

**Politechnika Warszawska**

W Y D Z I A Ł E L E K T R Y C Z N Y



INSTYTUT ELEKTROTECHNIKI TEORETYCZNEJ  
I SYSTEMÓW INFORMACYJNO-POMIAROWYCH

# Praca dyplomowa magisterska

na kierunku Informatyka  
w specjalności Inżynieria Oprogramowania

ANALIZA AKTYWNOŚCI UŻYTKOWNIKA NA PODSTAWIE DANYCH  
O PRZYSPIESZENIU

**Dominik Gebert**

Numer albumu 238834

promotor  
dr inż. Marcin Kołodziej

Warszawa 2018

## **Streszczenie**

Celem pracy jest analiza danych z akcelerometru, znajdującego się w telefonie komórkowym, w celu wykrywania aktywności fizycznej użytkownika. W pierwszej części pracy przedstawiono wykorzystane narzędzia, które posłużyły do realizacji badania oraz zaprezentowano dostępne rozwiązania, pozwalające na rozpoznawanie aktywności fizycznej użytkownika.

W drugim rozdziale opisano autorską aplikację działającą pod kontrolą systemu Android, która posłużyła do zebrania materiału badawczego. Materiał badawczy zawiera dane zgromadzone od czterech osób. Każda z osób zarejestrowała sygnały odpowiadające pięciu klasom aktywności fizycznej: bezczynności, chodzeniu, bieganiu, schodzeniu i wchodzeniu po schodach.

Kolejny rozdział zawiera informacje o sposobie przetwarzania zebranych sygnałów. Do wykonania obliczeń wykorzystany został język Python. Sygnały podzielono na okna z których wyekstrahowano cechy sygnału: średnią arytmetyczną, wartość pierwszego i trzeciego kwartylu, medianę, odchylenie standardowe, wariancję, wartość minimalną i maksymalną, współczynnik skośności oraz kurtozę.

W czwartym rozdziale przeprowadzono klasyfikację, która objęła rozpoznawanie pięciu klas aktywności fizycznej. Proces klasyfikacji poddano następnie optymalizacji. W pierwszej części zweryfikowano jaki jest najbardziej efektywny sposób wyliczania cech sygnału dla trzech przypadków: obliczania cech z danych pochodzących bezpośrednio z akcelerometru, z wartości modułu wektora, oraz dla połączenia obu tych zbiorów. Weryfikację wykonano dla pięciu klasyfikatorów, którymi były: drzewo decyzyjne, las losowy, perceptron wielowarstwowy, klasyfikator Bayesa oraz klasyfikator k najbliższych sąsiadów. Weryfikując trafność klasyfikacji każdego z nich, do dalszej części pracy wybrano klasyfikator pozwalający uzyskać najlepsze rezultaty.

Kolejnym krokiem była weryfikacja w jaki sposób liczba próbek w oknie wpływa na trafność klasyfikacji. Zweryfikowano także jak modyfikacja liczby i zestawu cech sygnału wpływa na osiąganą trafność klasyfikacji. Po weryfikacji wyznaczono cechy pozwalające na osiągnięcie najlepszych wyników.

Na koniec przedstawiono podsumowanie z przeprowadzonych badań. Określono jakie parametry procesu klasyfikacji są optymalne w procesie klasyfikacji aktywności fizycznej użytkownika na podstawie sygnału przyspieszenia. Wskazano ponadto, gdzie sygnał przyspieszenia zarejestrowany przez smartfon z akcelerometrem może znaleźć zastosowanie. Wykrywanie aktywności fizycznej może bowiem zostać wykorzystanie w wielu dziedzinach życia.

## **Słowa kluczowe**

Wykrywanie aktywności fizycznej, akcelerometr, przyspieszenie, Android, klasyfikacja, przetwarzanie sygnału, optymalizacja parametrów

## **Abstract**

The aim of the study is to analyse accelerometer data which was collected with the usage of a smartphone in order to detect a user's physical activities. The first part presents the tools which were used to conduct the research and currently available solutions.

The second chapter presents research material together with the way in which it was obtained. An Android application, which was developed for this study, was also described. The main task of the application was to register acceleration data to text files. By means of that application four sets of files were collected from four different people. Each of these file sets contained the following five physical activities: inactivity, walking, running, going downstairs and upstairs.

The next chapter includes information about acceleration signal processing. The signal saved in files was processed using Python language. Data was subsequently divided into equal windows from which the features of the signal were extracted.

The fourth chapter starts with probationary classification conducted on one of the previously registered signals. There were five classes of physical activity. Thereafter, the optimization process was used to achieve the best classification accuracy. The process starts with verification of the most effective way of features extraction: from raw accelerometer data, from modulus or from both datasets combined. Further part of this study was carried out with the option which gave the best results. There were verified five classifiers: decision tree, random forest, multilayered perceptron, Bayes classifier and k nearest neighbours classifier.

The next step stage of the optimization process was to determine how the number of samples in the windows influences classification accuracy and what is the most appropriate signal features set.

Finally, a summary of the conducted research was presented. It was determined which classification parameters are optimal in the process of user physical activity recognition. It was also indicated where acceleration registered with a smartphone may be used since physical activity recognition can be used in many areas of life.

## **Keywords**

Activity recognition, accelerometer, accuracy, Android, classification, signal processing, parameter optimization

## **OŚWIADCZENIE**

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

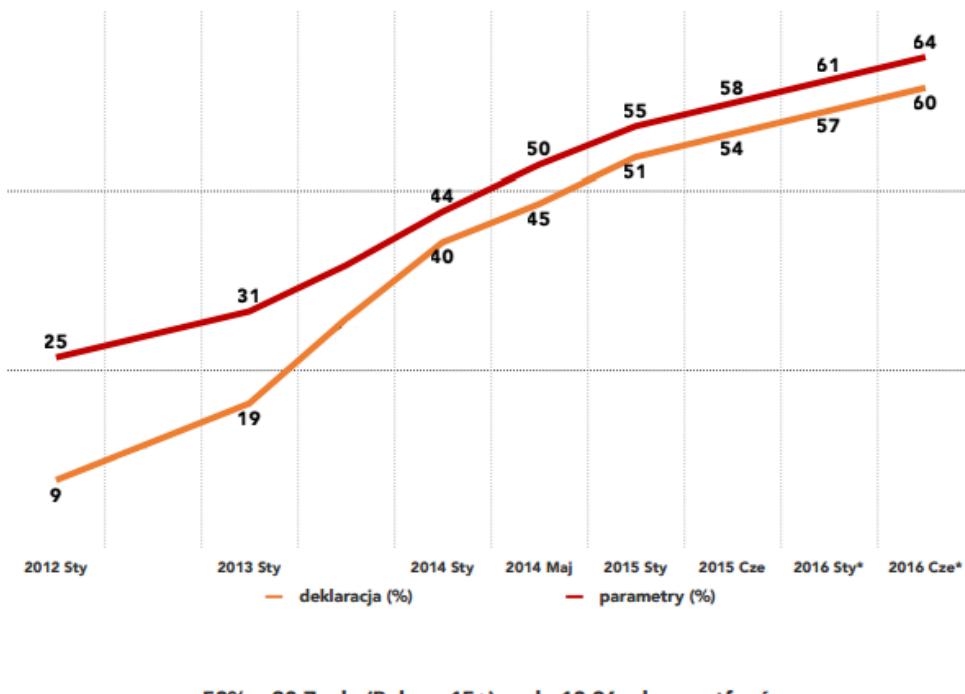
Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

## **Spis treści**

<b>1. WSTĘP.....</b>	<b>6</b>
<b>1.1. Cel i zakres pracy.....</b>	<b>8</b>
<b>1.2. Wykorzystane narzędzia .....</b>	<b>8</b>
<b>1.3. Przykładowe rozwiązania monitorowania aktywności fizycznej.....</b>	<b>11</b>
<b>2. MATERIAŁ BADAWCZY .....</b>	<b>15</b>
<b>2.1. Aplikacja rejestrująca sygnał przyspieszenia .....</b>	<b>15</b>
<b>2.2. Zarejestrowane sygnały.....</b>	<b>29</b>
<b>3. PRZETWARZANIE SYGNAŁU Z AKCELEROMETRU .....</b>	<b>31</b>
<b>3.1. Załadowanie danych .....</b>	<b>31</b>
<b>3.2. Obliczanie modułu wektora przyspieszenia.....</b>	<b>36</b>
<b>3.3. Podział sygnału na okna.....</b>	<b>40</b>
<b>3.4. Ekstrakcja cech .....</b>	<b>43</b>
<b>4. REZULTATY KLASYFIKACJI .....</b>	<b>47</b>
<b>5. PODSUMOWANIE.....</b>	<b>62</b>
<b>6. BIBLIOGRAFIA .....</b>	<b>64</b>

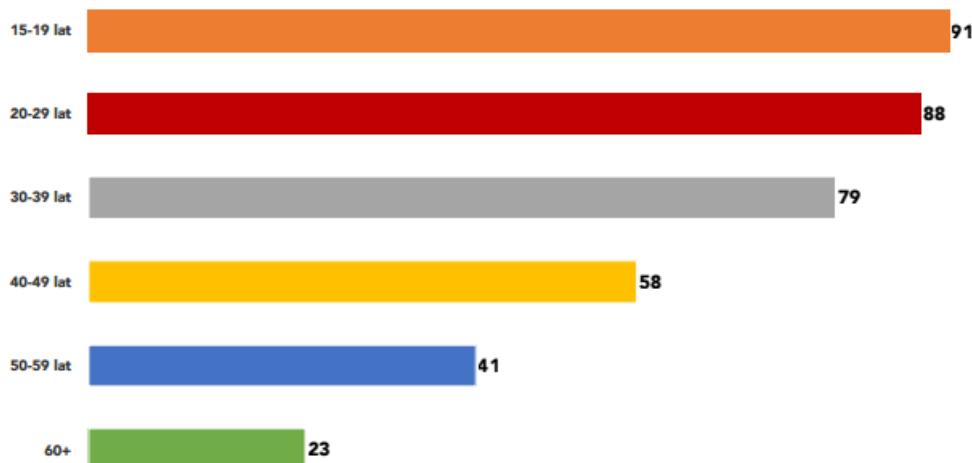
## 1. Wstęp

Kiedy jeszcze kilka lat temu telefon służył wyłącznie dzwonieniu i wymienianiu wiadomości tekstowych, niewiele osób przypuszczało, że dziś będziemy nosili przy sobie tak mocne obliczeniowo urządzenia. Co więcej, niemalże dwóch na trzech Polaków posiada smartfona, a w ciągu roku przybywa kolejne 6-8% nowych użytkowników. Dynamikę wzrostu liczby smartfonów wśród Polaków przedstawia poniższy Rysunek 1.1.



Rysunek 1.1. Dynamika wzrostu liczby smartfonów wśród Polaków [1].

W przypadku osób w wieku 15-19 lat procentowy udział użytkowników smartfonów wyniósł 91% [1]. Można zatem wnioskować, iż użytkowników smartfonów w kolejnych pokoleniach będzie co raz więcej. Tendencja jest wzrostowa, a perspektywy na przyszłość optymistyczne dla producentów telefonów. Już dziś w wielu zastosowaniach telefon komórkowy zastępuje komputer. Czynności które przeciętni użytkownicy dalej wolą wykonywać na komputerze, związane są przede wszystkim z większym ekranem. W zastosowaniach profesjonalnych komputery dalej mają przewagę ze względu na możliwości obliczeniowe. Pełne zestawienie obrazujące udział smartfonów w podziale na grupy wiekowe obrazuje Rysunek 1.2.



Rysunek 1.2. Diagram przedstawiający procentowy udział użytkowników smartfonów w poszczególnych grupach wiekowych [1].

Da się zauważać także, że coraz więcej osób zaczyna dbać o swoją aktywność fizyczną [2]. Z pewnością siedzący tryb życia, czy w domu, czy w pracy sprawia, że troska o kondycję fizyczną staje się nie tyle dodatkiem do codziennych czynności lecz także pewnym obowiązkiem.

Obecnie zalecane przez specjalistów wykonywanie 10 000 kroków dziennie [3] zajmuje przeciętnej osobie nieco ponad godzinę. Poziom aktywności fizycznej niezbędnej do zachowania zdrowia jest jednak sprawą indywidualną, a sama liczba kroków nie jest tutaj odpowiednią miarą. Każdy z nas powinien poświęcać od 20 minut do 2 godzin dziennie na ćwiczenia aerobowe. Pozwalają one zapobiec wielu chorobom układu krążenia, które są obecnie główną przyczyną zgonów. W przypadku Stanów Zjednoczonych wskazane choroby doprowadzają do śmierci większej liczby osób niż wszystkie formy raka razem wzięte [4]. Zważywszy na wymienione aspekty, na rynku pojawiają się coraz nowsze rozwiązania pozwalające na monitorowanie codziennej aktywności fizycznej. Zajmują się tym głównie firmy produkujące urządzenia dla sportowców jak Garmin czy Polar. Na rynku dostępne są także urządzenia przeznaczone dla przeciętnego użytkownika, chcącego monitorować swoją aktywność w ciągu dnia. Oprogramowanie obsługujące takie urządzenia dodaje zwykle do pomiarów element grywalizacji, co skutkuje większą motywacją do ruchu i pozwala na rywalizację ze znajomymi.

Posiadanie specjalistycznego rozwiązania do śledzenia aktywności fizycznej nie jest jednak niezbędnie. Posiadając smartfona mamy możliwość na wykorzystanie go właśnie w tym celu. Właścicieli takich telefonów jest obecnie co raz więcej [5]. Osiągana precyzja pomiaru przez dedykowane aplikacje jest wystarczająca by mierzyć aktywność dzięki wykorzystaniu wbudowanych, sprzętowych sensorów.

## **1.1. Cel i zakres pracy**

Celem pracy jest dokonanie analizy sygnału z akcelerometru znajdującego się w smartfonie, pod kątem rozpoznawania aktywności fizycznej użytkownika. Do zebrania danych, będących przedmiotem badań, posłuży autorska aplikacja na system Android, napisana w języku Java.

Zarejestrowane sygnały poddane będą analizie w celu wykrywania aktywności fizycznej użytkownika. Celem przetwarzania sygnału zaimplementowany zostanie projekt w języku Python, z wykorzystaniem bibliotek *sklearn*, *numpy* i *matplotlib*.

W kolejnym kroku proces klasyfikacji zostanie zoptymalizowany pod kątem sposobu ekstrakcji cech. Dokonana zostanie weryfikacja w jaki sposób relacje pomiędzy zbiorami treningowymi, szerokość okna i zestaw ekstrahowanych cech wpływają na osiągane rezultaty.

Ostatecznie na wynik pracy złoży się szereg elementów, którymi są:

- aplikacja służąca do rejestracji sygnału przyspieszenia,
- zebrane dane czterech osób, będące zestawem sygnałów odpowiadających pięciu klasom aktywności fizycznej dla każdej z osób,
- wytypowanie najlepszego klasyfikatora spośród badanych w omawianym zastosowaniu,
- ustalenie optymalnych parametrów przetwarzania sygnału dla rozpoznawania aktywności fizycznej, w tym zestawu cech sygnału, których wyodrębnianie przynosi najlepsze rezultaty.

## **1.2. Wykorzystane narzędzia**

Celem odpowiedniego wykonania pracy zostały wykorzystane liczne narzędzia, których użycie pozwoliło na odpowiednie opracowanie każdego z elementów pracy. Narzędzia jakie wykorzystano to przede wszystkim:

- Android Studio 3.0.1 – środowisko pozwalające na tworzenie aplikacji do zbierania próbek sygnału przyspieszenia,
- JetBrains PyCharm Community Edition 2017.3.3 – narzędzie do obsługi języka Python, wykorzystanego do przeprowadzenia procesu analizy i optymalizacji,
- Microsoft Word 2016 – edytor tekstu do sporządzenia dokumentu,
- serwis [github.com](https://github.com) – serwis do wersjonowania kodu wytworzzonego w ramach pracy.

Komplet powyższych narzędzi dostępny jest bezpłatnie. Wyjątkiem jest tutaj aplikacja Microsoft Word 2016, która została wykorzystana wyłącznie do sporządzenia tego dokumentu.

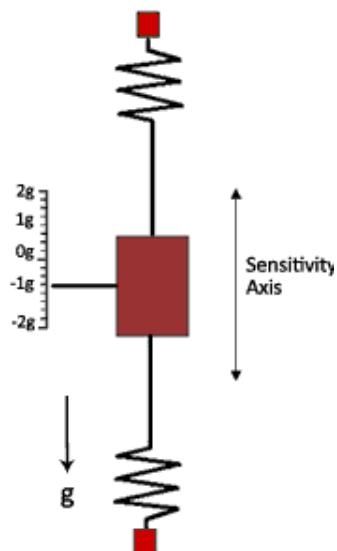
Dodatkowo w celu zarejestrowania sygnału przyspieszenia wykorzystane zostanie urządzenie Huawei P10. Smartfon pracuje pod kontrolą systemu Android. Wyposażono go w akcelerometr, który posłuży do realizacji zadania gromadzenia próbek sygnału przyspieszenia do plików. Główne parametry wykorzystanego telefonu to:

- procesor Kirin 960 (4 rdzenie, 2,40 GHz, Cortex A73 + 4 rdzenie, 1,80 GHz Cortex A53),
- 4GB Pamięci RAM,
- 64 GB Pamięci wewnętrznej,
- bateria litowo-polimerowa 3200 mAh,
- system Android 7.0 Nougat.

Wymagania stawiane przed telefonem, który mógłby posłużyć do rejestrowania aktywności fizycznych ograniczają się do obecności akcelerometru. Bateria powinna pozwolić na zarejestrowanie sygnału przyspieszenia w czasie niezbędnym do zgromadzenia danych. Odpowiednia moc obliczeniowa telefonu zapewniana jest obecnie przez większość telefonów komórkowych.

Sam akcelerometr, zwany też przyspieszeniomierzem czy przetwornikiem przyspieszenia znajduje się obecnie w większości modeli sprzedawanych współcześnie telefonów. Jak wskazuje nazwa, akcelerometr służy do pomiaru przyspieszenia oddziałującego na dany obiekt. Przetwornik przyspieszenia umożliwia wykrywanie położenia urządzenia w przestrzeni oraz pozwala na sterowanie nim poprzez poruszanie.

Akcelerometr mierzy przyspieszenie masy bezwładnej. Należy pamiętać, że dla pomiaru ruchu tejże masy w osi pionowej, prostopadłej do powierzchni Ziemi, zmierzona wartość będzie poddana działaniu grawitacji. Siła grawitacji jest siłą wynikającą z przyspieszenia ziemskiego. Grawitacja na naszej planecie wynosi w przybliżeniu  $9,81 \text{ m/s}^2$  co oznaczane jest jako  $1g$ . Wynik pomiaru akcelerometru przedstawiany jest właśnie w jednym ze wskazanych formatów tj. za pomocą jednostki  $\text{m/s}^2$  lub przedstawiając wartość będącą mnożnikiem jednostki „ $g$ ”, czyli przyspieszenia ziemskiego. Podstawowa zasada działania akcelerometru może zostać przedstawiona za pomocą umieszczonej w obudowie bezwładnej masy zawieszonej na sprężynie. Model działania akcelerometru został zaprezentowany na Rysunku 1.3.

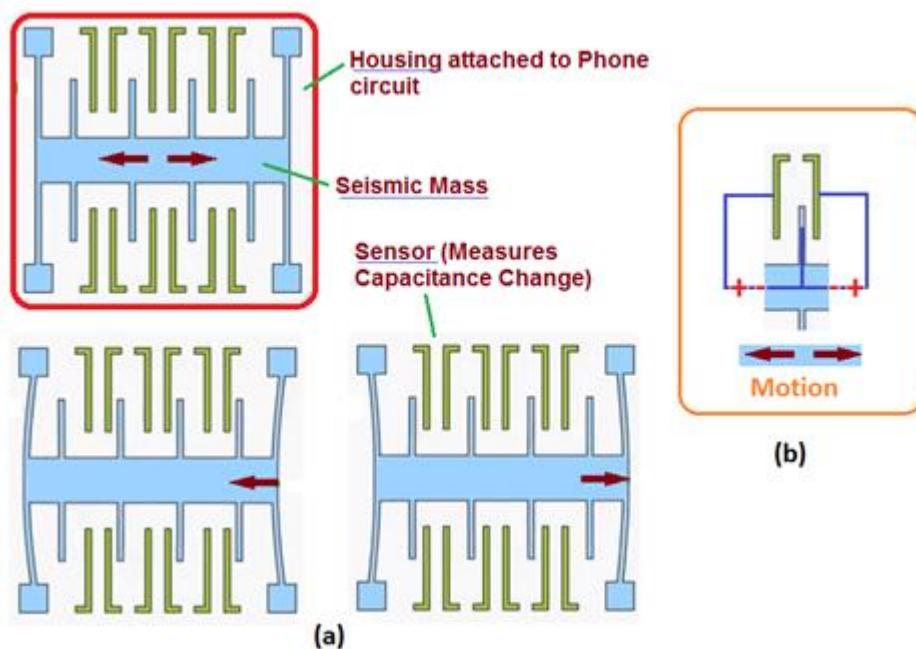


Rysunek 1.3. Schemat działania akcelerometru przedstawiony na rysunku poglądowym [6].

Umieszczenie trzech elementów, jak z Rysunku 1.3, wzdłuż trzech osi pozwala wyznaczyć stan przyspieszenia obiektu w trójwymiarowej przestrzeni. Wykonując pomiar jak bardzo każda ze sprężyn została ugięta lub rozciągnięta można obliczyć wartość siły bezwładności, a tym samym przyspieszenie obiektu.

Akcelerometr umieszczony w telefonie co do zasady działa podobnie do schematu przedstawionego na Rysunku 1.3, jednak jego budowa jest zgoła inna. Akcelerometr w telefonie zawiera obwód posiadający masę wykonaną z krzemu. Pomiary przyspieszenia opierają się na zmianie położenia tej masy wraz ze zmianą orientacji i poruszania się urządzenia [7].

Obecnie akcelerometr występujący w smartfonie jest obwodem będącym mikroukładem elektromechanicznym i pozwala na zmierzenie sił wynikających z przyspieszenia spowodowanego grawitacją, przemieszczania się lub zmiany położenia. Na potrzeby urządzeń mobilnych układ zwykle mierzy także kąt, pod którym telefon jest pochylony. Pomiar kątu wykorzystywany jest jednak przede wszystkim do obsługi zmiany orientacji ekranu z pionowej na poziomą i odwrotnie. Zasadę działania akcelerometru występującego w telefonie obrazuje Rysunek 1.4.



Rysunek 1.4. Schematyczny rysunek obrazujący akcelerometr w telefonie komórkowym [7].

Obraz oznaczony literą (b) jest wycinkiem figury(a). Rysunek 1.4 obrazuje zmiany pojemności w zależności od położenia masy, które ulega zmianie, kiedy orientacja telefonu w przestrzeni zostanie zmieniona. Zmiana położenia akcelerometru pozwala rozpoznać zmianę siły związanej z przyspieszeniem jako odpowiednik zmiany pojemności. Akcelerometr w telefonie komórkowym opiera się właśnie na zmianach pojemności.

Urządzeniem znajdującym się w wykorzystanym telefonie Huawei P10 jest moduł iNEMO oznaczony symbolem LSM6DSM, zaprojektowany i stworzony przez firmę ST. Urządzenie posiada trzyosiowy akcelerometr oraz żyroskop i stworzone zostało do wykorzystania w smartfonach, IoT oraz innych produktach, gdzie wymaganiem jest energooszczędność i mały rozmiar.

Cały moduł posiada 14 pinów, a jego rozmiar zamyka się w prostokącie o wymiarach 2,5mm x 3mm, przy wysokości 0,83mm. Sam akcelerometr może pracować w zakresach  $\pm 2g/\pm 4g/\pm 8g/\pm 16g$ . Akcelerometr posiada także kolejkę typu First In First Out o pojemności do 4 kilobajtów, w zależności od konfiguracji urządzenia. Sam sensor posiada kontroler przerwań generujący sygnał wyzwalany dla wykrycia:

- ruchu,
- nachylenia,
- pojedynczego dotknięcia ekranu,
- podwójnego kliknięcia ekranu,
- swobodnego opadania urządzenia,

Moduł pobiera 0,65 mA prądu w trybie wysokiej wydajności. Zaawansowane funkcje zarządzania zasileniem systemu pozwalają na dodatkową oszczędność energii. W przypadku trybu energooszczędnego konsumpcja prądu spada do 0,29 miliampera. Dodatkowo urządzenie pozwala na prawidłową pracę w zakresie od -40 do 85 stopni Celsjusza.

Zastosowania akcelerometru i żyroskopu są bardzo zróżnicowane. Ich brak uniemożliwiłby pełne wykorzystanie wszystkich funkcji w jakie wyposażony może zostać współczesny telefon. W szczególności utrudniona byłaby obsługa ekranu dotykowego oraz obsługa gier, w których sterowanie odbywa się za pomocą ruchów wykonywanych telefonem. Wykorzystanie akcelerometru i żyroskopu w telefonach sprowadza się przede wszystkim do:

- rozpoznawania gestów,
- sterowania w grach,
- stabilizacji obrazu,
- sterowania orientacją ekranu.

Umieszczenie modułu akcelerometru w telefonie sprawia, iż jest bardziej dostępny dla użytkowników. Kiedy obecność smartfonów na rynku nie była jeszcze tak duża, mało kto posiadał urządzenie do pomiarów przyspieszenia. Obecnie moduł akcelerometru dostępny jest niemalże dla każdego, co pozwala myśleć o dodatkowych zastosowaniach i eksperymentach, które może wykonać każdy z nas.

### **1.3. Przykładowe rozwiązania monitorowania aktywności fizycznej**

Na rynku obecne jest w tej chwili wiele rozwiązań pozwalających na monitorowanie aktywności fizycznej użytkownika. Część z nich wymaga dedykowanego sprzętu, podczas gdy druga część do rejestracji informacji wymaga jedynie smartfona. Do pierwszej grupy zaliczają się zegarki i opaski fitness. Pierwsze z nich dedykowane są przede wszystkim osobom nastawionym na dokładne badanie aktywności podczas sportu. Wskazywanie jaką czynność użytkownik wykonywał w danym momencie jest w tym wypadku jedynie dodatkową funkcją. Zegarki sportowe pozwalają dokładnie mierzyć takie wartości jak położenie na podstawie GPS i tempo ruchu. Ich możliwości wykraczają jednak znacznie ponad to. Obecnie zegarki wyposażone w specjalne opaski pozwalają na pomiary wychylenia użytkownika od pionu, wysokości podnoszenia stopy podczas ruchu, czy też jej czasu kontaktu z podłożem.

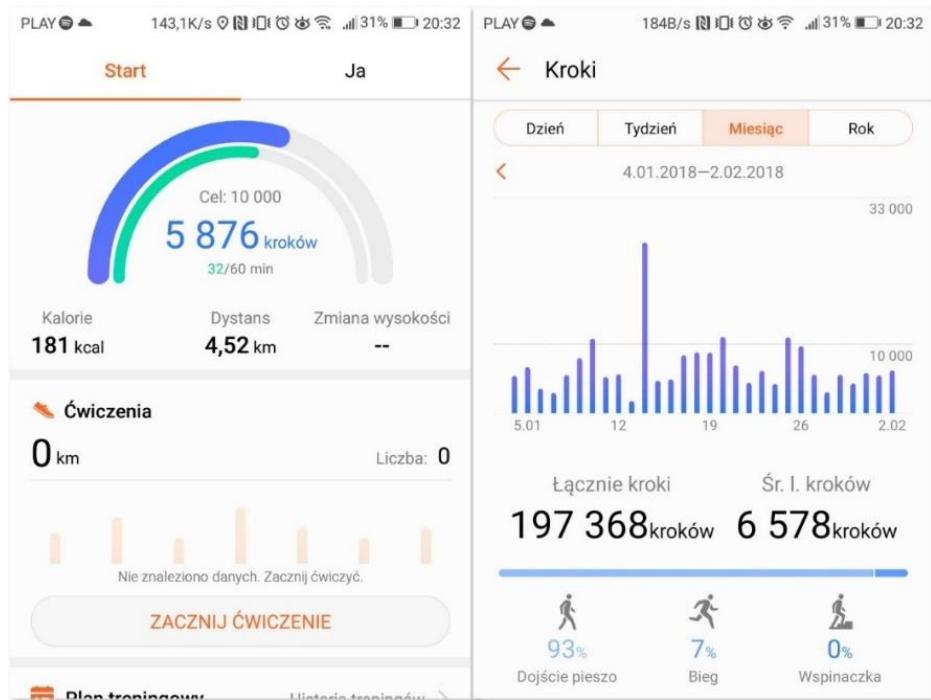
Drugie w kolejności opaski fitness posiadają część funkcji obecnych w zegarkach. Ich głównym zadaniem jest jednak monitorowanie aktywności fizycznej użytkowników w ciągu dnia oraz liczby kroków. Istnieją firmy takie jak Garmin, Suunto czy Polar, których działalność opiera się przede wszystkim na dostarczaniu urządzeń sportowych, a ich oferta jest niezwykle szeroka. Do urządzeń zawsze dołączane są dedykowane aplikacje na których można śledzić swoje postępy. Firma Garmin udostępnia aplikację o nazwie Garmin Connect.. Na głównym ekranie aplikacji znajdują się dane ostatniej aktywności fizycznej zarejestrowanej przez użytkownika. Jest tam obecna informacja o czasie, dystansie i spalonych kaloriach. Ponadto aplikacja przedstawia mapę trasy wraz z szacowanym przewyższeniem i tempem utrzymanym podczas treningu. Dla każdego dnia, rejestrowana jest także liczba pokonanych kroków i czas snu. Na podstawie tych statystyk oraz liczb zarejestrowanych w zewnętrznej aplikacji obliczającej spożyte kalorie, użytkownik aplikacji otrzymuje informacje o swoim bilansie kalorycznym w ramach wskazanego dnia. Liczba kroków, jak i inne dane możliwe są do

podejzenia także w ujęciu zbiorczym, na wykresach. Ekrany pochodzące z aplikacji Garmin Connect przedstawia Rysunek 1.5.



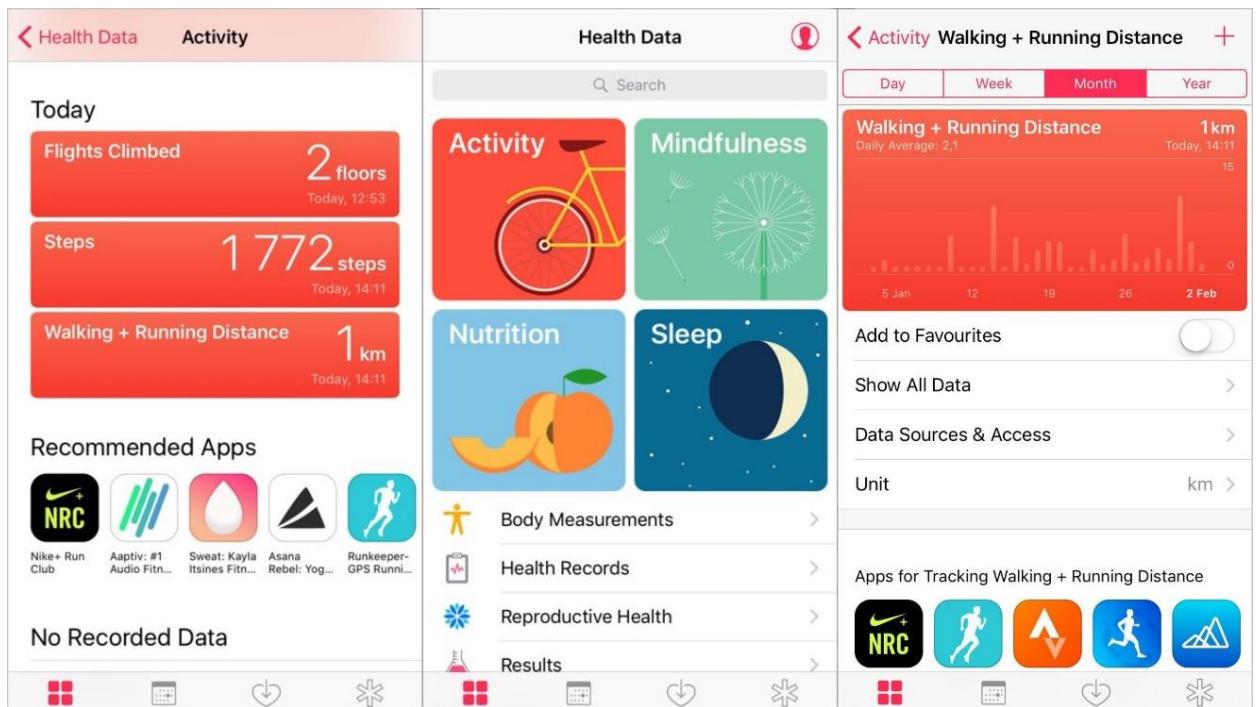
Rysunek 1.5. Ekrany pochodzące z aplikacji Garmin Connect służącej do rejestrowania aktywności [8].

Oddzielnym zestawem narzędzi do pomiaru aktywności fizycznej są aplikacje działające na telefonach bez potrzeby wykorzystania dodatkowych urządzeń. Obecnie każdy czołowy producent telefonów posiada własną aplikację, która służy do zbierania danych użytkownika i prezentuje mu je w formie statystyk. Jedną z nich jest aplikacja Zdrowie dostarczana przez firmę Huawei na ich smartfonach. W aplikacji tej mamy wyznaczony cel dzienny pokonanych kroków oraz wykresy analogiczne jak we wcześniej opisanej aplikacji firmy Garmin. Aplikacja na podstawie wykonanych kroków oblicza spalone kalorie oraz szacuje pokonany dystans. Brak wykorzystania w tym wypadku sygnału GPS wpływa negatywnie na dokładność pomiaru. Podstawowe ekrany aplikacji przedstawia Rysunek 2.2.



Rysunek 1.6. Aplikacja Zdrowie firmy Huawei preinstalowana na telefonach tej marki [9].

Podobną aplikacją do przedstawionej jest aplikacja Health dostarczana na urządzenia firmy Apple. Przedstawiane w niej informacje w zakresie rozpoznawanych aktywności fizycznej są bardzo podobne. Zmieniony jest jedynie interfejs, na którym dane są prezentowane. Aplikację przedstawia Rysunek 1.7.



Rysunek 1.7. Aplikacja Health dostarczana na urządzenia z systemem iOS firmy Apple [10].

Aplikacji i urządzeń rejestrujących aktywność fizyczną jest znacznie więcej, a przedstawione powyżej rozwiązania są jedynie przykładami. Mnogość rozwiązań i rozwijanie

tak wielu aplikacji świadczy o dużym potencjale i chęci użytkowników do ich wykorzystywania. Obecne trendy sugerują, iż ludzie coraz częściej kontrolują swoją aktywność i zdrowie poprzez różne rozwiązania. Wykorzystanie w tym celu aplikacji wydaje się być jednym z prostszych i tańszych rozwiązań dla każdego posiadacza smartfona.

W aplikacji, która zostanie zaimplementowana w ramach pracy, przedstawiony zostanie bardziej szczegółowy podział na klasy aktywności fizycznej. W przypadku obu rozwiązań opisanych powyżej rozróżniane są trzy rodzaje ruchu. W obu z nich zestawy rozpoznawanych aktywności fizycznych różnią się od siebie. W ramach pracy zostanie zbadana możliwość realizacji rozróżniania pięciu klas ruchu użytkownika. Większa liczba rozpoznawanych klas przyczynia się do możliwości dokładniejszego śledzenia czynności wykonywanych w ciągu dnia.

Istnieje jednak wiele innych systemów rozpoznawania aktywności fizycznej, niewykorzystujących danych z pojedynczego czujnika. Do dyspozycji pozostaje między innymi GPS, czy nawet kamery śledzące ruch w celu określenia aktywności obserwowanych osób. Używanie kamer nie jest jednak powszechnie stosowane do tego celu. Rozwiązanie takie jest skomplikowane i wymaga stosowania dodatkowych urządzeń związanych nie z osobą, a ze środowiskiem, w którym przebywa dana osoba. W tej kwestii przewagę ma GPS który także obecny jest w niemalże każdym telefonie. Poza dostarczaniem danych, które pozwalają na zidentyfikowanie aktywności fizycznej, samo założenie systemu GPS pozwala by z dużą dokładnością określić położenie obiektu wyposażonego w odpowiedni czujnik. Problemem jest jednak jego działanie w zamkniętych pomieszczeniach oraz zużycie energii.

Akcelerometr nie pozwala w prosty sposób określić lokalizacji. Wykrywanie aktywności fizycznych na podstawie sygnału przyspieszenia rejestrowanego przez akcelerometr jest jednak możliwe. Ponadto działanie akcelerometru znacznie mniej obciąża baterię urządzenia przenośnego. Pozwala tym samym na obserwację w znacznie dłuższym czasie.

Jak pokazują badania użytkownicy nie dostrzegają potencjalnej możliwości śledzenia zachowań użytkownika poprzez wykorzystanie akcelerometru. Jednym z badań były badania przeprowadzone przez firmę UbiFit. Dotyczyły one wykorzystania technologii do zachęcania osób do aktywności fizycznej. Doświadczenie trwało 3 miesiące i zostało przeprowadzone na 24 osobach. Piętnaście z nich otrzymało urządzenie zawierające akcelerometr i barometr, podczas gdy pozostała ósemka spisywała swoje aktywności w tradycyjnym dzienniku.

W końcowej fazie doświadczenia, osobom biorącym w nim udział przedstawiono możliwe usprawnienia podkreślając dokładnie z czym wiąże się dodanie kolejnego czujnika jak GPS czy nagrywanie audio, zawierającego jedynie wybrane częstotliwości.

Co zrozumiałe uczestnicy reagowali w sposób zróżnicowany na potencjalną możliwość dodania kolejnych sensorów. Zauważali oni, że system GPS pozwoli określić zewnętrznej firmie i potencjalnie niebezpiecznym osobom dokładne miejsce zamieszkania, czy też miejsce w którym dana osoba aktualnie przebywa. Co ciekawe, żadna z ankietowanych osób nie miała zastrzeżeń dotyczących akcelerometru i barometru. Co więcej, nikt nie dostrzegał zagrożenia związanego z nagrywaniem danych z tych czujników przez 24 godziny, 7 dni w tygodniu. Na nikim nie wywarło wrażenia także przechowywanie zarejestrowanego sygnału w nieokreślonym czasie na telefonie komórkowym czy też urządzeniu fitness [11].

## **2. Materiał badawczy**

Zarejestrowane sygnały przyspieszenia zebrano za pomocą autorskiej aplikacji na smartfon działający pod kontrolą systemu Android. Aplikacja została przygotowana w ramach pracy i pozwala na zapis danych z akcelerometru do plików CSV. Aplikację napisano w języku Java w wersji ósmej.

Zbiór danych niezbędny do przeprowadzenia badania miał zawierać pliki odpowiadające poszczególnym aktywnościom fizycznym. Każda z aktywności fizycznych miała być zarejestrowana dla kilku osób tak by umożliwić przeprowadzenie badań w szerokim zakresie i z wysoką trafnością.

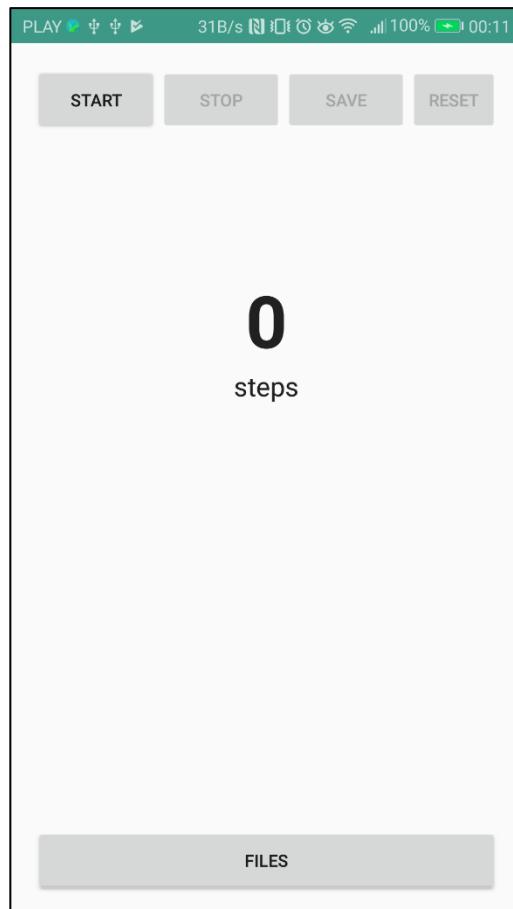
### **2.1. Aplikacja rejestrująca sygnał przyspieszenia**

Celem pozyskania odpowiednich danych konieczne było przygotowanie autorskiej aplikacji, która posłuży do zapisu sygnału rejestrowanego przez akcelerometr do plików CSV. Aplikacja została zaimplementowana na system Android, ze względu na wykorzystane w pracy urządzenie do pomiaru przyspieszenia. Aplikacja miała umożliwiać zarządzanie procesem zbierania próbek sygnału przyspieszenia. Koniecznym było zaimplementowanie mechanizmu, który pozwoli na rozpoczęcie i zatrzymanie procesu rejestrowania sygnału w dowolnym momencie. Aplikacja miała ponadto umożliwić zapisanie aktywności fizycznej do pliku, który w dalszym etapie mogłyby zostać udostępnione do aplikacji zewnętrznych. Celem spełnienia wymagań zaimplementowano aplikacje o nazwie *LifeTrack*. Po wejściu do aplikacji wyświetlna jest główna aktywność, na której użytkownik ma możliwość nawigowania do poszczególnych elementów. Na górze ekranu znajdują się cztery przyciski:

- START,
- STOP,
- SAVE,
- RESET.

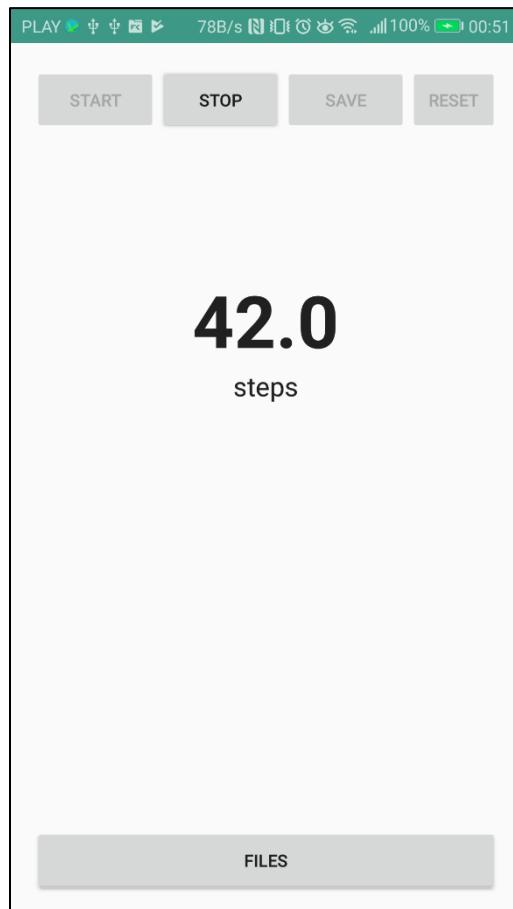
Środek ekranu zajmuje licznik kroków pozwalający na weryfikację działania aplikacji w trakcie pomiarów. Poprawne wyświetlanie i przyrost kroków w trakcie ruchu użytkownika świadczy o poprawnym działaniu.

Dół ekranu zajmowany jest przez przycisk FILES za pomocą którego uzyskujemy dostęp do listy plików. Ekran główny aplikacji przedstawiony został na Rysunku 2.1.



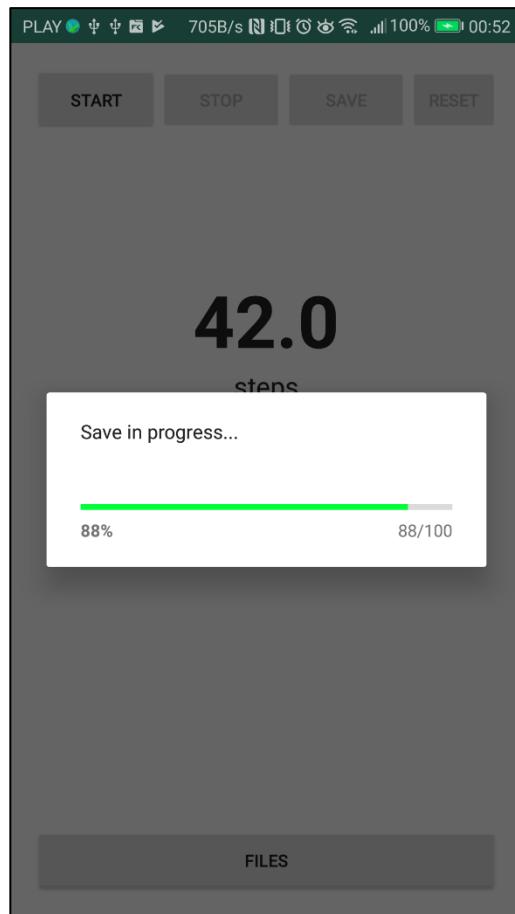
Rysunek 2.1. Ekran aplikacji wyświetlany użytkownikowi bezpośrednio po jej uruchomieniu.

Każdy z górnych przycisków odpowiada za przebieg procesu rejestrowania aktywności fizycznej. Przycisk START rozpoczyna proces rejestrowania sygnału przyspieszenia i bezpośrednio po uruchomieniu jako jedyny ze wskazanej grupy jest aktywny i gotowy do wyboru. Jego wcisnięcie powoduje zbieranie próbek sygnału do pamięci wewnętrznej, aż do momentu wcisnięcia przycisku STOP. Przycisk STOP ma na celu jedynie zatrzymanie procesu. Po jego wyborze możliwe jest wznowienie pomiaru za pomocą ponownego wcisnięcia START. W standardowym, pozytywnym przypadku, po zatrzymaniu pomiarów, wybierany jest przycisk SAVE, który powoduje zapisanie informacji o sygnale do pliku CSV. Zatrzymany przyciskiem STOP pomiar może zostać także zapomniany. W tym celu należy wybrać przycisk RESET, który usuwa zebrane dane w pamięci wewnętrznej urządzenia. Uruchomioną aplikację z zarejestrowanymi krokami przedstawiono na Rysunku 2.2.



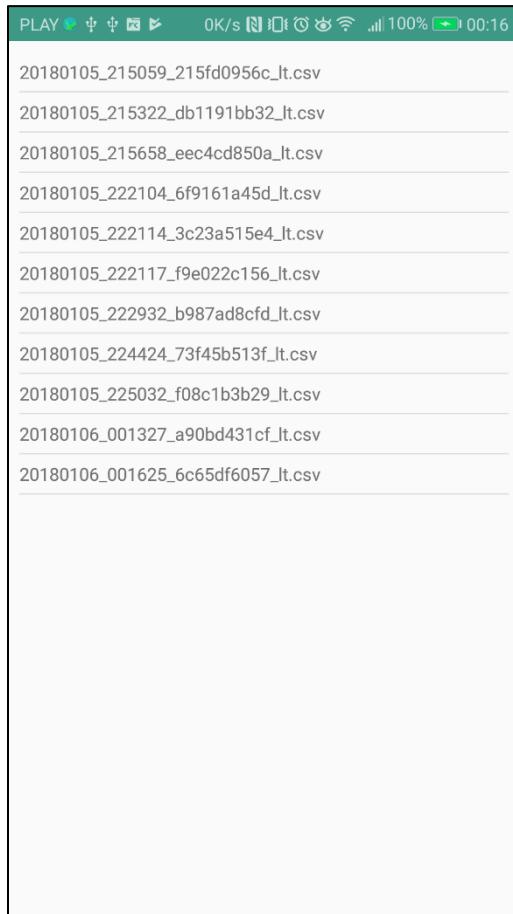
Rysunek 2.2. Ekran aplikacji z uruchomionym rejestraniem sygnału przyspieszenia. Na środku ekranu przedstawiona jest liczba kroków zarejestrowanych w trakcie działania.

Kończąc rejestrowanie ruchu poprzez przycisk STOP, kolejnym krokiem jest zapisanie zgromadzonych danych do pliku. Po wybraniu przycisku SAVE rozpoczęty jest proces zapisu. Jego przebieg obrazuje wyświetlany w trakcie pasek postępu. Ekran przedstawiający zapisywanie zarejestrowanej aktywności przedstawia Rysunek 2.3.



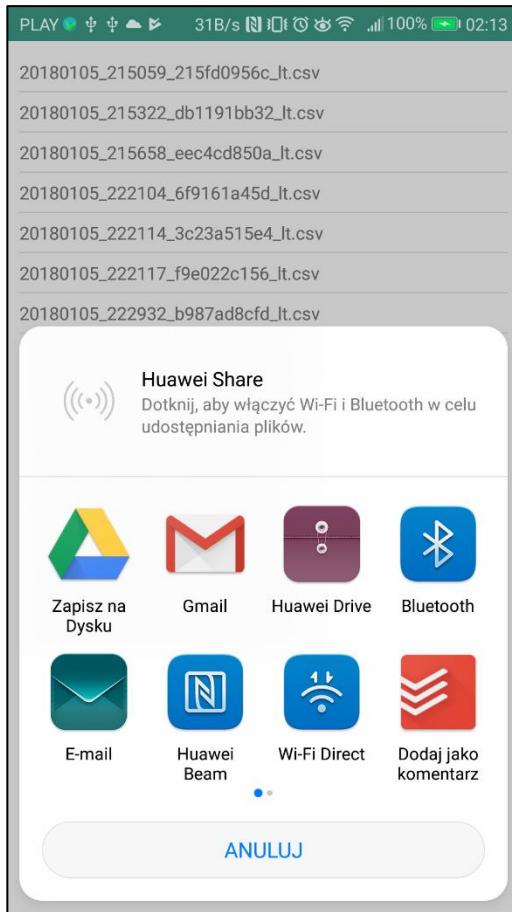
Rysunek 2.3. Ekran aplikacji z uruchomionym zapisywaniem sygnału z akcelerometru. Na ekranie wyświetlony zostaje pasek postępu procesu zapisywania.

Po wykonaniu zapisu aplikacja powraca do stanu z początku uruchomienia. Pliki CSV dostępne są w formie listy, po wcisnięciu przycisku FILES. Listę plików dostępną w aplikacji przedstawia Rysunek 2.4.



Rysunek 2.4 Ekran aplikacji przedstawiający listę zarejestrowanych i zapisanych do pliku aktywności fizycznych użytkownika.

Wyświetlona lista plików uporządkowana jest w porządku alfabetycznym. Jako iż początek nazwy pliku rozpoczyna się od daty i godziny, pliki uporządkowane są chronologicznie. Pierwszy element na górze listy jest plikiem najstarszym, a ostatni - najnowszym. Każdy z plików dostępny jest do udostępnienia z telefonu komórkowego na wybrany zasób. Kliknięcie na plik powoduje wyświetlenie okna udostępniania, zarządzanego przez system Android i tym samym nie będącego integralną częścią aplikacji. Lista dostępnych opcji zależy od modelu, producenta telefonu oraz zainstalowanych aplikacji. Przykładowe okno udostępniania przedstawiono na Rysunku 2.5



Rysunek 2.5. Ekran aplikacji przedstawiający wyświetlony pop-up do udostępniania plików.

Każdy plik z danymi zapisany przez aplikację zapisany jest w formacie CSV. Pliki zawierają dane w prostej formie tekstowej. Jeden wiersz pliku odpowiada pojedynczemu pomiarowi wartości poprzez akcelerometr. W każdym wierszu zapisano kolejno o:

- znaczniku czasowym w milisekundach,
- wartości odczytanej przez akcelerometr w osi X,
- wartości odczytanej przez akcelerometr w osi Y,
- wartości odczytanej przez akcelerometr w osi Z.

Fragment pliku zawierający przykładowy, zarejestrowany sygnał przedstawiono na Listingu 2.1. Zarejestrowane przyspieszenie przedstawione jest w metrach na sekundę do kwadratu - jednostce z układu SI [12].

Listing 2.1. Fragment pliku CSV przedstawiający format danych zarejestrowanych przez akcelerometr.

```
1512819615532, 4.10845, 9.394848, 0.5554548
1512819615553, 4.0222588, 8.90643, 2.0685902
1512819615573, 3.2465374, 9.490616, -0.34476504
1512819615593, 3.5529952, 9.184157, -0.40222588
1512819615612, 3.9552212, 9.356541, 0.31603462
1512819615633, 5.1523223, 9.548077, -0.38307226
1512819615653, 4.3957543, 9.069236, -0.45010993
1512819615673, 3.169923, 9.356541, 0.15322891
1512819615692, 1.934515, 9.356541, 0.21068975
```

Aplikacja *LifeTrack* złożona jest z dwóch ekranów, które odpowiadają dwóm klasom aktywności: *MainActivity* oraz *FilePickActivity*. Pierwsza z nich odpowiada za ekran wyświetlany bezpośrednio po uruchomieniu aplikacji. Klasa *MainActivity* posiada zestaw pól odpowiedzialnych za elementy interfejsu użytkownika, obsługę czujników, krokomierza czy też funkcjonalność zapisu sygnału przyspieszenia do pliku. Komplet pól przedstawia Listing 2.2.

Listing 2.2. Pola klasy *MainActivity* przechowujące dane na temat stanu działania aplikacji.

```
//Managers
private SensorManager sensorManager;
private PowerManager.WakeLock wakeLock;

//UI Elements
private Button buttonStart;
private Button buttonStop;
private Button buttonSave;
private Button buttonReset;
private Button buttonFiles;
private TextView stepCount;

//File saving fields
private File file;
private ProgressDialog progressBar;
private int progressBarStatus = 0;
private Handler progressBarHandler = new Handler();
private static final String fileNameSuffix = "_lt.csv";
private LinkedList<SensorData> collectedData = new LinkedList<>();

//Step fields
private float startStepCount;
private float actualStepCount;
private boolean isFirstStep;
```

Klasa *MainActivity* rozszerza, jak każda klasa aktywności w aplikacji systemu Android klasę *Activity*. Ponadto implementuje dwa interfejsy: *OnClickListener* na potrzebę obsługi przycisków występujących na ekranie oraz *SensorEventListener* który odpowiedzialny jest za obsługę akcelerometru i krokomierza.

Metodą uruchamianą bezpośrednio po starcie aplikacji jest metoda *onCreate*. W pierwszej kolejności wywoływany jest konstruktor klasy nadzędnej oraz ustawiana definicja widoku, który zostanie wyświetlony użytkownikowi aplikacji. Ponadto metoda *onCreate* pobiera usługę systemową do obsługi sensorów. W dalszej części do pól odpowiadających przyciskom oraz do pola tekstowego obrazującego liczbę kroków, przypisywane są wyszukane elementy interfejsu użytkownika. W ostatniej części dla każdego z przycisków ustawiany jest *OnClickListener*, który pozwala obsłużyć akcje wykonywane po wcisnięciu każdego z nich. Następnie ustawiana jest aktywność przycisków. Na tym etapie wyłącznie przycisk start jest aktywny i dostępny do kliknięcia przez użytkownika. Całość metody *onCreate* przedstawiona została na Listingu 2.3.

Listing 2.3. Metoda *onCreate* wywoływana podczas uruchamiania aplikacji.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```

sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
buttonStart = findViewById(R.id.buttonStart);
buttonStop = findViewById(R.id.buttonStop);
buttonSave = findViewById(R.id.buttonSave);
buttonReset = findViewById(R.id.buttonReset);
buttonFiles = findViewById(R.id.buttonFiles);
stepCount = findViewById(R.id.stepCount);

buttonStart.setOnClickListener(this);
buttonStop.setOnClickListener(this);
buttonSave.setOnClickListener(this);
buttonReset.setOnClickListener(this);
buttonFiles.setOnClickListener(this);

buttonStart.setEnabled(true);
buttonStop.setEnabled(false);
buttonSave.setEnabled(false);
buttonReset.setEnabled(false);
buttonFiles.setEnabled(true);
}

```

Kolejnym elementem klasy jest metoda odpowiedzialna za obsługę akcji wykonywanych po wybraniu poszczególnych przycisków. Metoda zawiera instrukcję *switch* delegującą obsługę do konkretnych metod zaimplementowanych pod kątem wykonywania operacji po naciśnięciu każdego z przycisków. Metody te zostaną szczegółowo omówione w dalszej części pracy. Poniżej Listing 2.4 przedstawiający metodę *onClick*.

**Listing 2.4.** Metoda *onClick* z zaimplementowanego interfejsu *OnClickListener* odpowiedzialna za obsługę przycisków na ekranie głównym.

```

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.buttonStart:
            buttonStartPressed();
            break;
        case R.id.buttonStop:
            buttonStopPressed();
            break;
        case R.id.buttonSave:
            buttonSavePressed();
            break;
        case R.id.buttonReset:
            buttonResetPressed();
            break;
        case R.id.buttonFiles:
            buttonFilesPressed();
            break;
        default:
            break;
    }
}

```

Pierwszym przyciskiem, którego obsługa wywoływana jest przez metodę `onClick`, jest przycisk START. Za operacje wykonywane po jego wyborze odpowiada implementacja `buttonStartPressed`. Ciało tej metody ustawia aktywność przycisków w sposób by aktywny był wyłącznie przycisk STOP. Inicjalizowana jest także obsługa sensorów oraz zmienne z nimi związane. W szczególności resetowana jest liczba kroków wyświetlana na ekranie aplikacji. Omawianą metodę przedstawia Listing 2.5.

Listing 2.5. Metoda `buttonStartPressed` odpowiedzialna za wykonanie akcji po wybraniu przycisku START.

```
private void buttonStartPressed() {
    buttonStart.setEnabled(false);
    buttonStop.setEnabled(true);
    buttonSave.setEnabled(false);
    buttonReset.setEnabled(false);
    wakeLock = createWakeLock();
    Sensor accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    Sensor stepCounter =
sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    isFirstStep = true;
    actualStepCount = 0.0f;
    sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_GAME);
    sensorManager.registerListener(this, stepCounter,
SensorManager.SENSOR_DELAY_NORMAL);
    file = new File.getExternalStorageDir(null), generateFileName());
    return;
}
```

W metodzie `buttonStartPressed` ustawiany jest także *Partial Wake Lock*. Jego ustawienie powoduje kontynuację pracy procesora na rzecz aplikacji, nawet w momencie, kiedy wyświetlacz zostanie wyłączony. Na potrzeby obsługi tego mechanizmu konieczne jest dodanie odpowiedniego wpisu w pliku `AndroidManifest.xml`, zezwalającego aplikacji na jego wykorzystanie. Dodatkowo na tę potrzebę, w kodzie Java, wydzielona została metoda `createWakeLock`. Metodę przedstawiono na poniższym Listingu 2.6.

Listing 2.6. Metoda `createWakeLock` odpowiedzialna za umożliwienie pracy aplikacji przy wyłączeniu wyświetlacza urządzenia.

```
private PowerManager.WakeLock createWakeLock() {
    PowerManager pm = (PowerManager)
getSystemService(Context.POWER_SERVICE);
    PowerManager.WakeLock wakeLock =
pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "accel_wake_lock");
    wakeLock.acquire();
    return wakeLock;
}
```

Drugim z przycisków jest przycisk STOP, który zatrzymuje proces rejestrowania aktywności fizycznej. W metodzie `buttonStopPressed`, która obsługuje zdarzenie po wciśnięciu przycisku zdejmowany jest wcześniej opisany Wake Lock. Ponadto modyfikowana jest dostępność pozostałych opcji. Wciśnięcie stopu powoduje, że wszystkie przyciski poza nim samym stają się aktywne. Aktywność przycisków jest modyfikowana także w przypadku wyboru przycisku RESET. Różnica w działaniu polega na tym, iż wciskając przycisk odpowiedzialny

za zatrzymanie pomiaru istnieje możliwość na jego wznowienie bez utraty wcześniejszej zarejestrowanych, niezapisanych informacji o sygnale przyspieszenia. Przycisk RESET w przeciwnieństwie do STOP czyści zawartość obiektu przechowującego próbki z akcelerometru. Resetując pomiar tracimy bezpowrotnie możliwość zapisu zarejestrowanego sygnału.

Ostatnim z przycisków dostępnych do nawigowania po procesie rejestracji przyspieszenia jest przycisk SAVE odpowiedzialny za zapis zebranych danych. Kod aplikacji dla wyboru tego przycisku przedstawiony został na Listingu 2.7.

Listing 2.7. Metoda *buttonSavePressed* odpowiedzialna za wykonanie akcji po wybraniu przycisku SAVE.

```
private void buttonSavePressed() {
    buttonStart.setEnabled(true);
    buttonStop.setEnabled(false);
    buttonSave.setEnabled(false);
    buttonReset.setEnabled(false);
    saveCollectedData();
}
```

Dostępność przycisków po wciśnięciu SAVE jest identyczna jak w przypadku inicjalnego stanu aplikacji, dostępnego bezpośrednio po jej uruchomieniu. Kluczowe zadania wykonywane w tym wypadku wydzielono jednak do metody *saveCollectedData*, która zależna jest od kolejnych metod prywatnych. Najważniejsze z nich przedstawiono na Listingu 2.8.

Listing 2.8. Metody odpowiedzialne za czynności związane z zapisem sygnału przyspieszenia zarejestrowanego przez aplikację do pliku CSV.

```
private void saveCollectedData() {
    prepareProgressBar();
    progressBar.show();

    new Thread(new Runnable() {
        public void run() {
            file = new File(getExternalFilesDir(null), generateFileName());
            try {
                saveActivityFile();
            } catch (InterruptedException e) {
                Log.e("MainActivity", "The thread is interrupted: " + e.getMessage());
            } catch (IOException e) {
                Log.e("MainActivity", "The file can't be created: " + e.getMessage());
            } catch (Exception e) {
                Log.e("MainActivity", "Unexpected exception " + e.getMessage());
            }
            progressBar.dismiss();
        }
    }).start();
}

private void saveActivityFile() throws IOException, InterruptedException {
    double dataSize = collectedData.size();
    double progress = 1;
    for(SensorData sd : collectedData) {
        progressBarStatus = (int) (progress / dataSize * 100);
        progress++;
    }
}
```

```

        Files.append(sd.toString(), file, Charset.defaultCharset());

    progressBarHandler.post(new Runnable() {
        public void run() {
            progressBar.setProgress(progressBarStatus);
        }
    });
}
Thread.sleep(1000); //sleep at the end of saving
}

private void prepareProgressBar() {
    progressBar = new ProgressDialog(this);
    progressBar.setCancelable(true);
    progressBar.setMessage("Save in progress...");
    progressBar.setStyleProgressDialog().STYLE_HORIZONTAL);
    progressBar.setProgress(0);
    progressBar.setMax(100);
    progressBarStatus = 0;
}

@NotNull
private String generateFileName() {
    String date = new SimpleDateFormat("yyyyMMdd_HH:mm:ss").format(new Date());
    date += "_" + UUID.randomUUID().toString().replace("-", "");
    return date + fileNameSuffix;
}

```

Ciąg wywołań metod przedstawionych na Listingu 2.8 rozpoczyna się od przygotowania i wyświetlenia paska postępu. Pasek postępu prezentuje wartości od 0 do 100 co odpowiada procentowemu postępowi zapisu danych do pliku. W kolejnym kroku generowana jest nazwa pliku do której zarejestrowany sygnał aktywności fizycznej zostanie zapisany. Na tym etapie użytkownik nie ma możliwości ustalenia własnej nazwy pliku. Wygenerowana nazwa rozpoczyna się od ciągu cyfr w formacie yyyyMMdd\_HH:mm:ss, gdzie yyyy oznacza rok, MM - numer miesiąca, dd - numer dnia w miesiącu, HH - godzinę w formacie 24h, mm - minuty oraz ss - sekundy. Do znacznika czasu doklejany jest losowy ciąg dziesięciu znaków który zapewnia unikalność nazw plików. Do nazwy na końcu dołączany jest także sufiks „\_lt.csv”, który wskazuje na pochodzenie pliku z aplikacji *LifeTrack*, oraz definiuje format danych.

Po wygenerowaniu nazwy pliku, w którym zostanie zapisany sygnał z akcelerometru, rozpoczyna się proces zapisu. Odpowiada za niego metoda *saveActivityFile*. Jej działanie opiera się na dopisywaniu kolejnych, zarejestrowanych rekordów do pliku, aktualizując jednocześnie pasek postępu. Kiedy proces dobiegnie końca aplikacja przez dodatkową sekundę wyświetla zakończony pasek postępu, po czym jest zamknięty.

Istotnym elementem aktywności jest także metoda *onSensorChanged* obsługująca zmiany wskazań akcelerometru. Jej wywołanie następuje podczas każdego wykrycia przez system zmiany wskazań zarejestrowanych przez obserwowany sensor. Metoda *onSensorChanged* pochodzi z interfejsu *SensorEventListener* i została przedstawiona na Listingu 2.9.

Listing 2.9. Metoda *onSensorChanged* obsługująca zdarzenia związane z wykorzystywany sensorami.

```
@Override
public void onSensorChanged(SensorEvent event) {
    Sensor sensor = event.sensor;
    if(sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float x = event.values[0]; // Acceleration force along the x axis
        m/s^2
        float y = event.values[1]; // Acceleration force along the y axis
        m/s^2
        float z = event.values[2]; // Acceleration force along the z axis
        m/s^2
        collectedData.add(new SensorData(x, y, z, actualStepCount));
    }
    else if(sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
        if(isFirstStep) {
            startStepCount = (int) event.values[0];
            isFirstStep = false;
        }
        actualStepCount = event.values[0] - startStepCount;
        stepCount.setText(String.valueOf(actualStepCount));
    }
}
```

Metoda *onSensorChanged* obsługuje dwa sensory, których aktywność rozróżniana jest po typie zdarzenia obsłużonym w głównej instrukcji warunkowej *if*. W pierwszym wypadku, kiedy zdarzenie pochodzi z akcelerometru pobierane są próbki sygnału przyspieszenia wzdłuż trzech osi. Na podstawie danych tworzony jest nowy obiekt *SensorData*, który dodawany jest do kolekcji *collectedData*. Kolekcja *collectedData* podlega zapisywaniu do pliku w metodzie *saveActivityFile*. Drugim zdarzeniem obsługiwanym przez metodę jest zmiana liczby kroków wskazywana przez licznik. Licznik kroków dostarczany przez system, zlicza kroki od momentu uruchomienia telefonu. Chcąc wyświetlić w aplikacji wartość uwzględniającą jedynie kroki od momentu rozpoczęcia pomiaru, sprawdzane jest czy zmiana licznika nastąpiła pierwszy raz podczas tego procesu. Jeżeli tak, zapisywana jest wartość początkowa. Następnie, już dla każdego kroku (zarówno pierwszego, jak i każdego następnego), aktualna wartość liczby kroków zarejestrowanych w trakcie działania aplikacji obliczana jest na podstawie odejmowania od liczby aktualnej od startu systemu, liczby zarejestrowanej na starcie aplikacji. Tak obliczona liczba kroków zostaje wyświetlona w aplikacji, w elemencie przedstawionym na środku ekranu głównego.

Na ekranie głównej aktywności znajduje się ponadto przycisk FILES który powoduje przekierowanie do aktywności przedstawiającej listę zapisanych plików z zarejestrowanymi aktywnościami. Klasa tej aktywności *FilePickActivity* zawiera dwie metody, w tym główną metodę *onCreate* przedstawioną na Listingu 2.10.

Listing 2.10. Metoda *onCreate* klasy *FilePickActivity* wywoływana podczas przechodzenia do ekranu z listą plików.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_file_pick);
    shareIntent = new Intent(Intent.ACTION_SEND);
```

```

fileListView = findViewById(R.id.fileList);

filesDir = getExternalFilesDir(null);
files = filesDir.listFiles();

for (File file : files) {
    listItems.add(file.getName());
    filenamePathMap.put(file.getName(), file.getAbsolutePath());
}
Collections.sort(listItems);
fileListViewAdapter = new ArrayAdapter<String>(this,
R.layout.file_list_item, listItems);
fileListView.setAdapter(fileListViewAdapter);
fileListView.setClickable(true);
fileListView.setOnItemClickListener(
    new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView, View
view, int position, long rowId) {
            File requestFile = new
File(filenamePathMap.get(listItems.get(position)));
            try {
                Uri fileUri =
FileProvider.getUriForFile(FilePickActivity.this, "pl.gebert.lifetrack.FileP
rovider", requestFile);
                if (fileUri != null) {
                    shareIntent = createFileShareIntent(fileUri);
                    startActivity(Intent.createChooser(shareIntent,
"Share file"));
                }
                FilePickActivity.this.setResult(Activity.RESULT_OK, shareIntent);
            } else {
                shareIntent.setDataAndType(null, "");
            }
            FilePickActivity.this.setResult(Activity.RESULT_CANCELED, shareIntent);
        }
    } catch (IllegalArgumentException e) {
        Log.e("FilePickActivity",
"The selected file can't be shared: " +
e.getMessage());
    } catch (Exception e) {
        Log.e("FilePickActivity",
"Unexpected exception: " + e.getMessage());
    }
}
});
}

```

Metoda `onCreate` klasy `FilePickActivity`, przedstawiona na Listingu 2.10 uzyskuje dostęp do plików zapisanych w zewnętrznym katalogu danych aplikacji. Zapisywane są tam wszystkie sygnały aktywności fizycznych zarejestrowanych przez aplikację. Po uzyskaniu dostępu, pobierane są wszystkie pliki znajdujące się w katalogu. Aplikacja nie zapisuje nic poza plikami

z sygnałem pochodzącym z akcelerometru, więc nie ma potrzeby filtrowania pobieranych plików. Dla każdego z nich pobierana jest jego nazwa i ścieżka, pod którą się znajduje. Tak pobrana lista plików prezentowana jest w aplikacji, która wyświetla wyłącznie nazwę. Pliki posortowane są w kolejności alfabetycznej, co z racji na format ich nazw jest także kolejnością chronologiczną. Każdy z rekordów znajdujący się na liście ma także utworzony listener reagujący na wybór z poziomu aplikacji. W aplikacji umożliwiono bowiem funkcję udostępnienia danych na zewnątrz. Element odpowiedzialny za rozpoczęcie procesu udostępniania zwracany jest przez drugą z metod klasy *FilePickActivity*, przedstawioną na Listingu 2.11.

Listing 2.11. Metoda *createFileShareIntent* odpowiedzialna za proces udostępniania pliku.

```
private Intent createFileShareIntent(Uri fileUri) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    intent.setDataAndType(fileUri, getContentResolver().getType(fileUri));
    intent.putExtra(Intent.EXTRA_STREAM, fileUri);
    return intent;
}
```

Możliwość udostępniania dostarczana jest bezpośrednio przez system. Dlatego też dostępne jest wiele możliwości przekazania pliku, bez konieczności przeprowadzania procesu implementacji dla każdej z metod. Podstawowymi opcjami jest tutaj wysyłka pliku mailem oraz jego wysyłka na dysk Google Drive. Dostępnych jest jednak znacznie więcej możliwości, a ich liczba zależna jest od aplikacji wspierających, zainstalowanych na telefonie.

W aplikacji zaimplementowano ponadto klasę *SensorData*, niebędącą aktywnością. Klasa *SensorData* przechowuje dane pojedynczej próbki sygnału. Zawiera 5 pól które zawierają informacje o wartości pomiaru w każdej z osi, aktualnym czasie w milisekundach, oraz o liczbie wykonanych kroków. Klasa ma zdefiniowany konstruktor oraz metodę *toString*. Wynik wywołania metody *toString* odzwierciedla format zapisu sygnału do plików. Chcąc zmienić format danych należy zmodyfikować metodę *toString*, która ma bezpośrednie przełożenie na zapis. Opisywaną klasę *SensorData* przedstawia Listing 2.12.

Listing 2.12. Klasa *SensorData* przechowująca pojedynczy pomiar akcelerometru.

```
public class SensorData {
    float x;
    float y;
    float z;
    float step;
    final long time = System.currentTimeMillis();

    public SensorData(float x, float y, float z, float step) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.step = step;
    }

    public String toString() {
        return time + ", " + x + ", " + y + ", " + z + "\n";
    }
}
```

## 2.2. Zarejestrowane sygnały

Sygnały przyspieszenia zgromadzone za pomocą opisanej wcześniej aplikacji pochodzą od 4 osób w różnym wieku. Dla każdej z osób telefon znajdował się w prawej kieszeni spodni, odwrócony wyświetlaczem do zewnętrz. W przypadku osoby pierwszej telefon znajdował się w stosunkowo luźnej kieszeni. Pozostałe osoby miały telefon umieszczony nieruchomo, tak iż nie działały na niego dodatkowe siły bezwładności, które mogły wpływać na wyniki klasyfikacji. Poniższa tabela przedstawia zestawienie kompletnych danych, zebranych celem późniejszej analizy. Jeden wiersz odpowiada jednemu plikowi zawierającemu pojedynczą aktywność fizyczną dla poszczególnych osób. W Tabeli 2.1 przedstawiona została godzina rozpoczęcia i zakończenia zbierania próbek sygnału przyspieszenia oraz obliczony na tej podstawie czas trwania pomiaru.

Tabela 2.1. Zestawienie zarejestrowanych danych o aktywności fizycznych użytkowników wraz z czasami ich trwania.

Osoba	Aktywność fizyczna	Czas trwania
OSOBA_1	wchodzenie po schodach	00:01:10
OSOBA_1	schodzenie ze schodów	00:01:15
OSOBA_1	bezczytność	00:01:18
OSOBA_1	bieганie	00:01:23
OSOBA_1	chodzenie	00:01:17
OSOBA_2	schodzenie ze schodów	00:00:56
OSOBA_2	wchodzenie po schodach	00:00:57
OSOBA_2	bezczytność	00:01:21
OSOBA_2	bieganie	00:01:50
OSOBA_2	chodzenie	00:03:40
OSOBA_3	schodzenie ze schodów	00:01:02
OSOBA_3	wchodzenie po schodach	00:01:06
OSOBA_3	bezczytność	00:01:17
OSOBA_3	bieganie	00:01:54
OSOBA_3	chodzenie	00:03:16
OSOBA_4	wchodzenie po schodach	00:01:02
OSOBA_4	schodzenie ze schodów	00:01:03
OSOBA_4	bezczytność	00:01:17
OSOBA_4	bieganie	00:01:35
OSOBA_4	chodzenie	00:03:23

Na potrzeby prezentacji działania klasyfikatora przy rozpoznawaniu aktywności fizycznej użytkownika w ciągu dnia zarejestrowano dodatkowy plik z sygnałem przyspieszenia dla osoby trzeciej. Dane w nim przedstawiają zróżnicowaną aktywność fizyczną zarejestrowaną podczas krótkiego czasu. Aktywność dotyczy spaceru, który rozpoczął się od wejścia po schodach. Kolejno osoba rejestrująca sygnał zatrzymała się celem założenia butów i okrycia wierzchniego. Następnie schodząc po dwóch stopniach rozpoczął się krótki spacer,

bez zatrzymywania. W drodze powrotnej osoba wykonała aktywności poprzedzające spacer w odwrotnej kolejności. Sygnał został zebrany przez tę samą osobę która w zestawieniu prezentowana jest jako OSOBA\_3. W Tabeli 2.2 przedstawiono krótki opis zarejestrowanego sygnału.

Tabela 2.2. Opis mieszanej aktywności fizycznej zarejestrowanej przez osobę trzecią.

Osoba	Aktywność fizyczna	Czas trwania
OSOBA_3	mieszana	00:10:25

Sygnały zostały zapisane w plikach odpowiadających nazwom aktywności fizycznych oraz umieszczone w projekcie posegregowane według poszczególnych osób. Zapis sygnału dla każdej osoby odbywał się z prędkością 50 próbek przyspieszenia na sekundę.

Osoby biorące udział w badaniu były zróżnicowane pod kątem wieku i płci. Każda z nich korzystała z tego samego aparatu telefonicznego na którym uruchomiona była aplikacja rejestrująca sygnał. Charakterystykę tych osób przedstawia poniższa Tabela 2.3.

Tabela 2.3. Zestawienie osób biorących udział w procesie zbierania aktywności.

Osoba	Płeć	Wiek
OSOBA_1	Mężczyzna	26 lat
OSOBA_2	Kobieta	23 lata
OSOBA_3	Kobieta	46 lat
OSOBA_4	Mężczyzna	56 lat

Sygały przyspieszenia zarejestrowane przez osoby 2-4 dotyczące chodzenia po schodach zostały zarejestrowane w tym samym miejscu. Pozostałe aktywności fizyczne zostały zarejestrowane w różnych lokalizacjach tak by dane były zróżnicowane zarówno pod względem osób je rejestrujących, jak i terenu, w którym dokonywany był pomiar.

Ostatecznie zestaw sygnałów przyspieszenia zebranych z akcelerometru objął 20 plików CSV zawierających dane o pojedynczej aktywności fizycznej osoby oraz 1 plik CSV dotyczący wielu klas aktywności fizycznych.

### 3. Przetwarzanie sygnału z akcelerometru

Po pozyskaniu sygnałów przyspieszenia przez opisaną w poprzednim rozdziale aplikację na telefon z systemem Android rozpoczęto proces analizy. W tym celu załadowano dane z plików, a następnie zostały one poddane procesom mającym na celu przygotowanie do potrzeb nauki klasyfikatora [13]. Analiza została wykonana z wykorzystaniem języka Python i została przeprowadzona wyłącznie w dziedzinie czasu. Analiza w dziedzinie czasu przynosi bowiem najlepsze rezultaty przy rozpoznawaniu aktywności fizycznej na podstawie przyspieszenia [14].

#### 3.1. Załadowanie danych

W pierwszym kroku przygotowano metodę `read(filename)` pozwalającą na odczyt danych z plików CSV, która jest przystosowana do formatu zapisu sygnału przez opisaną wcześniej aplikację. Metoda napisana jest tak by pomijała pierwsze i ostatnie 300 próbek. Przy zastosowanej częstotliwości pomiaru 300 próbek odpowiada około 6 sekundom na początku i końcu sygnału. Czas na początku został uznany za niezbędny na uruchomienie aplikacji i umieszczenie telefonu w miejscu, w którym powinien się znajdować w trakcie pomiaru. Z drugiej strony czas na końcu pomiaru wykorzystywany jest na wyjęcie urządzenia i zatrzymanie pomiaru aplikacji. Metodę `read(filename)` przedstawia Listing 3.1.

Listing 3.1. Metoda `read(filename)` z pliku `readrawdata.py` zaimplementowana do odczytu danych z plików CSV wygenerowanych przez aplikację.

```
import pandas as pd

def read(filename):
    columns = ['timestamp', 'xAxis', 'yAxis', 'zAxis']
    data = pd.read_csv(filename, skiprows=300, skipfooter=300, header=None,
names=columns)
    return data
```

Kolejnym utworzonym elementem zostały metody `plot_graph` oraz `graph_activity` odpowiedzialne za rysowanie wykresów przedstawiających zarejestrowany sygnał z akcelerometru.

Pierwsza z nich odpowiada za wyrysowanie pojedynczego wykresu w obszarze przekazanym jako pierwszy argument. Na wykresie znajdują się dane odzwierciedlające informacje przekazane w parametrach `xAxis` oraz `yAxis`. Dodatkowo metoda ustawia przekazany tytuł pojedynczego wykresu oraz kolor w jakim przedstawiony zostanie zarejestrowany sygnał.

Druga metoda wykorzystywana jest bezpośrednio do rysowania wykresu aktywności fizycznych na podstawie zebranych sygnałów które przekazujemy w pierwszym parametrze. Wizualizacja wartości składa się z trzech wykresów obrazujących:

- odchylenie wzduż osi x,
- odchylenie wzduż osi y,

- odchylenie wzdłuż osi z.

Każda z danych przedstawiona jest w dziedzinie czasu, która oparta jest na parametrze `timestamp` z wczytanego pliku. Ponadto metoda wykorzystuje kolor podany w parametrze do przedstawienia sygnału na wszystkich trzech wykresach oraz ustawia ogólny tytuł. Zawartość metod przedstawiono na Listingu 3.2.

Listing 3.2. Metody `plot_graph` oraz `graph_activity` z pliku `visualize.py` zaimplementowana do odczytu danych z plików CSV wygenerowanych przez aplikację.

```
import matplotlib.pyplot as plt
import mgr.calc.signal as sig

def graph_activity(activity, color, title):
    """Prints a graph of passed activity. There are 3 plots for each
    axis"""
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(15, 10),
sharex=True)
    plot_graph(ax0, activity['timestamp'], activity['xAxis'], 'X-axis',
color)
    plot_graph(ax1, activity['timestamp'], activity['yAxis'], 'Y-axis',
color)
    plot_graph(ax2, activity['timestamp'], activity['zAxis'], 'Z-axis',
color)
    plt.suptitle(title)
    plt.show()

def plot_graph(axis, xAxis, yAxis, title, color):
    """Function for single graph plotting"""
    axis.plot(xAxis, yAxis, color=color)
    axis.set_title(title)
    axis.xaxis.set_visible(False)
    axis.set_xlim([min(xAxis), max(xAxis)])
    axis.grid(True)
```

Metody `plot_graph` oraz `graph_activity` pozwoliły na zwizualizowanie sygnałów 5 aktywności fizycznych, które w późniejszym etapie zostały wykorzystane do nauki klasyfikatora. Każda z aktywności: bezczynność, chodzenie, wchodzenie po schodach, schodzenie ze schodów oraz bieganie, została przedstawiona innym kolorem na poszczególnych wykresach. Kolory zostały zdefiniowane w zmiennych do późniejszego wykorzystania i oznaczania sygnałów odpowiednimi kolorami. Kod w języku Python, który posłużył do wygenerowania wykresów przedstawia Listing 3.3.

Listing 3.3. Kod programu w pliku `main.py` odpowiedzialny za zobrazowanie sygnału z akcelerometru dla wszystkich pięciu aktywności fizycznych.

```
STANDING_COLOR = 'gray'
WALKING_COLOR = 'green'
DOWNSTAIRS_COLOR = 'brown'
```

```

UPSTAIRS_COLOR = 'blue'
RUNNING_COLOR = 'red'

#Read standing activity data from csv and draw a graph
STANDING = rd.read('mgr/data/resources/person_1/standing.csv')
vis.graph_activity(STANDING, STANDING_COLOR, 'Standing')

#Read walking activity data from csv and draw a graph
WALKING = rd.read('mgr/data/resources/person_1/walking.csv')
vis.graph_activity(WALKING, WALKING_COLOR, 'Walking')

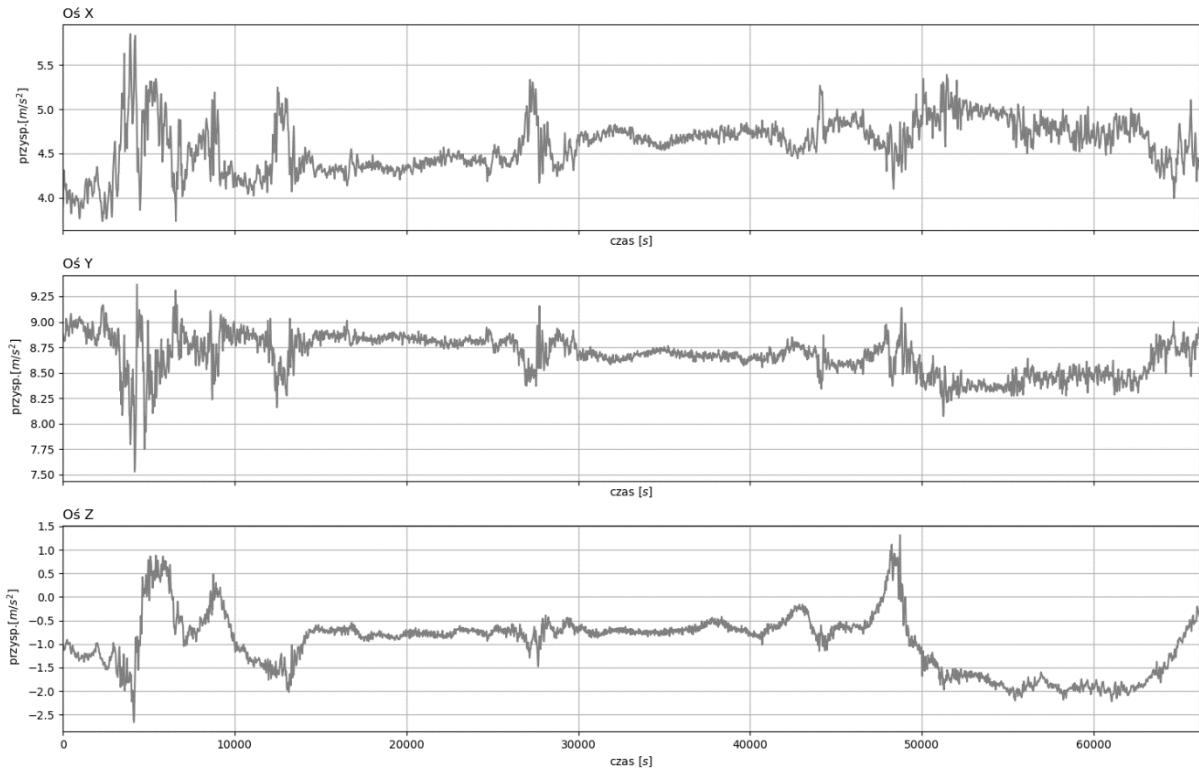
#Read downstairs activity data from csv and draw a graph
DOWNSTAIRS = rd.read('mgr/data/resources/person_1/downstairs.csv')
vis.graph_activity(DOWNSTAIRS, DOWNSTAIRS_COLOR, 'Downstairs')

#Read upstairs activity data from csv and draw a graph
UPSTAIRS = rd.read('mgr/data/resources/person_1/upstairs.csv')
vis.graph_activity(UPSTAIRS, UPSTAIRS_COLOR, 'Upstairs')

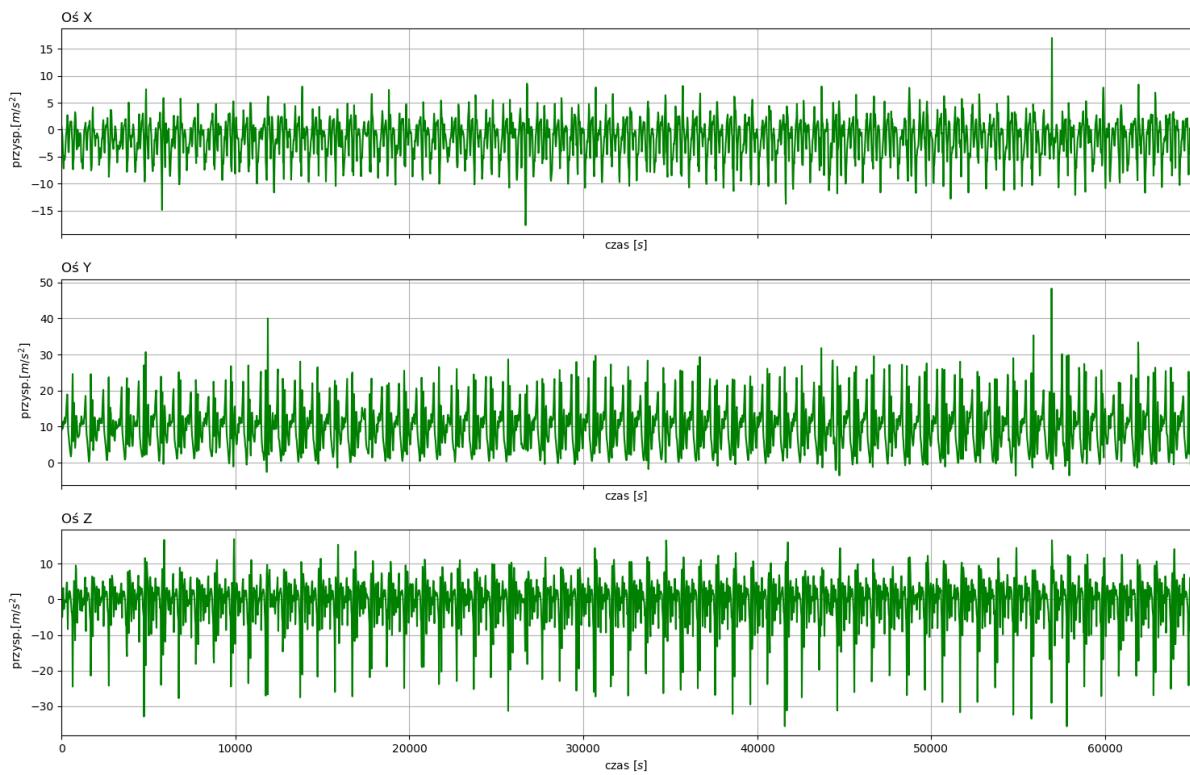
#Read running activity data from csv and draw a graph
RUNNING = rd.read('mgr/data/resources/person_1/running.csv')
vis.graph_activity(RUNNING, RUNNING_COLOR, 'Running')

```

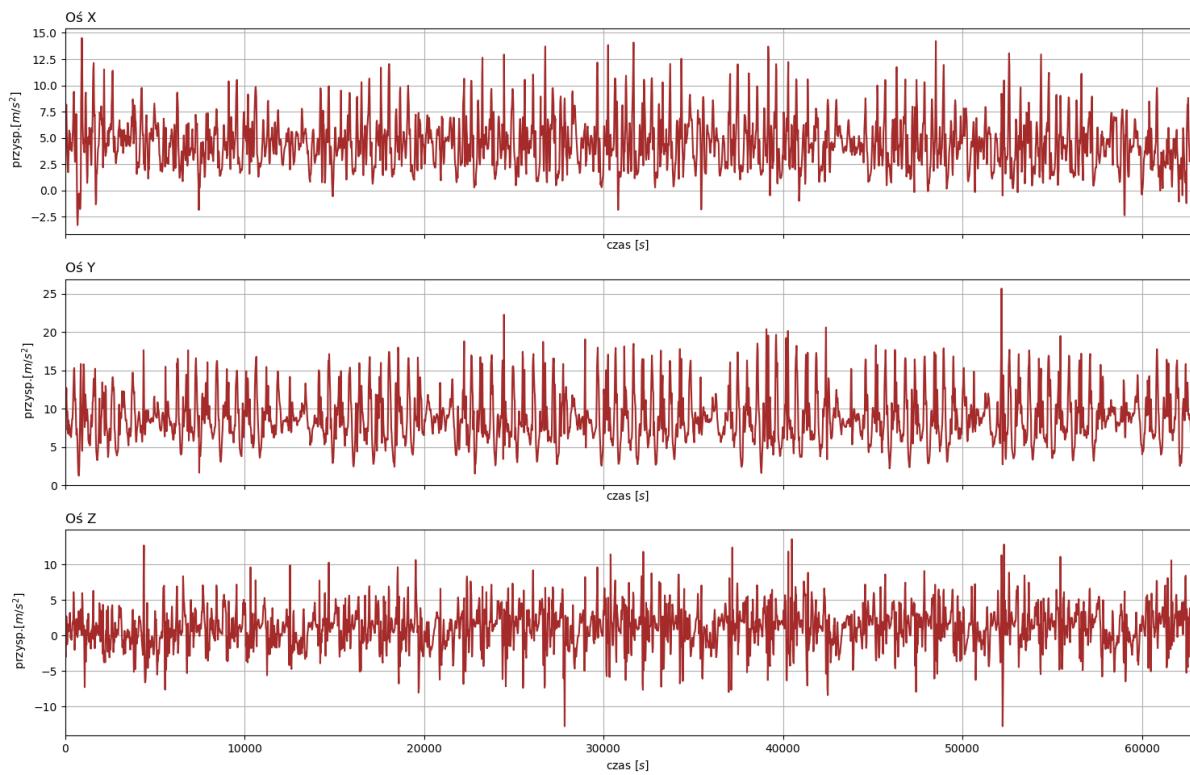
Wynik wykonania Listingu 3.3 przedstawiają Rysunki 3.1 – 3.5. Każdy z nich zawiera zobrazowane na 3 wykresach wskazania akcelerometru. Jeden wykres odpowiada jednej osi w przestrzeni trójwymiarowej.



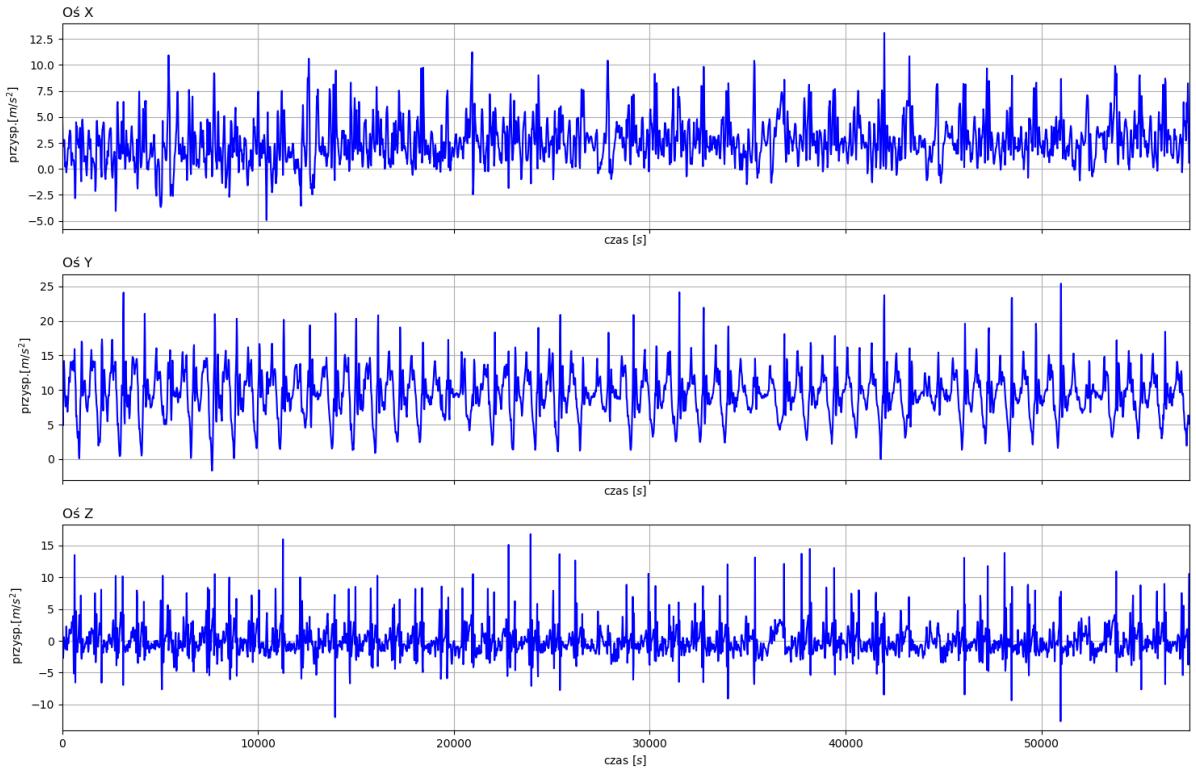
Rysunek 3.1. Sygnał z pliku standing.csv przedstawiające aktywność akcelerometru podczas bezczynności.



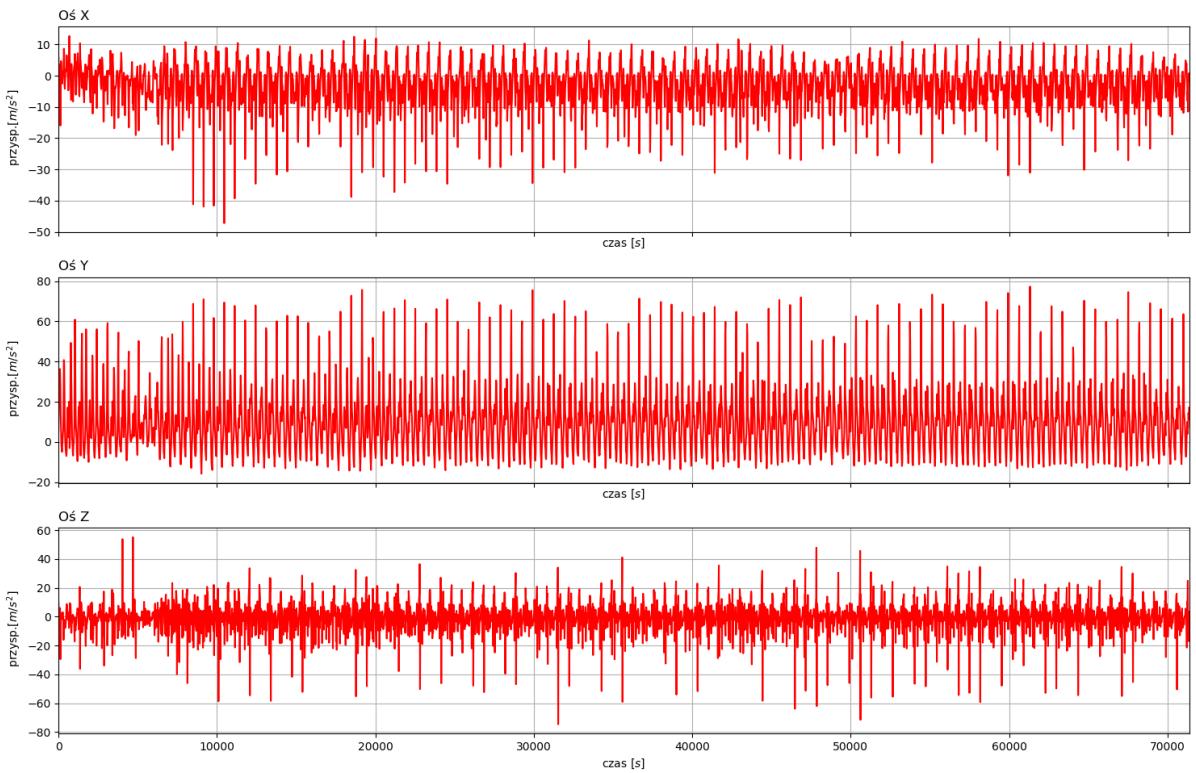
Rysunek 3.2. Sygnał z pliku walking.csv przedstawiające aktywność akcelerometru podczas chodzenia.



Rysunek 3.3. Sygnał z pliku downstairs.csv przedstawiające aktywność akcelerometru podczas schodzenia po schodach.



Rysunek 3.4. Sygnał z pliku upstairs.csv przedstawiający aktywność akcelerometru podczas wchodzenia po schodach.



Rysunek 3.5. Sygnał z pliku running.csv przedstawiający aktywność akcelerometru podczas biegania.

### 3.2. Obliczanie modułu wektora przyspieszenia

Po załadowaniu danych wykonano obliczenie mające na celu uzyskanie wartości modułu wektora przyspieszenia. Wartość modułu jest niezależna od położenia telefonu, a tym samym akcelerometru wykorzystywanego przez aplikację do zbierania sygnału przyspieszenia. Przeciwna sytuacja występuje przy wykorzystaniu pomiarów w trzech osiach. Orientacja w przestrzeni czujnika wpływa wtedy wyraźnie na pomiary. Położenie czujnika ma znaczenie w dwóch aspektach. Po pierwsze znak liczby reprezentującej pomiar zmienia się w zależności od niego. Trzymając telefon pionowo, w położeniu wykorzystywanym do standardowego użytku wskazanie osi y będzie dodatnie. Obracając go o 180 stopni, tak by wyświetlony obraz był odwrócony i skierowany do użytkownika, znak wartości zmieni się na ujemny. Ponadto na akcelerometr działa siła związana z przyspieszeniem ziemskim, którą czujnik odnotowuje w swoich wskazaniach. Każde pochylenie telefonu zmienia kierunek i zwrot działania siły, a tym samym do poszczególnych wartości dodawane są składowe związane z grawitacją. Wartość obliczanego wektora przyspieszenia dla nieruchomego telefonu wynosić będzie zatem zawsze tyle ile wartość przyspieszenia ziemskiego, niezależnie od położenia [15].

Celem wykonania obliczeń zaimplementowano funkcję *magnitude*, która zwraca wartość wektora przyspieszenia. Kod metody *magnitude* przedstawia Listing 3.4.

Listing 3.4. Kod funkcji *magnitude* z pliku signal.py, obliczającej wartość wektora przyspieszenia.

```
def magnitude(activity):
    res = np.sqrt(activity['xAxis'] * activity['xAxis'] +
                  activity['yAxis'] * activity['yAxis'] +
                  activity['zAxis'] * activity['zAxis'])
    return res
```

Po zaimplementowaniu funkcji *magnitude* dodano także funkcję odpowiedzialną za rysowanie wykresu wartości wektora wychylenia akcelerometru. Metoda *graph\_magnitude* kreśli na pojedynczym obszarze wykres wartości w zależności od czasu. Na wejściu przyjmuje obiekt aktywności fizycznej, kolor wykresu oraz jego tytuł. Opisywaną metodę *graph\_magnitude* przedstawia Listing 3.5.

Listing 3.5. Kod funkcji *graph\_magnitude* z pliku visualize.py odpowiedzialnej za sporządzenie wykresu przedstawiającego wartość wektora wychylenia.

```
def graph_magnitude(activity, color, title):
    """Prints a activity magnitude graph"""
    fig, axs = plt.subplots(nrows=1, figsize=(15, 10), sharex=True)
    plot_graph(axs, activity['timestamp'], activity['magnitude'], title,
               color)
    plt.subplots_adjust(hspace=0.2)
    plt.show()
```

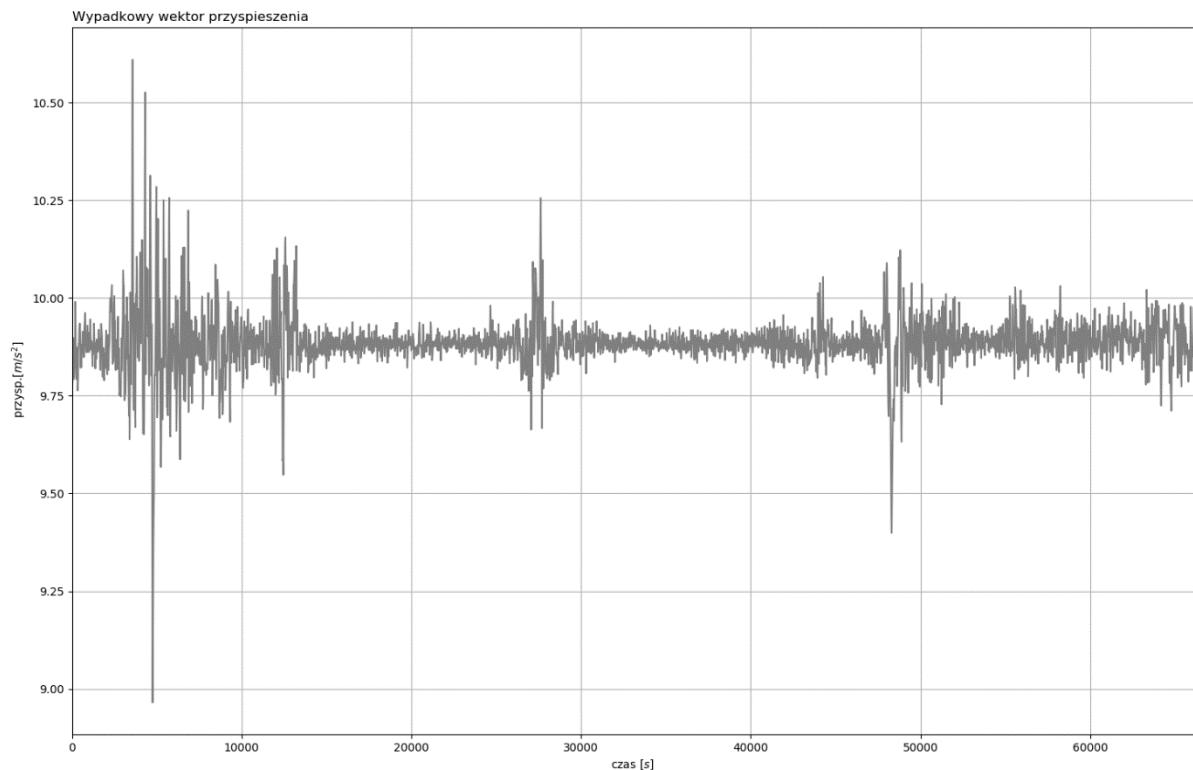
Funkcje dotyczące wartości wektora kolejno wykorzystano do wykonania i zobrazowania rezultatów dla każdej z zarejestrowanych aktywności fizycznych. Kod programu odpowiedzialny za wykonanie tych operacji przedstawia Listing 3.6.

Listing 3.6. Kod programu w pliku main.py odpowiedzialny wykonanie obliczeń wartości wektora wychylenia dla każdej aktywności fizycznej oraz zobrazowanie obliczenia na wykresach.

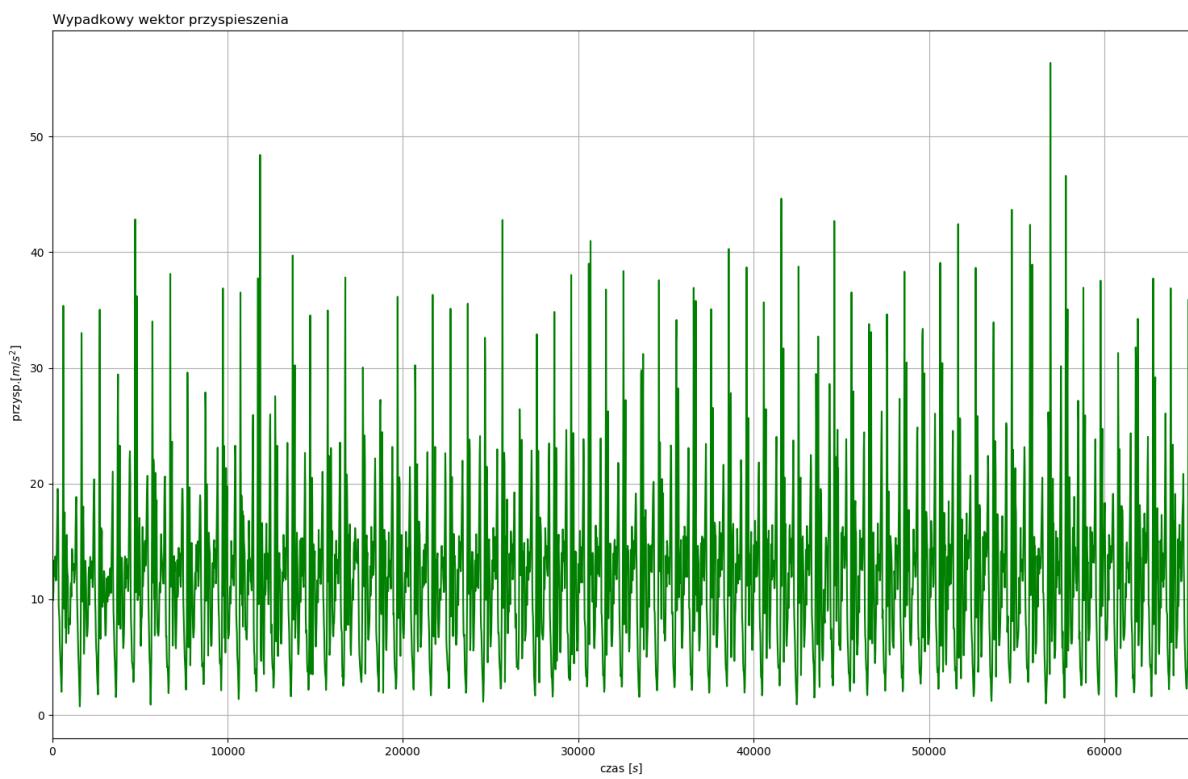
```
#Calculate magnitudes
STANDING['magnitude'] = sig.magnitude(STANDING)
WALKING['magnitude'] = sig.magnitude(WALKING)
DOWNSTAIRS['magnitude'] = sig.magnitude(DOWNSTAIRS)
UPSTAIRS['magnitude'] = sig.magnitude(UPSTAIRS)
RUNNING['magnitude'] = sig.magnitude(RUNNING)

#Draw a graphs of magnitudes
vis.graph_magnitude(STANDING, STANDING_COLOR, 'Standing - magnitude')
vis.graph_magnitude(WALKING, WALKING_COLOR, 'Walking - magnitude')
vis.graph_magnitude(DOWNSTAIRS, DOWNSTAIRS_COLOR, 'Downstairs -magnitude')
vis.graph_magnitude(UPSTAIRS, UPSTAIRS_COLOR, 'Upstairs - magnitude')
vis.graph_magnitude(RUNNING, RUNNING_COLOR, 'Running - magnitude')
```

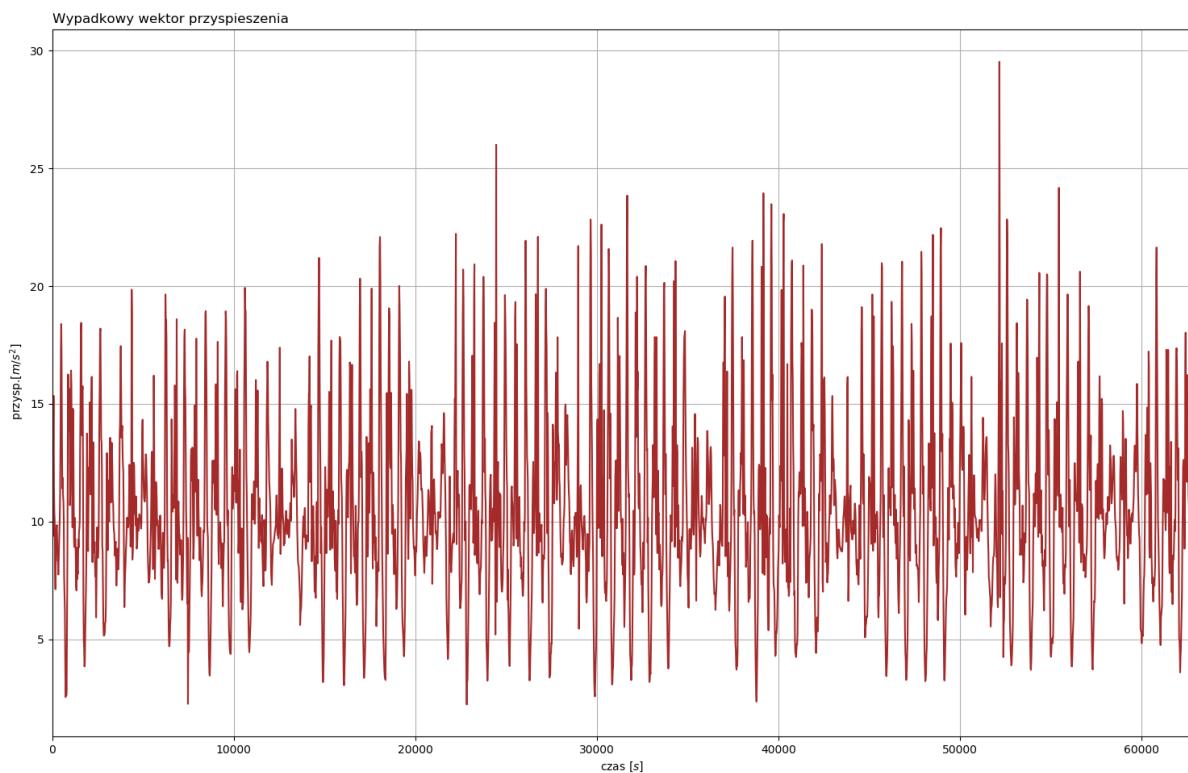
Wykonanie kodu z Listingu 3.6 dodało do każdego, zarejestrowanego sygnału dodatkową kolumnę *magnitude* w której wyliczona została wartość modułu wektora dla każdej z próbek akcelerometru. Zmianę tego parametru w czasie dla każdej z aktywności fizycznych przedstawiają Rysunki 3.6 – 3.11.



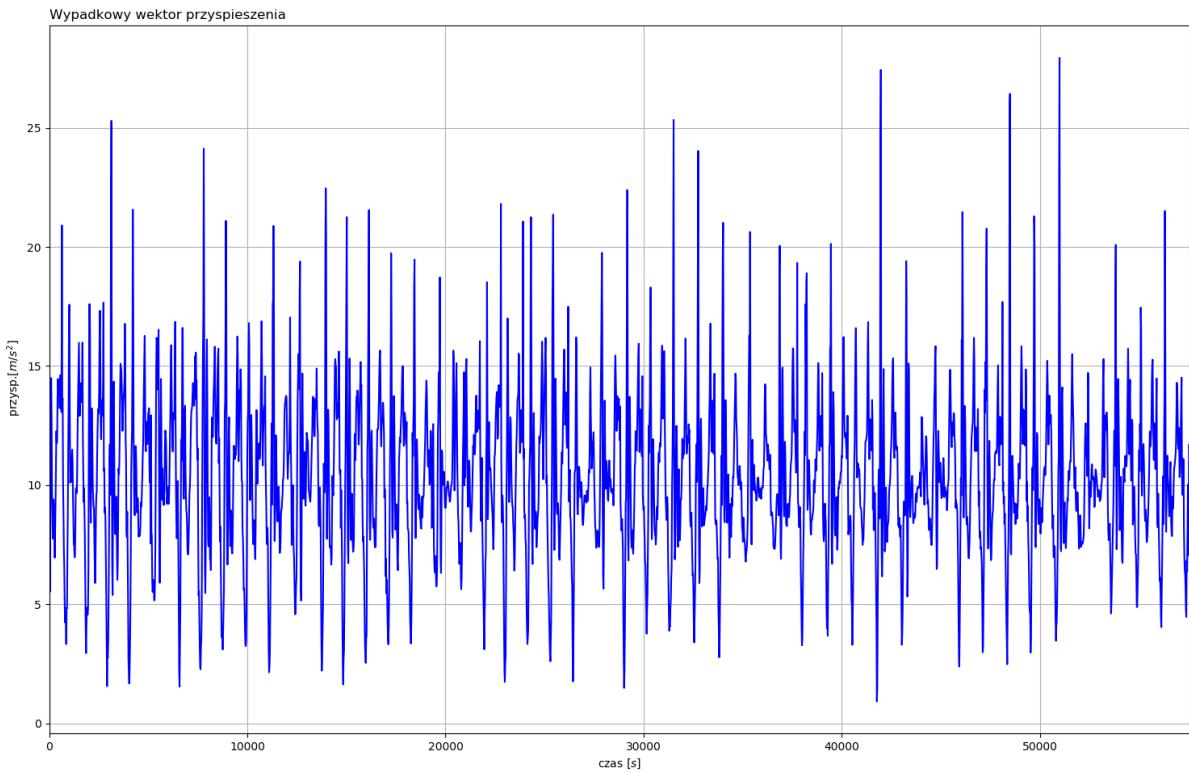
Rysunek 3.6. Zmiana wartości modułu wektora wychylenia akcelerometru w czasie, dla sygnału przyspieszenia zebranego podczas bezczynności.



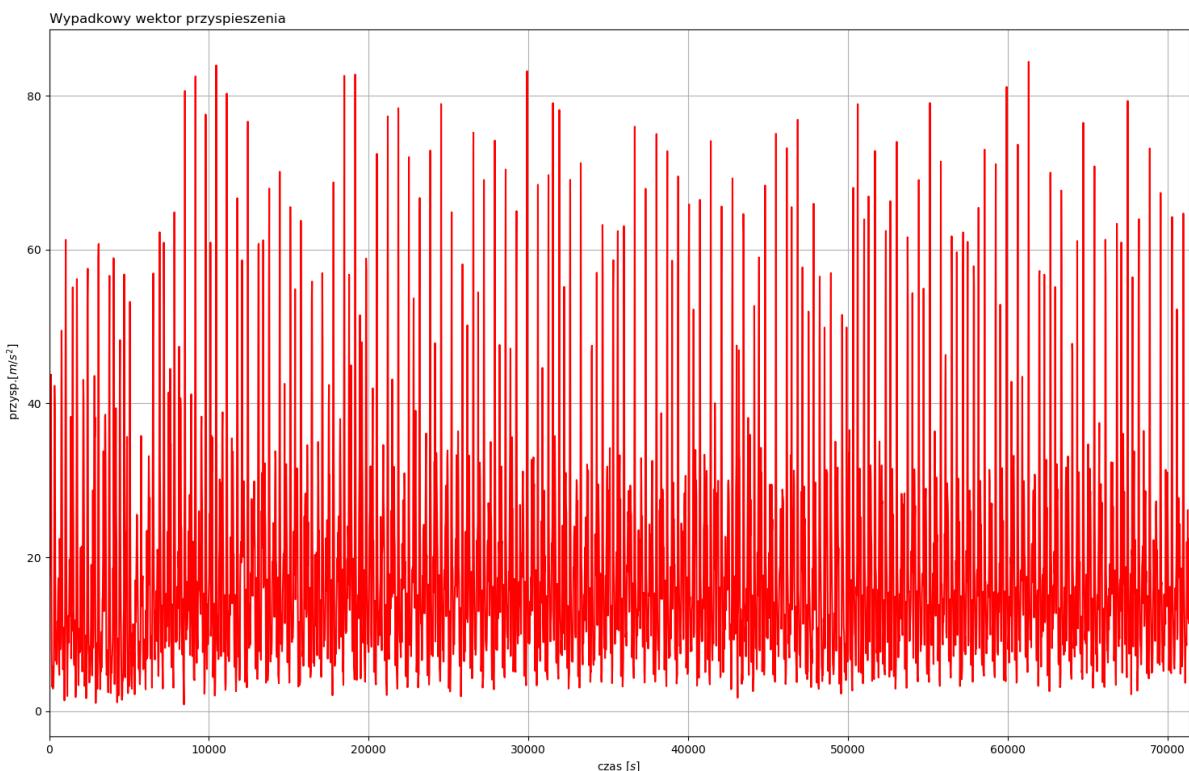
Rysunek 3.7. Zmiana wartości modułu wektora wychylenia akcelerometru w czasie, dla sygnału przyspieszenia zebranego podczas chodzenia.



Rysunek 3.8. Zmiana wartości modułu wektora wychylenia akcelerometru w czasie, dla sygnału przyspieszenia zebranego podczas schodzenia ze schodów.



Rysunek 3.9. Zmiana wartości modułu wektora wychylenia akcelerometru w czasie, dla sygnału przyspieszenia zebranego podczas wchodzenia po schodach.



Rysunek 3.10. Zmiana wartości modułu wektora wychylenia akcelerometru w czasie, dla sygnału przyspieszenia zebranego podczas biegania.

### 3.3. Podział sygnału na okna

Na potrzeby analizy sygnału przyspieszenia, należy podzielić sygnał na okna, dla których w późniejszym etapie zostaną wyodrębnione odpowiednie cechy. W tym celu przygotowano metodę *divide\_signal* przedstawioną poniżej na Listingu 3.7.

Listing 3.7. Metoda *divide\_signal* z pliku signal.py dzieląca sygnał na segmenty.

```
def divide_signal(df, size=100):
    start = 0
    while start < df.count():
        yield start, start + size
        start += (size / 2)
```

Metoda *divide\_signal(df, size= 100)* dzieli sygnał na równe partie. Podany na wejściu parametr jest jednowymiarową tablicą zawierającą elementy, które zostaną poddane podziałowi. Parametr size, domyślnie ustawiony na 100 wskazuje na liczbę próbek po której ma nastąpić podział. Podczas wykonywania badania, w początkowej fazie, liczba próbek pozostała ustawiona na tę wartość. Sto próbek przy częstotliwości zbliżonej do 50Hz odpowiada dwóm sekundom zarejestrowanego ruchu. Metoda zwraca kolejne indeksy początkowe i końcowe okien.

Dla zobrazowania podziału zaimplementowano metodę, która generuje wykresy z zaznaczonymi liniami podziału. Przedstawiono ją na Listingu 3.8

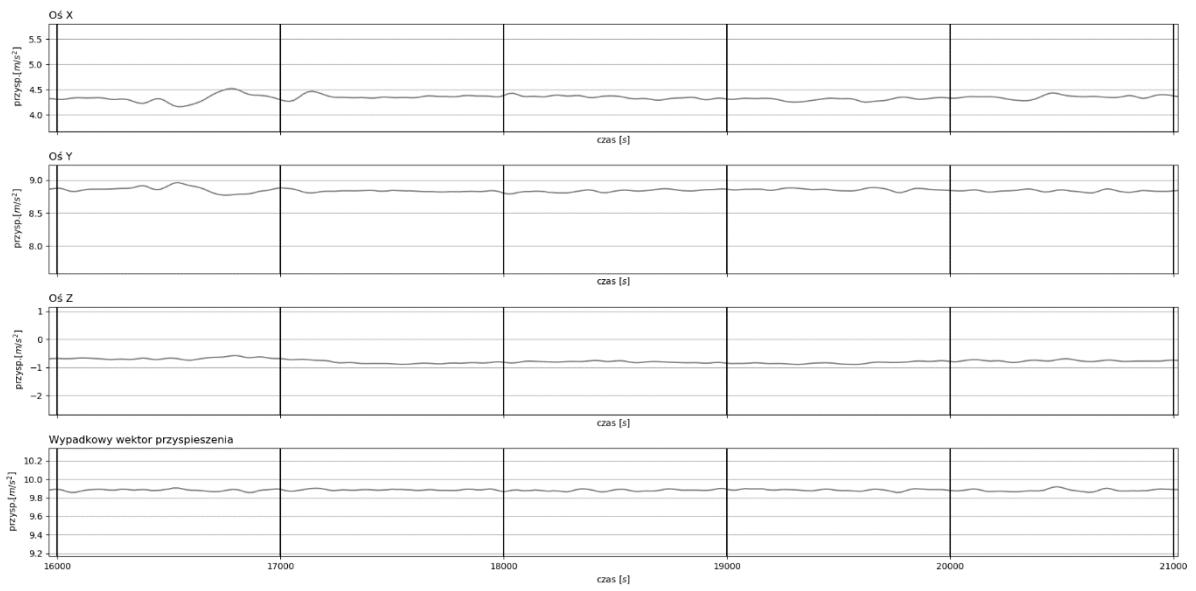
Listing 3.8. Metoda *graph\_divided\_signal* z pliku visualize.py wykorzystywana do wizualizacji podzielonego sygnału.

```
def graph_divided_signal(activity, color, title):
    fig, (ax0, ax1, ax2, ax3) = plt.subplots(nrows=4, figsize=(15, 10),
    sharex=True)
    plot_graph(ax0, activity['timestamp'], activity['xAxis'], 'X-
    axis(divided)', color)
    plot_graph(ax1, activity['timestamp'], activity['yAxis'], 'Y-
    axis(divided)', color)
    plot_graph(ax2, activity['timestamp'], activity['zAxis'], 'Z-
    axis(divided)', color)
    plot_graph(ax3, activity['timestamp'], activity['magnitude'],
    'magnitude(divided)', color)

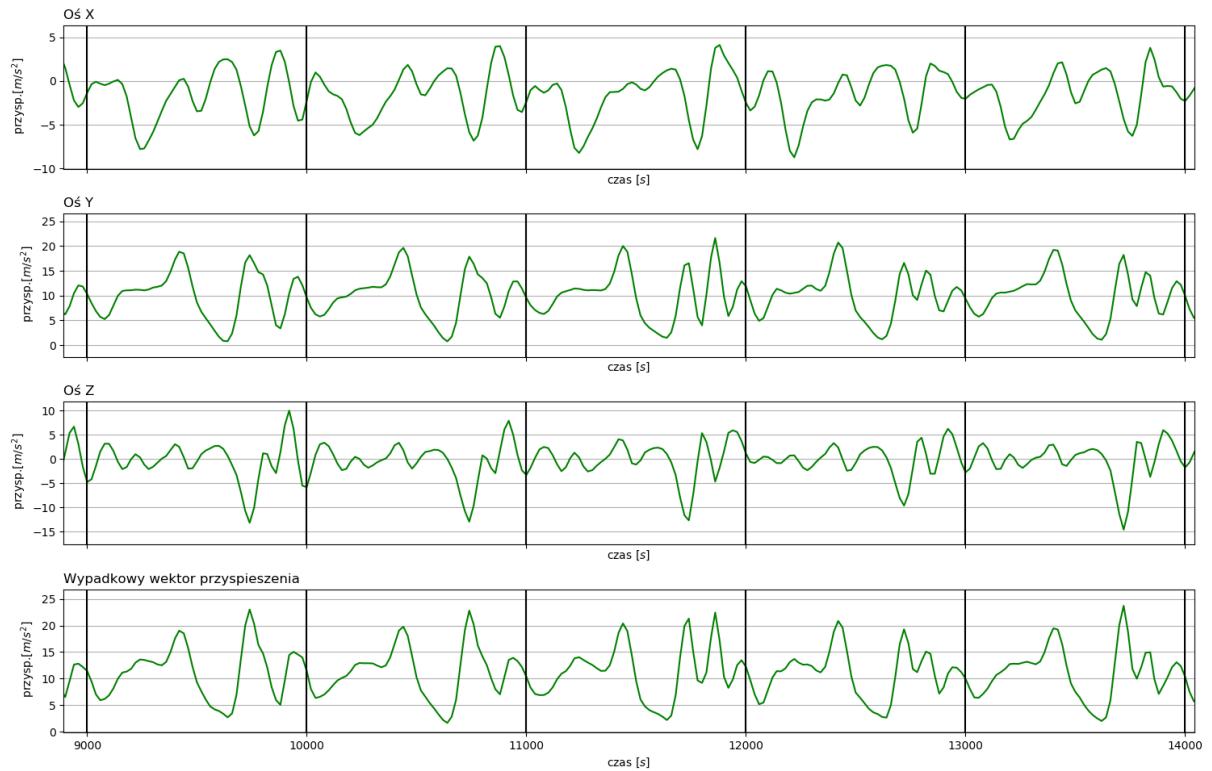
    for (start, end) in sig.divide_signal(activity['timestamp']):
        ax0.axvline(activity['timestamp'][start], color='black')
        ax1.axvline(activity['timestamp'][start], color='black')
        ax2.axvline(activity['timestamp'][start], color='black')
        ax3.axvline(activity['timestamp'][start], color='black')

    plt.suptitle(title)
    plt.show()
```

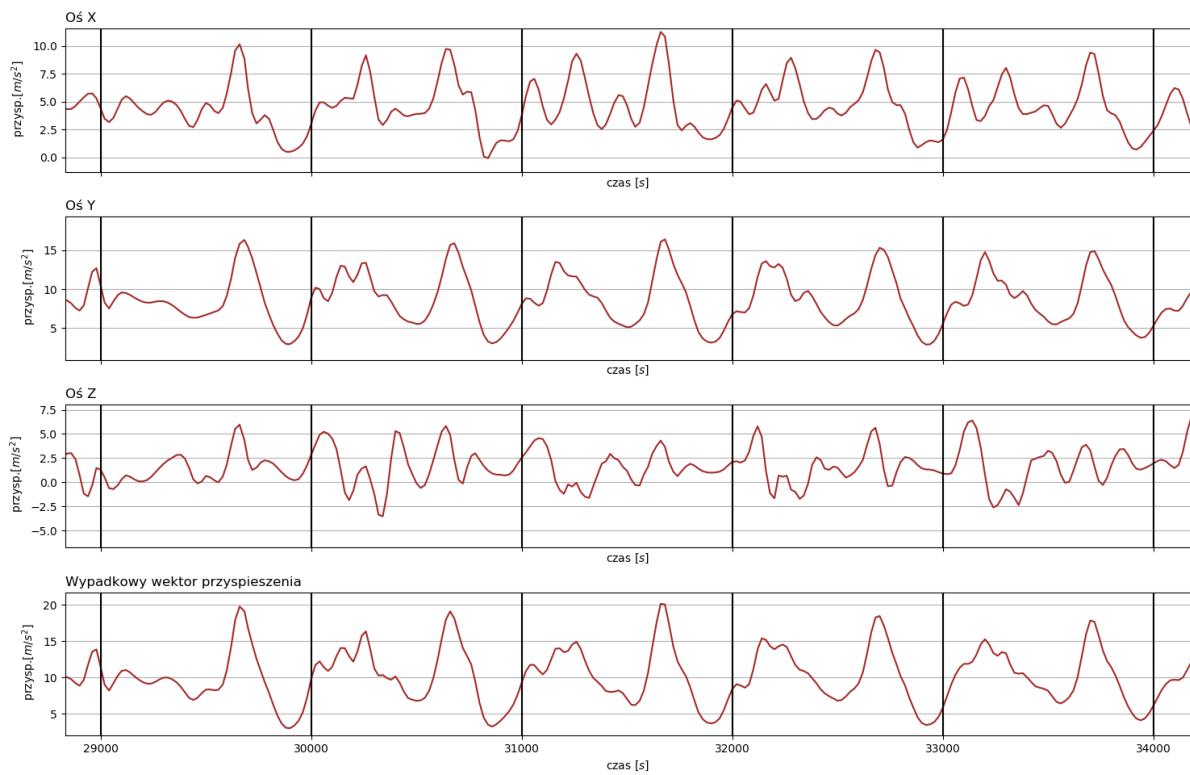
Metoda *graph\_divided\_signal* generuje cztery wykresy dla osi x, y, z oraz dla wartości wektora przyspieszenia. Na każdy z wykresów nanoszone są czarne linie podziału według metody *divide\_signal* przedstawionej we wcześniejszej części. Wywołanie metody rysującej wykresy z zaznaczonym podziałem zostało wykonane dla każdej aktywności fizycznej. Wynikiem są wykresy przedstawione na Rysunkach 3.11 – 3.15.



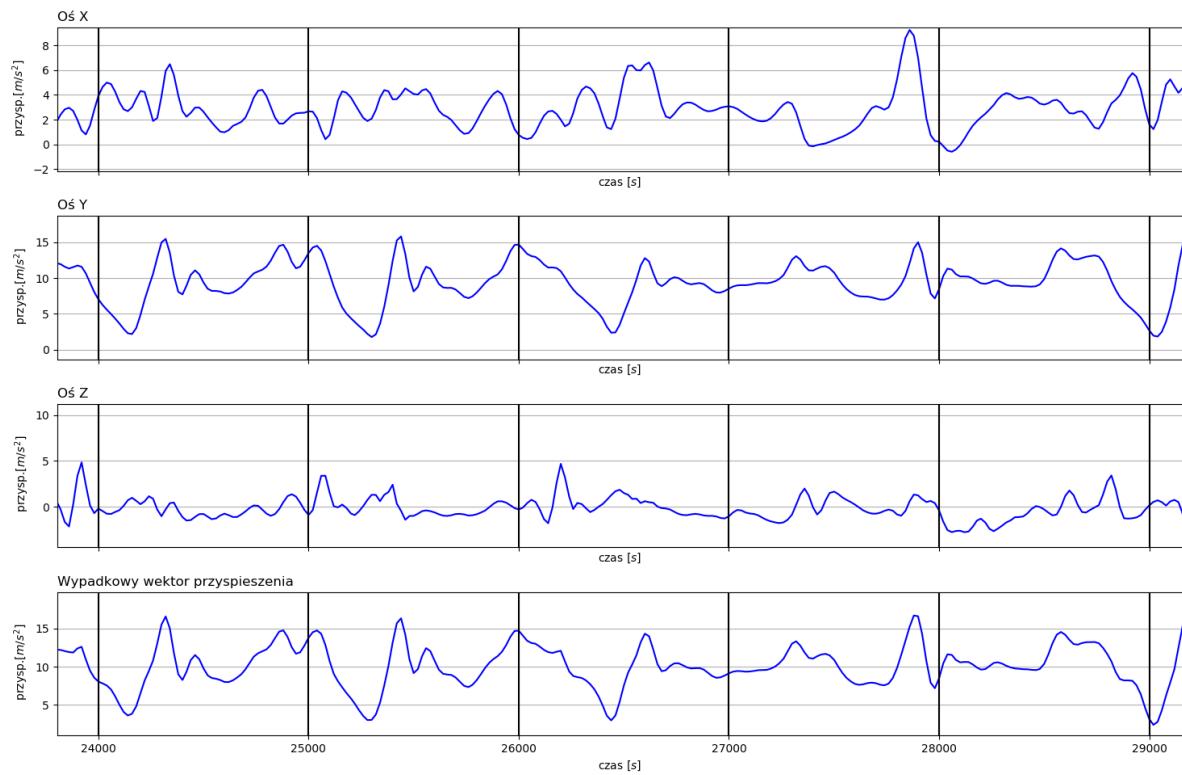
Rysunek 3.11. Fragment sygnału zarejestrowanego podczas bezczynności podzielony na okna składające się ze 100 próbek.



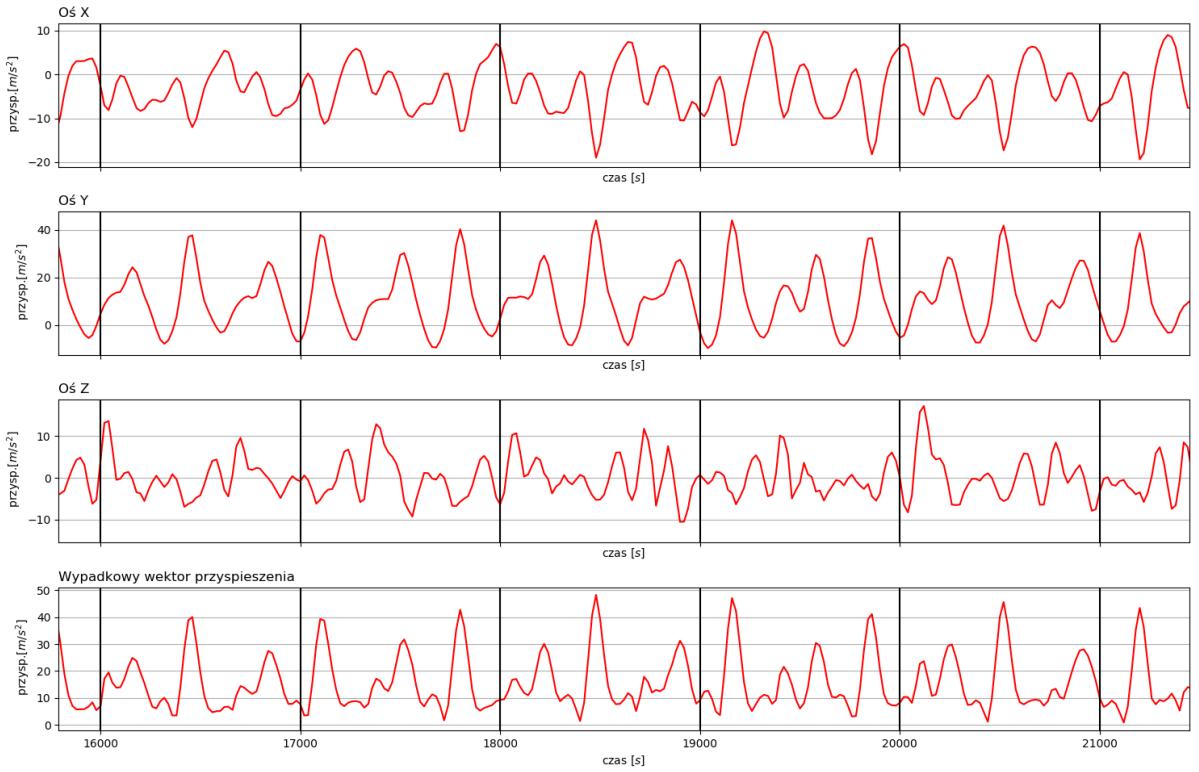
Rysunek 3.12. Fragment sygnału zarejestrowanego podczas chodzenia podzielony na okna składające się ze 100 próbek.



Rysunek 3.13. Fragment sygnału zarejestrowanego podczas schodzenia ze schodów podzielony na okna składające się ze 100 próbek.



Rysunek 3.14. Fragment sygnału zarejestrowanego podczas wchodzenia po schodach podzielony na okna składające się ze 100 próbek.



Rysunek 3.15. Fragment sygnału zarejestrowanego podczas biegania podzielony na okna składające się ze 100 próbek.

### 3.4. Ekstrakcja cech

W kolejnym etapie wyekstrahowano cechy sygnału dla każdego okna powstało w wyniku podziału. W tym celu przygotowano kod programu mający na celu pobranie sygnału w zakresie zdefiniowanych okien oraz obliczenie wartości cech. Metody przedstawia poniższy Listing 3.9.

Listing 3.9. Metody *window\_values* oraz *extract\_features* z pliku *signal.py* wykorzystywane do wyodrębnienia cech sygnału zarejestrowanego przez akcelerometr.

```
def window_values(axis, start, end):
    start = int(start)
    end = int(end)
    p25 = np.percentile(axis[start:end], 25)
    median = np.percentile(axis[start:end], 50)
    p75 = np.percentile(axis[start:end], 75)
    return [
        axis[start:end].mean(),
        p25,
        median,
        p75,
        axis[start:end].std(),
        axis[start:end].var(),
        axis[start:end].min(),
        axis[start:end].max(),
        skew(axis[start:end]),
        kurtosis(axis[start:end]),
```

```

]

def extract_features(activity):
    for (start, end) in divide_signal(activity['timestamp']):
        activity_values = ['xAxis', 'yAxis', 'zAxis', 'magnitude']
        features = []
        for axis in activity_values:
            features += window_values(activity[axis], start, end)
    yield features

```

Pierwsza ze wskazanych metod – `window_values` jako wejście przyjmuje wektor odpowiadający pomiarom jednej osi akcelerometru lub wartości średniej wektora. Definiowany jest także przedział ograniczony przekazanymi wartościami `start` i `end`. Metoda oblicza komplet cech sygnału, które wykorzystano w badaniu. Wywołanie `window_values` ma miejsce wewnętrz metody `extract_features`, która dla zadanej w parametrze aktywności fizycznej dzieli sygnał, a następnie przekazuje poszczególne okna do obliczeń. Zwracane są obliczone wartości cech dla caego sygnału.

Opisany kod programu posłużył do wygenerowania pliku z cechami każdego z sygnałów aktywności. Plik z cechami sygnału zawiera na pierwszym miejscu liczbę określającą jakiej aktywności fizycznej dotyczy dany wiersz, a następnie komplet cech dla danego okna. Na potrzeby opisanej procedury stworzono tablicę aktywności fizycznych zawierającą komplet danych oraz wyodrębniono nazwę pliku, do którego miało nastąpić zapisanie obliczeń. W dalszej części plik zawierający obliczone cechy posłuży do nauki klasyfikatora. Dla każdej aktywności fizycznej w zdefiniowanej tablicy została wywołana funkcja `extract_features`. Całość kodu programu wykorzystanego do procedury ekstrakcji cech znajduje się na Listingu 3.10.

Listing 3.10. Kod programu w pliku main.py odpowiedzialny za wygenerowanie pliku z wartościami cech sygnału.

```

activities = [STANDING, WALKING, DOWNSTAIRS, UPSTAIRS, RUNNING]
output_file_path = 'mgr/data/resources/Features.csv'

with open(output_file_path, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities)):
        for f in sig.extract_features(activities[i]):
            rows.writerow([i] + f)

```

W pliku tym, w jeden linii zapisane są wszystkie cechy dla danego okna, bez podziału na poszczególne osie czy też wektor przyspieszenia. W każdym wierszu zapisane są zatem cztery zestawy wartości obejmujące:

- średnią arytmetyczną, która jest podstawową miarą określającą średnią wartość. Definiowana jako iloraz sumy liczb i ilości tych liczb [16].

$$A = \frac{(x_1 + x_2 + \dots + x_n)}{n}$$

$x_1, x_2, x_n$  – kolejne wartości  
 $n$  – liczba wartości

- medianę, czyli miarę położenia obserwacji, która dzieli zbiór na połowę [17].

$$Me = \begin{cases} \frac{x_{n+1}}{2}, & \text{gdy } n \text{ jest nieparzyste} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right), & \text{gdy } n \text{ jest parzyste} \end{cases}$$

$x_1, x_2, x_n$  – kolejne wartości  
 $n$  – liczba wartości

- wartość pierwszego kwartyłu, czyli miarę położenia obserwacji, w której 25% obserwacji położonych jest poniżej wskazanej wartości. Wartość pierwszego kwartyłu wyznaczana jest poprzez obliczenie mediany, która dzieli zbiór na dwa podzbiory. Następnie w pierwszym podzbiorze mediana obliczana jest ponownie [17].
- wartość trzeciego kwartyłu, który dzieli zbiór obserwacji na dwie części, z czego 75% leży poniżej, a 25% powyżej wskazanej wartości. Wartość trzeciego kwartyłu wyznaczana jest poprzez obliczenie mediany, która dzieli zbiór na dwa podzbiory. Następnie w drugim podzbiorze mediana obliczana jest ponownie [17].
- wariancję, będącą miarą służącą do określenia zmienności sygnału. Wariancja o tym jak szeroko wartość zarejestrowanego przyspieszenia jest rozrzucona względem średniej. Mniejsza wartość wariancji oznacza większe skupienie wartości wokół średniej [18].

$$\sigma^2 = \frac{(x_1 - A)^2 + (x_2 - A)^2 + \dots + (x_n - A)^2}{n}$$

$x_1, x_2, x_n$  – kolejne wartości  
 $n$  – liczba wartości  
 $A$  – średnia arytmetyczna

- odchylenie standardowe, które podobnie jak wariancja charakteryzuje rozproszenie danych. Odchylenie standardowe obliczane jest jako pierwiastek kwadratowy z wariancji [19].

$$\sigma = \sqrt{\sigma^2}$$

$\sigma^2$  – wariancja

- wartość minimalną, której funkcja ją określająca w zbiorze  $P$  wynosi  $x$  dla takiego  $x$  należącego do zbioru  $P$ , że dla każdego  $p$  ze zbioru  $P$ , wartość  $x$  jest mniejsza lub równa  $p$  [20].

$$\min(P) = x \Leftrightarrow x \in P \wedge \forall_{p \in P} x \leq p$$

- wartość maksymalną, której funkcja ją określająca w zbiorze  $P$  wynosi  $x$  dla takiego  $x$  należącego do zbioru  $P$ , że dla każdego  $p$  ze zbioru  $P$ , wartość  $x$  jest większa lub równa  $p$  [21].

$$\max(P) = x \Leftrightarrow x \in P \wedge \forall_{p \in P} x \geq p$$

- współczynnik skośności, który dla rozkładu normalnego wynosi w przybliżeniu 0. Jeżeli współczynnik ma wartość większą od zera, oznacza to, że ciężar przeniesiony jest na lewą stronę sygnału [22].

$$O_d = \frac{A - d}{\sigma}$$

*$\sigma$  – odchylenie standardowe  
 $A$  – średnia arytmetyczna  
 $d$  – dominanta*

- Kurtozę, będącą miarą spłaszczenia rozkładu wartości sygnału. Poniższy wzór określa definicję kurwozy gdzie  $\sigma$  oznacza odchylenie standardowe, a  $\mu_4$  czwarty moment centralny. Kurtoza zatem jest czwartym momentem centralnym podzielonym przez odchylenie standardowe podniesione do 4 potęgi. Dla definicji Fishera odejmowana jest liczba 3, tak by dla rozkładu normalnego rezultatem była liczba zero [23].

$$Kurt = \frac{\mu_4}{\sigma^4} - 3$$

*$\sigma$  – odchylenie standardowe  
 $\mu_4$  – czwarty moment centralny*

## 4. Rezultaty klasyfikacji

Po przetworzeniu sygnału i wyodrębnieniu wartości cech sygnału przystąpiono do wykorzystania ich w zadaniu klasyfikacji. W tym celu przygotowano metodę przeznaczoną do przeprowadzenia testów klasyfikatorów. Dla podanego w parametrach wejściowych klasyfikatora i zestawu danych wyodrębniane są dwa zbiory: zawierający cechy zarejestrowanego sygnału oraz zbiór określający jaka aktywność fizyczna obrazowana jest przez cechy przedstawione w danym wierszu. Kolejno przeprowadzane jest 20 prób nauki i oceny działania klasyfikatora. Zbiór przeznaczony do oceny zawiera 25% danych przekazanych w parametrze wejściowym. Zwracanym przez metodę wynikiem jest średnia arytmetyczna, odchylenie standardowe wartości otrzymanych rezultatów oraz minimalna i maksymalna osiągnięta trafność klasyfikacji. Kod programu obrazuje Listing 4.1.

Listing 4.1. Metoda *test\_and\_learn\_classifier* z pliku signal.py odpowiedzialna za naukę i ocenę działania klasyfikatora.

```
def test_and_learn_classifier(classifier, features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        af_train, af_test, am_train, am_test =
train_test_split(activity_features, activity_markers, test_size=.25)
        classifier.fit(af_train, am_train)
        res = classifier.score(af_test, am_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results),
        np.min(results),
        np.max(results)
    ]
```

Testy za pomocą metody *test\_and\_learn\_classifier* przeprowadzono na podstawie wartości cech sygnału zapisanych w pliku w jednym z poprzednich kroków. Do klasyfikacji wykorzystano arbitralnie wybrany klasyfikator k-najbliższych sąsiadów.

Za wczytanie wartości z pliku CSV, utworzenie wszystkich, powyższych klasyfikatorów oraz wypisanie trafności klasyfikacji odpowiedzialny był kod programu przedstawiony na Listingu 4.2.

Listing 4.2. Kod programu w pliku main.py odpowiedzialny wczytanie danych, utworzenie i wykonanie oceny klasyfikatora.

```
features = np.loadtxt('mgr/data/features/person_1/Features.csv',
delimiter=',')
k_neighbors_cls = KNeighborsClassifier()
print('K-Neighbors Classifier ', sig.test_and_learn_classifier(k_neighbors_cls, features))
```

Przedstawiony na Listingu 4.2 kod programu zwrócił i wypisał wyniki dla klasyfikatora wypisując kolejno w wierszu jego nazwę, średnią trafność klasyfikacji oraz odchylenie standardowe wyników osiągniętych podczas 20 prób. Wynik przedstawia Tabela 4.1.

Tabela 4.1. Wynik działania klasyfikatora przedstawiony jako średnia arytmetyczna trafności klasyfikacji osiągniętej podczas 20 prób oraz odchylenie standardowe wyniku.

Klasyfikator	Trafność klasyfikacji	Odchylenie standardowe
<b>K najbliższych sąsiadów</b>	99,8%	0,4%

Nauczony według opisanego procesu klasyfikator k najbliższych sąsiadów wykorzystano do klasyfikacji aktywności fizycznej zebranej podczas kilkuminutowego spaceru. Aktywność rozpoczęła się od zejścia po schodach, chwili bezczynności i spaceru. Następnie w drodze powrotnej, osoba rejestrująca aktywność zatrzymała się i weszła po schodach do miejsca początkowego. Zarejestrowaną aktywność fizyczną przedstawia Rysunek 4.1.



Rysunek 4.1. Wizualizacja klasyfikacji sygnału z pliku zawierającego zróżnicowaną pod względem klas aktywność użytkownika, przy użyciu klasyfikatora k najbliższych sąsiadów.

Oś pozioma wykresu przedstawia kolejne okna klasyfikacji dla których wyodrębniano wartości cech sygnału. Wartości na osi pionowej odpowiadają natomiast numerom przyporządkowanym do poszczególnych aktywności fizycznych, gdzie numery oznaczają:

- 0 – bezczynność (kolor szary)
- 1 – chodzenie (kolor zielony)
- 2 – schodzenie ze schodów (kolor czerwony)
- 3 – wchodzenie po schodach (kolor niebieski)
- 4 – bieganie (kolor pomarańczowy)

Jak można zaobserwować klasyfikator bez problemu rozpoznał główną część aktywności fizycznej, czyli spacer. Można stwierdzić, że cała aktywność fizyczna związana z chodzeniem została zakwalifikowana poprawnie. Podobnie sytuacja wygląda z bezczynnością. Tutaj także

przed i po spacerze mamy zaznaczoną odpowiednią klasę aktywności. Klasyfikator nie poradził sobie jednak z wykryciem wchodzenia i schodzenia po schodach.

Po zaimplementowaniu mechanizmów odpowiedzialnych za przetwarzanie sygnału przyspieszenia przystąpiono do optymalizacji procesu nauki, pod kątem zwiększenia trafności klasyfikacji. Chcąc dobrać odpowiedni klasyfikator do omawianego zastosowania w początkowej fazie optymalizacji wykorzystano 5 klasyfikatorów:

- klasyfikator k najbliższych sąsiadów,
- drzewo decyzyjne,
- las losowy,
- perceptron wielowarstwowy,
- naiwny klasyfikator Bayesa.

Posiadając wartości przyspieszenia w trzech osiach, na potrzeby klasyfikacji konieczne jest wyodrębnienie cech sygnału. W tym celu można obliczać bezpośrednio z wartości zebranych przez akcelerometr lub też skorzystać z modułu wektora przyspieszenia. Trzecią możliwością jest obliczenie cech z wszystkich danych łącznie. Wykonując pierwsze badanie, klasyfikatory operowały na sygnałach zebranych przez pojedynczą osobę. Weryfikacja trafności klasyfikacji dla jednej osoby odbywała się za pomocą klasyfikatora uczonego na sygnale przyspieszenia zebranych od tej osoby.

Pierwszą próbą była nauka klasyfikatorów na podstawie cech wyekstrahowanych z sygnału przyspieszenia dla każdej z osi. Wyniki przedstawia Tabela 4.2.

Tabela 4.2. Zestawienie trafności klasyfikacji dla cech wyliczonych dla każdej osi. Sygnały treningowe i testowe pochodzą od jednej osoby.

Klasyfikator	OSOBA_1	OSOBA_2	OSOBA_3	OSOBA_4	Średnia
K najbliższych sąsiadów	98,78%	97,41%	100,00%	96,40%	98,15%
Drzewo decyzyjne	99,39%	98,06%	98,30%	95,40%	97,79%
Las losowy	99,02%	98,15%	98,83%	96,67%	98,17%
Perceptron wielowarstwowy	89,27%	97,28%	100,00%	91,04%	94,40%
Naiwny Bayesa	99,76%	98,66%	98,66%	95,81%	98,22%
				Średnia ogółem:	97,34%

Średni wynik uzyskany po pierwszym badaniu wyniósł ponad 97%. Osiągnięta trafność klasyfikacji świadczy o bardzo dokładnym rozpoznawaniu aktywności fizycznej. Zweryfikowano jednak jaka trafność klasyfikacji zostanie otrzymana w wyniku wykorzystania wyłącznie wartości cech obliczonych na podstawie modułu wektora przyspieszenia. Wyniki tego badania przedstawia Tabela 4.3.

Tabela 4.3. Zestawienie trafności klasyfikacji dla wartości cech wyliczonych z modułu wektora przyspieszenia. Sygnały treningowe i testowe pochodzą od jednej osoby.

Klasyfikator	OSOBA_1	OSOBA_2	OSOBA_3	OSOBA_4	Średnia
<b>K najbliższych sąsiadów</b>	85,37%	95,69%	88,70%	90,99%	90,19%
<b>Drzewo decyzyjne</b>	88,41%	90,99%	90,91%	89,64%	89,99%
<b>Las losowy</b>	90,79%	91,64%	90,74%	89,91%	90,77%
<b>Perceptron wielowarstwowy</b>	87,13%	87,41%	85,13%	83,02%	85,67%
<b>Naiwny Bayesa</b>	90,73%	90,69%	89,51%	89,77%	90,18%
				<b>Średnia ogółem:</b>	89,36%

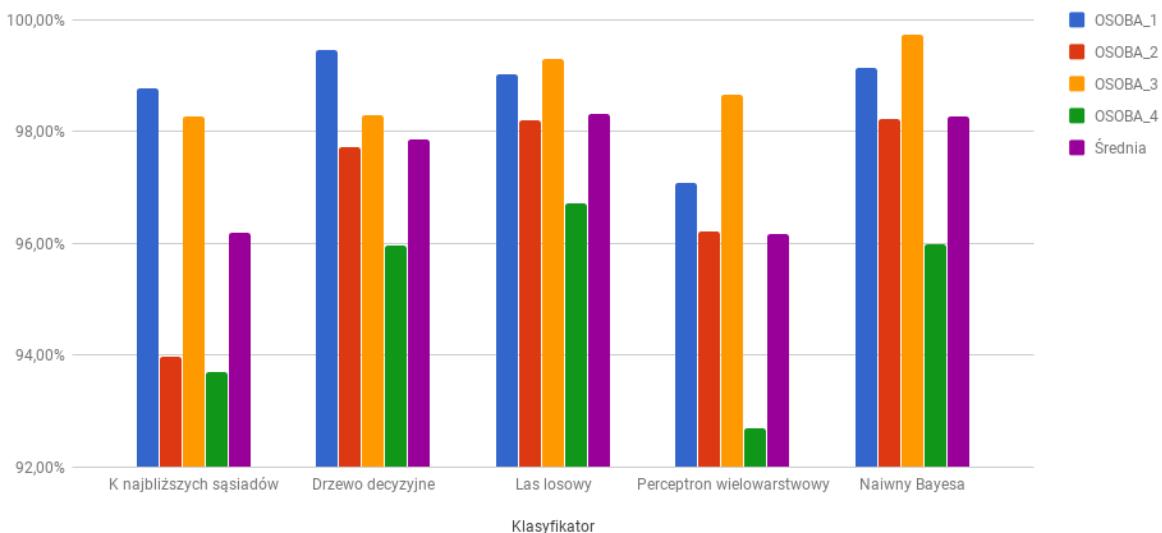
Odnosząc się do poprzedniej próby zaobserwowano spadki w trafności klasyfikacji każdego z klasyfikatorów. W szczególności, w przypadku perceptronu wielowarstwowego wynik spadł do blisko 85%. Średnia trafność klasyfikacji dla każdego klasyfikatora łącznie wyniosła 89,36%.

Ostatnim zestawem danych, dla których zweryfikowano trafność klasyfikacji jest połączenie dwóch poprzednich wariantów. Wartości cech wyliczono bowiem zarówno dla sygnału z trzech osi, jak i dla wartości średniej wektora przyspieszenia. Osiągnięte rezultaty przedstawiono w Tabeli 4.4.

Tabela 4.4. Zestawienie trafności klasyfikatorów dla wartości cech wyliczonych z poszczególnych osi oraz modułu wektora przyspieszenia. Sygnały treningowe i testowe pochodzą od jednej osoby.

Klasyfikator	OSOBA_1	OSOBA_2	OSOBA_3	OSOBA_4	Średnia
<b>K najbliższych sąsiadów</b>	98,78%	93,97%	98,26%	93,69%	96,18%
<b>Drzewo decyzyjne</b>	99,45%	97,72%	98,30%	95,95%	97,86%
<b>Las losowy</b>	99,02%	98,19%	99,30%	96,71%	98,31%
<b>Perceptron wielowarstwowy</b>	97,07%	96,21%	98,65%	92,70%	96,16%
<b>Naiwny Bayesa</b>	99,15%	98,23%	99,74%	95,99%	98,28%
				<b>Średnia ogółem:</b>	97,35%

Połączenie kompletu danych pozwoliło osiągnąć rezultat porównywalny do klasyfikacji wyłącznie na podstawie przyspieszenia w trzech osiach. Trafność klasyfikacji wyniosła średnio 97,35% i była jedynie o 0,01 punktu procentowego lepsza od wykorzystania wyłącznie sygnału z osi. Widać zatem, iż dodanie wartości dla modułu wektora przyspieszenia, nie pogarsza znacznie wyników. Trafności klasyfikatorów dla ostatniego wariantu obrazuje Rysunek 4.2.



Rysunek 4.2. Wykres obrazujący trafność klasyfikatorów. Sygnały treningowe i testowe pochodzą od jednej osoby.

Omówiony przykład nie odzwierciedla jednak idealnie możliwości wykrywania aktywności fizycznych w życiu codziennym. Aplikacje nie mają bowiem próbek aktywności fizycznych zebranych dla osoby korzystającej z aplikacji. Dlatego też w kolejnym kroku zweryfikowano trafność klasyfikacji dla tych samych trzech przypadków, zmieniając jednak zależności pomiędzy zbiorami danych wykorzystywany do nauki i weryfikacji trafności klasyfikacji. Chcąc przetestować klasyfikator na danych wybranej osoby, wykorzystano do nauki sygnały przyspieszenia pozyskane od pozostałych osób. Oba zbiory sygnałów pozostały zatem rozłączne. Spodziewano się, iż nastąpi spadek trafności rozpoznawania aktywności. Wyniki zostały przedstawione w tabelach, gdzie kolumny oznaczają osobę wykluczoną ze zbioru do nauki klasyfikatora i tym samym oznaczającą osobę, na której dany klasyfikator był testowany. Osiągnięte rezultaty dla osi x, y, z przedstawia Tabela 4.5.

Tabela 4.5. Zestawienie trafności klasyfikatorów dla wartości cech wyliczonych dla poszczególnych osi. Zbiory sygnałów treningowych i testowych są rozłączne i pochodzą od różnych osób.

Klasyfikator	OSOBA_1	OSOBA_2	OSOBA_3	OSOBA_4	Średnia
K najbliższych sąsiadów	53,54%	87,93%	86,68%	46,38%	68,63%
Drzewo decyzyjne	65,03%	52,46%	77,14%	77,47%	68,03%
Las losowy	55,28%	78,32%	83,94%	80,31%	74,46%
Perceptron wielowarstwowy	46,66%	84,27%	87,23%	66,66%	71,21%
Naiwny Bayesa	23,08%	56,47%	46,07%	69,91%	48,88%
				Średnia ogółem:	66,24%

Zgodnie z oczekiwaniemi średnia wartość trafności klasyfikacji dla wszystkich klasyfikatorów zauważalnie spadła. Spadek widać w szczególności u osoby pierwszej, której telefon z włączoną aplikacją był umocowany nieco luźniej od pozostałych osób. Średnia

trafność klasyfikatorów wyniosła 66,24% i była znacznie niższa od analogicznego badania przedstawionego we wcześniejszej części.

Kolejną weryfikacją było sprawdzenie trafności klasyfikacji dla wartości średniej wektora. Rezultat badania został przedstawiony w Tabeli 4.6

Tabela 4.6. Zestawienie trafności klasyfikacji dla wartości cech wyliczonych dla modułu wektora przyspieszenia. Zbiory sygnałów treningowych i testowych są rozłączne i pochodzą od różnych osób.

<b>Klasyfikator</b>	<b>OSOBA_1</b>	<b>OSOBA_2</b>	<b>OSOBA_3</b>	<b>OSOBA_4</b>	<b>Średnia</b>
<b>K najbliższych sąsiadów</b>	67,38%	76,72%	82,75%	79,41%	76,57%
<b>Drzewo decyzyjne</b>	59,46%	64,47%	53,49%	71,74%	62,29%
<b>Las losowy</b>	64,51%	71,15%	68,70%	76,09%	70,11%
<b>Perceptron wielowarstwowy</b>	65,42%	79,26%	84,19%	80,66%	77,38%
<b>Naiwny Bayesa</b>	52,92%	50,22%	48,25%	66,29%	54,42%
				<b>Średnia ogółem:</b>	68,15%

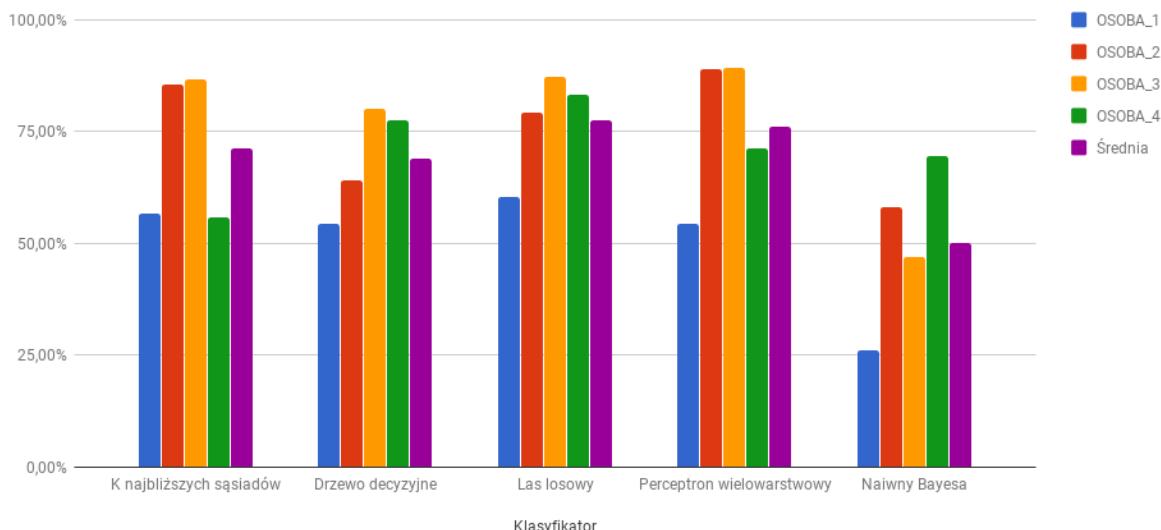
W omawianym przypadku średnia trafność klasyfikacji wyniosła 68,15%, tak więc ponownie zanotowano jej spadek. Warto zauważyć, iż w przypadku wartości średniej znaczenie położenia i umocowania telefonu maleje. Dlatego też osoba pierwsza zanotowała poprawę wyników klasyfikacji względem poprzedniego badania.

Łącząc oba zestawy danych do wyodrębniania wartości cech sygnału, przystąpiono do kolejnej weryfikacji, gdzie klasyfikatory korzystały z wartości wyodrębnionych zarówno bezpośrednio z osi, jak i z modułu wektora przyspieszenia. Rezultat przedstawia Tabela 4.7.

Tabela 4.7. Zestawienie trafności klasyfikacji dla wartości cech wyliczonych z poszczególnych osi oraz modułu wektora. Zbiory sygnałów treningowych i testowych są rozłączne i pochodzą od różnych osób.

<b>Klasyfikator</b>	<b>OSOBA_1</b>	<b>OSOBA_2</b>	<b>OSOBA_3</b>	<b>OSOBA_4</b>	<b>Średnia</b>
<b>K najbliższych sąsiadów</b>	56,61%	85,56%	86,68%	55,66%	71,13%
<b>Drzewo decyzyjne</b>	54,38%	64,14%	80,13%	77,57%	69,06%
<b>Las losowy</b>	60,29%	79,28%	87,37%	83,30%	77,56%
<b>Perceptron wielowarstwowy</b>	54,31%	88,98%	89,33%	71,19%	75,95%
<b>Naiwny Bayesa</b>	26,15%	57,97%	46,94%	69,46%	50,13%
				<b>Średnia ogółem:</b>	68,77%

W powyższym przypadku uzyskano najlepszy średni wynik klasyfikacji. Średnia trafność klasyfikacji wszystkich klasyfikatorów wyniosła 68,77% i była tym samym lepsza od wyniku uzyskanego w poprzednich dwóch badaniach. Wykres obrazujący trafność klasyfikacji poszczególnych klasyfikatorów dla najlepszego wariantu przedstawia Rysunek 4.3.



Rysunek 4.3. Wykres obrazujący trafność klasyfikatorów. Zbiory sygnałów treningowych i testowych są rozłączne oraz pochodzą od różnych osób.

Łącząc rezultaty osiągnięte we wszystkich badaniach stwierdzono, że wyodrębnianie wartości cech sygnału dla modułu wektora i poszczególnych osi jednocześnie jest najlepszym rozwiązańiem. W dalszej części optymalizacji procesu wszystkie wartości cech będą zatem wyodrębniane dla zestawu danych zawierającego zarówno moduł, jak i dane zebrane bezpośrednio z akcelerometru.. Ponadto dla każdej z osób w wybranym ostatecznie wariancie zauważono, że las losowy pozwalał uzyskać średnio najlepsze wyniki. Dla przypadków z weryfikacją na osobie pierwszej i czwartej wynik klasyfikacji był najlepszy. W pozostałych przypadkach, kiedy osiągnięty wynik nie był najlepszy, dalej trafność klasyfikacji nie odstawała znaczco od pozostałych wyników osiąganych przez klasyfikatory. Klasyfikacja za pomocą lasu losowego pozwoliła ponadto uzyskać blisko 80% trafności lub więcej w trzech przypadkach. W związku z powyższym do dalszych badań jako klasyfikator wykorzystano las losowy.

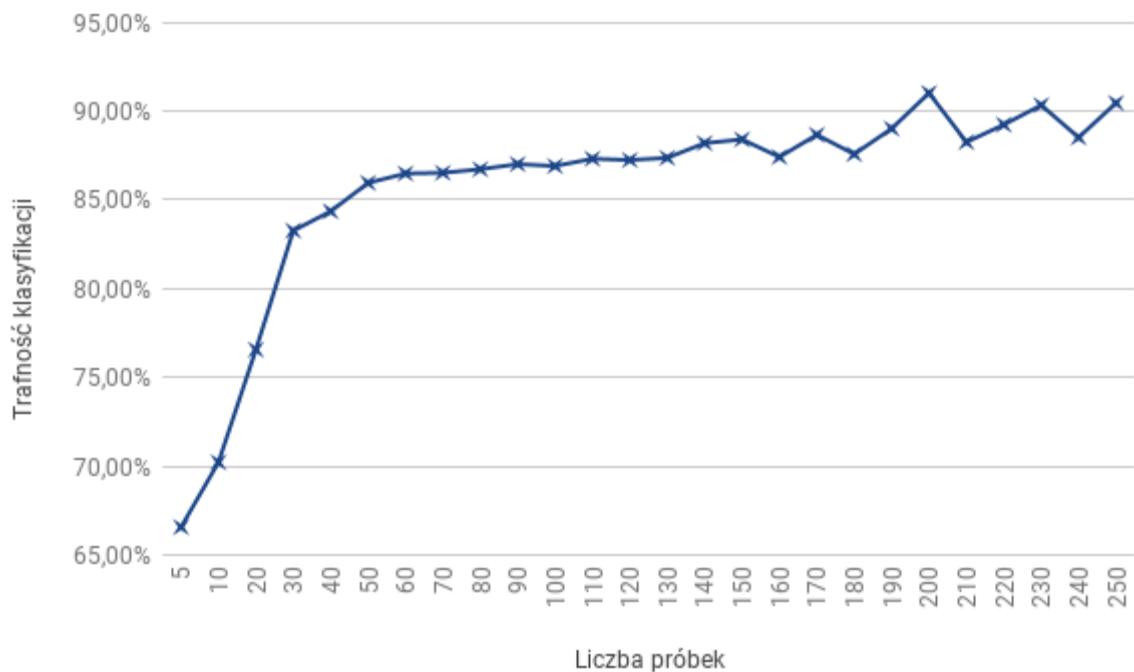
W kolejnym kroku procesu optymalizacji zweryfikowano w jaki sposób szerokość okna, w ramach którego rozpoznawana jest aktywność fizyczna wpływa na trafność klasyfikacji. W tym celu uruchomiono klasyfikację za pomocą lasu losowego dla różnych liczb prób w oknie. Próbki zebrane zostały z częstotliwością 50Hz. Długość okna weryfikowano w zakresie od około 0,1s do 5s. Dla każdego ustawienia wykonano 20 prób klasyfikacji. Tabela 4.8 przedstawia średnią trafność osiągniętą podczas tych prób, odchylenie standardowe oraz maksymalny i minimalny wynik.

Tabela 4.8. Zestawienie trafności klasyfikacji lasu losowego w zależności od rozmiaru przyjętego okna, dla którego wyliczane są wartości cech sygnału.

Liczba próbek	Trafność			
	klasyfikacji	Odchylenie	Minimum	Maksimum
5	66,57%	0,62%	65,25%	67,96%
10	70,22%	0,71%	68,58%	71,99%
20	76,56%	1,59%	73,47%	79,69%
30	83,26%	1,38%	78,95%	84,95%
40	84,35%	3,30%	71,05%	86,75%

50	85,95%	0,71%	85,15%	87,64%
60	86,48%	1,07%	84,76%	88,84%
70	86,52%	1,71%	81,65%	89,40%
80	86,72%	1,39%	84,27%	89,33%
90	87,01%	1,51%	84,11%	90,84%
100	86,90%	1,37%	84,16%	89,59%
110	87,31%	1,02%	85,32%	90,30%
120	87,24%	1,96%	81,57%	90,51%
130	87,36%	1,88%	84,75%	92,08%
140	88,19%	2,65%	85,17%	93,38%
150	88,40%	1,91%	84,80%	94,26%
160	87,41%	1,63%	85,25%	91,37%
170	88,66%	2,56%	85,06%	93,87%
180	87,57%	1,82%	84,62%	94,33%
190	89,01%	2,71%	81,12%	95,28%
200	91,01%	2,88%	86,94%	96,85%
210	88,25%	3,39%	83,49%	94,81%
220	89,23%	2,72%	85,15%	94,55%
230	90,33%	2,71%	85,13%	94,87%
240	88,51%	2,08%	85,41%	93,51%
250	90,45%	3,78%	84,27%	96,63%

Minimalna trafność klasyfikacji jaką zarejestrowano wyniosła 66,57%, podczas gdy maksymalna kilkukrotnie powyżej 95%. Rezultat pomiaru został zobrazowany na Rysunku 4.4.



Rysunek 4.4. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego w zależności od rozmiaru przyjętego okna, dla którego wyliczane są wartości cech sygnału.

Na Rysunku 4.4 widać, iż zwiększenie szerokości okna znacznie wpływa na trafność klasyfikacji w zakresie od 5 do 50 próbek. Zmiana trafności, porównując krańce przedziału, wynosi 20%. Następnie w przedziale od 50 do 130 próbek następuje stabilizacja osiąganych wyników. Wahają się one od 85% do 87,5%. Wskazany przedział charakteryzuje się także stałymi osiąganyimi rezultatami. Dla każdej szerokości okna, odchylenie standardowe wyników wyniosło poniżej 2%. Zwiększając szerokość okna do 250 próbek, osiągana trafność klasyfikacji zaczęła się wachać. Pozwalała osiągać lepsze rezultaty, jednak kosztem większej zmienności.

Mając na uwadze powyższe oraz pamiętając, że zwiększanie szerokości okna wpływa jednocześnie negatywnie na szczegółowość klasyfikacji, zdecydowano na przeprowadzenie dalszych badań bez zmiany parametru określającego liczbę próbek w oknie. Osiągana przy stu próbkach w oknie trafność jest zadowalająca i pozwala osiągać stabilne rezultaty. W przypadku wyboru zbyt długiego czasu, dokładność maleje. Przykładowo, mając dziesięciosekundowe okno, podczas którego osoba zdążyła zejść i wejść po schodach klasyfikator będzie oceniał jedynie pod kątem wytypowania jednej aktywności fizycznej. Zbyt krótkie okno powoduje z kolei brak możliwości oceny rodzaju aktywności. Dla okna trwającego ułamek sekundy dochodzi do sytuacji, gdy nie wiadomo czy dana osoba obecnie wchodzi po schodach czy stoi na nich. Do dalszych badań ustalenie szerokości okna wynosi zatem w dalszym ciągu 100 próbek, co odpowiada około dwóm sekundom.

Po weryfikacji doboru optymalnego rozmiaru okna przystąpiono do selekcji odpowiedniego zestawu wartości cech sygnału, celem zapewnienia jak najlepszej trafności klasyfikacji. Podczas tego procesu rozpatrywano 10 wartości cech. Każda z nich jest obliczana dla każdej osi oraz wartości średniej wektora. Jedno okno z obliczoną jedną cechą posiada 4 wartości, zaś gdy obliczone są wszystkie, występuje 40 wartości cech danego przedziału.

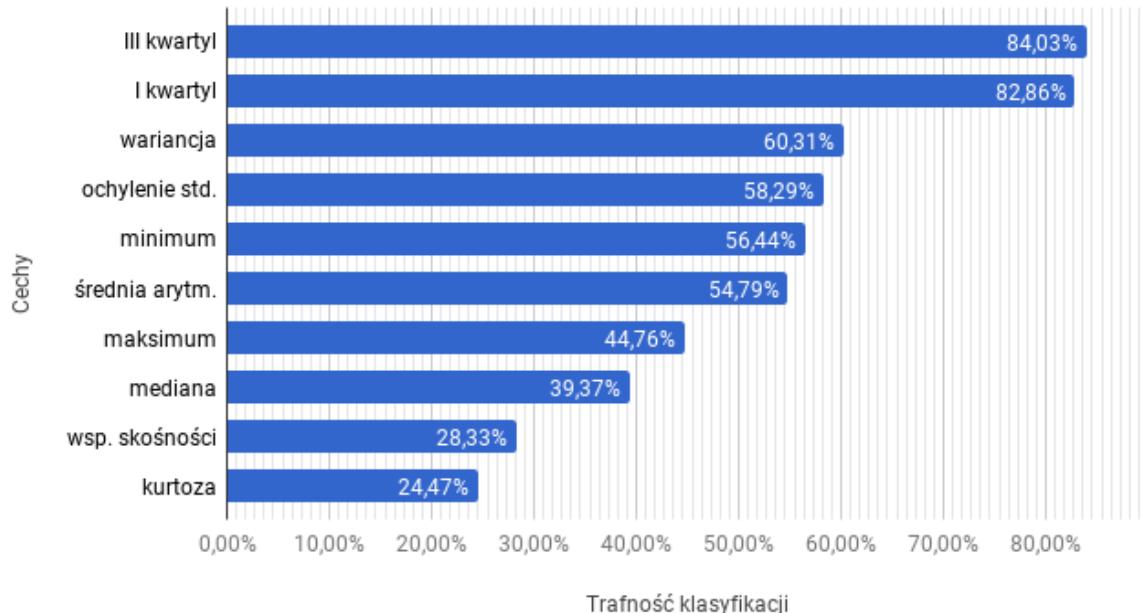
Analizę rozpoczęto obliczając trafności klasyfikacji za pomocą lasu losowego dla każdej cechy z osobna. Rezultat tych działań zaprezentowano w Tabeli 4.9.

Tabela 4.9. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu rozpoznawania aktywności fizycznych, wykorzystującego jeden parametr jako cechę sygnału.

Cechy	Trafność klasyfikacji	Odchyl.	Min	Max
<b>średnia arytm.</b>	54,79%	3,93%	50,23%	66,52%
<b>I kwartyl</b>	82,86%	3,12%	75,11%	88,01%
<b>median</b>	39,37%	3,14%	35,07%	47,06%
<b>III kwartyl</b>	84,03%	1,53%	81,00%	86,65%
<b>ochylenie std.</b>	58,29%	14,12%	43,67%	81,90%
<b>wariancja</b>	60,31%	13,43%	44,12%	78,73%
<b>minimum</b>	56,44%	3,24%	50,00%	62,90%
<b>maksimum</b>	44,76%	1,24%	42,53%	47,29%
<b>wsp. skośności</b>	28,33%	2,15%	23,53%	32,81%
<b>kurtoza</b>	24,47%	1,80%	21,95%	30,32%

Wynik badania pozwolił określić, że sama wartość trzeciego kwartylu pozwala uzyskać bardzo dobrą trafność klasyfikacji na poziomie ponad 84% przy niewiele ponad półtoraprocentowym odchyleniu standardowym. Jednocześnie podczas 20 wykonanych prób

klasyfikacji minimalna trafność wyniosła 81%, przy maksymalnej trafności klasyfikacji równej 86,65%. Z drugiej strony znajduje się kurtoza która pozwoliła osiągnąć trafność klasyfikacji na poziomie 24,47%, nieznacznie tylko większą od prawdopodobieństwa przypadkowego wskazania jeden z pięciu klas. Osiągnięte wyniki zobrazowano na Rysunku 4.5.



Rysunek 4.5. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego dla pojedynczych parametrów przyjętych jako cecha sygnału.

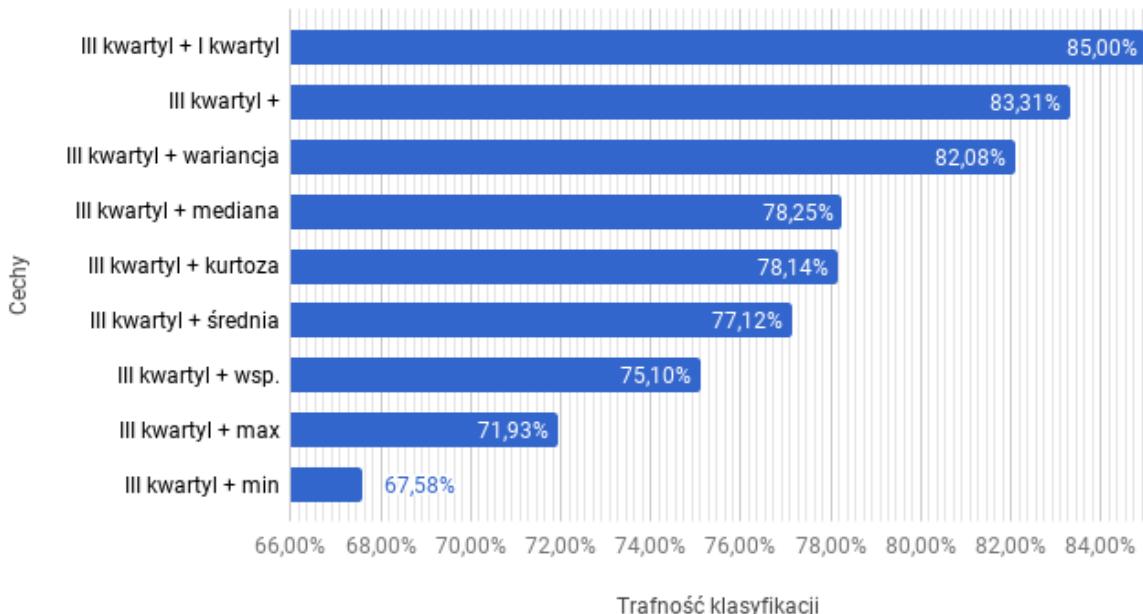
W kolejnej iteracji procesu selekcji wartości cech sygnału, do wartości trzeciego kwartyłu dodano pozostałe parametry. Utworzono zostało 9 par, dla których ponownie obliczono trafność klasyfikacji. Wynik przedstawia Tabela 4.10.

Tabela 4.10. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu rozpoznawania aktywności fizycznych, wykorzystującego dwa parametry jako cechy sygnału.

Cechy	Trafność klasyfikacji	Zmiana	Odchylenie	Min	Max
III kwartyl + średnia arytm.	77,12%	-6,91%	4,45%	69,91%	83,26%
III kwartyl + I kwartyl	85,00%	0,97%	2,13%	80,77%	87,56%
III kwartyl + mediana	78,25%	-5,78%	3,47%	71,04%	83,26%
III kwartyl + odchylenie std.	83,31%	-0,71%	1,40%	80,09%	85,52%
III kwartyl + wariancja	82,08%	-1,95%	1,71%	76,70%	85,52%
III kwartyl + min	67,58%	-16,45%	5,17%	57,01%	77,38%
III kwartyl + max	71,93%	-12,09%	4,80%	63,35%	79,86%
III kwartyl + wsp. skośności	75,10%	-8,93%	4,51%	66,29%	83,03%
III kwartyl + kurtoza	78,14%	-5,88%	3,54%	67,65%	82,35%

Jedynie jedna para wartości pozwoliła na uzyskanie trafności większej niż w przypadku, kiedy obliczana była wyłącznie wartość trzeciego kwartyłu. Najwyższa trafność klasyfikacji wzrosła o 0,97 punktu procentowego. Warto zauważyć, iż w obecnym badaniu wartość minimalna nie spadła poniżej 57% dla żadnej z kombinacji. W każdym przypadku, przynajmniej

jedna z iteracji osiągała trafność klasyfikacji powyżej 77%. Kombinacje parametrów uszeregowane według osiągniętych rezultatów przedstawia Rysunek 4.6.



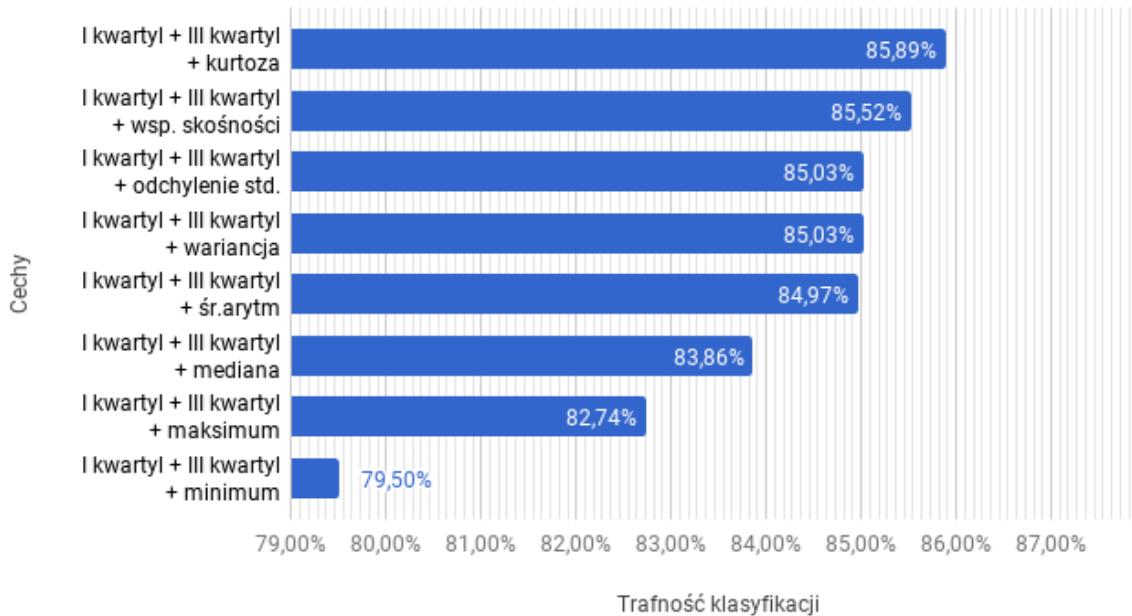
Rysunek 4.6. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego dla dwóch parametrów przyjętych jako cechy sygnału.

Kolejny krok, zgodnie z przyjętą metodyką rozpoczęto od utworzenia 8 zestawów parametrów, będących cechami sygnału. Każdy z nich zawierał wartość pierwszego i trzeciego kwartylu. Osiągnięte rezultaty przedstawiono w Tabeli 4.11.

Tabela 4.11. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu klasyfikacji aktywności fizycznych, wykorzystującego trzy parametry jako cechy sygnału.

Cechy	Trafność klasyfikacji	Odchylenie			
		Zmiana	e	Min	Max
I kwartyl + III kwartyl + śr.arytm	84,97%	-0,03%	2,12%	77,83%	87,56%
I kwartyl + III kwartyl + mediana	83,86%	-1,14%	2,75%	79,19%	88,91%
I kwartyl + III kwartyl + odchylenie std.	85,03%	0,03%	1,28%	83,26%	87,78%
I kwartyl + III kwartyl + wariancja	85,03%	0,03%	1,41%	82,13%	87,78%
I kwartyl + III kwartyl + minimum	79,50%	-5,50%	5,30%	67,19%	86,65%
I kwartyl + III kwartyl + maksimum	82,74%	-2,26%	4,41%	73,08%	88,46%
I kwartyl + III kwartyl + wsp. skośności	85,52%	0,52%	3,41%	77,38%	89,59%
I kwartyl + III kwartyl + kurtoza	85,89%	0,89%	3,02%	74,43%	88,24%

Trzecia iteracja pozwoliła osiągnąć rezultat po raz kolejny, lepszy od osiągniętego dla zestawu zawierającego dwie cechy sygnału. Wyłącznie kombinacja zawierająca wartość minimalną osiągnęła średni wynik poniżej 80%. Osiągnięte wyniki klasyfikacji zobrazowano na Rysunku 4.7.



Rysunek 4.7. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego dla trzech parametrów przyjętych jako cechy sygnału.

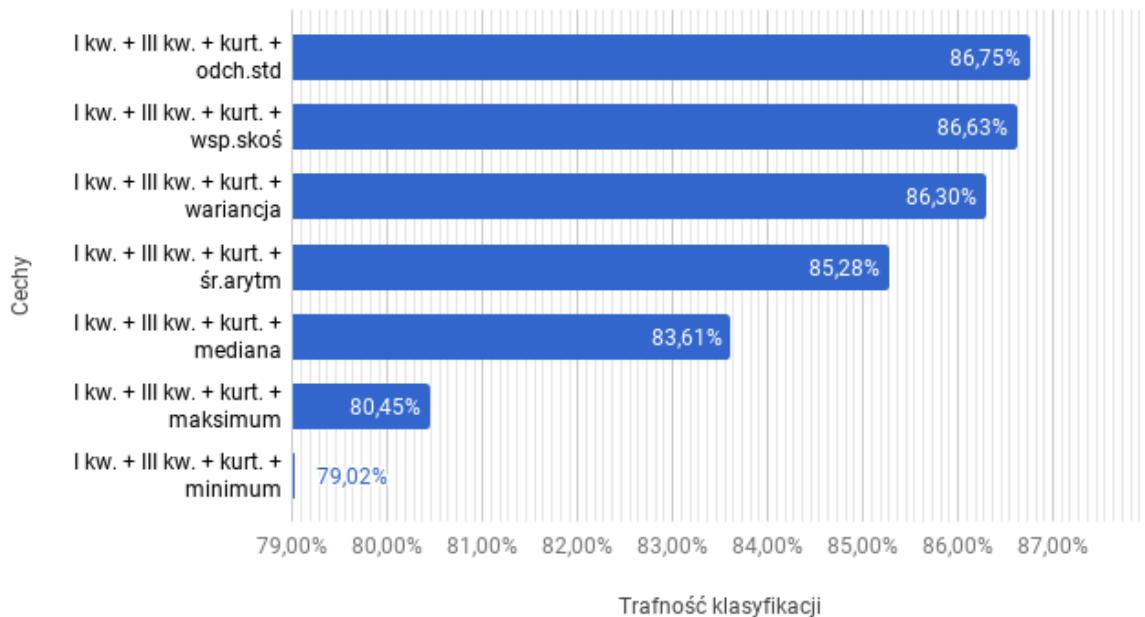
Osiągnięta trafność klasyfikacji w uległa dalszej poprawie o blisko 1%. Zdecydowano o przeprowadzeniu kolejnej iteracji. Wzięto w niej pod uwagę zestawy czterech cech zawierające cechy, które pozwoliły na osiągnięcie najwyższej trafności, którymi były wartości pierwszego i trzeciego kwartylu oraz kurtoza Wyniki kolejnej iteracji, w której pod uwagę wzięto zestawy czterech cech sygnału, przedstawią Tabela 4.12.

Tabela 4.12. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu rozpoznawania aktywności fizycznych, wykorzystującego cztery parametry jako cechy sygnału.

Cechy	Trafność klasyfikacji	Zmiana	Odchylenie	Min	Max
I kw. + III kw. + kurt. + śr.arytm	85,28%	-0,61%	1,80%	79,86%	87,33%
I kw. + III kw. + kurt. + mediana	83,61%	-2,29%	1,98%	80,09%	88,46%
I kw. + III kw. + kurt. + odch.std	86,75%	0,86%	2,04%	83,71%	92,08%
I kw. + III kw. + kurt. + wariancja	86,30%	0,41%	1,61%	83,03%	91,86%
I kw. + III kw. + kurt. + minimum	79,02%	-6,88%	7,93%	62,67%	87,78%
I kw. + III kw. + kurt. + maksimum	80,45%	-5,44%	7,33%	61,76%	91,18%
I kw. + III kw. + kurt. + wsp.skoś	86,63%	0,74%	1,88%	82,58%	89,59%

Czwarta iteracja badania trafności klasyfikacji, dla każdego zestawu cech z wyłączeniem tego zawierającego obliczone minimum, pozwoliła osiągnąć średni wynik powyżej 80 procent. Zestaw składający się z wartości pierwszego i trzeciego kwartylu, kurtozy oraz odchylenia standardowego pozwolił natomiast na osiągnięcie najlepszego do tej pory wyniku. Odnosząc się do poprzedniego zestawu cech, odchylenie standardowe zmalało, a wartość maksymalna

trafności klasyfikacji po raz kolejny wzrosła. Wskazany zestaw cech uznano zatem za najlepszy. Osiągnięte rezultaty uszeregowane malejąco przedstawione zostały na Rysunku 4.8.



Rysunek 4.8. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego dla czterech parametrów przyjętych jako cechy sygnału.

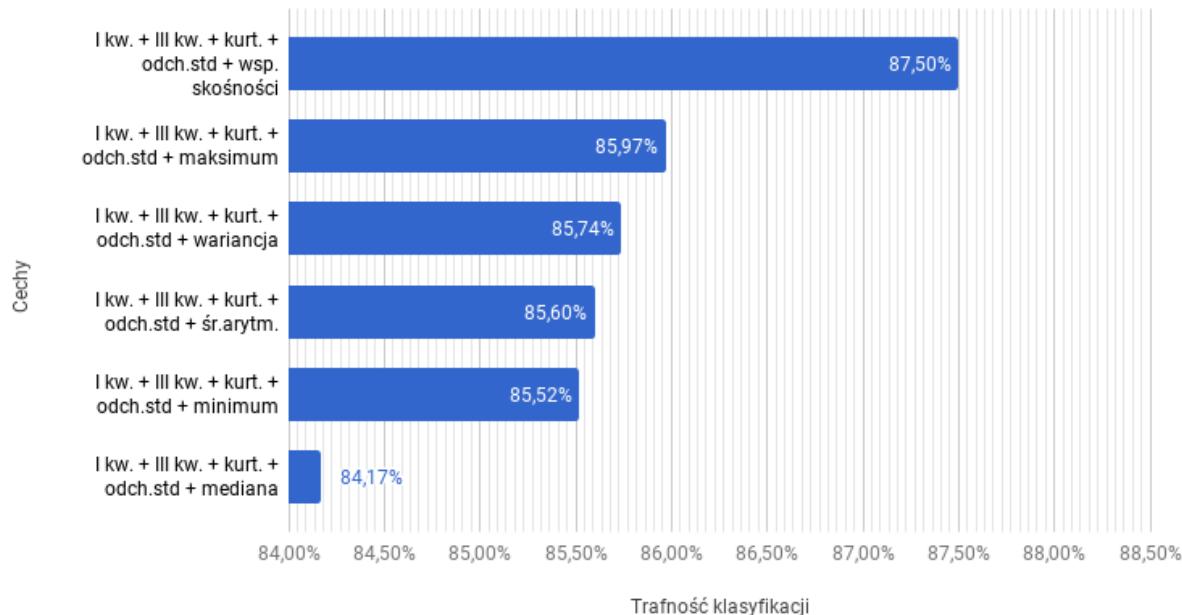
Kolejna iteracja w każdym zestawie zawierała obliczone wartości pierwszego i trzeciego kwartylu, kurtozę oraz odchylenie standardowe. Dołączając do tego zestawu pozostałe metryki powstało 6 zestawów danych, dla których ponownie obliczono cztery wartości i porównano z wartościami osiągniętymi w poprzedniej iteracji. Wynik przedstawiono w Tabeli 4.13

Tabela 4.13. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu rozpoznawania aktywności fizycznych, wykorzystującego pięć parametrów jako cechy sygnału.

Cechy	Trafność klasyfikacji	Zmiana	Odchylenie	Min	Max
I kw. + III kw. + kurt. + odch.std + śr.arytm.	85,60%	-1,15%	1,25%	83,48%	88,24%
I kw. + III kw. + kurt. + odch.std + mediana	84,17%	-2,58%	2,44%	78,28%	86,88%
I kw. + III kw. + kurt. + odch.std + wariancja	85,74%	-1,02%	1,37%	82,58%	88,01%
I kw. + III kw. + kurt. + odch.std + minimum	85,52%	-1,23%	2,17%	80,32%	88,91%

I kw. + III kw. + kurt. + odch.std + maksimum	85,97%	-0,78%	2,06%	82,13%	91,40%
I kw. + III kw. + kurt. + odch.std + wsp. skośności	87,50%	0,75%	3,08%	83,26%	94,12%

Wszystkie kombinacje wartości cech sygnału, z wyjątkiem jednej, zanotowały spadek trafności klasyfikacji względem wcześniej osiągniętego, najlepszego wyniku. Poprawę osiąganych rezultatów odnotowano w przypadku zestawu z dodanym współczynnikiem skośności. Komplet wyników dla zestawów pięciu parametrów przedstawiony został na Rysunku 4.9.



Rysunek 4.9. Wykres obrazujący trafność klasyfikacji przy wykorzystaniu lasu losowego dla pięciu parametrów przyjętych jako cechy sygnału.

Do kolejnej iteracji, w związku z przyjętą metodologią trafiły zestawy cech, z których każdy zawierał wartość pierwszego i trzeciego kwartylu, kurtozę, odchylenie standardowe i medianę. Kombinacje z pozostałymi cechami pozwoliły na utworzenie 5 zestawów. Obliczoną trafność klasyfikacji dla każdego z nich przedstawiono w Tabeli 4.14.

Tabela 4.14. Zestawienie trafności klasyfikacji lasu losowego, w zadaniu rozpoznawania aktywności fizycznych, wykorzystującego sześć parametrów jako cechy sygnału.

Cechy	Trafność klasyfikacji	Zmiana	Odchylenie	Min	Max
I kw. + III kw. + kurt. + odch.std + wsp. skośności + śr.arytm.	86,15%	-1,35%	1,92%	82,58%	90,27%
I kw. + III kw. + kurt. + odch.std + wsp. Skośności + mediana	85,34%	-2,16%	1,62%	82,58%	88,24%

I kw. + III kw. + kurt. + odch.std + wsp. Skośności + wariancja	87,19%	-0,31%	2,18%	81,90%	92,08%
I kw. + III kw. + kurt. + odch.std + wsp. Skośności +minimum	85,58%	-1,92%	2,52%	80,32%	88,69%
I kw. + III kw. + kurt. + odch.std + wsp. Skośności + maksimum	85,14%	-2,36%	2,43%	80,32%	89,37%

W iteracji nie zanotowano poprawy żadnego ze wskaźników, względem wartości osiąganych wcześniej, dlatego też zaprzestano dalszego doboru parametrów. Zestaw składający się z obliczonych wartości pierwszego i trzeciego kwartyłu, kurtozy, odchylenia standardowego i mediany dał najlepsze rezultaty w zaprezentowanym badaniu. Osiągnięta wartość średnia trafności klasyfikacji na poziomie 87,5% przy odchyleniu standardowym dla 20 iteracji wynoszącym 3,08% jest satysfakcjonująca i pozwala na bardzo dokładne określenie wykonywanej aktywności fizycznej z sygnału przyspieszenia dla użytkownika. Przyglądając się bliżej najlepszemu z osiągniętych wyników można zweryfikować rozpoznawanie których aktywności stanowi największy problem dla klasyfikatora. Macierz pomyłek obrazująca wyniki klasyfikacji poszczególnych aktywności fizycznych została przedstawiona poniżej w Tabeli 4.15

Tabela 4.15. Macierz pomyłek przedstawiająca rezultaty klasyfikacji poszczególnych aktywności fizycznych dla klasyfikatora wykorzystującego zestaw cech, który pozwolił osiągnąć najlepszą trafność klasyfikacji.

Przewidywanie						
	bezczytność	chodzenie	schodzenie ze schodów	wchodzenie po schodach	bieganie	
Stan faktyczny	<b>bezczytność</b>	100,00%	0,00%	0,00%	0,00%	0,00%
	<b>chodzenie</b>	0,03%	93,31%	0,70%	1,88%	4,09%
	<b>schodzenie ze schodów</b>	0,00%	21,61%	71,02%	7,37%	0,00%
	<b>wchodzenie po schodach</b>	0,10%	39,90%	28,40%	31,60%	0,00%
	<b>bieganie</b>	0,00%	0,06%	0,00%	0,00%	99,94%

Tabela 4.15 przedstawia trafność klasyfikacji poszczególnych aktywności fizycznych w ujęciu procentowym. Z tabeli wynika, iż klasyfikator bezbłędnie radzi sobie z rozpoznawaniem bezczytności. Każde z okien zawierające tę aktywność fizyczną zostało rozpoznane poprawnie. Zdecydowanie najgorszą trafność odnotowano w przypadku rozpoznawania wchodzenia po schodach. W tym przypadku osiągnięto trafność wynoszącą jedynie 31,6%. Więcej danych odnoszących się do wchodzenia po schodach zostało rozpoznanych jako chodzenie. Weryfikując wszystkie aktywności fizyczne widać, iż rozpoznawanie bezczytności, chodzenia i biegania, było zadaniem prostym i wykonanym z wysoką trafnością klasyfikacji. Aktywności fizyczne na schodach, często były rozpoznawane jednak jako chodzenie. W wielu przypadkach doszło także do rozpoznania wchodzenia po schodach jako schodzenie i odwrotnie.

## 5. Podsumowanie

W ramach pracy utworzono aplikację działającą pod kontrolą systemu Android, która posłużyła za narzędzie rejestrujące sygnał przyspieszenia. Aplikacja powstała w języku Java i pozwoliła na zapisywanie danych sygnału do plików CSV. Za jej pomocą zebrano dane o aktywnościach fizycznych czterech osób. Każda z osób zarejestrowała zestawy danych odpowiadające pięciu klasom aktywności fizycznych: bezczynności, chodzeniu, bieganiu, wchodzeniu i schodzeniu ze schodów.

Zebrane dane zostały poddane analizie, celem wykrywania aktywności fizycznych użytkowników smartfonów. Badanie wykonane w ramach pracy pozwoliło na przeprowadzenie procesu klasyfikacji sygnału z akcelerometru w sposób optymalny. W ramach przeprowadzonej optymalizacji ustalono, iż najlepszym rozwiązaniem jest wyodrębnianie cech sygnału przyspieszenia zarejestrowanego w trzech kierunkach w połączeniu z obliczonym modelem wektora przyspieszenia. Osiągnięte wyniki pozwoliły na uzyskanie średniej trafności klasyfikacji na poziomie powyżej 70% dla wykorzystania lasu losowego, perceptronu wielowarstwowego oraz klasyfikatora k najbliższych sąsiadów. Wskazane wyniki zostały osiągnięte dla klasyfikatorów uczonych na sygnatach pozyskanych od trzech osób, a weryfikowane na sygnale przyspieszenia od czwartej osoby. Najlepszą trafność klasyfikacji pozwoliło osiągnąć wykorzystanie lasu losowego, dla którego przeprowadzono dalsze kroki badania. W przypadku, w którym nauka i weryfikacja klasyfikatorów opierała się na danych pochodzących od jednej osoby, trafność wielokrotnie wyniosła blisko 100 procent. W realnych zastosowaniach taka sytuacja zazwyczaj jednak nie występuje.

Następnie zweryfikowano wpływ liczby próbek w oknie, na osiągane rezultaty klasyfikacji. Zaobserwowano wzrost trafności klasyfikacji wraz ze zwiększaniem szerokości okna. Wzrost trafności klasyfikacji od 50 do 250 próbek był nieznaczny i w raz z dodawaniem kolejnych próbek, trafność klasyfikacji podlegała coraz większym wahaniom. Stwierdzono zatem, że przy częstotliwości wynoszącej 50 próbek na sekundę, rozmiar okna powyżej 50 próbek jest odpowiedni. Mając na uwadze szczegółowość klasyfikacji okno nie powinno być też zbyt długie. Do dalszych badań ustalono szerokość okna na 100 próbek.

W ostatnim etapie optymalizacji procesu klasyfikacji zweryfikowano jaki wpływ na osiągane rezultaty ma zestaw wyodrębnianych cech sygnału. Przeprowadzając badanie polegające na stopniowym zwiększaniu liczby obliczanych dla okna cech, osiągnięto trafność klasyfikacji lasu losowego równą 87,5%, przy odchyleniu standardowym 3,08%. Optymalnym zestawem parametrów w zadaniu rozpoznawania aktywności fizycznej użytkownika określono wartości pierwszego i trzeciego kwartylu, kurtozy, odchylenia standardowego oraz współczynnika skośności. Odpowiedni dobór parametrów, wyodrębnianych jako cechy sygnału, wpływał znacząco na osiągane wyniki, pozwalając na poprawienie początkowego wyniku o ponad cztery punkty procentowe. W szczególności zauważono, iż wzrost liczby parametrów, nie wpływa bezpośrednio na lepszą trafność klasyfikacji.

Wysoka dokładność klasyfikacji aktywności fizycznej użytkownika na podstawie sygnału przyspieszenia zebranego za pomocą telefonu komórkowego pozwala myśleć o wykorzystaniu mechanizmu w realnych zastosowaniach. Rozwiązanie wykorzystujące akcelerometr idealnie sprawdzi się w miejscach, gdzie istotna jest energooszczędność i tym samym możliwość przeprowadzania klasyfikacji w długich oknach czasowych. W tym wypadku akcelerometr sprawdza się znacznie lepiej niż sygnał pochodzący z GPS. Akcelerometr ma także przewagę nad śledzeniem aktywności fizycznej za pomocą kamer. Znajduje się

w niemalże każdym smartfonie, a samo urządzenie jest bezpośrednio związane z jego użytkownikiem. Jest zatem łatwo dostępny i stosunkowo tani. Rozwiązania oparte na kamerach są znacznie droższe. Dodatkowo analizując aktywność fizyczną wielu osób w jednym czasie może powodować niejednoznaczności, jeśli chodzi o przypisanie rezultatów do poszczególnych osób.

W związku z powyższym klasyfikacja na podstawie przyspieszenia znajduje zastosowanie w aplikacjach fitness monitorujących aktywność fizyczną użytkownika. Opisywany mechanizm może także znaleźć zastosowanie w audycie czasu pracy zatrudnionych osób w czasie dnia. Powyższe zastosowania z pewnością jednak nie wyczerpują tematu. Wraz z rozwojem techniki dostępne są coraz nowsze czujniki, jednak akcelerometr z pewnością w dalszym ciągu będzie znajdował zastosowanie w wykrywaniu aktywności fizycznych.

## 6. Bibliografia

- [1] Mikowska, Monika, POLSKA.JEST.MOBI, 2015, s.5-8
- [2] Kantar TNS, Aktywność fizyczna Polaków, 2017
- [3] FitBit, Should you really take 10,000 steps a day?, [online] 2018  
Dostęp z:  
[https://www.argos.co.uk/wcsstore/argos/en\\_GB/images/suppliershop/sitebuilder/shops/fitbit/downloads/steps.pdf](https://www.argos.co.uk/wcsstore/argos/en_GB/images/suppliershop/sitebuilder/shops/fitbit/downloads/steps.pdf)
- [4] Fetters, Aleisha, How Many Daily Steps Do You Really Need for Better Health?, [online] 2016,  
Dostęp z: <https://blog.myfitnesspal.com/many-daily-steps-really-need-better-health>
- [5] International Data Corporation, Smartphone Market Share 2017 Q1, [online] 2017,  
Dostęp z: <https://www.idc.com/promo/smartphone-market-share/os>
- [6] Vectornav Embedded Navigation Solutions, Accelerometer, [online] 2018  
Dostęp z: <https://www.vectornav.com/support/library/accelerometer>
- [7] Gujarati, Paresh, What is Accelerometer and how does it work on smartphones, [online] 2013,  
Dostęp z: <http://www.techulator.com/resources/8930-How-does-smart-phone-accelerometer-work.aspx>
- [8] Garmin, Garmin Connect Mobile, [online] 2017  
Dostęp z: <https://buy.garmin.com/pl-PL/PL/p/125677>
- [9] Google Play, Huawei Health, [online] 2018  
Dostęp z: <https://play.google.com/store/apps/details?id=com.huawei.health>
- [10] Apple, Health, [online] 2018  
Dostęp z: <https://www.apple.com/ios/health/>
- [11] Klasnja, P., Consolvo, S., Choudhury, T., Beckwith R., Hightower, J., Exploring Privacy Concerns about Personal Sensing, [online] 2009,  
Dostęp z: [http://pac.cs.cornell.edu/pubs/Pervasive09\\_sensor\\_privacy.pdf](http://pac.cs.cornell.edu/pubs/Pervasive09_sensor_privacy.pdf)
- [12] Bureau International des Poids et Mesures, Base units, [online] 2018  
Dostęp z: <https://www.bipm.org/en/measurement-units/base-units.html>
- [13] Lathia, Neal, Mining Smartphone sensor data with python, PyData London 2016, [online] 2016,  
Dostęp z: [https://www.youtube.com/watch?v=Bidl8\\_1ikiQ](https://www.youtube.com/watch?v=Bidl8_1ikiQ)
- [14] Erdaş, Ç., Atasoy, I., Açıci, K., Oğul, H., Integrating features for accelerometer-based activity recognition, s.5
- [15] Kunze, K., Lukowicz, P., Junker, H., Tröster, G., Where Am I: Recognizing on-body positions of wearable sensors, Berlin, 2005, Springer, s.264-275, ISBN 978-3-540-25896-4

- [16] Scipy.org, numpy.mean, [online] 2018  
Dostęp z: <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.mean.html>
- [17] ekonometria.4me.pl, Statystyka opisowa i matematyczna, [online] 2018  
Dostęp z: <http://www.ekonometria.4me.pl/statystyka2.htm>
- [18] Scipy.org, numpy.var, [online] 2018  
Dostęp z: <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.var.html>
- [19] Scipy.org, numpy.std, [online] 2018  
Dostęp z: <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.std.html>
- [20] Wolfram Mathworld, Minimum, [online] 2018  
Dostęp z: <http://mathworld.wolfram.com/Minimum.html>
- [21] Wolfram Mathworld, Maximum, [online] 2018  
Dostęp z: <http://mathworld.wolfram.com/Maximum.html>
- [22] Scipy.org, scipy.stats.skew, [online] 2018  
Dostęp z: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.skew.html>
- [23] Scipy.org, scipy.stats.kurtosis, [online] 2018  
Dostęp z: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kurtosis.html>

## **Spis załączników**

1. Kod programu w języku Python wykorzystany do analizy danych.
2. Kod aplikacji Android do zapisywania danych z akcelerometru.

# Załączniki

## Kod programu w języku Python wykorzystany do analizy sygnału

### Plik prepare\_data.py

```
import mgr.data.readrawdata as rd
import mgr.calc.signal as sig
import csv

STANDING_COLOR = 'gray'
WALKING_COLOR = 'green'
DOWNSTAIRS_COLOR = 'brown'
UPSTAIRS_COLOR = 'blue'
RUNNING_COLOR = 'red'

""" PERSON_1 - prepare data - start """
STANDING_PERSON_1 = rd.read('mgr/data/resources/person_1/standing.csv')
STANDING_PERSON_1['magnitude'] = sig.magnitude(STANDING_PERSON_1)

WALKING_PERSON_1 = rd.read('mgr/data/resources/person_1/walking.csv')
WALKING_PERSON_1['magnitude'] = sig.magnitude(WALKING_PERSON_1)

DOWNSTAIRS_PERSON_1 = rd.read('mgr/data/resources/person_1/downstairs.csv')
DOWNSTAIRS_PERSON_1['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_1)

UPSTAIRS_PERSON_1 = rd.read('mgr/data/resources/person_1/upstairs.csv')
UPSTAIRS_PERSON_1['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_1)

RUNNING_PERSON_1 = rd.read('mgr/data/resources/person_1/running.csv')
RUNNING_PERSON_1['magnitude'] = sig.magnitude(RUNNING_PERSON_1)

activities_person_1 = [STANDING_PERSON_1, WALKING_PERSON_1,
                      DOWNSTAIRS_PERSON_1, UPSTAIRS_PERSON_1, RUNNING_PERSON_1]
features_file_person_1 = 'mgr/data/features/person_1/Features.csv'

with open(features_file_person_1, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_1)):
        for f in sig.extract_features(activities_person_1[i]):
            rows.writerow([i] + f)

""" PERSON_1 - prepare data - stop """
""" PERSON_2 - prepare data - start """

STANDING_PERSON_2 = rd.read('mgr/data/resources/person_2/standing.csv')
STANDING_PERSON_2['magnitude'] = sig.magnitude(STANDING_PERSON_2)

WALKING_PERSON_2 = rd.read('mgr/data/resources/person_2/walking.csv')
WALKING_PERSON_2['magnitude'] = sig.magnitude(WALKING_PERSON_2)

DOWNSTAIRS_PERSON_2 = rd.read('mgr/data/resources/person_2/downstairs.csv')
DOWNSTAIRS_PERSON_2['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_2)

UPSTAIRS_PERSON_2 = rd.read('mgr/data/resources/person_2/upstairs.csv')
```

```

UPSTAIRS_PERSON_2['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_2)

RUNNING_PERSON_2 = rd.read('mgr/data/resources/person_2/running.csv')
RUNNING_PERSON_2['magnitude'] = sig.magnitude(RUNNING_PERSON_2)

activities_person_2 = [STANDING_PERSON_2, WALKING_PERSON_2,
DOWNSTAIRS_PERSON_2, UPSTAIRS_PERSON_2, RUNNING_PERSON_2]
features_file_person_2 = 'mgr/data/features/person_2/Features.csv'

with open(features_file_person_2, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_2)):
        for f in sig.extract_features(activities_person_2[i]):
            rows.writerow([i] + f)

""" PERSON_2 - prepare data - stop """
""" PERSON_3 - prepare data - start """

STANDING_PERSON_3 = rd.read('mgr/data/resources/person_3/standing.csv')
STANDING_PERSON_3['magnitude'] = sig.magnitude(STANDING_PERSON_3)

WALKING_PERSON_3 = rd.read('mgr/data/resources/person_3/walking.csv')
WALKING_PERSON_3['magnitude'] = sig.magnitude(WALKING_PERSON_3)

DOWNSTAIRS_PERSON_3 = rd.read('mgr/data/resources/person_3/downstairs.csv')
DOWNSTAIRS_PERSON_3['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_3)

UPSTAIRS_PERSON_3 = rd.read('mgr/data/resources/person_3/upstairs.csv')
UPSTAIRS_PERSON_3['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_3)

RUNNING_PERSON_3 = rd.read('mgr/data/resources/person_3/running.csv')
RUNNING_PERSON_3['magnitude'] = sig.magnitude(RUNNING_PERSON_3)

activities_person_3 = [STANDING_PERSON_3, WALKING_PERSON_3,
DOWNSTAIRS_PERSON_3, UPSTAIRS_PERSON_3, RUNNING_PERSON_3]
features_file_person_3 = 'mgr/data/features/person_3/Features.csv'

with open(features_file_person_3, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_3)):
        for f in sig.extract_features(activities_person_3[i]):
            rows.writerow([i] + f)

""" PERSON_3 - prepare data - stop """
""" PERSON_4 - prepare data - start """

STANDING_PERSON_4 = rd.read('mgr/data/resources/person_4/standing.csv')
STANDING_PERSON_4['magnitude'] = sig.magnitude(STANDING_PERSON_4)

WALKING_PERSON_4 = rd.read('mgr/data/resources/person_4/walking.csv')
WALKING_PERSON_4['magnitude'] = sig.magnitude(WALKING_PERSON_4)

DOWNSTAIRS_PERSON_4 = rd.read('mgr/data/resources/person_4/downstairs.csv')
DOWNSTAIRS_PERSON_4['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_4)

UPSTAIRS_PERSON_4 = rd.read('mgr/data/resources/person_4/upstairs.csv')
UPSTAIRS_PERSON_4['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_4)

RUNNING_PERSON_4 = rd.read('mgr/data/resources/person_4/running.csv')
RUNNING_PERSON_4['magnitude'] = sig.magnitude(RUNNING_PERSON_4)

```

```

""" PERSON_4 - prepare data - stop """

activities_person_4 = [STANDING_PERSON_4, WALKING_PERSON_4,
DOWNSTAIRS_PERSON_4, UPSTAIRS_PERSON_4, RUNNING_PERSON_4]
features_file_person_4 = 'mgr/data/features/person_4/Features.csv'

with open(features_file_person_4, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_4)):
        for f in sig.extract_features(activities_person_4[i]):
            rows.writerow([i] + f)

```

## Plik test.py

```

import numpy as np
from prepare_data import *

features_person_1 = np.loadtxt(features_file_person_1, delimiter=",")
features_person_2 = np.loadtxt(features_file_person_2, delimiter=",")
features_person_3 = np.loadtxt(features_file_person_3, delimiter=",")
features_person_4 = np.loadtxt(features_file_person_4, delimiter=",")
features_all = np.concatenate([features_person_1, features_person_2,
features_person_3, features_person_4])
features_person_1_2_3 = np.concatenate([features_person_1,
features_person_2, features_person_3])
features_person_1_2_4 = np.concatenate([features_person_1,
features_person_2, features_person_4])
features_person_1_3_4 = np.concatenate([features_person_1,
features_person_3, features_person_4])
features_person_2_3_4 = np.concatenate([features_person_2,
features_person_3, features_person_4])

print('K-Neighbors Classifier ', sig.test_and_learn_knn_cls(features_person_1_2_3, features_person_4))
print('Decision Tree Classifier', sig.test_and_learn_decision_tree_cls(features_person_1_2_3,
features_person_4))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_person_1_2_3,
features_person_4))
print('MLP Classifier           ',
sig.test_and_learn_mlp_cls(features_person_1_2_3, features_person_4))
print('GaussianNB               ',
sig.test_and_learn_mlp_cls(features_person_1_2_3, features_person_4))

print('\nTest Classifiers - learn: 1,2,4 test: 3')
print('K-Neighbors Classifier ', sig.test_and_learn_knn_cls(features_person_1_2_4, features_person_3))
print('Decision Tree Classifier', sig.test_and_learn_decision_tree_cls(features_person_1_2_4,
features_person_3))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_person_1_2_4,
features_person_3))
print('MLP Classifier           ',
sig.test_and_learn_mlp_cls(features_person_1_2_4, features_person_3))
print('GaussianNB               ',
sig.test_and_learn_mlp_cls(features_person_1_2_4, features_person_3))

```

```

print('\nTest Classifiers - learn: 1,3,4 test: 2')
print('K-Neighbors Classifier ',
sig.test_and_learn_knn_cls(features_person_1_3_4, features_person_2))
print('Decision Tree Classifier',
sig.test_and_learn_decision_tree_cls(features_person_1_3_4,
features_person_2))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_person_1_3_4,
features_person_2))
print('MLP Classifier           ',
sig.test_and_learn_mlp_cls(features_person_1_3_4, features_person_2))
print('GaussianNB               ',
sig.test_and_learn_mlp_cls(features_person_1_3_4, features_person_2))

print('\nTest Classifiers - learn: 2,3,4 test: 1')
print('K-Neighbors Classifier ',
sig.test_and_learn_knn_cls(features_person_2_3_4, features_person_1))
print('Decision Tree Classifier',
sig.test_and_learn_decision_tree_cls(features_person_2_3_4,
features_person_1))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_person_2_3_4,
features_person_1))
print('MLP Classifier           ',
sig.test_and_learn_mlp_cls(features_person_2_3_4, features_person_1))
print('GaussianNB               ',
sig.test_and_learn_mlp_cls(features_person_2_3_4, features_person_1))

```

## Plik signal.py

```

import numpy as np
from scipy.stats import skew, kurtosis
from sklearn.model_selection import train_test_split
import math

from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB


def magnitude(activity):
    res = np.sqrt(activity['xAxis'] * activity['xAxis'] +
                  activity['yAxis'] * activity['yAxis'] +
                  activity['zAxis'] * activity['zAxis'])
    return res

def divide_signal(df, size=100):
    start = 0
    while start < df.count():
        yield start, start + size
        start += (size / 2)

def window_values(axis, start, end):
    start = int(start)

```

```

end = int(end)
p25 = np.percentile(axis[start:end], 25)
median = np.percentile(axis[start:end], 50)
p75 = np.percentile(axis[start:end], 75)
std = axis[start:end].std()
var = axis[start:end].var()
if math.isnan(var):
    std = 0
    var = 0
return [
    p25,
    kurtosis(axis[start:end]),
    median,
    axis[start:end].min(),
    axis[start:end].mean(),
    p75,
    std,
    var,
    axis[start:end].max(),
    skew(axis[start:end]),
]

```

```

def extract_features(activity):
    for (start, end) in divide_signal(activity['timestamp']):
        activity_values = ['xAxis', 'yAxis', 'zAxis']
        activity_values = ['magnitude']
        activity_values = ['xAxis', 'yAxis', 'zAxis', 'magnitude']

        features = []
        for axis in activity_values:
            features += window_values(activity[axis], start, end)
    yield features

```

```

def test_and_learn_classifier(classifier, features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        af_train, af_test, am_train, am_test =
train_test_split(activity_features, activity_markers, test_size=.25)
        classifier.fit(af_train, am_train)
        res = classifier.score(af_test, am_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results),
        np.min(results),
        np.max(results)
    ]

```

```

def test_dummy_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test\
        = train_test_split(activity_features, activity_markers,

```

```

test_size=.25)

classifier = DummyClassifier()
classifier.fit(activity_features_learn, activity_markers_learn)
res = classifier.score(activity_features_test, activity_markers_test)
return res

def test_knn_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test \
        = train_test_split(activity_features, activity_markers,
test_size=.25)

    classifier = KNeighborsClassifier()
    classifier.fit(activity_features_learn, activity_markers_learn)
    res = classifier.score(activity_features_test, activity_markers_test)
    return res

def test_decision_tree_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test \
            = train_test_split(activity_features, activity_markers,
test_size=.25)
        classifier = DecisionTreeClassifier()
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results)
    ]

def test_random_forest_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test \
            = train_test_split(activity_features, activity_markers,
test_size=.25)
        classifier = RandomForestClassifier()
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results)
    ]

```

```

]

def test_mlp_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test \
            = train_test_split(activity_features, activity_markers,
test_size=.25)
        classifier = MLPClassifier(max_iter=1000)
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results)
    ]

def test_gaussian_nb_cls_one_set(features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        activity_features_learn, activity_features_test,
activity_markers_learn, activity_markers_test \
            = train_test_split(activity_features, activity_markers,
test_size=.25)
        classifier = GaussianNB()
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results)
    ]

def test_classifier(classifier, features_data):
    activity_features = features_data[:, 1:]
    activity_markers = features_data[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        res = classifier.score(activity_features, activity_markers)
        results.append(res)
    return [
        np.mean(results),
        np.std(results),
    ]

def test_and_learn_dummy_cls(features_learn, features_test):
    activity_features_learn = features_learn[:, 1:]

```

```

activity_markers_learn = features_learn[:, 0]
activity_features_test = features_test[:, 1:]
activity_markers_test = features_test[:, 0]
classifier = DummyClassifier()
classifier.fit(activity_features_learn, activity_markers_learn)
res = classifier.score(activity_features_test, activity_markers_test)
return res

def test_and_learn_knn_cls(features_learn, features_test):
    activity_features_learn = features_learn[:, 1:]
    activity_markers_learn = features_learn[:, 0]
    activity_features_test = features_test[:, 1:]
    activity_markers_test = features_test[:, 0]
    classifier = KNeighborsClassifier()
    classifier.fit(activity_features_learn, activity_markers_learn)
    res = classifier.score(activity_features_test, activity_markers_test)
    return res

def test_and_learn_decision_tree_cls(features_learn, features_test):
    activity_features_learn = features_learn[:, 1:]
    activity_markers_learn = features_learn[:, 0]
    activity_features_test = features_test[:, 1:]
    activity_markers_test = features_test[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        classifier = DecisionTreeClassifier()
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
                               activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results)
    ]

def test_and_learn_random_forest_cls(features_learn, features_test):
    activity_features_learn = features_learn[:, 1:]
    activity_markers_learn = features_learn[:, 0]
    activity_features_test = features_test[:, 1:]
    activity_markers_test = features_test[:, 0]
    results = []
    loops = 20
    for i in range(0, loops):
        classifier = RandomForestClassifier()
        classifier.fit(activity_features_learn, activity_markers_learn)
        res = classifier.score(activity_features_test,
                               activity_markers_test)
        results.append(res)
    return [
        np.mean(results),
        np.std(results),
        np.min(results),
        np.max(results)
    ]

def test_and_learn_mlp_cls(features_learn, features_test):

```

```

activity_features_learn = features_learn[:, 1:]
activity_markers_learn = features_learn[:, 0]
activity_features_test = features_test[:, 1:]
activity_markers_test = features_test[:, 0]
results = []
loops = 20
for i in range(0, loops):
    classifier = MLPClassifier(max_iter=1000)
    classifier.fit(activity_features_learn, activity_markers_learn)
    res = classifier.score(activity_features_test,
activity_markers_test)
    results.append(res)
return [
    np.mean(results),
    np.std(results)
]

def test_and_learn_gaussian_nb_cls(features_learn, features_test):
activity_features_learn = features_learn[:, 1:]
activity_markers_learn = features_learn[:, 0]
activity_features_test = features_test[:, 1:]
activity_markers_test = features_test[:, 0]
results = []
loops = 20
for i in range(0, loops):
    classifier = GaussianNB()
    classifier.fit(activity_features_learn, activity_markers_learn)
    res = classifier.score(activity_features_test,
activity_markers_test)
    results.append(res)
return [
    np.mean(results),
    np.std(results)
]

```

## Plik visualize.py

```

import matplotlib.pyplot as plt
import mgr.calc.signal as sig

def graph_activity(activity, color, title):
    """Prints a graph of passed activity. There are 3 plots for each
axis"""
    fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(15, 10),
sharex=True)
    plot_graph(ax0, activity['timestamp'], activity['xAxis'], 'X-axis',
color)
    plot_graph(ax1, activity['timestamp'], activity['yAxis'], 'Y-axis',
color)
    plot_graph(ax2, activity['timestamp'], activity['zAxis'], 'Z-axis',
color)
    plt.suptitle(title)
    plt.show()

def plot_graph(axis, xAxis, yAxis, title, color):
    """Function for single graph plotting"""

```

```

        axis.plot(xAxis, yAxis, color=color)
        axis.set_title(title)
        axis.xaxis.set_visible(False)
        axis.set_xlim([min(xAxis), max(xAxis)])
        axis.grid(True)

    def graph_magnitude(activity, color, title):
        """Prints a activity magnitude graph"""
        fig, axs = plt.subplots(nrows=1, figsize=(15, 10), sharex=True)
        plot_graph(axs, activity['timestamp'], activity['magnitude'], title,
color)
        plt.subplots_adjust(hspace=0.2)
        plt.show()

    def graph_divided_signal(activity, color, title):
        fig, (ax0, ax1, ax2, ax3) = plt.subplots(nrows=4, figsize=(15, 10),
sharex=True)
        plot_graph(ax0, activity['timestamp'], activity['xAxis'], 'X-
axis(divided)', color)
        plot_graph(ax1, activity['timestamp'], activity['yAxis'], 'Y-
axis(divided)', color)
        plot_graph(ax2, activity['timestamp'], activity['zAxis'], 'Z-
axis(divided)', color)
        plot_graph(ax3, activity['timestamp'], activity['magnitude'],
'magnitude(divided)', color)

        for (start, end) in sig.divide_signal(activity['timestamp']):
            ax0.axvline(activity['timestamp'][start], color='black')
            ax1.axvline(activity['timestamp'][start], color='black')
            ax2.axvline(activity['timestamp'][start], color='black')
            ax3.axvline(activity['timestamp'][start], color='black')

        plt.suptitle(title)
        plt.show()

```

## Plik main.py

```

from scipy.signal import lfilter
import mgr.calc.signal as sig
import matplotlib.pyplot as plt
import matplotlib as mpl
import csv
import datetime
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import mgr.data.readrawdata as rd
import mgr.visualization.visualize as vis

# Step 1 - read and draw raw data

```

```

STANDING_COLOR = 'gray'
WALKING_COLOR = 'green'
DOWNSTAIRS_COLOR = 'brown'

```

```

UPSTAIRS_COLOR = 'blue'
RUNNING_COLOR = 'red'

#Read standing activity data from csv and draw a graph
STANDING = rd.read('mgr/data/resources/person_1/standing.csv')
vis.graph_activity(STANDING, STANDING_COLOR, 'Standing')

#Read walking activity data from csv and draw a graph
WALKING = rd.read('mgr/data/resources/person_1/walking.csv')
vis.graph_activity(WALKING, WALKING_COLOR, 'Walking')

#Read downstairs activity data from csv and draw a graph
DOWNSTAIRS = rd.read('mgr/data/resources/person_1/downstairs.csv')
vis.graph_activity(DOWNSTAIRS, DOWNSTAIRS_COLOR, 'Downstairs')

#Read upstairs activity data from csv and draw a graph
UPSTAIRS = rd.read('mgr/data/resources/person_1/upstairs.csv')
vis.graph_activity(UPSTAIRS, UPSTAIRS_COLOR, 'Upstairs')

#Read running activity data from csv and draw a graph
RUNNING = rd.read('mgr/data/resources/person_1/running.csv')
vis.graph_activity(RUNNING, RUNNING_COLOR, 'Running')

# Step 3 - calculate magnitude and draw a graphs

#Calculate magnitudes
STANDING['magnitude'] = sig.magnitude(STANDING)
WALKING['magnitude'] = sig.magnitude(WALKING)
DOWNSTAIRS['magnitude'] = sig.magnitude(DOWNSTAIRS)
UPSTAIRS['magnitude'] = sig.magnitude(UPSTAIRS)
RUNNING['magnitude'] = sig.magnitude(RUNNING)

#Draw a graphs of magnitudes
vis.graph_magnitude(STANDING, STANDING_COLOR, 'Standing - magnitude')
vis.graph_magnitude(WALKING, WALKING_COLOR, 'Walking - magnitude')
vis.graph_magnitude(DOWNSTAIRS, DOWNSTAIRS_COLOR, 'Downstairs - magnitude')
vis.graph_magnitude(UPSTAIRS, UPSTAIRS_COLOR, 'Upstairs - magnitude')
vis.graph_magnitude(RUNNING, RUNNING_COLOR, 'Running - magnitude')

# Step 4 - draw divided signal

vis.graph_divided_signal(STANDING, STANDING_COLOR, 'Standing - divided')
vis.graph_divided_signal(WALKING, WALKING_COLOR, 'Walking - divided')
vis.graph_divided_signal(DOWNSTAIRS, DOWNSTAIRS_COLOR, 'Downstairs - divided')
vis.graph_divided_signal(UPSTAIRS, UPSTAIRS_COLOR, 'Upstairs - divided')
vis.graph_divided_signal(RUNNING, RUNNING_COLOR, 'Running - divided')
"""

# Step 5 - extract features
"""
activities = [STANDING, WALKING, DOWNSTAIRS, UPSTAIRS, RUNNING]
output_file_path = 'mgr/data/features/person_1/Features.csv'

with open(output_file_path, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities)):
        for f in sig.extract_features(activities[i]):
            rows.writerow([i] + f)

```

```

# Step 6 - test classifier

features = np.loadtxt('mgr/data/features/person_1/Features.csv',
delimiter=',')
k_neighbors_cls = KNeighborsClassifier()
decision_tree_cls = DecisionTreeClassifier()
random_forest_cls = RandomForestClassifier()
mlp_cls = MLPClassifier()
gaussian_nb_cls = GaussianNB()

print('K-Neighbors Classifier ', sig.test_and_learn_classifier(k_neighbors_cls, features))
print('Decision Tree Classifier ', sig.test_and_learn_classifier(decision_tree_cls, features))
print('Random Forest Classifier ', sig.test_and_learn_classifier(random_forest_cls, features))
print('MLP Classifier ', sig.test_and_learn_classifier(mlp_cls, features))
print('GaussianNB ', sig.test_and_learn_classifier(gaussian_nb_cls, features))

```

## Plik statistics.py

```

import mgr.data.readrawdata as rd
import mgr.calc.signal as sig
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import datetime
import csv

""" PERSON_1 - prepare data - start """
STANDING_PERSON_1 = rd.read('mgr/data/resources/person_1/standing.csv')
STANDING_PERSON_1['magnitude'] = sig.magnitude(STANDING_PERSON_1)

WALKING_PERSON_1 = rd.read('mgr/data/resources/person_1/walking.csv')
WALKING_PERSON_1['magnitude'] = sig.magnitude(WALKING_PERSON_1)

DOWNSTAIRS_PERSON_1 = rd.read('mgr/data/resources/person_1/downstairs.csv')
DOWNSTAIRS_PERSON_1['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_1)

UPSTAIRS_PERSON_1 = rd.read('mgr/data/resources/person_1/upstairs.csv')
UPSTAIRS_PERSON_1['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_1)

RUNNING_PERSON_1 = rd.read('mgr/data/resources/person_1/running.csv')
RUNNING_PERSON_1['magnitude'] = sig.magnitude(RUNNING_PERSON_1)

""" PERSON_1 - prepare data - stop """

activities_person_1 = [STANDING_PERSON_1, WALKING_PERSON_1,
DOWNSTAIRS_PERSON_1, UPSTAIRS_PERSON_1, RUNNING_PERSON_1]
output_file_path_person_1 = 'mgr/data/features/person_1/Features.csv'

```

```

with open(output_file_path_person_1, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_1)):
        for f in sig.extract_features(activities_person_1[i]):
            rows.writerow([i] + f)

features_person_1 = np.loadtxt('mgr/data/features/person_1/Features.csv',
                             delimiter=",") 

print('\nTest Classifiers on PERSON_1 learned with data collected by
PERSON_1')
print('Dummy Classifier      ',
sig.test_dummy_cls_one_set(features_person_1))
print('K-Neighbors Classifier  ',
sig.test_knn_cls_one_set(features_person_1))
print('Decision Tree Classifier',
sig.test_decision_tree_cls_one_set(features_person_1))
print('Random Forest Classifier',
sig.test_random_forest_cls_one_set(features_person_1))
print('MLP Classifier        ',
sig.test_mlp_cls_one_set(features_person_1))
print('GaussianNB             ',
sig.test_gaussian_nb_cls_one_set(features_person_1))

""" PERSON_2 - prepare data - start """
STANDING_PERSON_2 = rd.read('mgr/data/resources/person_2/standing.csv')
STANDING_PERSON_2['magnitude'] = sig.magnitude(STANDING_PERSON_2)

WALKING_PERSON_2 = rd.read('mgr/data/resources/person_2/walking.csv')
WALKING_PERSON_2['magnitude'] = sig.magnitude(WALKING_PERSON_2)

DOWNSTAIRS_PERSON_2 = rd.read('mgr/data/resources/person_2/downstairs.csv')
DOWNSTAIRS_PERSON_2['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_2)

UPSTAIRS_PERSON_2 = rd.read('mgr/data/resources/person_2/upstairs.csv')
UPSTAIRS_PERSON_2['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_2)

RUNNING_PERSON_2 = rd.read('mgr/data/resources/person_2/running.csv')
RUNNING_PERSON_2['magnitude'] = sig.magnitude(RUNNING_PERSON_2)

""" PERSON_2 - prepare data - stop """

activities_person_2 = [STANDING_PERSON_2, WALKING_PERSON_2,
                      DOWNSTAIRS_PERSON_2, UPSTAIRS_PERSON_2, RUNNING_PERSON_2]
output_file_path_person_2 = 'mgr/data/features/person_2/Features.csv'

with open(output_file_path_person_2, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_2)):
        for f in sig.extract_features(activities_person_2[i]):
            rows.writerow([i] + f)

features_person_2 = np.loadtxt('mgr/data/features/person_2/Features.csv',
                             delimiter=",")

print('\nTest Classifiers on PERSON_2 learned with data collected by
PERSON_2')
print('Dummy Classifier      ',
sig.test_dummy_cls_one_set(features_person_2))
print('K-Neighbors Classifier  ',

```

```

sig.test_knn_cls_one_set(features_person_2))
print('Decision Tree Classifier',
sig.test_decision_tree_cls_one_set(features_person_2))
print('Random Forest Classifier',
sig.test_random_forest_cls_one_set(features_person_2))
print('MLP Classifier      ',
sig.test_mlp_cls_one_set(features_person_2))
print('GaussianNB          ',
sig.test_gaussian_nb_cls_one_set(features_person_2))

""" PERSON_3 - prepare data - start """
STANDING_PERSON_3 = rd.read('mgr/data/resources/person_3/standing.csv')
STANDING_PERSON_3['magnitude'] = sig.magnitude(STANDING_PERSON_3)

WALKING_PERSON_3 = rd.read('mgr/data/resources/person_3/walking.csv')
WALKING_PERSON_3['magnitude'] = sig.magnitude(WALKING_PERSON_3)

DOWNSTAIRS_PERSON_3 = rd.read('mgr/data/resources/person_3/downstairs.csv')
DOWNSTAIRS_PERSON_3['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_3)

UPSTAIRS_PERSON_3 = rd.read('mgr/data/resources/person_3/upstairs.csv')
UPSTAIRS_PERSON_3['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_3)

RUNNING_PERSON_3 = rd.read('mgr/data/resources/person_3/running.csv')
RUNNING_PERSON_3['magnitude'] = sig.magnitude(RUNNING_PERSON_3)

""" PERSON_3 - prepare data - stop """

activities_person_3 = [STANDING_PERSON_3, WALKING_PERSON_3,
DOWNSTAIRS_PERSON_3, UPSTAIRS_PERSON_3, RUNNING_PERSON_3]
output_file_path_person_3 = 'mgr/data/features/person_3/Features.csv'

with open(output_file_path_person_3, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_3)):
        for f in sig.extract_features(activities_person_3[i]):
            rows.writerow([i] + f)

features_person_3 = np.loadtxt('mgr/data/features/person_3/Features.csv',
delimiter=",")

print('\nTest Classifiers on PERSON_3 learned with data collected by
PERSON_3')
print('Dummy Classifier      ',
sig.test_dummy_cls_one_set(features_person_3))
print('K-Neighbors Classifier  ',
sig.test_knn_cls_one_set(features_person_3))
print('Decision Tree Classifier',
sig.test_decision_tree_cls_one_set(features_person_3))
print('Random Forest Classifier',
sig.test_random_forest_cls_one_set(features_person_3))
print('MLP Classifier      ',
sig.test_mlp_cls_one_set(features_person_3))
print('GaussianNB          ',
sig.test_gaussian_nb_cls_one_set(features_person_3))

""" PERSON_4 - prepare data - start """
STANDING_PERSON_4 = rd.read('mgr/data/resources/person_4/standing.csv')
STANDING_PERSON_4['magnitude'] = sig.magnitude(STANDING_PERSON_4)

```

```

WALKING_PERSON_4 = rd.read('mgr/data/resources/person_4/walking.csv')
WALKING_PERSON_4['magnitude'] = sig.magnitude(WALKING_PERSON_4)

DOWNSTAIRS_PERSON_4 = rd.read('mgr/data/resources/person_4/downstairs.csv')
DOWNSTAIRS_PERSON_4['magnitude'] = sig.magnitude(DOWNSTAIRS_PERSON_4)

UPSTAIRS_PERSON_4 = rd.read('mgr/data/resources/person_4/upstairs.csv')
UPSTAIRS_PERSON_4['magnitude'] = sig.magnitude(UPSTAIRS_PERSON_4)

RUNNING_PERSON_4 = rd.read('mgr/data/resources/person_4/running.csv')
RUNNING_PERSON_4['magnitude'] = sig.magnitude(RUNNING_PERSON_4)

""" PERSON_4 - prepare data - stop """

activities_person_4 = [STANDING_PERSON_4, WALKING_PERSON_4,
DOWNSTAIRS_PERSON_4, UPSTAIRS_PERSON_4, RUNNING_PERSON_4]
output_file_path_person_4 = 'mgr/data/features/person_4/Features.csv'

with open(output_file_path_person_4, 'w') as features_file:
    rows = csv.writer(features_file)
    for i in range(0, len(activities_person_4)):
        for f in sig.extract_features(activities_person_4[i]):
            rows.writerow([i] + f)

features_person_4 = np.loadtxt('mgr/data/features/person_4/Features.csv',
delimiter=",")

print('\nTest Classifiers on PERSON_4 learned with data collected by
PERSON_4')
print('Dummy Classifier      ',
sig.test_dummy_cls_one_set(features_person_4))
print('K-Neighbors Classifier  ',
sig.test_knn_cls_one_set(features_person_4))
print('Decision Tree Classifier',
sig.test_decision_tree_cls_one_set(features_person_4))
print('Random Forest Classifier',
sig.test_random_forest_cls_one_set(features_person_4))
print('MLP Classifier         ',
sig.test_mlp_cls_one_set(features_person_4))
print('GaussianNB             ',
sig.test_gaussian_nb_cls_one_set(features_person_4))

features_all = np.concatenate([features_person_1, features_person_2,
features_person_3, features_person_4])

dummy_cls_all = DummyClassifier()
k_neighbors_cls_all = KNeighborsClassifier()
decision_tree_cls_all = DecisionTreeClassifier()
random_forest_cls_all = RandomForestClassifier()
mlp_cls_all = MLPClassifier()
gaussian_nb_cls_all = GaussianNB()

print('\nTest Classifiers on PERSON_1 learned with all data')
print('Dummy Classifier      ',
sig.test_and_learn_dummy_cls(features_all, features_person_1))
print('K-Neighbors Classifier  ', sig.test_and_learn_knn_cls(features_all,
features_person_1))
print('Decision Tree Classifier',

```

```

sig.test_and_learn_decision_tree_cls(features_all, features_person_1))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_all, features_person_1))
print('MLP Classifier      ', sig.test_and_learn_mlp_cls(features_all,
features_person_1))
print('GaussianNB           ',
sig.test_and_learn_gaussian_nb_cls(features_all, features_person_1))

print('\nTest Classifiers on PERSON_2 learned with all data')
print('Dummy Classifier     ',
sig.test_and_learn_dummy_cls(features_all, features_person_2))
print('K-Neighbors Classifier', sig.test_and_learn_knn_cls(features_all,
features_person_2))
print('Decision Tree Classifier',
sig.test_and_learn_decision_tree_cls(features_all, features_person_2))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_all, features_person_2))
print('MLP Classifier      ', sig.test_and_learn_mlp_cls(features_all,
features_person_2))
print('GaussianNB           ',
sig.test_and_learn_gaussian_nb_cls(features_all, features_person_2))

print('\nTest Classifiers on PERSON_3 learned with all data')
print('Dummy Classifier     ',
sig.test_and_learn_dummy_cls(features_all, features_person_3))
print('K-Neighbors Classifier', sig.test_and_learn_knn_cls(features_all,
features_person_3))
print('Decision Tree Classifier',
sig.test_and_learn_decision_tree_cls(features_all, features_person_3))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_all, features_person_3))
print('MLP Classifier      ', sig.test_and_learn_mlp_cls(features_all,
features_person_3))
print('GaussianNB           ',
sig.test_and_learn_gaussian_nb_cls(features_all, features_person_3))

print('\nTest Classifiers on PERSON_4 learned with all data')
print('Dummy Classifier     ',
sig.test_and_learn_dummy_cls(features_all, features_person_4))
print('K-Neighbors Classifier', sig.test_and_learn_knn_cls(features_all,
features_person_4))
print('Decision Tree Classifier',
sig.test_and_learn_decision_tree_cls(features_all, features_person_4))
print('Random Forest Classifier',
sig.test_and_learn_random_forest_cls(features_all, features_person_4))
print('MLP Classifier      ', sig.test_and_learn_mlp_cls(features_all,
features_person_4))
print('GaussianNB           ',
sig.test_and_learn_gaussian_nb_cls(features_all, features_person_4))

```

## Kod aplikacji Android do zapisywania danych z akcelerometru.

### FilePickActivity.java

```

package pl.gebert.lifetrack;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.content.FileProvider;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class FilePickActivity extends Activity {
    private ListView fileListView;
    private ArrayAdapter<String> fileListViewAdapter;
    private File filesDir;
    File[] files;
    ArrayList<String> listItems = new ArrayList<String>();
    Map<String, String> filenamePathMap = new HashMap<String, String>();
    Intent shareIntent;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_file_pick);
        shareIntent = new Intent(Intent.ACTION_SEND);
        fileListView = findViewById(R.id.fileList);

        filesDir = getExternalFilesDir(null);
        files = filesDir.listFiles();

        for (File file : files) {
            listItems.add(file.getName());
            filenamePathMap.put(file.getName(), file.getAbsolutePath());
        }
        Collections.sort(listItems);
        fileListViewAdapter = new ArrayAdapter<String>(this,
R.layout.file_list_item, listItems);
        fileListView.setAdapter(fileListViewAdapter);
        fileListView.setClickable(true);
        fileListView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> adapterView,
View view, int position, long rowId) {
                    File requestFile = new
File(filenamePathMap.get(listItems.get(position)));
                    try {
                        Uri fileUri =
FileProvider.getUriForFile(FilePickActivity.this,"pl.gebert.lifetrack.FileP
rovider", requestFile);
                        if (fileUri != null) {
                            shareIntent =
createFileShareIntent(fileUri);

```

```

startActivity(Intent.createChooser(shareIntent, "Share file"));

FilePickActivity.this.setResult(Activity.RESULT_OK, shareIntent);
} else {
    shareIntent.setDataAndType(null, "");
}

FilePickActivity.this.setResult(RESULT_CANCELED, shareIntent);
}

} catch (IllegalArgumentException e) {
    Log.e("FilePickActivity",
        "The selected file can't be shared: " +
e.getMessage());
} catch (Exception e) {
    Log.e("FilePickActivity",
        "Unexpected exception: " +
e.getMessage());
}
})
);
}

private Intent crateFileShareIntent(Uri fileUri){
    Intent intent= new Intent(Intent.ACTION_SEND);
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    intent.setDataAndType(fileUri,
getContentResolver().getType(fileUri));
    intent.putExtra(Intent.EXTRA_STREAM, fileUri);
    return intent;
}
}

```

## MainActivity.java

```

package pl.gebert.lifetrack;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.PowerManager;
import android.support.annotation.NonNull;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

import com.google.common.io.Files;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;

```

```

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.LinkedList;
import java.util.UUID;

import pl.gebert.lifetrack.data.SensorData;

public class MainActivity extends Activity implements
OnClickListener, SensorEventListener {

    //Managers
    private SensorManager sensorManager;
    private PowerManager.WakeLock wakeLock;

    //UI Elements
    private Button buttonStart;
    private Button buttonStop;
    private Button buttonSave;
    private Button buttonReset;
    private Button buttonFiles;
    private TextView stepCount;

    //File saving fields
    private File file;
    private ProgressDialog progressBar;
    private int progressBarStatus = 0;
    private Handler progressBarHandler = new Handler();
    private static final String fileNameSuffix = "_lt.csv";
    private LinkedList<SensorData> collectedData = new LinkedList<>();

    //Step fields
    private float startStepCount;
    private float actualStepCount;
    private boolean isFirstStep;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
        buttonStart = findViewById(R.id.buttonStart);
        buttonStop = findViewById(R.id.buttonStop);
        buttonSave = findViewById(R.id.buttonSave);
        buttonReset = findViewById(R.id.buttonReset);
        buttonFiles = findViewById(R.id.buttonFiles);
        stepCount = findViewById(R.id.stepCount);

        buttonStart.setOnClickListener(this);
        buttonStop.setOnClickListener(this);
        buttonSave.setOnClickListener(this);
        buttonReset.setOnClickListener(this);
        buttonFiles.setOnClickListener(this);

        buttonStart.setEnabled(true);
        buttonStop.setEnabled(false);
        buttonSave.setEnabled(false);
        buttonReset.setEnabled(false);
        buttonFiles.setEnabled(true);
    }
}

```

```

    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.buttonStart:
                buttonStartPressed();
                break;
            case R.id.buttonStop:
                buttonStopPressed();
                break;
            case R.id.buttonSave:
                buttonSavePressed();
                break;
            case R.id.buttonReset:
                buttonResetPressed();
                break;
            case R.id.buttonFiles:
                buttonFilesPressed();
                break;
            default:
                break;
        }
    }

    private void buttonStartPressed() {
        buttonStart.setEnabled(false);
        buttonStop.setEnabled(true);
        buttonSave.setEnabled(false);
        buttonReset.setEnabled(false);
        wakeLock = createWakeLock();
        Sensor accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        Sensor stepCounter =
sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
        isFirstStep = true;
        actualStepCount = 0.0f;
        sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_GAME);
        sensorManager.registerListener(this, stepCounter,
SensorManager.SENSOR_DELAY_NORMAL);
        file = new File(getExternalFilesDir(null), generateFileName());
        return;
    }

    private void buttonStopPressed() {
        buttonStart.setEnabled(true);
        buttonStop.setEnabled(false);
        buttonSave.setEnabled(true);
        buttonReset.setEnabled(true);
        sensorManager.unregisterListener(this);
        wakeLock.release();
    }

    private void buttonSavePressed() {
        buttonStart.setEnabled(true);
        buttonStop.setEnabled(false);
        buttonSave.setEnabled(false);
        buttonReset.setEnabled(false);
        saveCollectedData();
    }
}

```

```

private void buttonResetPressed() {
    buttonStart.setEnabled(true);
    buttonStop.setEnabled(false);
    buttonSave.setEnabled(false);
    buttonReset.setEnabled(false);
    collectedData.clear();
}

private void buttonFilesPressed() {
    Intent filesIntent = new Intent(this, FilePickActivity.class);
    startActivity(filesIntent);
}

@Override
public void onSensorChanged(SensorEvent event) {
    Sensor sensor = event.sensor;
    if(sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float x = event.values[0]; // Acceleration force along the x
        axis m/s^2
        float y = event.values[1]; // Acceleration force along the y
        axis m/s^2
        float z = event.values[2]; // Acceleration force along the z
        axis m/s^2
        collectedData.add(new SensorData(x, y, z, actualStepCount));
    }
    else if(sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
        if(isFirstStep) {
            startStepCount = (int) event.values[0];
            isFirstStep = false;
        }
        actualStepCount = event.values[0] - startStepCount;
        stepCount.setText(String.valueOf(actualStepCount));
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@NonNull
private String generateFileName() {
    String date = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    date += "_" + UUID.randomUUID().toString().replace("-", "").substring(0,10);
    return date + fileNameSuffix;
}

private void saveCollectedData() {
    prepareProgressBar();
    progressBar.show();

    new Thread(new Runnable() {
        public void run() {
            file = new File(getExternalFilesDir(null),
generateFileName());
            try {
                saveActivityFile();
            } catch (InterruptedException e) {
                Log.e("MainActivity",

```

```

        "The thread is interrupted: " +
e.getMessage());
    } catch (IOException e) {
        Log.e("MainActivity",
              "The file can't be created: " +
e.getMessage());
    } catch (Exception e) {
        Log.e("MainActivity",
              "Unexpected exception " + e.getMessage());
    }
    progressBar.dismiss();
}
}).start();

}

private void saveActivityFile() throws IOException,
InterruptedException {
    double dataSize = collectedData.size();
    double progress = 1;
    for(SensorData sd : collectedData){
        progressBarStatus = (int) (progress/dataSize * 100);
        progress++;
        Files.append(sd.toString(), file, Charset.defaultCharset());

        progressBarHandler.post(new Runnable() {
            public void run() {
                progressBar.setProgress(progressBarStatus);
            }
        });
    }
    Thread.sleep(1000); //sleep at the end of saving
}

private void prepareProgressBar() {
    progressBar = new ProgressDialog(this);
    progressBar.setCancelable(true);
    progressBar.setMessage("Save in progress...");
    progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    progressBar.setProgress(0);
    progressBar.setMax(100);
    progressBarStatus = 0;
}

private PowerManager.WakeLock createWakeLock() {
    PowerManager pm = (PowerManager)
getSystemService(Context.POWER_SERVICE);
    PowerManager.WakeLock wakeLock =
pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "accel_wake_lock");
    wakeLock.acquire();
    return wakeLock;
}

}

```