**Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie**

**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

Praca dyplomowa inżynierska

# Implementacja algorytmu detekcji osi pojazdów samochodowych w oparciu o profile R i X

# Implementation of algorithm for vehicle axles detection based on R and X-profile

Autor: Wojciech Gumuła
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: dr inż. Zbigniew Marszałek

Kraków, 2016

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

.....

# Spis treści

**Wstęp** ... 3
**1. Czujniki indukcyjne pętlowe w pomiarach parametrów ruchu pojazdów** ... 5
**2. Cel i zakres realizacji projektu** ... 9
2.1. Przyjęte założenia ... 9
2.2. Przyjęte ograniczenia ... 12
2.3. Interfejsy wejścia i wyjścia programu ... 13
**3. Algorytm detekcji liczby osi pojazdu** ... 17
**4. Detekcja długości pojazdu i położenia osi w jego bryle. Weryfikacja poprawności działania algorytmu** ... 27
4.1. Weryfikacja poprawności działania algorytmu przy użyciu sygnałów z czujników piezoelektrycznych ... 27
4.2. Algorytm detekcji przybliżonej długości pojazdu i położenia osi w jego bryle ... 28
**5. Analiza skuteczności algorytmu** ... 33
**6. Opis wykonanego projektu** ... 39
6.1. Wymagania i instalacja ... 39
6.2. Metody uruchamiania ... 41
6.3. Obsługiwane parametry ... 42
6.4. Przykładowe aplikacje ... 43
6.5. Interfejs graficzny ... 45
**7. Analiza pracy programu** ... 47
7.1. Poprawność algorytmu ... 47
7.2. Stabilność pracy programu ... 49
7.3. Wydajność pracy programu ... 50
**Podsumowanie** ... 54
**Bibliografia** ... 54

# Wstęp

Zadanie wykrywania osi pojazdów samochodowych w ruchu drogowym wykonywane jest przy użyciu czujników różnego typu. Wśród najpopularniejszych metod wymienić można piezoelektryczne czujniki nacisku i czujniki kwarcowe. Znane są również aplikacje wykorzystujące metody wizyjne [13], niezależnie lub we współpracy z detektorami innego typu.

Pośród innych parametrów, które podlegają analizie wymienić należy prędkość, kolor oraz wymiary pojazdu i jego wagę (lub nacisk wywierany przez poszczególne osie). Uzyskanie tych informacji wymaga często wykorzystania czujników różnych typów. Konieczna bywa również kategoryzacja pojazdów na podstawie ich cech - na przykład liczby osi lub długości.

Uzyskiwane w ten sposób informacje znajdują wiele zastosowań. Systemy badania wagi pojazdu *WIM* (ang. weigh-in-motion) wykorzystywane są przez służby policji do wykrywania pojazdów przeciążonych i ich eliminacji z dróg, zwiększając w ten sposób bezpieczeństwo. Czujniki określające liczbę osi pozwalają na budowę systemów automatycznego naliczania opłat na płatnych odcinkach dróg. Układy detekcji pojazdów wykorzystywane są do budowy oprogramowania sterującego sygnalizacją świetlną na skrzyżowaniach. Ponadto, zbiór danych zawierający informacje o natężeniu ruchu oraz parametrach pojazdów pozwala na analizę przepływu pojazdów i zaplanowanie rozwoju infrastruktury drogowej.

Pracownicy Katedry Metrologii i Elektroniki Akademii Górniczo-Hutniczej w swych badaniach poruszają temat wykorzystania czujników indukcyjnych w celu badania parametrów pojazdu. Użycie sensorów tego typu można uzasadnić ich dużą trwałością, niską ceną i niedużym skomplikowaniem instalacji. Pozwalają na badanie szeregu parametrów pojazdów, takich jak prędkość, długość czy liczba osi, oraz analizę zbiorczą strumienia, na przykład natężenia ruchu czy odległości czasowych pomiędzy kolejnymi pojazdami. Algorytm wykorzystujący czujniki indukcyjne, który zostanie zaimplementowany, pozwala na detekcję liczby osi pojazdu z dokładnością ponad 95%, cechując się przy tym niewielką złożonością obliczeniową [10].

Zagadnienie wykorzystania czujników indukcyjnych do badania parametrów pojazdów samochodowych wciąż nie zostało omówione w sposób obszerny. Zaprojektowanie oprogramowania umożliwiającego wydajną analizę uzyskiwanych ze stano-

wisk pomiarowych danych pozwoli na rozwój zagadnienia oraz zademonstruje sposób praktycznego wykorzystania systemu czujników przez użytkownika.

## Cel i zakres pracy

Celem pracy jest implementacja algorytmu detekcji osi pojazdów samochodowych na podstawie profili $R$ i $X$ pochodzących z czujników indukcyjnych pętlowych. W ramach pracy zrealizowane zostanie oprogramowanie do detekcji osi pojazdów na podstawie profili indukcyjnych, wykorzystując do tego język programowania C [33] w środowisku Linux [35]. Algorytmu rozbudowany będzie o funkcje określania położeń osi oraz estymacji długości pojazdu. Zaprojektowane zostaną funkcje charakterystyczne dla oprogramowania GNU-Coreutils [23], w tym obsługa wejść i wyjść, kodów błędów, potoków oraz argumentów wiersza poleceń [16, 37, 22, 9].

Zaprojektowane zostaną narzędzia umożliwiające pracę programu w trybie ciągłym oraz jego testowanie przy użyciu języka Python [30]. Zademonstrowana zostanie możliwość rozbudowy programu na przykładzie modułu oferującego *GUI* - graficzny interfejs użytkownika. Projekt zgodny będzie ze standardem języka C oraz POSIX [17].

Prezentowany materiał zgromadzony został w siedmiu rozdziałach.

Rozdział pierwszy zawiera opis czujnika indukcyjnego pętlowego oraz przedstawienie jego roli w pomiarach parametrów pojazdów samochodowych.

Rozdział drugi przedstawia założenia oraz zakres realizowanego projektu, wymienione są kluczowe funkcje programu oraz wykorzystany interfejs.

W rozdziale trzecim omawia algorytm detekcji osi pojazdu, wraz z procedurą poszukiwania brakujących osi, która znajduje zastosowanie w przypadku pracy z danymi niepozbawionymi zakłóceń oraz metodę detekcji podniesionej osi, umożliwiającą wykrywanie pojazdów pięcioosiowych poruszających się z uniesioną osią.

W rozdziale czwartym opisane są moduły, które dodają kolejne funkcje do algorytmu detekcji osi. Są to algorytmy detekcji położenia osi wewnątrz pojazdu oraz estymacji jego długości.

Rozdział piąty zawiera analizę skuteczności algorytmu dla wyselekcjonowanych kategorii pojazdów - samochodów osobowych, dostawczych oraz ciężarowych, z uwzględnieniem różnej liczby osi.

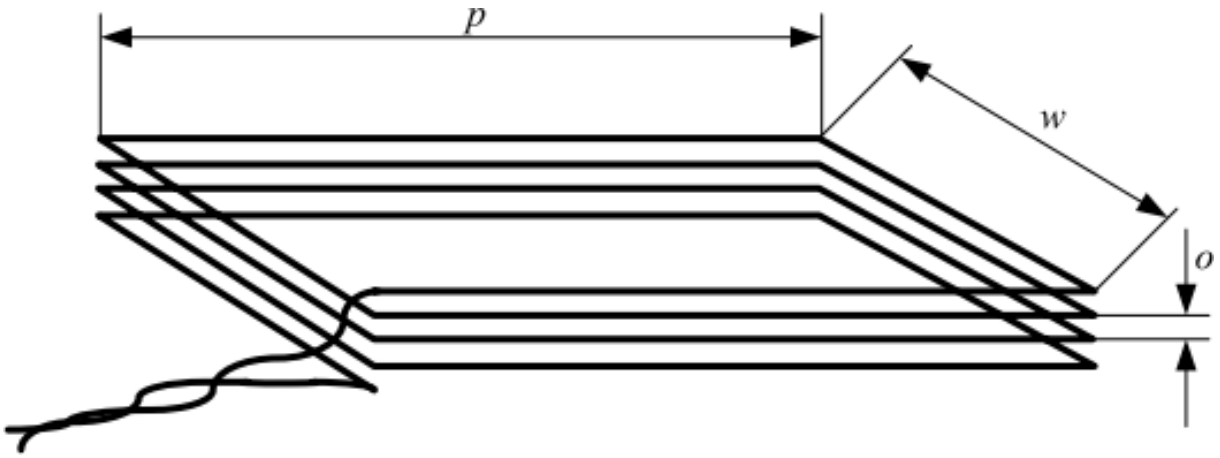
W rozdziale szóstym przedstawiono funkcje powstałego oprogramowania, w tym sposób instalacji i uruchamiania, omówienie interfejsu wejściowego i przykładowe aplikacje. Uzasadniono również możliwość dalszej rozbudowy programu na przykładzie graficznego interfejsu użytkownika do prezentacji wyników.

Rozdział siódmy zawiera analizę działania programu. Poruszono w nim zagadnienia stabilności i wydajności pracy. Przedstawiono również oprogramowanie, przy którego użyciu przeprowadzono testy poprawności działania algorytmu.

# 1. Czujniki indukcyjne pętlowe w pomiarach parametrów ruchu pojazdów

Pośród parametrów ruchu drogowego, które podlegają analizie, znajdują się wartości związane ze strumieniem pojazdów, takie jak natężenie ruchu, interwały czasowe pomiędzy kolejnymi pojazdami, czy płynność ruchu. Celem badań są również parametry pojazdu, do których należą prędkość, długość, masa pojazdu, liczba osi i ich położenie. Przyczyną obserwacji może być również klasyfikacja na podstawie standardów zdefiniowanych przez organizację *Federal Highway Administration* [19], wyszczególniającej trzynaście kategorii, lub innej, zaprojektowanej w celu klasyfikacji dedykowanego zagadnienia - na przykład masy pojazdu.

Wśród metod wykorzystywanych do pomiarów parametru ruchu znajdują się algorytmy przetwarzające dane z czujników indukcyjnych. Są one używane między innymi do badania prędkości, długości, liczby i rozkładu osi pojazdu. Budowę czujnika tego typu przedstawiono na rysunku 1.1.

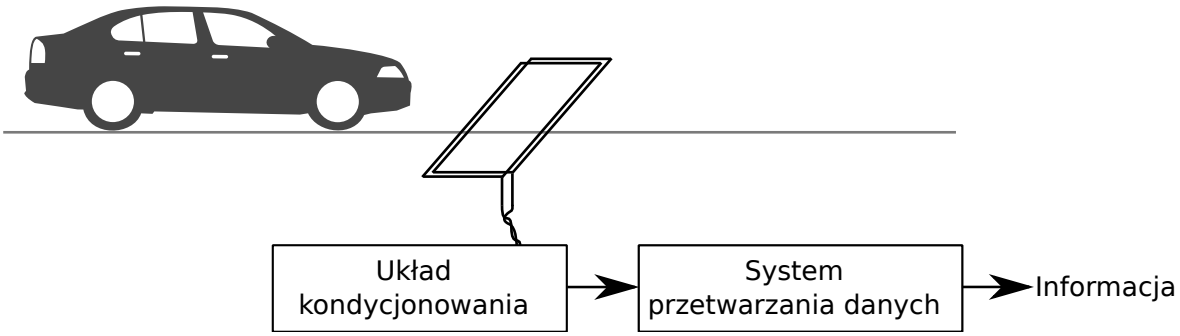



**Rysunek 1.1.** Schemat pętlowego czujnika indukcyjnego o wymiarach $w \times p$ w poziomie oraz odległości $o$ pomiędzy zwojami. Źródło grafiki: [10].

Czujnik indukcyjny pętlowy zbudowany jest z kilku zwojów izolowanego przewodu, który zazwyczaj umieszcza się pod nawierzchnią jezdni. Szerokość czujnika $p$ jest najczęściej co najmniej równa szerokości pasa ruchu, natomiast długość $w$ w typowych zagadnieniach, takich jak detekcja długości pojazdu czy położenia osi, nie przekracza jednego metra. Odległość $o$ pomiędzy zwojami wynika z grubości izolacji

sąsiadujących ze sobą przewodów. W praktyce wykorzystuje się różne kształty i wymiary czujników oraz stanowiska składające się z jednego lub wielu sensorów tego typu, zależnie od celów stawianych aplikacji. Wybór kształtu czujnika podyktowany jest jego zastosowaniem. Znajdują się wśród nich zbieranie informacji o parametrach przejeżdżających pojazdów, sterowanie sygnalizacją świetlną czy współpraca z czujnikami innego typu, na przykład w celu detekcji auta i wyzwalania w ten sposób pracy sensorów dokonujących dalsze pomiary.

Wykorzystanie czujników indukcyjnych pętlowych wymaga zbudowania aplikacji, w skład której wchodzą również układ kondycjonowania i system przetwarzania danych. Budowę stanowiska pomiarowego przedstawiono na rysunku 1.2.




**Rysunek 1.2.** Schemat systemu pomiarowego.

Układ kondycjonowania jest modułem elektronicznym odpowiedzialnym za pomiar stanu czujnika indukcyjnego. Stan ten można opisać poprzez wartości dwóch składowych impedancji czujnika. Uzyskuje się w ten sposób sygnały (profile) $R$ i $X$, proporcjonalne do wartości rezystancji zastępczej i reaktancji zastępczej czujnika indukcyjnego. Stosuje się również układy kondycjonowania pozwalające na uzyskanie jednego sygnału $M$, będącego pewną określoną funkcją (na przykład pierwiastkiem z sumy kwadratów) profili $R$ i $X$. Metoda ta uniemożliwia jednak uzyskanie wartości obu sygnałów na dalszym etapie przetwarzania, co znacznie ogranicza możliwe zastosowania układów kondycjonowania tego typu. Metody projektowania tych układów opisane zostały w pracach [7, 8, 10].

Układ kondycjonowania komunikuje się z systemem przetwarzania danych. Jego rolę może pełnić układ *FPGA* (rodzaj programowalnego układu logicznego, ang. *field-programmable gate array*), układ mikroprocesorowy lub komputer PC. Do zadań systemu przetwarzania należeć może analiza danych w celu otrzymania wyników, zapis pomiarów do bazy danych lub przesłanie uzyskanych informacji przez sieć do innego stanowiska. Wykorzystanie komunikacji sieciowej pozwala na ciągłą obserwację pracy układu z oddalonego laboratorium. Możliwość wykorzystania stosunkowo tanich i niewymagających wiele energii układów mikroprocesorowych ułatwia natomiast montaż stanowiska pomiarowego w trudnych warunkach, na przykład w miejscu uniemożliwiającym wykorzystanie komunikacji sieciowej.

Zdaniem autora, liczba publikacji naukowych poruszających temat wykorzystania czujników indukcyjnych do badania parametrów ruchu drogowego jest wciąż niewielka. Jedną z organizacji pracujących nad tym zagadnieniem jest Katedra Metrologii i Elektroniki Akademii Górniczo-Hutniczej. Powstałe tam publikacje poruszają tematy budowy i właściwości czujników pętlowych [11], projektowania układów kondycjonowania [12] i zastosowań pętli indukcyjnych do pomiarów parametrów ruchu drogowego [6, 4, 5]. Powstała również praca podsumowująca dotychczas zdobyte informacje w formie podręcznika akademickiego [7]. W rozprawie doktorskiej [10] zaproponowano algorytm detekcji liczby osi pojazdu, którego implementacji podjęto się w niniejszej pracy, zwiększając jego możliwości o określanie położenia osi oraz przybliżonej długości pojazdu.

Do stawianych przed indukcyjnymi czujnikami pętlowymi zadań należy detekcja obecności pojazdu [2]. Może to znaleźć zastosowanie w systemach sterujących sygnalizacją świetlną czy zliczających pojazdy. Znane są metody detekcji prędkości pojazdu, korzystając z właściwości czujników indukcyjnych [14], co może być użyte do badania płynności ruchu. Informacja na ten temat jest jedną z funkcji współczesnych systemów nawigacji, które sugerują trasę prowadzącą przez najlepiej przejezdne połączenia. Czujniki tego typu są również wykorzystywane do klasyfikacji pojazdów [3, 15]. Dzięki temu możliwe staje się projektowanie systemów automatycznego pobierania opłat czy planowania rozwoju połączeń.

Czujniki indukcyjne wykorzystywane do badania parametrów ruchu drogowego są tanie i odporne na większość czynników atmosferycznych i kontakt fizyczny z podwoziem pojazdu dzięki umieszczeniu pod nawierzchnią jezdni. Montaż jest prosty, a budowa czujnika pozwala na wykorzystanie uzyskiwanych informacji w układzie mikroprocesorowym lub komputerze wyposażonym w kartę pomiarową. Uzyskane w ten sposób profile pozwalają na analizę szeregu parametrów opisujących pojazd, ale również charakterystyki przepływu pojazdów.

# 2. Cel i zakres realizacji projektu

Przed przystąpieniem do wykonania programu, przyjęte zostały założenia opisujące sposób implementacji i cel projektu. Określone na tym etapie wymagania pozwoliły na zaplanowanie pracy i zachowanie konsekwencji w trakcie jej realizacji. Podstawowym założeniem była decyzja o wykorzystaniu systemu operacyjnego Linux do uruchamiania zaprojektowanego oprogramowania. Kierowano się przy tym możliwościami oferowanymi przez ten system, które omówione zostały w sekcji 2.1. Ponadto, użytkownik tego systemu może liczyć na duże wsparcie społeczności programistów i administratorów, dzięki czemu nietrudno było znaleźć materiały opisujące poszczególne funkcje. Przygotowanie oprogramowania w języku C uzasadniono jego wydajnością oraz możliwością kontroli zasobów na niskim poziomie.

Poniżej przedstawiono pozostałe główne założenia projektu i ograniczenia przyjęte podczas jego realizacji, a także opis interfejsów wejścia i wyjścia systemu.

## 2.1. Przyjęte założenia

### Idea oprogramowania GNU

Podczas pracy inspirowano się programami powstającymi wokół ruchu *GNU* [32]. W ciągu kilkunastu lat publikacji oprogramowania związanego z tym systemem powstał zbiór zasad, zebranych w dokumencie *GNU Coding Standard*, których celem było „zachowanie przejrzystości i jednolitości systemu oraz umożliwienie pisania przenośnych, stabilnych i niezawodnych programów” [23]. Dokument ten opisuje takie zagadnienia jak styl formatowania kodu źródłowego i wykorzystanie komentarzy, obsługa flag, czy dokumentacja programu.

Podczas pisania kodu programu stosowano się do *Google C++ Style Guide* - szeregu reguł opisujących sposób formatowania kodu w takiej formie, aby zapewnić jego przejrzystość i czytelność [24]. Pozwoliło to na utrzymanie podobnego stylu dla całego kodu źródłowego. Dzięki temu wyszukiwanie błędów i rozwój programu był łatwiejszy.

## Podział na moduły

Program podzielono na mniejsze elementy - moduły. Każdy z nich jest odpowiedzialny za wyłącznie jedno zadanie. Kod źródłowy każdego składnika stał się dzięki temu bardziej zwięzły, a program przenośny.

Wyszczególniono trzy główne funkcje pakietu:

1. moduł algorytmu - odpowiedzialny za detekcję liczby osi i ich położeń oraz długości pojazdu,
2. moduł obserwatora - odpowiedzialny za śledzenie zmian w strukturze ścieżek systemowych w oczekiwaniu na nowe pliki, których zawartość jest przekazywana do modułu algorytmu,
3. moduł graficzny - umożliwiający przedstawienie wyników pracy pogramu przy użyciu interfejsu użytkownika.

## Potoki

Komunikacja pomiędzy modułami została zaprojektowana przy użyciu potoków dostępnych w systemie Linux. Jest to forma „przekierowania umożliwiającego przesłanie wyjścia jednego programu do dalszego przetworzenia w kolejnym" [37]. Dzięki wykorzystaniu tak zwanego operatora potoku (|) można uniknąć tworzenia plików tymczasowych, a forma wykonywanych poleceń staje się bardziej zwięzła. Z tego powodu, jest to jedna z najczęściej wykorzystywanych funkcji powłoki systemu Linux [1].

Przykład operacji potokowej przedstawiono na listingu 2.1.

**Listing 2.1:** Przykład użycia potoków. Operacja `find` jako wynik zwróci wszystkie pliki znajdujące się w katalogu *dokumenty* oraz jego podkatalogach. Korzystając z polecenia `grep`, na wyjściu uzyskane zostaną tylko pliki o rozszerzeniu doc. Przy użyciu polecenia `tar` zbudowane będzie archiwum o nazwie *archiwum.tar*. Argument „`-T -`" informuje program, by odczytał listę plików do archiwizacji ze strumienia wejściowego.

```
find dokumenty | grep '.doc' | tar cfz archiwum.tar -T -
```

## Obsługa flag

Projektowany program jest wywoływany z poziomu linii poleceń systemu Linux. Aby umożliwić konfigurację, zdecydowano się na zaimplementowanie obsługi flag programowych [22]. Jest to standardowa metoda wykorzystywana przez większość programów działających z poziomu linii poleceń. Korzystając z tej metody, cała konfiguracja może mieć miejsce przed uruchomieniem programu. Umożliwia to wyko-

rzystanie aplikacji w skryptach, których praca odbywa się bez ciągłego nadzoru użytkownika.

W skład zaprojektowanego interfejsu wchodzą flagi konfigurujące tryb pracy algorytmu, włączające funkcje detekcji położenia osi, wykrywania długości pojazdu i weryfikacji przy użyciu czujników piezoelektrycznych. Obsługiwane są również flagi konfigurujące ilość danych wyświetlanych na ekran oraz udostępniających informację o wersji programu i stronę pomocy, zwyczajowo dostępne w oprogramowaniu GNU. Listę obsługiwanych flag przedstawiono w rozdziale 6.3.

Konsolę systemową, służącą do obsługi środowiska systemowego i uruchamianych programów przedstawiono na rysunku 2.1.


**Rysunek 2.1.** Konsola systemowa z powłoką *bash* [1].

## Wysoka wydajność

Analiza danych uzyskanych z czujników indukcyjnych może wymagać przetworzenia informacji dla setek czy tysięcy pojazdów. W stosunku do programów działających na takiej ilości danych stawiane są wymogi związane z ich wydajnością. Wymagania te spełniono dzięki wykorzystaniu języka C do zaprojektowania modułu obliczeniowego. Pozwoliło to na zachowanie dużej kontroli nad takimi elementami jak operacje wejścia/wyjścia i zarządzanie pamięcią, co w konsekwencji umożliwiło optymalizację czasu pracy programu. Dokładna analiza tego zagadnienia przedstawiona została w rozdziale 7.3.

## Dwufunkcyjność

Program może być wywoływany na dwa sposoby.

Pierwszy umożliwia przeprowadzenie obliczeń dla jednego lub wielu plików zawierających dane pomiarowe, które są dostępne w chwili uruchomienia. Pozwala to na przeprowadzenie analizy zebranych wcześniej danych.

Program może również zostać wywołany w trybie oczekiwania na dane. W momencie pojawienia się nowego pliku jest on przekazywany do analizy. Umożliwia to pracę bezpośrednio w systemie zbierania danych. Praktyczne może być zaprojektowanie ciągu operacji, w którym dane pomiarowe trafiają do określonego folderu, są analizowane przez przygotowany program, a następnie prezentowane w czytelnej formie, na przykład przez stronę internetową. Umożliwia to nadzorowanie pracy systemu pomiarowego i zbieranie wyników bez konieczności fizycznej obecności na stanowisku.

Pozwalałoby to na nadzorowanie pracy czujnika i zbieranie wyników bez konieczności fizycznego kontaktu z urządzeniem.

### Wysoka stabilność pracy

Ze względu na możliwość użycia programu w trybie ciągłym i analizy dużego zbioru danych podczas jednego uruchomienia konieczne jest zagwarantowanie stabilności pracy w każdych warunkach. Aby zwiększyć czytelność i przenośność projektu, nie wykorzystano niezdefiniowanych zachowań języka C, których nie opisano w standardzie [33], a których implementacja w środowiskach kompilacji może być różna i podlegać zmianom w przyszłości. Rozwiązano również przyczyny wszystkich ostrzeżeń, które przekazywał kompilator języka C.

Temat ten został poruszony również w rozdziale 7.2.

## 2.2. Przyjęte ograniczenia

Przyjęte zostały następujące ograniczenia w pracę programu.

- Przyjęto, że jeden pomiar, identyfikowany na podstawie wektora czasu, zawiera dane dotyczące jednego przejazdu.
- Program współpracuje ze stanowiskiem pomiarowym wyposażonym w dwie pętle indukcyjne, a także dwa czujniki nacisku, które są wykorzystywane do dokładnego określenia prędkości pojazdu.
- Algorytm wykrywa poprawnie auta o liczbie osi od dwóch do pięciu. Wynik powyżej pięciu osi uważany jest za błąd. Pojazdy o takiej liczbie osi stanowią niewielki podzbiór ruchu samochodowego, a przyjęte założenie pozwala na analizę poprawności pracy algorytmu. Wartość ta również może być zmodyfikowana w kodzie źródłowym.

# 2.3. Interfejsy wejścia i wyjścia programu

## Interfejs wejścia

Przed uruchomieniem programu konieczne jest przygotowanie danych z czujników pętlowych, zawierających informacje dotyczące jednego pojazdu. Projekt zorganizowano w taki sposób, by zintegrować go z systemem odpowiedzialnym za rejestrację profili $R$ i $X$.

Algorytm wymaga informacji pochodzących z dwóch pętli indukcyjnych, węższej - wykorzystywanej do detekcji liczby osi oraz ich położenia, oraz szerszej - służącej do estymacji długości pojazdu. Ponadto wykorzystywane są dwa czujniki nacisku umożliwiające wyznaczenie prędkości pojazdu.

Przyjęto założenie, że system ten dostarcza dane w następujący sposób.

- Dane dostępne są w formie tekstowej poprzez strumień wejściowy lub plik.
- Każdy wiersz zawiera dane dla jednej chwili czasowej w formacie siedmiu liczb. Pierwsza liczba oznacza czas pomiaru w sekundach, a pozostałe opisują wartości uzyskane z czujników pętlowych i piezoelektrycznych.
- Symbolem dziesiętnym jest znak kropki „.”, a liczba ujemna opisywana jest przez znak myślnika „-” bez odstępu, na przykład: $-2.4321$.
- Dane są uporządkowane względem czasu. Pozwala to na odseparowanie danych pomiarowych dotyczących wielu pojazdów w przypadku wykorzystania strumienia wejściowego.

Przykładowy fragment pliku spełniającego te założenia przedstawiono na listingu 2.2.

**Listing 2.2:** Fragment pliku danych wejściowych

```
0.023  0.16   −0.08  0.12   −0.06  0.07   −0.01
0.024  0.18   −0.06  0.12   −0.08  0.10    0.02
```

Przyjęto następującą kolejność danych:

- czas w sekundach,
- pętla szeroka (sygnały $X$ i $R$),
- pętla wąska (sygnały $X$ i $R$),
- czujnik nacisku pierwszy,
- czujnik nacisku drugi.

## Interfejs wyjścia

Program wykonuje algorytm, korzystając z załadowanych danych wejściowych i wyświetla wynik pracy poprzez standardowy strumień wyjściowy.

W podstawowej konfiguracji wykonywana jest wyłącznie procedura detekcji liczby osi. Wyście programu ma wtedy postać przedstawioną na listingu

Przyjęto również konwencję formatu wyjścia programu.

W przypadku korzystania z podstawowej funkcji algorytmu, a więc wykrywania liczby osi pojazdu, przyjmuje postać przedstawioną na listingu 2.3.

**Listing 2.3:** Wynik działania programu w podstawowej konfiguracji. `źródło` to nazwa pliku wejściowego lub ciąg `stdin` w przypadku wykorzystania strumienia wejściowego. n to liczba lub ciąg znaków będący wynikiem działania algorytmu detekcji liczby osi. Wystąpienie ciągu `5up` informuje o wykryciu pojazdu pięcioosiowego jadącego z podniesioną osią, natomiast ciąg `error` informuje o błędzie działania algorytmu.

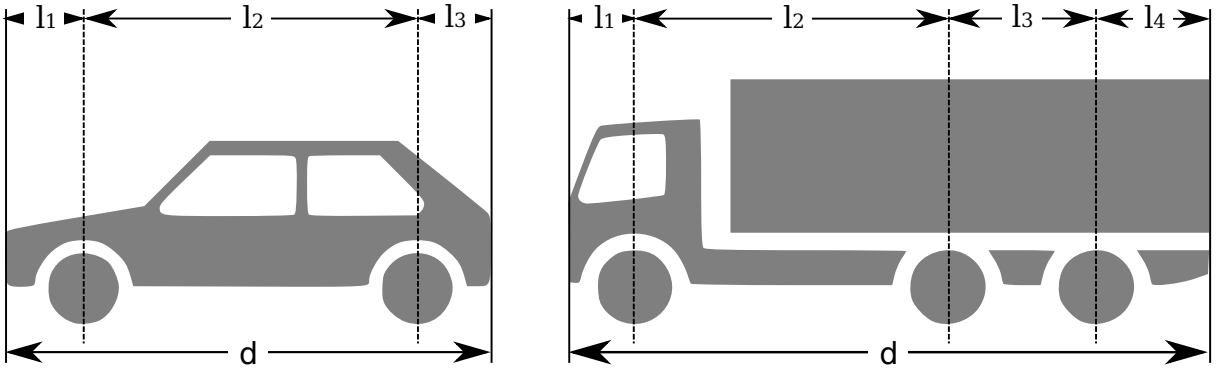
```
źródło n
```

Format wyjścia zmienia się wraz z aktywowaniem kolejnych funkcji programu.

Procedura wyznaczania położeń osi i długości pojazdu wyświetla na wyjściu dodatkowy ciąg, przedstawiony na listingu 2.4.

**Listing 2.4:** Wynik działania programu po włączeniu funkcji wyznaczania położeń osi i długości pojazdu. Początek linii ma formę zgodną z listingiem 2.3, natomiast kolejne wartości opisują odległości wyrażone w metrach, których znaczenie przedstawiono na rysunku 2.2.

```
źródło n d l_1 l_2 l_3 ... l_n+1
```




**Rysunek 2.2.** Schemat odległości wewnątrz pojazdu osobowego oraz trójosiowego pojazdu ciężarowego.

W przypadku weryfikacji wyników algorytmu przy pomocy czujników piezoelektrycznych, na wyjściu pojawia się druga linia. Jej znaczenie opisano na listingu 2.5.

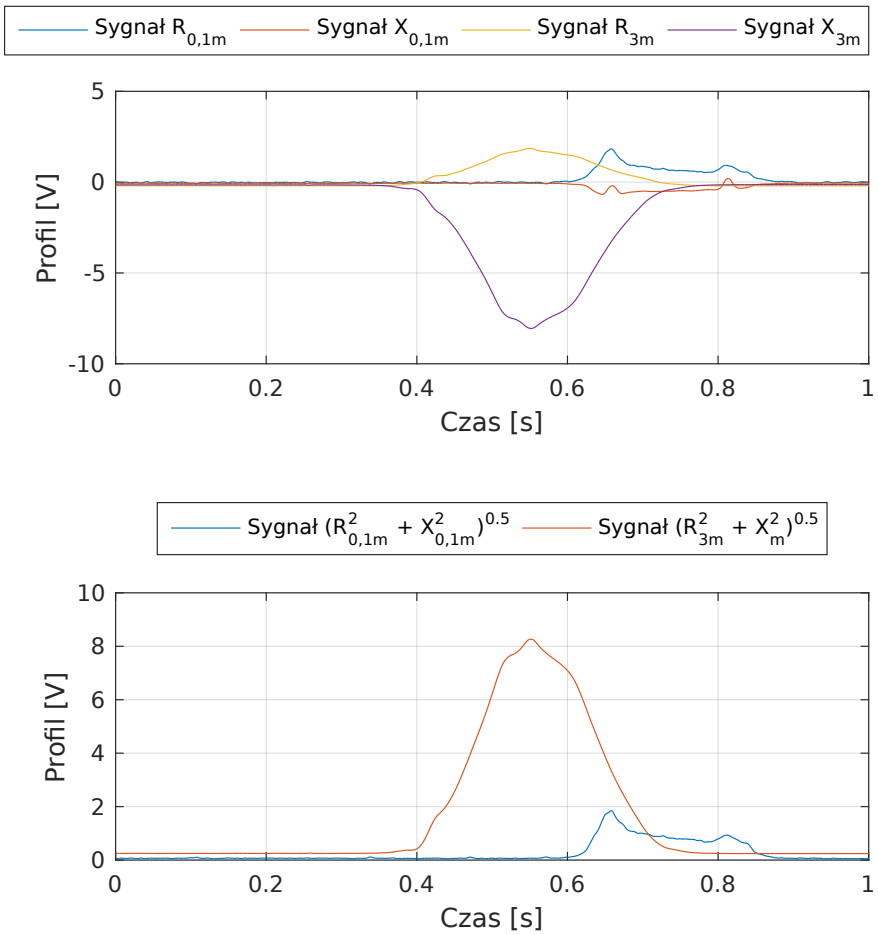
**Listing 2.5:** Wynik działania programu przy włączonej weryfikacji wyników. Kolejne wartości opisują odległości zgodnie z rysunkiem 2.2.

```
źródło n d l_1 l_2 l_3 ... l_n+1
piezo N L_2 L_3 ... L_n
```

Możliwe jest również manipulowanie ilością informacji wyświetlanymi na wyjściu. Funkcja ta została opisana w rozdziale 6.3.

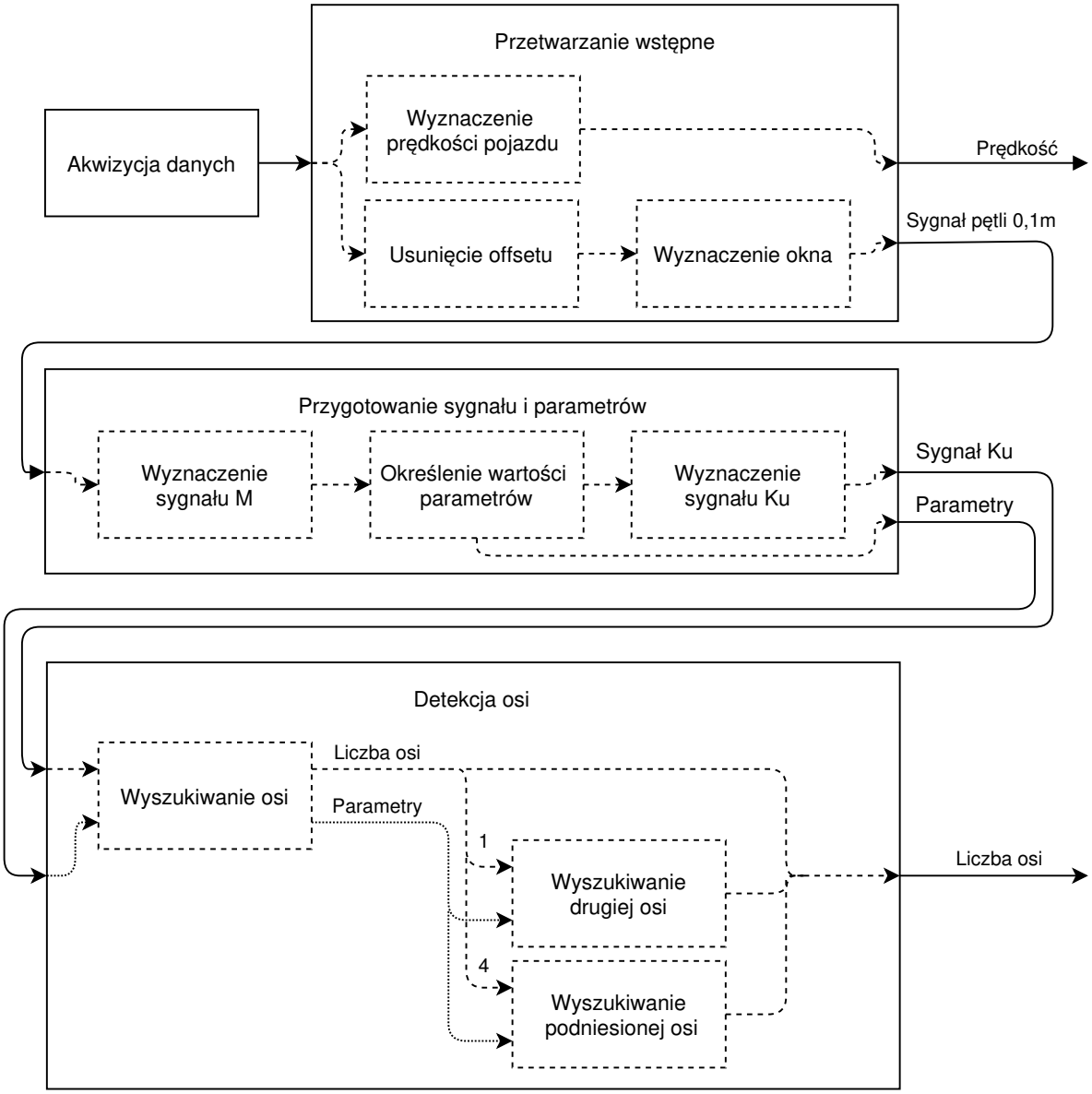
# 3. Algorytm detekcji liczby osi pojazdu

Dane pomiarowe z pętli indukcyjnych nie umożliwiają detekcji liczby osi pojazdu bez dalszej analizy. Na rysunku 3.1 przedstawiono wartości sygnałów $R$ oraz $X$ dla pętli o długościach $10cm$ oraz $3m$, oraz, pomocniczo, sygnały $M_i = \sqrt{R_i^2 + X_i^2}$ każdej pary. Wykres 3.1 i kolejne prezentują dane uzyskane w trakcie przejazdu pojazdu osobowego.




**Rysunek 3.1.** Przebieg sygnałów $R$ i $X$ dla pętli o długościach $10cm$ oraz $3m$.

Zaproponowano algorytm umożliwiający detekcję liczby osi pojazdu poprzez analizę sygnałów przedstawionych na rysunku 3.1. Uproszczony schemat tego algorytmu zaprezentowano na schemacie 3.2.




**Rysunek 3.2.** Schemat blokowy algorytmu detekcji liczby osi pojazdu.

W kolejnych sekcjach opisano etapy pracy algorytmu wraz z wynikami pracy dla danych wejściowych zaprezentowanych na rysunku 3.1.

## Przetwarzanie wstępne

Przed przystąpieniem do analizy danych, konieczne jest wstępne ich przetworzenie w taki sposób, by ze zbioru wejściowego odizolować wyłącznie użyteczne informacje.

Analizując przebiegi sygnałów przedstawione na rysunku 3.1 można zaobserwować przesunięcie w czasie dla okien, w których sygnały uzyskują niezerowe warto-

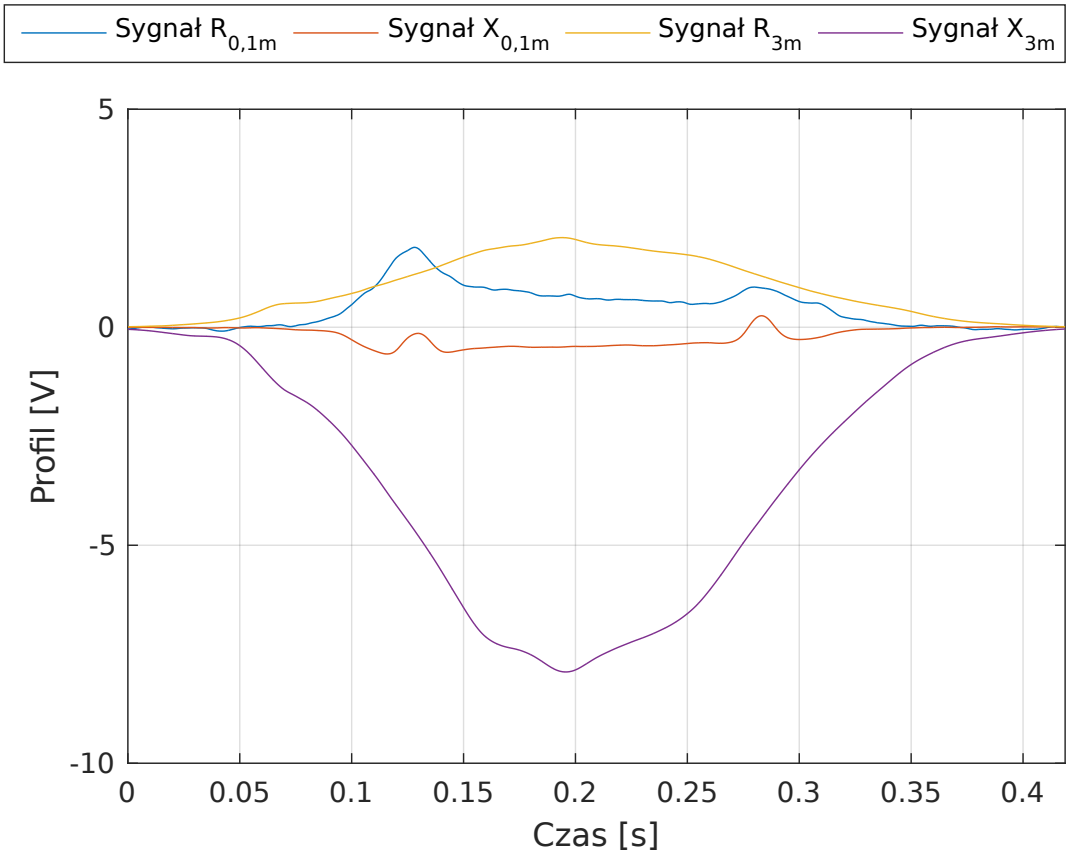
ści. Wynika to z budowy stanowiska pomiarowego, w którym kilka pętli indukcyjnych umieszczone jest kolejno wzdłuż jezdni. Z tego powodu w odczytach pojawia się przesunięcie, które należy usunąć w początkowej fazie wstępnego przetwarzania danych.

Na podstawie pierwszych trzydziestu wartości dla każdego sygnału wyznaczane jest jego średnie przesunięcie względem wartości zerowej. Przyjęto, że w pierwsze pomiary powinny być w idealnym środowisku równe zero, ponieważ pojazd nie znajduje się wtedy w zasięgu pracy czujnika indukcyjnego. Dzięki temu możliwe jest „wyzerowanie" sygnału, a więc odjęcie od wszystkich wartości wyznaczonego przesunięcia.

Przejeżdżający pojazd pozostaje w obszarze aktywnym pętli indukcyjnej (to znaczy w obszarze, w którym wartości sygnałów pętli przyjmują niezerowe wartości) nie dłużej niż kilka dziesiątych sekundy. Z tego powodu, dysponując danymi odczytanymi w ciągu kilku sekund, duża ich część nie jest użyteczna z punktu widzenia algorytmu. Zdecydowano się na usunięcie zbędnych wartości w celu uniknięcia niepotrzebnych obliczeń, a zatem zwiększenia wydajności algorytmu.

Na etapie wstępnego przetwarzania sygnałów wyznaczana jest również prędkość pojazdu. Wykorzystano do tego dwa czujniki piezoelektryczne, których odczyty znajdują się w zbiorze danych wejściowych. Czujniki te cechują się binarną wartością sygnału. Przy braku nacisku, odczyt wynosi $0V$, natomiast w przeciwnym przypadku $5V$. Dzięki znajomości odległości pomiędzy dwoma czujnikami oraz okresu próbkowania możliwe jest wyznaczenie prędkości poprzez wykrycie zbocza sygnalizującego nacisk wywołany przez pierwszą oś przejeżdżającego pojazdu. Uzyskana wartość nie jest wykorzystywana przez algorytm detekcji liczby osi, znalazła jednak zastosowanie w procedurze wyznaczania długości pojazdu, szerzej opisanym w rozdziale 4.

Na rysunku 3.3 przedstawiono sygnału po zakończeniu przetwarzania wstępnego.




**Rysunek 3.3.** Przebiegi sygnałów R i X po przetwarzaniu wstępnym.

# Przygotowanie sygnału i parametrów

Po zakończeniu przetwarzania wstępnego, możliwe jest przystąpienie do wyznaczenia sygnału, na podstawie którego przeprowadzona zostanie detekcja osi oraz parametrów $Y$ i $H$, decydujących o jej przebiegu.

Na podstawie profili $R_{0,1m}$ i $X_{0,1m}$ pętli o długości $10cm$ wyznaczany jest sygnał $M$, opisany wzorem 3.1.

$$M(R_{0,1m}, X_{0,1m}) = R_{0,1m}^2 + X_{0,1m}^2 \tag{3.1}$$

Na bazie profilu $X_{0,1m}$ oraz sygnału $M$ wyznacza się wartości $L_x$- liczby pomiarów, dla których wartość $X_{0,1m}$ przekracza $0,1V$ oraz $L_m$- liczby pomiarów dla których $M$ osiąga wartość większą od $0,5V$. Poprzez analizę stosunku $L_x$ do $L_m$ przyjmuje się optymalne wartości parametrów $a/b$, $Y$, $H$ zgodnie z tabelą 3.1.

| warunek | $\frac{L_x}{L_m} > 0,1$ | $\frac{L_x}{L_m} \leq 0,1$ |
|---|---|---|
| $a/b$ | 0,21 | 0,5 |
| $H$ | 0,45 | 4 |
| $Y$ | 0,8 | 0,5 |

**Tablica 3.1.** Dobór wartości parametrów.

Parametr $H$ określa szerokość pętli histerezy, natomiast parametr $Y$ - poziom odniesienia, względem którego powstaje pętla.

Kolejnym krokiem jest wyznaczenie sygnału $K_p$, opisanego wzorem 3.2.

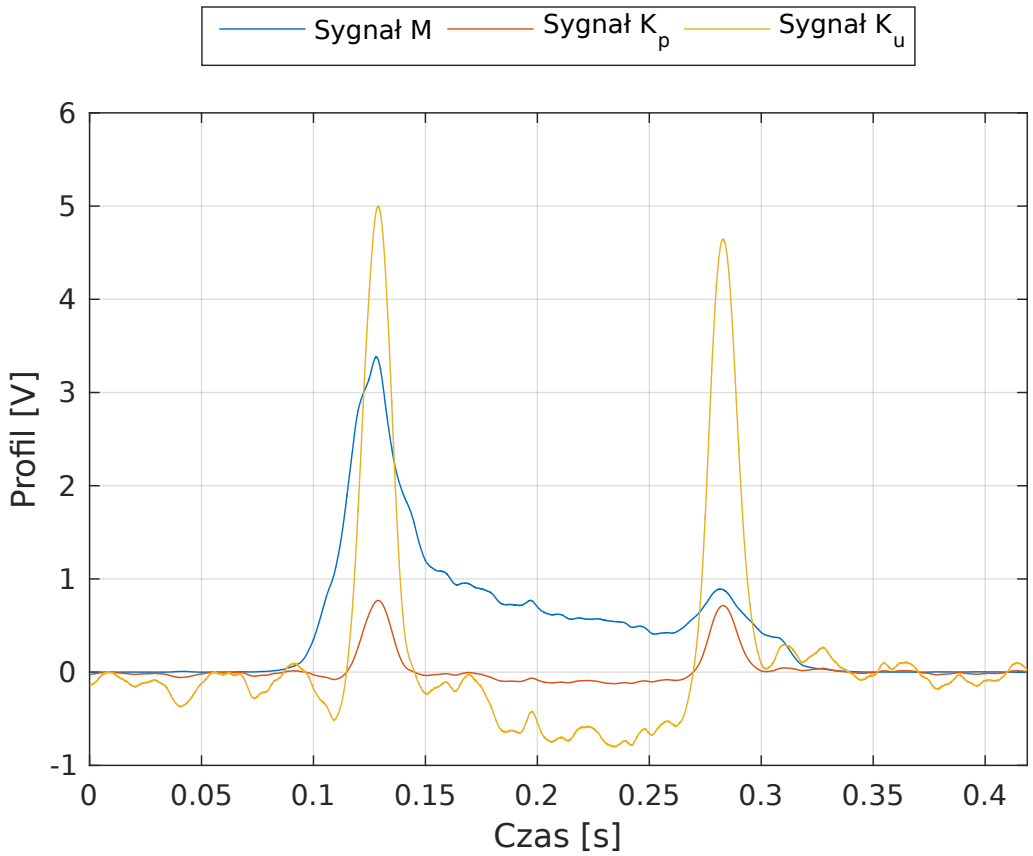
$$K_p = a/b \cdot R + X \tag{3.2}$$

Pomocniczo oblicza się stałą $K_{pmax} = max(K_p)$. Jeśli spełniony jest warunek $K_{pmax} > 3$, wartości parametrów ulegają zmianie na $Y = 0,8$, $H = 0,45$.

Sygnał $K_p$ poddawany jest normalizacji opisanej równaniem 3.3, w wyniku której uzyskuje się sygnał $K_u$ o wartościach nieprzekraczających $5V$.

$$K_u = \frac{K_p}{K_{pmax}} 5V \tag{3.3}$$

Sygnały $M$ oraz $K_p$ i $K_u$ przedstawiono na wykresie 3.4.




**Rysunek 3.4.** Sygnały $M$, $K_p$ i $K_u$. Dla powyższych sygnałów uzyskano wartości poszczególnych parametrów: $L_x = 91$, $L_m = 1697$, $K_{pmax} = 0,77$,$a/b = 0,5$, $H = 0,5$, $Y = 4$.

Sygnał $K_u$ oraz parametry $H$ i $Y$ stanowią dane wejściowe właściwej części algorytmu.

# Detekcja osi

Sygnał $K_u$, którego typowy przebieg przedstawiono na rysunku 3.4, pozwala zazwyczaj na detekcję liczby osi pojazdu poprzez wyznaczenie ilości impulsów sygnału.

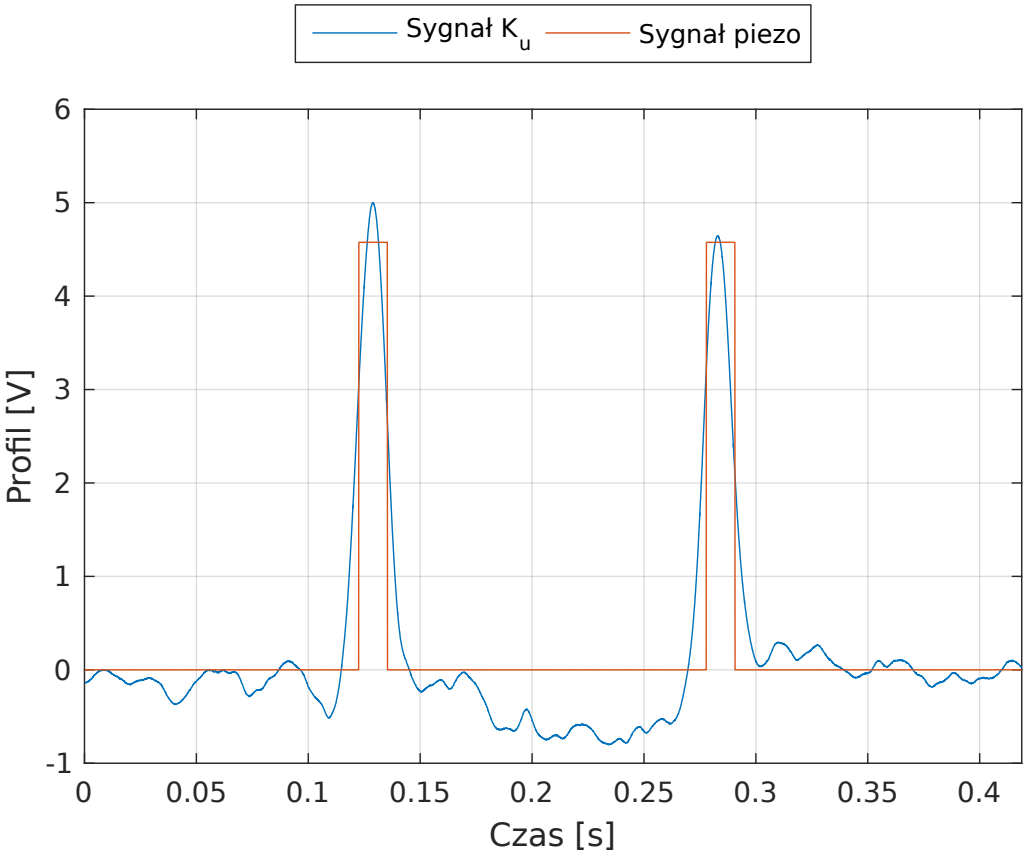
W tym celu zastosowano licznik z histerezą, opisaną wyznaczonymi wcześniej parametrami $H$ i $Y$. Uproszczony sposób implementacji licznika został przedstawiony na listingu 3.1.

**Listing 3.1:** Funkcja licznika osi.

```
unsigned counter(double *Ku, unsigned len, double Y, double H)
{
    unsigned num_axles = 0;
    unsigned is_high = 0; // flaga stanu
    const double level_top = Y + H / 2;
    const double level_bottom = Y - H / 2;

    for (unsigned i = 0; i < len; i++) {
        if (is_high == 0 && Ku[i] >= level_top) {
            is_high = 1;
            num_axles++;
        }
        else if (is_high == 1 && Ku[i] < level_bottom) {
            is_high = 0;
        }
    }
    return num_axles;
}
```

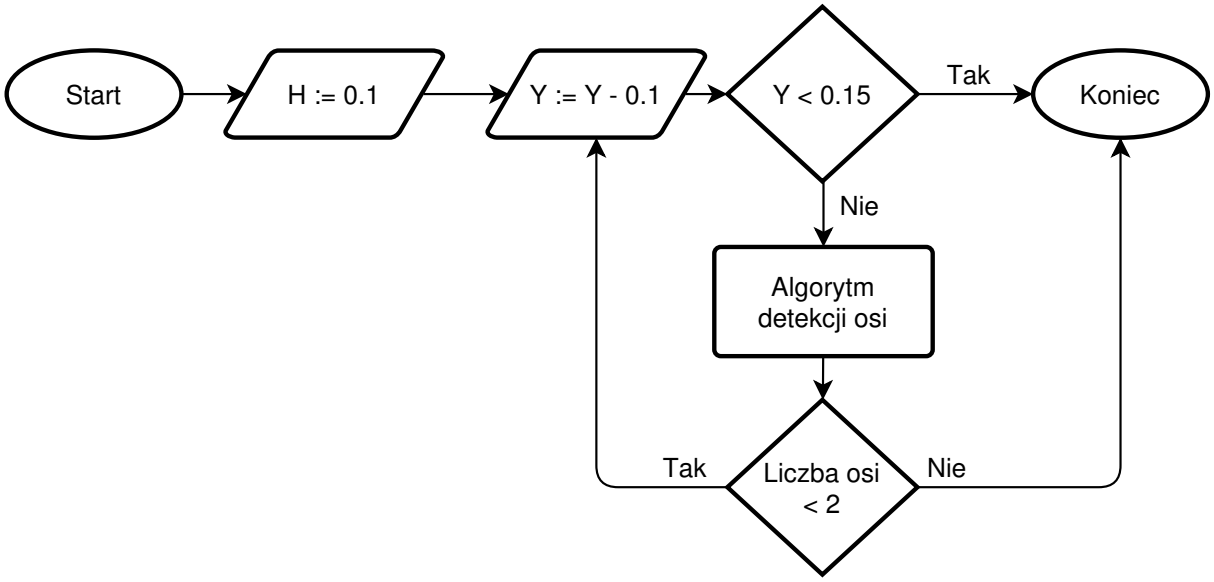
W wyniku wywołania funkcji dla sygnału $K_u$ przedstawionego na rysunku 3.4, uzyskano informację o wykryciu dwóch osi. Na wykresie 3.5 zaprezentowano sygnał $K_u$ i sygnał z czujnika piezoelektrycznego, potwierdzający poprawność wyniku. Algorytm może zakończyć działanie, nie odnajdując żadnej osi lub wykrywając tylko jedną oś. Taki wynik traktowany jest jako błędny i w celu rozwiązania błędu wykorzystano metodę detekcji brakujących osi.




**Rysunek 3.5.** Sygnał $K_u$ oraz sygnał czujnika piezoelektrycznego.

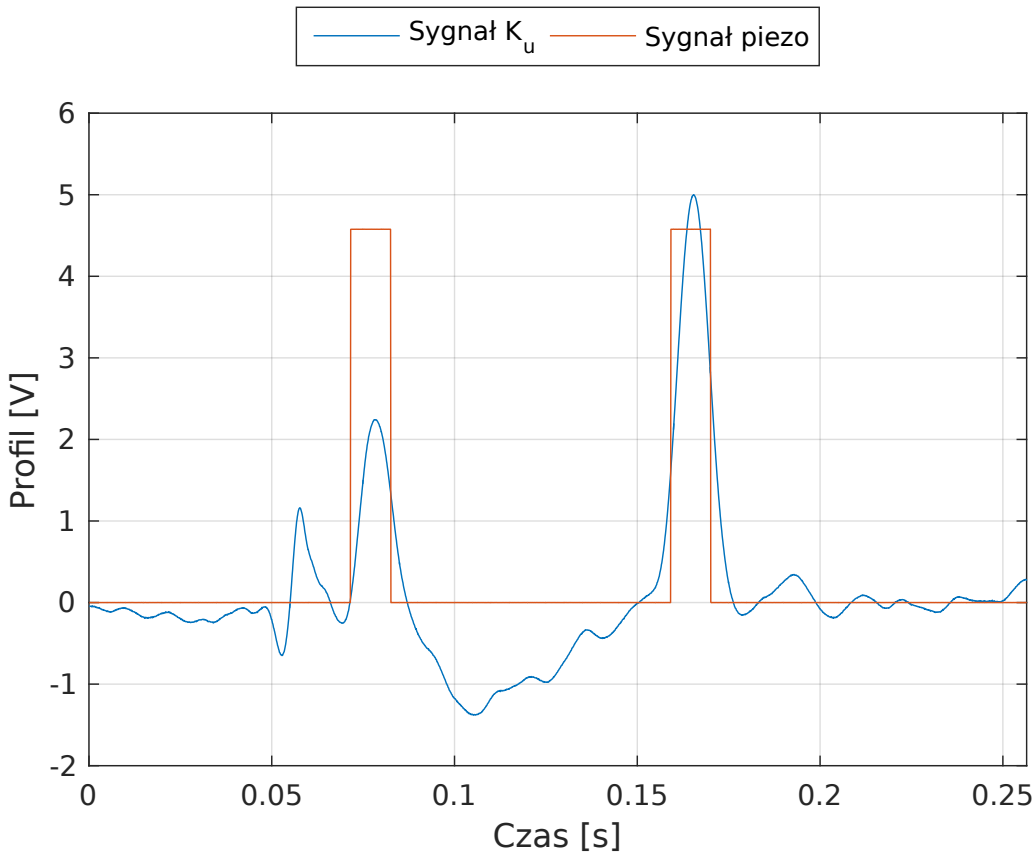
## Detekcja brakujących osi

Autor przyjął, że wykrycie minimum dwóch osi jest wymagane, by uznać wynik pracy algorytmu za poprawny. W przypadku wykrycia zera lub jednej osi, parametry ulegają zmianie, po czym próba detekcji jest ponawiana. Metoda jest powtarzana do czasu uzyskania poprawnego wyniku lub osiągnięcia wartości parametrów niepozwalających na dalszą analizę. Schemat opisujący algorytm przedstawiono na rysunku 3.6.




**Rysunek 3.6.** Algorytm detekcji brakujących osi.

Wykorzystanie algorytmu detekcji brakujących osi pozwala na zwiększenie skuteczności programu w poszukiwaniu osi na podstawie danych zawierających większe zakłócenia i mniej dokładne odczyty. Przykład działania algorytmu zaprezentowano na wykresie 3.7.




**Rysunek 3.7.** Sygnał $K_u$ oraz sygnał czujnika piezoelektrycznego.

Badając przebieg sygnału $K_u$ można zauważyć zwiększone wartości wielkości impulsów zakłóceń w stosunku do impulsów sygnalizujących istnienie osi. Może to doprowadzić do sytuacji, kiedy wykryta byłaby nieistniejąca oś, jednak nie zaobserwowano takiego przypadku podczas testowania algorytmu.

## Detekcja podniesionej osi

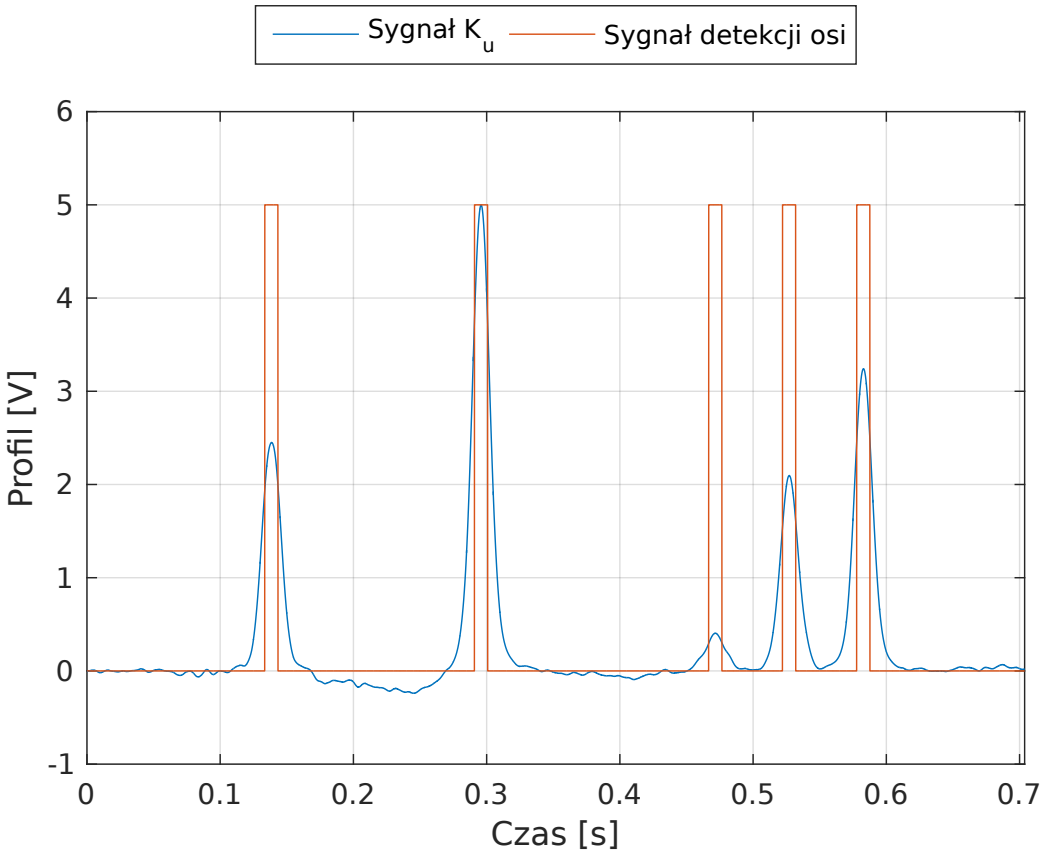
Interesujące zagadnienie z punktu widzenia detekcji liczby osi stanowią pojazdy ciężarowe pięcioosiowe, które mają możliwość uniesienia jednej z osi. W ich przypadku zawodna staje się metoda detekcji osi przy użyciu sensorów naciskowych, natomiast czujniki indukcyjne wciąż pozwalają na przeprowadzenie próby detekcji.

Zaprojektowano procedurę umożliwiającą wykrycie osi podniesionej, która uruchamiana jest w przypadku wykrycia pojazdu czteroosiowego.

Przed przystąpieniem do wykonania algorytmu przyjmuje się nowe wartości parametrów: $a/b = 0,3$ , $H = 0,1$. Następnie wyznaczany jest nowy sygnał $K_p$ zgodnie z formułą 3.2 oraz wykonywane są kolejne kroki algorytmu bazowego do uzyskania znormalizowanego sygnału $K_u$, opisanego równaniem 3.3.

Po wyznaczeniu wartości $K_u$ uruchamiany jest algorytm analogiczny do opisanego na schemacie 3.6, z uwzględnieniem nowego warunku stopu, którym jest wykrycie liczby osi różnej od czterech. W przypadku wykrycia pięciu osi wynik uznawany jest za poprawny. W przeciwnym razie, za właściwy uważany jest wcześniej uzyskany wynik czterech osi.

Przykład działania procedury i położenia wykrytych osi zaprezentowano na wykresie 3.8.




**Rysunek 3.8.** Sygnał $K_u$ oraz sygnał czujnika piezoelektrycznego.

Powyższe operacje stanowią kompletny algorytm poszukiwania osi pojazdu. W kolejnym rozdziale przedstawiono modyfikacje, które zwiększają jego funkcjonalność oraz umożliwiają weryfikację wyników.

# 4. Detekcja długości pojazdu i położenia osi w jego bryle. Weryfikacja poprawności działania algorytmu

Algorytm, który opisano w poprzednim rozdziale, został rozbudowany o dwie dodatkowe funkcje. Umożliwiono weryfikację wyników pracy algorytmu przez porównanie z wartościami uzyskiwanymi na podstawie pomiarów pochodzących z czujników piezoelektrycznych. Druga z zaprojektowanych procedur pozwala wyznaczyć położenie osi w bryle pojazdu oraz oszacować jego długość. Znajomość tych parametrów pozwala na klasyfikację, na przykład korzystając ze standardu opracowanego przez organizację *FHWA* [19]. Dzięki temu możliwe jest zaprojektowanie systemów odpowiedzialnych za automatyczny pobór opłat za przejazd lub zbieranie informacji o użytkownikach drogi.

## 4.1. Weryfikacja poprawności działania algorytmu przy użyciu sygnałów z czujników piezoelektrycznych

Stanowisko pomiarowe wyposażone jest w dwa czujniki piezoelektryczne, które wykorzystywane są do określania prędkości pojazdu. Czujniki te mogą również posłużyć do określenia liczby osi przejeżdżającego auta. Zostało to wykorzystane w programie w celu weryfikacji działania algorytmu.

Posłużono się sygnałem z jednego czujnika - zakładając, że oba działają poprawnie, ich odczyty powinny mieć przebieg różniący się wyłącznie czasem wystąpienia impulsów. Sygnał przetwarzany jest przez funkcję zaprezentowaną na listingu 3.1 z wartościami parametrów równymi $Y = 2$, $H = 0,1$. Sygnał z czujników piezoelektrycznych jest zbliżony do binarnej funkcji osiągającej wyłącznie wartości $0V$ (brak nacisku na czujnik) oraz $5V$ (po pojawieniu się nacisku). Wykorzystanie licznika impulsów z histerezą pozwala uzyskać wartość poprawną w większości przypadków. Wynikiem działania funkcji zliczającej jest liczba osi zarejestrowana w pomiarach przejazdu przez czujnik piezoelektryczny.

Wyznaczenie liczby osi stanowi koniec pierwszej fazy testu. W przypadku, gdy nie jest ona równa wartości uzyskanej przez algorytm detekcji osi, program kończy dzia-

łanie. W przeciwnym razie wyznaczany jest sygnał $CP$ będący funkcją sygnału z czujnika piezoelektrycznego $P_1$ oraz uzyskanego wcześniej sygnału $K_u$. Sposób obliczania wartości $CP$ pokazano na listingu 4.1.

**Listing 4.1:** Generowanie sygnału CP.

```
for (unsigned i = 0; i < length; i++) {
    if (is_high == 0 && Ku[i] >= level_top) {
        is_high = 5;
    }
    else if (is_high == 5 && Ku[i] < level_bottom) {
        is_high = 0;
    }
    CP[i] = is_high + P1[i];
}
```

Otrzymany sygnał $CP$ przetwarzany jest ponownie przez funkcję z listingu 3.1 z parametrami $Y = 8$, $H = 0$. Uzyskana wartość stanowi liczbę osi wyznaczoną przez algorytm detekcji poprawności wyniku. Informacja podsumowująca pracę algorytmu przekazywana jest użytkownikowi programu, umożliwiając porównanie z wartością uzyskaną podczas działania algorytmu detekcji osi.

## 4.2. Algorytm detekcji przybliżonej długości pojazdu i położenia osi w jego bryle

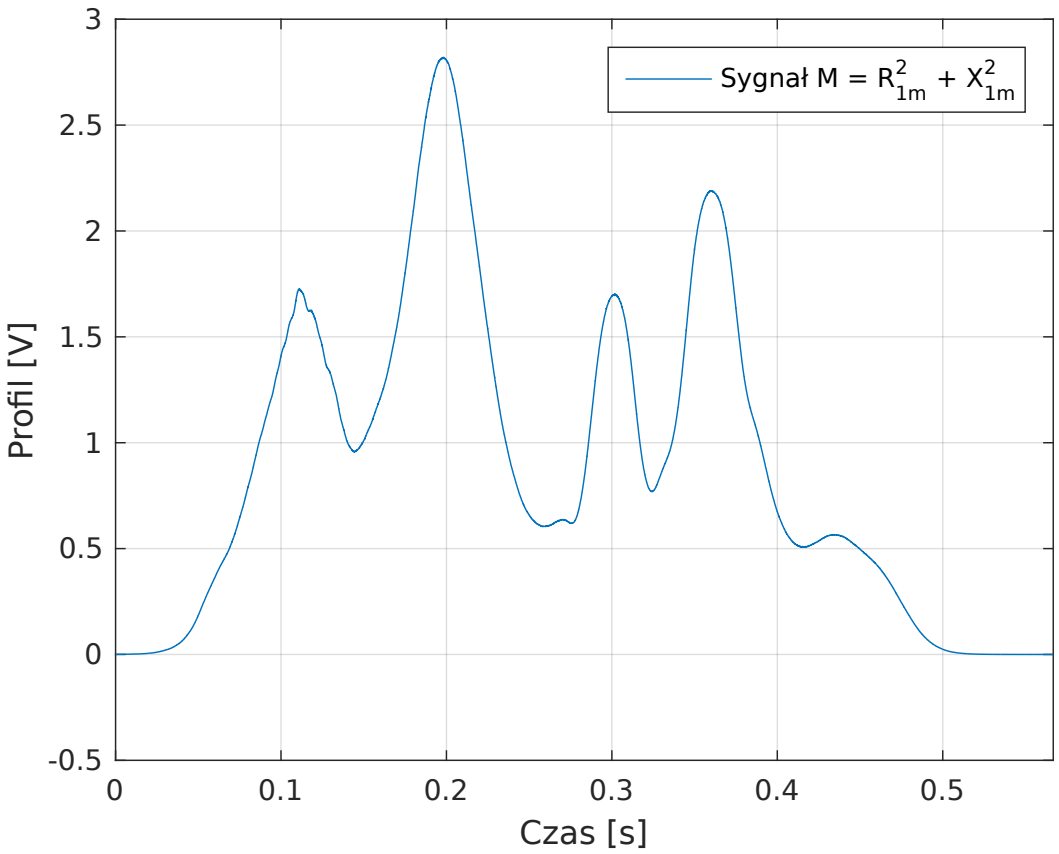
Informacja na temat liczby osi pojazdu nie pozwala określić dokładnie klasy pojazdu. Dwie osie posiadają zarówno małe auta osobowe, jak i pojazdy ciężarowe. Z tego powodu dane na temat osi można uzupełnić o długość auta oraz o rozstaw poszczególnych osi. Taki zbiór danych umożliwia dokładniejsze określenie typu pojazdu, dla którego badane są dane pomiarowe.

W celu wyznaczenia przybliżonej długości pojazdu zaimplementowano algorytm, który na bazie profili z czujników indukcyjnych określa przybliżoną długość pojazdu oraz położenie osi w jego bryle.

Pierwszym etapem jest wyznaczenie sygnału M opisanego zależnością 4.1. Wykorzystuje się w tym celu profile $R_1$ i $X_1$ opisujące zachowanie pętli indukcyjnej o długości jednego metra.

$$M(R_1, X_1) = R_1^2 + X_1^2 \tag{4.1}$$

W większości przypadków sygnał taki umożliwia badanie położenia pojazdu względem czujnika. Jego przebieg w czasie przedstawiono na wykresie 4.1.




**Rysunek 4.1.** Sygnał $M$ pojazdu ciężarowego trzyosiowego.

Zbadano krzywe sygnału $M$ dla różnych pojazdów, co pozwoliło opracować metodę wyznaczania przybliżonej długości. Na podstawie wartości sygnału $M$ wybierany jest ciągły przedział, w którym ssą one większe od progu odcięcia $K_l$. Przyjęto stałą $K_l = 0,5V$. Wyznaczony przedział jest opisany parą indeksów określających elementy brzegowe $[index_{start}; index_{end}]$.

Następnie obliczana jest długość okna zgodnie z równaniem 4.2

$$l = index_{end} - index_{start} \tag{4.2}$$

Na podstawie znajomości przyrostu czasu pomiędzy dwoma kolejnymi pomiarami $dt$ oraz prędkości pojazdu w metrach na sekundę $v$, można uzyskać długość pojazdu $d$ w metrach, zgodnie z równaniem 4.3. Prędkość pojazdu wyznaczana jest na etapie przetwarzania wstępnego algorytmu detekcji osi, co pokazano na schemacie 3.2.

$$d = l \cdot v \cdot dt \tag{4.3}$$

Wartość ta stanowi przybliżenie rzeczywistej długości pojazdu, co umożliwia przeprowadzenie wstępnej klasyfikacji.

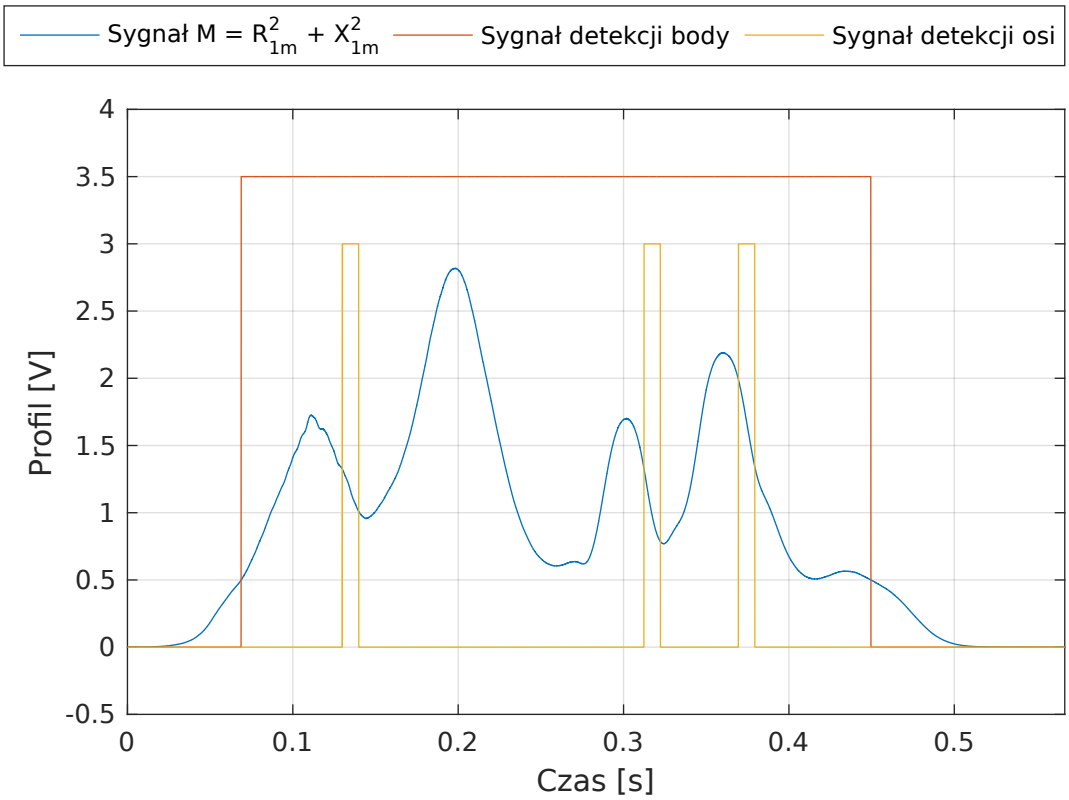
Położenia poszczególnych osi wyznaczane są we wcześniejszej części algorytmu, podczas wywołania procedury detekcji liczby osi pojazdu. Przyjęto , że środek osi znajduje się w punkcie, gdzie wartość sygnału $K_p$[3.2] osiąga lokalne maksimum. Znając indeks położenia osi $p_i$ oraz prędkość $v$ i różnicę czasu pomiędzy kolejnymi pomiarami $dt$, obliczona może być odległość osi od początku pojazdu, zgodnie z zależnością 4.4

$$P_i = p_i \cdot v \cdot dt \tag{4.4}$$

Wynik jest następnie weryfikowany - suma długości opisujących segmenty pojazdu, zgodnie z rysunkiem 2.2, nie powinna różnić się od wyznaczonej długości o więcej niż próg błędu $e$, co opisuje równanie 4.5. Przyjęto wartość $e = 0.02$.

$$|d - \sum_{i=0}^{liczba\,osi} l_i| \leq e \tag{4.5}$$

Wynik pracy algorytmu dla pojazdu ciężarowego trzyosiowego przedstawiono na wykresie 4.2. Sygnał detekcji nadwozia (ang. *body*) pozwala na estymację rzeczywistej długości auta. Pomocniczo zaprezentowano również sygnał detekcji osi, ułatwiający wyobrażenie rzeczywistej budowy pojazdu.




**Rysunek 4.2.** Wynik pracy algorytmu dla pojazdu ciężarowego trzyosiowego.

Na wykresie zaobserwować można brak zależności pomiędzy wartościami sygnału $M$ a położeniami kolejnych osi, co uzasadnia konieczność użycia profili po-

chodzących z pętli indukcyjnej o różnych parametrach do badania tych dwóch zagadnień.

Na żądanie użytkownika możliwa jest również weryfikacja poprawności wyznaczonych wartości położeń osi na podstawie odczytów z czujników piezoelektrycznych. Wymaga to wybrania właściwej instrukcji przy starcie programu, co zostało opisane w rozdziale 6.3.

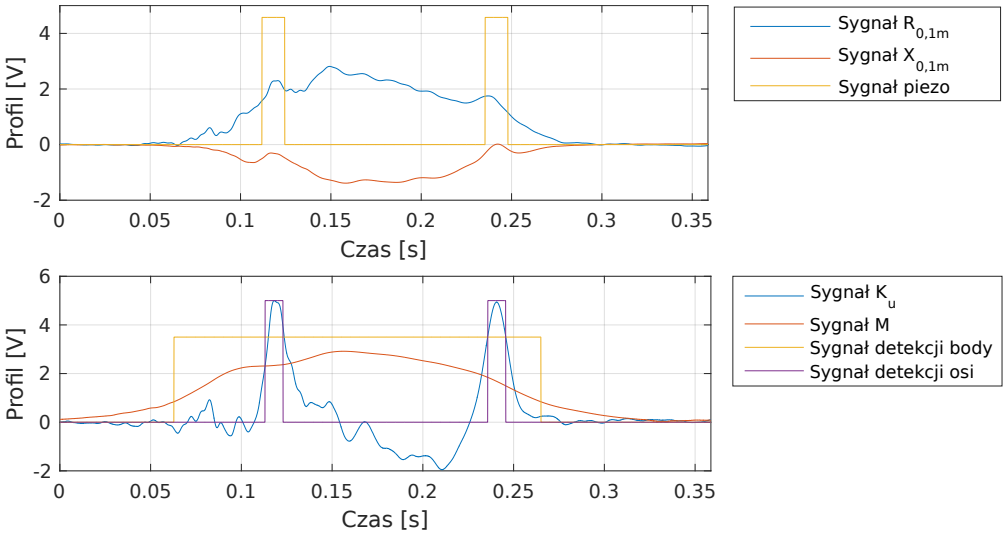
Procedura detekcji położenia osi w bryle pojazdu oferuje dokładność podobną do algorytmów wykorzystujących w tym celu detektorów nacisku. Wyniki nie różnią się zwykle o więcej niż kilka centymetrów. Wykorzystanie czujników indukcyjnych pozwala również na uniknięcie zjawiska drgań, które może wystąpić w przypadku sensorów piezoelektrycznych i powodować błędne wyniki. Ponadto, przewagą czujników indukcyjnych jest z pewnością możliwość wykrycia położenia osi podniesionej, co opisano w sekcji 3.

# 5. Analiza skuteczności algorytmu

Wyniki pracy algorytmu detekcji liczby osi oraz ich położenia i długości pojazdu silnie uzależniony jest od jakości danych wejściowych. Wystąpienie zakłóceń wpływa zwłaszcza na wynik szacowania długości auta. Ponadto, kształt uzyskiwanych profili czujnika pętlowego uzależniony jest od kategorii pojazdu - inne wartości uzyskiwane są dla aut osobowych, inne dla autobusów czy ciężarówek.

Zdecydowano się na przeprowadzenie analizy wyników algorytmu uzyskiwanych dla pojazdów różnych kategorii - aut osobowych, dostawczych oraz ciężarowych dwuosiowych i pięcioosiowych. Powinno to pozwolić na zbadanie wpływu wymienionych czynników na skuteczność programu.

Na wykresie 5.1 przedstawiono przykładowy wynik algorytmu dla pojazdu osobowego.

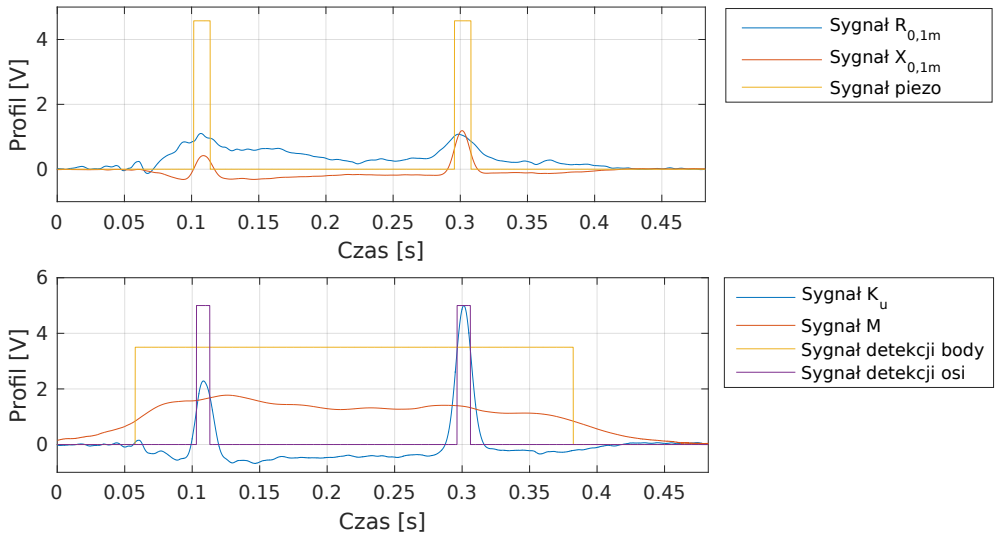



**Rysunek 5.1.** Wynik działania algorytmu dla pojazdu osobowego. Wykryto dwie osie, estymowana długość pojazdu wyniosła $d = 4.37m$, natomiast odległości pokazane na schemacie 2.2: $l_1 = 1,194m, l_2 = 2,653m, l_3 = 0,523m$. Weryfikacja przy użyciu czujnika piezoelektrycznego zwróciła wartość $lp_2 = 2,865m$. Pojazd poruszał się z prędkością $v = 21,62\frac{m}{s}$.

Wykres u góry przedstawia wartości profili $R$ i $X$ uzyskanych z czujnika indukcyjnego o długości $10cm$ oraz sygnału piezoelektrycznego, zsynchronizowane w czasie i ograniczone do okna przejazdu. Kolejny wykres zawiera dane opisujące sygnały wy-

korzystane przez algorytm detekcji liczby osi. Sygnał $K_u$, dany zależnością 3.3, wykorzystywany jest do wykrycia liczby osi, natomiast sygnał $M$, opisany formułą 4.1, pozwala wyznaczyć przybliżoną długość pojazdu. Sygnał $M$ został poddany przekształceniu (pierwiastek czwartego stopnia) w celu zwiększenia przejrzystości wykresów. Dołączono również sygnały, pokazujące przedziały, w których algorytm wykrył osie oraz nadwozie pojazdu.

Na wykresie 5.2 przedstawiono natomiast wynik działania programu dla auta dostawczego, przyjmując takie same oznaczenia.

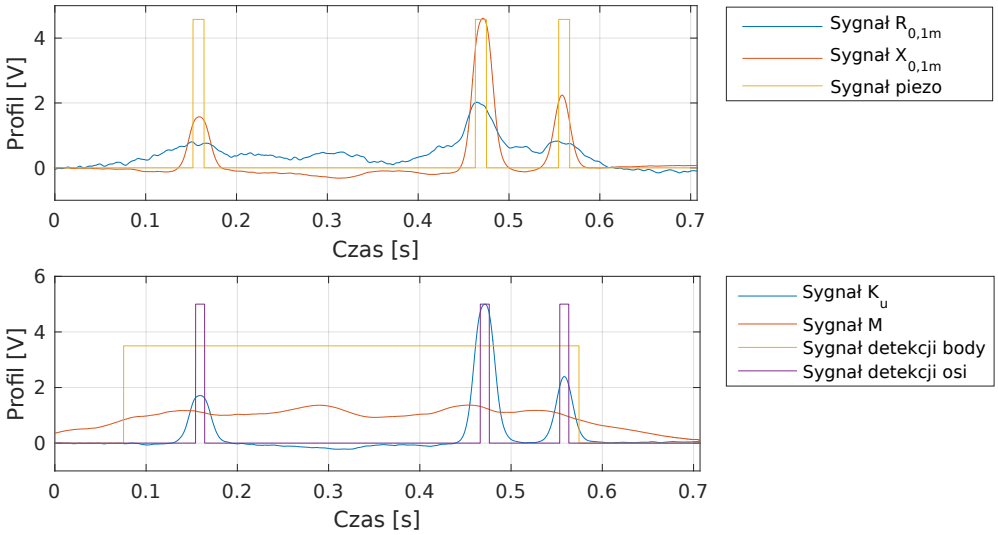



**Rysunek 5.2.** Wynik działania algorytmu dla pojazdu dostawczego. Wykryto dwie osie, wartości parametrów opisanych pod wykresem 5.1 wyniosły: $d = 6,803m, l_1 = 1,057m, l_2 = 4,048m, l_3 = 1,698m, lp_2 = 3,994m, v = 20,96\frac{m}{s}$.

Wykresy 5.1 oraz 5.2 prezentują typowe charakterystyki sygnałów czujnika indykcyjnego dla pojazdów osobowych oraz dostawczych. Wyznaczenie położenia osi poprzez bezpośrednią analizę tych sygnałów jest utrudnione. Dzięki procedurze wykonywanej przez algorytm, w wyniku której powstaje sygnał $K_u$, detekcja osi staje się zagadnieniem stosunkowo łatwym, polegającym na wykryciu i zliczeniu impulsów sygnału.

W obu zaprezentowanych powyżej przypadkach algorytm detekcji osi zwrócił poprawny wynik, co potwierdziła analiza sygnałów z czujników piezoelektrycznych. Wyznaczona odległość pomiędzy osiami również była poprawna, zachowując dokładność około dwudziestu centymetrów. Utrudniona natomiast okazała się poprawna detekcja długości pojazdu. Podczas projektowania i testowania algorytmu nie udało się uzyskać wartości parametrów gwarantujących dokładność większą niż kilkadziesiąt centymetrów.

Na wykresie 5.3 przedstawiono wynik pracy algorytmu dla pojazdu ciężarowego trzyosiowego.

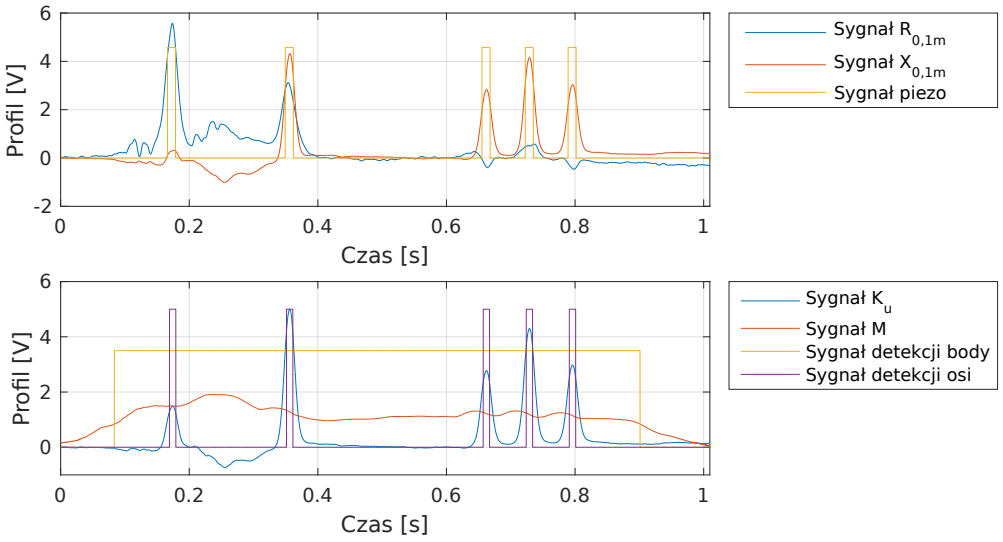



**Rysunek 5.3.** Wynik działania algorytmu dla pojazdu ciężarowego trzyosiowego. Wykryto trzy osie, wartości parametrów opisanych pod wykresem 5.1 wyniosły: $d = 7,797m, l_1 = 1,311m, l_2 = 4,875m, l_3 = 1,361m, l_4 = 0,250m, lp_2 = 4,937m, lp_3 = 1,350m, v = 15,62\frac{m}{s}$.

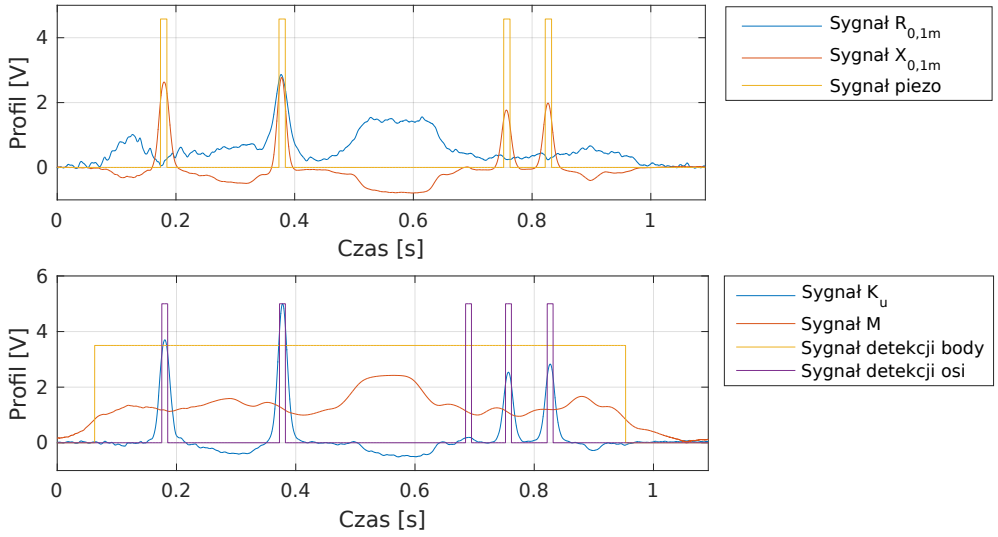
Na wykresie zaprezentowano typowe charakterystyki sygnałów dla pojazdów ciężarowych trzyosiowych. Widoczne są impulsy profilu $X_{0,1m}$, czego nie dało się zaobserwować w przypadku pojazdów osobowych i dostawczych. Procedura wyznaczania sygnału $K_u$ pozwala na uzyskanie właściwej liczby osi dla sygnałów o pokazanym przebiegu. Jest to istotne zwłaszcza dla poprawnego działania algorytmu detekcji położenia osi, gwarantując dokładność detekcji rzędu kilku centymetrów względem wyników uzyskiwanych przez analizę pomiarów z czujników piezoelektrycznych. Problemem okazała się poprawne oszacowanie długości pojazdu, zwłaszcza oszacowanie odległości pomiędzy ostatnią osią a końcem bryły. Uzasadniono to budową pojazdów ciężarowych i większą odległością podwozia od jezdni.

Ogranicza to dokładność czujników indukcyjnych, a w konsekwencji uniemożliwia detekcję długości pojazdu z dokładnością większą niż metr. Powoduje to uzyskanie fałszywego wyniku, sugerującego znacznie mniejszą od rzeczywistej odległość ostatniej osi od końca auta. Problem ten wystąpił dla około $30$ procent przeanalizowanych danych. Analogiczne wyniki uzyskano dla pojazdów ciężarowych dwu- i czteroosiowych.

Na wykresach 5.4 i 5.5 pokazano efekt pracy algorytmu dla pojazdu ciężarowego pięcioosiowego jadącego z opuszczoną oraz uniesioną jedną z osi.




**Rysunek 5.4.** Wynik działania algorytmu dla pojazdu ciężarowego pięciosiowego. Wykryto pięć osi, wartości parametrów opisanych pod wykresem 5.1 wyniosły: $d = 15,826m, l_1 = 1,756m, l_2 = 3,524m, l_3 = 5,923m, l_4 = 1,295m, l_5 = 1,297m, l_6 = 2,031m, lp_2 = 3,559m, lp_3 = 5,832m, lp_4 = 1,317m, lp_5 = 1,320m, v = 19,36\frac{m}{s}$.




**Rysunek 5.5.** Wynik działania algorytmu dla pojazdu ciężarowego pięcioosiowego jadącego z podniesioną osią. Wykryto pięć osi, wartości parametrów opisanych pod wykresem 5.1 wyniosły: $d = 16,566m, l_1 = 2,188m, l_2 = 3,673m, l_3 = 5,807m, l_4 = 1,247m, l_5 = 1,300m, l_6 = 2,352m, lp_2 = 3,630m, lp_3 = 7,051m, lp_5 = 1,310m, v = 18,60\frac{m}{s}$.

Wykresy prezentują typowe charakterystyki profili uzyskane dla pojazdów pięcioosiowych w dwóch konfiguracjach. Pojawieniu się osi w sąsiedztwie czujnika ponownie towarzyszy skok wartości sygnału $X$. Impuls wywołany przejazdem osi uniesionej jest jednak znacznie mniejszy, co utrudnia detekcję poprzez bezpośrednią analizę tego sygnału. Zastosowanie znajduje w tej sytuacji procedura detekcji osi uniesionej, pozwalająca z wysokim prawdopodobieństwem na wykrycie opisanego zjawiska. Al-

gorytm detekcji położenia osi gwarantuje dokładność rzędu kilku centymetrów. Ponownie zaobserwowano problem nieprecyzyjnej detekcji odległości ostatniej osi od końca pojazdu, występujący dla około $30 - 40\%$ pomiarów.

**Podsumowanie**

Interesująca jest możliwość próby detekcji liczby i położenia osi pojazdu wyłącznie na podstawie wartości profilu $X$ pochodzącego z pętli indukcyjnej o długości dziesięciu centymetrów. Sygnał ten zawiera impulsy towarzyszące osiom, widoczne zwłaszcza dla pojazdów dostawczych i ciężarowych. Wykluczono jednak możliwość praktycznego wykorzystania tej metody, uzasadniając to kilkoma argumentami. Korzystając z niej, utrudniona staje się detekcja położenia osi samochodów osobowych. Rozpoznanie osi podniesionej również uznane zostało za znacznie utrudnione. Konieczne byłoby zaprojektowanie niezależnych procedur detekcji osi pojazdów różnych kategorii, co wymusiłoby wstępną analizę sygnału i wydłużyło czas pracy algorytmu. Ponadto, wykorzystanie procedury badającej dwa profile gwarantuje większą dokładność wyniku, co przekłada się na uzyskane wartości opisujące wzajemne położenie osi.

O ile wyniki opisujące położenie osi uzyskiwane są z dokładnością kilku centymetrów, to detekcja długości pojazdu wymaga uwzględnienia większej wartości możliwego odchylenia od wartości długości rzeczywistej. Jest to widoczne w niewielkim stopniu dla aut osobowych i dostawczych, gdzie dokładność wyniosła kilkadziesiąt centymetrów, a znacznie bardziej dla pojazdów ciężarowych, gwarantując dokładność około jednego metra. Sytuacja ta może wynikać z ze specyfiki budowy samochodów ciężarowych, cechującej się dużą odległością podwodzia od jezdni i jego strukturą budowy. Niewykluczony jest również wpływ przewożonego ładunku lub jego braku na uzyskiwane wyniki.

Algorytm jest odporny na zakłócenia o stałej wartości - są one filtrowane na etapie przetwarzania wstępnego. Wyznaczana jest średnia wartość sygnału dla kilkudziesięciu pierwszych pomiarów i odjęcie wyznaczonego offsetu od sygnału.

Nie zaobserwowano również wpływu naturalnie występujących zakłóceń na wynik działania algorytmu detekcji liczby i położenia osi. Zbadano wpływ wprowadzenia zewnętrznego zakłócenia o rozkładzie Gaussa i wartości oczekiwanej równej zero oraz odchyleniu standardowym przekraczającym $0,1$. Zakłócenie miało marginalny wpływ na procedurę wyznaczania liczby osi i ich położenia, w przeciwieństwie do algorytmu szacowania długości pojazdu. Wyniki uzyskiwane były zawyżone o kilkanaście procent. Konieczne jest odfiltrowanie zakłóceń na etapie przetwarzania wstępnego, co pozwala na zwiększenie dokładności algorytmu.

Wykluczono również wpływ innych pojazdów na wartość pomiaru, zakładając, że minimalna odległość pomiędzy kolejnymi autami wynosi w normalnych warunkach co najmniej kilka metrów.

# 6. Opis wykonanego projektu

Komunikacja pomiędzy użytkownikiem a programem odbywa się dzięki rozbudowanemu interfejsowi, dostępnemu z linii poleceń. W kolejnych sekcjach przedstawiono zagadnienia instalacji, uruchamiania i sposobu użycia aplikacji. Przedstawiono również przykłady wykorzystania programu.

## 6.1. Wymagania i instalacja

W sekcji omówione zostały wymagania aplikacji wraz z uzasadnieniem. Następnie opisano zwięźle przebieg procesu instalacji oprogramowania. Omówiono również moduły będące składowymi projektu.

### Wymagania

Jedno z założeń przyjętych przed wykonaniem projektu definiowało środowisko, jakim powinien dysponować użytkownik. Założono, że program uruchamiany jest z poziomu systemu z rodziny *Unix* [36] bądź uniksopodobnego - na przykład *Linux* [35]. Przyjęto, że w systemie dostępna jest biblioteka języka C w standardzie *POSIX*, na przykład kompatybilna z tą normą biblioteka *GNU C* [20]. Opcjonalnie wymagane są również standardowa biblioteka języka *C++ GNU* [21] oraz biblioteka *Qt* w wersji 5.0 lub wyższej [31]. Umożliwia to skompilowanie modułu graficznego. Program wykorzystuje elementy języka C, których implementacja zależna jest od systemu - w przypadku *Linuxa* konieczne funkcje dostępne są od wersji 2.6.13 i znajdują się w bibliotece języka C.

W celu uproszczenia procesu budowana i instalacji posłużono się oprogramowaniem *cmake* [18] - program ten umożliwia przeprowadzenie procesu budowania i rozwiązywanie zależności niezależnie od systemu operacyjnego użytkownika, przenosząc odpowiedzialność na autora aplikacji. Przeprowadzono testy z wersjami 2.6 oraz 3.3 tej aplikacji i potwierdzono poprawność budowania wszystkich modułów projektu.

Program *cmake* umożliwia przeprowadzenie procesu budowania przy użyciu kompilatora zdefiniowanego przez użytkownika. Projektując program, duży nacisk położono na poprawność implementacji i pełną zgodność ze standardem języka.

Dzięki temu dowolny kompilator języka C kompatybilny ze standardem C11 umożliwia zbudowanie modułów [33] . Moduł graficzny może zostać skompilowany przy użyciu dowolnego kompilatora języka C++ wspierającego standard C++11 [34] . Moduł ten zgodny jest również ze standardem C++14, nie wykorzystuje przy tym jednak funkcji języka udostępnionych w tej normie. Poprawność przebiegu procesu budowania potwierdzono w trakcie testów z dwoma kompilatorami - *clang* w wersji 3.7.0 oraz *gcc* w wersjach 4.8.1 i 5.2.0 [26, 27] . Na podstawie logów, które są udostępniane przez oba kompilatory po zakończeniu pracy, nie stwierdzono błędów związanych z projektem aplikacji.

## Instalacja

Proces kompilacji powinien zostać wykonany z poziomu katalogu niebędącego główną ścieżką projektu. Jedna z najpopularniejszych metod kompilacji polega na utworzeniu podkatalogu *build* wewnątrz katalogu projektu i przejście do niego.

```
[user@pc projekt]$ mkdir build && cd build
```

Następnie, wykorzystując oprogramowanie *cmake,* wygenerować należy zbiór plików używanych w trakcie budowania aplikacji.

```
[user@pc build]$ cmake ..
```

Argumentem programu *cmake* jest ścieżka do głównego katalogu budowanego projektu. W przypadku kompilacji w niestandardowej lokacji konieczne jest podanie poprawnej ścieżki do katalogu zawierającego kod programu i plik *CMakeLists.txt*.

Domyślnie nie zostaną wygenerowane pliki potrzebne do zbudowania modułu graficznego. Można to zmienić, dodając do powyższego polecenia flagę:

```
[user@pc build]$ cmake .. –DCOMPILE_GUI=ON
```

Proces ten powinien zakończyć się powodzeniem, o ile spełnione zostały wszystkie wymagania związane ze środowiskiem, w którym budowany jest projekt. Kolejnym krokiem jest wygenerowanie plików binarnych dla każdego modułu, stosując poniższe polecenie.

```
[user@pc build]$ make
```

Instrukcja powinna zakończyć działanie nie zgłaszając żadnego błędu o ile spełnione zostały wymagania. Wszystkie użyteczne pliki powstałe na etapie kompilacji znajdują się wewnątrz podkatalogu */bin*.

Możliwe jest również zainstalowanie uzyskanych w ostatnim kroku plików w systemie, wykorzystując do tego poniższe polecenie. Należy jednocześnie pamiętać o konieczności posiadania uprawnień administratora, by komenda się powiodła.

```
[user@pc build]# make install
```

W przypadku powodzenia akcji moduły i ich podręczniki zostały zainstalowane i powinny być dostępne z poziomu terminala. W kolejnych sekcjach przyjęto założenie, że użytkownik zdecydował się na instalację. Pozwoliło to maksymalnie uprościć prezentowane polecenia.

### Podział na moduły

Projekt podzielony został na mniejsze moduły, co umożliwia wykorzystanie w sposób bardziej elastyczny. Możliwa jest współpraca zarówno w prostych aplikacjach, jak i budowanie złożonych łańcuchów wykonań.

W podstawowym module, nazwanym *axles,* znalazł się algorytm służący do wyznaczania liczby osi pojazdu, a także procedury określania położenia osi i długości pojazdu. Znalazły się w nim również funkcje odpowiedzialne za operacje wejścia-wyjścia. Kolejny moduł, nazwany *watcher*, odpowiada za komunikację z systemem użytkownika w celu śledzenia zmian w systemie plików. Dzięki temu możliwa jest detekcja pojawienia się nowych plików z danymi pomiarowymi i rozpoczęcie dalszej ich obróbki. Powstał również prosty interfejs graficzny, nazwany *axles_gui*, który umożliwia przedstawienie wyników działania algorytmu w formie bardziej przyjaznej użytkownikowi. Sposób jego kompilacji przedstawiony został w listingu 6.1. Udostępniono również dwa moduły pomocnicze odpowiedzialne za przystosowanie danych pomiarowych w dwóch znanych konfiguracjach do współpracy z algorytmem detekcji osi.

Moduły nie zostały ze sobą powiązane na poziomie implementacji, dzięki czemu możliwe było udostępnienie kilku odrębnych metod obsługi programu. Zostało to dokładniej opisane w rozdziale 6. Moduły dostosowano do pracy ze sobą na poziomie interfejsu w celu zapewnienia kompatybilności.

Projektowaniu interfejsu modułów programowych towarzyszyła inspiracja oprogramowaniem powstającym wokół ruchu *GNU,* co miało wpływ na zbudowanie programu w sposób sprzyjający dalszej rozbudowie. Przykładem tego jest interfejs graficzny, którego projektowanie nie było celem pracy, stanowi dowód na potwierdzenie tezy o możliwości rozwoju oprogramowania.

Przykładowe kierunki rozbudowy to aplikacja odpowiedzialna za komunikację sieciową z czujnikami i odbierająca dane - stanowiłaby pierwsze ogniwo łańcucha wywołań bądź program zbierający wyniki działania algorytmu i przechowujący je w bazie danych - byłoby to ostatnie ogniwo łańcucha.

## 6.2. Metody uruchamiania

Przyjęto założenie, że podstawową funkcją programu stanowić powinno wyznaczenie liczby osi i innych parametrów pojazdu na podstawie danych przechowywanych w pliku tekstowym. Oprogramowanie zostało zaprojektowane w taki sposób,

by domyślne działanie było zgodne z tym założeniem. Udostępniono również inne funkcje, których wywołanie jest możliwe poprzez przekazanie parametrów przy starcie programu, co zostało opisane w sekcji 6.3.

Wyszczególniono dwie podstawowe metody użytkowania programu. Pierwszą z nich jest uruchomienie modułu *axles* i przekazanie jako argumenty wiersza poleceń ścieżek do plików, które powinny zostać przeanalizowane. Program wczytuje następnie każdy plik, traktując go jako kompletny zbiór danych i kończy działanie po wykonaniu procedury algorytmu dla wszystkich plików.

Druga metoda polega na uruchomieniu aplikacji w trybie oczekiwania na dane. Ma to miejsce w sytuacji, gdy przy wywołaniu nie przekazano nazw plików poprzez argumenty wiersza poleceń. Program uruchamiany jest wtedy w trybie odczytu danych ze standardowego strumienia wejściowego (*stdin*) i w chwili pojawienia się nowego zbioru pomiarów uruchamiany jest algorytm detekcji, po którego działania zakończeniu program pozostaje w trybie oczekiwania na kolejne dane. Opisany tryb pracy jest wyjątkowo użyteczny w połączeniu z wykorzystaniem innych modułów, w tym zaprojektowanego, który obserwuje katalog w oczekiwaniu na zmiany, lub zaproponowany w rozdziale 6.1 - odpowiadający za komunikację sieciową z czujnikami.

Podczas pracy nad programem posłużono się ideą często wykorzystywaną w oprogramowaniu *GNU* - zdecydowano się możliwość przekazywania danych pomiędzy modułami przy użyciu strumieni. Pozwala to na budowanie łańcuchów wywołań, które umożliwiają obsługę zaawansowanych zadań. Moduł *axles* działający niezależnie pozwala wyłącznie na wywołanie algorytmu dla zadanych plików lub danych ze strumienia wejściowego. Po połączeniu pracy z modułem *watcher* możliwa jest się praca w trybie oczekiwania, bez konieczności uruchamiania programu na nowo dla każdego zestawu danych. Połączenie wyjścia modułu algorytmicznego z kolejnym pozwala na wyświetlanie informacji o pracy algorytmu przez interfejs graficzny.

Są to jedynie przykłady możliwych zastosowań oprogramowania- możliwe jest również wykorzystanie wbudowanych w system narzędzi lub zaprojektowanie kolejnych modułów, dedykowanych do konkretnych zadań.

## 6.3. Obsługiwane parametry

Interfejs programów obsługuje szereg flag modyfikujących ich pracę. Pozwala to na zdefiniowanie reguł działania programu na etapie uruchomienia i wykluczenie konieczności dalszej interakcji z użytkownikiem w trakcie wykonywania zadania. Poniżej przedstawiono flagi, których użycie może być przydatne dla użytkownika.

### Moduł *axles*

- `NAZWA_PLIKU_1 ... NAZWA_PLIKU_N`

Uruchamia działanie algorytmu dla każdego z podanych plików. W przypadku, gdy nie przekazano żadnego argumentu tego typu, następuje odczyt ze strumienia wejściowego *stdin*.

- `-c --verify`

  Wymusza uruchomienie procedury weryfikacji poprawności wyniku algorytmu przy użyciu czujników piezoelektrycznych.

- `-p --positions`

  Wymusza uruchomienie procedury określania położenia osi i przybliżonej długości pojazdu.

- `-f --config PLIK_KONFIGURACJI`

  Umożliwia załadowanie konfiguracji opisującej stanowisko pomiarowe, z którego pochodzą analizę danych.

- `-o --output NAZWA_PLIKU`

  Umożliwia zapisanie wyniku pracy algorytmu w pliku o podanej nazwie.

- `-q`

  Wymusza pracę w trybie cichym, bez wyświetlania żadnych komunikatów w trakcie pracy.

**Moduł *watcher***

- `NAZWA_KATALOGU_1 ... NAZWA_KATALOGU_N`

  Lista ścieżek, które są obserwowane przez program. W przypadku, gdy nie przekazano żadnego argumentu tego typu, obserwowana jest ścieżka, w której uruchomiony został program.

- `-e --extension ROZSZERZENIE`

  Wymusza śledzenie plików o zadanym rozszerzeniu. W przypadku, gdy nie użyto tego argumentu przy uruchomieniu, śledzone są pliki o rozszerzeniu *lvm*.

Oba moduły obsługują ponadto dwie wspólne flagi, `-h` - wyświetlającą skróconą pomoc, oraz `-v` - pokazującą informację o wersji programu.

## 6.4. Przykładowe aplikacje

Aby wyznaczyć liczbę osi pojazdu, dla którego dane pomiarowe znajdują się w pliku *PLIK.lvm*, należy posłużyć się poniższym poleceniem.

```
axles PLIK.lvm
```

Aby wykonać to zadanie dla wszystkich plików o rozszerzeniu *lvm* w katalogu *dane*, można wykorzystać jedno z dwóch poleceń.

```
axles dane/*.lvm
cat dane/*.lvm | axles
```

W drugim poleceniu pokazano ponadto metodę wykorzystania wbudowanych narzędzi systemowych w celu zwiększenia możliwości programu.

W celu wykrycia liczby osi, weryfikacji przy użyciu czujników piezoelektrycznych i określenia położenia osi oraz długości pojazdu, można wykorzystać poniższą komendę.

```
axles -pc PLIK.lvm
```

Na wyjściu pojawi się ciąg znaków podobny do poniższego. Wyjaśnienie znaczenia poszczególnych elementów wyjścia przedstawiono w rozdziale 2.3.

```
PLIK.lvm 2 12.064 3.295 5.859 2.909
piezo 2 5.954
```

Aby uruchomić program w trybie oczekiwania na dane przechowywane w plikach o rozszerzeniu *data* w katalogu *odczyty*, a wyniki pracy algorytmu zapisać do pliku *wyniki.txt* i nie wyświetlać ich bezpośrednio poprzez interfejs użytkownika, można posłużyć się łańcuchem wywołań zbudowanym przy użyciu poniższego polecenia.

```
watcher -e data odczyty | axles -o wyniki.txt
```

Następny przykład przedstawia możliwość rozbudowy powyższej komendy w taki sposób, by wyniki były prezentowane również poprzez interfejs graficzny.

**Listing 6.1:** Metoda uruchamiania interfejsu graficznego.

```
watcher -e data odczyty | axles -p -o wyniki.txt | axles_gui
```

Wykorzystanie programu we współpracy ze znanymi konfiguracjami stanowiska pomiarowego możliwe jest dzięki zastosowaniu plików konfiguracyjnych oraz narzędzi odpowiedzialnych za wstępne przetworzenie danych. Przygotowano dwa skrypty formatujące dane wejściowe (nazwane *swapper*) oraz pliki konfiguracyjne. Możliwe jest również przygotowanie własnym narzędzi odpowiedzialnych za formatowanie danych, na przykład przy użyciu języka *awk*, dostępnego w większości dystrybucji systemu Linux [25].

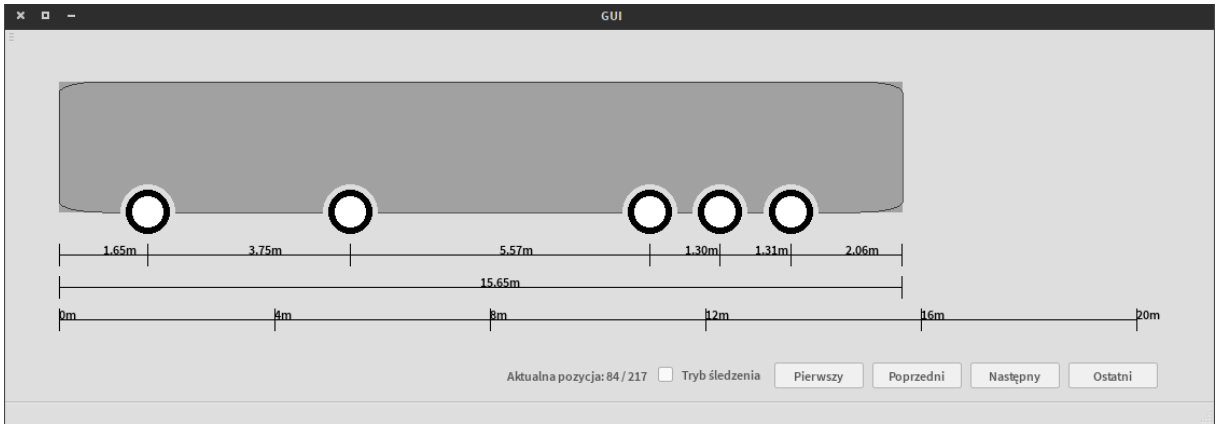
Poniżej przedstawiono przykładowe polecenie korzystające z omawianych funkcji.

```
cat *.data | ./swapper | axles -f konfiguracja.cfg -pc
```

## 6.5. Interfejs graficzny

Po ukończeniu podstawowych modułów aplikacji zdecydowano się na zaprojektowanie prostego interfejsu graficznego. Wykorzystanie interfejsu tego typu umożliwia prezentację wyświetlanych przez program w terminalu wyników w uproszczonej, czytelnej formie. Taka metoda przetwarzania otrzymanych danych może znaleźć zastosowanie w sytuacji, gdy wyniki pracy algorytmu przechowywane są w pliku, natomiast użytkownik powinien mieć dostęp do przeglądu danych. W ten sposób możliwa byłaby natychmiastowa weryfikacji poprawności działania systemu bez wykorzystania innych narzędzi. Sposób uruchamiania umożliwiający pracę w opisanej konfiguracji przedstawiono na listingu 6.1.

Okno interfejsu graficznego zaprezentowano na rysunku 6.1.




**Rysunek 6.1.** Okno interfejsu graficznego. W oknie wyświetlany jest wynik działania algorytmu dla pojazdu pięcioosiowego.

W interfejsie graficznym przedstawiana jest bryła przybliżonego wyglądu pojazdu, dzięki czemu w intuicyjny sposób możliwe jest odczytywanie liczby osi na podstawie grafiki. Zaprezentowane są również wartości opisujące położenie osi w bryle pojazdu. Informacja na temat wykrycia uniesionej osi przedstawiana była poprzez wizualne przesunięcie figury związanej z odpowiednią osią na grafice. Interfejs wyposażony jest w przyciski umożliwiające nawigację. Wyświetlana jest również informacja o liczbie pojazdów w bazie. W dolnej części okna umiejscowiono stałą skalę, ułatwiającą wizualny odbiór informacji.

Zaprojektowany interfejs stanowi jedynie uproszczony przykład demonstrujący możliwość realizacji takiego zadania. Zaprojektowanie dedykowanego interfejsu graficznego stanowi kierunek praktycznego rozwoju omawianego oprogramowania, nie należało jednak do celów pracy.

# Dokumentacja aplikacji

W rozdziale przedstawiono szereg zagadnień związanych z obsługą aplikacji - instalację, metody uruchamiania, modyfikowania zasad działania programu, czy możliwe zastosowania. Zdecydowano się na ograniczenie do zwięzłej prezentacji możliwości programu, bez zbędnego w tej sytuacji omawiania reguł działania poszczególnych elementów. Dokładniejszy opis programu został przedstawiony przy użyciu mechanizmu podręczników dostępnego w systemie Linux [29]. Strony zawierają przedstawiony w zwięzłej formie zbiór informacji na temat programu, który rozwija zagadnienia omówione w rozdziale tym pracy.

Dostęp do stron podręcznika można uzyskać korzystając z poleceń przedstawionych na listingu 6.2.

**Listing 6.2:** Dostęp do stron podręcznika.

```
man axles
man watcher
```

# 7. Analiza pracy programu

Wśród cech wykonanego projektu, które gwarantują jego użyteczność, znajdują się stabilne działanie aplikacji niezależnie od środowiska i sposobu użycia. Istotne jest również zapewnienie jak najwyższej skuteczności algorytmu oraz optymalizacja wydajności działania aplikacji. W kolejnych sekcjach opisano poszczególne wymagania oraz przedstawiono metody ich weryfikacji. Oceniono również zgodność wykonanego projektu z wymaganiami.

## 7.1. Poprawność algorytmu

Niezależnie od pozostałych kryteriów oceny budowanej aplikacji, konieczne jest zapewnienie poprawności wyników pracy algorytmu. Detekcja liczby osi oraz ich położeń i długości pojazdu stanowi podstawową funkcję programu. Tylko po osiągnięciu zadowalających wyników pracy algorytmów możliwe było przystąpienie do kolejnych etapów projektu.

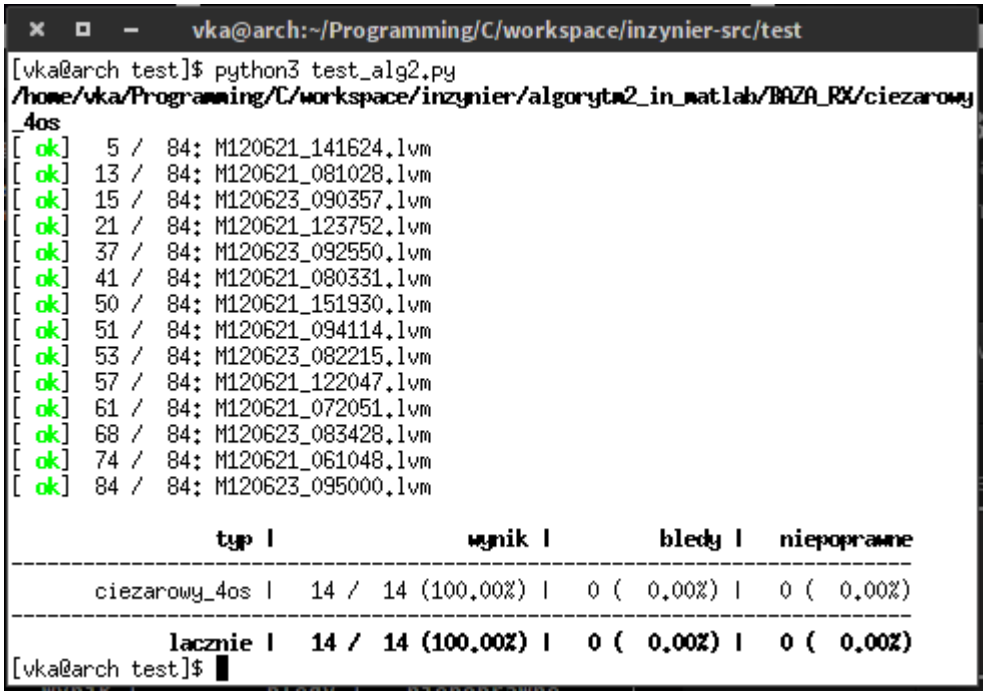
Konieczne okazało się umożliwienie badania działania algorytmu wraz z powstawaniem kolejnych wersji programu. Zdecydowano się na zaprojektowanie skryptu testującego otrzymywane wyniki na podstawie skatalogowanej bazy pomiarów, zawierającej zbiór odczytów z czujników indukcyjnych dla 220 pojazdów. Skrypt napisany został przy użyciu języka programowania *Python*, uzasadniając to wygodą pisania narzędzi testowych w tym środowisku[30].

Przed rozpoczęciem testu możliwa jest wstępna konfiguracja poprzez edycję załączonego pliku *config.json*. Wewnątrz można zdefiniować ścieżkę katalogu zawierającego dane. Definiuje się również podkatalogi, w których pogrupowane są pliki danych o rozszerzeniu *lvm*, dodając poprawną wartość wyniku algorytmu. Możliwe jest również uruchomienie testu dla innego niż domyślny pliku binarnego poprzez edycję odpowiedniego pola. Do projektu dołączono plik zawierający przykładową konfigurację, aby ułatwić to zadanie w przyszłości.

Po zakończeniu wstępnej konfiguracji możliwe jest uruchomienie zestawu testów, stosując poniższe polecenie z poziomu podkatalogu *test* projektu.

```
python3 test_alg2.py
```

W wyniku uruchomienia polecenia w wierszu poleceń pojawił się tekst w formacie przedstawionym na rysunku 7.1.




**Rysunek 7.1.** Wynik pracy programu testowego, badający poprawność pracy algorytmu dla pojazdów ciężarowych czteroosiowych.

W przypadku wykrycia niezgodności wyniku algorytmu z oczekiwanym, poniżej tabeli zawierającej dane liczbowe znajdowała się sekcja z listą plików, dla których wystąpiły niezgodności.

Wyniki pracy finalnej wersji algorytmu dla pełnego zbioru danych, zawierającego dane 220 pojazdów przedstawiono w tabeli 7.1.

| Kategoria | Wynik | Błędy pracy algorytmu | Niepoprawne dane wejściowe |
|---|---|---|---|
| Ciężarowe 2-osiowe | 27 / 27 | 0 | 0 |
| Ciężarowe 3-osiowe | 28 / 28 | 0 | 0 |
| Ciężarowe 4-osiowe | 14 / 14 | 0 | 0 |
| Ciężarowe 5-osiowe | 31 / 31 | 0 | 0 |
| Ciężarowe 2-osiowe, podn. oś | 17 / 17 | 0 | 0 |
| Dostawcze | 33 / 34 | 0 | 1 |
| Osobowe, podwozie aluminiowe | 28 / 29 | 0 | 1 |
| Osobowe, podwozie stalowe | 40 / 40 | 0 | 0 |
| Łącznie | 218 / 220 (99,09%) | 0 (0%) | 2 ( 0,91%) |

**Tablica 7.1.** Wyniki testów

Kolumna *„Wynik”* zawiera liczbę poprawnych detekcji liczby osi dla pojazdu. Kolumna *„Błędy pracy algorytmu”* informuje o liczbie pojazdów, dla których wyznaczono błędną liczbę osi. Kolumna *„Niepoprawne dane wejściowe”* zawiera natomiast

informację o liczbie plików, dla których detekcja zakończyła się błędem związanym z nieczytelnością danych wejściowych, co uniemożliwiło przeprowadzenie procedury algorytmu.

Należy wspomnieć, że badany zbiór testowy zawierał dane wstępnie wyselekcjonowane, których analiza nie powinna zakończyć się błędem pracy algorytmu. Dzięki temu możliwe było badanie poprawności implementacji kolejnych wersji - pojawienie się niezerowej ilości błędów lub wzrost liczby niepoprawnych danych pozwalał na natychmiastowe wykrycie błędu programistycznego. Wystąpienie danych, które nie mogą zostać przeanalizowane przez algorytm, wytłumaczono pojawieniem się zakłóceń pomiarowych, które znacznie zniekształciły odczyt wyników.

Szczegółową analizę działania algorytmu zawarto w rozdziale 5 pracy.

## 7.2. Stabilność pracy programu

Do przyjętych zastosowań projektowanego oprogramowania należą współpraca ze stanowiskiem pomiarowych i analiza napływających danych w sposób ciągły lub uruchomienie programu na serwerze oferującym wysoką wydajność w celu szybkiej analizy dużego zbioru danych. Z tego względu przez cały okres realizacji projektu dużą wagę przykładano do zapewnienia stabilności pracy programu, niezależnie od warunków - metody uruchomienia, formatu danych wejściowych, konfiguracji stanowiska.

Wymuszono ustawianie flag wykrywających niepoprawnie wykorzystywane struktury języka programowania. Umożliwiło to znalezienie błędów programistycznych na możliwie wczesnym etapie projektu. Ostrzeżenia - wskazujące na niekoniecznie niepoprawne instrukcje, ale nieudokumentowane funkcje języka programowania i możliwie błędne zachowania - traktowane były jako błędy i wystrzegano się ich. Ponadto wymuszono detekcję struktur, które domyślnie nie są traktowane jako niebezpieczne, ale mogą powodować błędy w specyficznych sytuacjach.

Dzięki zastosowaniu takiego podejścia zaprojektowane oprogramowanie budowane jest bez zgłoszenia błędu lub ostrzeżenia na testowanych kompilatorach. Znacznie ogranicza to możliwość wystąpienia niepoprawnego zachowania w trakcie działania aplikacji.

Drugą metodą zagwarantowania stabilności pracy jest uważna analiza wykorzystywanych zasobów systemowych. Dostęp do pamięci nienależącej do procesu stanowi naruszenie zasad pracy w środowisku Linux i może spowodować przerwanie działania programu. Należy również pamiętać o zwolnieniu wszelkich wykorzystywanych zasobów. W przeciwnym razie, w przypadku uruchomienia aplikacji dla dużego zbioru danych, liczba zajętych zasobów mogłaby wzrastać podczas wykonania i ostatecznie doprowadzić do wyczerpania dostępnych zasobów pamięci i przerwania pracy programu.

Analizę pracy programu przeprowadzono przy użyciu oprogramowania *Valgrind* - narzędzia dedykowanego do wykrywania wycieków pamięci [28]. Proces badania aplikacji wymagał uruchomienia algorytmu dla zbioru zawierającego kilkaset pomiarów. Narzędzie testowe nie wykryło żadnych problemów z alokacją i wyciekami pamięci. Pozwoliło to na wysunięcie założenia o poprawności implementacji, co zwiększa gwarancję stabilności działania programu.

## 7.3. Wydajność pracy programu

Algorytm detekcji liczby osi pojazdu może być uruchamiany dla zbioru zawierającego wiele pomiarów, a także pracować połączony bezpośrednio ze stanowiskiem pomiarowym. Wymaga to szybkości przetwarzania pozwalającej na uzyskanie wyniku działania dla danych pomiarowych przed pojawieniem się kolejnego zbioru. Konieczne było więc zapewnienie odpowiedniej szybkości działania programu.

Zastosowanie języka C gwarantuje duży wpływ na sposób realizacji operacji algorytmicznych. Jest to język niskiego poziomu, wykorzystujący instrukcje, które są kompilowane bezpośrednio do asemblera. Kilkukrotnie podejmowano zadanie zwiększenia wydajności aplikacji na różnych etapach wykonania projektu. Pozwoliło to ostatecznie na osiągnięcie wydajności, którą uznano za zadowalającą. W tabeli 7.2 przedstawiono wyniki analizy wydajności dla kluczowych wartości, natomiast na wykresie 7.2 zamieszczono wartości uzyskane dla większej liczby pomiarów.

| Liczba pomiarów | Metoda uruchomienia | Czas pracy programu [s] | Czas analizy jednego zbioru danych [s] |
|---|---|---|---|
| 1 | Strumień wejścia[1] | 0,055 | 0,055 |
| | Argument[2] | 0,054 | 0,054 |
| 10 | Strumień wejścia | 0,508 | 0,051 |
| | Argument | 0,499 | 0,050 |
| 100 | Strumień wejścia | 5,174 | 0,052 |
| | Argument | 4,947 | 0,049 |
| 1000 | Strumień wejścia | 50,149 | 0,050 |
| | Argument | 47,597 | 0,048 |
| 10000 | Strumień wejścia | 522,708 | 0,052 |
| | Argument | 510,345 | 0,051 |
| Średni czas analizy pomiaru | | | 0,051 |

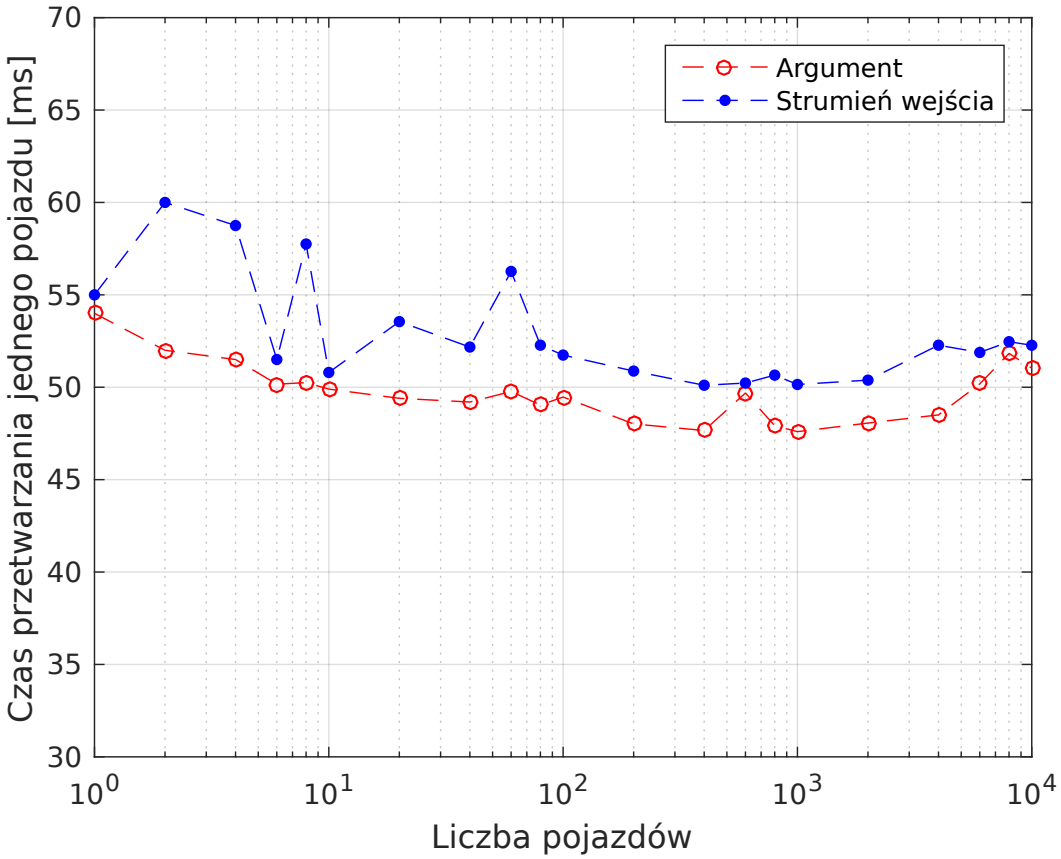
**Tablica 7.2.** Wyniki analizy wydajności pracy programu

Analiza wykonana została na komputerze klasy PC. Użycie urządzenia oferującego większą moc obliczeniową powinno wpłynąć na czas pracy programu.

Czas analizy jednego danych dotyczących jednego pojazdu zbliżony jest do $50$ milisekund, niezależnie od wielkości badanego zbioru wejściowego. Uznano ten wynik

[1] Program uruchomiony przy pomocy polecenia `cat PLIKI_DANYCH |axles -q`
[2] Program uruchomiony przy pomocy polecenia `axles -q PLIKI_DANYCH`

za zadowalający, umożliwiający płynną pracę przy analizie danych nawet z najbardziej obciążonych ruchem połączeń drogowych.




**Rysunek 7.2.** Wyniki analizy wydajności pracy programu

Wydajność programu w sytuacji, gdy dane przekazano poprzez strumień wejścia była nieznacznie gorsza. Wytłumaczono to zwiększeniem nakładu obliczeń na etapie akwizycji danych, ze względu na konieczność wykonania polecenia `cat` przed uruchomieniem modułu algorytmu. Różnica wydajności jest jednak niewielka, rzędu kilku procent, i nie ma dużego wpływu na pracę programu w normalnych zastosowaniach.

Przeprowadzono również profilowanie, dynamiczną analizę kodu przy użyciu oprogramowania *Valgrind*. Uzyskano wyniki wskazujące, że około $85\%$ czasu wykonania poświęcane jest na wczytanie danych z pliku tekstowego lub strumienia wejścia. Jedynie $15\%$ czasu zajmuje wykonanie algorytmu i uzyskanie wyników. Ze względu na użycie dostępnych w bibliotece języka C funkcji wczytywania danych, niemożliwa okazała się dalsza poprawa szybkości w tym zakresie. Jednym z możliwych rozwiązań, które umożliwiłyby wzrost wydajności, jest wykorzystanie wielowątkowego modelu pracy. Użycie kilku wątków programu do ładowania danych, a tylko jednego do wywołań algorytmu pozwoliłoby znacznie zwiększyć wydajność. Byłoby to zauważalne zwłaszcza w przypadku wykorzystania serwerów obliczeniowych, gwarantujących dostęp do wielu rdzeni procesora. Zaprojektowanie aplikacji wielowątkowej

znacznie przedłużyłoby jednak czas wykonania aplikacji i utrudniło jej testowanie, przez co zdecydowano się na użycie tylko jednego wątku.

# Podsumowanie

Celem niniejszej pracy było zaprojektowanie oprogramowania umożliwiającego wykorzystanie danych pomiarowych ze stanowiska wyposażonego w czujniki indukcyjne pętlowe. Detektory tego typu, ze względu na swoją trwałość, niską cenę i użyteczność uzyskiwanych informacji, wykorzystywane mogą być do badania parametrów pojazdów samochodowych oraz ruchu drogowego. Zaprojektowanie oprogramowania pozwala na uzyskanie informacji na temat długości pojazdu, jego prędkość, a także liczby osi oraz ich położenia w jego bryle. W konsekwencji pozwala to na kategoryzację na podstawie wymienionych cech.

W pierwszym rozdziale pracy autor omówił budowę czujników indukcyjnych pętlowych oraz ich zastosowania, na przykładzie dostępnej literatury. Liczba publikacji związanych z badanym zagadnieniem wciąż jest niewielka, co ograniczyło zakres omawianych publikacji.

W rozdziale drugim przedstawiony został zakres realizowanego projektu. Uzasadniono założenia, wymieniono również ograniczenia projektu i opisano interfejs aplikacji. Sprecyzowanie zakresu realizacji pozwoliło zwiększyć kontrolę nad powstającym oprogramowaniem na dalszym etapie prac. Opis ten stanowi również część dokumentacji projektu.

W kolejnym rozdziale omówiono algorytm detekcji liczby osi pojazdów, którego implementacji podjęto się w pracy. Przedstawiony został szczegółowy opis kolejnych etapów, a także procedury korekcji w przypadku uzyskania niepoprawnego wyniku oraz rozpoznania osi podniesionej.

Rozdział czwarty zawiera opis algorytmu detekcji położenia osi w bryle pojazdu. Omówiono również możliwość weryfikacji poprawności wyników na podstawie danych uzyskiwanych z czujników nacisku.

W piątym rozdziale przeprowadzona została analiza skuteczności algorytmu. Omówiono wyniki uzyskiwane dla pojazdów różnych kategorii. Oszacowano spodziewaną dokładność algorytmu wraz z uzasadnieniem. Uzyskane wyniki pozwoliły przyjąć tezę o możliwości praktycznego wykorzystania czujników indukcyjnych w większości aplikacji.

Kolejny rozdział zawiera opis wykonanego programu. Wyszczególnione zostały wymagania systemowe, które muszą być spełnione przed instalacją projektu. Przed-

stawiono opis interfejsu i przykładowe aplikacje. Zaprezentowany został również interfejs graficzny, będący przykładem dalszego rozwoju aplikacji.

W ostatnim rozdziale pracy rozważono trzy czynniki, których spełnienie wymagane jest od dobrze zaprojektowanego programu. Zbadano zachowanie algorytmu oraz stabilność i wydajność aplikacji. Uzyskane wyniki pozwoliły na przyjęcie tezy o poprawności implementacji.

Zaprojektowany program stanowi praktyczny przykład wykorzystania czujników indukcyjnych pętlowych w celu uzyskania informacji na temat pojazdów samochodowych. Dołączone do projektu narzędzia umożliwiają współpracę programu ze stanowiskami pomiarowymi o różnych konfiguracjach. Udostępniono również dokumentację w postaci stron podręcznika, stanowiących podstawowe medium tego typu w środowisku Linux.

Zdaniem autora, możliwy jest dalszy rozwój aplikacji, pozwalający na dostosowanie jej do konkretnych potrzeb. Zaprojektowanie interaktywnego interfejsu graficznego pozwoliłoby na wykorzystanie implementacji algorytmu w praktycznych zastosowaniach, takich jak na przykład systemy pobierania opłat za przejazd pojazdu. Oprogramowanie powinno również zostać zintegrowane z systemem bazodanowym, co pozwoliłoby na zapis wyników w wygodnej formie.

Zaproponowany algorytm detekcji długości pojazdu może zostać rozbudowany, tak by wykorzystywał informację o kategorii uzyskanej na wyjściu algorytmu detekcji liczby osi. Pozwoliłoby to na zwiększenie jego dokładności dla pojazdów każdego typu. Możliwa jest również poprawa wydajności oprogramowania dzięki wykorzystaniu kilku współbieżnych wątków. Znalazłoby to zastosowanie w aplikacjach wymagających największej wydajności. Nie zdecydowano się jednak na wdrożenie tego rozwiązania w omawianej aplikacji ze względu na trudności związane z obsługą i synchronizacją wątków.

# Bibliografia


[1] Albing C., Vossen J., Newham C., *Bash. Receptury.* Helion, 2012.

[2] Ali S. S. M., George B., Vanajakshi L., Venkatraman J., *A Multiple Inductive Loop Vehicle Detection System for Heterogeneous and Lane-Less Traffic*, 2012.

[3] Cheung S. Y., Coleri S., Dundar B., Ganesh S., Tan C.-W., Varaiya P., *Traffic Measurement and Vehicle Classification with a Single Magnetic Sensor*, 2004.

[4] Gajda J., Marszałek Z., Piwowar P., Sroka R., Stencel M., Zegleń T., *System o zmiennej strukturze do pomiaru parametrów ruchu drogowego*, 2010.

[5] Gajda J., Sroka R., Stencel M., Wajda A., Zegleń T., *A Vehicle Classification Based on Inductive Loop Detectors, IEEE Instrumentation and Measurement Technology Conference*, 2001.

[6] Gajda J., Sroka R., Stencel M., Zegleń T., *Metody klasyfikacji pojazdów samochodowych w ruchu*, 2000.

[7] Gajda J., Sroka R., Stencel M., Żegleń T., Burnos P., Piwowar P., *Pomiary parametrów ruchu drogowego.* Wydawnictwa AGH, 2012.

[8] Kaewkamnerd S., Chinrungrueng J., Pongthornseri R., Dumnin S., *Vehicle Classification Based on Magnetic Sensor Signal*, 2010.

[9] Love R., *Linux. Programowanie systemowe.* Helion, 2014.

[10] Marszałek Z., *Analiza właściwości meterologicznych układów do detekcji osi pojazdów z wykorzystaniem czujników indukcyjnych pętlowych*, Praca doktorska, Akademia Górniczo-Hutnicza, 2013.

[11] Marszałek Z., "Detekcja osi pojazdów z użyciem pętli indukcyjnej," *II Sympozjum - Aktualne problemy metrologii*, 2013.

[12] Marszałek Z., Sroka R., Stencel M., *A new method of inductive sensor impedance measurement applied to the identification of vehicle parameters*, 2011.

[13] Meller S. A., Zabaronick N., Ghoreishian I., Allison J., Arya V., Vries M. J.de, Claus R. O., *Performance of fiber optic vehicle sensors for highway axle detection*, 1997.

[14] Robbins A., *Unix in a Nutshell, 4th Edition*. O'Reilly Media, 2005.

[15] Sun C., *An Investigation in the Use of Inductive Loop Signatures for Vehicle Classification*, 2000.

[16] *Biblioteka getopt*. Online: http://man7.org/linux/man-pages/man3/getopt.3.html (Dostęp: 04.08.2015).

[17] *Biblioteka POSIX języka C*. Online: http://pubs.opengroup.org/onlinepubs/9699919799/idx/head.html (Dostęp: 12.11.2015).

[18] *Cmake - narzędzie zarządzania procesem kompilacji*. Online: https://cmake.org/ (Dostęp: 11.11.2015).

[19] *Federal Highway Administration Vehicle Classification*. Online: http://www.dot.ca.gov/hq/tpp/offices/ogm/trucks/FHWA_Vehicle-Classes-With-Definitions%20cs.pdf (Dostęp: 03.09.2015).

[20] *GNU C Library*. Online: https://www.gnu.org/software/libc/index.html (Dostęp: 04.08.2015).

[21] *GNU C++ Library*. Online: https://gcc.gnu.org/onlinedocs/libstdc++/ (Dostęp: 04.08.2015).

[22] *GNU Command Line Interface Standard*. Online: https://www.gnu.org/prep/standards/standards.html#Command_002dLine-Interfaces (Dostęp: 06.09.2015).

[23] *GNU Standard*. Online: https://www.gnu.org/prep/standards/standards.html (Dostęp: 03.09.2015).

[24] *Google C++ Style Guide*. Online: https://google-styleguide.googlecode.com/svn/trunk/cppguide.html (Dostęp: 06.09.2015).

[25] *Język programowania Awk*. Online: https://www.gnu.org/software/gawk/manual/gawk.html (Dostęp: 18.11.2015).

[26] *Kompilator języka C Clang*. Online: http://clang.llvm.org/ (Dostęp: 11.11.2015).

[27] *Kompilator języka C GCC*. Online: https://gcc.gnu.org/ (Dostęp: 04.08.2015).

[28] *Oprogramowanie Valgrind*. Online: http://valgrind.org/ (Dostęp: 04.08.2015).

[29] *Podręczniki programowe w systemie Linux*. Online: https://en.wikipedia.org/wiki/Man_page (Dostęp: 04.08.2015).

[30] *The Python Standard Library*. Online: https://docs.python.org/3/library/index.html (Dostęp: 04.08.2015).

[31] *Qt Documentation*. Online: http://doc.qt.io/qt-5/reference-overview.html (Dostęp: 04.08.2015).

[32] *Ruch GNU*. Online: https://www.gnu.org/home.pl.html (Dostęp: 06.09.2015).

[33] *Standard języka C11*. Online: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf (Dostęp: 06.09.2015).

[34] *Standard języka C++11*. Online: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372 (Dostęp: 11.11.2015).

[35] *System Linux*. Online: http://www.linuxfoundation.org/what-is-linux (Dostęp: 04.08.2015).

[36] *Unix System*. Online: http://www.unix.org/what_is_unix.html (Dostęp: 04.08.2015).

[37] *Potoki w systemie Linux*. Online: http://www.linfo.org/pipes.html (Dostęp: 04.08.2015).