzenia, gdy żadna z funkcji rejestracyjnych nie jest aktywna, system powinien przechodzić w tryb minimalnego poboru energii, ograniczający aktywność do peryferiów odpowiedzialnych za reakcję na zdarzenie wybudzające. Powrót do trybu pełnej funkcjonalności powinien następować automatycznie w odpowiedzi na fizyczne poruszenie urządzenia. Mechanizm wybudzenia powinien charakteryzować się znikomym własnym poborem energii w stanie spoczynku, gdyż przy zakładanym czasie czuwania liczonym w tygodniach, nawet prąd rzędu pojedynczych mikroamperów przekłada się na zauważalne straty energetyczne.

System spełnia następujące wymagania energetyczne, definiujące oczekiwane charakterystyki eksploatacyjne urządzenia:

- Aktywna rejestracja przejazdu - minimum 24 godziny nieprzerwanej pracy ze wszystkimi modułami akwizycji danych aktywnymi;
- Tryb głębokiego uśpienia z monitorowaniem wybudzenia - minimum 7 dni utrzymania zdolności do wybudzenia;
- Tryb serwisowy z aktywnym interfejsem konfiguracyjnym - minimum 30 minut nieprzerwanej pracy.

Powyższe założenia odpowiadają realistycznym scenariuszom użytkowania urządzenia: pełnodniowej wycieczce rowerowej w trybie aktywnym, tygodniowemu okresowi przechowywania roweru pomiędzy używaniem (na przykład w garażu lub piwnicy) oraz pojedynczej sesji konfiguracyjnej. Weryfikację realnych charakterystyk poboru prądu i czasu pracy zaprojektowanego prototypu opisano w Rozdziale 7.

3.2. Wymagania нефunkcjonalne

Wymagania нефunkcjonalne określają jakościowe charakterystyki systemu, definiujące „w jaki sposób” urządzenie ma realizować swoje funkcje, w odróżnieniu od wymagań funkcjonalnych opisujących „co” ma być zrealizowane. Obejmują one zagadnienia czasu reakcji, niezawodności, odporności środowiskowej, modułowości konstrukcji oraz łatwości konserwacji i kalibracji. W przypadku urządzenia wbudowanego, eksploatowanego w warunkach polowych i pełniącego funkcję ratunkową, wymagania нефunkcjonalne mają porównywalnie istotne znaczenie jak wymagania funkcjonalne, ponieważ to one decydują o praktycznej użyteczności rozwiązania.

3.2.1. Charakterystyka czasowa reakcji

Reakcja systemu na zdarzenie krytyczne jest procesem wielofazowym, w którym poszczególne etapy charakteryzują się odmiennymi wymaganiami czasowymi i podlegają różnym ograniczeniom technologicznym. Sformułowanie pojedynczego, ogólnego wymagania dotyczącego "czasu reakcji" byłoby w tym przypadku nieadekwatne - z tego powodu wymagania czasowe sformułowano oddzielnie dla trzech etapów procedury alarmowej.

Pierwszym etapem reakcji jest detekcja zdarzenia w warstwie decyzyjnej oprogramowania. Ze względu na trójfazową strukturę algorytmu detekcji upadku, charakterystyka czasowa tego etapu nie jest pojedynczą liczbą, lecz sumą trzech składników odpowiadających kolejnym fazom algorytmu.

Faza pierwsza (wykrycie piku przyspieszenia) - powinna zakończyć się w czasie nieprzekraczającym jednej sekundy od fizycznego wystąpienia zdarzenia. Detekcja opiera się na chwilowym pomiarze magnitudy przyspieszenia, którego pojedyncza próbka, przekraczająca zdefiniowany próg, wystarcza do uruchomienia dalszej analizy. Faktyczna latencja zależy w tym przypadku przede wszystkim od częstotliwości próbkowania modułu inercyjnego - przy częstotliwości próbkowania na poziomie kilkudziesięciu próbek na sekundę latencja pomiarowa nie przekracza pojedynczych dziesiątek milisekund, a do całkowitego czasu reakcji dochodzi jedynie narzut wykonania algorytmu klasyfikującego.

Faza druga (obserwacja stanu dynamiki ruchu po wystąpieniu piku) - stanowi z natury rzeczy element wprowadzający opóźnienie do całkowitej latencji detekcji. Algorytm musi odczekać określony czas po wystąpieniu piku, aby zweryfikować, czy stan unieruchomienia utrzymuje się, czy też dynamika ruchu wraca do normy (co oznaczałoby, że rowerzysta zachował kontrolę nad pojazdem). Długość tej fazy stanowi parametr konfigurowalny, wczytywany z plików konfiguracyjnych na karcie SD podczas rozruchu urządzenia. Wybór optymalnej wartości stanowi kompromis pomiędzy czułością - krótsza obserwacja przekłada się na szybszą reakcję - a odpornością na fałszywe alarmy, których ryzyko maleje wraz z wydłużeniem fazy obserwacji.

Faza trzecia (klasyfikacja końcowa zdarzenia) - wprowadza pomijalny narzut czasowy w porównaniu z poprzednimi fazami, ponieważ sprowadza się do oceny zgromadzonych wskaźników i wydania decyzji binarnej.

Całkowita latencja detekcji upadku jest zatem sumą latencji wykrycia piku (rzędu dziesiątek milisekund) oraz długości fazy obserwacji (rzędu pojedynczych sekund). Wartości te muszą być uwzględnione przy planowaniu okna potwierdzenia alarmu - odliczanego dopiero po zakończonej klasyfikacji - oraz przy interpretacji wyników prób terenowych w Rozdziale 7

Drugim etapem reakcji jest okno potwierdzenia alarmu, którego długość ustalono na 30 sekund. Jest to parametr projektowy, wynikający z kompromisu pomiędzy szybkością powiadomienia osób kontaktowych a możliwością odwołania fałszywego alarmu przez użytkownika. Wartość ta nie podlega weryfikacji w sensie technicznym, lecz powinna zostać poddana ocenie empirycznej w trakcie prób terenowych - zbyt krótkie okno utrudnia anulowanie, zbyt długie opóźnia reakcję służb ratunkowych.

Trzecim etapem jest wysyłka powiadomienia SMS, której charakterystyka czasowa zależy w pierwszej kolejności od warunków zewnętrznych poza kontrolą systemu: jakości pokrycia sieci GSM w miejscu wystąpienia zdarzenia, obciążenia sieci operatora oraz dostępności świeżego fixa GPS. Typowy czas dostarczenia wiadomości w sieci GSM mieści się w zakresie od kilku do kilkunastu sekund. W warunkach prototypowych nie zdefiniowano twardego limitu czasowego dla tego etapu, lecz wprowadzono mechanizm logowania statusu wysyłki oraz mechanizm ponowień, zapewniający dostarczenie wiadomości nawet w przypadku przejściowych problemów z dostępem do sieci.

3.2.2. Niezawodność rejestracji danych

System pełni funkcję rejestratora zdarzeń, co czyni integralność zapisywanych danych krytycznym wymaganiem jakościowym - utrata zarejestrowanych przejazdów lub uszkodzenie struktury plików na karcie SD zniweczyłyby wartość urządzenia jako narzędzia analitycznego. Niniejszy podrozdział definiuje wymagania dotyczące mechanizmów ochrony integralności danych oraz strategii obsługi sytuacji awaryjnych.

Warstwę zapisu zaprojektowano z uwzględnieniem trzech mechanizmów odpornościowych.

- Pierwszym mechanizmem jest synchronizacja systemu plików co dziesięć rekordów, ograniczająca rozmiar danych potencjalnie utraconych w wyniku nagłej utraty zasilania do co najwyżej dziesięciu próbek bieżącej sesji.
- Drugim jest odrębne traktowanie danych krytycznych i pomocniczych - surowy log w formacie CSV jest synchronizowany na bieżąco, podczas gdy pliki podsumowujące (meta.json, gpx) generowane są w trybie wsadowym przy zakończeniu sesji, co oznacza, że ich potencjalna utrata nie wpływa na integralność danych pomiarowych.

3.2.3. Odporność środowiskowa

Urządzenie przeznaczone do montażu na rowerze podlega ekspozycji na warunki środowiskowe odmienne od typowych dla elektroniki konsumenckiej - zmienne temperatury, opady atmosferyczne, wibracje pochodzące od nierówności nawierzchni oraz mechaniczne wstrząsy. Z uwagi na prototypowy charakter konstrukcji, przyjęto pragmatyczne wymagania dostosowane do najczęściej spotykanych warunków eksploatacji rekreacyjnej.

Zakres temperatur pracy aktywnej ustalono na 0°C do +40°C, odpowiadając typowym warunkom rekreacyjnej jazdy rowerem w polskich szerokościach geograficznych. Zakres temperatur przechowywania określono jako -10°C do +50°C, uwzględniając okresy zimowego przechowywania roweru w nieogrzewanych pomieszczeniach oraz letniej ekspozycji na pełne nasłonecznienie.

Odporność na wibracje i wstrząsy mechaniczne, generowane przez nierówności nawierzchni oraz przenoszone z ramy roweru, wynika ze świadomie zaprojektowanej dwupoziomowej strategii montażu modułów. Pierwszym poziomem są złącza pinowe (*goldpin*), zapewniające zarówno połączenie elektryczne, jak i podstawową stabilizację mechaniczną poprzez siły tarcia w gniazdach. Drugim poziomem, zastosowanym dla modułów o większej masie lub wymiarach wykorzystano tulejki dystansowe z tworzywa sztucznego. Tulejki te utrzymują stałą wysokość modułu nad płytą bazową oraz zapewniają punkty mocowania. Konstrukcja taka łączy elastyczność montażu modułowego - umożliwiającą wymianę pojedynczego modułu w razie awarii - z mechaniczną stabilnością wystarczającą dla przewidywanych warunków eksploatacji.

3.2.4. Modułowość konstrukcji

Prototyp pełni funkcję dowodu koncepcji i nie podlega wymaganiom typowym dla produktu rynkowego - minimalizacji rozmiaru, optymalizacji kosztu jednostkowego ani certyfikacjom zgodności elektromagnetycznej. Z tego powodu w doborze architektury sprzętowej priorytetem stało się wsparcie procesu iteracyjnego projektowania i diagnostyki, a nie ostateczna integracja konstrukcji.

Prototyp zaprojektowano w architekturze opartej na komercyjnie dostępnych modułach elektronicznych (*commercial off-the-shelf*, COTS), łączonych ze sobą poprzez autorsko zaprojektowaną płytkę drukowaną pełniącą funkcję płyty bazowej. Każdy moduł funkcjonalny - moduł nawigacyjny GPS, czujnik inercyjny, czujnik środowiskowy, moduł GSM, czytnik kart microSD, ekspander portów oraz przetwornica - występuje jako odrębna jednostka konstrukcyjna, podłączana do płyty bazowej.

Architektura modułowa zapewnia trzy istotne korzyści: możliwość szybkiej wymiany pojedynczego modułu w przypadku jego uszkodzenia bez konieczności reorganizacji całego systemu, możliwość iteracyjnego testowania alternatywnych komponentów bez modyfikacji pozostałych warstw oraz przejrzystość konstrukcji ułatwiającą diagnostykę i analizę problemów na poziomie poszczególnych podzespołów.

Świadomie przyjęto przy tym, że architektura modułowa nie jest rozwiązaniem docelowym dla potencjalnej wersji produkcyjnej urządzenia. Integracja funkcjonalności poszczególnych modułów na pojedynczej, gęsto zaprojektowanej płytce drukowanej umożliwiłaby istotne zmniejszenie wymiarów urządzenia, redukcję kosztu jednostkowego, poprawę charakterystyk kompatybilności elektromagnetycznej oraz wyeliminowanie zawodnych połączeń mechanicznych pomiędzy płytkami. Kierunek ten wskazano w Rozdziale 8.4 jako jedną z głównych ścieżek rozwoju projektu.

3.2.5. Łatwość konserwacji i kalibracji

System zaprojektowano z uwzględnieniem realiów eksploatacji prototypu badawczego, w której bieżąca konserwacja i kalibracja stanowią naturalny element pracy z urządzeniem. Niniejszy podrozdział grupuje wymagania dotyczące mechanizmów wspierających te zadania.

Trzy mechanizmy wbudowano w architekturę systemu w celu wsparcia konserwacji i kalibracji.

- Pierwszym mechanizmem jest rozdzielenie kodu od konfiguracji - wszystkie parametry decyzyjne (progi detekcji uderzenia i upadku, czasy oczekiwania, częstotliwości próbkowania, kontakty alarmowe) wczytywane są z plików konfiguracyjnych w formacie JSON na karcie SD

podczas rozruchu urządzenia. Modyfikacja parametru nie wymaga rekompilacji oprogramowania ani podłączenia urządzenia do środowiska deweloperskiego - wystarczy edycja odpowiedniego pliku tekstowego lub użycie zakładki deweloperskiej w interfejsie serwisowym;

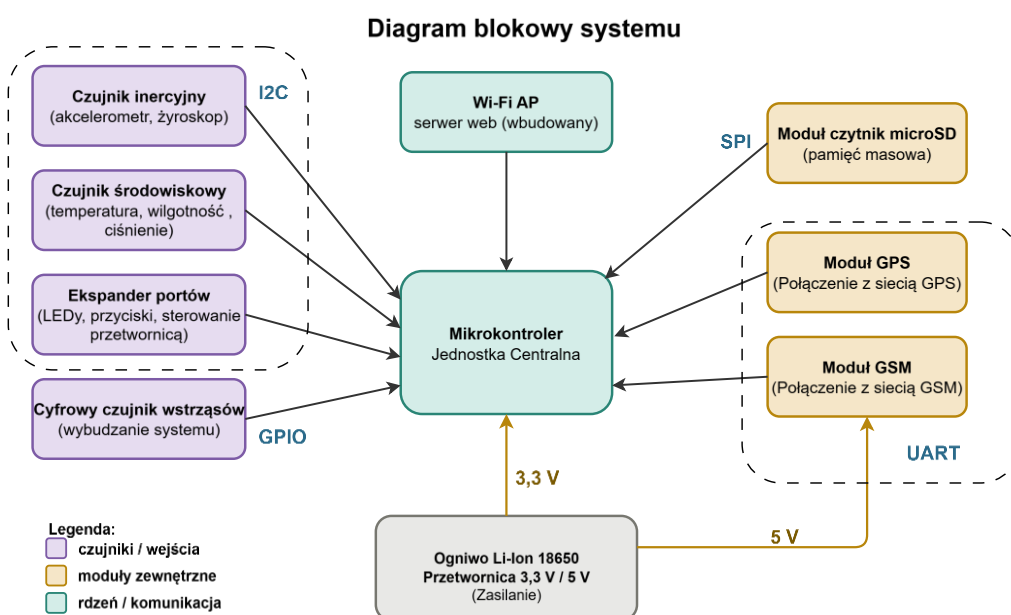
- Drugim mechanizmem jest strukturalne logowanie zdarzeń diagnostycznych - system rejestruje na karcie SD nie tylko dane pomiarowe, ale także zdarzenia diagnostyczne dotyczące inicjalizacji modułów, statusu komunikacji z modemem GSM, błędów akwizycji danych GPS oraz prób wysyłki wiadomości alarmowych. Logi te umożliwiają retrospektywną analizę zachowania systemu po zakończeniu sesji, bez konieczności podłączania urządzenia do komputera podczas eksploatacji;
- Trzecim mechanizmem jest dedykowana infrastruktura kalibracyjna w postaci modułu rejestratora wstrząsów umożliwiającego zbieranie surowych danych inercyjnych z wysoką częstotliwością. Dane te stanowią podstawę dla iteracyjnego dostrajania progów algorytmów detekcyjnych w oparciu o rzeczywiste sygnały pomiarowe, a nie wyłącznie wartości szacunkowe lub literaturowe. Metodologię i wyniki tego procesu kalibracyjnego opisano w Rozdziale 7.

3.3. Architektura systemu

Niniejszy podrozdział przedstawia architekturę zaprojektowanego urządzenia na poziomie koncepcyjnym, abstrahując od szczegółów technicznych warstwy sprzętowej i programowej, opisanych odpowiednio w rozdziałach 5 i 6. Celem jest ukazanie organizacji systemu jako spójnej całości - jego dekompozycji na bloki funkcjonalne, trybów pracy oraz sposobu, w jaki dane przepływają przez kolejne warstwy przetwarzania. Architekturę zaprezentowano w trzech komplementarnych ujęciach: strukturalnym (diagram blokowy), behawioralnym (diagram stanów) oraz informacyjnym (diagram przepływu danych).

3.3.1. Diagram blokowy systemu

Strukturę sprzętowo-funkcjonalną urządzenia przedstawiono na Rysunek 5. Centralnym elementem systemu jest mikrokontroler, pełniący funkcję jednostki nadzorującej pracę wszystkich pozostałych podzespołów. Wokół niego zorganizowano trzy grupy bloków funkcjonalnych, rozróżnione ze względu na pełnioną rolę oraz wykorzystywany interfejs komunikacyjny.



Rysunek 5. Diagram blokowy systemu (źródło: Opracowanie własne).

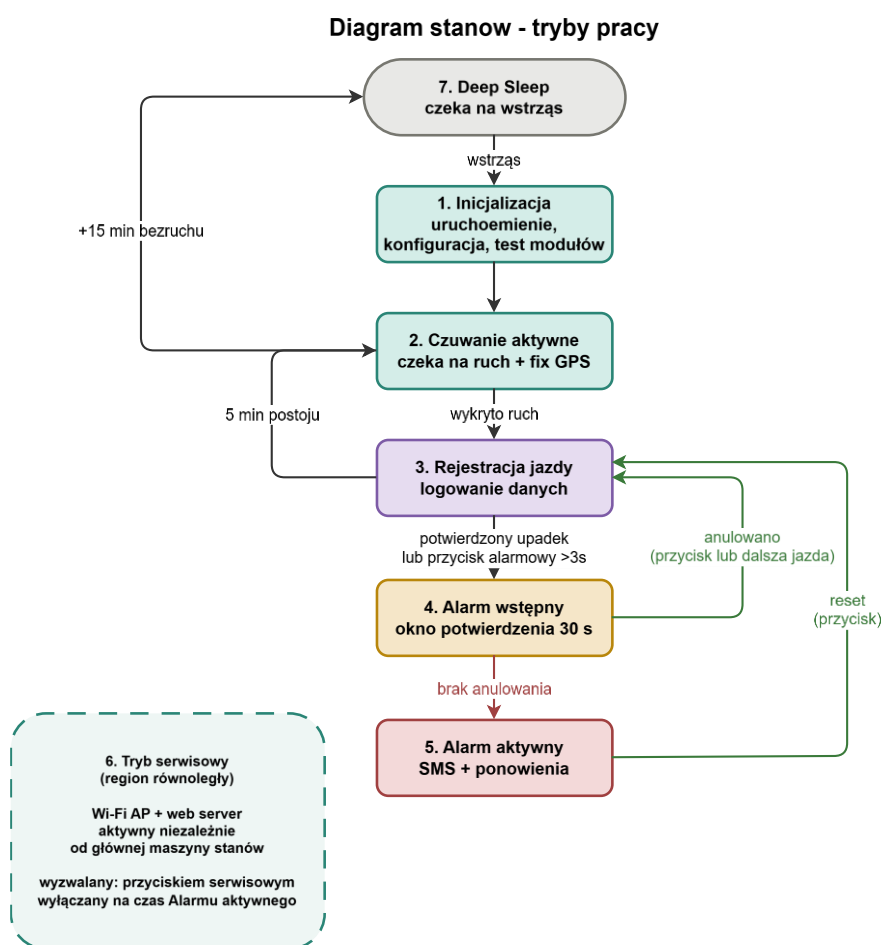
Pierwszą grupę stanowią czujniki i elementy wejściowe komunikujące się magistralą I²C - czujnik, czujnik środowiskowy oraz ekspander portów, obsługujący diody sygnalizacyjne, przyciski użytkownika, oraz uruchamianie przetwornicy. Współdzielenie pojedynczej magistrali I²C przez trzy układy peryferyjne pozwoliło na oszczędność wyprowadzeń mikrokontrolera.

Drugą grupę tworzą moduły zewnętrzne wymagające indywidualnych, dedykowanych interfejsów: moduł GPS, komunikujący się przez interfejs szeregowy UART, czytnik kart microSD, wykorzystujący magistralę SPI oraz moduł GSM, również obsługiwany przez osobny UART. Rozdzielenie tych modułów na osobne interfejsy podyktowane jest zarówno różnicami w protokołach komunikacyjnych, jak i wymaganiami przepustowości - w szczególności zapis na kartę SD przy częstotliwości próbkowania rejestratora wstrząsów wymaga większej przepustowości, niż zapewniłaby współdzielona magistrala.

Trzecią grupę stanowi blok zasilania, opisany szczegółowo w podrozdziale 3.5. Zawiera on ogniwo litowo-jonowe 18650 oraz przetwornicę, dostarczającą napięcie 5 V dla modemu GSM, podczas gdy pozostałe układy zasilane są napięciem 3,3 V ze stabilizatora zintegrowanego z mikrokontrolerem. Funkcję interfejsu użytkownika pełni serwer webowy, hostowany bezpośrednio przez mikrokontroler z wykorzystaniem wbudowanego radia Wi-Fi, aktywowany wyłącznie w trybie serwisowym.

3.3.2. Tryby pracy i przejścia między nimi

Zachowanie systemu opisano za pomocą diagramu stanów przedstawionego na Rysunek 6. Urządzenie funkcjonuje w jednym z pięciu sekwencyjnych trybów pracy głównej maszyny stanów, uzupełnionych o tryb głębokiego uśpienia oraz równoległy tryb serwisowy.



Rysunek 6. Diagram stanów pracy (źródło: Opracowanie własne).

Po wybudzeniu z trybu głębokiego uśpienia, wywołanym sygnałem z czujnika wstrząsów, system przechodzi przez fazę inicjalizacji, obejmującą wczytanie konfiguracji z karty SD oraz weryfikację dostępności poszczególnych modułów. Następnie urządzenie wchodzi w stan czuwania aktywnego, w którym oczekuje na spełnienie warunków rozpoczęcia rejestracji - wykrycia ruchu oraz uzyskania pozycji satelitarnej. Gdy warunki te zostaną spełnione, system przechodzi w tryb rejestracji jazdy, stanowiący podstawowy stan operacyjny urządzenia.

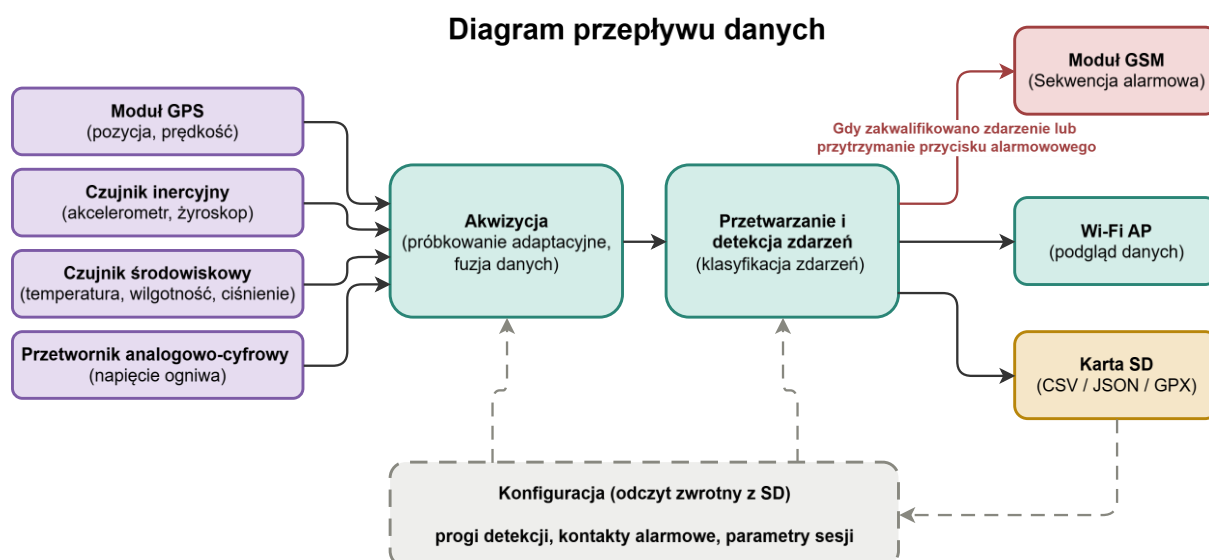
W trakcie rejestracji aktywny jest mechanizm detekcji zdarzeń krytycznych. Wykrycie wzorca upadku powoduje przejście do stanu alarmu wstępnego, w którym uruchamiane jest okno potwierdzenia o długości 30 sekund. Anulowanie alarmu - manualne lub automatyczne, poprzez wznowienie ruchu - przywraca system do trybu rejestracji. Brak anulowania skutkuje przejściem do stanu alarmu aktywnego, w którym realizowana jest procedura powiadomienia poprzez wiadomość SMS oraz monitoring po-alarmowy. Powrót z tego stanu następuje wyłącznie poprzez interwencję użytkownika lub wykrycie utrzymującego się ruchu. Możliwe jest też manualne wywołanie alarmu, poprzez przytrzymanie przycisku alarmowego (*panic-button*) w sytuacji kryzysowej poprzez przytrzymanie przycisku przez 3 sekundy.

Powrót do trybów o niższym poborze energii następuje na podstawie kryteriów czasowych. Utrzymujący się postój przez pięć minut powoduje zakończenie sesji jazdy i powrót do czuwania aktywnego, a dalszy bezruch - przejście w głębokie uśpienie. Świadomie wprowadzono opóźnienie pomiędzy zakończeniem rejestracji a uśpieniem urządzenia, zapobiegające przedwczesnej utracie sesji w sytuacji chwilowego zatrzymania.

Tryb serwisowy, obejmujący uruchomienie punktu dostępu Wi-Fi i serwera webowego, stanowi region równoległy względem głównej maszyny stanów. Oznacza to, że może być aktywny jednocześnie z dowolnym stanem operacyjnym - w szczególności umożliwia podgląd bieżących danych podczas trwającej rejestracji jazdy. Aktywacja trybu serwisowego następuje przez zwarcie wydzielonego wyprowadzenia ekspandera portów do masy i jest niezależna od przejść głównej maszyny stanów.

3.3.3. Przepływ danych

Sposób przetwarzania informacji w systemie zilustrowano na diagramie przepływu danych przedstawionym na Rysunek 7. Dane pochodzące ze źródeł pomiarowych - modułu GPS, czujnika inercyjnego, czujnika środowiskowego oraz przetwornika analogowo-cyfrowego monitorującego napięcie ogniwa - trafiają do warstwy akwizycji. Warstwa ta odpowiada za adaptacyjne próbkowanie oraz za fuzję danych z różnych źródeł, w szczególności wyznaczenie prędkości na podstawie połączenia informacji inercyjnej i satelitarnej.



Rysunek 7. Diagram przepływu danych (źródło: pracowanie własne).

Zebrane i scalone dane przekazywane są do warstwy przetwarzania, w której realizowana jest detekcja zdarzeń krytycznych oraz budowa kompletnego rekordu pomiarowego. Z tego punktu dane rozchodzą się trzema ścieżkami. Ścieżką podstawową, działającą w sposób ciągły, jest zapis rekordu na kartę microSD. Ścieżką drugą, dostępną na żądanie, jest udostępnienie danych poprzez serwer webowy w trybie serwisowym. Ścieżką trzecią, uruchamianą warunkowo wyłącznie po wykryciu i potwierdzeniu zdarzenia krytycznego, jest wygenerowanie i wysłanie wiadomości alarmowej SMS poprzez modem GSM. Strategia zarządzania energią

Urządzenie zasilane bateryjnie, przeznaczone do pracy w trybie ciągłym z wielodniowymi okresami czuwania, wymaga strategii zarządzania energią obejmującej zarówno warstwę sprzętową, jak i programową. Niniejszy podrozdział omawia przyjętą strategię w sposób całościowy, uzupełniając wymagania energetyczne sformułowane w podrozdziale 3.1.5 o uzasadnienie stojących za nimi decyzji projektowych. Weryfikację skuteczności tej strategii metodą pomiarową przedstawiono w rozdziale 7.

3.3.4. Strategia sterowanego włączania modemu GSM

Modemy GSM charakteryzują się wysokim poborem energii zarówno podczas aktywnej transmisji, jak i w stanie nominalnie nieaktywnym, w którym moduł musi utrzymywać rejestrację w sieci komórkowej, monitorować zasięg oraz odpowiadać na pakiety kontrolne sieci. Pobór ten, rzędu kilkudziesięciu miliamperów w stanie czuwania, stałby w jaskrawej sprzeczności z całościowymi celami energetycznymi urządzenia, zwłaszcza biorąc pod uwagę, że modem jest faktycznie wykorzystywany jedynie w krótkich, sporadycznych okresach - wyłącznie podczas wysyłki powiadomienia alarmowego.

Z tego powodu w architekturze sprzętowej przewidziano mechanizm umożliwiający kontrolowane włączanie i wyłączanie modemu w zależności od stanu pracy systemu. Realizacja polega na wykorzystaniu wyjścia cyfrowego ekspandera portów, połączonego z wejściem aktywującym przetwornicę zasilającą modem. W stanie domyślnym mikrokontroler utrzymuje przetwornicę w stanie wyłączonym, skutecznie odcinając modem od źródła zasilania. Aktywacja modemu następuje dopiero w chwili inicjowania sekwencji powiadomienia alarmowego, a po jej zakończeniu modem jest ponownie wyłączany.

3.3.5. Tryb głębokiego uśpienia i strategia wybudzania

Kluczowym elementem strategii energetycznej jest wykorzystanie trybu głębokiego uśpienia mikrokontrolera w okresach, gdy rower pozostaje nieruchomy. W trybie tym pobór prądu zredukowany jest do poziomu rzędu kilkunastu mikroamperów, co pozwala na utrzymanie zdolności do wybudzenia przez okres liczony w tygodniach bez istotnego rozładowania ogniwa.

Warunki wejścia w tryb głębokiego uśpienia podlegają kryteriom czasowym dobranym jako kompromis pomiędzy oszczędnością energii a użytecznością. Po upływie pięciu minut postoju system kończy aktywną sesję jazdy, a po dalszym okresie bezruchu przechodzi w głębokie uśpienie. Świadomie wprowadzono opóźnienie pomiędzy zakończeniem rejestracji a uśpieniem, zapobiegające przedwczesnemu uśpieniu podczas krótkich, planowanych zatrzymań.

Centralną decyzją projektową w tym obszarze był wybór źródła wybudzenia. Rozważano dwa podejścia: utrzymywanie modułu inercyjnego w trybie nasłuchu i wykorzystanie jego funkcji detekcji ruchu jako źródła przerwania wybudzającego, albo zastosowanie dedykowanego, pasywnego czujnika wstrząsów. Wybrano rozwiązanie drugie. Mechaniczny czujnik wstrząsów nie wymaga aktywnego zasilania w stanie spoczynku - reaguje na fizyczne poruszenie, zwierając styk i generując sygnał wybudzający.

Sygnał z czujnika wstrząsów doprowadzony jest do wyprowadzenia mikrokontrolera obsługującego wybudzanie z trybu głębokiego uśpienia. Wstrząs rowerem powoduje natychmiastowe wznowienie pracy systemu i przejście przez fazę inicjalizacji do trybu czuwania aktywnego, zgodnie z diagramem stanów przedstawionym na Rysunek 6.

4. Wykorzystane technologie i narzędzia

Niniejszy rozdział zawiera przegląd technologii, języków programowania, bibliotek oraz narzędzi wykorzystanych w ramach realizacji Czarnej skrzynki rowerowej, wraz z ich krótką charakterystyką w oderwaniu od konkretnego zastosowania w projekcie. Szczegółowy opis sposobu wykorzystania poszczególnych elementów w implementacji sprzętowej i programowej znajduje się w rozdziałach 5 i 6.

4.1. Środowisko i narzędzia programistyczne warstwy firmware

Warstwa firmware wymaga zarówno języka programowania dostosowanego do ograniczeń sprzętowych mikrokontrolera, jak i narzędzi wspierających proces kompilacji, wgrywania oraz debugowania kodu. Poniżej opisano wykorzystany język i framework programistyczny, a następnie środowisko budowania i IDE.

4.1.1. Język programowania i framework

C++ to obiektowo zorientowany język programowania o silnym typowaniu, opracowany na początku lat osiemdziesiątych XX wieku przez Bjarne Stroustrupa jako rozszerzenie języka C. Język łączy efektywność kodu niskopoziomowego z mechanizmami programowania obiektowego, w tym dziedziczeniem, polimorfizmem oraz szablonami klas. Współcześnie szeroko stosowany w systemach wbudowanych, oprogramowaniu systemowym oraz aplikacjach wymagających ścisłej kontroli zasobów sprzętowych [18].

Arduino-ESP32 to oficjalny framework programistyczny dla mikrokontrolerów rodziny ESP32 firmy Espressif Systems, oparty na ekosystemie i bibliotekach platformy Arduino. Framework dostarcza warstwę abstrakcji ułatwiającą programowanie peryferiów oraz integruje się z natywnym pakietem deweloperskim ESP-IDF, zapewniając dostęp do zaawansowanych funkcji platformy [19].

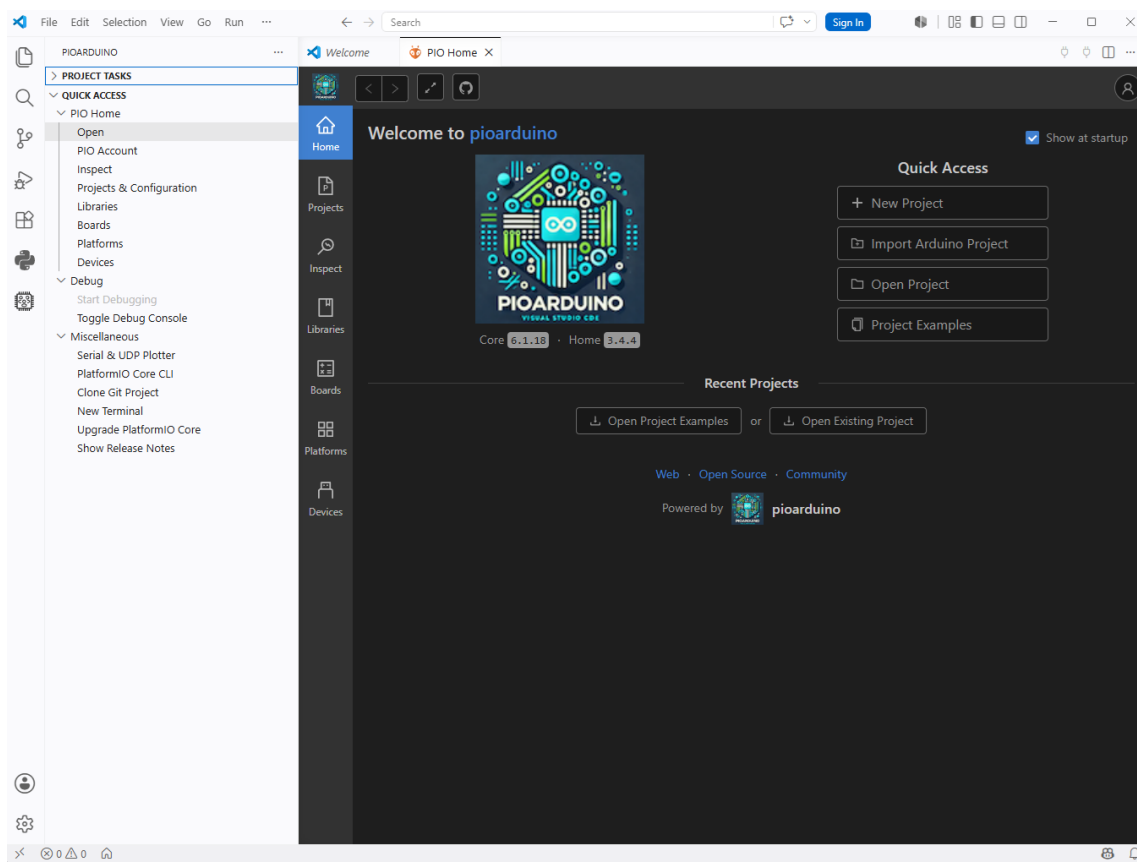
FreeRTOS to system operacyjny czasu rzeczywistego (*real-time operating system*) o otwartym kodzie źródłowym, opracowany przez Richarda Barry'ego, obecnie rozwijany pod opieką Amazon Web Services. System zapewnia wielowątkowe wykonanie zadań z deterministycznym szeregowaniem priorytetowym, mechanizmy synchronizacji oraz prymitywy komunikacji międzyzadaniowej. FreeRTOS stanowi standardowe jądro systemowe dla układów ESP32 firmy Espressif Systems [20].

4.1.2. Systemy budowania i IDE

Visual Studio Code to darmowy, otwartoźródłowy edytor kodu źródłowego opracowany przez firmę Microsoft, dostępny na systemy Windows, macOS oraz Linux. Oferuje wbudowane wsparcie dla kontroli wersji Git, debugowania oraz podświetlania składni i inteligentnego podpowiadania kodu (IntelliSense) dla wielu języków programowania. Funkcjonalność edytora można rozszerzać poprzez system wtyczek dostępnych w Visual Studio Code Marketplace, co umożliwia m.in. integrację z narzędziami takimi jak PlatformIO lub pioarduino [21].

PlatformIO to wieloplatformowe środowisko rozwoju oprogramowania dla systemów wbudowanych, dostarczające ujednolicony system kompilacji, zarządzania bibliotekami oraz konfiguracji projektów dla ponad tysiąca platform sprzętowych. Narzędzie integruje się z popularnymi edytorami kodu w postaci rozszerzeń, eliminując konieczność ręcznej konfiguracji łańcucha narzędziowego dla poszczególnych platform docelowych [22].

Pioarduino to rozszerzenie do Visual Studio Code stanowiące fork PlatformIO (w tym platformy Espressif32), zapewniający szybsze wsparcie dla aktualnych wersji ESP-IDF wykorzystywanych przez mikrokontrolery z rodziny ESP32, niż oficjalna platforma PlatformIO. W projekcie wykorzystane jako środowisko programistyczne dla firmware w ramach Visual Studio Code [23][24]. Ekran startowy środowiska pioarduino uruchomionego w edytorze Visual Studio Code przedstawiono na Rysunek 8.



Rysunek 8. Ekran startowy środowiska pioarduino w edytorze Visual Studio Code (źródło: opracowanie własne).

4.2. Biblioteki warstwy firmware

Poza samym językiem i frameworkiem, implementacja firmware korzysta z bibliotek dostarczających gotowe sterowniki peryferiów oraz funkcjonalności systemowe. Wyróżniono tu dwie grupy: biblioteki zewnętrzne instalowane jako zależności projektu oraz biblioteki standardowe wchodzące w skład frameworka Arduino-ESP32.

4.2.1. Biblioteki zewnętrzne

Implementacja wykorzystuje cztery biblioteki zewnętrzne dostarczające gotowe sterowniki peryferiów oraz strukturę systemu plików. Wszystkie biblioteki dostępne są w publicznym repozytorium PlatformIO i instalowane są automatycznie podczas kompilacji projektu. Wykaz bibliotek wraz z numerami wersji oraz zastosowaniem przedstawiono w Tabeli 1.

Tabela 1. Zewnętrzne biblioteki programowe

Biblioteka	Wersja	Zastosowanie
Adafruit BME280 Library [25]	2.3.0	Odczyt temperatury, wilgotności i ciśnienia atmosferycznego
Adafruit Unified Sensor [26]	1.1.15	Biblioteka pomocnicza wymagana przez sterownik BME280
TinyGPSPlus [27]	1.1.0	Parsowanie wiadomości NMEA otrzymywanych z modułu GPS
Adafruit SdFat Fork [28]	2.3.102	System plików FAT na karcie microSD, operacje na plikach i katalogach

Adafruit BME280 to sterownik czujnika środowiskowego BME280 firmy Bosch Sensortec, udostępniony przez firmę Adafruit Industries jako część ekosystemu bibliotek wspierających moduły tego producenta. Zapewnia inicjalizację układu poprzez magistralę I²C lub SPI, konfigurację trybów pracy (m.in. oversamplingu i filtracji) oraz odczyt skalibrowanych wartości temperatury, wilgotności względnej i ciśnienia atmosferycznego bezpośrednio w jednostkach fizycznych. Biblioteka implementuje algorytmy kompensacji temperatury opisane w nocie katalogowej producenta czujnika, eliminując konieczność ich ręcznej implementacji. Wymaga współpracy z biblioteką Adafruit Unified Sensor [25].

Adafruit Unified Sensor definiuje jednolity interfejs programistyczny (API) dla różnych typów czujników obsługiwanych przez biblioteki firmy Adafruit, niezależnie od konkretnego modelu układu czy sposobu komunikacji. Ujednolica sposób odczytu danych pomiarowych poprzez wspólną strukturę `sensors_event_t`, zawierającą m.in. typ czujnika, znacznik czasu oraz zmierzoną wartość. Dzięki temu sterowniki poszczególnych czujników (np. BME280) mogą być wykorzystywane w sposób spójny z innymi modułami korzystającymi z tego samego standardu. W projekcie pełni rolę zależności wymaganej przez bibliotekę obsługi czujnika BME280 [26].

TinyGPSPlus to lekka biblioteka języka C++ przeznaczona do parsowania komunikatów NMEA 0183, standardowego formatu danych wykorzystywanego przez moduły odbiorników GPS/GNSS. Dekoduje zdania NMEA (m.in. GGA, RMC) napływające strumieniowo z portu szeregowego, udostępniając gotowe wartości współrzędnych geograficznych, wysokości, prędkości, kursu oraz znacznika czasu, bez konieczności ręcznej implementacji parsera protokołu. Zaprojektowana z myślą o systemach wbudowanych o ograniczonych zasobach pamięciowych, przetwarza dane przyrostowo, znak po znaku, zamiast buforować całe komunikaty. Autorem i głównym opiekunem projektu jest Mikal Hart [27].

Adafruit SdFat Fork to fork biblioteki SdFat autorstwa Billa Greimana, dostosowany i utrzymywany przez firmę Adafruit Industries pod kątem kompatybilności z jej modułami kart microSD. Implementuje obsługę systemu plików FAT16/FAT32/exFAT na kartach SD, udostępniając interfejs zbliżony do standardowej biblioteki `SD.h`, lecz oferujący wyższą wydajność odczytu/zapisu oraz szerszy zestaw operacji na plikach i katalogach (m.in. tworzenie, usuwanie, iterację po zawartości katalogu). Komunikacja z kartą odbywa się poprzez interfejs SPI lub SDIO, w zależności od konfiguracji sprzętowej [28].

Sterowniki ekspandera MCP23008 oraz akcelerometra MPU-6500 zaimplementowano samodzielnie, bez wykorzystania zewnętrznych bibliotek. Decyzja ta podyktowana została dwoma względami. Pierwszym była chęć zachowania pełnej kontroli nad konfiguracją tych układów - zakresem pomiarowym akcelerometru, filtrami cyfrowymi (*Digital Low Pass Filter*), mapowaniem wyprowadzeń ekspandera oraz dostępem do rzadko używanych rejestrów konfiguracyjnych. Drugim - względna prostota protokołu komunikacyjnego, gdyż obsługa obu układów sprowadza się do podstawowych operacji odczytu i zapisu rejestrów przez magistralę I²C, nie uzasadniających wprowadzania dodatkowej zależności zewnętrznej.

4.2.2. Standardowe biblioteki framework Arduino-ESP32

Dodatkowo wykorzystano standardowe biblioteki frameworka Arduino-ESP32. Wykaz bibliotek oraz zastosowaniem przedstawiono w Tabeli 2.

Tabela 2. Standardowe biblioteki frameworku Arduino-ESP32

Biblioteka	Zastosowanie
WiFi.h [29]	Hostowanie sieci WiFi oraz punktu dostępu
Wire.h [30]	Obsługa interfejsu I ² C
SPI.h [31]	Obsługa interfejsu SPI
HardwareSerial.h [32]	Obsługa portu szeregowego UART

WiFi.h stanowi część frameworka Arduino-ESP32, udostępniając API do obsługi wbudowanego w mikrokontroler ESP32-C6 układu radiowego WiFi. Umożliwia konfigurację urządzenia zarówno w trybie stacji (klienta sieci), jak i punktu dostępowego (*Access Point*). Biblioteka zarządza również parametrami transmisji, takimi jak moc nadawania czy tryb oszczędzania energii karty radiowej [29].

Wire.h to standardowa biblioteka frameworka Arduino implementująca obsługę magistrali I²C (*Inter-Integrated Circuit*) w trybie master. Udostępnia funkcje inicjalizacji magistrali, adresowania urządzeń podrzędnych oraz odczytu i zapisu danych w formie sekwencji bajtów. Biblioteka obsługuje bufor transmisji oraz automatyczne generowanie sygnałów start/stop wymaganych przez protokół I²C [30].

SPI.h to standardowa biblioteka frameworka Arduino obsługująca komunikację poprzez magistralę SPI (*Serial Peripheral Interface*). Udostępnia funkcje konfiguracji parametrów transmisji (częstotliwości zegara, trybu SPI, kolejności bitów) oraz metody transferu pojedynczych bajtów i bloków danych. Zapewnia pełny duplex komunikacji, co pozwala na jednoczesny odczyt i zapis danych w ramach pojedynczej transakcji [31].

HardwareSerial.h udostępnia dostęp do sprzętowych kontrolerów UART mikrokontrolera. Umożliwia niezależną konfigurację parametrów transmisji (prędkości, liczby bitów danych, parzystości) na wielu niezależnych portach szeregowych jednocześnie, co odróżnia ją od podstawowej klasy Serial ograniczonej zwykle do portu diagnostycznego. Obsługuje transmisję z wykorzystaniem buforów sprzętowych oraz przerwań, minimalizując ryzyko utraty danych przy wysokiej częstotliwości nadawania [32].

esp_sleep.h to biblioteka natywnego API ESP-IDF, dostępna również z poziomu frameworka Arduino, udostępniająca funkcje zarządzania trybami oszczędzania energii mikrokontrolera (m.in. light sleep, deep sleep). Pozwala na konfigurację źródeł wybudzenia (*timer*, sygnał zewnętrzny, zmiana stanu GPIO). W trybie deep sleep większość peryferiów mikrokontrolera zostaje odłączona od zasilania, co pozwala na znaczącą redukcję poboru prądu kosztem utraty stanu pamięci RAM [33].

4.3. Technologie warstwy webowej

Interfejs konfiguracyjny oraz wizualizacja danych pomiarowych zrealizowane zostały jako aplikacja webowa. Poniżej opisano języki oraz biblioteki wykorzystane po stronie przeglądarki. Od struktury i stylu dokumentu, po logikę i wizualizację danych w JavaScript.

4.3.1. Języki znaczników i stylów

HTML (*HyperText Markup Language*) to standardowy język znaczników służący do definiowania struktury i zawartości dokumentów wyświetlanych w przeglądarce internetowej. Dokument HTML składa się z hierarchii elementów (znaczników) opisujących poszczególne fragmenty treści, takie jak nagłówki, akapity, tabele czy formularze. Aktualną wersją standardu jest HTML5, rozwijana i utrzymywana przez organizację WHATWG (*Web Hypertext Application Technology Working Group*) [34].

CSS (*Cascading Style Sheets*) to język arkuszy stylów wykorzystywany do definiowania prezentacji wizualnej dokumentów HTML, w tym układu, kolorystyki i responsywności interfejsu. Reguły CSS wiążą wartości stylów z elementami dokumentu poprzez selektory, umożliwiając oddzielenie warstwy prezentacji od struktury treści. Mechanizmy takie jak Flexbox czy CSS Grid pozwalają na tworzenie elastycznych, responsywnych układów strony dostosowujących się do rozmiaru ekranu urządzenia [35].

4.3.2. JavaScript i wykorzystane biblioteki

JavaScript to dynamicznie typowany, interpretowany język programowania, pierwotnie opracowany przez Brendana Eichę dla przeglądarki Netscape Navigator w 1995 roku. Współcześnie najpopularniejszy język programowania klienckiej warstwy aplikacji internetowych, wykonywany bezpośrednio w środowisku przeglądarki użytkownika. Standaryzowany jest jako ECMAScript przez organizację ECMA International, co zapewnia spójność implementacji pomiędzy różnymi przeglądarkami i silnikami wykonawczymi (m.in. V8, SpiderMonkey, JavaScriptCore). Język umożliwia manipulację strukturą dokumentu (DOM), obsługę zdarzeń użytkownika oraz asynchroniczną komunikację sieciową (m.in. poprzez Fetch API), co czyni go podstawowym narzędziem do budowy interaktywnych interfejsów webowych [36].

Leaflet.js to lekka, open-source'owa biblioteka JavaScript służąca do tworzenia interaktywnych map w przeglądarce internetowej, wykorzystywana m.in. do wizualizacji danych geoprzestrzennych, takich jak trasy zapisane w formacie GPX [37].

4.4. Środowisko i biblioteki analizy danych

Analiza danych pomiarowych realizowana jest w osobnej aplikacji desktopowej napisanej w Pythonie. Poniżej opisano język oraz bibliotekę wykorzystaną do budowy interfejsu graficznego, a następnie biblioteki odpowiedzialne za przetwarzanie i wizualizację danych.

4.4.1. Python i interfejs graficzny

Python to interpretowany, wysokopoziomowy język programowania ogólnego przeznaczenia, szeroko wykorzystywany w zadaniach związanych z przetwarzaniem i analizą danych ze względu na czytelną składnię oraz bogaty ekosystem bibliotek [38].

Tkinter to standardowa biblioteka języka Python służąca do tworzenia graficznych interfejsów użytkownika (GUI). Udostępnia zestaw gotowych widżetów (przyciski, pola tekstowe, listy, okna dialogowe) oraz mechanizm zdarzeń pozwalający na budowę interaktywnych aplikacji desktopowych. Jako element biblioteki standardowej nie wymaga instalacji dodatkowych zależności [39].

4.4.2. Biblioteki przetwarzania i wizualizacji danych

pandas Pandas to biblioteka języka Python dostarczająca struktury danych (m.in. DataFrame) oraz narzędzia do efektywnego wczytywania, filtrowania i przetwarzania danych tabelarycznych. Umożliwia m.in. wczytywanie danych z plików CSV (read_csv), parsowanie znaczników czasu, filtrowanie i grupowanie rekordów oraz obliczanie statystyk kroczących (rolling mean/rolling sum). Dzięki wektoryzacji operacji na danych zapewnia znacząco wyższą wydajność niż standardowe pętle Pythona przy przetwarzaniu dużych zbiorów danych [40].

NumPy to biblioteka języka Python dostarczająca wsparcie dla wielowymiarowych tablic oraz zwektoryzowanych operacji numerycznych. Udostępnia m.in. funkcje algebry liniowej, transformatę Fouriera (moduł numpy.fft), generowanie liczb losowych oraz szeroki zestaw funkcji statystycznych (percentyle, odchylenie standardowe, korelacje). Stanowi fundament większości bibliotek naukowych i analitycznych w ekosystemie Pythona, w tym pandas i matplotlib [41].

Matplotlib to biblioteka języka Python do tworzenia statycznych, animowanych oraz interaktywnych wizualizacji danych. Udostępnia interfejs pyplot wzorowany na środowisku MATLAB, a także moduł gridspec umożliwiający kompozycję układów wielopanelowych na pojedynczej figurze. Wspiera eksport wykresów do wielu formatów graficznych (PNG, SVG, PDF) oraz integrację z bibliotekami takimi jak pandas i NumPy [42].

4.5. Narzędzia wspomagające proces deweloperski

Oprócz technologii bezpośrednio wchodzących w skład firmware, aplikacji webowej i narzędzia analitycznego, w procesie tworzenia projektu wykorzystano narzędzia wspierające sam warsztat programistyczny kontrolę wersji oraz wspomaganie programistyczne oparte na sztucznej inteligencji.

4.5.1. Kontrola wersji

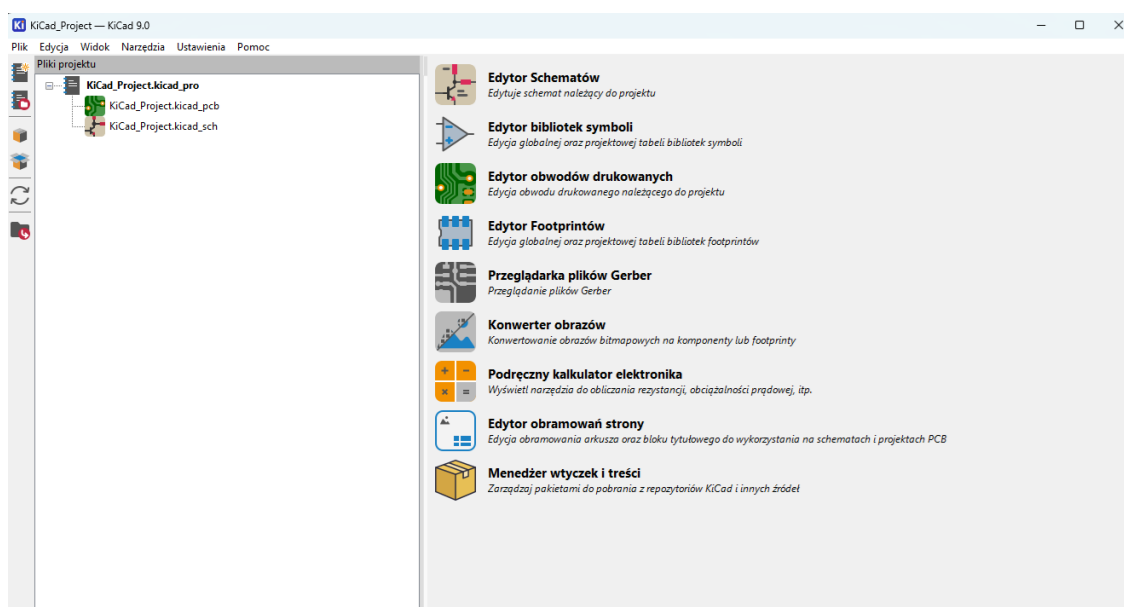
Git to rozproszony system kontroli wersji opracowany przez Linusa Torvaldsa, umożliwiający śledzenie zmian w kodzie źródłowym, pracę równoległą wielu programistów oraz zarządzanie wieloma gałęziami rozwoju projektu. Powszechnie stosowany standard w inżynierii oprogramowania, niezależny od konkretnego języka programowania ani platformy docelowej [43].

4.5.2. Wspomaganie programistyczne AI

Cursor to edytor kodu oparty na Visual Studio Code, rozszerzony o funkcje wspomagania programowania z wykorzystaniem dużych modeli językowych (LLM). Udostępnia m.in. inteligentne uzupełnianie kodu, generowanie fragmentów na podstawie opisu w języku naturalnym oraz czat kontekstowy operujący na zawartości otwartego projektu [44].

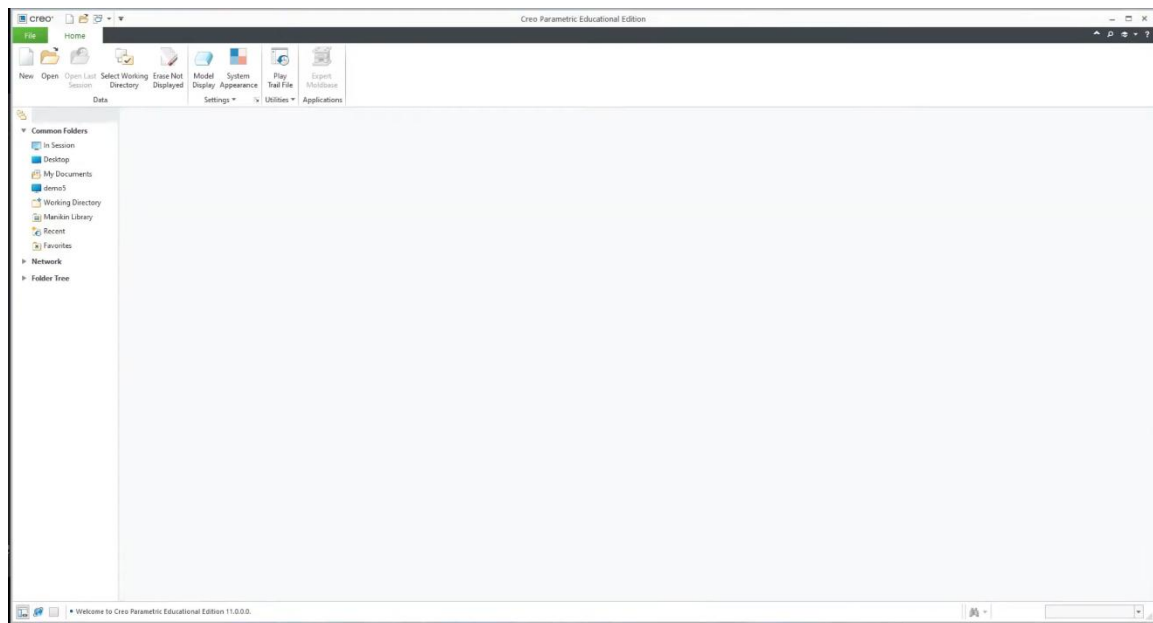
4.5.3. Narzędzia projektowania sprzętowego

KiCad to wieloplatformowy pakiet oprogramowania o otwartym kodzie źródłowym przeznaczony do projektowania obwodów elektronicznych, obejmujący narzędzia do tworzenia schematów elektrycznych, projektowania płytek drukowanych oraz generowania dokumentacji produkcyjnej. Stanowi alternatywę dla komercyjnych pakietów EDA w środowiskach akademickich oraz w projektach prototypowych [45]. Menedżer projektu środowiska KiCad, wraz z dostępnymi modułami do projektowania schematów oraz obwodów drukowanych, przedstawiono na Rysunek 9.



Rysunek 9. Menedżer projektu środowiska KiCad z zestawem dostępnych narzędzi (źródło: opracowanie własne).

PTC Creo Parametric oprogramowanie CAD (*Computer-Aided Design*) służące do parametrycznego projektowania modeli 3D oraz dokumentacji technicznej. Umożliwia tworzenie brył i powierzchni w oparciu o historię operacji (*feature-based modeling*), co pozwala na modyfikację wcześniejszych etapów projektu bez odtwarzania całego modelu od podstaw. W projektowaniu obudów urządzeń elektronicznych wykorzystywane jest m.in. do weryfikacji mieszczących się komponentów w obudowie (*fit check*) oraz przygotowania modeli pod druk 3D lub obróbkę CNC.[46]. Interfejs środowiska PTC Creo Parametric przedstawiono na Rysunek 10.



Rysunek 10. Ekran startowy środowiska projektowego PTC Creo Parametric (źródło: opracowanie własne).

5. Implementacja sprzętowa

Niniejszy rozdział dokumentuje fizyczną realizację urządzenia, omawiając kolejno dobór komponentów elektronicznych, schemat elektryczny, projekt płytki drukowanej oraz konstrukcję mechaniczną. Treść stanowi inżynierski opis tego, w jaki sposób zrealizowano decyzje koncepcyjne sformułowane w rozdziale 3, obejmując uzasadnienie wyboru poszczególnych podzespołów, topologię połączeń, fizyczne rozmieszczenie komponentów oraz mocowania mechaniczne. Warstwę programową, działającą na zaprojektowanej platformie sprzętowej, opisano w rozdziale 6.

Diagramy koncepcyjne przedstawione w podrozdziale 3.3 stanowią punkt odniesienia dla niniejszego rozdziału - wszystkie bloki funkcjonalne ujęte w diagramie blokowym znajdują tu swoje fizyczne odpowiedniki w postaci konkretnych modułów, magistral i ścieżek na płycie. Decyzje opisane w niniejszym rozdziale podporządkowano dwóm nadrzędnym kryteriom - spełnieniu wymagań sformułowanych w rozdziale 3 oraz wsparciu iteracyjnego procesu prototypowania, umożliwiającego diagnostykę i wymianę pojedynczych podzespołów na poziomie modułów. Świadomie zrezygnowano z dążenia do minimalizacji wymiarów oraz integracji komponentów na pojedynczej, gęsto zaprojektowanej płycie drukowanej, traktując prototyp jako dowód koncepcji, a nie wersję produkcyjną.

5.1. Dobór komponentów

Niniejszy podrozdział przedstawia uzasadnienie wyboru kluczowych technologii i komponentów zastosowanych w projekcie. Omówienie uporządkowano tematycznie, według roli pełnionej w systemie, koncentrując się na motywacji stojącej za poszczególnymi decyzjami. Dobór każdego elementu rozpatrywano w kontekście całościowych wymagań sformułowanych w podrozdziałach 3.1 i 3.2 - w szczególności ograniczeń energetycznych, wymiarowych oraz wymagań niezawodnościowych urządzenia przeznaczonego do pracy polowej.

5.1.1. Jednostka centralna

Sercem systemu jest mikrokontroler ESP32-C6 w postaci modułu Seeed Studio XIAO, który został przedstawiony na Rysunek 11. Wybór ten wynikał z kilku zbieżnych przesłanek, które łącznie czyniły go rozwiązaniem najlepiej dopasowanym do specyfiki projektu.

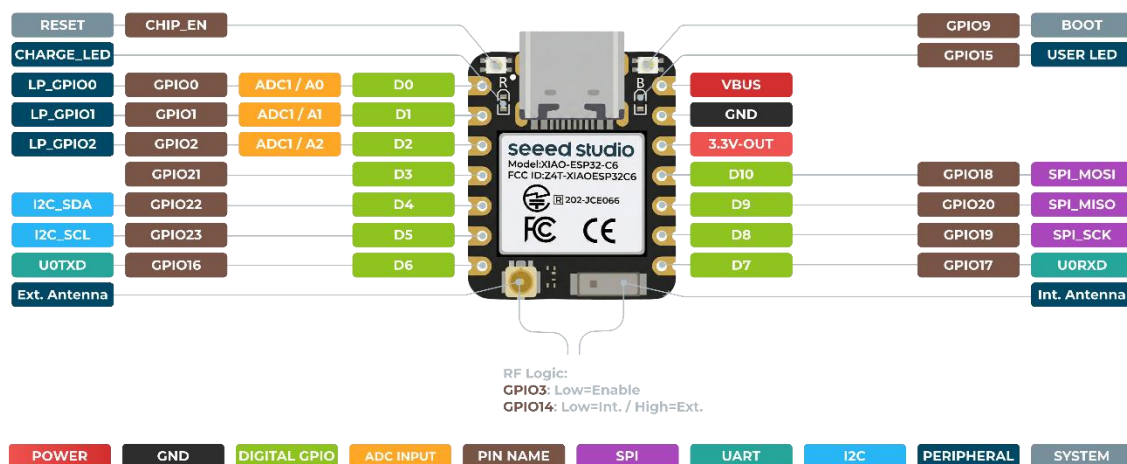
Pierwszą przesłanką była zintegrowana łączność Wi-Fi, umożliwiająca realizację lokalnego serwera webowego bez konieczności stosowania dodatkowego układu komunikacyjnego. Drugą był bardzo niski pobór prądu w trybie głębokiego uśpienia, na poziomie kilkunastu mikroamperów, kluczowy dla osiągnięcia wielodniowego czasu pracy w trybie czuwania. Trzecią - kompaktowa konstrukcja modułu XIAO, o powierzchni nieprzekraczającej kilku centymetrów kwadratowych, istotna w urządzeniu montowanym na rowerze. Czwartą, równie ważną, była obecność zintegrowanego układu zarządzania zasilaniem i ładowania ogniwa litowo-jonowego, który pozwolił uniknąć projektowania dedykowanego obwodu ładowania - zadania złożonego i obciążonego ryzykiem w kontekście bezpieczeństwa baterii litowych [47].



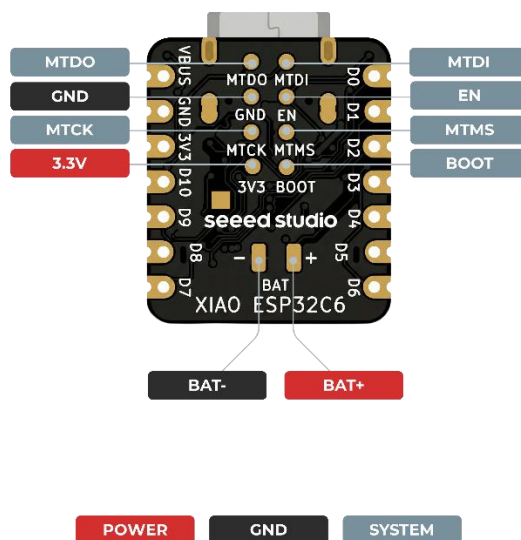
Rysunek 11. Mikrokontroler Seeed Studio XIAO ESP32C6 (źródło: [47]).

Dodatkowymi atutami były architektura rdzenia RISC-V, otwarta i dobrze udokumentowana, wbudowane złącze USB-C upraszczające programowanie i ładowanie, oraz szeroka dostępność bibliotek programistycznych w ekosystemie Arduino i ESP-IDF, co skróciło czas rozwoju oprogramowania. Zestawienie tych cech w pojedynczym, niedrogim module przeważało nad alternatywnymi platformami, z których żadna nie łączyła w sobie wszystkich wymienionych właściwości w równie kompaktowej formie [47].

Na Rysunek 12 i Rysunek 13 zostały przedstawione wyprowadzenia mikrokontrolera Seeed ESP32-C6 z obydwu stron płytki, wraz z ich funkcjami.



Rysunek 12. Wyprowadzenia pinów mikrokontrolera Seeed ESP32-C6, widok z przodu (źródło: [48]).



Rysunek 13. Wyprowadzenia pinów modułu ESP32-C6, widok od spodu (źródło: [48]).

5.1.2. Komunikacja alarmowa

Kanałem powiadamiania o zdarzeniach krytycznych jest wiadomość SMS wysyłana za pośrednictwem modemu GSM SIM800L, który został przedstawiony na Rysunek 14. Decydującą cechą tego rozwiązania jest niezależność od infrastruktury internetowej i od urządzeń pośredniczących - wiadomość SMS dociera na dowolny telefon komórkowy, bez konieczności instalowania aplikacji po stronie odbiorcy ani dostępu do sieci Wi-Fi czy transmisji danych w miejscu wystąpienia zdarzenia. W kontekście funkcji ratunkowej, realizowanej często w terenie pozamiejskim o ograniczonym pokryciu sieci, prostota i uniwersalność kanału SMS stanowi istotną zaletę [49].

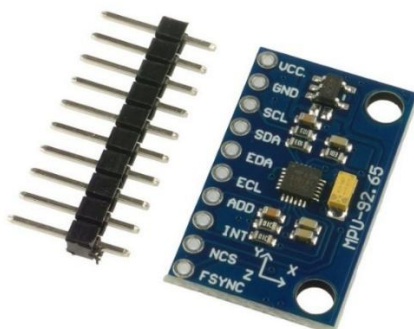


Rysunek 14. Moduł GPRS GSM SIM800L V2.0 (źródło: [49]).

Moduł SIM800L pracuje w sieci drugiej generacji (2G/GSM), co zapewnia najszersze pokrycie geograficzne oraz najniższy koszt komponentu i transmisji. Należy odnotować, że operatorzy sieci komórkowych stopniowo wygaszają sieci 2G, co w perspektywie wieloletniej stanowi ograniczenie tego rozwiązania. Migrację kanału alarmowego w stronę technologii nowszej generacji wskazano w rozdziale 8.4 jako jeden z kierunków dalszego rozwoju [49].

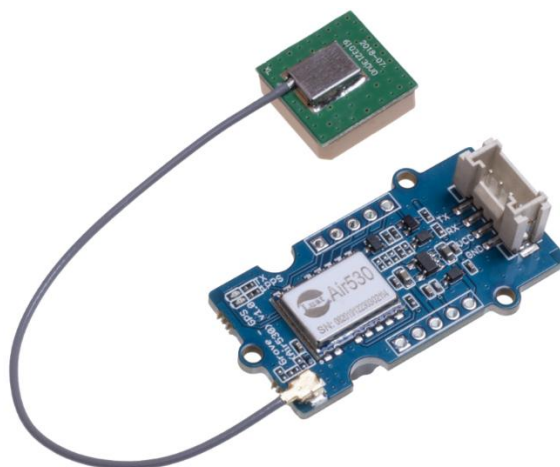
5.1.3. Czujniki i akwizycja danych

Dobór czujników podporządkowano funkcjom rejestratora i detektora zdarzeń. Pomiar dynamiki ruchu realizuje moduł inercyjny MPU-6500, który został przedstawiony na Rysunek 15. W projekcie wykorzystano funkcjonalność akcelerometru i żyroskopu, kierunek ruchu wyznaczany jest na podstawie fuzji danych inercyjnych i satelitarnych, co eliminuje potrzebę korzystania z kompasu magnetycznego, silnie podatnego na zakłócenia pochodzące od pracującego w bezpośrednim sąsiedztwie modemu GSM oraz przetwornicy impulsowej [49].



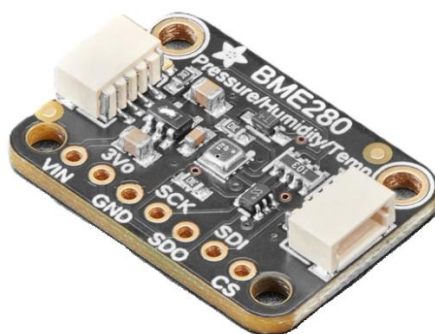
Rysunek 15. Moduł akcelerometru oraz żyroskopu MPU-6500 (źródło: [50]).

Pozycjonowanie zapewnia moduł nawigacji satelitarnej Air530Z, obsługujący wiele systemów nawigacyjnych jednocześnie, co skraca czas uzyskania pozycji i poprawia jej dokładność w warunkach miejskich. Moduł GPS Air530z został przedstawiony na Rysunek 16 [51].



Rysunek 16. Moduł GPS Air530z (źródło: [51]).

Pomiar parametrów środowiskowych realizuje czujnik BME280, dostarczający jednocześnie informacji o temperaturze, wilgotności względnej i ciśnieniu atmosferycznym. Pomiar ciśnienia, poza wartością samą w sobie, umożliwia szacowanie wysokości metodą barometryczną, stanowiącą uzupełnienie danych wysokościowych z modułu GPS. Czujnik środowiskowy został przedstawiony na Rysunek 17 [52].



Rysunek 17. Moduł czujnika parametrów środowiskowych Adafruit BME280 (źródło: [52]).

5.1.4. Magistrale komunikacyjne

Komunikację pomiędzy mikrokontrolerem a peryferiami zrealizowano z wykorzystaniem trzech standardów magistral.

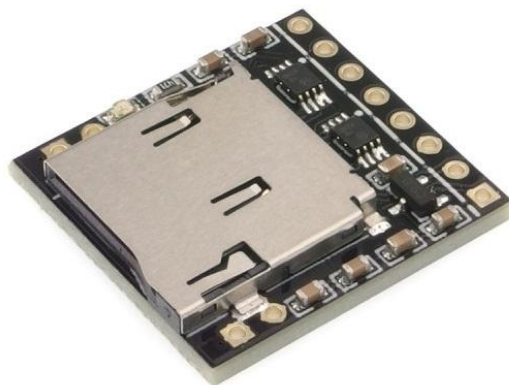
Magistralę I²C wybrano do obsługi czujnika inercyjnego, czujnika środowiskowego oraz ekspandera portów. Decydującą zaletą była możliwość podłączenia wielu układów do pojedynczej, dwuprzewodowej magistrali, co pozwoliło zaoszczędzić ograniczoną liczbę wyprowadzeń kompaktowego modułu XIAO. Umiarkowana przepustowość I²C jest w pełni wystarczająca dla okresowego odczytu rejestrów tych układów.

Magistralę SPI zarezerwowano dla czytnika kart microSD. Wybór ten podyktowany był wymaganiem wysokiej przepustowości zapisu - w szczególności rejestrator wstrząsów próbujący dane z częstotliwością 200 Hz generuje strumień danych, którego obsługa magistralą I²C byłaby utrudniona. Wydzielona magistrala SPI zapewnia przepustowość niezbędną do zapisu w czasie rzeczywistym bez ryzyka utraty próbek.

Interfejs szeregowy UART wykorzystano do komunikacji z dwoma niezależnymi modułami: nawigacyjnym Air530Z oraz modemem GSM SIM800L. Oba urządzenia komunikują się w sposób strumieniowy - moduł GPS nadaje cyklicznie ramki danych nawigacyjnych, a modem GSM obsługiwany jest zestawem poleceń tekstowych. Charakter tej komunikacji w naturalny sposób odpowiada specyfice interfejsu szeregowego.

5.1.5. Przechowywanie danych

Nośnikiem danych w systemie jest karta microSD o pojemności 32GB, wybrana ze względu na pojemność, powszechną dostępność oraz możliwość odczytu danych na komputerze osobistym bez specjalistycznego oprzyrządowania. Karta pełni w systemie podwójną rolę: magazynu danych pomiarowych oraz nośnika konfiguracji, z którego podczas rozruchu wczytywane są parametry sterujące pracą urządzenia. Wykorzystany adapter kart microSD został przedstawiony na Rysunek 18 [54].



Rysunek 18. Moduł czytnika kart MicroSD (źródło: [54]).

Dane rejestrowane są w trzech komplementarnych formatach plikowych, z których każdy odpowiada innej potrzebie. Format CSV przechowuje surowe dane pomiarowe w postaci tabelarycznej, stanowiąc kompletny materiał do analizy w zewnętrznych narzędziach obliczeniowych. Format JSON wykorzystywany jest do zapisu podsumowań sesji oraz plików konfiguracyjnych - jego struktura, czytelna zarówno dla człowieka, jak i dla maszyny, ułatwia parsowanie danych przez serwer webowy. Format GPX, będący standardem wymiany danych nawigacyjnych, umożliwia bezpośredni import zarejestrowanych tras do popularnych aplikacji mapowych i treningowych, bez konieczności konwersji. Współistnienie trzech formatów stanowi świadomą decyzję projektową, podporządkowaną zasadzie interoperacyjności - każdy z nich zaspokaja odmienną potrzebę, której pozostałe nie realizują równie skutecznie.

Wybór ekspandera portów MCP23008, komunikującego się magistralą I²C, podyktowany był ograniczoną liczbą wyprowadzeń dostępnych w module XIAO. Przeniesienie obsługi elementów interfejsu fizycznego: diod i przycisków na ekspander pozwoliło zachować wyprowadzenia mikrokontrolera dla magistral komunikacyjnych i funkcji krytycznych, takich jak wybudzanie z uśpienia. Ekspander portów spełnia też jedną dodatkową funkcję, steruje pinem EN przetwornicy TPS63020, który odpowiada za uruchamianie przetwornicy. Ekspander wyprowadzeń został przedstawiony na Rysunek 19 [55].



Rysunek 19. Ekspander wyprowadzeń MCP23008-E/P (źródło: [55]).

5.1.6. Zasilanie i konwersja napięcia

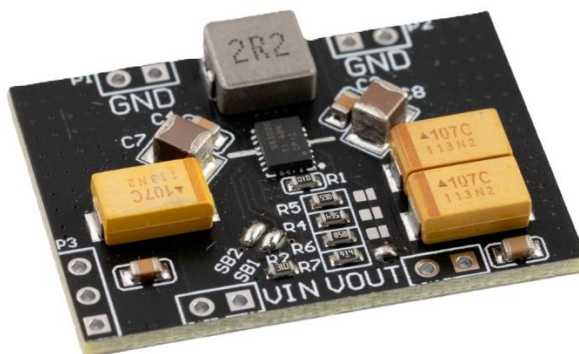
Zasilanie systemu oparto na pojedynczym ogniwie litowo-jonowym typu 18650 Sony US18650VTC6, które zostało przedstawione na Rysunek 20. Napięcie ogniwa zmienia się w zakresie od 4,2 V w stanie pełnego naładowania do około 2,5 V w stanie głębokiego rozładowania. Ta zmienność

napięcia stanowi istotne wyzwanie konstrukcyjne, ponieważ poszczególne podzespoły wymagają stabilnych napięć zasilania. Ogniwo ma pojemność równą 3000mAh.



Rysunek 20. Ogniwo litowo-jonowe US18650VTC6 (źródło: [56]).

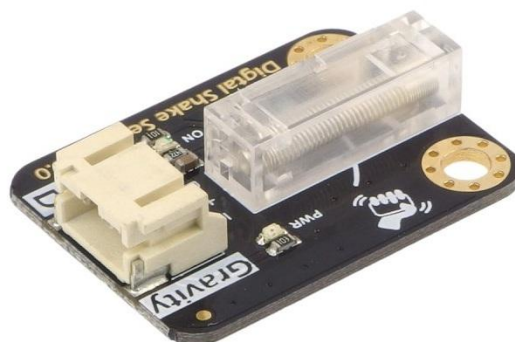
Większość układów cyfrowych zasilana jest napięciem 3,3 V dostarczonym przez stabilizator zintegrowany z modulem XIAO. Modem GSM SIM800L, wymagający wyższego i stabilnego napięcia oraz zdolności do dostarczenia wysokich impulsów prądowych, zasilany jest z przetwornicy (*buck-boost*) TPS63020 dostarczającej napięcie 5 V. Moduł z przetwornicą posiada możliwość zmiany napięcia na wyjściu przetwornicy, poprzez zwarcie odpowiedniej zworki obok rezystorów R5, R4, R6 i R7, przedstawiona konfiguracja na Rysunek 21 daje na wyjściu 5V. Wybór topologii *buck-boost* był tu rozstrzygający: prosta przetwornica podwyższająca napięcie nie zapewniłaby stabilnego napięcia wyjściowego w całym zakresie rozładowania ogniwa, podczas gdy układ *buck-boost* utrzymuje zadane napięcie niezależnie od tego, czy napięcie wejściowe jest wyższe, czy niższe od wyjściowego. Dzięki temu modem GSM zachowuje stabilność pracy nawet przy znacznie rozładowanym ogniwie, co bezpośrednio przekłada się na niezawodność funkcji alarmowej. Moduł z Przetwornicą został przedstawiony na Rysunek 21 [57].



Rysunek 21. Przetwornica Step-Down/Up 5V 3A TPS63020 (źródło: [57]).

Wybór wysokoprądowego ogniwa o niskiej impedancji wewnętrznej omówiono w wymaganiu impulsowym charakterem poboru prądu modemu GSM, sięgającego chwilowo wartości rzędu dwóch amperów podczas transmisji.

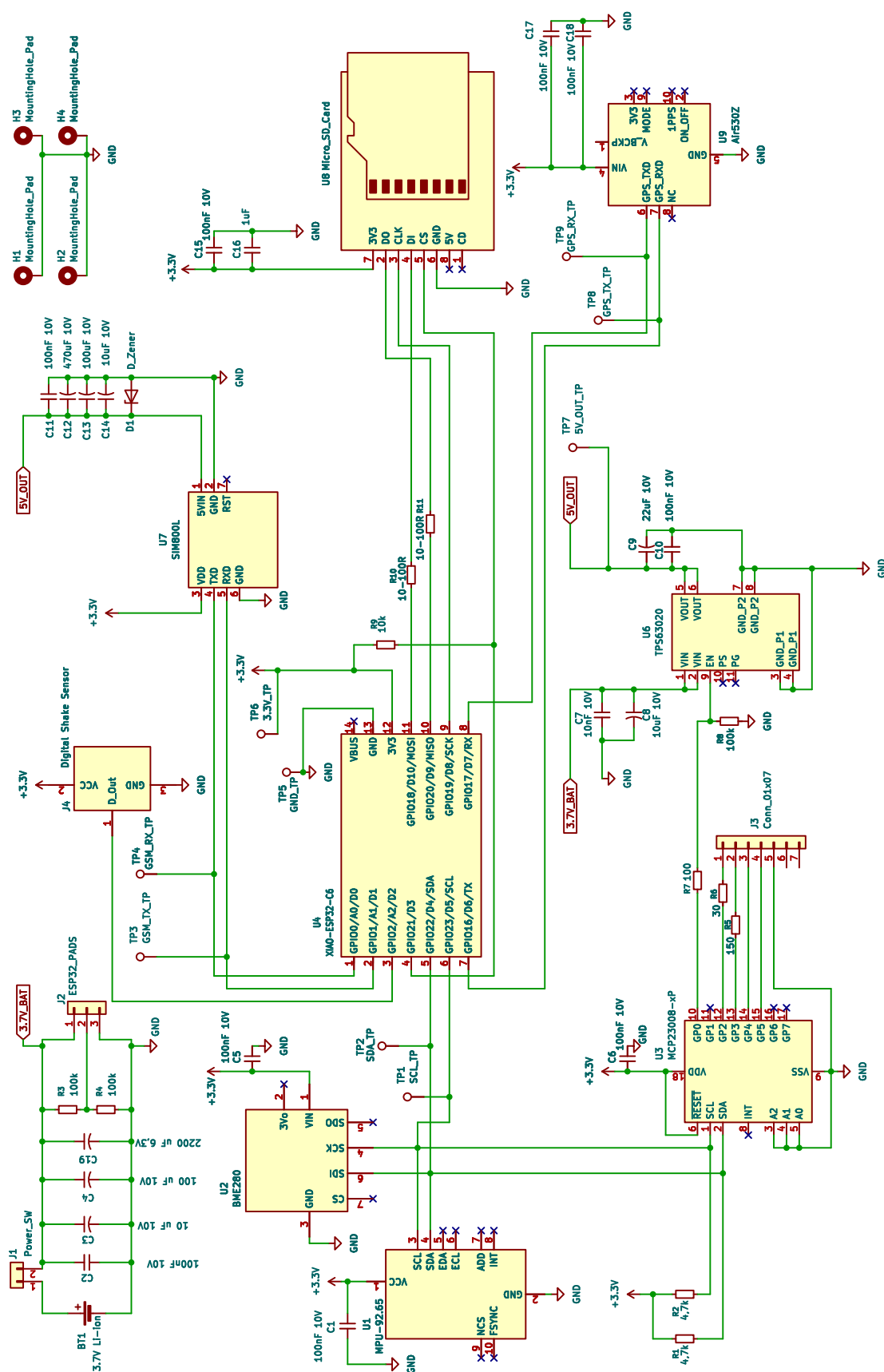
Funkcję źródła wybudzenia z trybu głębokiego uśpienia pełni cyfrowy czujnik wstrząsów SEN0289 firmy DFRobot. Decydującą cechą tego rozwiązania jest pasywny charakter pracy w stanie spoczynku - czujnik nie wymaga zasilania w przerwach między zdarzeniami wybudzającymi, w odróżnieniu od alternatywnego rozwiązania opartego na funkcji detekcji ruchu wbudowanej w moduł inercyjny MPU-6500, który nawet w trybie niskiego poboru utrzymuje pobór prądu rzędu pojedynczych mikroamperów. Przy zakładanym czasie czuwania liczonym w tygodniach różnica ta przekłada się na zauważalne straty energetyczne. Sygnał z wyjścia czujnika doprowadzony jest do wyprowadzenia mikrokontrolera obsługującego funkcję wybudzania z trybu głębokiego uśpienia - szczegóły topologii podłączenia opisano w podrozdziale 5.2.1. Cyfrowy czujnik wstrząsów został przedstawiony na Rysunek 22 [53].



Rysunek 22. Cyfrowy czujnik wstrząsów DFRobot SEN0289 (źródło: [53]).

5.2. Schemat elektroniczny

Schemat elektryczny urządzenia opracowano w środowisku KiCad EDA w wersji 9.0.7. Przedstawiono go w całości na Rysunek 23.



Rysunek 23. Schemat elektroniczny Czarnej Skrzynki Rowerowej (źródło: opracowanie własne).

5.2.1. Topologia magistral i połączeń

Jednostką centralną systemu jest moduł Seeed XIAO ESP32-C6, do którego doprowadzono wszystkie pozostałe podzespoły z wykorzystaniem czterech rodzajów interfejsów elektrycznych: magistrali I²C, magistrali SPI, dwóch niezależnych portów szeregowych UART oraz wyprowadzeń ogólnego przeznaczenia GPIO. Tabelę przyporządkowania konkretnych pinów modułu XIAO do poszczególnych funkcji oraz modułów przedstawiono w Tabeli 3.

Tabela 3. Zestawienie pinów mikrokontrolera ESP32-C6, wraz z ich funkcjami oraz modułów, do których są podłączone.

Pin Seeed XIAO ESP32C6	Pin modułu ESP32-C6	Wykorzystana funkcja pinu	Podłączony moduł
A0/D0	GPIO0	Sprzętowy UART0 (RxD)	Sim800L (TxD)
A1/D1	GPIO1	Sprzętowy UART0 (TxD)	Sim800L (RxD)
A2/D2	GPIO2	RTC_GPIO (Wybudzanie modułu)	Cyfrowy czujnik wstrząsów
D3	GPIO21	SPI (CS, <i>Chip Select</i>)	Moduł MicroSD (CS)
D4	GPIO22	I ² C (SDA)	MCP23008, MPU-6500, BME280
D5	GPIO23	I ² C (SCL)	MCP23008, MPU-6500, BME280
D6	GPIO16	Sprzętowy UART1 (TxD)	Air530Z (RxD)
D7	GPIO17	Sprzętowy UART1 (RxD)	Air530Z (TxD)
D8	GPIO19	SPI (CLK, <i>Clock</i>)	Moduł MicroSD (CLK)
D9	GPIO20	SPI (MISO, <i>Master In Slave Out</i>)	Moduł MicroSD (DO)
D10	GPIO18	SPI (MOSI, <i>Master Out Slave In</i>)	Moduł MicroSD (DI)
MTCK	GPIO6	ADC (Pomiar napięcia ogniwa)	Linia z dzielnika napięcia (R3/R4)

Dwa niezależne sprzętowe porty szeregowo UART obsługują moduły komunikujące się strumieniowo. Sprzętowy port UART0 obsługuje komunikację z modemem GSM SIM800L na wyprowadzeniach GPIO0 (RX, odbiór odpowiedzi modemu) oraz GPIO1 (TX, wysyłanie komend AT). Sprzętowy port UART1 obsługuje moduł nawigacji satelitarnej Air530Z na wyprowadzeniach GPIO17 (RX, odbiór wiadomości NMEA z modułu) oraz GPIO16 (TX, kanał konfiguracyjny do modułu, wykorzystywany jednorazowo podczas inicjalizacji). Rozdzielenie obu modułów na osobne magistrale UART jest podyktowane ich równoczesną aktywnością w niektórych stanach pracy systemu - w szczególności podczas wyzwolenia alarmu, gdy moduł GSM nawiązuje połączenie z siecią, a moduł GPS aktualizuje aktualną pozycję i kopiuje ją do treści powiadomienia SMS.

Magistralę I²C zrealizowano na liniach GPIO22 (SDA) oraz GPIO23 (SCL), wspólnych dla trzech układów peryferyjnych: czujnika inercyjnego MPU-6500, czujnika środowiskowego BME280 oraz ekspandera portów MCP23008. Linie magistrali zaopatrzone w rezystory podciągające (*pull-up*) o wartości 4,7 kΩ względem napięcia 3,3 V, zlokalizowane fizycznie na płycie bazowej. Każdy z układów ma przypisany unikatowy adres na magistrali. Adresy urządzeń I²C zostały przedstawione w Tabeli 4.

Tabela 4. Urządzenia I²C wraz z adresami.

Lp.	Urządzenie I ² C	Adres (hex)
1.	MCP23008	0x20
2.	MPU-6500	0x68
3.	BME280	0x77

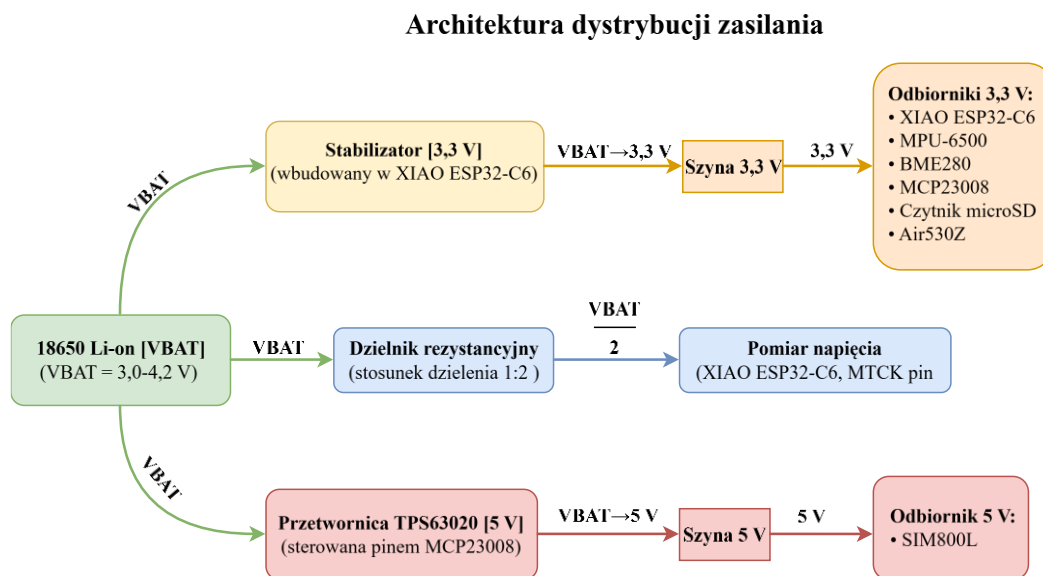
Magistralę SPI poprowadzono jako trzy linie sygnałowe - MOSI (GPIO18), MISO (GPIO20), SCK (GPIO19) - oraz linię wyboru układu CS (GPIO21), dedykowaną wyłącznie dla czytnika kart microSD. Na linii sygnałowej zastosowano rezystory szeregowo o wartości w zakresie 10-100 Ω , których rolą jest dopasowanie impedancji ścieżek przy wyższych częstotliwościach pracy magistrali oraz ograniczenie przesłuchów do sąsiednich linii. Pojedyncza karta microSD jest jedynym urządzeniem na tej magistrali, co eliminuje konieczność arbitrażu wielu urządzeń podrzędnych.

Cyfrowy czujnik wstrząsów (*Digital Shake Sensor*) podłączono bezpośrednio do wyprowadzenia GPIO2 mikrokontrolera, skonfigurowanego jako wejście przerwań w trybie RTC GPIO. Konfiguracja ta umożliwia wykorzystanie tego sygnału jako źródła wybudzenia z trybu głębokiego uśpienia ESP32-C6, zgodnie ze strategią opisaną w podrozdziale 3.3.5.

Na schemacie umieszczono również punkty testowe (oznaczone TP1-TP9), wyprowadzone do kluczowych linii sygnałowych: SDA i SCL magistrali I²C (TP1, TP2), linii nadawczych i odbiorczych GPS i GSM (TP3, TP4, TP8, TP9), magistrali zasilania 3,3 V (TP6) i 5 V (TP7) oraz masy układu (TP5). Punkty te ułatwiają diagnostykę elektryczną na etapie uruchomienia i kalibracji prototypu.

5.2.2. Układ zasilania

Sekcję zasilania zaprojektowano w architekturze dwóch szyn napięciowych: szyny 3,3 V zasilającej cyfrowe peryferia oraz szyny 5 V zasilającej moduł GSM SIM800L. Ogólną topologię dystrybucji zasilania przedstawiono w sposób blokowy na Rysunek 24.

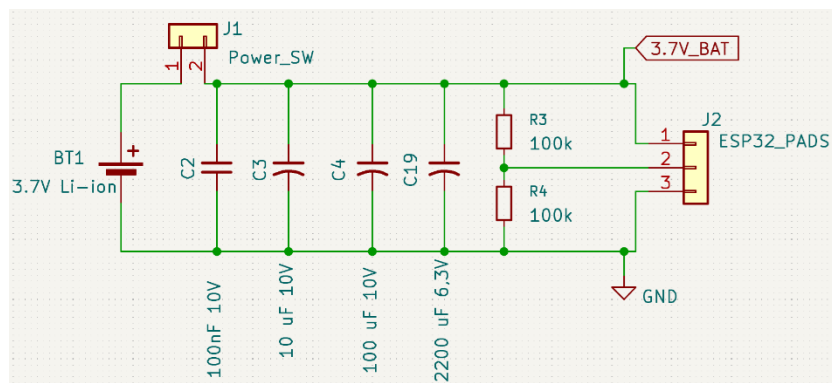


Rysunek 24. Architektura dystrybucji zasilania (źródło: opracowanie własne).

Źródłem zasilania jest pojedyncze ogniwo litowo-jonowe Sony US18650VTC6 (BT1 na Rysunek 23). Pomiędzy złączem baterii a resztą układu znajduje się główny wyłącznik mechaniczny (J1).

Bezpośrednio za wyłącznikiem na linii zasilającej umieszczono komplet kondensatorów filtrujących pełniących dwie komplementarne funkcje. Trzy kondensatory odsprężające (C2, C3, C4

na Rysunek 25) o pojemnościach 100 nF, 10 μ F oraz 100 μ F tworzą kaskadę filtrującą pokrywającą szerokie spektrum częstotliwości zakłóceń. Cztery kondensator ma funkcję bufora energetycznego C19 o pojemności 2200 μ F wykonany w technologii niskiej impedancji szeregowej (low ESR), którego zadaniem jest pokrycie chwilowych, dużych zapotrzebowań prądowych w szczególności impulsów rzędu pojedynczych amperów generowanych przez modem GSM SIM800L podczas nawiązywania połączenia z siecią. Bez tego rezerwuaru energii nagły wzrost poboru prądu prowadziłby do zapaści napięcia na linii zasilającej (*brownout*) i niestabilnej pracy całego układu.



Rysunek 25. Fragment schematu obejmujący sekcję zasilania (źródło: opracowanie własne).

Linia zasilania jest następnie rozdzielana na trzy ścieżki:

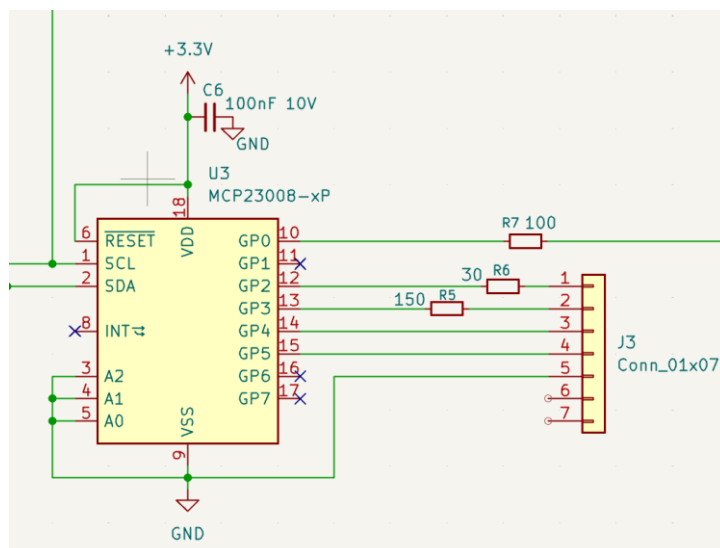
- Bezpośrednie zasilanie modułu XIAO ESP32-C6, którego zintegrowany stabilizator wytwarza napięcie dla wszystkich pozostałych układów cyfrowych;
- wejście przetwornicy (*buck-boost*) TPS63020, dostarczającej napięcie 5 V dla modułu GSM SIM800L;
- linia pomiarowa, doprowadzona do dzielnika rezystancyjnego umożliwiającego odczyt aktualnego napięcia ogniwa.

Pomiar napięcia ogniwa zrealizowano przez dzielnik rezystancyjny zbudowany z dwóch rezystorów 100 k Ω (R3 i R4 na Rysunek 25) zapewniający stosunek dzielenia 1:2. Wyjście dzielnika doprowadzono do złącza J2, a stamtąd dedykowanym padem spodnim modułu XIAO (MTCK na Rysunek 13) do wejścia przetwornika analogowo-cyfrowego. Zastosowanie wysokich wartości rezystancji ogranicza ciągły pobór prądu z baterii do poziomu pojedynczych mikroamperów, pomijalnego w bilansie energetycznym urządzenia.

Przetwornicę TPS63020 (U6 w schemacie) wykorzystano w konfiguracji tak aby na wyjściu uzyskać 5V poprzez zwarcie zwory przy R7 (widoczny na Rysunek 21) oraz pozostawienie pozostałych rozwartych. Odłączono pin aktywujący EN (*Enable*) od GND, aby możliwe było uruchamianie przetwornicy wyjściem GP0 ekspandera portów MCP23008, zgodnie ze strategią opisaną w podrozdziale 3.3.4. Dodatkowo wejście EN uzupełniono rezystorem podciągającym 100 k Ω (R8) do masy, zapewniającym stan domyślnie wyłączony w przypadku braku aktywnego sygnału sterującego z ekspandera. Wyjście przetwornicy zaopatrzone w równoległy zestaw kondensatorów filtrujących o pojemnościach 100 nF, 470 μ F, 100 μ F i 10 μ F (C11-C14), niezbędny do tłumienia szybkich impulsów prądowych pobieranych przez modem podczas nawiązywania połączenia z siecią GSM. Dioda Zenera (D1) na wyjściu pełni funkcję zabezpieczenia przed nadmiernym wzrostem napięcia.

5.2.3. Interfejs użytkownika

Fizyczny interfejs użytkownika składa się z dwóch przycisków oraz dwóch diod sygnalizacyjnych LED, obsługiwanych przez ekspander portów MCP23008. Schemat tej części układu przedstawiono na Rysunek 26.



Rysunek 26. Fragment schematu: ekspander MCP23008 i interfejs (źródło: opracowanie własne).

Ekspander MCP23008 (U3 na Rysunek 23) komunikuje się z mikrokontrolerem przez magistralę I²C. Adres układu na magistrali ustalono przez podłączenie wszystkich trzech wyprowadzeń adresowych (A0, A1, A2) do masy, co odpowiada adresowi 0x20 w przestrzeni adresowej I²C. Wyprowadzenie INT układu, oznaczające wystąpienie zmiany stanu wejść, pozostawiono niewykorzystane odczyt stanu przycisków odbywa się przez okresowe odpytywanie ekspandera (*polling*) w głównej pętli oprogramowania.

Przyporządkowanie wyprowadzeń GP0-GP7 ekspandera do konkretnych funkcji zestawiono w Tabela 5.

Tabela 5. Funkcje wyprowadzeń GP0-GP7 ekspandera MCP23008

Pin MCP23008	Kierunek	Funkcja sprzętowa
GP0	Wyjście	Sterowanie załączeniem przetwornicy
GP1		Rezerwa
GP2	Wyjście	Dioda sygnalizacji WiFi AP
GP3	Wyjście	Dioda sygnalizacji alarmu
GP4	Wejście + pull-up	Przycisk uruchomienia WiFi AP (do GND)
GP5	Wejście + pull-up	Przycisk odwołania Alarmu (do GND)
GP6		Rezerwa
GP7		Rezerwa

Diody LED zasilane są napięciem 3,3V, przyciski wykorzystują wewnętrzne rezystory podciągające ekspandera MCP23008, eliminując konieczność stosowania dodatkowych elementów zewnętrznych.

5.2.4. Złącza zewnętrzne i punkty pomiarowe

Płytką bazową wyposażoną jest w komplet rozłącznych złączy umożliwiających podłączenie elementów znajdujących się poza nią - mechanicznego wyłącznika napięcia, czujnika wstrząsów oraz interfejsu użytkownika. Złącza te zestawiono w Tabela 6, wraz z opisem ich roli i typu.

Tabela 6. Złącza zewnętrzne płytki bazowej

Oznaczenie	Typ złącza	Funkcja
J1	Złącze 2-stykowe	Wyprowadzenie głównego wyłącznika do mechanicznego przełącznika zamontowanego w obudowie
J2	Złącze 3-stykowe	Wyprowadzenie styków ze spodu ESP32C6
J3	Złącze 7-stykowe	Wyprowadzenie interfejsu użytkownika na obudowę (2 przyciski i 2 diody)
J4	Złącze 3-stykowe	Podłączenie cyfrowego czujnika wstrząsów

Ogniwo zasilające Li-Ion umieszczone jest w koszyku przewlekany wlotowany bezpośrednio do płyty bazowej, co eliminuje konieczność stosowania dodatkowych przewodów łączących baterię z układem i zwiększa odporność zespołu na wibracje wynikające z eksploatacji rowerowej.

Sygnały radiowe poprowadzono dwoma złączami koncentrycznymi zlokalizowanymi na modułach komunikacyjnych - pierwsze na module GSM SIM800L, drugie na module nawigacji satelitarnej Air530Z.

Dla pasma GSM zastosowano gumową antenę dipolową ze złączem SMA, połączoną z modulem za pośrednictwem krótkiego adaptera koncentrycznego zakończonego wtyczką IPEX po stronie modułu i gniazdem SMA po stronie anteny.

Dla pasma GPS wykorzystano aktywną antenę ceramiczną typu *patch* o wymiarach 25×25 mm, wyposażoną we wbudowany wzmacniacz niskoszumny (*Low Noise Amplifier*) zasilany napięciem stałym podawanym przez moduł Air530Z bezpośrednio po linii koncentrycznej, co istotnie poprawia czułość odbiornika satelitarnego.

Obie anteny umieszczone są wewnątrz obudowy, przymocowane do wewnętrznej strony pokrywy, materiał PLA jest praktycznie przezroczysty dla fal radiowych w obu wykorzystywanych pasmach (900/1800 MHz dla GSM oraz 1575 MHz dla GPS), co umożliwia rezygnację z wyprowadzania anten na zewnątrz urządzenia, upraszczając konstrukcję obudowy i zwiększając jej szczelność.

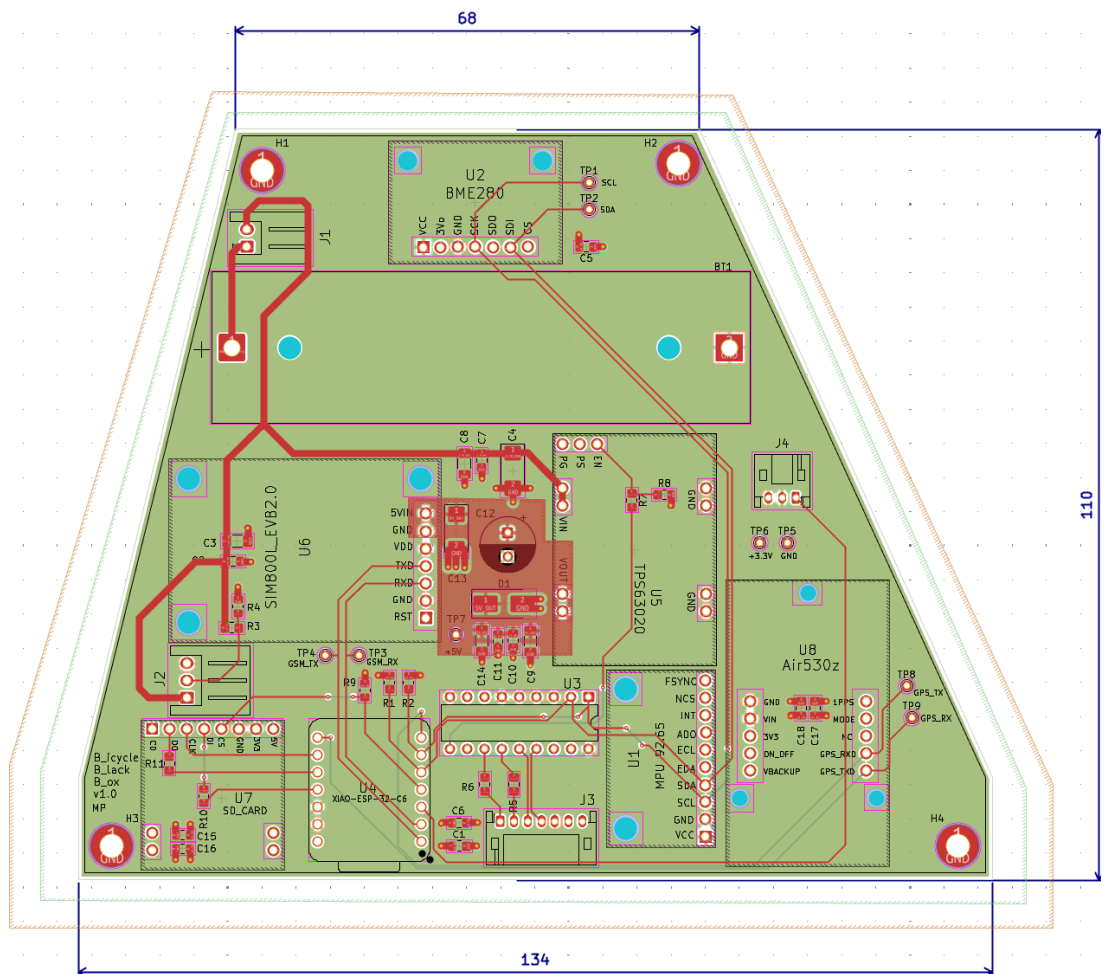
Trzecie złącze antenowe, dostępne na module XIAO ESP32-C6, pozostawiono niewykorzystane, komunikacja Wi-Fi w trybie serwisowym wykorzystuje zintegrowaną antenę PCB modułu, której zasięg jest wystarczający dla bezpośredniego kontaktu z urządzeniem klienckim podczas konfiguracji.

Otwory montażowe (H1-H4) w czterech narożnikach płytki, o średnicy odpowiadającej standardowemu śrubom M3, służą do mocowania płytki w obudowie. Pola lutownicze wokół tych otworów podłączono galwanicznie do masy układu, co stanowi standardową praktykę zwiększającą stabilność potencjału referencyjnego oraz zapewniającą potencjał ekranujący w razie zastosowania obudowy z materiału przewodzącego.

5.3. Projekt płytki drukowanej

Płytkę drukowaną zaprojektowano w środowisku KiCad EDA wraz ze schematem elektrycznym omówionym w poprzednim podrozdziale. Niniejsza forma urządzenia stanowi trzecią, finalną iterację prototypu projektowego - została poprzedzona dwoma wcześniejszymi etapami konstrukcji eksperymentalnej. W pierwszym etapie układ zrealizowano na płytce stykowej (*breadboard*), służącej walidacji działania poszczególnych modułów funkcjonalnych w izolacji oraz weryfikacji podstawowej komunikacji z mikrokontrolerem. W drugim etapie układ przeniesiono na uniwersalną płytkę drukowaną (*perfboard*) z trwale lutowanymi połączeniami, umożliwiającą pierwsze testy w warunkach zbliżonych do docelowych. Doświadczenia zebrane podczas obu wcześniejszych iteracji stanowiły bezpośrednią podstawę do zaprojektowania dedykowanej płytki drukowanej, której opis stanowi

przedmiot niniejszego podrozdziału. Efekt przeniesienia schematu z Rysunek 23 na projekt PCB przedstawiono na Rysunek 27.



Rysunek 27. Projekt PCB dedykowanej płytki (źródło: opracowanie własne).

Wybór konkretnych parametrów PCB - kształtu, struktury warstwowej, rozmieszczenia komponentów oraz prowadzenia ścieżek - podporządkowany został wymaganiom kompatybilności elektromagnetycznej, ograniczeniom wymiarowym oraz cechom mechanicznym wynikającym z koncepcji konstrukcji opisanej w Rozdziale 3.

5.3.1. Wymiary i kształt płytki

Płytkę ma kształt nieregularnego trapezu o wymiarach gabarytowych 134×110 mm, z krótszą podstawą o długości 68 mm. Nieregularny kontur jest świadomą decyzją konstrukcyjną, podyktowaną docelowym miejscem mocowania urządzenia na rowerze. Zaprojektowana płytkę została przedstawiona na

Płytkę zaprojektowaną została tak, aby kompletne urządzenie - wraz z obudową - zmieściło się w przestrzeni dostępnej w głównym trójkącie ramy roweru, pomiędzy górną a dolną rurą, w sąsiedztwie główki ramy (rury kierownicy). Trapezoidalna geometria PCB pozwala maksymalnie wykorzystać dostępną przestrzeń przy zachowaniu wystarczającej powierzchni dla rozmieszczenia wszystkich modułów funkcjonalnych. Dodatkowym, drugorzędnym efektem niesymetrycznej geometrii jest redukcja całkowitej powierzchni płytki w stosunku do prostokąta opisanego na konturze, co przekłada się bezpośrednio na koszt wytworzenia w usłudze produkcyjnej.

W czterech narożnikach kontur PCB zaopatrzone w otwory montażowe (H1-H4) o średnicy 3,2 mm, dopasowane do śrub mocujących typu M3. Pola lutownicze wokół tych otworów podłączono

galwanicznie do potencjału masy układu, co stanowi standardową praktykę zapewniającą dodatkową ścieżkę dla potencjału referencyjnego oraz potencjał ekranujący w przypadku zastosowania obudowy z materiału przewodzącego.

5.3.2. Stos warstwowy

Płytkę zaprojektowano jako konstrukcję czterowarstwową o grubości całkowitej 1,6 mm. Decyzja o zastosowaniu czterech warstw miedzi, zamiast prostszej i tańszej konstrukcji dwuwarstwowej, była podyktowana priorytetem niezawodności elektromagnetycznej projektu - w szczególności w kontekście współwystępowania na płycie obwodów mieszanych: czujników analogowych o niskim poziomie sygnału (BME280, MPU-6500), cyfrowych magistral komunikacyjnych (I²C, SPI, UART) oraz silnie zaburzającego modemu GSM, generującego impulsy prądowe o amplitudzie do dwóch amperów podczas nawiązywania połączenia. Struktura warstwowa, zilustrowana w Tabeli 7, umożliwia fizyczne rozdzielenie tych domen.

Tabela 7. Stos warstwowy PCB

Warstwa	Oznaczenie KiCad	Zawartość	Funkcja
L1	F.Cu	Zasilanie z ogniwa 18650, wyspa zasilania 5V, sygnały	Warstwa zewnętrzna górna
L2	In1.Cu	Pełna płaszczyzna GND	Ekranowanie i potencjał referencyjny
L3	In2.Cu	Płaszczyzna zasilania 3,3 V	Dystrybucja zasilania
L4	B.Cu	Przelotki sygnałowe	Warstwa zewnętrzna dolna

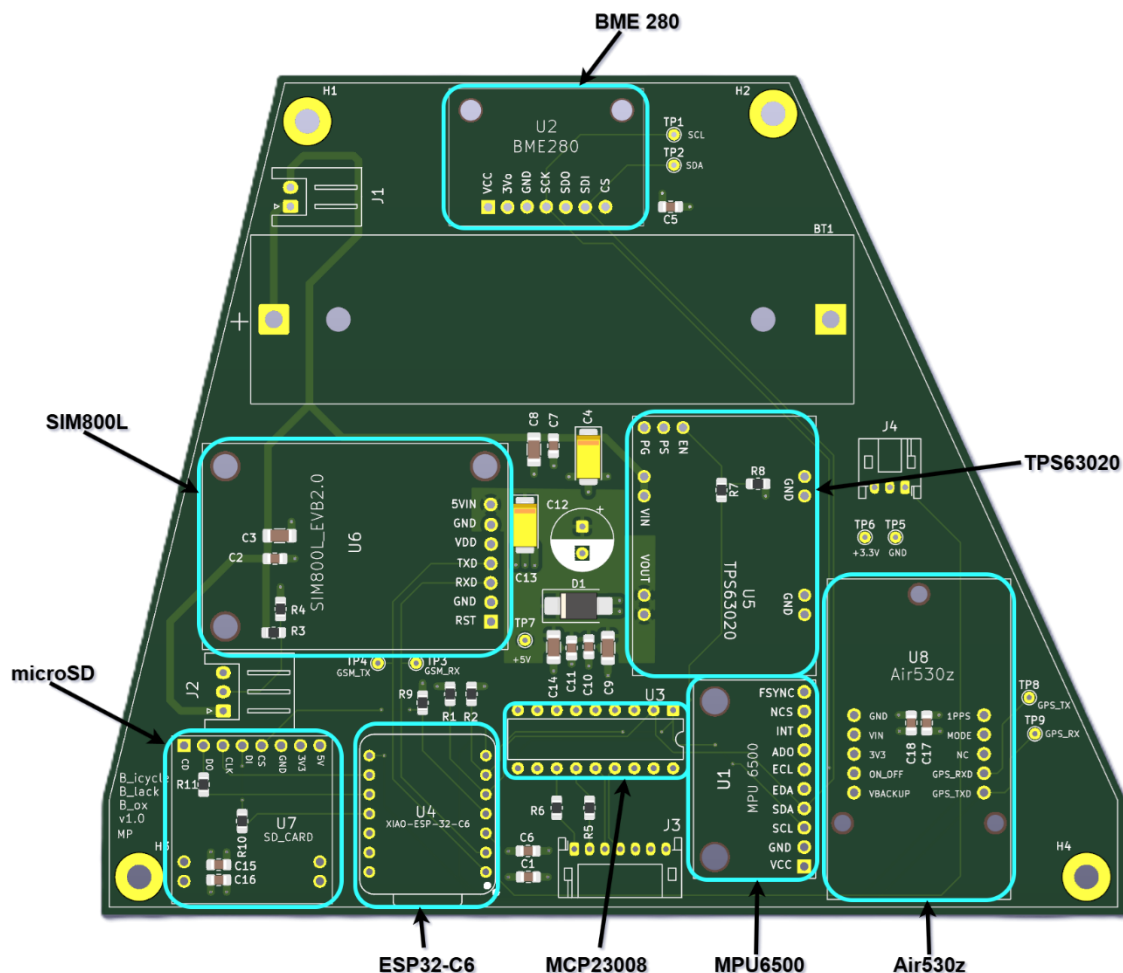
Warstwy wewnętrzne (L2 i L3) pełnią funkcję płaszczyzn ciągłych - odpowiednio masy i zasilania 3,3 V - dystrybuujących energię do układów cyfrowych z minimalną impedancją oraz zapewniających niski potencjał referencyjny dla sygnałów na warstwach zewnętrznych. Pełna, nieprzerwana płaszczyzna masy (L2) umieszczona bezpośrednio pod warstwą sygnałową (L1) tworzy zwarte pętle prądowe sygnałów wysokiej częstotliwości, ograniczając emisję promieniowaną oraz podatność na zakłócenia zewnętrzne. Warstwy zewnętrzne L1 i L4 wykorzystywane są do prowadzenia sygnałów cyfrowych oraz wyspy 5 V opisanej szczegółowo w kolejnym podrozdziale.

5.3.3. Rozdzielenie domen zasilania

Ze względu na impulsowy pobór prądu modemu SIM800L, mogący wprowadzać zaburzenia do wrażliwych obwodów analogowych i cyfrowych, jego obwód zasilania 5 V odseparowano fizycznie od pozostałej części układu. Separację zrealizowano jako wydzieloną wyspę napięciową na warstwie górnej PCB, przedstawioną na Rysunek 28.

5.3.4. Rozmieszczenie komponentów i prowadzenie ścieżek

Rozmieszczenie komponentów na płycie, widoczne na wizualizacji trójwymiarowej przedstawionej na Rysunek 29, podporządkowane zostało dwóm zasadom: skróceniu krytycznych ścieżek sygnałowych oraz ergonomii montażu i serwisowania.



Rysunek 29. Wizualizacja trójwymiarowa płytki z opisanymi komponentami (źródło: opracowanie własne).

Czujnik środowiskowy BME280 umieszczono w pobliżu krótszej krawędzi płytki, jak najdalej od modemu GSM SIM800L oraz przetwornicy TPS63020. Decyzja ta wynika z dwóch względów: po pierwsze, oba wymienione elementy stanowią lokalne źródła ciepła, co wpływałoby na wskazania temperatury z czujnika, a po drugie, większa odległość od modemu zmniejsza ryzyko zakłóceń elektromagnetycznych w pomiarach.

Czujnik inercyjny MPU-6500 umieszczono w środkowej części płytki, w obszarze stabilnym mechanicznie. Lokalizacja w pobliżu osi geometrycznej minimalizuje wpływ ewentualnych odkształceń obudowy podczas mocowania urządzenia na ramie roweru, które mogłyby wprowadzać niezerowy offset osi referencyjnych czujnika.

Modem GSM SIM800L wraz z otaczającym go obwodem zasilającym - przetwornicą TPS63020, kondensatorami filtrującymi C7, C8, C11-C14 oraz diodą zabezpieczającą D1 - zlokalizowano w centralnej części płytki, w bezpośrednim sąsiedztwie złącza anteny GSM. Skrócenie ścieżki sygnału RF od modemu do złącza antenowego minimalizuje straty transmisyjne oraz emisję pasożytniczą do pozostałych obwodów.

Moduł nawigacyjny Air530Z umieszczono po przeciwnej stronie płytki w stosunku do SIM800L, z analogicznych względów - wzajemne oddalenie obu modułów radiowych ogranicza interferencje

sygnałowe oraz ciepłone. Złącze anteny GPS poprowadzono bezpośrednio z modułu, również minimalizując długość ścieżki sygnału satelitarnego.

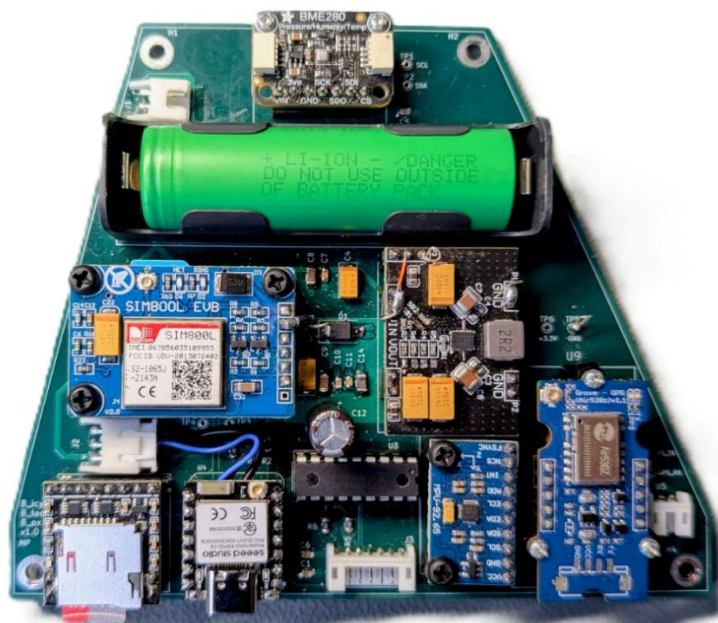
Czytnik kart microSD umieszczono w sąsiedztwie modułu XIAO ESP32-C6, w celu skrócenia ścieżek magistrali SPI pracującej z relatywnie wysokimi częstotliwościami zegarowymi. Skrócenie magistrali SPI poprawia integralność sygnałów cyfrowych oraz redukuje emisję elektromagnetyczną z linii zegarowych.

Ekspander portów MCP23008 w obudowie przewlekanej DIP-18 umieszczono w pobliżu złącza taśmowego J3 wyprowadzającego sygnały interfejsu użytkownika, co minimalizuje długość ścieżek przyciski-LED oraz upraszcza topologię routingu sygnałów wolnych.

Ścieżki sygnałów cyfrowych prowadzono głównie po warstwach zewnętrznych (L1 i L4), z wykorzystaniem przelotek do zmian płaszczyzn w razie potrzeby. Magistrala I²C, jako kluczowa dla wielu układów peryferyjnych, prowadzona została stosunkowo grubymi ścieżkami (większymi od standardowej szerokości sygnałowej) w celu obniżenia rezystancji i pojemności rozłożonej, co zwiększa odporność magistrali na zakłócenia.

Szczególne uwagę poświęcono również prowadzeniu ścieżek wokół kondensatorów filtrujących o większych pojemnościach, w szczególności na wyjściu przetwornicy TPS63020 zasilającej modem GSM. Pola lutowicze kondensatorów C13 i C4 połączone zostały z odpowiednimi płaszczyznami masy i zasilania za pośrednictwem wielu przelotek równoległych. Taka technika, znana jako przeszywanie przelotkami (*via stitching*), zmniejsza wypadkową impedancję indukcyjną ścieżki powrotnej prądu, co jest istotne przy szybkozmiennych obciążeniach impulsowych - szczególnie przy nawiązywaniu połączenia z siecią GSM, kiedy modem pobiera krótkotrwałe impulsy prądowe o amplitudzie rzędu pojedynczych amperów. Bez wystarczająco niskoimpedancyjnej ścieżki powrotnej kondensatory nie mogłyby skutecznie spełniać swojej funkcji bufora energetycznego.

Tak zaprojektowaną płytkę wyeksportowano do formatu Gerber zgodnego z wymaganiami producenta i zlecono wytworzenie zewnętrznej usługi produkcyjnej JLCPCB. Parametry zamówienia obejmowały: materiał laminatu FR4, cztery warstwy miedzi, grubość 1,6 mm, pokrycie ścieżek HASL (*Hot Air Solder Leveling*), zieloną maskę lutowniczą oraz biały nadruk opisowy. Wybór standardowych parametrów we wszystkich pozycjach był podyktowany dążeniem do minimalizacji kosztu prototypu oraz brakiem szczególnych wymagań technicznych uzasadniających zastosowanie wariantów rozszerzonych. Po otrzymaniu wyprodukowanych płytek przeprowadzono samodzielny montaż wszystkich komponentów elektronicznych przy użyciu stacji lutowiczej z grotem precyzyjnym. Wynik montażu w postaci fizycznej, zmontowanej płytki bazowej przedstawiono na Rysunek 30.

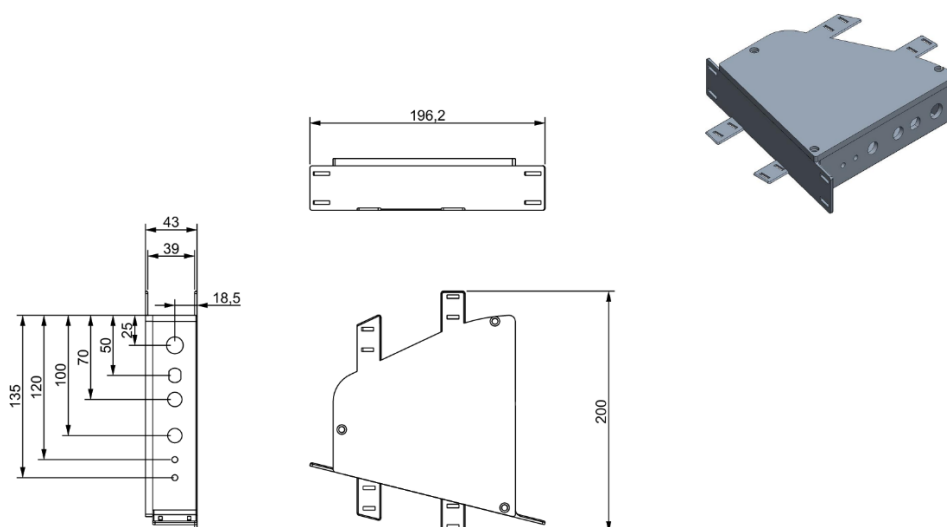


Rysunek 30. Zdjęcie zmontowanej płytki bazowej (źródło: opracowanie własne).

Po zakończeniu montażu wykonano inspekcję wzrokową połączeń oraz pomiar ciągłości elektrycznej kluczowych ścieżek za pomocą multimetru cyfrowego, weryfikując poprawność montażu przed pierwszym uruchomieniem prototypu.

5.4. Konstrukcja mechaniczna

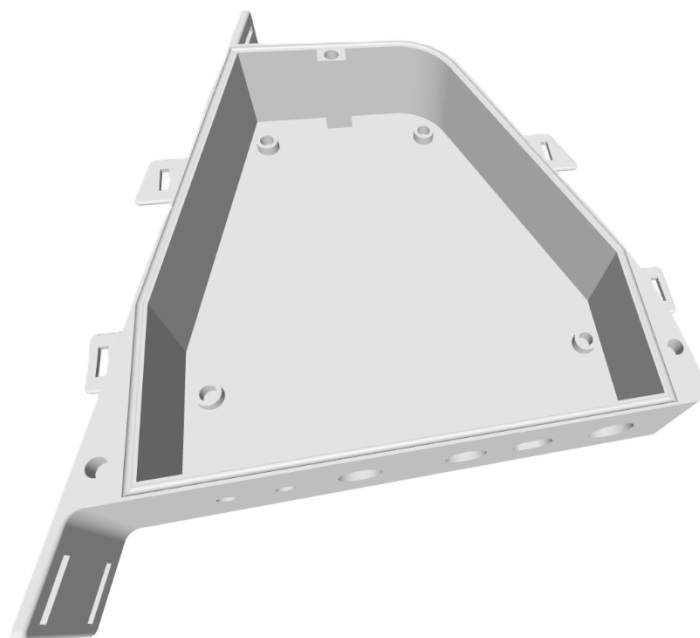
Konstrukcja mechaniczna urządzenia obejmuje obudowę chroniącą elektronikę, mechanizmy mocowania modułów wewnętrznych oraz sposób mocowania całego urządzenia do roweru. Wszystkie elementy mechaniczne, z wyjątkiem standardowych elementów złącznych (śruby, koszyk baterii), zaprojektowano w środowisku PTC Creo Parametric jako odrębne komponenty przeznaczone do wytworzenia w technologii druku trójwymiarowego. Uproszczony rysunek techniczny obudowy został przedstawiony na Rysunek 31.



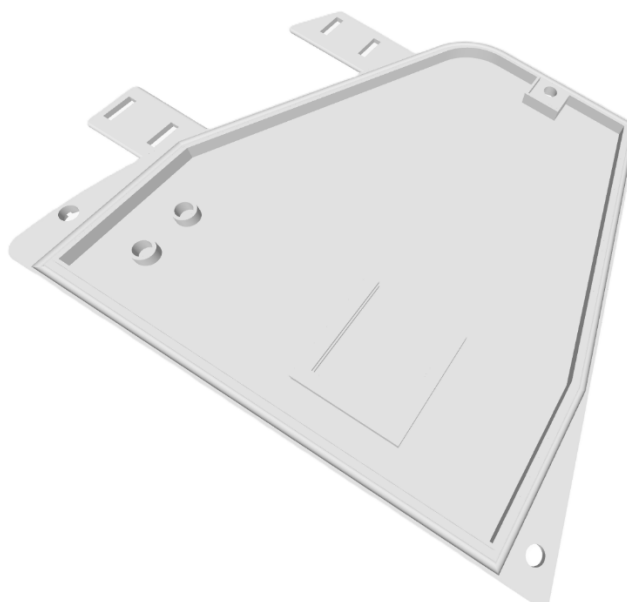
Rysunek 31. Uproszczony rysunek techniczny obudowy (źródło: opracowanie własne).

5.4.1. Obudowa drukowana w technologii 3D

Obudowę zaprojektowano jako konstrukcję dwuczęściową - dolna część stanowi podstawę z otworami montażowymi pod płytkę drukowaną, wraz z otworami na montaż elementów interfejsu użytkownika na panelu oraz koszyk ogniwa, górna część pełni funkcję pokrywy, do której montowane są anteny oraz czujnik wstrząsów. Wizualizacje trójwymiarowe obudowy oraz pokrywy zostały przedstawione na Rysunek 32 i Rysunek 33.



Rysunek 32. Wizualizacja trójwymiarowa obudowy (źródło: opracowanie własne).



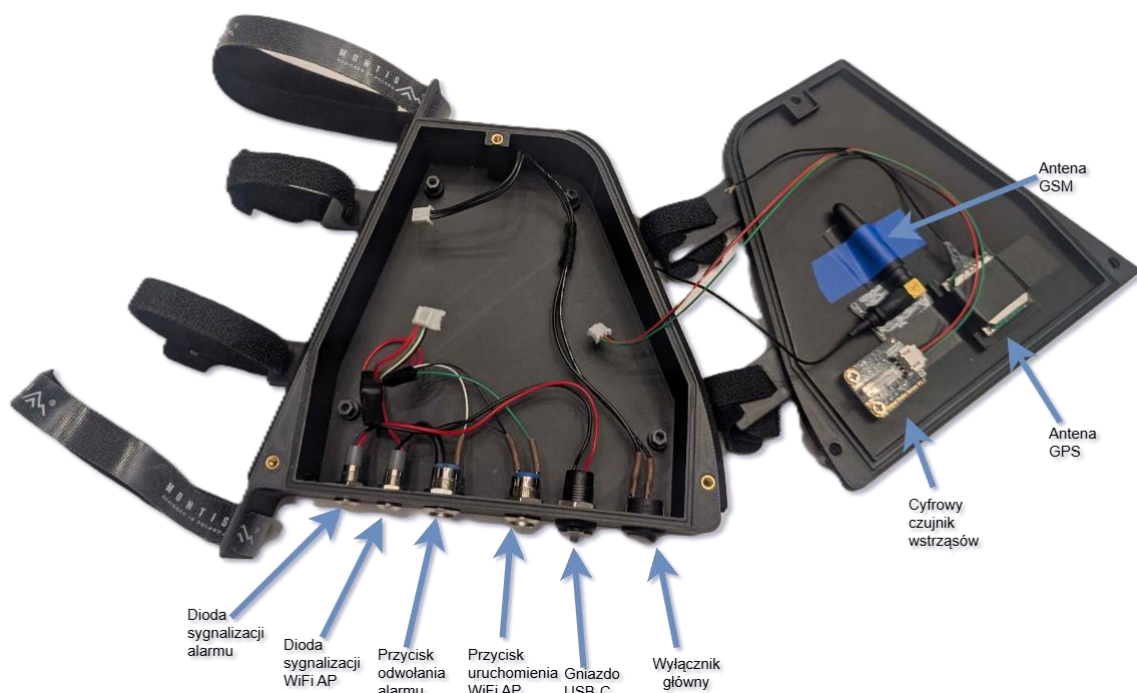
Rysunek 33. Wizualizacja trójwymiarowa pokrywy (źródło: opracowanie własne).

Geometria obudowy nawiązuje do nieregularnego kształtu płytki drukowanej, opisanego w podrozdziale 5.3.1, z dodaniem marginesu konstrukcyjnego pozwalającego na rozmieszczenie wewnętrznych elementów konstrukcyjnych - tulei dystansowych dla modułów, kanałów prowadzenia

przewodów oraz miejsca na ogniwo zasilające. Zewnętrzne wymiary obudowy uwzględniają również minimalną grubość ścianek wymaganą dla zachowania sztywności konstrukcji wykonanej z PLA.

Wytworzenie obu części obudowy zrealizowano na drukarce trójwymiarowej Bambu Lab P1S w technologii osadzania topionego materiału, z wykorzystaniem filamentu PLA w kolorze czarnym. Drukarka P1S charakteryzuje się zamkniętą komorą roboczą, co zapewnia stabilne warunki termiczne podczas druku oraz ogranicza wpływ przeciągów na jakość pierwszych warstw wydruku. Wybór czarnego koloru materiału podyktowany był względami estetycznymi oraz redukcją widoczności drobnych zarysowań i zabrudzeń podczas eksploatacji w warunkach polowych.

Po zakończeniu druku obie części obudowy poddano oczyszczeniu - usunięto materiał podporowy oraz wygładzono ostre krawędzie wynikające z procesu warstwowego. W otworach montażowych przewidzianych pod śruby M3 i M2 wprowadzono metalowe wkładki gwintowane przy pomocy grotu lutownicy. Następnie zmontowano elementy interfejsu użytkownika na panelu bocznym, przymocowano cyfrowy czujnik wstrząsów oraz przymocowano anteny GSM i GPS. Wstępnie zmontowaną obudowę przedstawiono na Rysunek 34.



Rysunek 34. Wstępnie zmontowana obudowa (źródło: opracowanie własne).

W bocznej ścianie obudowy przewidziano dedykowane otwory dla elementów interfejsu fizycznego - dwóch diod sygnalizacyjnych, dwóch przycisków, przełącznika oraz gniazda USB-C. W wewnętrznej stronie pokrywy obudowy umieszczono również mocowania dla anten GSM i GPS - szczegóły konstrukcji torów radiowych opisano w podrozdziale 5.2.4. Widok interfejsu użytkownika po zmontowaniu przedstawiono na Rysunek 35.



Rysunek 35. Interfejs użytkownika na obudowie (źródło: opracowanie własne).

5.4.2. Mocowanie modułów i tuleje dystansowe

Wewnętrzna konstrukcja mechaniczna urządzenia obejmuje dwa niezależne poziomy mocowań śrubowych: mocowanie płytki bazowej do obudowy oraz mocowanie poszczególnych modułów funkcjonalnych do płytki bazowej. W obu przypadkach zastosowano rozwiązania zapewniające trwałość połączeń w długim okresie eksploatacji, uwzględniające specyfikę montażu w wydrukach 3D wykonanych z PLA.

W czterech narożnikach dna obudowy zaprojektowano dedykowane tuleje wzmacniające, w które po zakończeniu druku wtopiono metalowe wkładki gwintowane (*threaded inserts*) o gwincie M3. Operację osadzenia insertów wykonano techniką termiczną - przy użyciu rozgrzanej lutownicy stopiono otaczający materiał PLA, umożliwiając wsunięcie wkładki na żadaną głębokość, gdzie po zastygnięciu tworzywa wkładka pozostaje trwale osadzona. Tak przygotowane wkładki tworzą cztery punkty mocowania, do których przykręcono płytkę bazową czterema stalowymi śrubami M3, przechodzącymi przez otwory montażowe H1-H4 zlokalizowane w narożnikach PCB. Zastosowanie metalowych wkładek w miejsce gwintu wykonanego bezpośrednio w tworzywie zapewnia znacznie większą trwałość połączenia, umożliwiając wielokrotne demontaże i ponowne montaż urządzenia bez ryzyka uszkodzenia gwintu.

Cztery moduły SIM800L, BME280, MPU-6500 oraz Air530Z pozwalają na dodatkowe mocowanie mechaniczne, niezależnego od podstawowego połączenia elektrycznego realizowanego przez złącza pinowe (*goldpin*). Fizyczne mocowanie modułu nylonowym zestawem śrubowym, składającym się z tulei dystansowej, śruby oraz nakrętki, wykonanych w całości z tworzywa sztucznego. Tuleja dystansowa zachowuje stały odstęp między modułem a płytą bazową, zapobiegając zwarciom oraz nadmiernym naprężeniom na złączach pinowych. Śruba przechodzi przez otwór montażowy modułu, tuleję oraz odpowiedni otwór w płycie bazowej, gdzie jest zabezpieczona nakrętką od strony dolnej. Wybór elementów z tworzywa sztucznego eliminuje ryzyko zwarć elektrycznych w obszarze gęsto rozmieszczonych ścieżek sygnałowych, a jednocześnie zachowuje wystarczającą wytrzymałość mechaniczną dla niewielkich obciążeń wynikających z masy poszczególnych modułów. Zamontowaną w obudowie PCB wraz z antenami oraz elementami interfejsu użytkownika przedstawiono na Rysunek 36.



Rysunek 36. Zmontowane urządzenie przed zamknięciem pokrywy (źródło: opracowanie własne).

Łączenie obu części obudowy - dolnej oraz pokrywy - zrealizowano w sposób analogiczny do mocowania płytki bazowej. W trzech punktach pokrywy obudowy wtopiono metalowe wkładki gwintowane M3, wykorzystując tę samą technikę termicznego osadzenia rozgrzaną lutownicą. Trzy stalowe śruby M3, przechodzące przez odpowiednie otwory w dolnej części obudowy, łączą obie części w trwałą całość.

5.4.3. Mocowanie urządzenia do roweru

Urządzenie mocowane jest w głównym trójkącie ramy roweru - w przestrzeni rozciągającej się między górną a dolną rurą ramy. Lokalizacja ta zapewnia kilka korzyści praktycznych: chroni urządzenie przed bezpośrednim uderzeniem w przypadku upadku roweru, utrzymuje je w polu widzenia rowerzysty (umożliwiając łatwą obserwację diod sygnalizacyjnych), oraz minimalizuje wpływ na aerodynamikę i geometrię prowadzenia roweru. Trapezoidalny kształt obudowy, opisany w podrozdziale 5.3.1, stanowi bezpośrednie dopasowanie geometryczne do zbieżności dwóch rur ramy - górnej i dolnej, zwierających się ku przodowi w kierunku główki ramy. Krótszy bok obudowy o długości 68 mm znajduje się od strony przedniej, w obszarze, gdzie obie rury są bliżej siebie, natomiast dłuższy bok kieruje się ku tylnej części ramy, gdzie odstęp między rurami jest większy.

Fizyczne mocowanie zrealizowano za pomocą sześciu pasków rzepowych z tworzywa tekstylnego o szerokości 10 mm każdy. W obecnej konfiguracji wykorzystywane są dwa paski przebiegające przez prowadnice górne oraz cztery paski przebiegające przez prowadnice dolne. Większa liczba pasków zaciskanych wokół dolnej rury ramy zwiększa stabilność mocowania w tej części obudowy i pozwala równomiernie rozłożyć siły zacisku po większej powierzchni styku. Wykorzystanie wielu pasków o niewielkiej szerokości - zamiast pojedynczych szerokich - zapewnia lepsze dopasowanie do rur o różnych średnicach oraz zwiększa pewność mocowania przez wzajemną redundancję poszczególnych pasków. Sposób montażu został przedstawiony na Rysunek 37.



Rysunek 37. Montaż urządzenia na rowerze (źródło: opracowanie własne).

Zastosowanie pasków rzepowych jako mechanizmu mocującego podyktowane zostało trzema względami praktycznymi. Po pierwsze, paski rzepowe dopasowują się do różnych średnic rur ramy, co umożliwia montaż urządzenia na rowerach o różnych konfiguracjach bez konieczności wymiany mocowania. Po drugie, mechanizm umożliwia szybki demontaż urządzenia po zakończeniu jazdy bez użycia narzędzi, co zwiększa wygodę eksploatacji. Po trzecie, paski rzepowe nie wymagają żadnej trwałej ingerencji w ramę roweru - w przeciwieństwie do uchwytów śrubowych wymagających wiercenia otworów - co umożliwia stosowanie urządzenia bez ryzyka uszkodzenia roweru.

6. Implementacja oprogramowania

Niniejszy rozdział dokumentuje implementację oprogramowania działającego na mikrokontrolerze urządzenia. Zakresem objęto wszystkie warstwy systemu - od niskopoziomowych sterowników peryferiów, poprzez algorytmy akwizycji danych i detekcji upadku, aż po procedurę alarmową oraz interfejs serwisowy dostępny przez serwer WWW. Oprogramowanie zostało zaprojektowane jako kompletne, wbudowane (*embedded*) rozwiązanie, niezależne od zewnętrznych usług chmurowych - cały kod uruchamiany jest lokalnie na mikrokontrolerze ESP32-C6, a komunikacja z otoczeniem ogranicza się do wysyłki wiadomości SMS przez sieć GSM, lokalnej sieci Wi-Fi w trybie punktu dostępowego oraz zapisu danych na karcie microSD.

Rozdział zorganizowano w siedmiu podrozdziałach prowadzących czytelnika od ogólnej architektury kodu, poprzez szczegóły implementacyjne poszczególnych funkcjonalności, aż po opis interfejsu serwisowego.

6.1. Architektura i pętla główna

Oprogramowanie urządzenia zostało napisane w języku C++ z wykorzystaniem frameworka Arduino dla mikrokontrolera ESP32-C6. Projekt utrzymywany jest w środowisku PlatformIO, opartym na platformie platform-espressif32 w wersji 55.3.35 oraz rdzeniu Arduino ESP32. Jako edytor kodu źródłowego wykorzystano Visual Studio Code z rozszerzeniem PlatformIO IDE. Wybór tego stosu technologicznego podyktowany został trzema względami: bogatym zestawem dostępnych bibliotek do obsługi peryferiów (czujnik środowiskowy BME280, parser NMEA, system plików microSD), wbudowanym wsparciem PlatformIO dla zarządzania zależnościami i wieloplatformowej kompilacji, oraz zintegrowanymi w środowisku Visual Studio Code narzędziami diagnostycznymi (monitor portu szeregowego, kompilator z obsługą błędów, podpowiadanie kodu).

Architekturę oprogramowania zaprojektowano jako jednowątkową, opartą na nieskończonej pętli głównej. Pomimo dostępności w ESP32-C6 systemu czasu rzeczywistego FreeRTOS, zrezygnowano z jawnego tworzenia dodatkowych zadań aplikacyjnych na rzecz pojedynczej pętli sterującej. Decyzja ta podyktowana została trzema czynnikami: prostotą diagnostyki, dla której deterministyczna kolejność wykonania operacji jest łatwa do prześledzenia w logach, brakiem konieczności synchronizacji dostępu do zasobów współdzielonych - czujników na wspólnej magistrali I²C, karty microSD, modułu GSM - co eliminuje ryzyko stanów wyścigu oraz niskim zapotrzebowaniem czasowym, gdyż żadne z realizowanych zadań nie wymaga rygorystycznego determinizmu w skali pojedynczych milisekund.

Pętla główna nie korzysta z funkcji `delay()` ani innych mechanizmów blokujących. Wszystkie operacje cykliczne (logowanie, detekcja upadku, miganie diod, obsługa retransmisji SMS) realizowane są asynchronicznie przez porównywanie znaczników czasowych z wartością `millis()`. Wyjątki od tej reguły, dotyczące operacji synchronicznych z natury (inicjalizacja sprzętu, transmisja AT do modemu GSM, obsługa zapytań HTTP), zostały opisane w odpowiednich podrozdziałach.

6.1.1. Podział oprogramowania na moduły funkcjonalne

Kod źródłowy urządzenia podzielono na dziesięć modułów funkcjonalnych, z których każdy realizuje wyodrębniony obszar odpowiedzialności. Wykaz modułów wraz z opisem ich roli zestawiono w Tabeli 8.

Tabela 8. Moduły oprogramowania urządzenia

Plik źródłowy	Zakres odpowiedzialności
<code>main.cpp</code>	Pełni funkcję punktu wejścia programu oraz orkiestratora systemu. Realizuje inicjalizację wszystkich modułów funkcjonalnych w określonej kolejności oraz nieskończoną pętlę sterującą, w której cyklicznie wywoływane są operacje

	akwizycji danych, detekcji zdarzeń, rejestracji oraz obsługi interfejsów. Odpowiada również za koordynację zapisu logu jazdy w odpowiednich interwałach czasowych oraz za warunki przejścia urządzenia w tryb głębokiego uśpienia.
sensors.cpp	Realizuje obsługę wszystkich czujników urządzenia - akcelerometra i żyroskopu MPU-6500, czujnika środowiskowego BME280, modułu nawigacyjnego Air530Z oraz pomiaru napięcia baterii przez przetwornik analogowo-cyfrowy. Zawiera również procedurę parsowania sygnału NMEA, mechanizm kalibracji jednostki inercyjnej oraz właściwy algorytm detekcji upadku.
shockcapture.cpp	Stanowi wysokoczęstotliwościowy rejestrator akceleracji pracujący z częstotliwością 200 Hz, zapisujący osobne pliki CSV uderzeń z wykorzystaniem bufora cyklicznego (<i>ring buffer</i>). Pełni wyłącznie rolę źródła danych do późniejszej analizy - nie uczestniczy w decyzji o wykryciu upadku ani w sekwencji alarmowej.
storage.cpp	Zarządza sesjami jazdy - otwarciem, zapisem rekordów oraz zamknięciem sesji wraz z generowaniem metadanych i tras GPX. Odpowiada również za strukturę katalogów na karcie microSD oraz za trwałą konfigurację urządzenia, obejmującą tryby pracy, parametry alarmu wstępnego, kontakt alarmowy oraz hasło punktu dostępowego Wi-Fi.
sdcard.cpp	Stanowi niskopoziomową warstwę dostępu do karty microSD, opartą na bibliotece SdFat. Realizuje operacje na systemie plików, generowanie ścieżek katalogów według daty oraz tworzenie nagłówek plików CSV.
userInterface.cpp	Obsługuje interfejs użytkownika urządzenia - odczyt stanu przycisków podłączonych do ekspandera MCP23008, sterowanie diodami LED sygnalizacyjnymi, realizację sekwencji alarmowej oraz procedury alarmu wstępnego, a także przełączanie trybu serwisowego z uruchamianiem i zatrzymywaniem serwera WWW.
communication.cpp	Realizuje komunikację z modemem GSM SIM800L przez interfejs UART0. Zawiera sekwencję komend AT, procedurę wysyłki wiadomości SMS, obsługę timeoutów oraz mechanizm ponawiania prób (<i>retry</i>) w przypadku niepowodzenia transmisji.
webserver.cpp	Implementuje serwer WWW dostępny w trybie serwisowym urządzenia. Obejmuje własny parser żądań HTTP, zestaw endpointów REST API, generowanie dynamicznych stron HTML.
power.cpp	Odpowiada za tryb głębokiego uśpienia mikrokontrolera ESP32-C6. Realizuje konfigurację źródeł wybudzenia oraz przygotowanie peryferiów do uśpienia, w tym wyłączenie modułów radiowych i ograniczenie poboru prądu układów peryferyjnych.
mcp23008.cpp	Stanowi niskopoziomowy sterownik ekspandera wejść/wyjść MCP23008, realizujący komunikację z układem poprzez magistralę I ² C. Udostępnia operacje odczytu i zapisu rejestrów wykorzystywane przez pozostałe moduły systemu.

W dalszej części rozdziału omówiono sekwencję inicjalizacji oraz przebieg pętli głównej.

6.1.2. Sekwencja inicjalizacji i pętla główna

Wykonanie programu rozpoczyna się od funkcji `setup()`, w której odbywa się inicjalizacja wszystkich modułów funkcjonalnych w określonej kolejności, zapewniającej dostępność zasobów wymaganych przez moduły aktywowane w późniejszych krokach. Bezpośrednio po zakończeniu inicjalizacji program przechodzi do nieskończonej pętli głównej `loop()`, w której cyklicznie realizowane są operacje akwizycji danych, detekcji zdarzeń, rejestracji oraz obsługi interfejsów. Pseudokod przebiegu sterowania na najwyższym poziomie przedstawiono na Listing 1.

Listing 1. Pseudokod sekwencji inicjalizacji i pętli głównej

```
setup() :
Serial.begin(115200)           // diagnostyczny UART
storage.init()                 // karta microSD + plik dev.json
sensorsInit()                  // I2C, MPU-6500, BME280, GPS
configManager.loadConfig()     // emergency.json
shockCapture.init()            // rejestrator 200 Hz
bootmgrMcpInit()               // ekspander MCP23008
weryfikacja inicjalizacji modułów // sygnalizacja błędu LED

loop() :                       // nieskończona pętla
d = sensorsRead()              // odczyt wszystkich czujników
shockCapture.update(d)          // rejestrator 200 Hz
fdr = updateFallDetection(d)    // algorytm detekcji upadku
if fdr == FALL_CONFIRMED:
    uruchom sekwencję alarmową
storage.save(d)
userInterfacePoll()
if webserver aktywny:
    webserver.handleClient()    // obsługa serwera WWW
if warunki deep sleep spełnione:
    przejście do trybu głębokiego uśpienia
yield()                         // oddanie kontroli systemowi
```

W trakcie iteracji pętli głównej nie jest stosowana funkcja blokującego oczekiwania `delay()` znana z frameworku Arduino, co umożliwia zachowanie wysokiej responsywności na zdarzenia. Wszystkie operacje wymagające okresowego uruchamiania - zapis CSV, obsługa serwera WWW, próbkowanie napięcia baterii, wykorzystują mechanizm sprawdzania licznika czasu systemowego (`millis()`) w odniesieniu do zadanego interwału, określanego w niniejszej pracy jako bramka czasowa. W praktyce oznacza to, że pętla wykonuje się z maksymalną dostępną szybkością, a poszczególne zadania są aktywowane tylko wówczas, gdy upłynął wymagany czas od ich poprzedniego wykonania. Zasadę działania bramki czasowej ilustruje Listing 2.

Listing 2. Wzorzec bramki czasowej.

```
// --- Pętla główna (main.cpp) ---
void loop() {
    const uint32_t now = millis();
    SensorData d = sensorsRead();
    userInterfacePoll(now)
    // Bramka czasowa: zapis logu jazdy
    if (now - lastSampleTime >= SAMPLE_INTERVAL) {
        lastSampleTime = now;
        if (storage.isRideActive()) {
```

```

        storage.save(d);
    }
}
// Bramka czasowa: obsługa serwera WWW
if (now - lastWebServerTime >= WEB_POLL_INTERVAL) {
    lastWebServerTime = now;
    web.handleClient();
}
yield(); // oddanie CPU dla FreeRTOS
}

```

Pomimo jednowątkowej architektury, w obrębie pętli głównej współistnieją kilka mechanizmów czasowych pracujących z różnymi częstotliwościami, od ciągłego odpytywania interfejsu użytkownika i krótkich interwałów obsługi serwera WWW, przez próbkowanie czujnika inercyjnego w module `shockcapture.cpp`, aż po zapis logu jazdy realizowany ze zmienną częstotliwością zależną od bieżącego stanu ruchu pojazdu. Wszystkie te mechanizmy zrealizowano poprzez wspomniany wcześniej wzorzec bramki czasowej opartej o funkcję `millis()`. Najwyższą częstotliwość 200 Hz osiąga moduł `shockcapture.cpp`, dedykowany rejestrator pików akceleracji, stanowiący uzupełnienie głównego logu jazdy zapisywanego ze znacznie niższą częstotliwością. Moduł ten pełni wyłącznie rolę rejestratora danych sam algorytm detekcji upadku, opisany w podrozdziale 6.3, działa w module `sensors.cpp` na strumieniu danych z czujników.

Reguła nieblokującej pętli głównej dotyczy cyklicznych zadań pętli aplikacyjnej. Krótkotrwałe wywołania funkcji `delay()` występują w fazie inicjalizacji urządzenia, w szczególności procedury miękkiego resetu (*soft-reset*), wybudzenia z trybu uśpienia oraz konfiguracji filtrów cyfrowych czujnika inercyjnego MPU-6500, a także procedura uruchamiania modułu nawigacji satelitarnej Air530Z, uzasadniają zastosowanie synchronicznych pauz. Specyficzne mechanizmy oczekiwania stosowane w warstwie komunikacji z modemem GSM oraz w obsłudze zapytań HTTP serwera konfiguracyjnego, których natura wymaga sekwencyjnego przetwarzania, opisano odpowiednio w podrozdziałach 6.4 i 6.6.

6.2. Akwizycja danych z czujników

Moduł `sensors.cpp` stanowi główną warstwę akwizycji danych w urządzeniu. Odpowiada za inicjalizację oraz cykliczny odczyt wszystkich czujników środowiskowych i inercyjnych, parsowanie strumienia danych z odbiornika GPS, pomiar napięcia ogniwa zasilającego oraz wstępne filtrowanie sygnałów przed dalszym wykorzystaniem w pozostałych modułach systemu. Centralnym elementem modułu jest struktura danych `SensorData`, zawierająca komplet aktualnych pomiarów ze wszystkich źródeł, aktualizowana w każdej iteracji pętli głównej poprzez wywołanie funkcji `sensorsRead()`. Wynik tego wywołania przekazywany jest następnie do modułów rejestracji, detekcji upadku oraz serwera WWW w trybie serwisowym. Struktura danych `Sensor Data` została przedstawiona w Listing 3.

Listing 3. Struktura danych `SensorData`

```

struct SensorData {
    // Czujniki środowiskowe
    float temperature;
    float humidity;
    float pressure;

    // Pomiar napięcia ogniwa 18650
    float batteryCellVolts;
}

```

```

// Pomiar odczytu akcelerometru i żyrometru
float  accelX, accelY, accelZ;
float  gyroX, gyroY, gyroZ;

// Pomiar GPS
double latitude, longitude;
uint8_t sats;           // liczba satelit
bool hasFix;

// Dane jazdy
float  speed;           // km/h liczone z GPS
float  distance;        // całkowity dystans w metrach
float  elevation;       // wysokość w metrach
float  heading;         // kierunek w kątach

// Dane czasowe
uint32_t timestamp;    // uptime w sekundach
uint16_t year;         // rok
uint8_t month, day;    // miesiąc, dzień
char    localTime[9];  // "HH:MM:SS" czas lokalny

// Dane aktualnej jazdy
bool    isMoving;      // 1 jeśli prędkość > próg
uint32_t rideStartTime; // znacznik czasowy początku jazdy
uint32_t rideDuration; // sekundy od rozpoczęcia jazdy

// Dane wykrywacza wypadku
bool    impactDetected; // 1 jeśli wstrząs wykryty
float    impactForce;   // magnituda wstrząsu [g]
uint32_t lastImpactTime; // znacznik czasowy ostatniego wstrząsu
uint8_t  impactSeverity; // 0=brak, 1=lekki, 2=średni, 3=silny
bool    fallDetected;   // 1 jeśli wykryto wzorzec upadku
uint32_t fallStartTime; // znacznik czasu rozpoczęcia upadku
};

```

6.2.1. Inicjalizacja czujników

Inicjalizacja warstwy czujników realizowana jest w funkcji `sensorsInit()`, wywoływanej z funkcji `setup()` modułu `main.cpp`. Procedura ta przeprowadzana jest w określonej kolejności, zapewniającej dostępność magistrali komunikacyjnych przed konfiguracją układów peryferyjnych.

W pierwszym kroku uruchamiana jest magistrala I²C poprzez wywołanie `Wire.begin()`, na której pracują dwa układy pomiarowe - akcelerometr i żyroskop MPU-6500 oraz czujnik środowiskowy BME280. Magistrala konfigurowana jest z domyślną częstotliwością 100 kHz, wystarczającą dla obu układów, a piny SDA i SCL doprowadzone są do wyprowadzeń D4 i D5 modułu XIAO ESP32-C6 (odpowiadających wewnętrznym GPIO22 i GPIO23 mikrokontrolera).

Następnie konfigurowany jest czujnik inercyjny MPU-6500 poprzez bezpośredni zapis rejestrów I²C. Procedura inicjalizacji obejmuje weryfikację identyfikatora układu (rejestr `WHO_AM_I` o wartości `0x70`), wybudzenie układu z trybu uśpienia (rejestr `PWR_MGMT_1`), wybór źródła zegara, konfigurację cyfrowego filtra dolnoprzepustowego (*Digital Low Pass Filter*) oraz zakresu pomiarowego akcelerometru - zakres wybierany jest na podstawie wartości zapisanej w pliku konfiguracyjnym `dev.json` (2, 4, 8 lub 16 g, domyślnie 8 g). Czujnik środowiskowy BME280 inicjalizowany jest z wykorzystaniem biblioteki Adafruit BME280 Library, pod adresem I²C `0x77`. Bezpośrednio po inicjalizacji wykonywany jest jednorazowy odczyt parametrów środowiskowych, którego wynik

zachowywany jest w buforach pamięci podręcznej i wykorzystywany przez moduł `sensors.cpp` do momentu kolejnej próbkę - mechanizm szczegółowo opisano w punkcie 6.2.6.

Odbiornik GPS Air530Z komunikuje się z mikrokontrolerem przez interfejs UART1 obsługiwany w środowisku Arduino przez obiekt `Serial1`, z prędkością transmisji 9600 baud. Po inicjalizacji portu UART odbiornik rozpoczyna automatyczną emisję komunikatów w standardzie NMEA 0183, które parsowane są w czasie rzeczywistym przez bibliotekę `TinyGPSPlus`. Pomiar napięcia ogniwa zasilającego realizowany jest natomiast przez wbudowany w mikrokontroler dwunastobitowy przetwornik analogowo-cyfrowy, podłączony do wyprowadzenia GPIO6 (oznaczonego jako MTCK na płytce XIAO ESP32-C6), do którego doprowadzony jest sygnał z rezystorowego dzielnika napięcia opisanego w podrozdziale 5.2.2.

Po wykonaniu inicjalizacji wszystkich czujników wykonywane jest jednorazowe wczytanie zapisanej kalibracji jednostki inercyjnej z pliku `/Config/imu_cal.json` (jeśli plik istnieje), a wyznaczone wcześniej wartości przesunięć (*offsets*) trzech osi akcelerometru są stosowane przez funkcję `sensorsApplyImuCal()`. Mechanizm kalibracji szczegółowo omówiono w punkcie 6.2.3.

6.2.2. Częstotliwości próbkowania

Poszczególne czujniki próbkowane są z różnymi częstotliwościami, dobranymi indywidualnie do charakterystyki sygnału oraz zapotrzebowania systemu na dane. Zestawienie częstotliwości akwizycji przedstawiono w Tabeli 9.

Tabela 9. Częstotliwość akwizycji danych z czujników

Źródło danych	Częstotliwość [Hz]	Kontekst pomiaru
MPU-6500 - odczyt, detekcja	75	<code>sensorsRead()</code> w pętli głównej, <code>updateFallDetecion()</code>
MPU-6500 - log jazdy	1	Zapis do strumienia CSV
MPU-6500 - shockcapture	200	Osobny strumień <code>sensorReadImuOnly()</code>
BME280 (środowisko)	1	Odczyt danych środowiskowych: <code>Temperature</code> , <code>humidity</code> , <code>pressure</code>
GPS Air530z (NMEA)	1	Odczyt danych GPS <code>latitude</code> , <code>longitude</code> , <code>sats</code> , <code>hasFix</code>
Pomiar napięcia ogniwa	0,5	Pomiar <code>batteryCellVolts</code>

Najwyższą częstotliwość 200 Hz osiąga próbkowanie czujnika inercyjnego w module `shockcapture`. Realizowane jest ono wyłącznie poprzez dedykowaną funkcję `sensorsReadImuOnly()`, która pomija pełną aktualizację struktury `SensorData` oraz odczyt pozostałych czujników, ograniczając się do bezpośredniego odczytu rejestrów MPU-6500. Takie rozwiązanie minimalizuje czas wykonywania pojedynczej próby do kilkudziesięciu mikrosekund, co umożliwia utrzymanie stabilnego interwału 5 ms w obrębie wspólnej pętli głównej, mimo braku dedykowanego zadania FreeRTOS.

6.2.3. Akwizycja danych z czujnika inercyjnego i kalibracja

Czujnik inercyjny MPU-6500 dostarcza sześciu kanałów pomiarowych: trzech osi przyspieszenia (a_x , a_y , a_z) oraz trzech osi prędkości kątowej (g_x , g_y , g_z). Wartości odczytywane są w jednostkach surowych z szesnastobitowych rejestrów wyjściowych czujnika i przeliczane na jednostki fizyczne

(przyspieszenie w g, prędkość kątowna w °/s) zgodnie z aktualnie skonfigurowanym zakresem pomiarowym.

Surowe odczyty akcelerometru obciążone są niezerowym przesunięciem (*bias*) wynikającym z tolerancji wykonania czujnika oraz drobnych odchyłek montażu modułu MPU-92.65 na płycie bazowej. Aby skorygować ten błąd, zaimplementowano procedurę kalibracji jednostki inercyjnej, wywoływaną z poziomu interfejsu serwisowego WWW (endpoint POST /api/imu/calibrate). Procedura ta wykonuje serię dwustu próbek akcelerometru przy nieruchomo leżącym urządzeniu, oblicza wartości średnie z każdej osi, a następnie wyznacza przesunięcia jako różnice między średnimi a oczekiwanymi wartościami referencyjnymi (zero dla osi poziomych, jeden g dla osi pionowej w kierunku przeciwnym do wektora przyspieszenia ziemskiego). Wyznaczone przesunięcia są zapisywane do pliku konfiguracyjnego /Config/imu_cal.json na karcie microSD i automatycznie wczytywane przy każdym uruchomieniu urządzenia.

W trakcie normalnej pracy każdy odczyt z akcelerometru korygowany jest poprzez odjęcie odpowiedniego przesunięcia ($ax_corrected = ax_raw - offsetX$), co realizowane jest w funkcji `sensorsApplyImuCal()`. Kalibracja żyroskopu nie jest implementowana - przesunięcia osi prędkości kątowej są w MPU-6500 znacznie mniejsze niż w przypadku akcelerometru i nie wpływają istotnie na działanie algorytmu detekcji upadku, który operuje wyłącznie na magnitudzie wektora przyspieszenia.

6.2.4. Pomiar napięcia ogniwa

Pomiar napięcia ogniwa zasilającego realizowany jest przez wbudowany w ESP32-C6 dwunastobitowy przetwornik analogowo-cyfrowy, podłączony do wyprowadzenia GPIO6 mikrokontrolera. Sygnał z ogniwa Sony VTC6 o napięciu nominalnym 3,7 V doprowadzony jest do przetwornika przez rezystorowy dzielnik zapewniający stosunek podziału 1:2, opisany w podrozdziale 5.2.2. Stała przeliczeniowa dzielnika wynosi zatem 2, co oznacza, że napięcie na wejściu przetwornika stanowi połowę rzeczywistego napięcia ogniwa.

Pomiar realizowany jest co 2000 ms, jednocześnie z odczytami pozostałych czujników w funkcji `sensorsRead()`. Pojedynczy pomiar składa się z ośmiu kolejnych odczytów przetwornika, których średnia stanowi chwilową wartość napięcia na wejściu ADC. Wartość ta jest następnie przeliczana na rzeczywiste napięcie ogniwa według zależności (1):

$$U_{ogniwa} = U_{odczyt} \cdot 2,0 \quad (1)$$

gdzie:

U_{ogniwa} – Obliczone napięcie ogniwa 18650

U_{odczyt} – Zmierzona wartość napięcia na wejściu ADC

Tak wyznaczona wartość napięcia poddawana jest dalszemu wygładzaniu poprzez filtr wykładniczy o współczynniku wygładzania $\alpha = 0,20$, co odpowiada efektywnej stałej czasowej rzędu dziesięciu sekund. Filtr ten eliminuje krótkotrwałe wahania napięcia spowodowane impulsowym poborem prądu przez modem GSM lub odbiornik GPS, zapewniając stabilną wartość napięcia używaną w dalszych obliczeniach.

6.2.5. Akwizycja sygnału z odbiornika GPS

Dane nawigacyjne pozyskiwane są z odbiornika GPS Air530Z, komunikującego się z mikrokontrolerem przez interfejs UART1 z prędkością 9600 baud. Odbiornik automatycznie emituje komunikaty w standardzie NMEA 0183, w tym między innymi GPRMC (czas, pozycja, prędkość, status), GPGLL (pozycja, liczba widocznych satelitów, jakość fixu) oraz \$GPGSV (lista śledzonych satelitów). Strumień znakowy z UART1 odczytywany jest w każdej iteracji pętli głównej i przekazywany do parsera biblioteki TinyGPSPlus, która utrzymuje wewnętrzny stan ostatnich poprawnie zinterpretowanych pól.

Dostęp do pojedynczych pól danych nawigacyjnych odbywa się przez wywołania metod biblioteki w funkcji `sensorsRead()`. Pozyskiwane są w ten sposób: szerokość i długość geograficzna, liczba widocznych satelitów, czas UTC, prędkość chwilowa w kilometrach na godzinę, wysokość nad poziomem morza, kąt kierunku ruchu (*heading*) oraz wskaźnik aktualnego ustalenia pozycji (*hasFix*). Czas UTC odczytany z komunikatów GPS wykorzystywany jest również do synchronizacji wewnętrznego zegara RTC mikrokontrolera w strefie czasowej Europy Środkowej, na podstawie którego generowane są znaczniki czasowe wpisów w logu jazdy.

Sygnał prędkości z odbiornika GPS poddawany jest dodatkowej walidacji w celu odrzucenia wartości jawnie nierealnych, mogących wynikać z błędów odbioru przy słabym sygnale satelitarnym. Wartości prędkości przekraczające próg `MAX_VALID_SPEED` (100 km/h) traktowane są jako artefakty i odrzucane na poziomie modułu `sensors.cpp`, a w polu `is_moving` struktury `SensorData` ustawiana jest flaga ostatniej znanej wiarygodnej wartości. Mechanizm ten zapobiega błędnym aktywacjom progów ruchowych wykorzystywanych w logice zarządzania trybem głębokiego uśpienia.

Dodatkowo, gdy odbiornik raportuje fix, ale liczba widocznych satelitów spada poniżej minimalnego progu wiarygodności, w module `sensors.cpp` aktywowana jest tymczasowa flaga oznaczająca zaszumiony sygnał GPS (*GPS noise*). W tym stanie dane pozycyjne są nadal odczytywane i logowane, jednak nie są wykorzystywane do podejmowania decyzji w algorytmie detekcji upadku - w szczególności warunek niskiej prędkości weryfikujący nieruchomość urządzenia po wstrząsie traktowany jest jako niespełniony, co dodatkowo zwiększa odporność algorytmu na fałszywe alarmy w warunkach złego pokrycia satelitarnego.

6.2.6. Akwizycja parametrów środowiskowych

Czujnik środowiskowy BME280 dostarcza trzech parametrów otoczenia: temperatury w stopniach Celsjusza, wilgotności względnej w procentach oraz ciśnienia atmosferycznego w hektopaskalach. Odczyt realizowany jest z wykorzystaniem biblioteki Adafruit BME280 Library, pracującej w trybie ciągłej akwizycji z wewnętrznym uśrednianiem próbek konfigurowanym przez parametry rejestrowe układu.

Ze względu na niską dynamikę zmian parametrów środowiskowych, odczyt czujnika BME280 świadomie odseparowano od tempa głównej pętli akwizycji - pomiary realizowane są z częstotliwością jednego herca poprzez dedykowaną bramkę czasową w funkcji `sensorsRead()`. Pomiedzy kolejnymi odczytami wartości zachowywane są w buforach pamięci podręcznej (`_bmeTempCached`, `_bmeHumCached`, `_bmePresCached`), które są zwracane modułom konsumującym dane bez ponownego odpytywania czujnika. Pierwsza próbka pobierana jest jednorazowo w funkcji `sensorsInit()`, zapewniając dostępność danych natychmiast po uruchomieniu urządzenia.

Takie rozwiązanie przynosi dwie wymierne korzyści techniczne. Po pierwsze, skraca czas wykonania pojedynczej iteracji pętli głównej, ponieważ większość iteracji nie wymaga już kosztownej operacji odczytu rejestrów czujnika BME280 przez magistralę I²C. Po drugie, zmniejsza obciążenie magistrali I²C współdzielonej między czujnikiem BME280, akcelerometrem MPU-6500 oraz ekspanderem MCP23008 - co jest istotne w kontekście wysokoczęstotliwościowego próbkowania jednostki inercyjnej w module `shockcapture` (200 Hz).

Pomiary środowiskowe wykorzystywane są wyłącznie w celach rejestracyjnych - wartości temperatury, wilgotności i ciśnienia zapisywane są w odpowiednich kolumnach logu jazdy CSV (opisanego w podrozdziale 6.5) oraz wykorzystywane do obliczania wysokości barometrycznej. Pomiary te nie wpływają bezpośrednio na działanie algorytmu detekcji upadku ani na procedurę alarmową - pełnią rolę kontekstową, umożliwiając późniejszą analizę warunków, w jakich rejestrowane były dane z pozostałych czujników. Lokalizacja czujnika BME280 na płytce bazowej została dobrana z uwzględnieniem odległości od źródeł ciepła (modemu SIM800L oraz przetwornicy TPS63020), zgodnie z założeniami opisanymi w podrozdziale 5.3.4.

6.3. Algorytm detekcji upadku

Algorytm detekcji upadku stanowi centralny element funkcjonalny urządzenia. Jego zadaniem jest automatyczne rozpoznanie sytuacji krytycznej, czyli zdarzenia, w którym rowerzysta uległ wypadkowi i pozostaje nieruchomy oraz uruchomienie procedury alarmowej opisanej w podrozdziale 6.4. Algorytm zaimplementowano w module `sensors.cpp`, w funkcji `updateFallDetection()`, wywoływanej z głównej pętli `loop()` w każdej iteracji.

Detekcja oparta jest wyłącznie na sygnale z jednostki inercyjnej MPU-6500 oraz danych prędkościowych z odbiornika GPS Air530Z. Zaprojektowany algorytm wykorzystuje wieloetapową analizę progową z weryfikacją czasową, zapewniającą wysoką odporność na fałszywe alarmy przy zachowaniu krótkiego czasu reakcji.

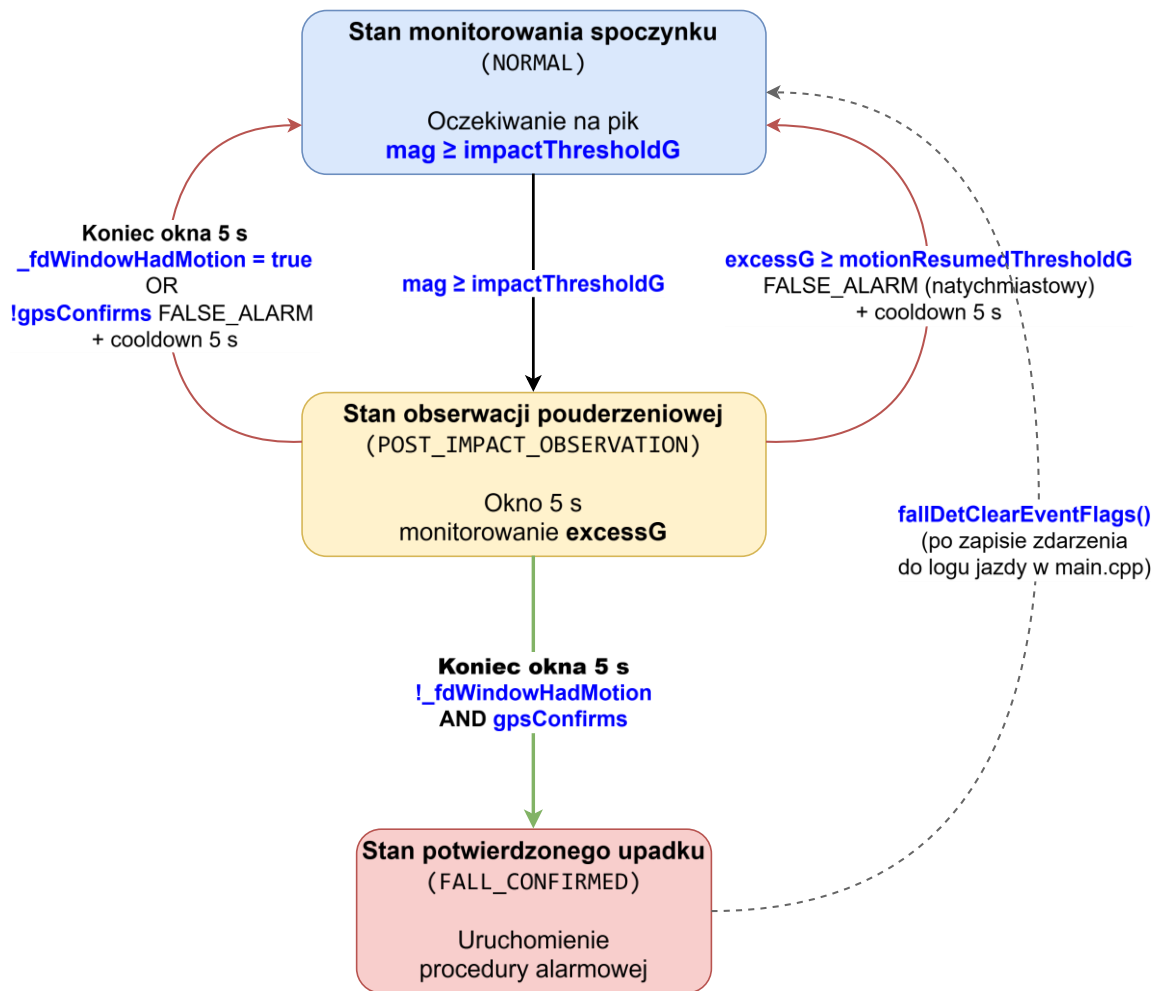
6.3.1. Architektura algorytmu - maszyna stanów

Logika detekcji zorganizowana jest jako trójstanowa maszyna stanów, w której każdy stan realizuje wyodrębniony etap analizy zdarzenia. Wyróżniono następujące stany:

- **stan monitorowania spoczynku** (`NORMAL`) - bazowy stan oczekiwania na zdarzenie potencjalnego upadku;
- **stan obserwacji poudarzeniowej** (`POST_IMPACT_OBSERVATION`) - uruchamiany po wykryciu uderzenia, w trakcie którego algorytm weryfikuje, czy zdarzenie spełnia kryteria rzeczywistego upadku;
- **stan potwierdzonego upadku** (`FALL_CONFIRMED`) - końcowa klasyfikacja zdarzenia jako prawdziwego upadku, uruchamiająca procedurę alarmową.

Schemat maszyny stanów wraz z warunkami przejść przedstawiono na Rysunek 38.

Maszyna stanów algorytmu detekcji upadku



Rysunek 38. Maszyna stanów algorytmu detekcji upadku (źródło: opracowanie własne).

Stan algorytmu utrzymywany jest w zmiennej `_fdState` typu `FallDetectionState`. Przejścia między stanami uwarunkowane są spełnieniem konkretnych kryteriów opisanych w kolejnych punktach. Parametry konfiguracyjne algorytmu - progi detekcji oraz długości okien czasowych - przechowywane są w strukturze `ShockConfig`, której pola zapisywane są w pliku konfiguracyjnym `/Config/shock.json` na karcie microSD i wczytywane przy uruchomieniu urządzenia. W trakcie pracy parametry są mapowane do struktury `FallDetParams` przekazywanej do funkcji `updateFallDetection()`. Wartości progów są również dostępne do modyfikacji poprzez interfejs serwisowy WWW (endpoint `POST /shock/config`). Struktura `FallDetParams` została przedstawiona w Listing 4.

Listing 4. Struktura `FallDetParams`.

```

struct FallDetParams {
    float impactThresholdG;           // próg piku magnitudy [g]
    float observationWindowS;       // długość okna obserwacji [s]
    float immobilityThresholdG;     // powyżej tego = brak bezruchu
    float motionResumedThresholdG;  // powyżej tego = ruch wrócił
    float gpsLowSpeedThresholdKmh;  // poniżej tego = potwierdza upadek
    float impactCooldownS;         // cooldown po fałszywym alarmie
    float moderateImpactG;         // próg umiarkowanego uderzenia
    float severeImpactG;           // próg silnego uderzenia

```

};

Parametry konfiguracyjne algorytmu zostały dokładniej opisane w podrozdziale 6.3.5.

Sygnałem wejściowym dla algorytmu jest magnituda wektora przyspieszenia, wyznaczana w każdej iteracji według zależności (2):

$$mag = \sqrt{(ax^2 + ay^2 + az^2)} \quad (2)$$

gdzie:

ax, ay, az – wartości przyspieszenia w trzech osiach układu współrzędnych czujnika,

6.3.2. Faza 1 - Monitorowanie spoczynku

W stanie monitorowania spoczynku algorytm prowadzi ciągle monitorowanie sygnału inercyjnego, oczekując na wystąpienie zdarzenia mogącego oznaczać upadek. W każdej iteracji pętli głównej wyznaczana jest aktualna magnituda przyspieszenia, która następnie porównywana jest z progiem detekcji uderzenia (`impactThresholdG`).

W momencie wykrycia przekroczenia progu algorytm dokonuje klasyfikacji intensywności uderzenia na podstawie wartości magnitudy porównywanej z dwoma dodatkowymi progami:

- **uderzenie lekkie** - gdy magnituda mieści się w przedziale od `impactThresholdG` do `moderateImpactG`;
- **uderzenie umiarkowane** - gdy magnituda przekracza `moderateImpactG`;
- **uderzenie silne** - gdy magnituda przekracza `severeImpactG`.

Klasyfikacja intensywności uderzenia nie wpływa bezpośrednio na decyzję o uruchomieniu alarmu, lecz zapisywana jest w metadanych zdarzenia i wykorzystywana w późniejszej analizie. Po wyznaczeniu intensywności algorytm dokonuje przejścia do stanu obserwacji poudzerzeniowej, zapisując znacznik czasowy zdarzenia (`_fdImpactStartMs = millis()`).

6.3.3. Faza 2 - Obserwacja poudzerzeniowa

Faza obserwacji poudzerzeniowej stanowi kluczowy element algorytmu, pozwalający na rozróżnienie pomiędzy:

- **prawdziwym upadkiem** - po którym rowerzysta pozostaje nieruchomy na podłożu (i wymaga pomocy);
- **falszywym alarmem** - wynikającym z silnego, ale nieszkodliwego wstrząsu, po którym jazda kontynuowana jest normalnie (np. zjazd z krawężnika, gwałtowne hamowanie, uderzenie roweru o przeszkodę przy postoju).

W tym stanie algorytm uruchamia okno obserwacji o długości `observationWindowS` (domyślnie 5 sekund), w trakcie, którego monitorowane są dwa kryteria nieruchomości urządzenia. Dodatkowo, w każdej iteracji wewnątrz okna sprawdzany jest warunek bezpośredniego powrotu ruchu, którego spełnienie powoduje natychmiastowe zakończenie fazy obserwacji.

Warunek powrotu ruchu (sprawdzany w każdej iteracji wewnątrz okna obserwacji) - odchylenie magnitudy od wartości grawitacyjnej (`excessG`) porównywane jest z progiem powrotu ruchu (`motionResumedThresholdG`). Przekroczenie progu interpretowane jest jako wznowienie aktywności użytkownika i powoduje natychmiastowe przejście do stanu monitorowania spoczynku z klasyfikacją fałszywego alarmu oraz aktywacją okresu odporności. Mechanizm ten zapewnia szybką reakcję algorytmu w sytuacjach, w których po krótkotrwałym wstrząsie rowerzysta natychmiast

kontynuuje jazdę co typowo występuje w scenariuszach takich jak zjazd z krawężnika, gwałtowne hamowanie czy uderzenie roweru o przeszkodę podczas postoju.

Monitorowanie ruchu w oknie (sprawdzone w każdej iteracji) w przypadku, gdy warunek powrotu ruchu nie jest spełniony, algorytm sprawdza odchylenie magnitudy od wartości grawitacyjnej. Jeśli odchylenie przekracza próg nieruchomości (`immobilityThresholdG`), w wewnętrznej fladze stanu okna (`_fdWindowHadMotion`) zapisywana jest informacja o wystąpieniu ruchu. Flaga ta zachowywana jest do końca okna obserwacji i wykorzystywana w końcowej klasyfikacji zdarzenia.

Weryfikacja końcowa (po upływie okna obserwacji), gdy upłynął pełny okres `observationWindows`, algorytm sprawdza dwa warunki łącznie:

- `!_fdWindowHadMotion` - w trakcie całego okna nie wystąpiło istotne odchylenie magnitudy od wartości grawitacyjnej;
- `gpsConfirms` - odbiornik GPS potwierdza nieruchomość pojazdu lub jego sygnał nie jest dostępny.

Warunek `gpsConfirms` jest złożony i obejmuje trzy sytuacje, w których nieruchomość pojazdu uznawana jest za potwierdzoną: brak ustalenia pozycji (`!hasFix`), wyłączenie weryfikacji GPS z konfiguracji (`gpsLowSpeedThresholdKmh ≤ 0`) lub raportowana prędkość pojazdu poniżej progu (`speed < gpsLowSpeedThresholdKmh`). Mechanizm pomocniczy filtra zaszumionego sygnału GPS (omówiony w podrozdziale 6.2.5) wymusza zerową wartość prędkości w sytuacji niespójności z czujnikiem inercyjnym, co automatycznie spełnia trzecie kryterium.

Jeśli oba warunki końcowe są spełnione, algorytm dokonuje przejścia do stanu potwierdzonego upadku. W przeciwnym wypadku, gdy wystąpił ruch w oknie lub prędkość GPS przekroczyła próg, następuje powrót do stanu monitorowania spoczynku z klasyfikacją fałszywego alarmu i aktywacją okresu odporności.

Dwukryterialny mechanizm weryfikacji nieruchomości znacząco zwiększa specyficzność detekcji, sytuacja, w której urządzenie wykazuje brak ruchu jednocześnie z perspektywy czujnika inercyjnego (brak drgań) oraz odbiornika GPS (zerowa prędkość lub brak fixu) jest charakterystyczna dla rzeczywistego unieruchomienia rowerzysty na podłożu po wypadku, podczas gdy fałszywe alarmy wynikające z silnych wstrząsów podczas jazdy są niemal zawsze odrzucane przez kryterium prędkościowe.

6.3.4. Faza 3 - Klasyfikacja zdarzenia i okres odporności

Stan potwierdzonego upadku osiągany jest, gdy spełnione zostały oba warunki końcowe fazy obserwacji. Jest to końcowa klasyfikacja zdarzenia jako prawdziwego upadku, uruchamiająca dalsze działania systemu.

Po osiągnięciu stanu potwierdzonego upadku funkcja `updateFallDetection()` zwraca wynik `FallDetectionResult::FALL_CONFIRMED`. Na podstawie tego wyniku główna pętla `loop()` w `main.cpp` inicjuje obsługę zdarzenia. Zapis odpowiednich metadanych do logu jazdy CSV oraz uruchomienie procedury alarmowej szczegółowo opisanej w podrozdziale 6.4.

Stan potwierdzonego upadku utrzymywany jest do momentu jawnego wyzerowania przez funkcję `fallDetClearEventFlags()`, wywoływanej w `main.cpp` po zapisie zdarzenia do logu jazdy. Mechanizm ten zapewnia, że informacja o wykrytym upadku nie zostanie utracona pomiędzy iteracjami pętli, co byłoby istotne w przypadku, gdyby zapis danych do logu wymagał kilku iteracji ze względu na ograniczenia karty microSD.

Okres odporności (`impactCooldownS`) aktywowany jest natomiast po każdym fałszywym alarmie niezależnie od mechanizmu jego wystąpienia (powrót ruchu, ruch w oknie lub niezerowa prędkość GPS). W trakcie okresu odporności kolejne przekroczenia progu uderzenia są ignorowane, a algorytm pozostaje w stanie monitorowania spoczynku. Świadomie zrezygnowano z aktywacji okresu odporności po stanie potwierdzonego upadku w sytuacji wystąpienia rzeczywistego upadku system

powinien zachować możliwość rejestracji ewentualnych następnych zdarzeń (na przykład wtórnych uderzeń), które mogą być istotne dla późniejszej analizy.

Aktywacja okresu odporności po fałszywym alarmie powiązana jest dodatkowo z wyzerowaniem wewnętrznych flag stanu uderzenia (`_fdPendingImpact`), co zapobiega propagacji informacji o zdarzeniu odrzuconym do strumienia rejestracji CSV. Mechanizm ten zapewnia integralność danych w logu jazdy. W pliku CSV zarejestrowane są wyłącznie zdarzenia zaklasyfikowane jako rzeczywiste upadki, bez śladów zdarzeń odrzuconych w fazie obserwacji.

6.3.5. Zestawienie parametrów algorytmu

Wszystkie parametry konfiguracyjne algorytmu detekcji upadku zestawiono w Tabeli 5.4 wraz z ich domyślnymi wartościami oraz krótkim opisem znaczenia funkcjonalnego. Parametry przechowywane są w strukturze `ShockConfig`, zapisywanej w pliku `/Config/shock.json` i mogą być modyfikowane przez interfejs serwisowy WWW. Parametry zostały przedstawione w Tabeli 10.

Tabela 10. Parametry konfiguracyjne algorytmu detekcji upadku.

Parametr	Znaczenie
<code>impactThresholdG</code>	Próg magnitudy uruchamiający przejście z fazy monitorowania spoczynku do fazy obserwacji poudzerzeniowej. Wartość konserwatywna, zapewniająca wysoką czułość detekcji.
<code>moderateImpactG</code>	Próg klasyfikacji uderzenia jako umiarkowanego (intensywność = 2). Wykorzystywany w metadanych zdarzenia.
<code>severeImpactG</code>	Próg klasyfikacji uderzenia jako silnego (intensywność = 3). Wykorzystywany w metadanych zdarzenia.
<code>observationWindowS</code>	Długość okna obserwacji poudzerzeniowej, w trakcie, którego weryfikowana jest nieruchomość urządzenia przed wydaniem ostatecznej klasyfikacji.
<code>immobilityThresholdG</code>	Próg odchylenia magnitudy od wartości grawitacyjnej (<code>excessG</code>), powyżej którego w danej iteracji ustawiana jest flaga ruchu w oknie obserwacji.
<code>motionResumedThresholdG</code>	Próg odchylenia magnitudy od wartości grawitacyjnej (<code>excessG</code>), którego przekroczenie w trakcie okna obserwacji powoduje natychmiastową klasyfikację zdarzenia jako fałszywego alarmu.
<code>gpsLowSpeedThresholdKmh</code>	Maksymalna prędkość raportowana przez odbiornik GPS, przy której pojazd uznawany jest za nieruchomy. Wartość 0 oznacza wyłączenie weryfikacji GPS w decyzji algorytmu.
<code>impactCooldownS</code>	Czas blokady detekcji kolejnego uderzenia po wystąpieniu fałszywego alarmu, zapobiegający łańcuchowi błędnych klasyfikacji w sytuacjach o intensywnym sygnale inercyjnym.

6.3.6. Fragment implementacji

Listing 5 prezentuje kluczowy fragment implementacji algorytmu, ukazujący strukturę maszyny stanów oraz logikę przejść między fazami. Fragment ten stanowi rdzeń funkcji `updateFallDetection()`.

Listing 5. Fragment implementacji algorytmu detekcji upadku

```
FallDetectionResult updateFallDetection(SensorData &d, const
FallDetParams &p) {
    const float mag = sqrtf(d.accelX * d.accelX
                          + d.accelY * d.accelY
                          + d.accelZ * d.accelZ);
    const float excessG = fabsf(mag - 1.0f);
    const uint32_t now = millis();

    switch (_fdState) {
        // Faza 1: monitorowanie spoczynku
        case FallDetectionState::NORMAL:
            if (now < _fdCooldownUntilMs) {
                return FallDetectionResult::NO_EVENT;
            }
            if (mag >= p.impactThresholdG) {
                _fdState = FallDetectionState::POST_IMPACT_OBSERVATION;
                _fdImpactStartMs = now;
                _fdWindowHadMotion = false;
                _fdPendingImpact = true;
                _fdImpactSeverity = classifySeverity(mag, p);
                return FallDetectionResult::IMPACT_DETECTED;
            }
            return FallDetectionResult::NO_EVENT;
        // Faza 2: obserwacja poudrzeniowa
        case FallDetectionState::POST_IMPACT_OBSERVATION:
            // Warunek powrotu ruchu (natychmiastowy fałszywy alarm)
            if (excessG >= p.motionResumedThresholdG) {
                _fdState = FallDetectionState::NORMAL;
                _fdCooldownUntilMs = now +
                (uint32_t) (p.impactCooldownS * 1000.0f);
                _fdPendingImpact = false;
                return FallDetectionResult::FALSE_ALARM;
            }

            // Monitorowanie ruchu w oknie obserwacji
            if (excessG >= p.immobilityThresholdG) {
                _fdWindowHadMotion = true;
            }

            // Weryfikacja końcowa po upływie okna
            if (now - _fdImpactStartMs >=
            (uint32_t) (p.observationWindowS * 1000.0f)) {
                const bool gpsConfirms = !d.hasFix
                || (p.gpsLowSpeedThresholdKmh <= 0.0f)
                || (d.speed < p.gpsLowSpeedThresholdKmh);

                if (!_fdWindowHadMotion && gpsConfirms) {
                    _fdState = FallDetectionState::FALL_CONFIRMED;
                    return FallDetectionResult::FALL_CONFIRMED;
                }
            }
    }
}
```

```

        _fdState = FallDetectionState::NORMAL;
        _fdCooldownUntilMs = now +
(uint32_t) (p.impactCooldownS * 1000.0f);
        _fdPendingImpact = false;
        return FallDetectionResult::FALSE_ALARM;
    }
    return FallDetectionResult::NO_EVENT;

    case FallDetectionState::FALL_CONFIRMED:
        // Stan utrzymywany do fallDetClearEventFlags()
        return FallDetectionResult::NO_EVENT;
    }
    return FallDetectionResult::NO_EVENT;
}

```

6.4. Procedura alarmowa i komunikacja GSM

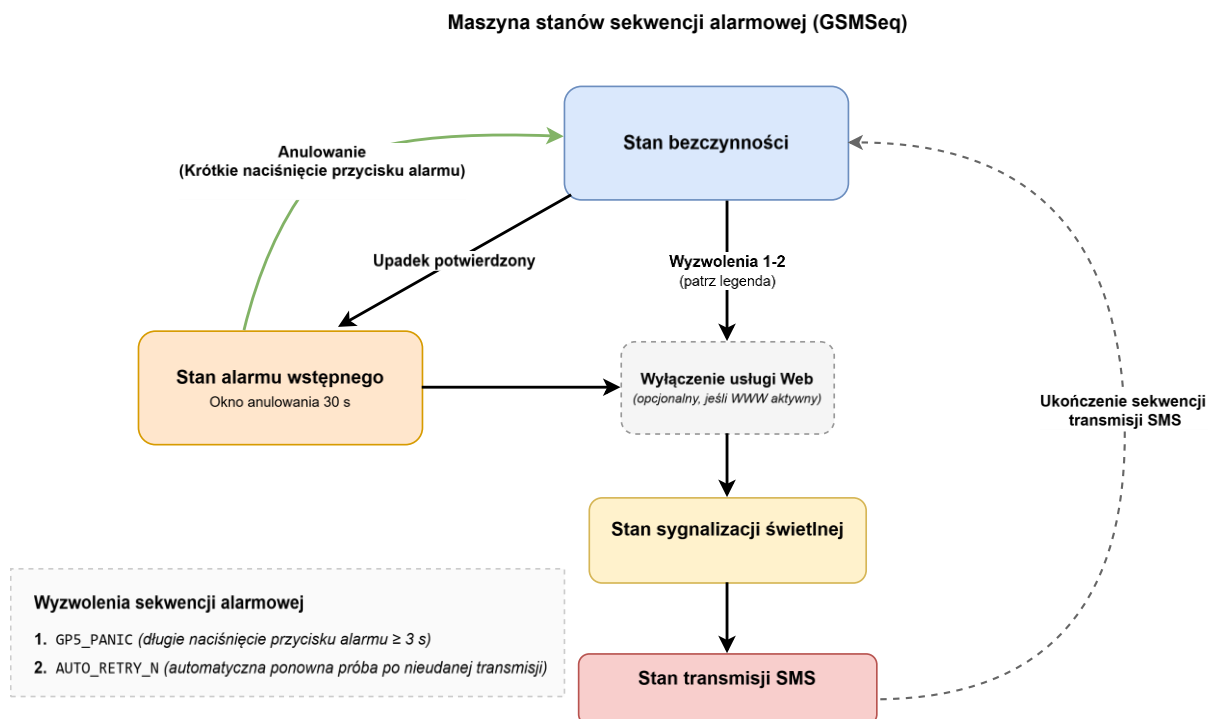
Procedura alarmowa stanowi reakcję systemu na wykrycie sytuacji krytycznej, potwierdzonego upadku rowerzysty lub świadomego wezwania pomocy przez użytkownika. Jej zadaniem jest niezawodne dostarczenie wiadomości tekstowej SMS na wcześniej skonfigurowany numer alarmowy, zawierającej znacznik czasowy oraz aktualną pozycję geograficzną urządzenia.

Procedura zaimplementowana została w dwóch warstwach architektonicznych. Warstwę logiki decyzyjnej stanowi moduł `userInterface.cpp`, w którym zaimplementowano maszynę stanów sekwencji alarmowej oraz mechanizm okna anulowania alarmu wstępnego. Warstwę niskopoziomowej komunikacji z modemem GSM stanowi moduł `communication.cpp`, realizujący sekwencję komend AT przez interfejs UART oraz obsługę odpowiedzi modemu. Taki podział odpowiedzialności umożliwia jasną separację logiki "kiedy wysłać SMS" (`userInterface.cpp`) od logiki "jak wysłać SMS" (`communication.cpp`).

6.4.1. Architektura sekwencji alarmowej

Sekwencja alarmowa zorganizowana jest jako pięciostanowa maszyna stanów reprezentowana przez typ wyliczeniowy `GSMSeq` w module `userInterface.cpp`. Każdy stan realizuje wyodrębniony etap procesu. Od oczekiwania na zdarzenie, przez sygnalizację świetlną i sterowanie zasilaniem modemu, aż po właściwą transmisję wiadomości. Schemat maszyny stanów sekwencji alarmowej Rysunek 39. Wyróżniono następujące stany:

- **stan bezczynności** - bazowy stan oczekiwania, w którym nie jest realizowana żadna sekwencja alarmowa;
- **stan alarmu wstępnego** - okno czasowe umożliwiające użytkownikowi anulowanie alarmu, aktywne wyłącznie w trybie pracy po wykryciu upadku;
- **stan wygaszania serwera WWW** - opcjonalny stan przejściowy, uruchamiany, gdy w momencie alarmu aktywny był interfejs serwisowy WWW;
- **stan sygnalizacji świetlnej** - sekwencja migania diody LED alarmu (GP3), poprzedzająca uruchomienie modemu;
- **stan transmisji SMS** - najdłuższy stan sekwencji, w trakcie którego zasilany jest modem SIM800L i realizowana właściwa transmisja wiadomości.



Rysunek 39. Schemat maszyny stanów sekwencji alarmowej (źródło: opracowanie własne).

Stan sekwencji utrzymywany jest w zmiennej `sGSM` typu `GSMSeq`, aktualizowanej w każdej iteracji pętli głównej w funkcji `userInterfacePoll()`. Wyzwolenie sekwencji realizowane jest przez wywołanie funkcji wewnętrznej `_startAlarmSequence()`, do której przekazywany jest powód uruchomienia, wykorzystywany w późniejszym logowaniu diagnostycznym (`FALL_CONFIRMED`, `GP5_FALL`, `GP5_PANIC`, `PRE_ALARM_TIMEOUT`, `AUTO_RETRY_N`, `AFTER_HARD_RESET`).

Funkcja `_startAlarmSequence()` realizuje trzy zadania inicjalizacyjne. Po pierwsze, inkrementuje licznik prób (`smsCurrentAttempt`) w pliku konfiguracyjnym `/Config/dev.json`, zapewniający odporność mechanizmu ponawiania próby (*retry*) na nagłe awarie zasilania. Po drugie, rozpoczyna sesję diagnostyczną w mechanizmie `smsDebugBeginSession()`, której wyniki zapisywane są w pliku `/Logs/sms_debug.log` na karcie microSD. Po trzecie, decyduje o ścieżce dalszego przejścia, gdy aktywny był serwer WWW, sekwencja kierowana jest do stanu wygaszania serwera, w przeciwnym wypadku bezpośrednio do stanu sygnalizacji świetlnej.

Kluczowym etapem sekwencji jest stan transmisji SMS (`HoldGp0`). W momencie wejścia w ten stan, poprzez funkcję `enterGp0Hold()`, pin GP0 ekspandera MCP23008 ustawiany jest w stan wysoki - co odpowiada aktywacji sygnału *enable* przetwornicy TPS63020 zasilającej modem SIM800L (opisanej w podrozdziale 5.2.3). Modem rozpoczyna procedurę rozruchu, której pełne ustabilizowanie wymaga około 5 sekund. W tym czasie wykonywane są dwie operacje pomocnicze: cykliczne wywoływanie funkcji `communicationGsmEarlyWakeDuringHold()` (od około 2500 ms po włączeniu zasilania), pozwalającej na wstępną inicjalizację portu szeregowego i konfigurację modemu w trybie wstrzymanej rejestracji sieci, oraz cykliczne monitorowanie warunku zakończenia sekwencji.

Po upływie czasu `smsAfterGp0Ms` (domyślnie 5000 ms) wywoływana jest funkcja `communicationSendBoostAlarmSms()`, realizująca właściwą transmisję wiadomości, która została szczegółowo opisana w punktach 6.4.3 i 6.4.4. Po zakończeniu transmisji sekwencja kończy się przejściem do stanu beczynności poprzez funkcję `finishGSMSquence()`. Funkcja ta przywraca konfigurację pinów MCP23008 do stanu spoczynkowego, ustawia GP0 w stan niski (odcinając zasilanie

modemu), wznawia GPS z trybu backup oraz przywraca pełną częstotliwość taktowania mikrokontrolera.

Sekwencja alarmowa może być wyzwolona z kilku miejsc kodu, w zależności od scenariusza:

- z głównej pętli `main.cpp` - po wykryciu upadku w trybie pracy, poprzez wywołanie `userInterfaceEnterPreAlarm("FALL_CONFIRMED")`, które jako pierwsze kieruje sekwencję do stanu alarmu wstępnego;
- z funkcji obsługi przycisku odwołania alarmu - przytrzymanie przycisku (≥ 3 s) wywołuje natychmiastową sekwencję alarmową z pominięciem alarmu wstępnego (`GP5_PANIC`);
- z mechanizmu ponawiania próby - po nieudanej próbie transmisji, z opóźnieniem 15 sekund;
- z funkcji `setup()` w `main.cpp` - gdy w pliku konfiguracyjnym `dev.json` wykryty zostanie niewyzerowany licznik prób (`smsCurrentAttempt > 0`), co oznacza, że poprzednia próba SMS została przerwana przez nagłe wyłączenie urządzenia.

6.4.2. Procedura alarmu wstępnego

Procedura alarmu wstępnego stanowi mechanizm umożliwiający użytkownikowi anulowanie wykrytego zdarzenia upadku, zanim system rozpocznie właściwą sekwencję alarmową. Mechanizm ten istotnie zwiększa odporność systemu na fałszywe alarmy, dając użytkownikowi czas na świadomą decyzję o przerwaniu procedury, na przykład w sytuacji, gdy algorytm detekcji upadku zaklasyfikował jako rzeczywisty upadek zdarzenie wynikające z innej przyczyny (na przykład upadek roweru bez udziału użytkownika).

Mechanizm uruchamiany jest poprzez wywołanie funkcji `userInterfaceEnterPreAlarm()` z głównej pętli w `main.cpp`, bezpośrednio po otrzymaniu wyniku `FallDetectionResult::FALL_CONFIRMED` z funkcji `updateFallDetection()` (omówionej w podrozdziale 6.3).

Długość okna alarmu wstępnego określana jest parametrem `preAlarmWindowSec` w pliku konfiguracyjnym `/Config/dev.json`, z domyślną wartością 30 sekund i zakresem dostosowalnym w przedziale od 5 do 300 sekund. W trakcie aktywnego okna alarmu wstępnego sygnalizacja świetlna realizowana jest poprzez miganie diody LED alarmu z częstotliwością określoną parametrem `alarmLedBlinkHz`. Wzór migania stanowi jednoznaczny sygnał dla użytkownika o aktywnej procedurze i dostępnym oknie anulowania.

Anulowanie alarmu wstępnego realizowane jest poprzez krótkie naciśnięcie przycisku odwołania alarmu (czas trzymania poniżej 1 sekundy). Algorytm wykrywa zbocze opadające na pinie GP5 ekspandera MCP23008, a w przypadku, gdy całkowity czas trzymania przycisku nie przekroczył progu długiego naciśnięcia, sekwencja przechodzi natychmiast do stanu bezczynności bez wysłania wiadomości SMS. Stan diody alarmu jest gaszony, a sesja diagnostyczna w `sms_debug.log` kończona z oznaczeniem anulowania.

W przypadku braku reakcji użytkownika w trakcie okna alarmu wstępnego, po upływie zaprogramowanego czasu wywoływana jest funkcja `userInterfaceTriggerAlarm()` z powodem `PRE_ALARM_TIMEOUT`, która z kolei wywołuje `_startAlarmSequence()` - uruchamiając pełną sekwencję alarmową opisaną w punkcie 5.4.1. Świadoma decyzja użytkownika o niedopuszczeniu do anulowania (np. pozostawanie nieruchomym po faktycznym upadku) prowadzi naturalnie do wysłania wiadomości alarmowej.

6.4.3. Sekwencja komend AT

Komunikacja z modemem GSM SIM800L realizowana jest poprzez standard komend AT (znanych również jako *Hayes command set*), wymienianych z modemem przez interfejs UART. Moduł `communication.cpp` realizuje pełną sekwencję komend wymaganą do wysłania wiadomości SMS w trybie tekstowym od testu łączności poprzez konfigurację parametrów, rejestrację w sieci komórkowej, aż do właściwej transmisji wiadomości.

Sekwencja podzielona jest na dwie fazy. Faza wczesnego budzenia realizowana jest w trakcie stanu transmisji SMS po upływie około 2500 ms od włączenia zasilania modemu - przed właściwą procedurą wysyłki. Jej celem jest wstępna inicjalizacja portu szeregowego oraz wprowadzenie modemu w tryb wstrzymanej rejestracji sieci poprzez komendę AT+CFUN=0. Mechanizm ten ogranicza chwilowy pobór prądu modemu w pierwszych sekundach po starcie zasilania. Bezpośrednia rejestracja w sieci komórkowej w tym okresie może powodować impulsowy pobór prądu rzędu kilkuset miliamperów, co przy ograniczonej wydajności przetwornicy TPS63020 prowadziłoby do spadku napięcia zasilania i niestabilności pracy układu (zjawisko *brown-out*). Wprowadzenie modemu w stan CFUN=0 na czas wstępnej stabilizacji eliminuje to ryzyko.

Faza właściwej transmisji rozpoczyna się po upływie pełnego okresu smsAfterGp0Ms i obejmuje sekwencję kilkunastu komend AT realizujących konfigurację oraz wysyłkę wiadomości. Pełne zestawienie komend wraz z ich rolami funkcjonalnymi oraz domyślnymi czasami oczekiwania na odpowiedź przedstawiono w Tabeli 11.

Tabela 11. Sekwencja komend AT w procedurze wysyłania wiadomości SMS.

Lp.	Komenda	Klasa	Rola funkcjonalna	Opóźnienie [s]
1.	AT	Miękka	Test poprawności łącza UART z modemem. Oczekiwana odpowiedź OK, potwierdzająca gotowość modemu do akceptacji kolejnych komend.	3
2.	AT+CFUN=0	Miękka	Wprowadzenie modemu w tryb wstrzymanej rejestracji sieci, wykonywane w fazie wczesnego budzenia w celu redukcji poboru prądu.	8
3.	AT+CFUN=1	Krytyczna	Przywrócenie pełnej funkcjonalności modemu przed właściwą transmisją SMS.	15
4.	ATE0	Miękka	Wyłączenie echa komend zwracanego przez mod.em, upraszczające późniejsze parsowanie odpowiedzi	1,5
5.	AT+CMEE=2	Miękka	Włączenie rozszerzonych kodów błędów w formacie tekstowym dla ułatwienia diagnostyki problemów komunikacji.	1,5
6.	AT+CMGF=1	Krytyczna	Ustawienie trybu tekstowego SMS (alternatywą jest tryb PDU); umożliwia bezpośrednie wysłanie treści wiadomości jako tekstu, bez konieczności kodowania binarnego.	1,5

7.	AT+CSCS="GSM"	Miękka	Wybór zestawu znaków GSM 7-bit dla wiadomości, odpowiedniego dla wiadomości w alfabecie łacińskim.	2
8.	AT+CSMP=17,255,0,0	Miękka	Konfiguracja parametrów PDU wiadomości tekstowej (typ, okres ważności, format identyfikatora protokołu, format kodowania danych).	2
9.	AT+CREG?	Krytyczna	Zapytanie o status rejestracji w sieci komórkowej. Algorytm cyklicznie powtarza zapytanie do momentu otrzymania stanu zarejestrowanego.	45
10.	AT+CGSMS=1	Miękka	Wybór kanału przesyłania SMS przez sieć komutowaną (<i>Circuit Switched</i>), z pominięciem GPRS.	2
11.	AT+CMGS="numer "	Krytyczna	Inicjacja transmisji wiadomości SMS na podany numer telefonu. Modem zwraca znak „>” oczekując na przesłanie treści wiadomości.	8
12.	treść + Ctrl+Z (0x1A)	Krytyczna	Przesłanie właściwej treści wiadomości zakończonej znakiem 0x1A (Ctrl+Z), inicjującym wysyłkę przez modem. Odpowiedź OK potwierdza pomyślne doręczenie do centrum SMS operatora.	60

Sekwencja komend nie jest realizowana w sposób ścisły "wszystko albo nic". Algorytm rozróżnia komendy **krytyczne**, których niepowodzenie powoduje przerwanie całej sekwencji i klasyfikację próby jako nieudanej, oraz komendy **miękkie**, których błędy są logowane diagnostycznie, ale nie blokują kontynuacji.

Do komend **krytycznych** zaliczono: brak odpowiedzi na pierwsze AT (oznacza brak łączności UART z modemem), niepowodzenie AT+CFUN=1 (uniemożliwiające pełną funkcjonalność modemu), niepowodzenie AT+CMGF=1 przed AT+CMGS (uniemożliwiające pracę w trybie tekstowym), brak znaku zachęty > po komendzie AT+CMGS oraz odpowiedź ERROR lub przekroczenie limitu czasu 60 sekund po przesłaniu treści wiadomości. Każde z tych zdarzeń powoduje przerwanie sekwencji z wynikiem negatywnym i uruchomienie mechanizmu ponowionego wysyłania opisanego w punkcie 6.4.5.

Do komend **miękkich** zaliczono: przekroczenie limitu czasu AT+CREG? (sekwencja kontynuowana nawet bez potwierdzenia rejestracji w sieci), niepowodzenie AT+CFUN=0 w fazie wczesnego budzenia (brak wpływu na właściwą transmisję), brak odebrania komunikatu SMS *Ready*

po AT+CFUN=1 (modem często gotowy do pracy mimo braku tego potwierdzenia) oraz niepowodzenia komend konfiguracyjnych typu ATE0 lub AT+CSCA?. Ta dwustopniowa strategia obsługi błędów istotnie zwiększa odporność systemu na nietypowe warunki sieciowe oraz pozwala na wysłanie wiadomości nawet przy częściowo niesprawnym łączu radiowym.

6.4.4. Treść wiadomości sms

Wiadomość SMS generowana przez procedurę alarmową ma jednolitą, zwartą strukturę, dostosowaną do ograniczeń długości pojedynczego segmentu SMS w standardzie GSM 7-bit (160 znaków). Treść wiadomości składana jest dynamicznie w funkcji `communicationSendBoostAlarmSms()` z trzech elementów: stałego nagłówka identyfikującego źródło alarmu, znacznika czasowego oraz informacji o pozycji geograficznej urządzenia.

Przykładowa wiadomość alarmowa z dostępnym sygnałem GPS przyjmuje postać:

!!ALARM CSR!! 14:32:05 <https://maps.google.com/?q=52.229676,21.012229>

Nagłówek !!ALARM CSR!! (od Czarna Skrzynka Rowerowa) pełni funkcję jednoznacznego identyfikatora pozwalającego odbiorcy szybko rozpoznać źródło i charakter wiadomości. Znacznik czasowy w formacie HH:MM:SS generowany jest na podstawie czasu UTC z odbiornika GPS (z poprawką do strefy czasowej Europy Środkowej) lub w przypadku braku synchronizacji satelitarnej wewnętrznego zegara czasu rzeczywistego mikrokontrolera. Współrzędne geograficzne formatowane są w postaci dziesiętnej (*decimal degrees*) z dokładnością do sześciu miejsc po przecinku, co odpowiada rozdzielczości około 0,11 metra na równiku, wystarczającej dla precyzyjnej lokalizacji miejsca zdarzenia.

Link do serwisu Google Maps pozwala odbiorcy wiadomości na natychmiastowe otwarcie pozycji w przeglądarce internetowej lub aplikacji mobilnej, bez konieczności ręcznego kopiowania współrzędnych. Wybrano serwis Google Maps ze względu na jego powszechną dostępność i standardową obsługę formatu linkowego z parametrem `q=` na większości urządzeń mobilnych.

Wybór danych nawigacyjnych przekazywanych w wiadomości realizowany jest według trójstopniowej hierarchii priorytetów. W pierwszej kolejności wykorzystywany jest aktualny pomiar pozycji z odbiornika GPS (`sLastSensor.hasFix`), jeśli urządzenie posiada bieżące ustalenie pozycji. W przypadku jego braku w trybie pracy weryfikowana jest dostępność ostatniego znanego fixu sprzed maksymalnie 10 minut (`sLastGoodFix`), pozwalający na wysłanie wiarygodnej informacji o pozycji nawet po krótkotrwałej utracie sygnału satelitarnego (na przykład w tunelu lub pod wiaduktem).

Maksymalna długość wiadomości ograniczona jest rozmiarem bufora body (160) w module `communication.cpp`, odpowiadającym standardowemu limitowi pojedynczego segmentu SMS w alfabecie GSM 7-bit. Funkcja `sprintf()` realizująca składanie wiadomości automatycznie obcina zawartość do 159 znaków + znak zakończenia łańcucha, eliminując ryzyko przepełnienia bufora. W praktyce treść wiadomości alarmowej nie zbliża się do tego limitu. Typowa pełna wiadomość z linkiem do Google Maps zajmuje około 80-85 znaków.

6.4.5. Mechanizm ponawiania prób i odporność na awarie

Niezawodność dostarczenia wiadomości alarmowej stanowi kluczową cechę funkcjonalną urządzenia. Aby zapewnić wysokie prawdopodobieństwo doręczenia komunikatu nawet w niekorzystnych warunkach sieciowych, w systemie zaimplementowano wieloetapowy mechanizm ponawiania prób transmisji z odpornością na nagłe awarie zasilania urządzenia.

Mechanizm ponawiania prób (*retry*) sterowany jest parametrem `smsRetryCount` w pliku konfiguracyjnym `/Config/dev.json`, określającym dopuszczalną liczbę ponownych prób po niepowodzeniu pierwszej transmisji, z konfigurowalnym zakresem od 0 do 3 prób.

Po niepowodzeniu transmisji wprowadzany jest czas odnowienia (*cooldown*) o długości 15 sekund (`kSmsRetryDelayMs`), w trakcie, którego sekwencja powraca do stanu bezczynności i

wstrzymuje kolejną próbę. Wprowadzenie odstępu czasowego pozwala na samoczynne odtworzenie się warunków sieciowych. W wielu przypadkach niepowodzenie pierwszej próby wynika z chwilowego pogorszenia jakości sygnału radiowego lub problemu rejestracji w sieci, ustępujących w krótkim czasie. Po upływie czasu odnowienia (*cooldownu*) uruchamiana jest pełna sekwencja alarmowa z powodem `AUTO_RETRY_N`, gdzie *N* oznacza numer kolejnej próby.

Szczególnie istotnym mechanizmem odporności jest persystencja licznika prób w pamięci nieulotnej. Pole `smsCurrentAttempt` w pliku `/Config/dev.json` zwiększane jest na początku każdej sekwencji alarmowej i zerowane dopiero po pomyślnym doręczeniu wiadomości lub po wyczerpaniu wszystkich dopuszczalnych prób. Mechanizm ten zapewnia kontynuację procedury alarmowej w sytuacji, gdy urządzenie zostanie nagle wyłączone w trakcie wysyłki SMS (na przykład wskutek rozładowania ogniwa, utraty styku z baterią po silnym uderzeniu czy awarii sprzętowej modemu).

Po ponownym uruchomieniu urządzenia funkcja `setup()` w `main.cpp` sprawdza wartość `smsCurrentAttempt` i jeśli jest niezerowa automatycznie planuje wznowienie sekwencji alarmowej z opóźnieniem 20 sekund, z powodem `AFTER_HARD_RESET`. Opóźnienie pozwala na stabilizację stanu zasilania oraz inicjalizację wszystkich modułów peryferyjnych przed ponowną próbą transmisji.

6.4.6. Interakcja z innymi modułami

Sekwencja alarmowa obejmuje znacząco wyższy pobór prądu (rzędu setek miliamperów w czasie aktywnej transmisji modemu) oraz wymaga zachowania ciągłości komunikacji z modemem GSM. Z tego względu w czasie aktywnej sekwencji zawieszane są lub modyfikowane kluczowe operacje pozostałych modułów systemu, zwiększając niezawodność procesu wysyłki i zmniejszając ogólny pobór prądu z ogniwa.

W przypadku gdy w momencie wyzwolenia alarmu aktywny był interfejs serwisowy WWW, funkcja `_startAlarmSequence()` realizuje jawne zatrzymanie serwera poprzez wywołanie `stopWebServer()` przed kontynuacją sekwencji. Mechanizm ten zwalnia zasoby pamięci wykorzystywane przez stos WiFi oraz eliminuje ryzyko zakłóceń pracy modemu GSM przez aktywny moduł radiowy 2,4 GHz.

W trakcie aktywnej sekwencji alarmowej w głównej pętli `loop()` ustawiana jest flaga `skipHeavySd`, której efektem jest pominięcie operacji odczytu czujników (`sensorsRead()`), zapisu wpisów do logu jazdy (`storage.save()`) oraz aktualizacji wysokoczęstotliwościowego rejestratora `ShockCapture::update()`. Pauza zapisu na karcie microSD eliminuje potencjalne konflikty zasobów oraz ogranicza pobór prądu związany z operacjami SPI.

Odbiornik GPS Air530Z przełączany jest w tryb backupu (*PMTK backup mode*) o poborze prądu rzędu 10 μ A, przy zachowaniu zapamiętanej pozycji ostatniego fixu w pamięci wewnętrznej odbiornika. Przejście do trybu backupu realizowane jest funkcją `sensorsPauseForSms()` na początku sekwencji, a powrót do pełnej funkcjonalności funkcją `sensorsResumeFromSms()` po zakończeniu. Ze względu na trwałą pamięć ostatniej pozycji, ponowne ustalenie fixu po powrocie z backupu trwa zazwyczaj około 3 sekund (*hot start*).

Częstotliwość taktowania procesora ESP32-C6 obniżana jest z domyślnych 160 MHz do 40 MHz na czas aktywnej sekwencji, co istotnie redukuje pobór prądu rdzenia procesora przy zachowaniu wystarczającej wydajności obliczeniowej do realizacji komunikacji UART z modemem. Po zakończeniu sekwencji częstotliwość przywracana jest do wartości domyślnej.

6.4.7. Fragment implementacji

Listing 6. Fragment obsługi stanu `HoldGp0` w maszynie stanów `GSMSeq` prezentuje kluczowy fragment funkcji `userInterfacePoll()` z modułu `userInterface.cpp`, ukazujący logikę obsługi stanu transmisji SMS w maszynie stanów. Fragment ten ilustruje synchronizację mechanizmu wczesnego budzenia modemu z właściwą procedurą wysyłki oraz obsługę zakończenia sekwencji.

Listing 6. Fragment obsługi stanu HoldGp0 w maszynie stanów GSMSeq

```
// userInterface.cpp - userInterfacePoll(), stan GSMSeq::HoldGp0
// --- Wejście w HoldGp0 ---
sGSM = GSMSeq::HoldGp0;
enterGp0Hold(nowMs, gp2Desired);    // GP0=HIGH-zasilanie modemu GSM
sSmsPending      = true;
sSmsDueMs        = nowMs + currentSmsAfterGp0Ms();
sGp0AssertMs     = nowMs;
sHoldEndMs       = nowMs + kGp0HoldMs;

//Faza oczekiwania: early-wake modemu przed pełną transmisją SMS
if (sSmsPending && nowMs >= sNextGp0TickMs && nowMs < sSmsDueMs) {
    const uint32_t elapsed = nowMs - sGp0AssertMs;
    if (!sEarlyGsmWakeDone) {
        sEarlyGsmWakeDone =
communicationGsmEarlyWakeDuringHold(elapsed);
    }
    sNextGp0TickMs += kGp0HoldTickMs;
}

// --- Właściwa transmisja SMS (gdy minie smsAfterGp0Ms) ---
if (sSmsPending && nowMs >= sSmsDueMs) {
    sSmsPending = false;

    const EmergencyConfig ec = storage.getEmergencyConfig();
    const SensorData* smsData = // bieżący fix GPS lub ostatni
świeży (<10 min) */;

    const bool smsOk = communicationSendBoostAlarmSms(
        ec.phone, elapsedMs, smsData, isWorkMode);

    // Skrócenie czasu trzymania GP0=HIGH po zakończeniu UART
    sHoldEndMs = min(sHoldEndMs, millis() + kPostSmsHoldMs);

    // Obsługa mechanizmu powtarzania sekwencji
    if (smsOk) {
        storage.setSmsCurrentAttempt(0);
    } else if (attempt <= maxRetry && maxRetry > 0) {
        sRetryScheduled = true;
        sRetryDueMs = millis() + kSmsRetryDelayMs + kPostSmsHoldMs;
    } else {
        storage.setSmsCurrentAttempt(0);
    }
}

//Domknięcie sekwencji: zwolnienie GP0, wyłączenie LED
if (nowMs >= sHoldEndMs) {
    sSmsPending = false;
    finishGp5Sequence(gp2Desired);    // GP0=LOW, exitSmsLowPower()
    applyOutputLevels(gp2Desired, false);
    sSGM = SGMSeq::Idle;
}
```

Listing 7 ukazuje fragment funkcji gsmTxSms() z modułu communication.cpp, prezentujący strukturę pętli oczekiwania na rejestrację w sieci komórkowej oraz właściwą transmisję wiadomości tekstowej.

Listing 7. Fragment implementacji transmisji SMS w module communication.cpp

```
bool gsmTxSms(const char* phone, const char* body) {
    // Test łącza i konfiguracja modemu
    if (!gsmSendAt("AT", "OK", 3000)) return false;

    if (sGsmHoldMinimized) {
        if (!gsmSendAt("AT+CFUN=1", "OK", 15000)) return false;
        gsmWaitForUrc("SMS Ready", 15000);
    }

    gsmSendAt("ATE0", "OK", 1500);
    gsmSendAt("AT+CMEE=2", "OK", 1500);
    gsmSendAt("AT+CMGF=1", "OK", 1500);
    gsmSendAt("AT+CSCS=\"GSM\"", "OK", 2000);
    gsmSendAt("AT+CSMP=17,255,0,0", "OK", 2000);

    // Pętla oczekiwania na rejestrację w sieci (do 45 s)
    const uint32_t cregStartMs = millis();
    bool registered = false;
    while (millis() - cregStartMs < 45000) {
        if (gsmSendAt("AT+CREG?", "+CREG: 0,1", 2000) ||
            gsmSendAt("AT+CREG?", "+CREG: 0,5", 2000)) {
            registered = true;
            break;
        }
        delay(1500);
    }
    if (!registered) {
        smsDebugStage("CREG_TIMEOUT");
        // Kontynuacja mimo braku CREG

        gsmSendAt("AT+CGSMS=1", "OK", 2000);
        gsmSendAt("AT+CMGF=1", "OK", 1500);

        // Inicjacja transmisji - oczekiwanie na prompt
        char cmgs[64];
        snprintf(cmgs, sizeof(cmgs), "AT+CMGS=\"%s\"", phone);
        if (!gsmSendAt(cmgs, ">", 8000)) return false;

        // Wysłanie treści + Ctrl+Z (0x1A) - oczekiwanie do 60 s
        GsmSerial.print(body);
        GsmSerial.write(0x1A);

        return gsmWaitForResponse("OK", 60000);
    }
}
```

6.5. Rejestracja danych na karcie microSD

Niniejszy podrozdział omawia warstwę trwałego zapisu danych pomiarowych i diagnostycznych na karcie pamięci microSD. Przedstawiono organizację plików na nośniku, format logu jazdy,

mechanizmy zapewniające odporność na utratę zasilania, strukturę plików pomocniczych generowanych po zakończeniu sesji oraz dedykowany rejestrator wstrząsów działający z wysoką częstotliwością próbkowania. Podrozdział zamykają informacje o obsłudze błędów wejścia-wyjścia oraz integracji z interfejsem WWW.

6.5.1. Organizacja danych na karcie

Karta microSD stanowi jednocześnie magazyn danych pomiarowych oraz nośnik konfiguracji urządzenia. Struktura katalogowa nie jest tworzona w jednym, dedykowanym kroku inicjalizacyjnym - poszczególne katalogi powstają, w momencie pierwszego zapisu pliku danego typu, co upraszcza logikę inicjalizacji i pozwala na łagodną tolerancję uszkodzonych lub niekompletnych struktur na nośniku.

Główne katalogi systemu zorganizowano w trzy logiczne grupy. Katalog /Logs/ przechowuje pliki sesji jazdy w hierarchii czasowej /Logs/RRRR/MM/DD/, co umożliwia naturalną archiwizację długoterminową i upraszcza eksport zbiorów danych do narzędzi analitycznych. Katalog /ShockData/ przechowuje pliki rejestratora wstrząsów w strukturze. Katalog /Config/ przechowuje pliki konfiguracyjne urządzenia w formacie JSON, wczytywane podczas rozruchu urządzenia.

Cykl życia sesji jazdy rozpoczyna się automatycznie po jednoczesnym spełnieniu trzech warunków: aktywacji funkcji logowania w pliku konfiguracyjnym `dev.json`, uzyskaniu fixu satelitarnego oraz wykryciu ruchu pojazdu przez algorytm dynamiki ruchu. Identyfikator sesji `sessionId` generowany jest jako wartość licznika czasu systemowego (`millis()`) w momencie utworzenia sesji, rozwiązanie zapewniające unikatowość w obrębie pojedynczego cyklu pracy urządzenia bez konieczności utrzymywania licznika sesji. Zakończenie sesji następuje automatycznie po upływie skonfigurowanego czasu bezruchu. Opóźnienie zapobiegające przedwczesnemu kończeniu sesji podczas krótkich postojów na sygnalizacji świetlnej lub przerw w trasie.

6.5.2. Format i częstotliwość zapisu danych z jazdy

Główny rejestr danych pomiarowych przechowywany jest w plikach o nazwach generowanych według schematu `RIDE_RRRRMMDD_HHMMSS_<sessionId>.csv`, umożliwiającego identyfikację sesji bezpośrednio z poziomu nazwy pliku. Format CSV zastosowano ze względu na powszechną interoperacyjność z narzędziami analitycznymi oraz prostotę parsowania - format wspierany jest natywnie przez wszystkie popularne arkusze kalkulacyjne oraz biblioteki języków programowania ogólnego przeznaczenia.

Każdy zapisany rekord zawiera 28 pól pogrupowanych w sześć kategorii tematycznych, omówionych na poziomie wymagań w podrozdziale 3.1.1. Separatorem pól jest znak przecinka, a pierwsza linia pliku zawiera nagłówek z nazwami kolumn, zapewniający samoopisywalność pliku. Znaczniki czasowe zapisywane są w formacie ISO 8601 (np. 2026-06-02T14:32:05) oraz dodatkowo jako liczba sekund od momentu uruchomienia urządzenia (*uptime*), umożliwiając niezależną korelację zdarzeń bez konieczności posiadania bieżącego czasu rzeczywistego. Współrzędne geograficzne zapisywane są jako liczby zmiennoprzecinkowe z dokładnością do sześciu miejsc dziesiętnych. W przypadku braku fixu satelitarnego, domyślną decyzją projektową jest pominięcie zapisu wiersza - mechanizm zapobiegający rejestrowaniu danych o niskiej wartości analitycznej.

Częstotliwość zapisu realizowana jest dwupoziomowo. Na poziomie pętli głównej obowiązuje adaptacyjna bramka czasowa, której interwał zależy od bieżącego stanu ruchu pojazdu (Tabela 5.6). Wewnątrz funkcji `Storage::save()` znajduje się dodatkowa, niezależna bramka czasowa narzucająca twardy limit maksymalnej częstotliwości zapisu wynoszący 1 Hz.

Tabela 12. Adaptacyjne interwały zapisu logu jazdy.

Stan ruchu	Warunek	Interwał bramki [ms]	Częstotliwość [Hz]
Postój/brak fixu GPS	<code>!isMoving</code> lub <code>!hasFix</code>	5000	0,2

Wolna jazda	Prędkość <15 km/h	2000	0,5
Szybka jazda	Prędkość > 15 km/h	1000	1

Zagnieżdżenie obu mechanizmów stanowi przykład wzorca opisanego w podrozdziale 5.1.3 - bramka wewnętrzna chroni warstwę systemu plików przed nadmiernym obciążeniem, niezależnie od logiki wywołań warstwy nadrzędnej.

6.5.3. Mechanizm trwałości zapisu

Plik logu jazdy pozostaje otwarty przez cały czas trwania sesji - jednokrotnie otwierany w funkcji `startNewRide()` i zamykany w `endRide()`. Decyzja ta eliminuje narzut operacji systemu plików (otwarcie i zamknięcie pliku, aktualizacja tabeli alokacji FAT) towarzyszący każdemu zapisowi pojedynczego wiersza. Przy częstotliwości zapisu 1 Hz operacje na strukturach systemu plików stanowiłyby znaczące obciążenie magistrali SPI i zwiększałyby ryzyko niespójności tabeli alokacji w przypadku nagłego odcięcia zasilania.

Bezpieczeństwo danych w obliczu awarii zasilania zapewnia mechanizm okresowej synchronizacji bufora systemu plików z fizycznym nośnikiem. Synchronizacja wymuszana jest co dziesięć zapisanych wierszy poprzez wywołanie funkcji `_logFile.sync()`, co w trybie szybkiej rejestracji odpowiada interwałowi około dziesięciu sekund. Wartość tę dobrano jako kompromis między dwiema przeciwstawnymi tendencjami: zbyt rzadka synchronizacja zwiększa ilość potencjalnie utraconych danych przy awarii, podczas gdy zbyt częsta - obciąża magistralę SPI i skraca żywotność komórek pamięci flash karty SD.

Niezależnie od cyklicznego mechanizmu synchronizacji, wymuszony flush bufora wykonywany jest w czterech sytuacjach krytycznych.

- **Pierwszą** jest formalne zakończenie sesji jazdy przez funkcję `endRide()` bufor jest synchronizowany przed zamknięciem pliku, gwarantując kompletność zapisu;
- **Drugą** jest przejście urządzenia w tryb głębokiego uśpienia - przed wywołaniem `esp_deep_sleep_start()` wykonywany jest globalny flush przez funkcję `storage.flushBuffer()`, zapobiegający utracie danych zarejestrowanych po ostatniej cyklicznej synchronizacji;
- **Trzecią** jest zapis dowolnego pliku konfiguracyjnego - operacja konfiguracji przerywa tymczasowo zapis logu, wymuszając pełną synchronizację;
- **Czwartą** jest rozpoczęcie nowej sesji jazdy bez formalnego zakończenia poprzedniej (sytuacja awaryjna), kiedy bufor poprzedniej sesji jest synchronizowany przed otwarciem nowego pliku.

Format CSV jest naturalnie odporny na nagłe odcięcie zasilania. W najgorszym przypadku - awaria zasilania pomiędzy dwoma synchronizacjami - utracone zostaje do dziesięciu ostatnich wierszy, a wcześniej zsynchronizowana zawartość pliku pozostaje w pełni czytelna. W szczególnie niekorzystnym scenariuszu - awarii zasilania w trakcie zapisu pojedynczego wiersza - utracony zostaje niepełny ostatni wiersz, jednak struktura pliku CSV pozostaje poprawna od pierwszego do przedostatniego rekordu. Mechanizm ten kontrastuje z formatami binarnymi (np. SQLite), które przy awarii w trakcie transakcji wymagają mechanizmów dziennika transakcji (*write-ahead log*) lub procedury naprawczej przy następnym otwarciu.

6.5.4. Metadane jazdy i eksport trasy w formacie GPX

Po formalnym zakończeniu sesji jazdy funkcja `endRide()` generuje dwa pliki uzupełniające w stosunku do głównego pliku CSV, współdzielące bazową nazwę z plikiem nadrzędnym oraz wspólny identyfikator sesji. Mechanizm ten upraszcza zarządzanie zbiorami danych - usunięcie pojedynczej jazdy sprowadza się do skasowania trzech plików o tej samej nazwie, niezależnie zarządzanych przez system plików karty pamięci.

Pierwszym z plików uzupełniających jest plik metadanych w formacie JSON o rozszerzeniu meta.json. Plik ten zawiera podsumowanie statystyczne sesji: identyfikator, znaczniki czasowe rozpoczęcia i zakończenia, czas trwania, liczbę zarejestrowanych próbek, liczbę próbek z fixem satelitarnym, przebyty dystans, prędkość średnią i maksymalną, maksymalne przyspieszenie wykryte podczas jazdy, liczbę wykrytych zdarzeń typu uderzenie oraz upadek, zakres temperatur i ciśnień atmosferycznych oraz średnie napięcie ogniwa zasilającego. Strukturyzowany format metadanych umożliwia szybkie wyświetlenie listy zarejestrowanych jazd w interfejsie konfiguracyjnym, bez konieczności parsowania pełnych plików CSV.

Drugim plikiem uzupełniającym jest plik trasy w standardzie GPX 1.1, generowany przez konwersję logu CSV - ponowny odczyt pliku po jego zamknięciu i wybiórczy zapis wierszy z fixem satelitarnym w strukturze XML. Każdy punkt trasy (<trkpt>) zawiera współrzędne geograficzne, wysokość nad poziomem morza, znacznik czasowy w formacie UTC, liczbę widzianych satelitów oraz prędkość chwilową umieszczoną w sekcji rozszerzeń. Format GPX zastosowano ze względu na powszechne wsparcie w popularnych platformach treningowych oraz aplikacjach mapowych, co umożliwia bezpośredni import zarejestrowanych tras bez konieczności konwersji formatów. Pełne dane inercyjne, środowiskowe oraz diagnostyczne pozostają w pliku CSV - eksport GPX jest świadomie ograniczony do trasy geograficznej.

W przypadku nieprawidłowego zakończenia sesji jazdy - na przykład w wyniku nagłej awarii zasilania lub błędu krytycznego - plik CSV może istnieć samodzielnie, bez plików uzupełniających. Mechanizm naprawczy zaimplementowany w warstwie serwera WWW umożliwia odtworzenie metadanych z istniejącego pliku CSV podczas listowania jazd w interfejsie konfiguracyjnym, zapewniając kompletność prezentacji danych historycznych nawet w obliczu nieoczekiwanych zdarzeń systemowych.

6.5.5. Rejestrator wstrząsów

Moduł shockcapture.cpp stanowi dedykowany rejestrator wysokoczęstotliwościowy, uzupełniający główny log jazdy o szczegółowy zapis przebiegu zdarzeń wstrząsowych. Próbkowanie modułu inercyjnego realizowane jest z częstotliwością 200 Hz (interwał próbkowania 5 ms) przez bezpośrednie odczyty rejestrów IMU wewnątrz funkcji `_processSample()`, niezależnie od częstotliwości pętli głównej. Mechanizm ten zapewnia stabilną, deterministyczną częstotliwość zapisu nawet w sytuacjach, gdy pętla główna chwilowo wykonuje operacje czasochłonne.

Decyzja o uruchomieniu zapisu do pliku realizowana jest przez mechanizm automatycznego wyzwalania, oparty o dwa niezależne progi rozstrzygające: progową wartość chwilowej magnitudy przyspieszenia oraz progową wartość gradientu zmian przyspieszenia. Przekroczenie któregokolwiek z progów inicjuje sesję zapisu, z mechanizmem czasu odnowienia ograniczającym zbyt częste wyzwalanie kolejnych sesji (domyślnie 5 sekund pomiędzy kolejnymi zdarzeniami). Parametry te są konfigurowalne przez plik `/Config/shock.json`, umożliwiając dostrojenie czułości algorytmu w oparciu o wyniki testów terenowych.

Plik zapisu generowany jest według schematu `SHOCK_RRRRMMDD_HHMMSS_<sessionId>.csv` w katalogu `/ShockData/`. Pojedynczy rekord zawiera dwanaście pól: znacznik czasowy w milisekundach od startu sesji, datę i czas lokalny, przyspieszenia w trzech osiach, magnitudę wektora przyspieszenia, gradient zmian (jerk), opcjonalny znacznik zdarzenia (etykieta klasyfikacyjna) oraz kontekst geolokalizacyjny obejmujący współrzędne geograficzne i liczbę widzianych satelitów. Powiązanie zdarzeń ShockCapture z głównym logiem jazdy realizowane jest przez czasową korelację, znaczniki zdarzeń w pliku ShockCapture umożliwiają retrospektywne zestawienie obu źródeł.

Mechanizm sterowania trwałością zapisu w module shockcapture.cpp zoptymalizowano pod większą częstotliwość strumienia danych - wymuszony flush bufora wykonywany jest co 32 zapisane wiersze, co przy częstotliwości 200 Hz odpowiada interwałowi około 160 milisekund. Zwiększona częstotliwość synchronizacji w stosunku do logu jazdy (co 10 wierszy $\approx$ 10 sekund) jest uzasadniona wyższą wartością informacyjną poszczególnych próbek - utrata segmentu pików akceleracji jest bardziej niekorzystna analitycznie niż utrata kilku sekund logu nawigacyjnego.

6.5.6. Obsługa błędów oraz integracja z interfejsem WWW

Brak dostępności karty microSD przy uruchomieniu urządzenia - spowodowany fizycznym brakiem karty, jej uszkodzeniem lub błędem inicjalizacji magistrali SPI - sygnalizowany jest dedykowanym wzorcem migania diody LED. System pozostaje funkcjonalny w trybie ograniczonym: moduły akwizycji danych, detekcji upadku oraz komunikacji GSM kontynuują pracę, jednak bez utrwalania zarejestrowanych pomiarów. Decyzja o przyjęciu modelu graceful degradation podyktowana jest priorytetem funkcji alarmowej - zachowanie zdolności do wysłania powiadomienia o wypadku jest istotniejsze niż zapis danych historycznych.

Błędy zapisu pojedynczych rekordów (przepełnienie karty, błąd I/O magistrali SPI, fizyczne wyjęcie karty w trakcie pracy) skutkują zwróceniem statusu błędu przez funkcję `Storage::save()`, bez mechanizmu automatycznego ponawiania ani buforowania w pamięci RAM. Decyzja ta wynika z charakterystyki błędów `storage.cpp` - większość przypadków ma charakter trwały (uszkodzona karta, przepełnienie) i ponawianie operacji nie przynosi poprawy, natomiast skomplikowana logika rezerwowa (*fallback*) obciążałaby system w warunkach normalnej pracy.

Integracja z interfejsem konfiguracyjnym WWW obejmuje cztery główne grupy funkcjonalności. Pierwsza to listowanie zarejestrowanych jazd z wyświetleniem metadanych każdej sesji, drugą - pobieranie surowych plików CSV oraz wygenerowanych plików GPX bezpośrednio z karty, trzecią - pobieranie listy zarejestrowanych zdarzeń `shockcapture.cpp` wraz z ich metadanymi, czwartą - selektywne usuwanie historycznych sesji.

6.6. Serwer WWW

Niniejszy podrozdział omawia warstwę interfejsu konfiguracyjnego urządzenia, realizowanego jako lokalny serwer HTTP hostowany bezpośrednio przez mikrokontroler ESP32-C6. Przedstawiono architekturę serwera, sposób aktywacji w trybie serwisowym, strukturę aplikacji webowej, interfejs programistyczny API, mechanizmy optymalizacji oraz integrację z pozostałymi modułami systemu. Wymagania funkcjonalne interfejsu sformułowano na poziomie koncepcyjnym w podrozdziale 3.1.4.

6.6.1. Architektura serwera i tryb dostępu Wi-Fi

Warstwa serwera HTTP zaimplementowana została jako moduł `WebServer`, zbudowana bezpośrednio na podstawowym obiekcie `WiFiServer` z frameworka `Arduino-ESP32`. Świadomie zrezygnowano z gotowych bibliotek wyższego poziomu, takich jak `WebServer` z `Arduino-ESP32` czy `ESPAsyncWebServer`, na rzecz implementacji zapewniającej pełną kontrolę nad cyklem życia żądania, parsowaniem nagłówek oraz mechanizmem routingu. Decyzja ta podyktowana była dwoma względami: minimalizacją zużycia pamięci RAM (gotowe biblioteki rezerwują znaczne bufora parsera niezależnie od liczby aktywnych klientów) oraz potrzebą deterministycznego zarządzania czasem trwania pojedynczego żądania w obliczu jednowątkowej architektury systemu.

Serwer operuje w modelu synchronicznym i blokującym - w danym momencie obsługiwane jest dokładnie jedno żądanie, którego pełna obsługa nie może przekroczyć skonfigurowanego limitu czasu wynoszącego pięć sekund (`WEB_REQUEST_TIMEOUT_MS`). Komunikacja realizowana jest wyłącznie w protokole HTTP/1.1, bez obsługi szyfrowania transportowego (HTTPS) oraz bez wsparcia dla `WebSocket`. Limit ten jest świadomym kompromisem, pełna obsługa szyfrowania wymagałaby dodatkowych kilkudziesięciu kilobajtów pamięci flash oraz znaczącego wzrostu narzutu obliczeniowego, podczas gdy warstwa fizyczna sieci AP zapewnia ograniczenie geograficzne dostępu wystarczające dla scenariusza serwisowego.

Tryb pracy radiowej zrealizowano w architekturze punktu dostępu (*Access Point*), w której urządzenie samodzielnie tworzy lokalną sieć Wi-Fi, do której łączy się urządzenie użytkownika z przeglądarką internetową. Konfiguracja sieci obejmuje stały identyfikator SSID, ograniczenie maksymalnej liczby jednoczesnych klientów oraz zabezpieczenie hasłem w standardzie WPA2. Hasło sieci zdefiniowane jest w pliku konfiguracyjnym, z mechanizmem *fallback* do wartości wkompiłowanej

w firmware. Adres IP urządzenia w utworzonej sieci to standardowy adres 192.168.4.1, pod którym dostępny jest interfejs webowy.

6.6.2. Cykl życia serwera i aktywacja w trybie serwisowym

Serwer WWW nie jest uruchamiany domyślnie podczas startu urządzenia. Aktywacja serwera następuje wyłącznie po wyzwoleniu trybu serwisowego przyciskiem uruchomienia WiFi AP (GP4 ekspandera portów MCP23008). Decyzja o świadomej aktywacji manualnej podyktowana jest dwoma względami: bezpieczeństwa (brak nieautoryzowanego dostępu do interfejsu konfiguracyjnego podczas normalnej eksploatacji) oraz energetycznym (radio Wi-Fi w trybie AP generuje pobór prądu rzędu kilkudziesięciu miliamperów, niedopuszczalny w trybie ciągłej rejestracji jazdy).

Sekwencja uruchomienia serwera obejmuje wizualne potwierdzenie operacji przez pięciokrotne mignięcie diody serwisowej (półokres 130 milisekund), inicjalizację radiowej warstwy AP, utworzenie gniazda nasłuchującego TCP oraz aktywację licznika bezczynności. Wszystkie te operacje wykonywane są synchronicznie w obrębie pojedynczego wywołania funkcji `startWebServer()` w module interfejsu użytkownika.

Mechanizm automatycznego wyłączenia serwera realizowany jest przez monitorowanie aktywności klientów - każde pomyślnie obsłużone żądanie HTTP wywołuje funkcję `userInterfaceNotifyWebActivity()`, resetującą licznik bezczynności. Po upływie skonfigurowanego czasu bez aktywności (domyślnie 120 sekund, wartość `webIdleTimeoutSec` w pliku `dev.json`) serwer jest automatycznie wyłączany. Mechanizm ten eliminuje konieczność pamiętania o ręcznym zamknięciu trybu serwisowego po zakończeniu konfiguracji. Niezależnie od automatycznego wyłączenia, użytkownik może ręcznie zakończyć tryb serwisowy ponownym przyciśnięciem przycisku GP4, co inicjuje analogiczną sekwencję wyłączenia z potwierdzeniem wizualnym diody.

Krytycznym przypadkiem wyłączenia serwera jest inicjacja procedury alarmowej GSM - przed uruchomieniem sekwencji wysyłki SMS funkcja `_startAlarmSequence()` bezwarunkowo zatrzymuje serwer poprzez wywołanie `stopWebServer()`. Operacja ta obejmuje zamknięcie gniazda TCP, rozłączenie wszystkich połączonych klientów AP oraz wyłączenie radia Wi-Fi (`WiFi.mode(WIFI_OFF)`). Jest to krok niezbędny dla zwolnienia zasobów energetycznych przed energochłonną transmisją modemu GSM, opisaną w podrozdziale 6.4.

6.6.3. Struktura aplikacji webowej i nawigacja

Aplikacja webowa urządzenia zrealizowana została w architekturze wielostronicowej (*multi-page application*), w której każdy ekran funkcjonalny stanowi odrębny dokument HTML serwowany pod własnym adresem URL. Świadomie zrezygnowano z architektury jednostronicowej (*single-page application*) z routowaniem po stronie klienta, ze względu na ograniczenia pamięciowe mikrokontrolera oraz prostotę utrzymania kodu. Każda strona stanowi samowystarczalny dokument z osadzonymi stylami i logiką, eliminujący zależności pomiędzy ekranami. Wszystkie zrzuty ekranu aplikacji webowej przedstawiono w wersji desktopowej, w celu poprawy czytelności. Zrzut ekranu strony głównej interfejsu ekranu służącą jako pulpit nawigacyjny do pozostałych zakładek funkcjonalnych, przedstawiono na Rysunek 40.

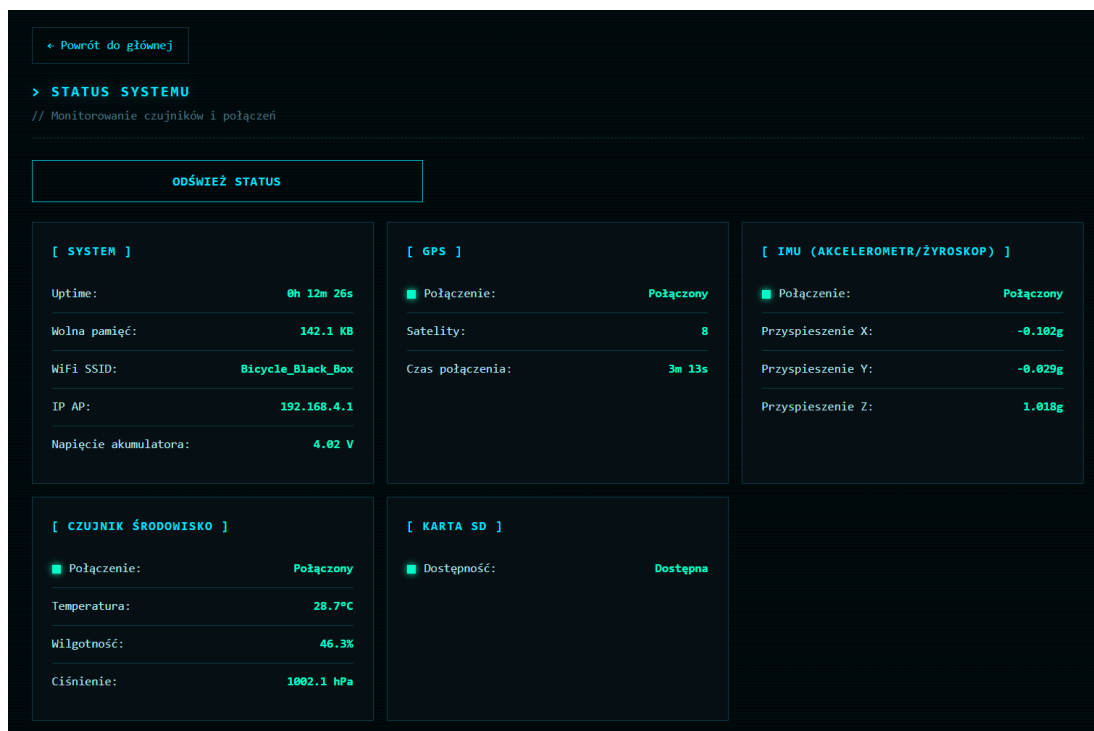


Rysunek 40. Strona główna interfejsu konfiguracyjnego z nawigacją do poszczególnych zakładki (źródło: opracowanie własne).

Wszystkie zasoby aplikacji webowej: dokumenty HTML, arkusze stylów CSS oraz skrypty JavaScript są wbudowane bezpośrednio w firmware urządzenia jako stałe znakowe w pamięci flash, generowane dynamicznie przez funkcje serwera w odpowiedzi na żądania klientów. Rozwiązanie to eliminuje konieczność wykorzystania zewnętrznego systemu plików (SPIFFS, LittleFS) dla zasobów interfejsu, kosztem zwiększenia rozmiaru obrazu firmware. Interfejs udostępnia sześć stron funkcjonalnych: pulpit główny z odnośnikami do pozostałych zakładek, podgląd statusu systemu w czasie rzeczywistym, historia zarejestrowanych jazd, lista zdarzeń bezpieczeństwa, konfiguracja modułu rejestratora wstrząsów, konfiguracja kontaktu alarmowego oraz panel deweloperski przeznaczony do zaawansowanej konfiguracji parametrów algorytmicznych.

Warstwa stylistyczna interfejsu zrealizowana została jako autorski arkusz CSS osadzony bezpośrednio w dokumentach HTML, bez zastosowania zewnętrznych frameworków stylistycznych (Bootstrap, Tailwind). Przyjęto motyw ciemny w stylistyce konsolowo-technicznej, wykorzystujący zmienne CSS dla spójności kolorystycznej oraz krój monospacujący JetBrains Mono dla wyświetlania danych technicznych. Interfejs zaprojektowano z uwzględnieniem responsywności poprzez zapytania medialne (@media), zapewniając czytelność zarówno na ekranach urządzeń mobilnych, jak i na monitorach komputerów stacjonarnych. Warstwa skryptowa zrealizowana została w czystym języku JavaScript, bez zastosowania zewnętrznych bibliotek aplikacyjnych. Komunikacja z serwerem realizowana jest przez funkcję `fetch()` opakowaną w autorski wrapper `webFetch()`, automatycznie obsługujący uwierzytelnienie HTTP Basic w nagłówkach żądań.

Zakładka statusu systemu, prezentująca dane diagnostyczne urządzenia w czasie rzeczywistym, ilustruje sposób prezentacji informacji w przyjętej stylistyce - kompaktowy układ pól numerycznych w siatce, wyróżnione kolorystycznie etykiety oraz cykliczne odświeżanie wartości przez zapytania do interfejsu API. Wygląd zakładki przedstawiono na Rysunek 41.



Rysunek 41. Zakładka statusu systemu prezentująca dane diagnostyczne w czasie rzeczywistym (źródło: opracowanie własne).

Wstępny szkielet kodu serwera HTTP oraz koncepcja interfejsu graficznego stron konfiguracyjnych zostały opracowane z wykorzystaniem asystenta programistycznego Cursor, po czym poddane samodzielnemu rozszerzeniu obejmującemu pełną logikę autoryzacji typu Basic Auth, zestaw endpointów REST API, integrację z modułami pomiarowymi, mechanizm zarządzania sesjami konfiguracyjnymi oraz cache historii jazd w pamięci operacyjnej. Logika integracji z firmware - implementacja endpointów API, obsługa uwierzytelnienia, walidacja struktur JSON oraz zarządzanie limitami parsera HTTP - została zaprojektowana i zaimplementowana autorsko w module `webserver.cpp`. Szczegółowy opis zakresu wykorzystania asystenta AI zawarto w Raporcie użycia AI dołączonym jako załącznik do pracy.

6.6.4. Interfejs programistyczny API

Komunikacja warstwy klienckiej z firmware urządzenia realizowana jest przez interfejs programistyczny API, oparty na protokole HTTP oraz formacie wymiany danych JSON. Interfejs udostępnia łącznie 22 endpointy, pogrupowane w pięć kategorii funkcjonalnych zestawionych w Tabeli 13. Wszystkie endpointy w obrębie ścieżek `/api/*` oraz `/shock/*` wymagają uwierzytelnienia podstawowego HTTP (*HTTP Basic Authentication*).

Funkcjonalność udostępniana przez interfejs API jest reprezentowana w warstwie prezentacyjnej w postaci dedykowanych zakładek. Zakładka historii jazd, agregująca dane z endpointów `/api/rides` oraz `/api/download`, prezentuje listę zarejestrowanych sesji wraz z ich metadanymi oraz udostępnia opcje pobrania surowych danych lub usunięcia wybranych wpisów. Wygląd zakładki przedstawiono na Rysunek 42.

/shock/config	GET	Odczyt konfiguracji rejestratora wstrząsów
/shock/config	POST	Aktualizacja konfiguracji wraz z walidacją parametrów
/shock/session/start	GET	Ręczne uruchomienie sesji rejestratora
/shock/session/stop	GET	Ręczne zakończenie sesji rejestratora
/shock/marker/<label>	GET	Dodanie etykiety zdarzenia do bieżącej sesji
Kalibracja modułu inercyjnego		
/api/imu/cal	GET	Odczyt aktualnych offsetów kalibracyjnych
/api/imu/calibrate	POST	Uruchomienie procedury kalibracyjnej (czas trwania ok. 2 s)
Konfiguracja urządzenia		
/api/config/dev	GET	Pełna konfiguracja deweloperska urządzenia
/api/config/dev	POST	Aktualizacja parametrów konfiguracji deweloperskiej
/api/dev/operating-mode	GET	Bieżący tryb pracy urządzenia
/api/dev/operating-mode	POST	Zmiana trybu pracy z trwałą persystencją
/api/contacts	GET	Konfiguracja kontaktu alarmowego
/api/contacts/update	POST	Aktualizacja kontaktu alarmowego

Endpointy realizujące modyfikację stanu systemu (POST, DELETE) operują na danych wejściowych w formacie JSON z walidacją struktury oraz zakresów dopuszczalnych wartości. Niedopuszczalne dane wejściowe skutkują odpowiedzią HTTP 400 z opisem błędu, bez modyfikacji stanu urządzenia. Operacje pobrania plików (/api/download) realizowane są jako strumień bajtów odczytywanych bezpośrednio z karty microSD, z wykorzystaniem mechanizmu generowania plików GPX oraz metadanych w czasie rzeczywistym, jeśli odpowiednie pliki uzupełniające nie zostały wcześniej wygenerowane.

6.6.5. Przepływ obsługi zapytań HTTP

Warstwa serwera HTTP zawiera mechanizmy wspierające zachowanie responsywności systemu podczas operacji o znacznej złożoności, w szczególności podczas skanowania struktury katalogowej karty microSD oraz generowania dużych odpowiedzi (pliki CSV jazd, pliki GPX). Centralnym mechanizmem jest przekazywanie sterowania systemowi operacyjnemu podczas długotrwałych pętli przetwarzania danych, funkcja `yield()` wywoływana jest co 32 wiersze podczas agregacji plików CSV jazd, co 16 trackpointów podczas generowania pliku GPX, co 32 wiersze podczas generowania metadanych oraz po każdym buforze 512 bajtów podczas pobierania plików binarnych. Mechanizm ten zapobiega aktywacji systemowego watchdoga oraz pozwala na obsługę innych zadań systemu FreeRTOS (w szczególności stosu Wi-Fi) w trakcie wykonywania operacji blokujących pętlę główną.

Drugim mechanizmem jest specjalizowany wariant wzorca bramki czasowej opisanego w podrozdziale 6.1.2, zastosowany w pętli oczekiwania na bajty protokołu TCP. Para `yield()` w połączeniu z `delay(1)` realizuje cykl oczekiwania na zdarzenia sieciowe z limitem czasowym wynoszącym pięć sekund `yield()` przekazuje sterowanie zadaniom systemu FreeRTOS odpowiedzialnym za obsługę stosu sieciowego, natomiast `delay(1)` zapobiega bezproduktywnemu wykorzystaniu czasu procesora w sytuacji, gdy bufor odbiorczy TCP jest tymczasowo pusty.

Przepływ obsługi pojedynczego żądania HTTP zilustrowano w Listing 8 w postaci uproszczonej, pominięto szczegóły parsowania nagłówków oraz obsługę błędów.

Listing 8. Uproszczony przepływ obsługi żądania HTTP w funkcji

```
bool WebServer::handleClient() {
    WiFiClient client = _server.accept();
    if (!client) return false;

    const uint32_t deadline = millis() + WEB_REQUEST_TIMEOUT_MS;
    char method[8], path[WEB_MAX_PATH_LEN];

    // Parsowanie żądania HTTP z timeoutem
    while (client.connected() && millis() < deadline) {
        if (!client.available()) {
            yield();
            delay(1); // throttle pollingu sieci
            continue;
        }
        // ... odczyt linii nagłówków HTTP
    }
    // Walidacja uwierzytelnienia HTTP Basic
    if (!checkAuth(client)) {
        sendUnauthorized(client);
        return true;
    }

    // wybór handlera na podstawie ścieżki
    if (strncmp(path, "/api/", 5) == 0) {
        handleAPIRequest(client, method, path);
    } else if (strncmp(path, "/shock/", 7) == 0) {
        handleShockAPIRequest(client, method, path);
    } else if (strcmp(path, "/") == 0) {
        sendDashboard(client);
    } else if (endsWith(path, ".html")) {
        handleHTMLRequest(client, path);
    } else {
        sendNotFound(client);
    }

    client.stop();
    return true;
}
```

`WebServer::handleClient()`. Routing realizowany jest w sposób manualny, bezpośrednie porównanie ścieżki żądania ze zdefiniowanymi prefiksami.

Świadome zrezygnowanie z mechanizmu rejestracji handlerów przez wzorec `server.on()`, charakterystycznego dla gotowych bibliotek HTTP, zostało podyktowane chęcią zachowania pełnej kontroli nad kolejnością walidacji oraz uproszczenia diagnostyki przepływu obsługi żądania. W obecnej implementacji wszystkie endpointy zdefiniowane są w pojedynczej funkcji nadrzędnej, co ułatwia analizę całościowego zachowania systemu w obliczu ograniczonej liczby endpointów.

6.6.6. Bezpieczeństwo i integracja z resztą systemu

Warstwa bezpieczeństwa interfejsu konfiguracyjnego zrealizowana została w trzech komplementarnych warstwach ochrony, których kombinacja zapewnia poziom bezpieczeństwa adekwatny do prototypowego charakteru urządzenia. Pierwszą warstwą jest fizyczna kontrola dostępu - interfejs WWW nie jest dostępny domyślnie, jego aktywacja wymaga fizycznego dostępu do urządzenia oraz przyciśnięcia dedykowanego przycisku serwisowego, co eliminuje możliwość zdalnego ataku w

warunkach normalnej eksploatacji. Drugą warstwą jest szyfrowanie warstwy radiowej w standardzie WPA2, zabezpieczające komunikację Wi-Fi pomiędzy urządzeniem a klientem przed pasywnym podsłuchem. Trzecią warstwą jest uwierzytelnienie HTTP Basic na poziomie aplikacyjnym, wymagające podania nazwy użytkownika oraz hasła przy każdym żądaniu do endpointów API oraz dokumentów HTML.

Konfiguracja warstw zabezpieczeń jest w pełni konfigurowalna z poziomu interfejsu konfiguracyjnego - hasło sieci Wi-Fi, dane uwierzytelniające HTTP Basic oraz maksymalna liczba jednocześnie klientów AP zapisywane są w pliku konfiguracyjnym na karcie microSD. Domyślne wartości skompilowane w firmware służą wyłącznie jako odwołanie w sytuacji braku konfiguracji użytkownika i są zaprojektowane do natychmiastowej zmiany po pierwszej sesji konfiguracyjnej. Edycja kontaktu alarmowego oraz wybranych parametrów pracy urządzenia odbywa się przez dedykowaną zakładkę ustawień, której wygląd przedstawiono na Rysunek 43. Trójwarstwowy model bezpieczeństwa wymaga, aby potencjalny napastnik jednocześnie znajdował się w fizycznej bliskości urządzenia, dysponował hasłem sieci Wi-Fi oraz danymi uwierzytelniającymi do interfejsu - kombinacja ta uznana została za wystarczającą dla prototypu badawczego, choć dla potencjalnej wersji produkcyjnej wskazano w rozdziale 8.4 kierunki dalszego rozwoju w obszarze bezpieczeństwa.

[← Powrot do glownej](#)

> **USTAWIENIA**

// Kontakt alarmowy i usypianie układu

[**KONTAKT ALARMOWY**]

Ustaw numer telefonu do SMS alarmowych.

AKTUALNY KONTAKT

Numer telefonu: +48

Nazwa kontaktu:

Ostatnia aktualizacja: 14.06.2026, 12:48:15

Kierunkowy Numer (9 cyfr)

np. 48 spacje OK - zapis bez spacji

Nazwa kontaktu

ZAPISZ KONTAKT ALARMOWY

[**DEEP SLEEP PO BEZRUCHU**]

Po N sekundach ciągłej pozycji statycznej (brak ruchu według GPS/IMU) układ wchodzi w deep sleep. Wartość 0 wylacza auto-wylaczanie Klamra: 0 lub 5..3600 s.

Aktualnie: WYLACZONY s

Sleep po bezruchu [s] (0 = wylaczony)

Rysunek 43. Zakładka konfiguracji kontaktu alarmowego (źródło: opracowanie własne).

Integracja serwera WWW z pozostałymi modułami systemu realizowana jest poprzez wzorzec wstrzykiwania zależności w konstruktorze klasy `WebServer`. Referencje do instancji menedżera karty SD, magazynu danych oraz rejestratora wstrząsów przekazywane są w momencie tworzenia obiektu w funkcji `setup()`. Mechanizm ten eliminuje konieczność stosowania globalnych singletonów oraz upraszcza testowanie poszczególnych komponentów w izolacji. Sterowanie modemem GSM nie jest dostępne z poziomu serwera WWW, celowo rozdzielono warstwę konfiguracyjną od warstwy alarmowej, zapobiegając niezamierzonemu uruchomieniu procedury alarmowej z poziomu interfejsu webowego.

Ze względu na jednowątkową architekturę systemu, obsługa pojedynczego żądania HTTP blokuje wykonanie pozostałych modułów pętli głównej na czas do pięciu sekund. W trakcie obsługi żądania funkcja `sensorsRead()`, algorytm detekcji upadku oraz mechanizmy aktualizacji modułu `shockcapture.cpp` pozostają nieaktywne. Decyzja ta jest świadomym kompromisem, akceptowalnym ze względu na ograniczony scenariusz wykorzystania interfejsu. Sesja serwisowa odbywa się typowo w warunkach kontrolowanych (rower nieruchomy, brak ekspozycji na zagrożenia), w których chwilowe wyłączenie detekcji upadku nie wpływa na bezpieczeństwo użytkownika. Po zakończeniu obsługi żądania pętla główna wznowia normalną pracę bez konieczności reinicjalizacji modułów.

7. Testy i walidacja prototypu

Niniejszy rozdział przedstawia weryfikację działania zaprojektowanego prototypu cyfrowej czarnej skrzynki rowerowej. Walidacja oparta została na metodzie hybrydowej, łączącej automatyczną akwizycję danych pomiarowych przez firmware urządzenia z niezależnymi pomiarami laboratoryjnymi oraz testami terenowymi. Strukturę rozdziału zorganizowano w cztery podrozdziały: metodologię badań, pomiary warstwy sprzętowej oraz charakterystyk energetycznych, walidację algorytmu detekcji upadku oraz testy terenowe w warunkach rzeczywistej eksploatacji, zakończone podsumowaniem osiągniętych rezultatów.

7.1. Metodologia testów

Ze względu na wieloaspektowy charakter weryfikowanego systemu, obejmującego zarówno warstwę sprzętową (pobór energii, charakterystyki czujników, komunikacja radiowa), jak i algorytmiczną (klasyfikacja zdarzeń krytycznych, agregacja danych nawigacyjnych), przyjęto hybrydową metodologię testową łączącą dwa komplementarne źródła danych. Pierwszym źródłem jest strumień danych rejestrowany automatycznie przez firmware urządzenia - pliki CSV logu jazdy z częstotliwością 1 Hz, pliki rejestratora wstrząsów z częstotliwością 200 Hz, metadane sesji w formacie JSON oraz logi diagnostyczne procedury alarmowej GSM. Drugim źródłem są pomiary laboratoryjne i obserwacje empiryczne wykonane niezależnymi przyrządami pomiarowymi. (multimetrem cyfrowym, oscyloskopem oraz przyrządami referencyjnymi dla pomiarów środowiskowych).

Kluczowym założeniem architektonicznym ułatwiającym proces walidacji była separacja kodu od konfiguracji, opisana w podrozdziale 3.2.5. wszystkie parametry decyzyjne algorytmów oraz parametry czasowe procedur są wczytywane z plików konfiguracyjnych w formacie JSON na karcie microSD podczas rozruchu urządzenia. Mechanizm ten umożliwił iteracyjne dostrajanie wartości progowych w toku testów terenowych bez konieczności rekompilacji firmware oraz ponownego flashowania urządzenia.

7.1.1. Tryby pracy w procedurach testowych

Architektura firmware przewiduje trzy tryby pracy różniące się reakcją systemu na zdarzenia krytyczne. Tryby te zostały świadomie wykorzystane podczas weryfikacji w celu rozdzielenia testów algorytmicznych od testów funkcjonalnych komunikacji GSM. Konfigurację trybu pracy realizowano z poziomu interfejsu konfiguracyjnego WWW lub przez bezpośrednią edycję pliku `dev.json` na karcie microSD, z trwałą persystencją wybranej wartości pomiędzy cyklami zasilania. Testowe tryby oraz ich funkcjonalność zostały przedstawione w Tabeli 14.

Tabela 14. Wykorzystanie trybów pracy w procedurach testowych.

Tryb	Reakcja na zdarzenie krytyczne	Zastosowanie w testach
Trening 1	Wykryty upadek nie wyzwala sekwencji alarmowej. Ręczne wyzwolenie przyciskiem alarmowym powoduje bezwarunkową wysyłkę SMS.	Testy funkcjonalne komunikacji GSM bez ryzyka generowania fałszywych alarmów podczas eksperymentów algorytmicznych.
Trening 2	Wykryty upadek generuj znacznik diagnostyczny w pliku rejestratora oraz wywołanie sekwencji alarmowej, bez wysyłki SMS	Wielokrotne testy algorytmu detekcji upadki zarówno w warunkach laboratoryjnych jak i terenowych, bez generowania niepotrzebnego ruchu w sieci komórkowej
Praca	Pełen scenariusz alarmowy z 30 sekundowym oknem wstępnego alarmu oraz wysyłką SMS, możliwość ręcznego	Testy integracyjne pełnego scenariusza alarmowego od detekcji do dostarczania powiadomienia

	wyzwolenia trybu paniki przez przytrzymanie przycisku alarmu	
--	--	--

7.1.2. Aparatura pomiarowa

Pomiary charakterystyk energetycznych oraz porównawcze pomiary referencyjne wykonano przy użyciu przyrządów zestawionych w Tabeli 15. Multimetr cyfrowy włączano szeregowo w obwód zasilający urządzenie, w miejscu pomiędzy ogniwem litowo-jonowym a układem dystrybucji zasilania, umożliwiając pomiar wartości średnich poboru prądu w stabilnych trybach pracy. Oscyloskop cyfrowy ustawiono w tryb przewijania (*roll*), aby zarejestrować przebieg amplitudy oraz czasu trwania szczytowych impulsów prądowych podczas inicjalizacji modemu, rejestracji w sieci komórkowej oraz transmisji właściwej wiadomości SMS. Pomiary referencyjne parametrów środowiskowych (temperatury i ciśnienia atmosferycznego - porównano z odczytami czujnika BME280 wbudowanego w prototyp, w warunkach stabilizacji termicznej trwającej co najmniej 30 minut.

Tabela 15. Aparatura pomiarowa wykorzystana w procedurach testowych.

Przyrząd	Model	Zakres / klasa dokładności	Zastosowanie
Multimetr cyfrowy	FLUKE 115 TRUE RMS MULTIMETER	10,00 A / 0,01 A	Pomiar wartości średnich poboru prądu w stabilnych trybach pracy
Oscyloskop cyfrowy	Teledyne LeCroy Wavesurfer 3034	350 MHz / 4 GS/s / 1 mV/div	Rejestracja przebiegów impulsowych prądu zasilania modemu GSM
Termometr referencyjny	Czujnik temperatury i wilgotności XIAOMI Mi 2	0,1°C	Porównanie weryfikacji odczytów czujnika BME280

Obok pomiarów wykonywanych zewnętrzną aparaturą, podczas weryfikacji systemu wykorzystano specjalizowane procedury akwizycji danych opierające się o wbudowane mechanizmy rejestracyjne firmware. W szczególności pomiar dryfu pozycji modułu nawigacji satelitarnej w warunkach statycznych realizowano przez aktywację parametru `rideStartWithoutMotion` w pliku konfiguracyjnym `dev.json`. Zmiana tego parametru wymuszała rozpoczęcie sesji jazdy mimo braku wykrytego ruchu pojazdu. Urządzenie pozostawione w stałej pozycji geograficznej przez czas trwania pomiaru rejestrowało dane geolokalizacyjne z częstotliwością 0,2 Hz, umożliwiając retrospektywną analizę odchylenia wskazań modułu od pozycji rzeczywistej oraz charakterystyki tego odchylenia w funkcji czasu. Procedura ta wykorzystywała dokładnie ten sam mechanizm akwizycji co podczas normalnej eksploatacji urządzenia, eliminując konieczność implementacji dedykowanego oprogramowania pomiarowego.

Przejazdy realizowano w różnych warunkach pokrycia siecią komórkową oraz w różnych warunkach widoczności satelitów, umożliwiając weryfikację działania systemu w pełnym zakresie scenariuszy eksploatacyjnych.

7.1.3. Metodologia pomiaru charakterystyk czujnika środowiskowego oraz modułu GPS

Celem badania czasu do pierwszego wskazania pozycji było określenie czasu potrzebnego modułowi GNSS do uzyskania pierwszego poprawnego wskazania pozycji (TTFF - *Time To First Fix*) w trzech scenariuszach uruchomienia, różniących się stopniem aktualności danych efemeryd przechowywanych w pamięci modułu. Wyróżniono scenariusz *cold start*, odpowiadający pierwszemu uruchomieniu modułu po długiej przerwie przy braku aktualnych danych w pamięci, scenariusz *warm*

start, odpowiadający uruchomieniu po kilkugodzinnej przerwie przy częściowo nieaktualnych danych efemeryd oraz scenariusz *hot start*, odpowiadający uruchomieniu po przerwie poniżej minuty przy w pełni aktualnych danych efemeryd. Dla każdego scenariusza wykonano po 10 niezależnych prób. Czas TTFF wyznaczano jako różnicę pomiędzy momentem uruchomienia modułu a momentem zarejestrowania pierwszej poprawnej ramki NMEA zawierającej ważne dane pozycji.

Celem badania dryfu pozycji w stanie statycznym było określenie precyzji (powtarzalności) wskazań pozycji modułu GNSS w warunkach statycznych, tj. bez rzeczywistego przemieszczania się urządzenia. Badanie to pozwala oszacować charakterystyczne dla odbiorników konsumenckich zjawisko "wędrowania" wskazywanej pozycji, wynikające z propagacji wielodrogowej sygnału, zmienności warunków atmosferycznych oraz zmieniającej się geometrii widocznych satelitów. Urządzenie umieszczono w lokalizacji o otwartym widoku nieba, a przy pomocy parametru konfiguracyjnego `rideStartWithoutMotion` wymuszono ciągłe rejestrowanie odczytów pozycji pomimo braku wykrytego ruchu (standardowo warunkującego rozpoczęcie zapisu trasy). Pomiar trwał 60 minut.

Dokładności pomiaru temperatury realizowanego przez czujnik BME280 zweryfikowano poprzez równoległy pomiar przy użyciu niezależnego czujnika referencyjnego o deklarowanej przez producenta dokładności $\pm 0,1^{\circ}\text{C}$. Czujnik referencyjny umieszczono bezpośrednio obok badanego urządzenia, tak aby oba czujniki znajdowały się w tych samych warunkach otoczenia. Odczyty rejestrowano co 30 minut, wykonując łącznie 10 pomiarów. Rozbieżność wskazań obu czujników posłużyła jako miara błędu pomiarowego BME280 względem przyjętego wzorca.

7.1.4. Metodologia walidacji algorytmu detekcji upadku

Walidacja algorytmu detekcji upadku przeprowadzona została w dwóch komplementarnych etapach. Etap pierwszy obejmował kontrolowane testy stanowiskowe w warunkach laboratoryjnych, podczas których odtwarzano cztery scenariusze klasyfikacyjne: symulację upadku statycznego (urządzenie zamontowane na ramie nieruchomego roweru, kontrolowane przewrócenie konstrukcji), symulację upadku w trakcie powolnej jazdy (prędkość do 5 km/h, kontrolowane wytrącenie z równowagi), test najazdu na krawężnik oraz test ręcznego trzęsienia urządzeniem. Dwa pierwsze scenariusze stanowiły wzorce pozytywne, algorytm powinien zaklasyfikować zdarzenie jako potwierdzony upadek. Dwa pozostałe stanowiły wzorce negatywne, algorytm powinien zarejestrować przekroczenie progu detekcji uderzenia, jednak w fazie obserwacji zaklasyfikować zdarzenie jako fałszywy alarm.

Etap drugi walidacji obejmował obserwację działania algorytmu podczas testów terenowych. w warunkach rzeczywistej eksploatacji rowerowej, bez celowego prowokowania zdarzeń krytycznych. Rolą tego etapu była weryfikacja odporności algorytmu na fałszywe alarmy generowane przez naturalne perturbacje sygnału inercyjnego najazdu na nierówności nawierzchni, gwałtowne hamowania, wstrząsy przenoszone z ramy podczas jazdy po nawierzchni kostkowej oraz typowe manewry ruchu rowerowego. Wszystkie testy etapu drugiego przeprowadzono w trybie Trening 2, co umożliwiło wielokrotną weryfikację bez wysyłki niepotrzebnych wiadomości SMS. Wyniki obu etapów walidacji przedstawiono w rozdziale 7.

7.2. Testy sprzętowe i charakterystyk energetycznych

Niniejszy podrozdział prezentuje wyniki pomiarów warstwy sprzętowej prototypu, obejmujących charakterystyki energetyczne systemu, parametry czujników akwizycji danych oraz niezawodność warstwy komunikacji komórkowej. Wszystkie pomiary wykonano zgodnie z metodologią opisaną w podrozdziale 7.1, z wykorzystaniem aparatury zestawionej w Tabeli 15.

7.2.1. Pomiary poboru prądu w trybach pracy

Pomiar wartości średnich poboru prądu w stabilnych trybach pracy wykonano multimetrem cyfrowym włączonym szeregowo w obwód zasilający, w miejscu pomiędzy ogniwem litowo-jonowym a układem dystrybucji zasilania. Pomiar realizowano przez czas co najmniej 60 sekund w każdym z

trybów, po wcześniejszej stabilizacji urządzenia w danym stanie. Pomiary szczytowego poboru prądu modemu GSM podczas transmisji wiadomości SMS przeprowadzono z wykorzystaniem programowalnego zasilacza imitującego charakterystykę ogniwa Sony US18650VTC6 w zakresie napięć od 3,1 V do 4,1 V z krokiem 0,1 V, rejestrując jednocześnie czterema kanałami oscyloskopu: napięcie linii zasilającej imitujące ogniwo, napięcie linii 5 V wyjściowej z przetwornicy TPS63020, sygnał ENABLE uruchamiający przetwornicę oraz przebieg prądowy z linii zasilania modemu. Wyniki pomiarów zestawiono w Tabeli 16.

Wyróżniono cztery stabilne tryby pracy odpowiadające kluczowym scenariuszom eksploatacyjnym urządzenia:

- **Tryb głębokiego uśpienia** (*deep sleep*) - reprezentuje stan oczekiwania urządzenia na zdarzenie wybudzające - mikrokontroler ESP32-C6 w stanie uśpienia, gotowy do wybudzenia sygnałem z cyfrowego czujnika wstrząsów;
- **Tryb aktywnej rejestracji** - odpowiada normalnej eksploatacji urządzenia podczas jazdy, wszystkie czujniki akwizycji danych aktywne, log CSV zapisywany z częstotliwością 1 Hz, rejestrator wstrząsów próbkujący z częstotliwością 200 Hz;
- **Tryb aktywny z aktywnym serwerem WWW** - reprezentuje sesję serwisową, czyli dodatkowy pobór generowany przez radio Wi-Fi w trybie punktu dostępu;
- **Szczytowy pobór modemu GSM** - podczas transmisji SMS krótkotrwały impuls prądowy w warunkach średniego zasięgu sieci komórkowej (obszar peryferyjny zabudowy miejskiej).

Tabela 16. Pomiar poboru prądu w trybach pracy.

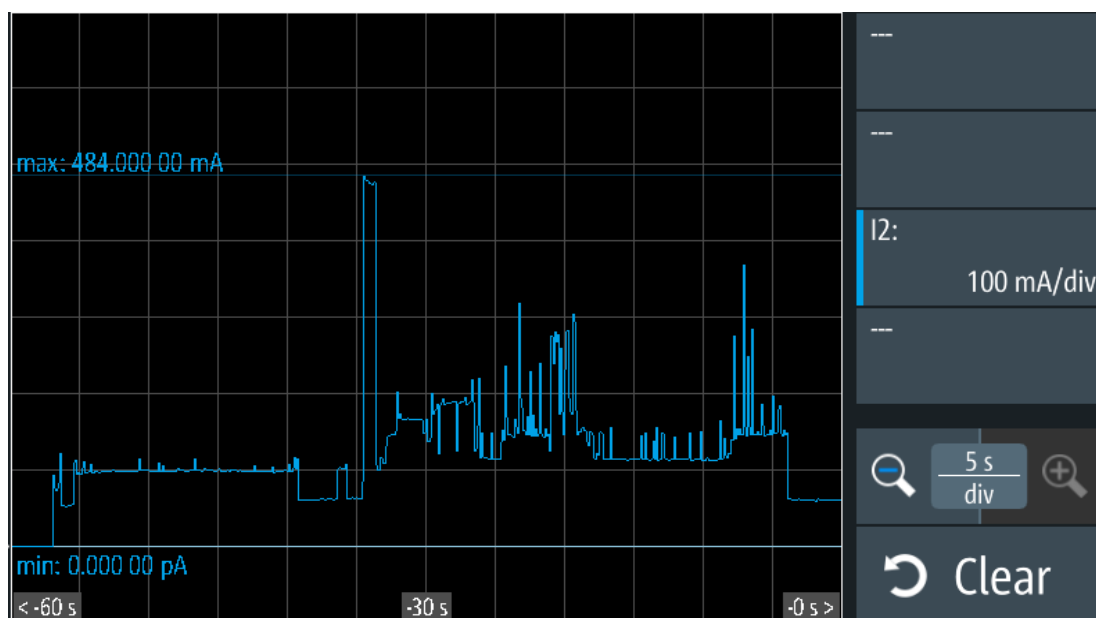
Tryb pracy	Aktywne moduły	Pobór prądu [mA]	Aparatura
Głębokie uśpienie	Mikrokontroler w stanie głębokiego snu, czujnik wstrząsów aktywny	31	Multimetr
Aktywna rejestracja	IMU, GPU, BME280, log SD 1 HZ, shockcapture 200Hz	63	Multimetr
Aktywna rejestracja i WWW	IMU, GPU, BME280, log SD 1 HZ, shockcapture 200Hz, Wi-Fi w trybie AP	165	Multimetr
Szczytowy impuls modemu GSM	Modem SIM800L podczas transmisji SMS	492	Zasilacz

Pobór prądu w trybie głębokiego uśpienia wynoszący 31 mA stanowi wartość znacząco wyższą od typowych charakterystyk samego mikrokontrolera ESP32-C6, którego pobór w stanie głębokiego snu deklarowany przez producenta wynosi rząd kilkudziesięciu mikroamperów. Zarejestrowana wartość jest wypadkową poboru wszystkich peryferiów obecnych na płycie drukowanej prototypu, zasilanych ze wspólnej szyny napięcia 3,3 V układ obecnej rewizji nie przewiduje niezależnego odłączania zasilania modułów peryferyjnych od mikrokontrolera w trakcie cykli uśpienia. Kierunki potencjalnych modyfikacji warstwy sprzętowej zmierzających do redukcji poboru w stanie uśpienia przedstawiono w podrozdziale 8.4.

Wartość poboru prądu w trybie aktywnej rejestracji wynosząca 63 mA jest zgodna z oczekiwaniami dla mikrokontrolera ESP32-C6 wraz z modułem nawigacji satelitarnej, czujnikiem inercyjnym oraz czujnikiem środowiskowym pracującymi w stabilnych warunkach akwizycji. Aktywacja serwera WWW w trybie punktu dostępu Wi-Fi generuje dodatkowy narzut poboru prądu wynoszący około 102 mA, mieszczący się w typowym zakresie zużycia energetycznego radia Wi-Fi

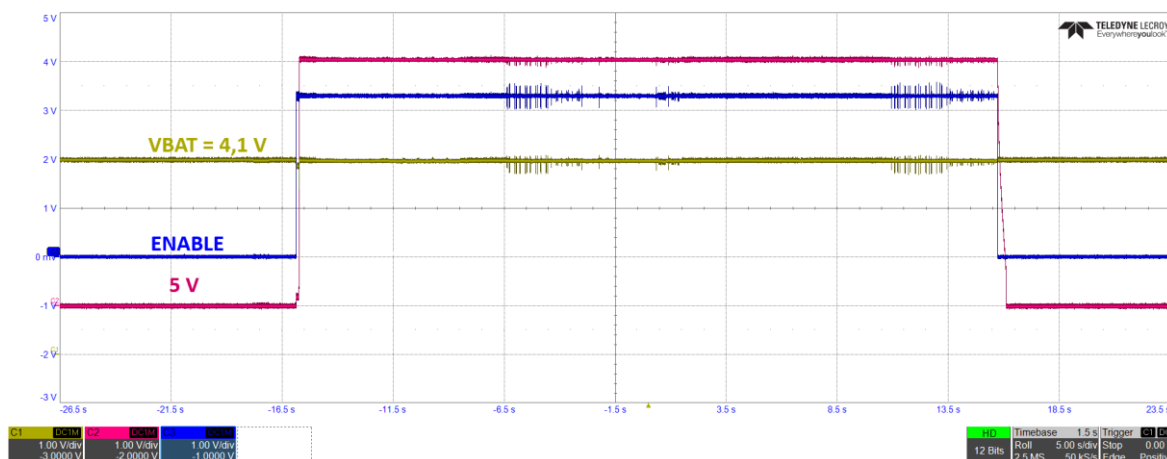
mikrokontrolera ESP32-C6 w konfiguracji AP. Zarejestrowane wartości potwierdzają zasadność rozłącznego traktowania trybu serwisowego. Automatyczne wyłączanie interfejsu WWW po okresie bezczynności, opisane w podrozdziale 6.6.2, zapobiega niepotrzebnemu obciążaniu ogniwa podczas normalnej eksploatacji urządzenia.

Profil poboru prądu modemu SIM800L podczas pełnej procedury wysyłki wiadomości SMS przedstawiono na Rysunek 44. Wykres prezentuje uśredniony pobór prądu w skali sekundowej, w pełnym cyklu pojedynczej procedury alarmowej. Od momentu czuwania i zaobserwowania zmiany stanu pinu ENABLE (generowany przez MCP23008, uruchamiający przetwornicę), uruchomienia przetwornicy zasilającej, przez fazę inicjalizacji modułu i rejestracji w sieci komórkowej, po właściwą transmisję wiadomości oraz wyłączenie modemu. Maksymalna zarejestrowana wartość średniego poboru wyniosła 492 mA, występująca w fazie aktywnej transmisji wiadomości; faza inicjalizacji modułu charakteryzowała się znacznie niższym, stabilnym poborem rzędu 80 do 100 mA. Czas trwania pełnej sesji od uruchomienia zasilania modemu do jego wyłączenia wyniósł około 30 sekund, z czego właściwa transmisja wiadomości obejmowała ostatnie kilkanaście sekund procedury. Pomiar wykonano w warunkach średniego zasięgu sieci komórkowej (obszar peryferyjny zabudowy miejskiej), w warunkach silniejszego pokrycia, gdy modem operuje z niższą mocą wyjściową nadajnika radiowego, wartość ta byłaby odpowiednio niższa, natomiast w warunkach marginalnego zasięgu modem podejmuje próby transmisji z maksymalną dostępną mocą.



Rysunek 44. Profil poboru prądu modemu SIM800L podczas pełnej procedury wysyłki wiadomości SMS (źródło: opracowanie własne).

Komplementarny obraz pełnej sekwencji sterowania modemem GSM, ilustrujący koordynację sygnału uruchamiającego przetwornicę zasilającą z linią wyjściową 5 V, przedstawiono na Rysunek 45. Pomiar zarejestrowano oscyloskopem cyfrowym z trzema kanałami: napięcie wejściowe imitujące ogniwo (C1, 4,1 V), napięcie wyjściowe przetwornicy TPS63020 zasilające modem (C2, 5 V) oraz sygnał ENABLE generowany przez ekspander portów MCP23008 (C3, poziom logiczny 3,3 V). Sekwencja uruchomienia modemu rozpoczyna się ustawieniem sygnału ENABLE w stan wysoki, co inicjuje pracę przetwornicy i natychmiastowe pojawienie się stabilnego napięcia 5 V na linii zasilania modemu. W trakcie aktywnej sesji wymiany komend AT oraz transmisji wiadomości widoczne są szумы modulacyjne na liniach zasilających, generowane przez impulsowy charakter pracy nadajnika modemu. Wyłączenie modemu po zakończeniu sesji realizowane jest opadającym zboczem sygnału ENABLE, skutkującym natychmiastowym odłączeniem zasilania od układu.



Rysunek 45. Poziomy napięć podczas trwania sekwencji GSM (źródło: opracowanie własne).

Widoczna sekwencja: stan wyłączony przed - 16,5 s, uruchomienie przetwornicy zboczem narastającym ENABLE, aktywna sesja SMS o czasie trwania około 30 sekund, wyłączenie modemu zboczem opadającym ENABLE.

7.2.2. Autonomia energetyczna systemu

Weryfikację realnego czasu pracy urządzenia w trybach eksploatacyjnych przeprowadzono trzema niezależnymi metodami, dostosowanymi do charakterystyki każdego z badanych scenariuszy.

Pomiar autonomii w trybie **aktywnej rejestracji** zrealizowano metodą bezpośrednią, naładowane do pełna ogniwo Sony US18650VTC6 o pojemności katalogowej 3000 mAh zostało podłączone do urządzenia, które uruchomiono w trybie aktywnej rejestracji jazdy, z `rideEndAfterStationarySec=0` (kontynuowanie logu mimo postoju) oraz `rideStartWithoutMotion` (rozpoczęcie logu bez ruszenia rowerem) a następnie pozostawiono do pełnego rozładowania. Czas trwania sesji rejestrowano automatycznie przez firmware w polu `uptime_s` plików metadanych - mechanizm ten umożliwił precyzyjny pomiar bez konieczności użycia zewnętrznego zegara, ponieważ mikrokontroler pozostawał aktywny przez cały czas trwania testu. Pomiar zakończono w momencie odłączenia urządzenia przez zabezpieczenie nadmiernego rozładowania ogniwa.

Pomiar autonomii w trybie **głębokiego uśpienia** wymagał odmiennego podejścia metodologicznego. W trybie głębokiego snu wewnętrzny licznik mikrokontrolera `millis()`, stanowiący podstawę pola `uptime_s`, nie jest inkrementowany, co uniemożliwia bezpośrednie wykorzystanie firmware do rejestracji czasu trwania testu. Pomiar realizowano metodą zewnętrzną, naładowane do pełna urządzenie zostało wprowadzone w stan uśpienia o znanym znaczniku czasowym, a następnie okresowo wybudzane cyfrowym detektorem wstrząsów w odstępach jednodobowych. Po każdym wybudzeniu odczytywano bieżące napięcie ogniwa przez interfejs konfiguracyjny WWW, monitorując postępujący proces rozładowania oraz multimetr cyfrowy ustawiony w tryb woltomierza. Test zakończono w momencie, w którym napięcie ogniwa zbliżyło się do progu odcięcia układu zabezpieczającego.

Pomiar autonomii w **trybie serwisowym** przeprowadzono analogicznie do trybu aktywnej rejestracji, z aktywowanym przyciskiem serwisowym uruchamiającym WiFi AP, podtrzymywanym w stanie aktywnym przez cykliczne zapytania interfejsu API z połączanego urządzenia klienckiego. Czas trwania testu rejestrowano na podstawie pola `uptime_s` w plikach metadanych jazdy zapisywanych równolegle w tle.

Tabela 17. Wyniki testów autonomii energetycznej urządzenia.

Scenariusz eksploatacyjny	Wymaganie [h]	Wartość zmierzona [h]	Spełnienie założeń
Aktywna rejestracja jazdy	≥ 24	32	TAK
Tryb głębokiego uśpienia z monitorowaniem wybudzenia	≥ 168	71	NIE
Tryb serwisowy z aktywnym serwerem WWW	$\geq 0,5$	12	TAK

Otrzymane wartości autonomii w trzech trybach pracy (Tabela 17) stanowią podstawę walidacji spełnienia wymagań sformułowanych w podrozdziale 3.1.5. Szczegółowa dyskusja uzyskanych wyników w kontekście możliwości optymalizacji warstwy sprzętowej prototypu przedstawiona została w podrozdziale 8.4.

7.2.3. Pomiary charakterystyk czujnika środowiskowego oraz modułu GPS

Walidacja modułu nawigacji satelitarnej oraz czujnika środowiskowego przeprowadzona została w warunkach umożliwiających niezależną weryfikację dostarczanych przez nie odczytów. Analizy dokonano poprzez porównanie wyników z aparaturą referencyjną, jak i przez analizę rejestrowanych danych w funkcji czasu.

Pomiary parametrów modułu GPS obejmowały trzy aspekty: czas uzyskania pierwszego fixu pozycji (TTFF - *Time To First Fix*), dryf pozycji w stanie statycznym oraz liczbę widzianych satelitów. Pomiary TTFF realizowano poprzez odczyt wartości pola `connectionTime` z interfejsu API `/api/status`, rejestrującego czas pomiędzy uruchomieniem modułu a momentem uzyskania pierwszego ważnego fixu.

Wyniki pomiarów TTFF dla trzech badanych scenariuszy przedstawiono w Tabeli 18. Dla scenariusza *cold start*, odpowiadającego pierwszemu uruchomieniu modułu po długiej przerwie przy braku aktualnych danych w pamięci, mediana czasu do uzyskania pierwszego fixu ($n=10$, liczba prób) wyniosła 38,05 s, przy rozstępie wyników od 30,20 s do 52,20 s. W scenariuszu *warm start*, przy częściowo nieaktualnych danych efemeryd po kilkugodzinnej przerwie, mediana TTFF spadła do 8,05 s przy zakresie odczytów 6,20 s do 10,56 s. W scenariuszu *hot start*, przy aktualnych danych efemeryd po przerwie poniżej minuty, mediana wyniosła zaledwie 1,11 s, przy zakresie 0,80 - 1,57 s.

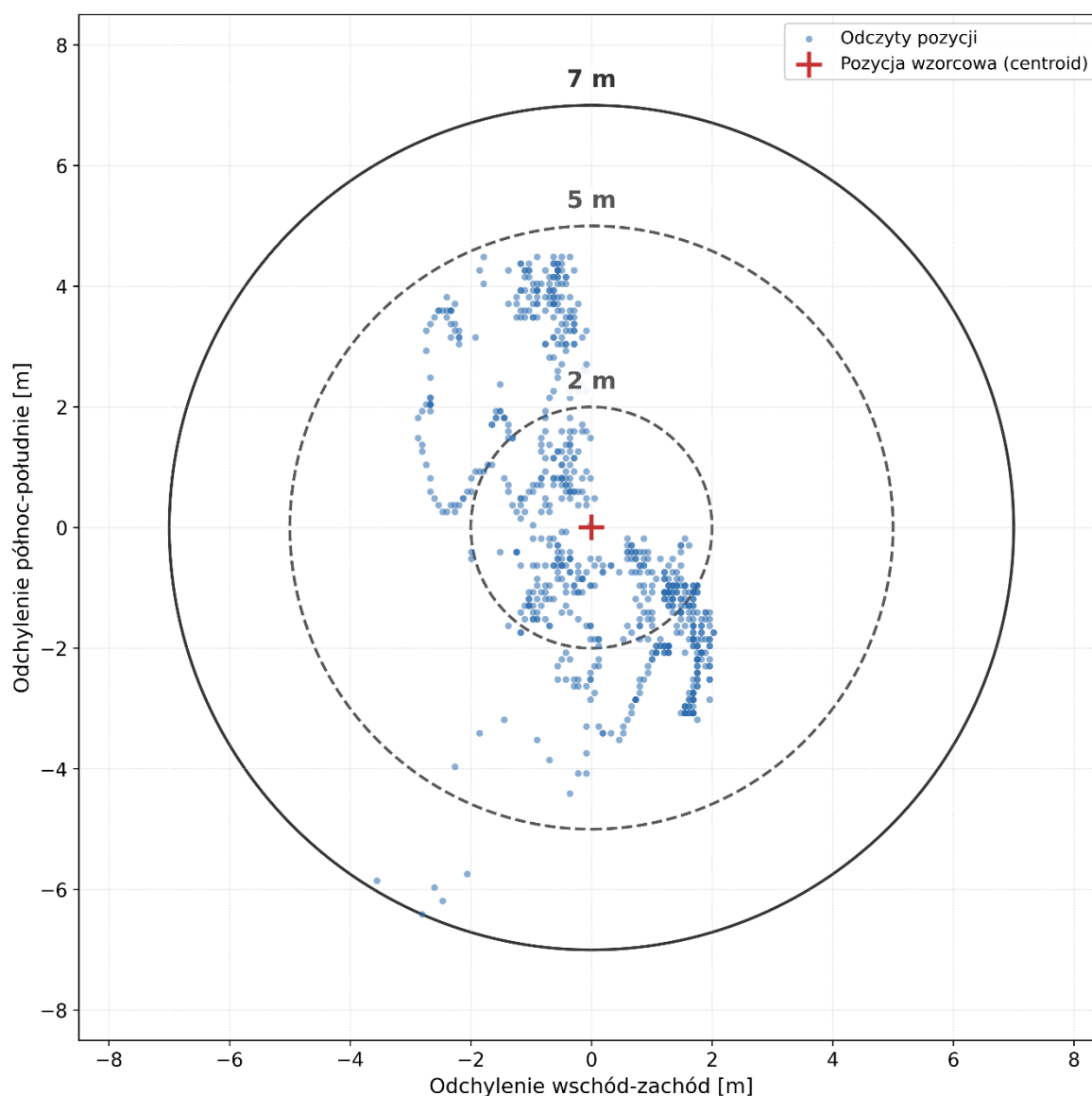
Zarejestrowane wartości TTFF w scenariuszach *warm start* oraz *hot start* mieszczą się w typowych zakresach charakterystycznych dla konsumenckich modułów nawigacji jednoczesnościowej wspierających jednocześnie systemy GPS oraz BeiDou. Krótki czas *hot start* rzędu jednej sekundy pozwala na szybkie odzyskanie funkcjonalności nawigacyjnej po chwilowej utracie sygnału, na przykład podczas przejazdu pod obiektem infrastruktury drogowej bez znaczącej przerwy w rejestracji trasy.

Tabela 18. Czas uzyskania pierwszego fixu pozycji (TTFF) modułu Air530z w warunkach otwartej przestrzeni.

Scenariusz	Warunki	Liczba prób	Mediana TTFF [s]	Zakres TTFF [s]
<i>Cold start</i>	Pierwsze uruchomienie modułu po długiej przerwie, brak aktualnych danych w pamięci	10	38,05	30,20 - 52,20
<i>Warm Start</i>	Po kilkugodzinnej przerwie, dane efemeryd częściowo nieaktualne	10	8,05	6,20 - 10,56

<i>Hot start</i>	Po krótkiej przerwie poniżej minuty, dane efemeryd aktualne	10	1,11	0,80 - 1,57
------------------	--	----	------	-------------

Pomiar dryfu pozycji w stanie statycznym realizowano zgodnie z procedurą opisaną w podrozdziale 7.1.3. Urządzenie z aktywowanym parametrem `rideStartWithoutMotion` pozostawione w stałej pozycji geograficznej rejestrowało odczyty pozycji przez czas trwania pomiaru. Retrospektywna analiza zarejestrowanego logu pozwoliła określić rozkład odchyleń wskazań modułu od pozycji wzorcowej. Maksymalne zaobserwowane odchylenie wyniosło 7 metrów, podczas gdy 99% odczytów mieściło się w promieniu 5 metrów od pozycji wzorcowej. Rozkład odchyleń przedstawiono na Rysunek 46.



Rysunek 46. Dryf pozycji GPS w stanie statycznym ($n = 736$, $t = 60$ min) (źródło: opracowanie własne).

Średnia liczba widzianych satelitów w warunkach otwartej przestrzeni oscylowała w zakresie od 5 do 14, zależnie od bieżącej konstelacji satelitów widocznych z danej lokalizacji oraz warunków atmosferycznych. Wartość ta jest zgodna z oczekiwaniami dla modułu obsługującego dwa niezależne systemy nawigacji satelitarnej, jednoczesne wykorzystanie konstelacji GPS oraz BeiDou zapewnia znacznie większą liczbę potencjalnie dostępnych satelitów w porównaniu do modułów operujących

wyłącznie na systemie GPS, przekładając się na wyższą stabilność fixu pozycji oraz krótsze czasy uzyskania nowego fixu w scenariuszach *warm start* i *hot start*.

Walidację czujnika środowiskowego BME280 przeprowadzono przez porównanie z przyrządem referencyjnym o dokładności 0,1°C, w warunkach stabilizacji termicznej trwającej co najmniej 30 minut. Pomiar obejmował zakres temperatur od 6°C do 29°C, charakterystyczny dla typowych warunków eksploatacji rowerowej w polskich szerokościach geograficznych. Wykonano po 10 pomiarów w temperaturze około 6°C i około 28°C. Uzyskanie wyniki zostały przedstawione w Tabeli 19 i Tabela 20.

Tabela 19. Zestawienie wyników testu układu BME280 w temperaturze około 28°C.

Lp.	Odczyt kontrolny [°C]	Odczyt z WEB AP urządzenia [°C]	Odchylenie [°C]
1	27	27	0
2	27,3	27,2	0,1
3	27,4	27,5	0,1
4	27,3	27,5	0,2
5	27,3	27,5	0,2
6	27,5	27,6	0,1
7	27,6	27,7	0,1
8	27,9	28	0,1
9	28,8	29,1	0,3
10	28,9	29,1	0,2

Tabela 20. Zestawienie wyników testu układu BME280 w temperaturze około 6°C.

Lp.	Odczyt kontrolny [°C]	Odczyt z WEB AP urządzenia [°C]	Odchylenie [°C]
1	5,9	6,1	0,2
2	6,1	6,3	0,2
3	6,1	6,4	0,3
4	6,6	6,5	0,1
5	6,3	6,5	0,2
6	6,3	6,5	0,2
7	6,3	6,6	0,3
8	6,4	6,7	0,3
9	6,5	6,7	0,2
10	6,5	6,7	0,2

Maksymalne odchylenie odczytów temperatury od wzorca wyniosło 0,3°C, mieszcząc się w deklarowanej przez producenta tolerancji $\pm 1,0^\circ\text{C}$.

7.2.4. Niezawodność komunikacji GSM

Walidacja warstwy komunikacyjnej GSM oparta została na danych automatycznie rejestrowanych przez firmware urządzenia w pliku diagnostycznym /Logs/sms_debug.log, generowanym podczas każdej procedury wysyłki wiadomości SMS. Plik zawiera szczegółowy zapis sekwencji komend AT wymienianych z modemem SIM800L wraz z znacznikami czasowymi względnymi, umożliwiając precyzyjną analizę czasu trwania poszczególnych etapów procedury alarmowej oraz identyfikację ewentualnych błędów komunikacyjnych.

Przeprowadzono łącznie 20 prób wysyłki wiadomości SMS w dwóch klasach środowisk różniących się charakterystyką pokrycia siecią komórkową. Po dziesięć prób w terenach leśnych oraz w obszarach miejskich. Podział ten pozwolił na oddzielne scharakteryzowanie zachowania systemu w warunkach niskiego pokrycia sygnałem radiowym (tereny leśne, zależność od widoczności nieba) oraz w warunkach silnego, lecz obciążonego pokrycia (tereny miejskie, natłok abonentów). Statystyczne wyniki obu grup testowych zestawiono w Tabeli 21.

Tabela 21. Niezawodność komunikacji GSM.

Parametr	Tereny leśne	Tereny miejskie	Łącznie
Liczba przeprowadzonych prób	10	10	20
Sukces przy pierwszej próbie	7 / 10	9 / 10	16 / 20
Liczba prób wymagających retry	3	1	4
Sukces po zadziałaniu mechanizmu retry	2 / 3	1 / 1	3 / 4
Trwałe niepowodzenie po retry	1 / 3	0 / 1	1 / 4
Skuteczność pierwszej próby	70%	90%	80%
Skuteczność łączna dostarczenia	90%	100%	95%
Średni czas wysyłki	38 s	38 s	38 s

Analiza poszczególnych etapów procedury wysyłki SMS pozwoliła zidentyfikować dwie dominujące składowe czasu trwania komunikacji. Pierwszą stanowi czas inicjalizacji modemu od momentu uruchomienia przetwornicy zasilającej (sygnał GP0=HIGH) do uzyskania gotowości modułu do obsługi komend AT. Wartość ta jest względnie stabilna, oscylując w granicach 3-5 sekund, niezależnie od warunków zasięgu. Drugą, zmienną i dominującą w warunkach marginalnego zasięgu, stanowi czas oczekiwania na rejestrację modemu w sieci (komenda AT+CREG? z pollingiem), wynoszący od kilku sekund w obszarach silnego sygnału do granicznej wartości 60 sekund, zgodnej z przyjętym w warstwie firmware limitem czasowym COMM_GSM_NETWORK_WAIT_MS.

W obszarach leśnych zaobserwowano trzy przypadki niepowodzenia pierwszej próby dostarczenia, wszystkie powiązane z ograniczoną widocznością nieba oraz cieniowaniem sygnału radiowego przez korony drzew. Skuteczność procesu komunikacji wykazywała wyraźną zależność od lokalizacji pomiaru. Próby wykonywane na otwartych polanach kończyły się pomyślnie w krótkim czasie, natomiast pomiary realizowane w gęstym zadrzewieniu wymagały wydłużonego oczekiwania na rejestrację modemu w sieci lub kończyły się przekroczeniem 60-sekundowego limitu. W dwóch z trzech niepowodzeń zadziałał mechanizm retransmisji opisany w podrozdziale 6.4.5. Kolejna próba po 15-sekundowym czasie odnowienia zakończyła się dostarczeniem wiadomości. Trzeci przypadek zakończył się trwałym niepowodzeniem również po ponowieniu, wskazując na sytuację utrzymującej się utraty pokrycia w danej lokalizacji, wykraczającej poza możliwości kompensacyjne prostego mechanizmu ponawiania.

W obszarach miejskich odnotowano pojedynczy przypadek niepowodzenia pierwszej próby dostarczenia. Mimo obserwowanego silnego sygnału radiowego stacji bazowej modem nie zdołał zakończyć procedury transmisji za pierwszym razem. Retrospektywna analiza pliku diagnostycznego

sms_debug.log wykazała, że przyczyną było chwilowe załamanie napięcia zasilającego modem SIM800L (zjawisko *brownout*), modem podczas próby rejestracji w silnie obciążonej sieci komórkowej podnosi moc wyjściową nadajnika radiowego, aby przebić się przez natłok równoczesnych transmisji innych abonentów, generując szczytowy pobór prądu przekraczający chwilową wydajność źródła zasilania. Spadek napięcia poniżej wartości progowej spowodował niekontrolowany restart modułu. Mechanizm ponawiania po upływie czasu odnowienia podjął ponowną próbę, która tym razem zakończyła się sukcesem, prawdopodobnie ze względu na chwilowe rozładowanie natłoku sieciowego.

Zestawienie wyników w obu środowiskach ujawnia dwie odrębne klasy zjawisk ograniczających niezawodność komunikacji:

- **Pierwszą klasę** stanowią zdarzenia sieciowe - czasowe odrzucenia rejestracji oraz chwilowe pogorszenie sygnału, kompensowane przez zaprojektowany mechanizm ponawiania próby;
- **Drugą klasę** stanowią ograniczenia warstwy sprzętowej, obserwowane *brownouty* zasilania, wskazujące na potrzebę rewizji układu dystrybucji zasilania modemu w kolejnej iteracji prototypu.

Skuteczność samego mechanizmu ponawiania okazała się przy tym wysoka: spośród czterech prób, które nie powiodły się za pierwszym razem, retransmisja pozwoliła odzyskać trzy (75%), podnosząc łączną skuteczność dostarczenia powiadomienia z 80% przy pierwszej próbie do 95%. Warto zaznaczyć, że jedyne trwałe niepowodzenie odnotowano w terenie leśnym, co potwierdza, że mechanizm ponawiania skutecznie kompensuje przejściowe zdarzenia sieciowe, pozostaje natomiast bezradny wobec utrzymującej się utraty pokrycia.

Kierunki potencjalnych modyfikacji obejmują wprowadzenie dodatkowej pojemności filtrującej blisko wejścia zasilania modemu, przegląd charakterystyk obciążalności impulsowej przetwornicy TPS63020 oraz rozszerzenie logiki mechanizmu ponawiania próby o automatyczną reinicjalizację modemu po wykryciu jego resetowania. Szczegółowa dyskusja proponowanych modyfikacji zaprezentowana została w rozdziale 8. Ostateczna skuteczność dostarczania powiadomień alarmowych na poziomie 95% w warunkach testowych stanowi solidny punkt wyjścia dla dalszej optymalizacji zarówno warstwy firmware, jak i warstwy sprzętowej urządzenia.

7.3. Walidacja algorytmu detekcji

Algorytm detekcji upadku, opisany szczegółowo w podrozdziale 6.3, stanowi centralny element funkcjonalny prototypu, decyzja o uruchomieniu procedury alarmowej opiera się bezpośrednio na jego klasyfikacji zdarzeń zarejestrowanych przez czujnik inercyjny MPU-6500. Ze względu na krytyczne znaczenie tego komponentu dla bezpieczeństwa użytkownika, walidacja algorytmu przeprowadzona została w dwóch komplementarnych etapach. Kontrolowanych testach stanowiskowych obejmujących cztery zdefiniowane scenariusze klasyfikacyjne oraz obserwacji zachowania algorytmu w warunkach rzeczywistej eksploatacji podczas testów terenowych. Wszystkie procedury walidacyjne przeprowadzono w trybie Trening 2, w którym wykryty upadek generuje wyłącznie znacznik diagnostyczny w pliku rejestratora wstrząsów, bez inicjowania procedury wysyłki wiadomości SMS.

7.3.1. Scenariusze testowe

W pierwszym etapie walidacji zdefiniowano cztery reprezentatywne scenariusze pokrywające dwie klasy zdarzeń - **wzorce pozytywne**, w których algorytm powinien zaklasyfikować zdarzenie jako potwierdzony upadek, oraz **wzorce negatywne**, w których zarejestrowany impuls przyspieszenia powinien zostać zaklasyfikowany jako fałszywy alarm w fazie obserwacji. Wybór scenariuszy podyktowany był potrzebą pokrycia typowych wzorców inercyjnych występujących w codziennej eksploatacji rowerowej.

Wzorce pozytywne - zdarzenia klasyfikowane jako upadek:

- **Upadek statyczny** - symulacja upadku roweru w miejscu, realizowana przez kontrolowane przewrócenie konstrukcji nieruchomego roweru zamontowanego w pozycji wyjściowej. Test

odtwarza scenariusz utraty stateczności podczas postoju (np. przy sygnalizacji świetlnej lub podczas przygotowania do jazdy);

- **Upadek w trakcie powolnej jazdy** - symulacja utraty równowagi podczas ruchu z prędkością nieprzekraczającą 5 km/h. Test odtwarza scenariusz upadku podczas manewrowania na małej prędkości (np. w tłoku, na wąskiej ścieżce).

Wzorce negatywne - zdarzenia klasyfikowane jako fałszywe alarmy:

- **Najazd na krawężnik** - kontrolowane pokonanie przeszkody terenowej o wysokości typowej dla krawężnika ulicznego. Test odtwarza typowe zdarzenie generujące impuls przyspieszenia porównywalny z upadkiem, ale bez utraty ciągłości ruchu;
- **Ręczne trzęsienie urządzeniem** - celowe wprowadzenie urządzenia w intensywne drgania rękoma, imitujące zdarzenia serwisowe (np. transport urządzenia, montaż na ramie roweru). Test weryfikuje odporność algorytmu na krótkotrwałe impulsy nie związane z rzeczywistym zdarzeniem drogowym.

7.3.2. Wyniki testów kontrolnych

Wyniki testów stanowiskowych zestawiono w Tabeli 22, obejmującej łącznie 10 prób podzielonych między cztery scenariusze. Dla każdego scenariusza rejestrowano liczbę wykonanych prób, liczbę zdarzeń zaklasyfikowanych przez algorytm jako potwierdzony upadek oraz oczekiwaną klasyfikację referencyjną.

Tabela 22. Wyniki walidacji algorytmu detekcji upadku w scenariuszach kontrolowanych

Scenariusz	Oczekiwana klasyfikacja	Liczba prób	Klasyfikacja pozytywna (fall = 1)	Zgodność z oczekiwaniem
Upadek statyczny	Pozytywny	10	8	80 %
Upadek w powolnej jeździe	Pozytywny	10	6	60 %
Najazd na krawężnik	Negatywny	10	1	90 %
Ręczne trzęsienie urządzeniem	Negatywny	10	3	70 %

Zestawienie wyników wskazuje na skuteczne odrzucanie zakłóceń typu najazd na krawężnik (90% poprawnych klasyfikacji) oraz zadowalającą wykrywalność upadków statycznych (80%). Zagregowana czułość algorytmu, wyznaczona jako stosunek poprawnie wykrytych upadków do wszystkich prób pozytywnych, wyniosła 70% (14 z 20 prób), natomiast swoistość, określająca zdolność do odrzucania zdarzeń niebędących upadkiem, osiągnęła 80% (16 z 20 prób).

Najniższą skuteczność zaobserwowano w scenariuszu upadku w powolnej jeździe (60%), co wynika z niższej wartości szczytowej przyspieszenia generowanego podczas takiego zdarzenia - łagodniejszy przebieg sygnału z czujnika inercyjnego łatwiej pozostaje poniżej progu detekcji przyjętego przez algorytm. Zaobserwowana rozbieżność stanowiła punkt wyjścia do dostrojenia parametrów progowych algorytmu, opisanego w podrozdziale 7.3.3.

7.3.3. Dostrajanie parametrów algorytmu

Wartości progowe algorytmu detekcji upadku zdefiniowane w pliku konfiguracyjnym `shock.json`, opisane w podrozdziale 6.3, podlegały iteracyjnemu dostrajaniu w toku testów walidacyjnych. Kluczowy wpływ na skuteczność klasyfikacji miały cztery parametry: próg wykrycia uderzenia `impactThresholdG`, próg powrotu ruchu `motionResumedThresholdG`, długość fazy obserwacji `observationWindowS` oraz progowa prędkość `GPS`.

gpsLowSpeedThresholdKmh służąca dodatkowej weryfikacji faktu zatrzymania pojazdu po wykryciu uderzenia.

Istotną rolę w procesie dostrajania odegrały dane zgromadzone podczas przejazdów terenowych, opisanych w podrozdziale 7.3.4. Obserwacja rzeczywistych przebiegów sygnału inercyjnego pozwoliła skorygować wartości progowe algorytmu, dążąc do zachowania równowagi pomiędzy dwiema przeciwstawnymi charakterystykami: czułością, rozumianą jako zdolność do wykrycia rzeczywistego upadku, oraz swoistością, rozumianą jako odporność na fałszywe alarmy. Zestawienie wartości domyślnych oraz dostrojonych przedstawiono w Tabeli 23

Tabela 23. Końcowe wartości parametrów algorytmu detekcji upadku wykorzystane w testach walidacyjnych.

Parametr	Wartość domyślna	Wartość dostrojona	Znaczenie
impactThresholdG	3,0 g	8,0 g	Próg amplitudy przyspieszenia inicjujący fazę obserwacji
motionResumedThresholdG	0,4 g	0,8 g	Próg wznowienia ruchu kończy fazę obserwacji jako fałszywy alarm
observationWindowS	5 s	7 s	Czas trwania fazy obserwacji po wykrytym uderzeniu
gpsLowSpeedThresholdKmh	1,0 km/h	4 km/h	Progowa prędkość GPS potwierdzając faktyczne zatrzymanie

Wartość progu detekcji uderzenia (impactThresholdG) podniesiono z 3,0 g do 8,0 g na podstawie obserwacji przebiegów magnitudy przyspieszenia zarejestrowanych podczas przejazdów terenowych. Pierwotna wartość okazała się zbyt niska, była regularnie przekraczana przez zwykłe wstrząsy generowane podczas normalnej jazdy po nierównej nawierzchni, co w warunkach rzeczywistych skutkowało licznymi fałszywymi alarmami. Podniesienie progu do 8,0 g pozwoliło umieścić go powyżej poziomu drgań charakterystycznych dla typowej jazdy, przy jednoczesnym zachowaniu wrażliwości na rzeczywiste, silne uderzenia towarzyszące upadkowi.

Analogicznie, wartość progu wznowienia ruchu (motionResumedThresholdG), decydującego o uznaniu, że urządzenie porusza się dalej po zarejestrowanym uderzeniu, zwiększono z 0,4 g do 0,8 g na podstawie danych z jazdy terenowych. Wyższa wartość ogranicza ryzyko błędnej interpretacji resztkowych drgań jako kontynuacji ruchu, czyniąc drugi stopień weryfikacji bardziej odpornym na zakłócenia.

Okno obserwacji (observationWindowS), określające czas analizy sygnału następujący po wykryciu uderzenia, wydłużono z 5 do 7 sekund, aby zapewnić wystarczający margines czasowy na wiarygodne stwierdzenie, czy po zdarzeniu następuje wznowienie ruchu, czy też trwały bezruch wskazujący na rzeczywisty upadek.

Wartość progu niskiej prędkości GPS (gpsLowSpeedThresholdKmh) podniesiono z 1,0 km/h do 4 km/h ze względu na okresowo występujące zjawisko dryfu pozycji GNSS. Pierwotna, niska wartość progu powodowała, że pozorny ruch wynikający z dryfu wskazań modułu w stanie statycznym mógł być błędnie interpretowany jako rzeczywiste przemieszczanie się urządzenia. Podniesienie progu do 4 km/h skutecznie odfiltrowuje te artefakty, nie wpływając przy tym na detekcję rzeczywistej jazdy, której prędkość znacząco przekracza tę wartość.

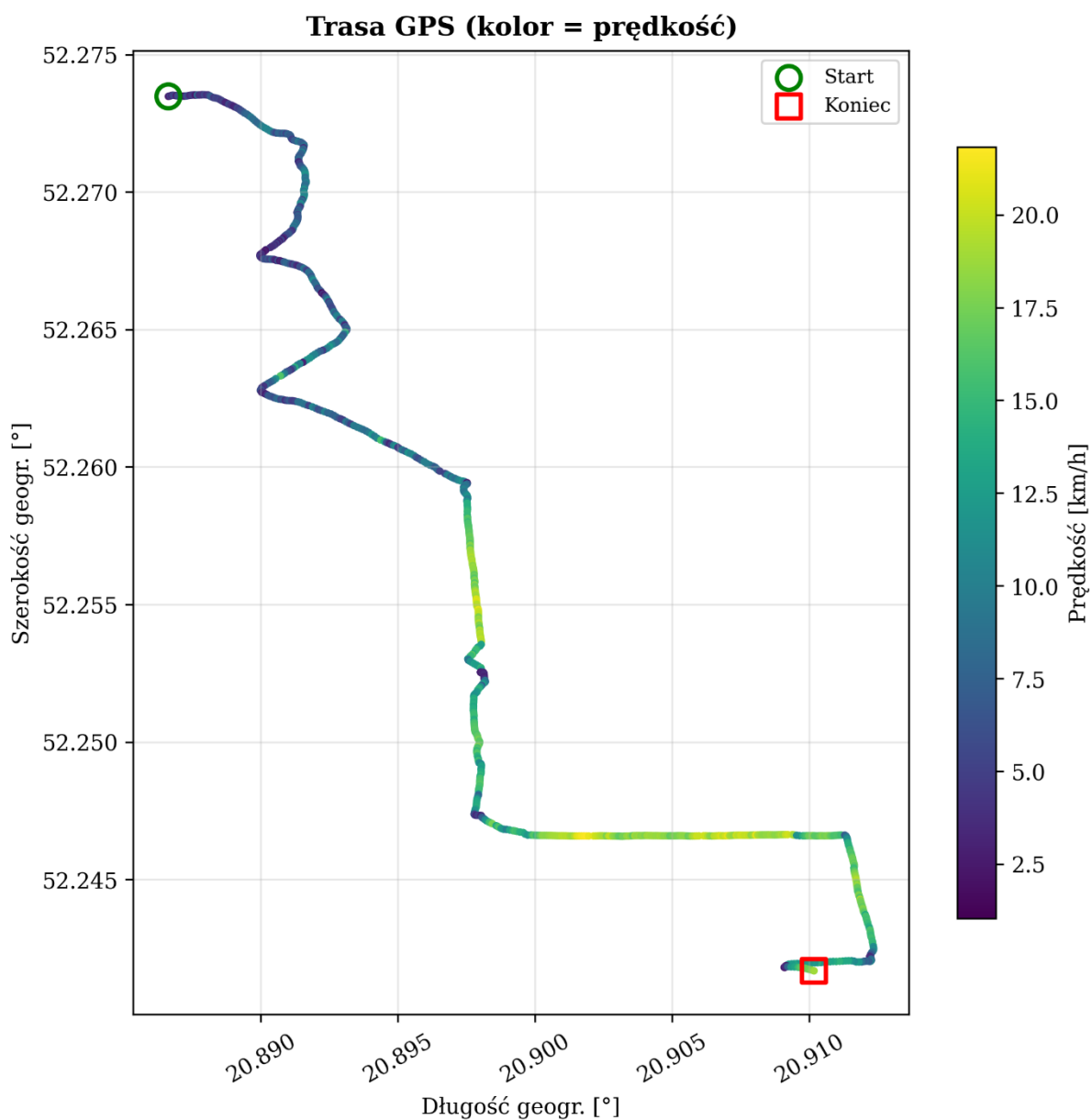
Ostatecznie przyjęte wartości parametrów stanowią kompromis wypracowany na podstawie danych z obu etapów walidacji, uwzględniający zarówno powtarzalność testów stanowiskowych, jak i realizm warunków terenowych.

7.3.4. Obserwacje z testów terenowych

Drugi etap walidacji algorytmu obejmował obserwację jego zachowania podczas testów terenowych w warunkach rzeczywistej eksploatacji rowerowej, bez celowego prowokowania zdarzeń krytycznych. Rolą tego etapu była weryfikacja odporności algorytmu na naturalne perturbacje sygnału inercyjnego generowane podczas typowej jazdy, najazdy na nierówności nawierzchni, gwałtowne hamowania, wstrząsy przenoszone z ramy podczas jazdy po nawierzchni kostkowej oraz manewry ruchu drogowego. Wszystkie sesje terenowe przeprowadzono w trybie Trening 2, umożliwiając wielokrotną weryfikację bez wysyłki niepotrzebnych wiadomości SMS.

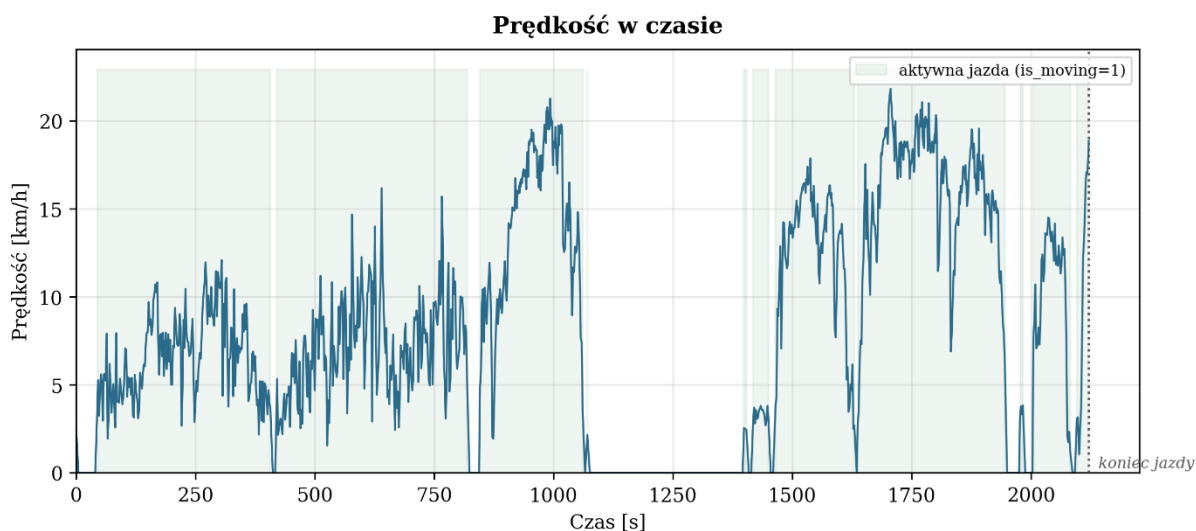
W ramach tego etapu zarejestrowano szereg przejazdów o zróżnicowanym charakterze nawierzchni. Na Rysunek 47, Rysunek 48, Rysunek 49, Rysunek 50 i Rysunek 51 przedstawiono dane z reprezentatywnej sesji terenowej o długości około 5,2 km, której trasa łączyła odcinki ścieżek rowerowych, chodników oraz nieutwardzonych drózek leśnych, obejmując tym samym pełne spektrum perturbacji sygnału inercyjnego wymienionych powyżej. Podczas przejazdu nie doszło do żadnego rzeczywistego zdarzenia krytycznego, a algorytm pracujący w trybie Trening 2 nie zgłosił alarmu, co czyni tę sesję reprezentatywnym przykładem weryfikacji odporności na fałszywe alarmy.

Zarejestrowaną trasę przejazdu, z naniesioną barwnie prędkością chwilową, przedstawiono na Rysunek 47. Przebieg trasy jest ciągły i pozbawiony wyraźnych artefaktów pozycjonowania, co potwierdza poprawność akwizycji danych GNSS w ruchu.

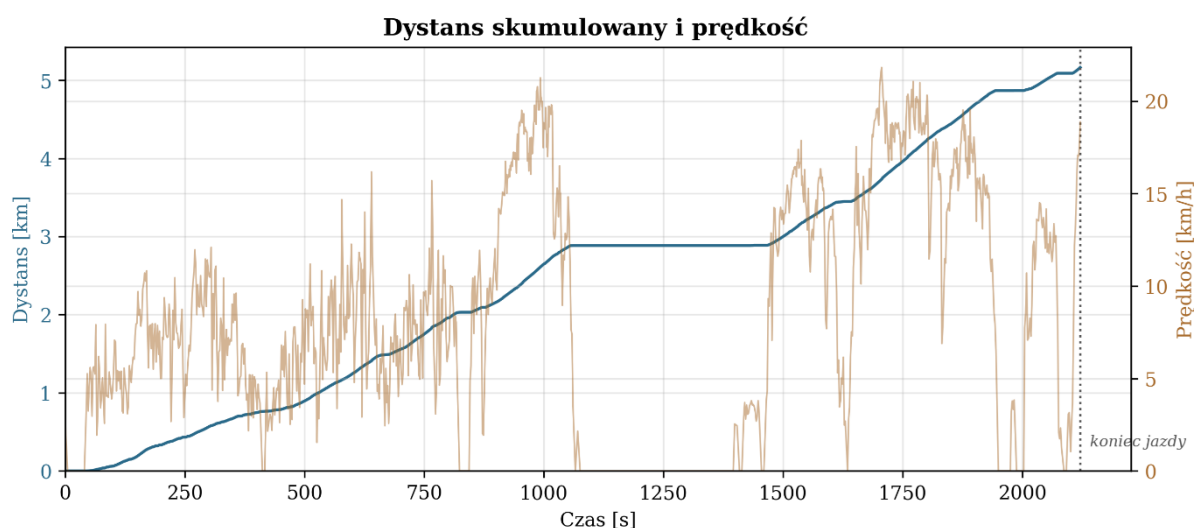


Rysunek 47. Trasa przejazdu testowego zarejestrowana przez moduł GNSS, z prędkością chwilową odwzorowaną kolorem (źródło: opracowanie własne).

Odpowiadający jej profil prędkości w czasie (Rysunek 48) oraz wykres dystansu skumulowanego (Rysunek 49) ukazują charakterystyczną dla jazdy miejsko-rekreacyjnej zmienność tempa naprzemienne odcinki jazdy z prędkością do około 21 km/h oraz postoje.



Rysunek 48. Profil prędkości chwilowej w funkcji czasu podczas przejazdu testowego. Obszarami zacienionymi zaznaczono okresy aktywnej jazdy ($is_moving = 1$) (źródło: opracowanie własne).

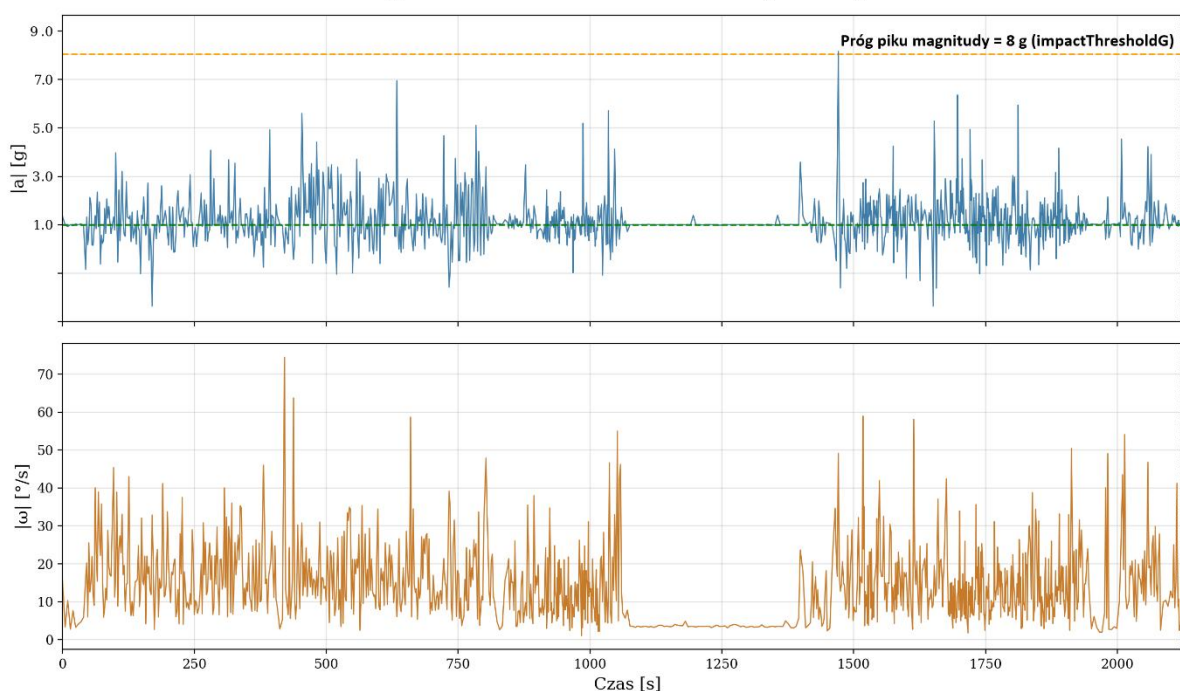


Rysunek 49. Dystans skumulowany oraz prędkość chwilowa w funkcji czasu. Płaski odcinek przebiegu dystansu odpowiada postojowi, podczas którego nie następował przyrost drogi.

Widoczne jest przy tym działanie mechanizmu detekcji ruchu (parametr is_moving): dłuższy postój w okolicach 1100-1400 s został poprawnie sklasyfikowany jako brak aktywnej jazdy, co znajduje odzwierciedlenie w płaskim odcinku wykresu dystansu skumulowanego (brak przyrostu drogi mimo upływu czasu).

Kluczowe znaczenie dla oceny odporności algorytmu ma przebieg magnitudy przyspieszenia oraz prędkości kątowej (Rysunek 50). Naturalne perturbacje sygnału inercyjnego, generowane przez przejazdy przez krawężniki, nierówności drózek leśnych oraz szczeliny chodnikowe, wygenerowały liczne, wyraźne piki przyspieszenia, z których część osiągała wartości rzędu 5-7 g. W okolicach 1470 s odnotowano pojedynczy pik osiągający próg detekcji uderzenia ($impactThresholdG = 8$ g), najprawdopodobniej odpowiadający dynamicznemu przejazdowi przez krawężnik.

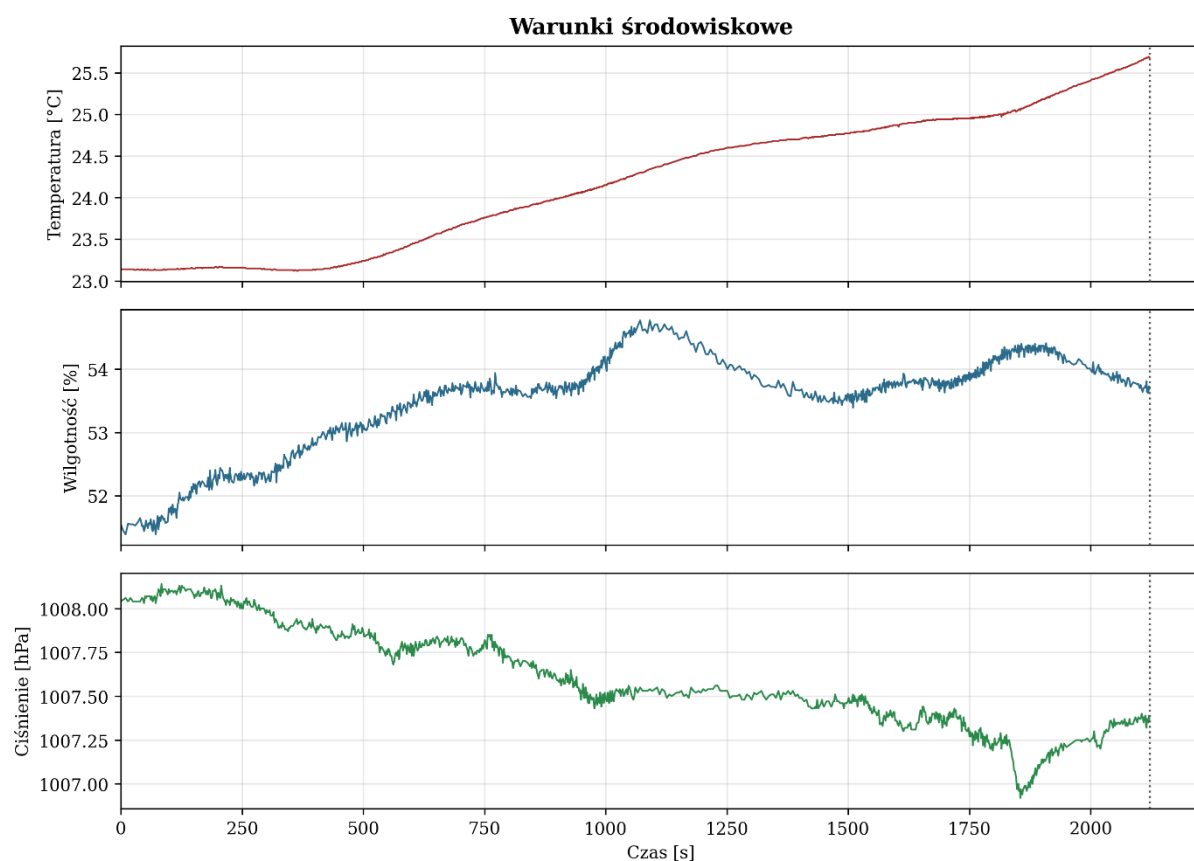
Magnituda akcelometru i żyroskopu



Rysunek 50. Przebieg magnitudy przyspieszenia oraz prędkości kątowej podczas przejazdu testowego. Linia przerywaną oznaczono próg detekcji uderzenia ($\text{impactThresholdG} = 8 \text{ g}$) (źródło: opracowanie własne).

Pomimo przekroczenia progu przyspieszenia algorytm nie zgłosił alarmu, na dalszym etapie obserwacji, następującym bezpośrednio po piku, wykrył kontynuację ruchu urządzenia, na podstawie czego stwierdził, że rower porusza się dalej, a zarejestrowane zdarzenie nie stanowiło poważnego upadku. Zachowanie to potwierdza poprawność trójfazowa logiki detekcji, w której samo przekroczenie progu przyspieszenia stanowi warunek konieczny, lecz niewystarczający do wygenerowania alarmu. Dopiero brak dalszego ruchu po uderzeniu jednoznacznie wskazywałby na rzeczywisty upadek. Właśnie ten mechanizm decyduje o odporności algorytmu na najsilniejsze nawet, lecz krótkotrwałe wstrząsy przenoszone z ramy podczas jazdy po nierównej nawierzchni.

Rejestrowane równoległe parametry środowiskowe (Rysunek 51) wykazały łagodny, monotoniczny wzrost temperatury (z około $23,1^{\circ}\text{C}$ do $25,7^{\circ}\text{C}$) oraz odpowiadające mu zmiany wilgotności względnej i ciśnienia atmosferycznego, zgodne z naturalną zmiennością warunków w trakcie trwania przejazdu. Potwierdza to stabilność akwizycji danych z czujnika środowiskowego w warunkach polowych.



Rysunek 51. Przebieg parametrów środowiskowych (temperatura, wilgotność względna, ciśnienie atmosferyczne) zarejestrowanych przez czujnik BME280 w trakcie przejazdu testowego (źródło: opracowanie własne)

Zaobserwowana podczas testów terenowych zdolność algorytmu do odróżnienia silnego, lecz krótkotrwałego wstrząsu od rzeczywistego upadku nawet w przypadku przekroczenia progu przyspieszenia stanowi praktyczne potwierdzenie odporności przyjętego rozwiązania na fałszywe alarmy w warunkach rzeczywistej eksploatacji.

8. Analiza rozwiązania i kierunki rozwoju

Niniejszy rozdział zawiera krytyczne omówienie zrealizowanego rozwiązania. Przedstawiono w nim trudności napotkane podczas realizacji pracy, zalety i wady przyjętych rozwiązań projektowych, potencjalne obszary zastosowania systemu, a także planowane kierunki dalszego rozwoju wraz z pracami, które należałoby wykonać w przyszłości.

8.1. Trudności napotkane podczas realizacji

Realizacja projektu, łączącego projektowanie sprzętowe, programowanie systemów wbudowanych oraz przetwarzanie danych, wiązała się z szeregiem wyzwań o charakterze zarówno konstrukcyjnym, jak i programistycznym.

Najistotniejsze trudności ujawniły się już na etapie projektowania płytki drukowanej. W pierwotnej wersji projektu nie uwzględniono tranzystorów MOSFET odcinających zasilanie linii 3,3 V od peryferiów, z wyjątkiem cyfrowego czujnika wstrząsów, który musiał pozostać aktywny w trybie niskiego poboru energii. Brak możliwości selektywnego odłączenia pozostałych układów od zasilania ograniczał efektywność zarządzania energią w stanach uśpienia urządzenia i stanowił istotne niedopatrzenie projektowe, które ujawniło się dopiero na etapie testów sprzętowych.

Kolejnym wyzwaniem konstrukcyjnym była ograniczona liczba wyprowadzeń mikrokontrolera ESP32-C6. Pomimo znacznej mocy obliczeniowej i bogatego zestawu peryferiów, niewielka liczba dostępnych pinów wymusiła staranne planowanie mapowania sygnałów oraz wykorzystanie ekspandera wyprowadzeń, co istotnie skomplikowało proces projektowania schematu i wymusiło szereg kompromisów w rozmieszczeniu funkcji na dostępnych liniach.

W obszarze zasilania poważną trudność stanowiła niestabilność toru zasilającego podczas transmisji danych przez moduł GSM. Krótkotrwałe, lecz wysokoprądowe impulsy poboru energii towarzyszące nadawaniu powodowały zapady napięcia oraz niestabilną pracę przetwornicy, co wymagało analizy przebiegów oscyloskopowych oraz korekty układu zasilania w celu zapewnienia stabilnej pracy urządzenia w warunkach obciążenia szczytowego.

Odrębnym wyzwaniem, tym razem o charakterze programistycznym, było dostrojenie algorytmu detekcji upadku. Parametry progowe wyznaczone na podstawie testów stanowiskowych w warunkach statycznych okazały się nieadekwatne do rzeczywistego charakteru sygnału inercyjnego rejestrowanego podczas jazdy. Dopiero dane zgromadzone w trakcie przejazdów terenowych umożliwiły właściwe skalibrowanie algorytmu, uwypuklając zasadniczą różnicę pomiędzy powtarzalnymi warunkami laboratoryjnymi a złożonością rzeczywistej eksploatacji.

8.2. Zalety i wady przyjętego rozwiązania

Do najważniejszych zalet zrealizowanego systemu należy jego kompleksowość. Urządzenie integruje w sobie akwizycję danych z wielu niezależnych czujników, lokalny zapis oraz interfejs konfiguracyjny dostępny przez sieć. Zastosowanie modułu GNSS obsługującego jednocześnie systemy GPS oraz BeiDou zapewnia wysoką dostępność i stabilność wskazań pozycji. Trójfazowa logika detekcji upadku, weryfikująca kontynuację ruchu po wykryciu uderzenia, zapewnia wysoką odporność na fałszywe alarmy, co potwierdziły testy terenowe. Oparcie oprogramowania na autorskich sterownikach kluczowych układów dało pełną kontrolę nad ich konfiguracją, a możliwość dostrajania parametrów algorytmu bez rekompilacji firmware znacząco usprawniła proces walidacji.

Istotną zaletą przyjętej architektury jest również jej pełna autonomiczność, system działa niezależnie od zewnętrznej infrastruktury serwerowej czy aplikacji mobilnej, zapisując dane lokalnie na karcie microSD. Takie podejście eliminuje zależność od łączności z Internetem oraz od ciągłości działania usług zewnętrznych, upraszcza ochronę prywatności danych użytkownika, które nie opuszczają urządzenia i czyni rozwiązanie odpornym na awarie po stronie zaplecza. W podobnym duchu

zaprojektowano fizyczny interfejs użytkownika, ograniczony do dwóch przycisków oraz dwóch diod sygnalizacyjnych. Jego prostota przekłada się na intuicyjną, bezobsługową eksploatację oraz niski pobór energii, choć jednocześnie oferuje użytkownikowi ograniczony zakres interakcji i informacji zwrotnej podczas jazdy.

Rozwiązanie nie jest jednak pozbawione wad. Wspomniane niedopatrzenia na etapie projektu płytki drukowanej ograniczają obecnie efektywność zarządzania energią. Skuteczność algorytmu detekcji upadku, choć zadowalająca, pozostawia margines do poprawy, szczególnie w scenariuszu upadku o łagodnym przebiegu, gdzie czułość była najniższa.

8.3. Potencjalne zastosowania

Zrealizowany system, w swoim podstawowym przeznaczeniu, adresowany jest do rowerzystów rekreacyjnych, umożliwiając rejestrację parametrów przejazdu oraz zwiększając bezpieczeństwo dzięki funkcji detekcji upadku. Naturalnym rozszerzeniem tego zastosowania jest obszar sportowy, w którym gromadzone dane o prędkości, dystansie i charakterystyce trasy mogą służyć analizie treningu.

Funkcja automatycznego powiadamiania o upadku czyni system szczególnie wartościowym dla grup użytkowników o podwyższonym ryzyku lub potrzebie zapewnienia dodatkowego bezpieczeństwa, czyli osób starszych, pozostających aktywnymi fizycznie, oraz dzieci, w przypadku których możliwość szybkiego powiadomienia opiekuna o zdarzeniu ma istotne znaczenie.

Odrębnym, obiecującym kierunkiem jest zastosowanie komercyjne, w systemach współdzielonych rowerów oraz hulajnóg elektrycznych. W tego typu flotach urządzenie mogłoby pełnić rolę rejestratora telemetrycznego oraz elementu systemu bezpieczeństwa, dostarczając operatorowi danych o lokalizacji, sposobie użytkowania pojazdu oraz potencjalnych zdarzeniach drogowych.

8.4. Kierunki dalszego rozwoju

Wskazane w niniejszym rozdziale ograniczenia wyznaczają zarazem naturalne kierunki dalszego rozwoju systemu. W warstwie sprzętowej priorytetem pozostaje korekta projektu płytki drukowanej, obejmująca uzupełnienie układów odcinających zasilanie peryferiów, co umożliwiłoby pełniejsze wykorzystanie trybów oszczędzania energii mikrokontrolera i wydłużenie czasu autonomicznej pracy urządzenia.

Odrębnym kierunkiem rozwoju warstwy sprzętowej jest miniaturyzacja urządzenia poprzez odejście od obecnej, modularnej konstrukcji na rzecz pojedynczej, w pełni zintegrowanej płytki drukowanej. W obecnym prototypie część funkcjonalności realizowana jest przy użyciu gotowych modułów zewnętrznych, co uprościło etap projektowania i uruchamiania, lecz odbywa się kosztem zwiększonych wymiarów oraz mniej efektywnego wykorzystania powierzchni płytki. Zaprojektowanie wszystkich podzespołów w postaci komponentów montowanych bezpośrednio na wspólnej płycie pozwoliłoby na znaczące zmniejszenie gabarytów urządzenia, redukcję liczby połączeń międzymodułowych, a tym samym poprawę niezawodności oraz na pełniejszą optymalizację rozprowadzenia zasilania i sygnałów, trudną do osiągnięcia w konstrukcji modularnej.

Rozwinięciem obecnego, celowo minimalistycznego interfejsu użytkownika mogłoby być wyposażenie urządzenia w niskoprądowy wyświetlacz, na przykład typu Sharp Memory LCD, charakteryzujący się skrajnie niskim poborem energii przy zachowaniu czytelności w pełnym oświetleniu słonecznym. Wraz z dodatkowymi przyciskami obsługi wyświetlacz taki stanowiłby uzupełniający interfejs, umożliwiający podgląd bieżących parametrów przejazdu bez konieczności korzystania z interfejsu webowego, przy zachowaniu niskiego zużycia energii charakterystycznego dla obecnego rozwiązania.

Kolejnym kierunkiem rozwoju konstrukcji jest uelastycznienie sposobu montażu urządzenia. Obecnie urządzenie stanowi jedną, zintegrowaną całość, co ogranicza swobodę jego umiejscowienia na rowerze. Rozdzielenie konstrukcji na moduł główny montowany w dowolnym, dyskretnym miejscu, na

przykład pod siodełkiem oraz odłączany panel interfejsu, mocowany na kierownicy, zapewniłoby użytkownikowi łatwy dostęp i podgląd danych w trakcie jazdy przy jednoczesnym zachowaniu swobody rozmieszczenia właściwego urządzenia.

W warstwie programowej dalszy rozwój mógłby objąć rozbudowę algorytmu detekcji zdarzeń o automatyczną klasyfikację różnych typów sytuacji krytycznych na podstawie danych z czujnika inercyjnego, a także zastosowanie metod uczenia maszynowego w celu poprawy czułości detekcji przy zachowaniu odporności na fałszywe alarmy.

Warto podkreślić, że autonomiczność urządzenia, rozumiana jako niezależność od zewnętrznej infrastruktury, stanowi świadomie przyjęte założenie projektowe, a nie ograniczenie do usunięcia. Ewentualna integracja z usługą chmurową lub aplikacją mobilną nie powinna zatem zastępować lokalnego trybu pracy, lecz stanowić jego opcjonalne rozszerzenie, aktywowane wyłącznie w tych zastosowaniach, w których zdalny dostęp do danych jest istotny, na przykład w scenariuszach flotowych lub w rozwiązaniach ukierunkowanych na bezpieczeństwo, gdzie natychmiastowe powiadomienie o zdarzeniu ma znaczenie krytyczne. W takim ujęciu kierunkiem rozwoju pozostaje nie tyle samo dodanie łączności, co zaprojektowanie jej w sposób zachowujący pełną funkcjonalność urządzenia również w trybie offline.

9. Podsumowanie

Głównym celem niniejszej pracy było opracowanie koncepcji oraz implementacja autonomicznego systemu rejestracji i analizy zdarzeń dynamicznych dla pojazdów lekkich, na przykładzie rowerowej czarnej skrzynki. Cel ten został osiągnięty, a postawiona teza o możliwości zapewnienia zaawansowanych funkcji bezpieczeństwa rowerzysty, w szczególności automatycznego alarmowania ratunkowego, przy zachowaniu pełnej autonomii sprzętowej i niezależności od infrastruktury smartfonowej znalazła potwierdzenie w działającym prototypie.

Zasadniczym rezultatem pracy jest wykazanie, że kompletny łańcuch funkcji bezpieczeństwa mikromobilności od akwizycji danych, przez detekcję zdarzenia krytycznego, po samodzielne powiadomienie ratunkowe może zostać zrealizowany w obrębie jednego, niezależnego urządzenia wbudowanego, bez pośrednictwa telefonu użytkownika. Rozwiązanie oparto na kilku spójnych paradygmatach: autonomii komunikacyjnej, zapewnionej przez bezpośredni tor łączności komórkowej odporny na zniszczenie pozostałych urządzeń użytkownika; wieloskalowym, dwuwarstwowym modelu akwizycji danych, łączącym ciągłą rejestrację telemetrii z wysokorozdzielczym zapisem zdarzeniowym w duchu rejestratorów danych zdarzeń (EDR, *Event Data Recorder*) oraz architekturze sterowanej zdarzeniami, wykorzystującej stany głębokiego uśpienia dla wydłużenia czasu pracy na zasilaniu baterijnym.

Centralnym wkładem koncepcyjnym pracy jest sposób rozwiązania kluczowego problemu detekcji odróżnienia rzeczywistego upadku od nieszkodliwych perturbacji sygnału inercyjnego generowanych przez normalną jazdę. Zamiast opierać klasyfikację na pojedynczym przekroczeniu progu przyspieszenia, zaproponowano ujęcie zdarzenia jako sekwencji czasowej, realizowane przez trójstanową maszynę stanów: od monitorowania spoczynku przez obserwację poudrzeniową, aż po potwierdzenie zdarzenia. To właśnie ten model traktujący upadek jako proces, a nie chwilowy odczyt zadecydował o odporności systemu na fałszywe alarmy, co potwierdziły testy terenowe, w których nawet przekroczenie progu uderzenia przez gwałtowny przejazd przez krawężnik nie wywołało błędnego alarmu, gdyż algorytm poprawnie wykrył kontynuację jazdy.

Przeprowadzone badania walidacyjne potwierdziły realizowalność przyjętych założeń również na poziomie parametrów. Moduł nawigacji satelitarnej, wykorzystujący jednocześnie konstelacje GPS i BeiDou, osiągnął czasy uzyskania pozycji rzędu jednej sekundy w scenariuszu ciepłego startu oraz dokładność pozycjonowania statycznego w granicach kilku metrów, wystarczającą dla celów lokalizacji ratunkowej. Zbudowany prototyp stanowi tym samym implementacyjny dowód wykonalności (*proof of concept*) opracowanej koncepcji, obejmujący kompletną realizację sprzętową, oprogramowanie układowe oraz lokalny serwis konfiguracyjny.

Praca nie jest wolna od ograniczeń, które wyznaczają kierunki dalszego rozwoju. Należą do nich niedostatki projektu płytki drukowanej ograniczające zarządzanie energią, przestrzeń do poprawy czułości detekcji przy upadkach o łagodnym przebiegu oraz obecna, modułarna konstrukcja prototypu. Ich usunięcie poprzez integrację układu na pojedynczej, zoptymalizowanej płytce, rozszerzenie strategii zarządzania energią oraz migrację kanału alarmowego ku nowszym standardom sieci komórkowych stanowi naturalną kontynuację przeprowadzonych prac. Zasadniczy wniosek pozostaje jednak niezmienny: w oparciu o powszechnie dostępne komponenty możliwe jest zbudowanie w pełni autonomicznego systemu bezpieczeństwa dla mikromobilności, który realnie działa poza laboratorium, a jego skuteczność rozstrzyga się nie na etapie pomiaru, lecz interpretacji sygnału różnicę między danymi laboratoryjnymi a rzeczywistością ujawnia dopiero teren.

Prace cytowane

- [1]. Chidester A. B., Hinch J., Mercer T. C., Schultz K. S., Recording Automotive Crash Event Data, Proceedings of the 17th International Technical Conference on the Enhanced Safety of Vehicles (ESV), Paper No. 01-S4-O-21, Amsterdam, 2001. <https://www-esv.nhtsa.dot.gov/Proceedings/17/Proceed/00021.pdf>. [Dostęp: 04.05.2026].
- [2]. Warren, D., Flight Memory: The Genesis of Aviation Safety Recording, Melbourne Research Press, 1955. Aeronautical Research Laboratories, The ARL 'Black Box' Flight Recorder, Australia: Department of Defence, https://kenblackbox.com/documents/The_ARL_Black_Box_Flight_Recorder.pdf. [Dostęp: 04.05.2026].
- [3]. Niehoff R. G., Kumagai J. S., Chidester A. B., Hinch J., Mercer T. C., Utilizing Data from Automotive Event Data Recorders, Proceedings of the 18th International Technical Conference on the Enhanced Safety of Vehicles (ESV), Paper No. 445, Nagoya, 2003. https://www.nhtsa.gov/sites/nhtsa.gov/files/utilizing_data_from_automotive_event_data_recorders.pdf. [Dostęp: 04.05.2026].
- [4]. Pierowicz, J., Fuglewicz, D. P., Wilson, G. R., Development of requirements and functional specifications for crash event data recorders: final report, General Dynamics Corporation. Advanced Information Systems, 2004. United States: Federal Highway Administration, <https://rosap.ntl.bts.gov/view/dot/4230>. [Dostęp: 04.05.2026].
- [5]. Parczewski, K., Romaniszyn, K., „Czarne skrzynki” w pojazdach samochodowych, Logistyka, nr 1, 2013. Instytut Logistyki i Magazynowania, <https://logistyka.net.pl/wp-content/uploads/2013/01/254.pdf>. [Dostęp: 04.05.2026].
- [6]. Lannuzel M., Bendjellal A., Digital investigation on ships: a Voyage Data Recorder implementation, Proceedings of the 7th International Conference on Computer Science and Application Engineering (CSAE 2023), Wuhan, China, 2023. https://www.researchgate.net/profile/Maxence-Lannuzel/publication/380760727_Digital_investigation_on_ships_a_Voyage_Data_Recorder_implementation/links/664dbd68bc86444c72f697f9/Digital-investigation-on-ships-a-Voyage-Data-Recorder-implementation.pdf. [Dostęp: 04.05.2026].
- [7]. Łukasz. Krzak, " Hardware abstraction layer for microcontrollers", Akademia Górniczo-Hutnicza. https://home.agh.edu.pl/~lkrzak/hal/hal_idea.pdf. [Dostęp: 04 Maja 2026].
- [8]. Espressif Systems, ESP32-C6 Series Datasheet v1.1, Espressif Systems (Shanghai) Co., Ltd., 2024. https://files.seeedstudio.com/wiki/SeeedStudio-XIAO-ESP32C6/res/esp32-c6_datasheet_en.pdf [Dostęp: 04.05.2026].
- [9]. Garmin Support „How Does the Incident Detection and Assistance Feature Work?” <https://support.garmin.com/en-US/?faq=RfaXahBWkH8Q7pVFLsuUmA&productID=621922&searchQuery=incident%20detection&tab=topics> [Dostęp: 04.05.2026].
- [10]. <https://www.garmin.com/pl-PL/p/1196129/> [Dostęp: 04.05.2026].
- [11]. <https://eu.wahoofitness.com/devices/bike-computers/elemt-roam-buy> [Dostęp: 04.05.2026]
- [12]. Maker, R., Specialized ANGi Helmet Sensor & App: Everything you need to know, DC Rainmaker, 2018. <https://www.dcrainmaker.com/2018/11/specialized-helmet-sensor.html>. [Dostęp: 04.05.2026].
- [13]. <https://uvex.com.pl/sensor-uvex-tocsen-crashsensor-pink-41-9-101-02/?srsltid=AfmBOoqhQ-zXJYWA-hEP0lxUTPdL09Lv56dxqmiusNIOU8nxLuvIajs> [Dostęp: 04.05.2026].

- [14]. LIVALL, Instrukcja obsługi inteligentnego kasku rowerowego (BH60SE/BH62/MT1), LIVALL Tech Co., Ltd., Shenzhen, 2020.
https://livall.pl/pl/index.php?controller=attachment&id_attachment=7. [Dostęp: 04.05.2026].
- [15]. <https://cycliq.com/products/fly6-pro-fly12-sport-bundle?srltid=AfmBOooDJQaqRNM-HKkBbhcWwAUHJonuU7gvxu7LSuSu7ae3kj50bBhu> [Dostęp 04.05.2026].
- [16]. BikeFinder, BikeFinder Tracker Generation 2: User Manual, BikeFinder AS, 2025.
<https://bikefinder.com/wp-content/uploads/2025/06/EN-Gen2-Manual.pdf>. [Dostęp: 04.05.2026].
- [17]. AlterLock, Anti-theft Alarm & GPS Tracker for Road Bikes, Nextscape Inc., Tokyo, 2024.
<https://alterlock.net/en/user-manual/alterlock-gen3>. [Dostęp: 04.05.2026].
- [18]. ISO/IEC 14882:2024 - Programming languages - C++ (7th edition; informally "C++23"). Organization: ISO/IEC JTC 1/SC 22/WG 21. Year: published 2024-10-19 (ISO stage 60.60);
<https://www.iso.org/standard/83626.html> [Dostęp 04.05.2026].
- [19]. Arduino-ESP32 documentation, Espressif Systems.
<https://docs.espressif.com/projects/arduino-esp32/en/latest/> [Dostęp: 04.05.2026].
- [20]. FreeRTOS official site, <https://www.freertos.org/> [Dostęp: 04.05.2026].
- [21]. Visual Studio Code Documentation, <https://code.visualstudio.com/docs> [Dostęp: 04.05.2026].
- [22]. PlatformIO documentation, <https://docs.platformio.org/> [Dostęp: 04.05.2026].
- [23]. Pioduino documentation, <https://github.com/pioarduino> [Dostęp: 04.05.2026].
- [24]. Espressif32 platform <https://github.com/platformio/platform-espressif32> [Dostęp: 04.05.2026].
- [25]. BME280 library https://github.com/adafruit/Adafruit_BME280_Library [Dostęp: 04.05.2026].
- [26]. Adafruit Sensor library https://github.com/adafruit/Adafruit_Sensor/ [Dostęp: 04.05.2026].
- [27]. TinyGPSPlus library <https://github.com/mikalhart/TinyGPSPlus> [Dostęp: 04.05.2026].
- [28]. Adafruit SdFat Fork <https://github.com/adafruit/SdFat/> [Dostęp: 04.05.2026].
- [29]. Wi-Fi API <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html> [Dostęp: 04.05.2026].
- [30]. Wire API <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/i2c.html> [Dostęp: 04.05.2026].
- [31]. SPI API <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/spi.html> [Dostęp 04.05.2026].
- [32]. Hardware Serial API <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/serial.html> [Dostęp 04.05.2026].
- [33]. Sleep Modes API https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html [Dostęp 04.05.2026].
- [34]. HTML Living Standard, WHATWG <https://html.spec.whatwg.org/> [Dostęp: 04.05.2026].
- [35]. CSS Snapshot 2026, W3C, <https://www.w3.org/TR/css-2026/> [Dostęp: 04.05.2026].
- [36]. JavaScript <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Dostęp: 04.05.2026].
- [37]. Leaflet Documentation, <https://leafletjs.com/reference.html> [Dostęp: 04.05.2026].
- [38]. Python Documentation, <https://docs.python.org/3/> [Dostęp: 04.05.2026].
- [39]. Python Documentation, Tkinter, <https://docs.python.org/3/library/tkinter.html> [Dostęp: 04.05.2026].
- [40]. pandas Documentation, <https://pandas.pydata.org/docs/> [Dostęp: 04.05.2026].
- [41]. NumPy Documentation, <https://numpy.org/doc/stable/> [Dostęp: 04.05.2026].

- [42]. Matplotlib Documentation, <https://matplotlib.org/stable/> [Dostęp: 04.05.2026].
- [43]. Git official site <https://git-scm.com/> [Dostęp: 04.05.2026]
- [44]. Cursor, <https://cursor.com> [Dostęp: 04.05.2026]
- [45]. Official site <https://www.kicad.org/> [Dostęp: 04.05.2026].
- [46]. Official PTC site <https://www.ptc.com/en/products/creo/parametric> [Dostęp: 04.05.2026].
- [47]. https://wiki.seeedstudio.com/xiao_esp32c6_getting_started/ [Dostęp: 04.05.2026].
- [48]. https://wiki.seeedstudio.com/xiao_pin_multiplexing_esp32c6/ [Dostęp: 04.05.2026].
- [49]. <https://kamami.pl/moduly-gsmgprs/580381-zestaw-z-modulem-gprs-gsm-sim800l-v20-5906623458332.html/> [Dostęp: 04.05.2026].
- [50]. <https://kamami.pl/czujniki-6dof-9dof-10dof/557796-gy-6500-modul-akcelometru-oraz-zyroskopu-z-ukladem-mpu-6500-5906623473274.html/> [Dostęp: 04.05.2026].
- [51]. <https://wiki.seeedstudio.com/Grove-GPS-Air530/> [Dostęp: 04.05.2026].
- [52]. https://www.adafruit.com/product/2652?srltid=AfmBOor8R-DWciiEZY_yo5R5je6Yg0FPo2hgrZcGfGbeaoXi6n4y50L/ [Dostęp: 04.05.2026].
- [53]. https://www.dfrobot.com/product-1796.html?srltid=AfmBOoqZXmw3saIErM3YsPOJkWYFVFTK-iNc937567keRteV6_CGfy9 [Dostęp: 04.05.2026]
- [54]. <https://kamami.pl/czytniki-kart-pamieci/587139-kamodmicrosd-5906623433377.html/> [Dostęp: 04.05.2026]
- [55]. <https://botland.com.pl/ekspandery-wyprowadzen/6505-mcp23008-e-p-ekspander-wyprowadzen-i2c-8-kanalowy-5904422362133.html/> [Dostęp: 04.05.2026]
- [56]. <https://bto.pl/akumulatory/przemyslowe/akumulatory-li-ion/2953,sony-murata-us18650vtc6-3000mah-hp/> [Dostęp: 04.05.2026]
- [57]. <https://sklep.msalamon.pl/produkt/przetwornica-step-down-up-5v-3a-tps63020/> [Dostęp: 04.05.2026]

Wykaz rysunków

Rysunek 1. Garmin Edge 1050 (źródło: [10]) .	14
Rysunek 2. Czujnik upadku Tocsen do montażu na kasku (źródło: [13]).	14
Rysunek 3. Kamera rowerowa przednia CYCLIQ FLY12 Sport i tylna FLY6 Pro (źródło: [15]) .	15
Rysunek 4. Lokalizator GPS/GSM BikeFinder (źródło: [16]).	16
Rysunek 5. Diagram blokowy systemu (źródło: Opracowanie własne).	24
Rysunek 6. Diagram stanów pracy (źródło: Opracowanie własne).	25
Rysunek 7. Diagram przepływu danych (źródło: pracowanie własne).	26
Rysunek 8. Ekran startowy środowiska pioarduino w edytorze Visual Studio Code (źródło: opracowanie własne).	29
Rysunek 9. Menedżer projektu środowiska KiCad z zestawem dostępnych narzędzi (źródło: opracowanie własne).	33
Rysunek 10. Ekran startowy środowiska projektowego PTC Creo Parametric (źródło: opracowanie własne).	34
Rysunek 11. Mikrokontroler Seeed Studio XIAO ESP32C6 (źródło: [47]) .	36
Rysunek 12. Wyprowadzenia pinów mikrokontrolera Seeed ESP32-C6, widok z przodu (źródło: [48]) .	36
Rysunek 13. Wyprowadzenia pinów modułu ESP32-C6, widok od spodu (źródło: [48]) .	37
Rysunek 14. Moduł GPRS GSM SIM800L V2.0 (źródło: [49]) .	37
Rysunek 15. Moduł akcelerometru oraz żyroskopu MPU-6500 (źródło: [50]) .	38
Rysunek 16. Moduł GPS Air530z (źródło: [51]) .	38
Rysunek 17. Moduł czujnika parametrów środowiskowych Adafruit BME280 (źródło: [52]) .	39
Rysunek 18. Moduł czytnika kart MicroSD (źródło: [54]) .	40
Rysunek 19. Ekspander wyprowadzeń MCP23008-E/P (źródło: [55]) .	40
Rysunek 20. Ogniwo litowo-jonowe US18650VTC6 (źródło: [56]) .	41
Rysunek 21. Przetwornica Step-Down/Up 5V 3A TPS63020 (źródło: [57]) .	41
Rysunek 22. Cyfrowy czujnik wstrząsów DFRobot SEN0289 (źródło: [53]) .	42
Rysunek 23. Schemat elektroniczny Czarnej Skrzynki Rowerowej (źródło: opracowanie własne).	43
Rysunek 24. Architektura dystrybucji zasilania (źródło: opracowanie własne).	45
Rysunek 25. Fragment schematu obejmujący sekcję zasilania (źródło: opracowanie własne).	46
Rysunek 26. Fragment schematu: ekspander MCP23008 i interfejs (źródło: opracowanie własne).	47
Rysunek 27. Projekt PCB dedykowanej płytki (źródło: opracowanie własne).	49
Rysunek 28. Widok warstwy F.Cu z widoczną na środku wyspą zasilania 5 V (źródło: opracowanie własne).	51
Rysunek 29. Wizualizacja trójwymiarowa płytki z opisanymi komponentami (źródło: opracowanie własne).	52
Rysunek 30. Zdjęcie zmontowanej płytki bazowej (źródło: opracowanie własne).	54
Rysunek 31. Uproszczony rysunek techniczny obudowy (źródło: opracowanie własne).	54
Rysunek 32. Wizualizacja trójwymiarowa obudowy (źródło: opracowanie własne).	55
Rysunek 33. Wizualizacja trójwymiarowa pokrywy (źródło: opracowanie własne).	55
Rysunek 34. Wstępnie zmontowana obudowa (źródło: opracowanie własne).	56
Rysunek 35. Interfejs użytkownika na obudowie (źródło: opracowanie własne).	57
Rysunek 36. Zmontowane urządzenie przed zamknięciem pokrywy (źródło: opracowanie własne).	58
Rysunek 37. Montaż urządzenia na rowerze (źródło: opracowanie własne).	59

Rysunek 38. Maszyna stanów algorytmu detekcji upadku (źródło: opracowanie własne).	69
Rysunek 39. Schemat maszyny stanów sekwencji alarmowej (źródło: opracowanie własne).....	75
Rysunek 40. Strona główna interfejsu konfiguracyjnego z nawigacją do poszczególnych zakładek (źródło: opracowanie własne).	88
Rysunek 41. Zakładka statusu systemu prezentująca dane diagnostyczne w czasie rzeczywistym (źródło: opracowanie własne).	89
Rysunek 42. Historia jazd z listą zarejestrowanych sesji oraz metadanymi (źródło: opracowanie własne).	90
Rysunek 43. Zakładka konfiguracji kontaktu alarmowego (źródło: opracowanie własne).	93
Rysunek 44. Profil poboru prądu modułu SIM800L podczas pełnej procedury wysyłki wiadomości SMS (źródło: opracowanie własne).....	99
Rysunek 45. Poziomy napięcie podczas trwania sekwencji GSM (źródło: opracowanie własne).	100
Rysunek 46. Dryf pozycji GPS w stanie statycznym (n = 736, t = 60 min) (źródło: opracowanie własne).	102
Rysunek 47. Trasa przejazdu testowego zarejestrowana przez moduł GNSS, z prędkością chwilową odwzorowaną kolorem (źródło: opracowanie własne).	109
Rysunek 48. Profil prędkości chwilowej w funkcji czasu podczas przejazdu testowego. Obszarami zacienionymi zaznaczono okresy aktywnej jazdy (is_moving = 1) (źródło: opracowanie własne).	110
Rysunek 49. Dystans skumulowany oraz prędkość chwilowa w funkcji czasu. Płaski odcinek przebiegu dystansu odpowiada postojowi, podczas którego nie następował przyrost drogi.	110
Rysunek 50. Przebieg magnitudy przyspieszenia oraz prędkości kątowej podczas przejazdu testowego. Linia przerywaną oznaczono próg detekcji uderzenia (impactThresholdG = 8 g) (źródło: opracowanie własne).	111
Rysunek 51. Przebieg parametrów środowiskowych (temperatura, wilgotność względna, ciśnienie atmosferyczne) zarejestrowanych przez czujnik BME280 w trakcie przejazdu testowego (źródło: opracowanie własne)	112

Wykaz tabel

Tabela 1. Zewnętrzne biblioteki programowe	29
Tabela 2. Standardowe biblioteki frameworku Arduino-ESP32	30
Tabela 3. Zestawienie pinów mikrokontrolera ESP32-C6, wraz z ich funkcjami oraz modułów, do których są podłączone.	44
Tabela 4. Urządzenia I ² C wraz z adresami.	45
Tabela 5. Funkcje wyprowadzeń GP0-GP7 ekspandera MCP23008	47
Tabela 6. Złącza zewnętrzne płytki bazowej.....	48
Tabela 7. Stos warstwowy PCB.....	50
Tabela 8. Moduły oprogramowania urządzenia	60
Tabela 9. Częstotliwość akwizycji danych z czujników	65
Tabela 10. Parametry konfiguracyjne algorytmu detekcji upadku.	72
Tabela 11. Sekwencja komend AT w procedurze wysyłania wiadomości SMS.	77
Tabela 12. Adaptacyjne interwały zapisu logu jazdy.	83
Tabela 13. Endpointy interfejsu programistycznego API.	90
Tabela 14. Wykorzystanie trybów pracy w procedurach testowych.	95
Tabela 15. Aparatura pomiarowa wykorzystana w procedurach testowych.	96
Tabela 16. Pomiar poboru prądu w trybach pracy.	98
Tabela 17. Wyniki testów autonomii energetycznej urządzenie.....	101
Tabela 18. Czas uzyskania pierwszego fixu pozycji (TTFF) moduł Air530z w warunkach otwartej przestrzeni.	101
Tabela 19. Zestawienie wyników testu układu BME280 w temperaturze około 28°C.	103
Tabela 20. Zestawienie wyników testu układu BME280 w temperaturze około 6°C.	103
Tabela 21. Niezawodność komunikacji GSM.	104
Tabela 22. Wyniki walidacji algorytmu detekcji upadku w scenariuszach kontrolowanych	106
Tabela 23. Końcowe wartości parametrów algorytmu detekcji upadku wykorzystane w testach walidacyjnych.	107

Wykaz kodu

Listing 1. Pseudokod sekwencji inicjalizacji i pętli głównej.....	62
Listing 2. Wzorzec bramki czasowej.....	62
Listing 3. Struktura danych SensorData	63
Listing 4. Struktura FallDetParams.....	69
Listing 5. Fragment implementacji algorytmu detekcji upadku.....	73
Listing 6. Fragment obsługi stanu HoldGp0 w maszynie stanów GSMSeq.....	81
Listing 7. Fragment implementacji transmisji SMS w module communication.cpp	82
Listing 8. Uproszczony przepływ obsługi żądania HTTP w funkcji	92