

POLITECHNIKA WARSZAWSKA

Wydział Mechatroniki

Praca dyplomowa inżynierska

Ireneusz Szulc

# Inteligentny interfejs dotykowy umożliwiający obsługę złożonych gestów

Opiekun pracy:  
mgr inż. Bogdan Harasymowicz-Boggio

Instytut Automatyki i Robotyki



Warszawa, 2016 r.



Kierunek:	Automatyka i Robotyka
Specjalność:	Informatyka Przemysłowa
Data urodzenia:	21.03.1993 r.
Data rozpoczęcia studiów:	01.10.2012 r.

## **Życiorys**

Urodziłem się 21 marca 1993 roku w Lublinie. W 2012 roku ukończyłem naukę w klasie o profilu matematyczno-fizycznym Liceum Ogólnokształcącego im. Stefana Czarnieckiego w Chełmie. W tym samym roku rozpocząłem studia dzienne I stopnia na Wydziale Mechatroniki Politechniki Warszawskiej. Od października 2014 roku moją specjalnością jest Informatyka Przemysłowa na kierunku Automatyka i Robotyka. Praktyki przeddyplomowe odbywałem w lipcu 2015 roku w firmie West-Tech Sp. z o. o. w Warszawie, gdzie zajmowałem się projektowaniem aplikacji sieciowych.

<b>PRACA DYPLOMOWA inżynierska</b>	
<b>Specjalność:</b> Informatyka Przemysłowa	<b>nr pracy:</b> D-IAiR-I-16-359
<b>Instytut prowadzący specjalność:</b> Instytut Automatyki i Robotyki	
<b>Instytut prowadzący pracę:</b> Instytut Automatyki i Robotyki	
<b>Temat pracy:</b> Inteligentny interfejs dotykowy umożliwiający obsługę złożonych gestów	
<b>Temat pracy (w jęz. ang.):</b> Intelligent touch user interface for handling complex gestures	
<b>Zakres pracy:</b> <ol style="list-style-type: none"> <li>1. Projekt i implementacja algorytmu rozpoznawania gestów.</li> <li>2. Projekt modułu wprowadzania znaków na podstawie pisma odręcznego.</li> <li>3. Projekt i implementacja oprogramowania na urządzenie mobilne.</li> </ol>	
<b>Podstawowe wymagania:</b> <ol style="list-style-type: none"> <li>1. Oprogramowanie powinno działać na urządzeniu mobilnym wyposażonym w system operacyjny Android.</li> <li>2. Interfejs powinien umożliwiać rozpoznawanie znaków pisma odręcznego wprowadzanych na panelu dotykowym.</li> <li>3. System powinien uczyć się w trakcie działania.</li> </ol>	
<b>Literatura:</b> <ol style="list-style-type: none"> <li>1. Tadeusiewicz R., <i>Rozpoznawanie obrazów</i>, PWN, Warszawa, 1991</li> <li>2. Bradski G., Kaehler A., <i>Learning OpenCV</i>, 2008</li> <li>3. Baggio D., <i>Mastering OpenCV with Practical Computer Vision Projects</i>, 2012</li> </ol>	
<b>Imię i nazwisko dyplomanta:</b> Ireneusz Szulc	<b>Imię i nazwisko promotora:</b> mgr inż. Bogdan Harasymowicz-Boggio
<b>Temat wydano dnia:</b> 19.10.2015	<b>Termin ukończenia pracy:</b> 01.02.2016
<b>Miejsce wykonywania praktyki przeddyplomowej:</b> West-Tech Sp. z o.o., ul. Odrowąża 15, 03-310 Warszawa	
<b>Zatwierdzenie tematu</b>	
dr inż. Paweł Wnuk  Opiekun specjalności	dr inż. Krzysztof Kukielka  Z-ca dyrektora d/s nauczania

## Streszczenie

Przedmiotem niniejszej pracy jest inteligentny interfejs dotykowy do obsługi złożonych gestów, mający formę aplikacji przeznaczonej na urządzenia mobilne z ekranem dotykowym. Tematyka pracy związana jest z zagadnieniem szeroko pojętego rozpoznawania obrazów, w szczególności skupiono się na klasyfikacji gestów.

W przedstawionej pracy zaproponowana została autorska metoda identyfikacji wieloelementowych gestów. Pozwala ona na rozpoznawanie znaków pisma odręcznego i w tym zakresie została przedstawiona główna funkcjonalność aplikacji. Stworzony interfejs dotykowy daje użytkownikowi możliwość definiowania własnych, dowolnych gestów o wybranej złożoności.

Do klasyfikacji nieznanych gestów opracowane zostały dwa algorytmy. Pierwszy z nich odpowiedzialny jest za rozpoznawanie pojedynczych konturów, natomiast drugi rozszerza tę funkcjonalność o identyfikację wieloelementowych, złożonych gestów. Zaprezentowano również własną metodę wprowadzania tekstu poprzez wykonywanie gestów reprezentujących znaki pisma odręcznego.

W kolejnych etapach poruszone zostało zagadnienie inteligencji stworzonego interfejsu dotykowego, polegającej m.in. na uczeniu się systemu w trakcie działania oraz dokonywaniu wnioskowania na podstawie niekompletnych informacji. W pracy zostało opisane zachowanie się aplikacji prowadzące do nauki charakteru pisma użytkownika. Zaprezentowano także mechanizm zwiększania skuteczności w rozpoznawaniu gestów oraz poprawy "samodzielności" systemu wraz z dłuższym czasem użytkowania aplikacji.

W ostatniej części zaprezentowano efekt końcowy - gotową aplikację działającą na urządzeniu mobilnym z systemem operacyjnym Android, która realizuje założoną funkcjonalność. Przedstawiono również wyniki przeprowadzonych testów stworzonego interfejsu dotykowego.

## **Abstract**

The subject of this thesis is related to pattern recognition. In particular, it focuses on the classification of complex gestures. The topic of the present thesis is intelligent touch user interface, which has the form of an application implemented on a mobile device with a touch screen.

The thesis describes a method for identifying multiple gestures, which allows to recognize handwriting characters and that is the field of the main functionality of the presented application. Created touch screen interface allows the user to define custom gestures for any complexity.

To recognize unknown gestures, two algorithms were implemented. The first one is responsible for identifying the single contours, while the second one extends this functionality with the recognition of multiple, complex gestures. The thesis also presents a new input method that is based on performing gestures representing the handwriting characters.

The next chapters describe the intelligence of the touch interface, which is based on machine learning and inference from incomplete information. The paper describes the behaviour of the application that leads to the handwriting training. It also presents an approach to improve the effectiveness in recognizing gestures and improve autonomous decision-making of the system with longer in-service time.

The last section presents the final result, which is an application running on a mobile device with Android operating system. The implemented software fulfills the functionality assumptions. Eventually, the present thesis describes the results of software testing that have been carried out.

# Spis treści

<b>1. Wprowadzenie .....</b>	<b>1</b>
<b>2. Metody wizyjne.....</b>	<b>4</b>
2.1. Rozpoznawanie obrazów .....	4
2.2. Syntaktyczne rozpoznawanie obrazów .....	4
2.3. Metody minimalnoodległościowe .....	5
2.4. Metody rozpoznawania pisma odręcznego.....	6
2.5. Kody łańcuchowe Freemana .....	6
2.6. Metody porównywania histogramów .....	7
2.7. Biblioteka OpenCV .....	8
2.8. Słownik używanych pojęć .....	9
<b>3. Algorytm rozpoznawania pojedynczych konturów .....</b>	<b>10</b>
3.1. Założenia algorytmu .....	10
3.2. Wybór metody .....	12
3.3. Akwizycja konturu .....	13
3.4. Przetwarzanie wstępne .....	13
3.4.1. Interpolacyjne uzupełnienie listy punktów .....	13
3.4.2. Filtracja .....	13
3.5. Ekstrakcja cech .....	14
3.6. Reprezentacja obiektu.....	15
3.7. Klasyfikacja .....	16
3.7.1. Korelacja histogramów .....	16
3.7.2. Korelacja punktów startowych .....	16
3.7.3. Korelacja długości gestu .....	17
3.7.4. Sumaryczny współczynnik korelacji .....	18
3.8. Przypadek krótkich konturów .....	19
3.9. Schemat algorytmu .....	20
3.10. Przykłady .....	22
<b>4. System rozpoznawania złożonych gestów .....</b>	<b>24</b>
4.1. Opis algorytmu .....	24
4.1.1. Kryteria wyboru .....	24
4.1.2. Dane wejściowe .....	25
4.1.3. Wstępna klasyfikacja .....	25

4.1.4. Wypadkowy współczynnik korelacji .....	26
4.1.5. Wybór najlepszego rozwiązania .....	26
4.2. Schemat algorytmu .....	27
4.3. Przykłady .....	29
<b>5. Inteligencja systemu .....</b>	<b>32</b>
5.1. Problem błędnych wzorców .....	32
5.2. Rejestrowanie bilansu identyfikacji.....	32
5.3. Automatyczne usuwanie błędnych wzorców .....	33
5.4. Automatyczne dodawanie wzorców .....	33
5.5. Optymalizator liczby wzorców .....	34
5.6. Interakcja z użytkownikiem.....	35
<b>6. Implementacja oprogramowania na urządzenie mobilne .....</b>	<b>36</b>
6.1. Urządzenie i środowisko programowania .....	36
6.2. Struktura aplikacji.....	37
6.3. Obsługa aplikacji .....	39
6.3.1. Interfejs użytkownika.....	40
6.4. Wykorzystane metody .....	43
6.4.1. Wykorzystanie biblioteki OpenCV .....	43
6.4.2. Mechanizm serializacji .....	43
6.5. Moduł wprowadzania znaków pisma odręcznego .....	44
6.6. Cechy systemu .....	44
<b>7. Testy aplikacji.....</b>	<b>45</b>
7.1. Dane statystyczne .....	45
7.2. Test uczenia się systemu.....	45
7.3. Dobór parametrów .....	46
7.4. Testy końcowe działającej aplikacji .....	47
<b>8. Wnioski.....</b>	<b>48</b>
<b>Literatura .....</b>	<b>49</b>
<b>Załączniki .....</b>	<b>50</b>

# 1. Wprowadzenie

Celem pracy jest zaprojektowanie i stworzenie inteligentnego interfejsu dotykowego umożliwiającego obsługę złożonych gestów. Taki interfejs w formie oprogramowania na urządzenie z ekranem dotykowym powinien być odpowiedzialny za rozpoznawanie sekwencji elementarnych konturów rysowanych przez użytkownika poprzez przeciąganie palca lub rysika po ekranie. Wynik analizy wprowadzonych kombinacji gestów może posłużyć do rozpoznawania znaków odręcznego pisma lub wykonania zdefiniowanych akcji w systemie, na którym pracuje urządzenie. Taki system powinien pozwalać użytkownikowi na swobodne definiowanie własnych gestów o dowolnej złożoności.

Inteligencja interfejsu, będącego przedmiotem pracy, ma polegać na zdolności systemu do samouczenia się, co w praktyce oznacza, że aplikacja powinna być w stanie zwiększać swoją skuteczność w rozpoznawaniu znaków poprzez dostosowywanie się do charakteru pisma odręcznego użytkownika oraz zmianę swojego zachowania wskutek wydawanych poleceń.

Opracowany system mógłby znaleźć zastosowanie jako wygodna metoda wprowadzania znaków na urządzeniach z panelem dotykowym. Takie rozwiązanie nie ograniczałoby się jedynie do wprowadzania liter, ale pozwalałoby także na równie szybkie wprowadzanie cyfr, znaków diakrytycznych (polskich liter) oraz znaków specjalnych (takich jak: przecinek, kropka, wykrzyknik, itd.), co mogłoby przemawiać na korzyść metody w porównaniu do konwencjonalnych klawiatur ekranowych, gdzie wprowadzanie znaków specjalnych może stać się nieco bardziej uciążliwe i czasochłonne. Metoda ta jest bardzo intuicyjna, charakteryzuje się prostotą i mogłaby być chętniej wykorzystywana np. przez ludzi starszych przyzwyczajonych do tradycyjnego wprowadzania tekstu.

Kolejnym polem zastosowań są urządzenia z bardzo małymi ekranami dotykowymi (np. smartwatche), gdzie często wykorzystanie klawiatury ekranowej może być uciążliwe z powodu bardzo małych przycisków, zaś wprowadzenie sterowania urządzeniem poprzez gesty oraz pisanie dowolnych znaków metodą pisma odręcznego mogłoby zwiększyć komfort użytkowania takiego urządzenia.

System mógłby posłużyć również jako metoda wprowadzania haseł w postaci sekwencji gestów w celu uzyskania dostępu do określonych zasobów. Przetrzymywanie hasła w postaci wzorca charakteru pisma mogłoby być wykorzystane w procesie uwierzytelniania użytkowników i przyznawać dostęp jedynie osobie, która ma dokładnie taki sam charakter



pisma lub w dokładnie taki sam sposób wykonała określone gesty. System z takim rozwiązaniem nie przyznawałby dostępu osobie, która poznała hasło, lecz wprowadziła je w inny sposób.

Interfejs dotykowy będący przedmiotem pracy można również wykorzystać jako rozszerzenie systemu, na którym pracuje urządzenie, do skrótowego uruchamiania aplikacji lub niektórych funkcji systemu. Opracowaną metodę można zaimplementować także na panelach operatorskich HMI z ekranem dotykowym.

Na rynku istnieje już wiele systemów (np. mobilne aplikacje) o funkcjonalności zbliżonej do tematyki pracy. Zauważa się jednak brak istnienia interfejsów dotykowych łączących wiele krytycznych wymagań stawianych takim systemom. Wiele istniejących systemów rozpoznaje znaki jedynie na podstawie statycznej informacji o niej, nie uczy się w trakcie działania, nie oferuje rozpoznawania wieloelementowych gestów, nie pozwala na zdefiniowanie własnych, dowolnych gestów i znaków lub też nie radzi sobie z rozpoznawaniem polskich liter.

Założeniem projektu jest, aby interfejs dotykowy miał formę aplikacji działającej na urządzeniu mobilnym wyposażonym w system operacyjny Android. Dodatkowo takie urządzenie powinno posiadać ekran dotykowy, który posłuży za źródło danych wejściowych dla aplikacji. Językiem programowania wykorzystanym do stworzenia całej aplikacji będzie obiektowy język Java.

Interfejs powinien umożliwiać rozpoznawanie znaków pisma odręcznego wprowadzanych przez użytkownika poprzez ruch palca lub rysika po ekranie dotykowym, zamianę ich na tekst w formie cyfrowej i przechowywanie go w celu dalszego wykorzystania. Wprowadzane znaki mogą składać się zarówno z wielu fragmentów pisanych z odrywaniem rysika od ekranu, jak również z pojedynczych konturów.

Dodatkowym założeniem jest, aby system uczył się w trakcie działania. Aplikacja powinna nauczyć się charakteru pisma odręcznego od użytkownika, powinna także zwiększać swoją skuteczność w rozpoznawaniu gestów w miarę dłuższego czasu użytkowania. Cały system powinien również reagować na polecenia uczące wydawane przez użytkownika i zmieniać pod ich wpływem sposób swojego działania.

Zakres pracy obejmował trzy główne zagadnienia:

- projekt i implementacja algorytmu rozpoznawania elementarnych gestów,
- opracowanie metody rozpoznawania złożonych gestów oraz wprowadzania znaków na podstawie pisma odręcznego,
- projekt i implementacja oprogramowania na urządzenie mobilne.

Praca skupia się głównie na rozpoznawaniu znaków pisma odręcznego, ale łatwo można rozszerzyć tę funkcjonalność systemu na identyfikację gestów wykonujących określone akcje w systemie (np. skrótowe uruchamianie aplikacji), gdyż zaprojektowany system nie ma ograniczenia rozpoznawania dowolnych gestów o dowolnej złożoności.

## **2. Metody wizyjne**

W tej części pojawią się pojęcia teoretyczne związane z tematyką widzenia maszynowego, które są szczególnie przydatne w klasyfikacji obrazów, a jeszcze szczególnie w rozpoznawaniu znaków pisanych.

### **2.1. Rozpoznawanie obrazów**

Zagadnienie rozpoznawania obrazów często ograniczane jest jedynie do identyfikacji dwuwymiarowych ilustracji lub trójwymiarowych scen, natomiast pojęcie obrazu powinno być rozpatrywane znacznie szerzej.

Według Ryszarda Tadeusiewicza w ogólnym zadaniu rozpoznawania obrazów chodzi o rozpoznawanie przynależności rozmaitego typu obiektów (lub zjawisk) do pewnych klas [6]. Proces rozpoznawania często jest prowadzony w sytuacji braku informacji o regułach przynależności obiektów do poszczególnych klas. W niektórych przypadkach jedyne informacje możliwe do wykorzystania przez algorytm rozpoznający są zawarte w ciągu uczącym, składającym się z obiektów, dla których znana jest poprawna klasyfikacja.

Algorytm rozpoznawania powinien "uczyć się" identyfikacji na podstawie przedstawionych przykładów z ciągu uczącego. Zasadniczym elementem tej nauki jest uogólnianie. Maszyna otrzymuje przykłady jedynie niektórych obiektów (pochodzących z ciągu uczącego), zaś w trakcie rozpoznawania powinna orzekać o wszystkich obiektach. Przekazanie maszynie umiejętności rozpoznawania musi bazować na prezentacji pokazów, ponieważ algorytm tej czynności jest nieznany.

Ujednolicone podejście do metod rozpoznawania wprowadza podział całego procesu na zasadnicze elementy. Wstępnym etapem jest pomiar cech, które opisują rozpoznawane obiekty, natomiast dalsze działanie bazuje na analizie tych cech. Po etapie określenia cech dowolny obiekt scharakteryzowany jest zbiorem wartości cech (wektorem cech) i na tych danych można wykonywać obliczenia w celu stwierdzenia, do jakiej klasy należy nieznany obiekt.

W niektórych sytuacjach możliwa oraz celowa jest transformacja cech. Polega to na przekształcaniu jednych cech, które opisują rozpoznawane obiekty, w inne, nowe cechy mogące pozwolić na łatwiejszą interpretację.

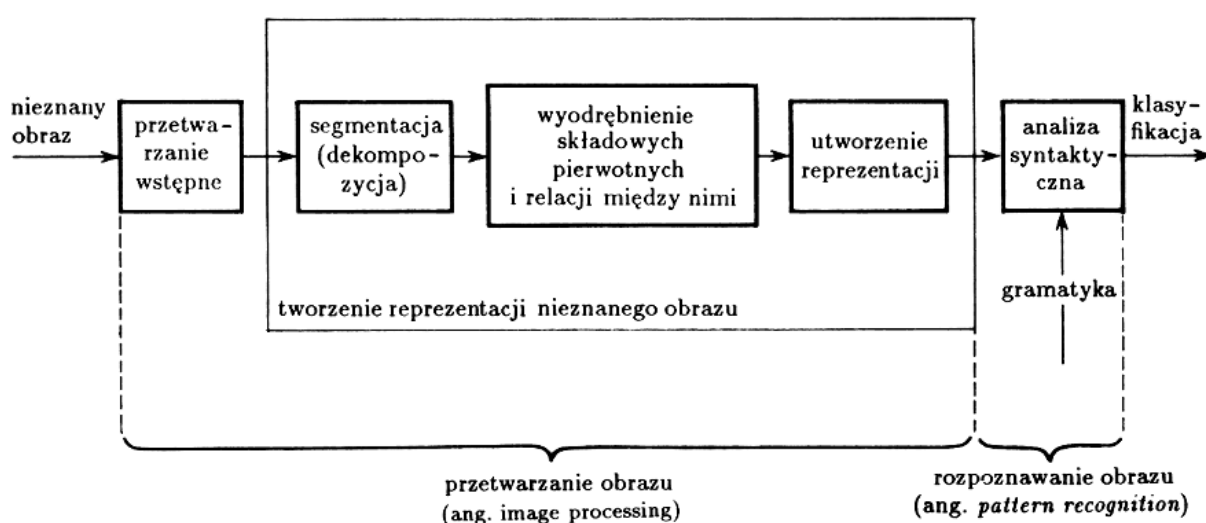
### **2.2. Syntaktyczne rozpoznawanie obrazów**

Cechą syntaktycznego rozpoznawania obrazów jest analiza struktury obiektu oraz poszczególnych elementów, z jakich się składa. Podejście syntaktyczne wprowadza podział

złożonego obrazu na prostsze podobrazy. Operacja podziału kontynuowana jest dotąd, aż otrzymane zostaną składowe pierwotne obrazu.

Jedną z grup strukturalnego rozpoznawania obrazów są metody ciągowe. Obiekt przedstawiany jest przez ciąg pewnych elementów. Elementy te powiązane są ze sobą relacją konkatenacji (są do siebie "doklejane"). Klasycznym przykładem języków ciągowych służących do opisu i identyfikacji składowych elementów obrazu są języki oparte na kodach łańcuchowych Freemana.

W ogólnym schemacie toru przetwarzania obiektów można wyróżnić zasadnicze etapy strukturalnego rozpoznawania obrazów. Zostały one zilustrowane na poniższym rysunku.



Rysunek 2.1. Schemat toru przetwarzania obiektów z wyróżnieniem zasadniczych etapów strukturalnego rozpoznawania obiektów [6]

### 2.3. Metody minimalnoodległościowe

Jedną z grup metod szeroko pojętego rozpoznawania obrazów są metody minimalnoodległościowe. Zasada takich metod opiera się na następującym podejściu: jako rozpoznanie należy wybrać tę klasę, do której należy obiekt najbliższy (według przyjętej metryki) rozpoznawanemu obiektowi (a dokładniej reprezentującemu go wektorowi cech). Rozumowanie to jest bardzo proste, natomiast trudnością może stać się sam wybór odpowiedniej metryki będącej miarą odległości między elementami. Do rozpoznawania według tej metody konieczne jest również zdefiniowanie zbioru uczącego, dla którego elementów znane są etykiety klas.

Warto dodać, że oprócz metod minimalnoodległościowych, ważną grupą metod rozpoznawania obrazów są także klasyfikatory dzielące przestrzeń cech na wiele obszarów

reprezentujących klasy. Przykładem takiej metody jest maszyna wektorów nośnych (SVM), której celem jest wyznaczenie hiperpłaszczyzny (lub zbioru hiperpłaszczyzn) w wielowymiarowej przestrzeni cech rozdzielającej obiekty z jak największym marginesem.

## **2.4. Metody rozpoznawania pisma odręcznego**

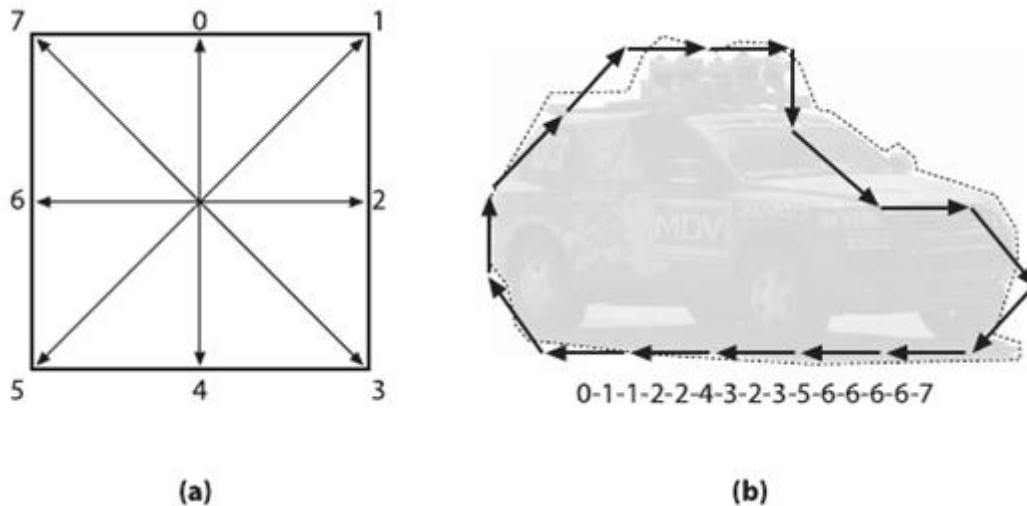
Istnieje wiele rozwiązań oraz aplikacji służących do rozpoznawania pisma odręcznego. Większość z nich zalicza się do grupy metod rozpoznawania charakteru pisma "off-line", których zadaniem jest automatyczna konwersja tekstu z obrazu zdjęcia na cyfrowe kody liter. Dwuwymiarowe zdjęcie litery jest statyczną reprezentacją znaku podlegającego identyfikacji. Popularnym oprogramowaniem realizującym to zadanie jest pakiet Tesseract będący darmowym silnikiem do optycznego rozpoznawania znaków (OCR) dla wielu systemów operacyjnych.

Metody rozpoznawania pisma odręcznego typu "on-line" polegają na nieco innym podejściu. Rozpoznawanie opiera się na analizowaniu na bieżąco procesu powstawania znaku. Umożliwiają to specjalne panele dotykowe, w których sensor odbiera informacje o ruchu rysika, zdarzeniu rozpoczęcia pisania lub oderwania rysika od ekranu. Taki sposób akwizycji znaku pozwala na reprezentację litery wraz z dynamiką jej powstawania.

Często w procesie identyfikacji nieznanych znaków wykorzystywane są metody oparte na sieciach neuronowych, maszynie wektorów nośnych lub kodach łańcuchowych Freemana. Wykorzystanie sieci neuronowych pozwala na stworzenie samouczącego się systemu, który automatycznie uaktualnia bazę danych dla nowych wzorców pisma odręcznego.

## **2.5. Kody łańcuchowe Freemana**

Zazwyczaj kontury przedstawiane są w postaci sekwencji punktów. Alternatywną reprezentacją obiektów są kody łańcuchowe Freemana. Polega to na tym, że kontur jest reprezentowany w postaci sekwencji kroków w jednym z ośmiu kierunków. Każdemu kierunkowi przypisana jest liczba całkowita od 0 do 7. Krok jest wyznaczany na podstawie wektora przemieszczenia dla dwóch sąsiednich punktów. Taka transformacja dotyczy zarówno konturów otwartych, jak i zamkniętych (wielokątów). Łańcuchy Freemana są użyteczne w rozpoznawaniu obrazów. Poniższy rysunek ilustruje zasadę powstawania łańcuchów Freemana:



Rysunek 2.2. a) kierunki przyporządkowane odpowiednim kodom, b) zewnętrzny kontur obiektu w reprezentacji kodów łańcuchowych Freemana [2]

## 2.6. Metody porównywania histogramów

Histogram jest sposobem przedstawienia rozkładu liczebności lub częstości występowania pewnego zbioru elementów poddawanego klasyfikacji według określonej cechy.

Normalizacją histogramu jest przekształcenie polegające na takim przeskalowaniu wszystkich wartości histogramu, aby ich suma była równa 1.

Istnieje wiele metryk, które wyrażają stopień podobieństwa między dwoma rozkładami. Najważniejsze z nich z perspektywy przetwarzania obrazów zaprezentowano poniżej:

- **Współczynnik korelacji Pearsona**

Współczynnik korelacji liniowej Pearsona to współczynnik, który określa poziom zależności liniowej między zmiennymi losowymi  $x$  i  $y$ . Niech  $x_i$  oraz  $y_i$  oznaczają wartości prób losowych tych zmiennych, zaś  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$  i  $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$  wartości średnie z tych prób.

Estymator współczynnika korelacji liniowej Pearsona wyraża się wzorem:

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (2.1)$$

Wartość współczynnika korelacji Pearsona zawsze mieści się w przedziale:

$$r_{xy} \in [-1; 1]$$

Większa wartość oznacza lepsze dopasowanie. Dla zupełnego dopasowania współczynnik wynosi 1, zaś wartość równa 0 wskazuje na brak korelacji (losowy związek).

- **Chi-kwadrat**

Korelacja dwóch histogramów może być wyrażona również poprzez wykonanie testu zgodności chi-kwadrat. Wartość testu jest oceniana za pomocą rozkładu chi kwadrat. Współczynnik zgodności dla histogramów  $H_1(i)$  oraz  $H_2(i)$  wyraża się wzorem:

$$d_{chi-square}(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)} \quad (2.2)$$

Mniejsza wartość współczynnika oznacza lepsze dopasowanie histogramów, natomiast idealna zgodność reprezentowana jest przez wartość 0.

- **Metoda przecięcia**

W metodzie przecięcia stopień podobieństwa histogramów wyrażony jest sumą części wspólnej z wartości histogramów  $H_1$  i  $H_2$ :

$$d_{intersection}(H_1, H_2) = \sum_i \min(H_1(i), H_2(i)) \quad (2.3)$$

Mniejsza wartość współczynnika oznacza lepszą korelację, natomiast jeżeli oba histogramy są znormalizowane do wartości 1, to w przypadku idealnego dopasowania współczynnik ten jest równy 1. Metoda przecięcia charakteryzuje się małą złożonością obliczeń, gdyż wykonywane są jedynie bardzo proste operacje.

- **Odległość Bhattacharyya**

Odległość Bhattacharyya jest wykorzystywana do oceny różnicy między dwoma rozkładami. Dla dyskretnych rozkładów  $H_1(i)$  oraz  $H_2(i)$  wyraża się wzorem:

$$d_{Bhattacharyya}(H_1, H_2) = \sqrt{1 - \sum_i \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum_i H_1(i) \cdot \sum_i H_2(i)}}} \quad (2.4)$$

Mała wartość odległości Bhattacharyya wskazuje na dobre dopasowanie. W przypadku zupełnej zgodności porównywanych rozkładów odległość ta wynosi 0.

## 2.7. Biblioteka OpenCV

OpenCV jest biblioteką oprogramowania do obróbki obrazów i uczenia maszynowego. Biblioteka została wydana na licencji BSD i jest darmowa dla użytku naukowego oraz komercyjnego. Spośród wielu języków programowania i platform systemowych posiada m.in. implementację w języku Java dla systemu operacyjnego Android. OpenCV zostało

zaprojektowane z myślą o efektywności obliczeń oraz zastosowanie w aplikacjach z przetwarzaniem w czasie rzeczywistym.

## 2.8. Słownik używanych pojęć

W pracy stosowane są następujące pojęcia, które mogą być niejasne i wymagać wyjaśnienia:

**Gest elementarny, gest składowy, gest pojedynczy** - kontur (uszeregowana lista punktów) powstały w wyniku jednorazowego przesunięcia palca lub rysika po ekranie bez odrywania go od ekranu. Istotna jest kolejność punktów, a więc kontur jest ciągiem a nie jedynie zbiorem punktów. W omawianych przypadkach kształt konturu najczęściej reprezentuje krzywą otwartą.

**Gest złożony, gest wieloelementowy** - sekwencja wielu następujących po sobie gestów elementarnych, w której istotna jest kolejność elementów;

**Złożoność gestu** - liczba gestów elementarnych wchodzących w skład gestu złożonego. Przykładem złożonego gestu jest rysowanie kształtu dużej litery "A", która pisana jest zazwyczaj dwuetapowo i składa się z dwóch elementów: kształtu "Λ" oraz poziomej kreski "-". Zatem złożoność takiego gestu wynosi 2.

**Punkt startowy konturu** - pierwszy punkt konturu, miejsce jego początku.



### 3. Algorytm rozpoznawania pojedynczych konturów

Rozdział ten skupia się na opracowaniu i zaimplementowaniu własnej metody rozpoznawania pojedynczych gestów składowych (bez rozważania kontekstu, w jakim zostały wprowadzone).

Ogólny proces rozpoznawania obiektów można podzielić na kilka zasadniczych części:

- przetwarzanie wstępne danych wejściowych,
- ekstrakcja cech,
- utworzenie reprezentacji nieznanego obiektu,
- właściwe rozpoznawanie (klasyfikacja).

Zastosowany algorytm opiera się na jednej z metod minimalnoodległościowych - metodzie najbliższego sąsiada. Polega ona na bardzo prostym rozumowaniu mówiącym, że wynikiem rozpoznania jest klasa, do której należy obiekt najbliższy (według przyjętej metryki) rozpoznawanemu obiektowi.

W przypadku rozważanego w projekcie zagadnienia rozpoznawania konturów (będących danymi wejściowymi dla algorytmu), obiektami są właśnie poszczególne kontury, które mogą być pogrupowane w klasy. Obiekty te leżą w wielowymiarowej przestrzeni (liczba wymiarów przestrzeni jest równa liczbie cech obiektów). Położenie obiektów jest więc w pełni uwarunkowane cechami obiektów, natomiast metryka określa w tym przypadku stopień podobieństwa konturów (jak bardzo dwa obiekty są do siebie zbliżone). Na tej podstawie można wyznaczyć stopień podobieństwa nowych danych wejściowych (niesklasyfikowanych) do każdego obiektu, który już został sklasyfikowany. Następnie można przyporządkować nieznanemu obiektowi klasę, do której należy najbardziej zbliżony (o minimalnej odległości) znany obiekt. Zasadniczą trudnością w tej metodzie staje się zdefiniowanie samej metryki określającej odległość między obiektami (stopień ich podobieństwa). Taka metoda implikuje również konieczność przechowywania wzorców w wewnętrznej bazie aplikacji, które przynależą już do określonych klas i mogą stać się podstawą do identyfikacji nowo wprowadzonych, nieznanymi obiektów.

#### 3.1. Założenia algorytmu

Aby uzyskać jak największą skuteczność w poprawnym rozpoznawaniu elementarnych gestów i jednocześnie zachować pewną tolerancję danych wejściowych (nowych, nierozpoznanych konturów) na zniekształcenia w stosunku do wzorca, konieczne jest

rozważenie potencjalnie problematycznych przypadków oraz określenie wymagań w stosunku do algorytmu. Może to pomóc w uniknięciu błędnego działania algorytmu w przyszłości.

Każdy kontur wpisany przez użytkownika jest zniekształconym naśladownictwem pewnego abstrakcyjnego idealnego wzorca, do którego dążą w sposób niedoskonały wszystkie konkretne realizacje obiektów z danej klasy. Należy więc pamiętać o tym, że proces identyfikacji konturów powinien być niewrażliwy (w odpowiednim zakresie) na wybrane transformacje obrazu.

Rozważany algorytm powinien spełniać poniższe założenia wynikające z charakteru wykonywania gestów przez użytkownika:

- **uwzględnianie kierunku wykonywania gestu** - Program powinien rozróżniać między sobą gesty, które mają ten sam kształt konturu, lecz powstały w wyniku odwróconej sekwencji ruchów na ekranie dotykowym. Właściwość kierunku konturu można odczytać tylko w przypadku metod "on-line Handwriting Recognition", gdyż znana jest cała historia i dynamika powstawania konturu. Podejście to pozwala śledzić na bieżąco cały proces powstawania konturu. Natomiast w przypadku metod "off-line Handwriting Recognition", nie można stwierdzić, jaki był kierunek powstawania konturu na podstawie samego statycznego obrazu (np. z kamery), będącego jedynie końcowym efektem pisania znaku. Znajomość cechy kierunku jest dużą zaletą metody, gdyż umiejętne jej wykorzystanie może posłużyć do łatwego odróżnienia pewnych obiektów, których odróżnienie mogłyby być problematyczne bez znajomości dynamiki konturu. Tym samym może się to przyczynić do poprawy skuteczności rozpoznawania algorytmu.
- **częściowa niewrażliwość na obrót** (jedynie w odpowiednim zakresie) - Użytkownik powinien mieć możliwość wpisania znaku w nieco innej, zniekształconej orientacji, ale tylko w ustalonym, niewielkim zakresie. Pełna niewrażliwość na obrót może prowadzić do błędnego, jednakowego interpretowania niektórych znaków, np. dużych liter "M" i "W", "N" i "Z", lub cyfr "6" i "9", których ogólny kształt może różnić się jedynie orientacją, co mogłoby prowadzić do nierozróżniania takich znaków między sobą.
- **częściowa niewrażliwość na przesunięcie** (jedynie w odpowiednim zakresie) - Za każdym razem gest wykonywany jest nieco inaczej przez użytkownika, również punkt rozpoczęcia rysowania znajduje się w różnych miejscach. Zatem tutaj także

pełna niewrażliwość na przesunięcie konturu w układzie współrzędnych ekranu nie sprawdziłaby się w pewnych przypadkach, np. napisanie znaku "ż" mogłoby być interpretowane przez aplikację jako dwa osobne gesty: kombinacja litery "z" i kropki ".". W takiej sytuacji znajomość umiejscowienia kropki nad literą albo w dolnej części ekranu może rozsądzić problematyczną analizę i dać poprawny rezultat. Podobna sytuacja mogłaby wystąpić w przypadku litery "ć" oraz kombinacji litery "c" i przecinka ",".

- **częściowa niewrażliwość na skalowanie** (jedynie w odpowiednim zakresie) - podobnie jak w poprzednich przypadkach właściwość ta powinna obowiązywać tylko w ustalonym zakresie. Ma to na celu lepsze rozpoznawanie między sobą małych i dużych liter alfabetu, które w niektórych przypadkach charakteru pisma mogą różnić się jedynie wielkością.
- **wydobycie ogólnych cech obiektów** - Algorytm nie powinien analizować szczegółowych kształtów konturu a powinien w miarę możliwości aproksymować jego cechy geometryczne.

### 3.2. Wybór metody

Do identyfikacji obiektów będących konturami zdecydowano się wykorzystać kody łańcuchowe Freemana oraz reprezentację cech geometrycznych konturu w postaci histogramu obrazującego rozkład tych kodów. Doświadczalnie określono, że najlepszy rezultat daje wykorzystanie właśnie tych technik. Dodatkowo informację o obiekcie rozszerza również informacja o współrzędnych punktu na ekranie, w którym gest zaczął być rysowany oraz liczba pikseli, z których składa się kontur wyrażająca jego długość. Podsumowując, pełna informacja o cechach geometrycznych pojedynczego gestu składowego, która wystarcza do identyfikacji obiektów, reprezentowana jest i przechowywana w trzech elementach:

- histogram kodów łańcuchowych Freemana,
- współrzędne punktu początku konturu,
- całkowita liczba pikseli konturu.

Przewiduje się, że cała aplikacja z kompletną bazą wzorców będzie przechowywać nawet do kilkuset wzorców po długim czasie użytkowania przez użytkownika. Dlatego też od aplikacji oczekuje się, że będzie rozpoznawać znaki w możliwie najkrótszym czasie tak, aby dać użytkownikowi wrażenie płynnego wprowadzania tekstu. Z tego względu kody łańcuchowe

Freemana jako metoda reprezentacji obiektu wydają się być dobrym wyborem z perspektywy małej zajętości miejsca w pamięci oraz stosunkowo szybkiego algorytmu porównywania wzorców, gdyż dane histogramu charakteryzujące kształt przechowywane mogą być jako tablica jedynie ośmiu liczb zmiennoprzecinkowych. Dokładny opis algorytmu i zasada działania zostały przedstawione w kolejnych podrozdziałach.

### **3.3. Akwizycja konturu**

Danymi wejściowymi dla algorytmu rozpoznawania elementarnych gestów jest sekwencja punktów, z jakich składa się kontur. Akwizycja tych danych odbywa się dzięki interakcji z systemem, na jakim pracuje urządzenie. Dane o położeniu palca lub rysika na ekranie dotykowym w systemie Android możliwe są do odczytania tylko co pewien, krótki okres czasu. Oznacza to, że gest wykonany przez użytkownika w sposób szybki może być odczytany z mniejszą dokładnością. Każdy gest poddawany jest aproksymacji (w mniejszym lub większym stopniu), a więc z ekranu dotykowego można odczytać zbiór punktów, który reprezentuje linię łamaną otwartą, gdzie odległość między sąsiednimi punktami nie musi być stała (i zwykle nie jest). Po sygnale oderwania palca od ekranu ze sterownika można stwierdzić, że rysowanie konturu zostało zakończone i można poddać go dalszej analizie.

### **3.4. Przetwarzanie wstępne**

#### **3.4.1. Interpolacyjne uzupełnienie listy punktów**

Zaraz po odczycie, kontur poddawany jest przetwarzaniu wstępnemu. Pierwszy krok polega na uzupełnieniu listy punktów otrzymanej ze sterownika ekranu o brakujące piksele tak, aby lista zawierała tyle pikseli, z ilu faktycznie składa się kontur na ekranie, i aby odległość między sąsiednimi punktami była stała (równa jednemu pikselowi). Liczba uzupełnianych pikseli jest proporcjonalna do odległości między punktami w metryce euklidesowej dla każdej pary sąsiednich punktów. Przekształcenie to odwraca proces aproksymacji konturu i ułatwia dalszą analizę gestu. Do wyznaczenia współrzędnych punktów pomiędzy sąsiednimi pikselami wykorzystywana jest interpolacja liniowa współrzędnych punktów będących najbliższymi sąsiadami. Należy zaznaczyć, że współrzędne interpolowanych pikseli zapisywane są jako liczby zmiennoprzecinkowe a nie jako całkowite, aby przechowywać położenie punktów z większą dokładnością.

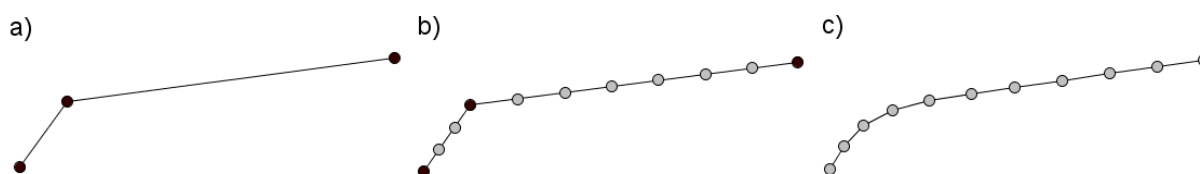
#### **3.4.2. Filtracja**

Kolejnym etapem przetwarzania wstępnego jest filtracja konturu otrzymanego z poprzedniego kroku. Filtracja odbywa się poprzez uśrednianie współrzędnych punktów

konturu. Współrzędne te zastępowane są nowymi współrzędnymi, które obliczane są jako średnia arytmetyczna współrzędnych jego  $n$  najbliższych sąsiadów ze starej listy punktów. Oczywiście w wyniku takiego uśrednienia liczba punktów konturu zmniejszy się o wartość  $n - 1$ .

Filtracja ma na celu wygładzenie konturu oraz redukcję szumów (drgań rysika podczas rysowania oraz kompensacji błędów niedokładności czujnika dotykowego). Przeprowadzone doświadczenia pokazały, że zwiększa to skuteczność w późniejszym wyznaczaniu wektorów kierunków między pikselami i wpływa na wyraźniejszy rozkład łańcuchów Freemana na histogramie. Zmniejsza się liczba wystąpień błędnie rozpoznanych kierunków, które wynikają z analizy położenia jedynie dwóch sąsiednich punktów, znajdujących się bardzo blisko siebie. Liczba sąsiadujących pikseli, z jakich wyznaczany jest nowy uśredniany punkt wyznaczona została eksperymentalnie i może zależeć od rozdzielczości ekranu oraz od gęstości pikseli na ekranie (liczby DPI), w omawianej aplikacji liczba  $n$  najbliższych sąsiadów do uśredniania wynosi 20.

Poniżej przedstawiono ideę przetwarzania wstępnego na przykładzie prostego konturu oraz efekty poszczególnych etapów. Kropki na rysunku reprezentują punkty przechowywane w liście.



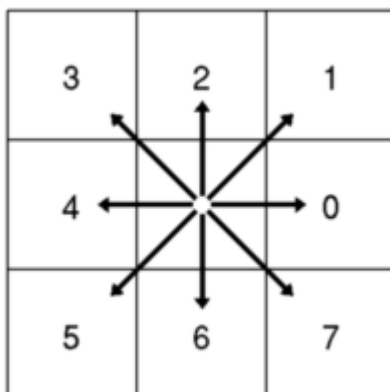
**Rysunek 3.1. Przykładowy kontur poddany przetwarzaniu wstępnemu: a) punkty odczytane ze sterownika ekranu dotykowego przed przetwarzaniem wstępnym, b) punkty uzyskane w wyniku uzupełnienia i interpolacji, c) lista uśrednionych punktów po etapie filtracji**

Dzięki takiemu podejściu z konturu wydobywane są jego ogólne cechy i przybliżony kształt, zaś szczegółowe cechy geometryczne mogą zostać pominięte na skutek filtracji.

### 3.5. Ekstrakcja cech

Do wyodrębnienia cech charakterystycznych obiektu wykorzystano kody łańcuchowe Freemana. W skrócie polega to na tym, że dla każdej pary sąsiednich pikseli z konturu obliczany jest wektor przemieszczenia (różnica współrzędnych) i na tej podstawie wyznaczany jest kąt, jaki tworzy powstały odcinek z niezmienną osią poziomą ekranu (za pomocą funkcji *arcus tangens*). Następnie wyznaczany jest jeden z 8 kierunków, do których przynależy taki

kąt. Szerokość jednego przedziału kątów dla każdego kierunku jest taka sama i jest równa ilorazowi kąta pełnego i liczby wszystkich przedziałów kierunków.



**Rysunek 3.2. Podział kąta pełnego na 8 przedziałów kierunków do zapisu kodów łańcuchowych Freemana**

Do zapisania całego konturu w postaci kodów łańcuchowych Freemana potrzebna jest lista liczb całkowitych, z których każda reprezentuje kierunek zmiany współrzędnej punktu w stosunku do poprzedniego piksela i wyraża się liczbą z zakresu od 0 do 7.

Taki sposób reprezentacji ma pożądaną cechę - częściową niewrażliwość na obrót. W niektórych sytuacjach obrót całego wejściowego konturu nawet do 45 stopni może skutkować zapisaniem dokładnie tymi samymi wartościami kodów łańcuchowych (np. w przypadku kształtu linii prostych). Wprowadza to także błąd kwantyzacji, ale jego znaczenie zanika dzięki występowaniu wielu pikseli w konturze. Warto zauważyć również, że taki zapis w postaci kodów łańcuchowych jest silnie związany z kierunkiem wykonywania gestu, co jest istotną zaletą, gdyż dzięki temu można łatwo odróżnić wiele gestów, np. pionową linię pisaną w dół lub w górę.

### **3.6. Reprezentacja obiektu**

Kolejnym etap jest wyznaczenie z listy kodów łańcuchowych Freemana histogramu liczby ich występowania. Polega to oczywiście na zliczeniu liczby wystąpień każdego takiego kodu, a następnie podzieleniu tej wartości przez liczbę wszystkich kodów występujących w konturze w celu normalizacji histogramu.

Taka reprezentacja obiektu jest bardzo wygodna w dalszej analizie (porównywaniu histogramów). Warto również zauważyć, że takie przedstawienie obiektu ma właściwość całkowitej niewrażliwości na przesunięcia (histogramy kodów łańcuchowych będą dokładnie takie same w wyniku dowolnego przesunięcia konturu) oraz całkowitej niewrażliwości na

skalowanie wejściowego konturu (z powodu normalizacji histogramu rozkład procentowy każdego z kierunków będzie również taki sam).

Ze względu na założenie częściowej a nie całkowitej niewrażliwości na przesunięcie oraz skalowanie w procesie rozpoznawania gestów elementarnych, konieczne jest przechowywanie dodatkowych informacji o obiekcie, które mogą być wykorzystane podczas wyznaczania podobieństwa obiektów, a które ograniczą w pewnym stopniu całkowitą niewrażliwość konturu na wybrane przekształcenia geometryczne. Do tego celu razem z histogramem kodów łańcuchowych wyrażającym sam kształt gestu elementarnego dołączana jest również informacja o współrzędnych punktu startowego dla konturu oraz jego długości (wyrażającej wielkość gestu).

### **3.7. Klasyfikacja**

Aby dokonać identyfikacji nieznanego gestu elementarnego, należy wyznaczyć współczynnik stwierdzający stopień podobieństwa gestów (korelację). W przypadku, gdy istnieje wiele cech obiektów, rozwiązaniem może być wyznaczenie współczynników podobieństwa dla każdej cechy, a następnie obliczenie sumarycznego współczynnika korelacji.

#### **3.7.1. Korelacja histogramów**

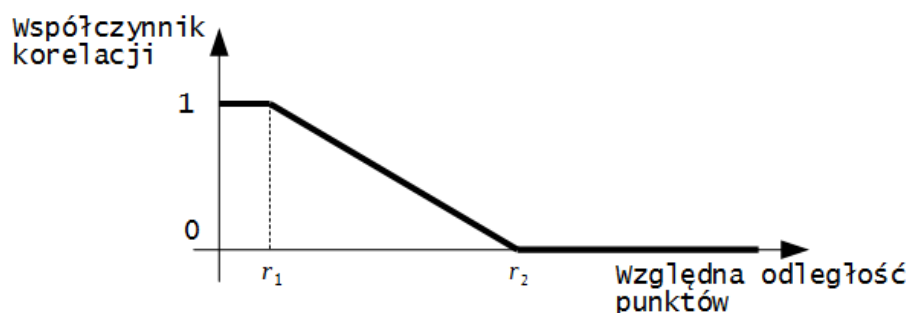
Algorytm identyfikacji pojedynczych gestów wymaga metody porównywania wprowadzonych konturów z przechowywanymi w wewnętrznej bazie wzorcami w celu wyznaczenia współczynnika podobieństwa każdego z nich i wybrania najlepszego dopasowania. Opisywany współczynnik to współczynnik korelacji liniowej Pearsona. Większa jego wartość oznacza większy stopień podobieństwa porównywanych ciągów liczb. Danymi wejściowymi dla korelacji Pearsona są histogramy kodów łańcuchowych określających kształt gestów. Dzięki temu wynikiem takiego zestawienia histogramów jest liczba określająca jak bardzo kształty gestów są do siebie zbliżone.

Wyznaczenie współczynnika korelacji Pearsona odbywa się poprzez wykorzystanie funkcji z biblioteki OpenCV. Obliczanie trwa bardzo szybko z powodu między innymi bardzo małej ilości danych do przeanalizowania (porównanie 2 tablic o rozmiarze 8) oraz zoptymalizowanego algorytmu zaimplementowanego w bibliotece OpenCV.

#### **3.7.2. Korelacja punktów startowych**

Aby stwierdzić podobieństwo dotyczące punktów startowych dla konturów, obliczana jest odległość między punktami, następnie dzielona jest przez mniejszy wymiar ekranu (liczbę

pikseli szerokości lub wysokości ekranu) tak, aby uzyskać wielkość niezależną od wielkości ekranu urządzenia, na jakim uruchomiona jest aplikacja. Tak uzyskana liczba nazwana jest tutaj względną odległością punktów. Wyznaczenie współczynnika korelacji punktów startowych zależne jest również od 2 parametrów wybranych doświadczalnie. Są to współczynniki  $r_1$  oraz  $r_2$ . Wartość współczynnika  $r_1$  reprezentuje promień odległości obszaru przy punkcie startowym, w którym tolerowane jest tak samo każde przesunięcie punktów między sobą i nie wpływa ono na współczynnik korelacji punktów startowych (uznaje się, że dla małych rozbieżności rozpoczęcie rysowania gestów odbyło się w tym samym miejscu). Natomiast współczynnik  $r_2$  określa maksymalny promień obszaru, w którym współczynnik korelacji jest niezerowy. Poza tym obszarem uznaje się, że nie występuje jakiegokolwiek podobieństwo dotyczące miejsca rozpoczęcia rysowania. W zakresie względnej odległości punktów między  $r_1$  a  $r_2$  wartość współczynnika korelacji interpolowana jest w sposób liniowy, co przedstawia się na poniższym wykresie.



**Rysunek 3.3.** Wykres zależności współczynnika korelacji punktów startowych od wartości względnej odległości punktów

Wartości parametrów  $r_1$  i  $r_2$  zostały wybrane na drodze doświadczalnej i wynoszą odpowiednio:  $r_1 = 0,2$  oraz  $r_2 = 1,5$ .

### 3.7.3. Korelacja długości gestu

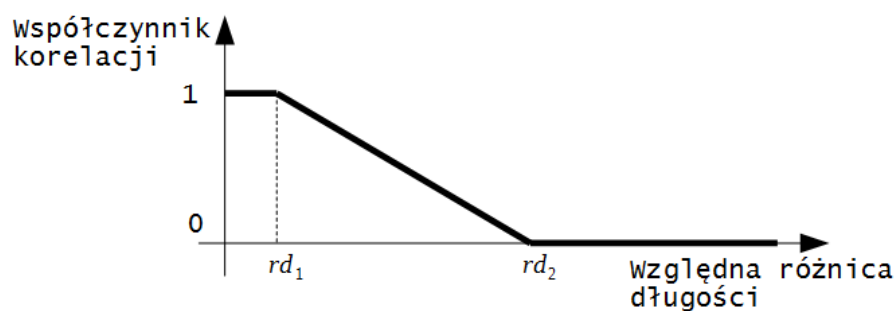
Do wyznaczenia podobieństwa gestów dotyczącego ich wielkości posłużono się współczynnikiem korelacji długości. Wprowadźmy pojęcie względnej różnicy długości konturów, która będzie wyrażać się wzorem:

$$rd = \frac{l_{max} - l_{min}}{l_{min}} \quad (3.1)$$

gdzie:  $rd$  - względna różnica długości gestów,  $l_{max}$  - większa wartość długości konturu spośród porównywanych konturów,  $l_{min}$  - mniejsza wartość długości konturu spośród porównywanych konturów.



Podobnie jak w poprzednim przypadku, współczynnik korelacji długości zależy od dwóch parametrów wybranych doświadczalnie. Są to współczynniki  $rd_1$  oraz  $rd_2$ . Wartość współczynnika  $rd_1$  określa martwą strefę niewrażliwości na skalowanie (dla  $rd \leq rd_1$  algorytm uznaje, że gesty są takiej samej wielkości). Zaś współczynnik  $rd_2$  określa górną granicę wrażliwości na skalowanie (dla  $rd \geq rd_2$  składowe gesty nie wykazują żadnego podobieństwa, jeśli chodzi o wspólną wielkość). W zakresie względnej różnicy długości między  $rd_1$  a  $rd_2$  wartość współczynnika korelacji interpolowana jest w sposób liniowy, przebieg jego funkcji w całym zakresie względnej różnicy długości przedstawiono na poniższym wykresie:



**Rysunek 3.4.** Wykres zależności współczynnika korelacji długości od wartości względnej różnicy długości

Wartości parametrów  $rd_1$  i  $rd_2$  zostały wybrane na drodze doświadczalnej i wynoszą odpowiednio:  $rd_1 = 0,5$  oraz  $rd_2 = 2,5$ .

#### 3.7.4. Sumaryczny współczynnik korelacji

Wprowadźmy pojęcie sumarycznego współczynnika korelacji. Niech będzie to liczba wyznaczana na podstawie współczynników podobieństwa dla każdej cechy obiektu i określająca wypadkową wartość podobieństwa. Taki współczynnik mógłby być prostym iloczynem wszystkich wartości korelacji dla każdej z cech, lecz przy takim podejściu sumaryczny wynik byłby silnie zależny od każdej z cech, np. gdy jeden ze współczynników składowych byłby równy 0, stwierdzony mógłby zostać zupełny brak podobieństwa obiektów.

Przeprowadzone testy pokazały, że iloczyn współczynników korelacji daje stosunkowo słabe rezultaty, a lepsze efekty można osiągnąć, korzystając ze średniej ważonej. W opisywanej aplikacji wartość sumarycznego współczynnika korelacji wyraża się wzorem na średnią ważoną:

$$c = \frac{c_1 \cdot w_1 + c_2 \cdot w_2 + c_3 \cdot w_3}{w_1 + w_2 + w_3} \quad (3.2)$$

gdzie:  $c_1$  - współczynnik korelacji histogramów kodów łańcuchowych Freemana,  $c_2$  - współczynnik korelacji punktów startowych,  $c_3$  - współczynnik korelacji długości,  $w_1, w_2, w_3$  - wagi dla odpowiednich współczynników  $c_1, c_2, c_3$ .

Wartości wag zostały wyznaczone na drodze eksperymentalnej i wynoszą odpowiednio:

$$w_1 = 0,668, w_2 = 0,166, w_3 = 0,166$$

W algorytmie zastosowany został również minimalny współczynnik korelacji dla pojedynczego gestu, wyrażający wartość współczynnika korelacji, poniżej której nie stwierdza się żadnego podobieństwa między elementarnymi gestami. Wynosi on  $c_{min} = 0,85$ .

### 3.8. Przypadek krótkich konturów

Istnieje przypadek typu gestu elementarnego wykonywanego przez użytkownika, dla którego powinno się zdefiniować inne, wyjątkowe zachowanie algorytmu, gdyż wyżej opisany algorytm może nie sprawdzić się dla tej sytuacji. Jest to przypadek wprowadzania przez użytkownika kropki - pojedynczego kliknięcia na ekranie dotykowym. Sytuacja ta może mieć miejsce np. przy wprowadzaniu liter "i", "j", "z", znaków interpunkcyjnych kropki, dwukropka lub znaku zapytania.

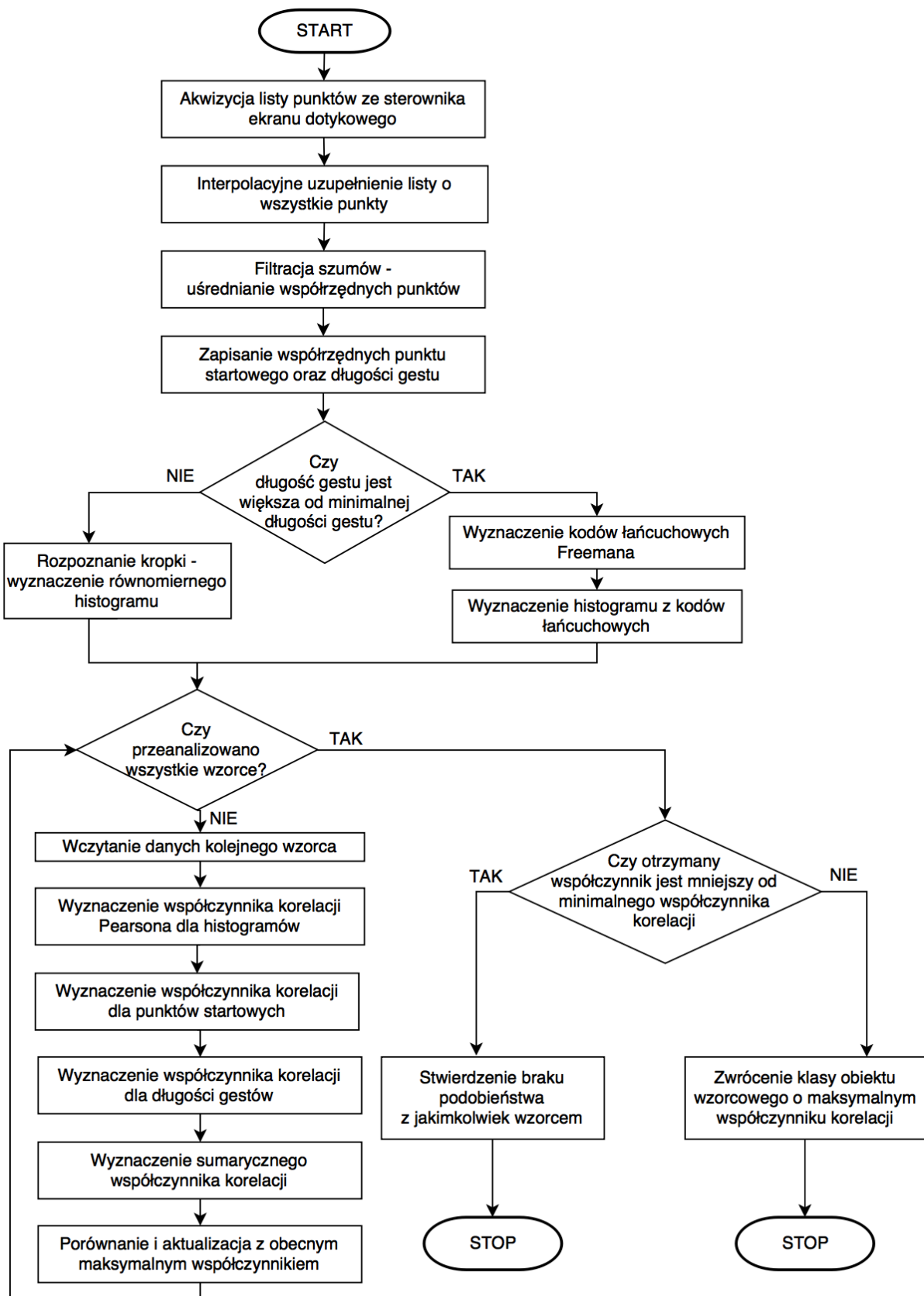
Wyżej opisany algorytm może niepoprawnie rozpoznawać kropki, gdyż wyznaczenie kodów łańcuchowych Freemana na podstawie jednego piksela może być niemożliwe, lub w przypadku bardzo małej liczby punktów może zwracać losowe kierunki między sąsiednimi punktami, tym samym zwracając prawie losowy rozkład na histogramie. Poza tym filtracja w trakcie przetwarzania wstępnego może być niemożliwa z powodu zbyt małej liczby pikseli uniemożliwiającej uśrednianie punktów.

Dlatego też w przypadku wpisania kropki algorytm zachowuje się nieco inaczej. Wprowadzenie kropki jest bardzo łatwe do wykrycia dzięki informacji o długości konturu. Jeśli liczba pikseli, z jakich składa się kontur jest mniejsza lub równa pewnej liczbie granicznej (w aplikacji wynosi ona 20), generowany jest histogram o rozkładzie równomiernym. Również w przypadku korelacji długości, gdy dwa kontury mają długość mniejszą lub równą liczbie granicznej, automatycznie rozpoznawane są jako gest kropki a korelacja ich histogramów oraz korelacji długości wynosi 1 (sposób obliczania korelacji punktów startowych pozostaje bez zmian). Takie rozwiązanie znacznie poprawia skuteczność w rozpoznawaniu tego typu gestów.

### **3.9. Schemat algorytmu**

Należy pamiętać, że opisywany algorytm stosowany może być jedynie do rozpoznawania podstawowego gestu, kiedy w bazie aplikacji znajdują się już zapamiętane wzorce. W kolejnych rozdziałach opisywana jest metoda, która jest modyfikacją tego algorytmu i rozszerza go na rozpoznawanie gestów złożonych (składających się z wielu elementarnych gestów), również zmieniając strukturę bazy wzorców.

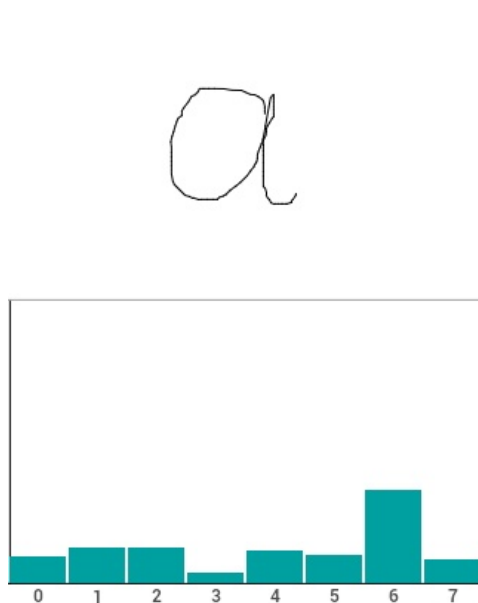
Podsumowaniem zasady działania algorytmu rozpoznawania pojedynczych gestów jest schemat blokowy przedstawiony poniżej.



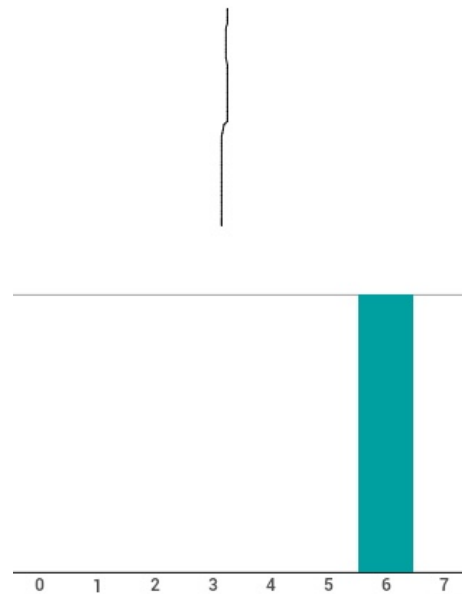
Rysunek 3.5. Schemat blokowy algorytmu rozpoznawania pojedynczych gestów

### 3.10. Przykłady

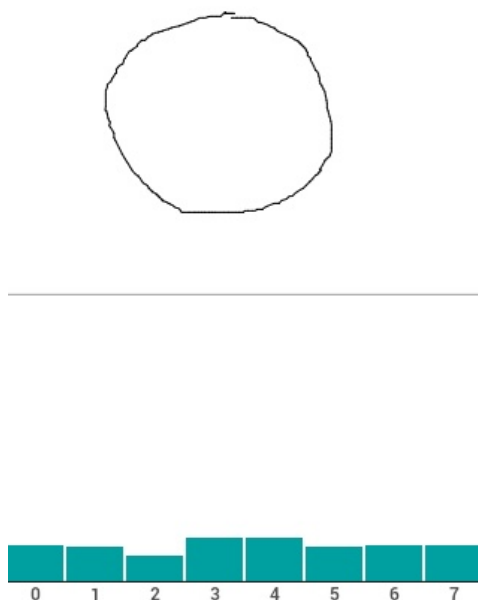
Aby zilustrować użytą metodę reprezentacji cech geometrycznych obiektu w postaci histogramu kodów łańcuchowych Freemana, poniżej przedstawiono wykresy histogramów uzyskanych dla przykładowych konturów. Numery przy słupkach histogramu oznaczają indeks kierunku, natomiast na osi pionowej zaznaczona jest częstość występowania.



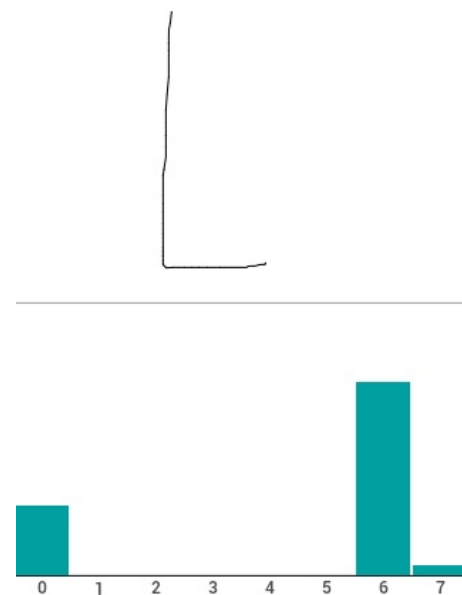
Rysunek 3.6. Histogram dla pojedynczego gestu litery "a"



Rysunek 3.7. Histogram dla gestu litery "I" (według histogramu kontur składa się wyłącznie z wektorów skierowanych pionowo w dół - kierunek o indeksie 6)

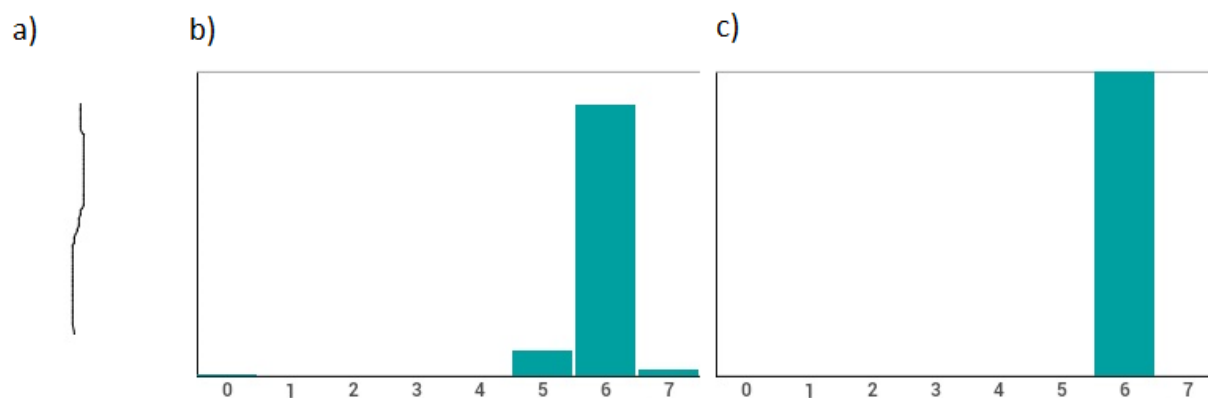


Rysunek 3.8. Histogram dla gestu składowego litery "o" (w przypadku idealnego okręgu rozkład kierunków na histogramie byłby równomierny)



Rysunek 3.9. Histogram dla gestu litery "L", na histogramie widoczne jest występowanie wektorów skierowanych w dół (6) oraz w prawo (0)

Aby pokazać jak istotną funkcję pełni filtracja w procesie przetwarzania wstępnego, poniżej przedstawiono dwa histogramy wygenerowane na podstawie tego samego konturu, lecz w jednym z nich nie zastosowano filtracji. Łatwo można zauważyć, że histogram utworzony z konturu niepoddanego filtracji ma większą wrażliwość na zakłócenia.



**Rysunek 3.10. Zestawienie kształtu konturu (a), histogramu utworzonego z konturu niepoddanego filtracji (b), histogramu utworzonego z wykorzystaniem filtracji (c)**

## **4. System rozpoznawania złożonych gestów**

Rozdział ten skupia się na opisie autorskiej metody rozpoznawania złożonych gestów, które składają się z wielu pojedynczych fragmentów.

### **4.1. Opis algorytmu**

Stworzony algorytm jest modyfikacją metody rozpoznawania pojedynczych konturów z poprzedniego rozdziału. Tu również wykorzystuje się współczynnik korelacji w celu porównywania gestów, natomiast wprowadzone kontury nie są porównywane ze wszystkimi wzorcami, gdyż nie ma takiej potrzeby. Identyfikacja nie polega jedynie na znalezieniu obiektu z maksymalnym współczynnikiem korelacji (tak jak to miało miejsce w poprzednim przypadku), lecz wiąże się to z głębszą analizą wzorców (będących kombinacją pojedynczych konturów o różnej liczbie gestów składowych) oraz z koniecznością rozważania wielu możliwości gestów spełniających ustalone kryteria, jak również odrzucania ich na bieżąco w wyniku wprowadzenia kolejnych gestów. W przypadku, gdy zostanie wiele takich możliwości, algorytm musi podjąć decyzję, które z tych rozwiązań będzie najlepsze. Kolejność wprowadzania konturów przez użytkownika jest bardzo istotna dla algorytmu i powinna być dokładnie taka sama, jak w zapamiętanym wzorcu.

Struktura bazy wzorców również musi być inna w przypadku identyfikacji gestów złożonych. Wiąże się to z koniecznością grupowania wielu konturów i przechowywania listy wielu gestów elementarnych przyporządkowanych do każdego wzorca oraz zapisania informacji o przynależności danego obiektu do klasy. W przypadku rozpoznawania pisma odręcznego sprowadza się to do przechowywania znaku, jaki odpowiada wzorcowi.

#### **4.1.1. Kryteria wyboru**

Przy wyborze najbardziej zbliżonego wzorca algorytm kieruje się następującymi kryteriami:

1) Próg współczynnika korelacji - Wszystkie pojedyncze gesty pochodzące z wzorca w porównaniu z odpowiadającymi im konturami wejściowych powinny spełniać warunek współczynnika korelacji większego lub równego minimalnemu współczynnikowi korelacji. Jest to warunek konieczny - niespełnienie tego założenia dla przynajmniej jednego gestu elementarnego skutkuje natychmiastowym odrzuceniem wzorca i zaprzestaniem analizowania jego kolejnych elementów.

2) Największy wypadkowy współczynnik korelacji oraz największa złożoność gestu - Najlepsze rozwiązanie spośród wszystkich rozważanych możliwości wybierane jest przy pomocy dwóch kryteriów jednocześnie. Pod uwagę brany jest jak największy współczynnik korelacji dla całego gestu (wypadkowy), jak i liczba konturów wchodzących w skład gestu. Gdyby program nie stosował kryterium największej złożoności gestu, mogłoby się okazać, że np. po wpisaniu litery "Ł" (jako dwa kontury składowe: litery "L" i kreski), aplikacja rozpoznałaby samą literę "L", nie czekając na dalsze kontury oraz zwróciła błąd nierozpoznania kreski. Dlatego też należy w takich przypadkach zwiększyć wpływ złożoności na wynik identyfikacji dla wieloelementowych gestów. Oczywiście jest również, że nie powinno się wybierać jedynie gestów z największą złożonością, ale rozważać najlepiej pasujące wzorce.

#### **4.1.2. Dane wejściowe**

Uruchomienie algorytmu odbywa się po każdorazowym wprowadzeniu gestu elementarnego przez użytkownika. W pierwszym kroku należy pobrać listę konturów wejściowych, które jeszcze nie zostały rozpoznane. Wiąże się to z zapamiętywaniem historii wejściowych gestów elementarnych wraz z informacją o dokonaniu ich przeanalizowania lub nie. Takie kontury wraz z zestawem wzorców są danymi wejściowymi dla algorytmu.

Jeśli lista konturów wejściowych jest pusta, algorytm kończy działanie nie zwracając żadnego wyniku. Jeżeli natomiast lista wzorców jest pusta, zwracany jest błąd nierozpoznania żadnego gestu. Do dalszej analizy wymagany jest zatem niepusty zbiór wzorców i lista wejściowych konturów.

#### **4.1.3. Wstępna klasyfikacja**

W kolejnym kroku badane jest podobieństwo konturów do każdego z wzorców bazy aplikacji w celu wstępnej klasyfikacji. Sprawdzane jest, czy spełnione jest kryterium przekroczenia progu minimalnego współczynnika korelacji (kryterium nr 1). Współczynnik ten jest wyznaczany metodą opisaną w poprzednim rozdziale o wyznaczaniu podobieństwa pojedynczych gestów. Odbywa się to poprzez porównanie pierwszego konturu pochodzącego z wzorca oraz odpowiadającemu mu konturu wejściowego (wpisanego przez użytkownika). Jeśli kryterium nie jest spełnione, wzorzec jest odrzucany i analizowany jest kolejny. Jeżeli natomiast zachodzi warunek współczynnika korelacji większego lub równego ustalonemu minimum, analizowany jest kolejny kontur z gestu (o kolejnym indeksie) w ten sam sposób. Jeżeli okaże się, że nie można dokonać dalszej analizy, gdyż wprowadzona liczba konturów jest mniejsza niż złożoność wzorca a istnieje możliwość, że właśnie ten wzorzec może posłużyć



do identyfikacji obiektu, to algorytm przerywa działanie, gdyż nie ma wystarczających danych, aby podjąć decyzję. Algorytm wykona się ponownie w następnym kroku, gdy użytkownik wprowadzi kolejny kontur, wtedy analiza zostanie wznowiona. Wyjątkowym przypadkiem jest jednak przekroczenie czasu przez użytkownika pomiędzy wpisywaniem kolejnych konturów (wpisanie znaku "L" i odczekanie przez użytkownika pewnego czasu nie wstrzyma pracy algorytmu, czekając na brakującą kreskę i rozpoznanie litery "Ł", ale pozwoli mu kontynuować jego działanie i odrzucenie możliwości dłuższych gestów). Jeśli wszystkie składowe elementy gestu spełniają kryterium nr 1, wzorzec dopisywany jest do tymczasowej listy możliwości.

#### **4.1.4. Wypadkowy współczynnik korelacji**

Wprowadźmy pojęcie wypadkowego współczynnika korelacji. Niech będzie to liczba wyznaczana z wielu współczynników korelacji, które zostały wyznaczone dla każdego konturu (jest ich tyle, ile wynosiła złożoność gestu) i określająca stopień podobieństwa całego gestu złożonego. Niech wartość tego współczynnika będzie równa średniej arytmetycznej wszystkich współczynników korelacji konturów, z których składał się gest. Liczba ta posłuży w kolejnym kroku do wyboru najlepszego rozwiązania.

#### **4.1.5. Wybór najlepszego rozwiązania**

Efektem zakończonej wstępnej klasyfikacji dla wszystkich wzorców jest lista potencjalnych możliwości zawierająca informacje o wzorcach i przebiegu ich wstępnej klasyfikacji. Jeśli ta lista jest pusta, algorytm zwraca błąd nierozpoznania żadnego gestu oraz zapisuje fakt dokonania analizy ostatnio wpisanego konturu (aby nie był analizowany następnym razem). Jeżeli zaś na liście możliwości znajdują się wzorce, można przystąpić do wyboru najlepszego wzorca. Polega to na prostym znalezieniu wzorca, dla którego liczba wyrażająca ocenę rozwiązania jest największa. Oczywiście konieczne jest zdefiniowanie, czym jest wprowadzona ocena rozwiązania. Jest to liczba, która wyznaczana jest na podstawie wypadkowego współczynnika korelacji oraz złożoności gestu. Ma to na celu jednoczesne rozważanie obu tych cech (zgodnie z kryterium nr 2). Ocena rozwiązania reprezentuje obie cechy w jednej liczbie (co bardzo ułatwia znalezienie jej maksimum) oraz pozwala wpłynąć na proces rozpoznawania poprzez dobór odpowiednich współczynników. Daje to możliwość ustalenia, w jakim stopniu mają być faworyzowane gesty o większej złożoności, a w jakim stopniu wpływ na wybór najlepszego rozwiązania ma mieć sam współczynnik korelacji.

Ocena potencjalnego rozwiązania dla wzorca wyraża się wzorem:

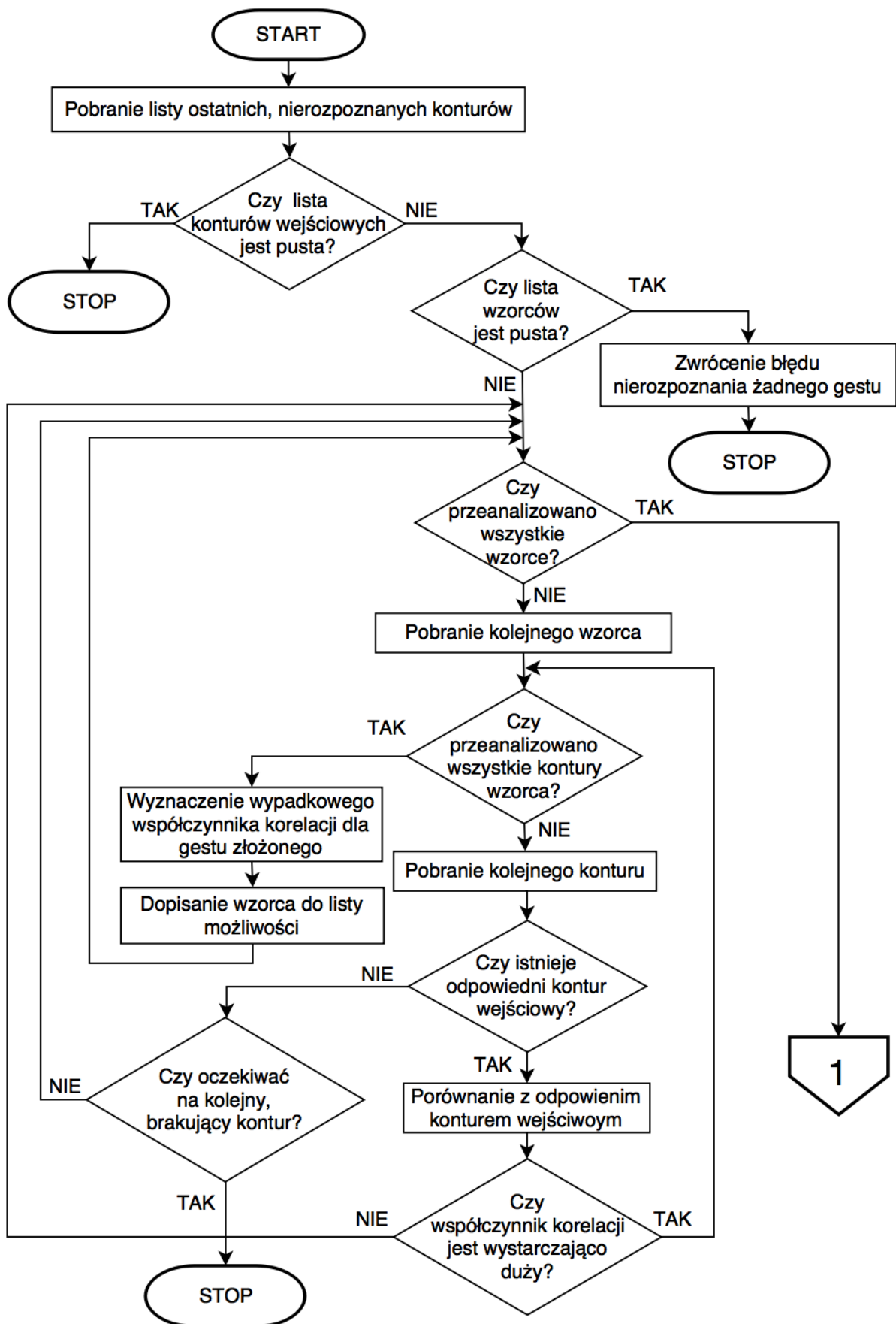
$$o = c \cdot (1 + z \cdot \alpha) \quad (4.1)$$

gdzie:  $o$  - ocena rozwiązania,  $c$  - wypadkowy współczynnik korelacji,  $z$  - złożoność gestu (liczba jego elementarnych konturów),  $\alpha$  - współczynnik dominacji złożoności (im większy jest współczynnik, tym większy wpływ na wybór rozwiązania ma złożoność gestu). Wartość współczynnika  $\alpha$  została dobrana na drodze empirycznej i wynosi  $\alpha = 0,067$ .

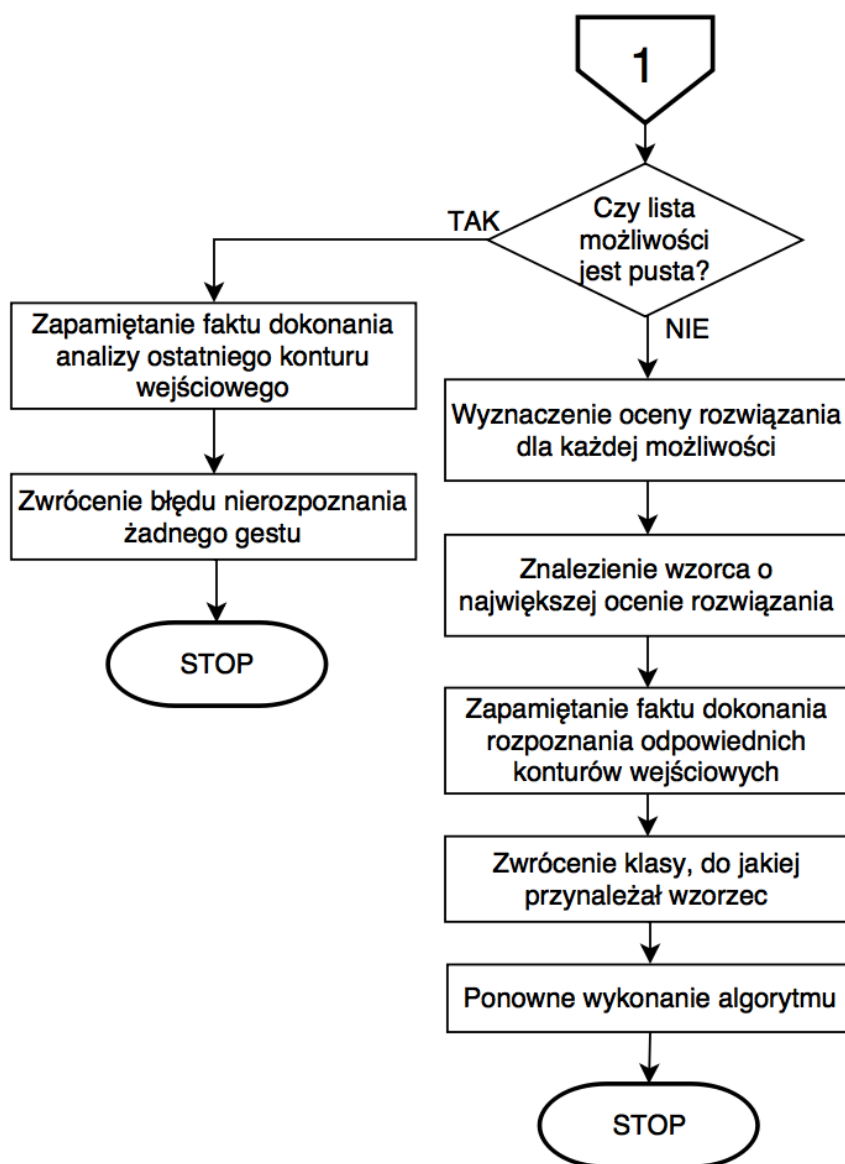
Po wyznaczeniu oceny rozwiązania dla każdej rozważanej możliwości wybierany jest wzorec o maksymalnej wartości tej oceny i zwracany jest jako wynik rozpoznania wprowadzonego gestu (a dokładniej znak, jaki reprezentował wzorec). Ostatnio wprowadzone gesty zostają zaznaczone jako przeanalizowane, aby nie zostały przetworzone przy następnej iteracji (liczba zaznaczonych gestów jest równa złożoności rozpoznanego wzorca). Wzorec zostaje dopisany do listy rozpoznanych gestów, a algorytm uruchamiany jest ponownie, gdyż na liście wejściowej mogą znajdować się jeszcze nierozpoznane gesty. Powtarzanie wykonywania trwa dopóki algorytm zwraca pomyślnie rozpoznane wzorce.

## 4.2. Schemat algorytmu

Poniższe schematy blokowe przedstawiają zasadę działania algorytmu rozpoznawania złożonych gestów.



Rysunek 4.1. Schemat blokowy algorytmu rozpoznawania złożonych gestów - część 1.  
(Wstępna klasyfikacja)




















Rysunek 4.2. Schemat blokowy algorytmu rozpoznawania złożonych gestów - część 2.  
(Wybór najlepszego rozwiązania)

### 4.3. Przykłady

Aby zilustrować działanie programu, przeanalizujemy pewien przypadek, kiedy w bazie aplikacji są już zdefiniowane pewne wzorce:

**Tabela 4.1. Zbiór przykładowych wzorców w bazie aplikacji**

<b>Wzorec - gest złożony</b>	<b>Gest składowy nr 1</b>	<b>Gest składowy nr 2</b>	<b>Gest składowy nr 3</b>
 mała litera "l" (złożoność 1)		Brak	Brak
 duża litera "I" (złożoność 1)		Brak	Brak
 mała litera "t" (złożoność 2)			Brak
 mała litera "i" (złożoność 2)			Brak
 mała litera "f" (złożoność 2)			Brak
 mała litera "k" (złożoność 3)			

Rozważmy zachowanie programu, gdy użytkownik wprowadza znak małej litery "l". Zakończenie rysowania konturu (oderwanie palca lub rysika od ekranu) powoduje wykonanie algorytmu, który jako dane wejściowe dostaje listę zawierającą jeden kontur. Program przegląda bazę wszystkich wzorców i wstępnie klasyfikuje je, porównując wprowadzony kontur z pierwszym gestem składowym. W wyniku takiego porównania otrzymany

współczynnik korelacji będzie najprawdopodobniej większy od minimalnego współczynnika dla wszystkich wzorców oprócz litery "f" (z powodu innego rozkładu kodów łańcuchowych na histogramie). Zatem ten wzorec zostanie natychmiast odrzucony. Z pozostałych pięciu wzorców, trzy z nich mają złożoność większą niż 1, lecz nie można dokonać porównania kolejnych gestów składowych (o indeksie 2), gdyż nie istnieje kolejny kontur wejściowy. Z tego powodu algorytm nie ma wystarczających danych, aby podjąć decyzję i kończy swoje działanie. Jeśli użytkownik nie wpisze kolejnych konturów i czas oczekiwania na kolejny kontur minie, algorytm zostanie wywołany ponownie. Tym razem nie będzie brał pod uwagę dłuższych wzorców i odrzuci trzy wzorce o złożoności większej niż 1 (litery "l", "i" oraz "k"), które wcześniej były brane pod uwagę. Tym samym do listy potencjalnych rozwiązań zostaną dodane 2 wzorce: mała litera "l" oraz duża litera "I". W kolejnym kroku zostanie wybrane najlepsze rozwiązanie z listy potencjalnych możliwości. Z racji tego, że oba wzorce mają jednakową złożoność, wybrany zostanie wzorec z większym współczynnikiem korelacji, którym najprawdopodobniej będzie mała litera "l". W ten sposób znak wpisany przez użytkownika zostanie poprawnie rozpoznany. Po pomyślnej identyfikacji algorytm uruchomi się ponownie w poszukiwaniu kolejnych gestów, lecz zostanie od razu zakończony, gdyż lista nierozpoznanych konturów będzie pusta.

Rozważmy teraz przypadek, kiedy użytkownik wprowadza do programu znak małej litery "l". Robi to poprzez wykreślenie dwóch konturów na ekranie. Wprowadzenie pierwszego konturu (podobnego do litery "l") skończy się brakiem podjęcia akcji, oczekując na więcej danych wejściowych (tak, jak to opisano rozważając poprzedni przypadek). Natomiast wprowadzenie kolejnego konturu przez użytkownika (brakującej kreski nad literą "l") skutkuje wznowieniem analizy wzorców i przeprowadzeniem kolejnego porównania gestów składowych nr 2 z drugim wprowadzonym konturem dla każdego rozważanego wzorca. Po takim porównaniu odrzucone zostaną wzorce małej litery "i" oraz litery "k" z powodu zbyt małego współczynnika korelacji. Zatem na liście potencjalnych rozwiązań znajdą się trzy wzorce: małej litery "l", dużej litery "I" oraz małej litery "l". Algorytm nie musi oczekiwać na kolejne kontury, gdyż nie rozważa już bardziej złożonych gestów. Następnie każde z tych rozwiązań zostanie poddane ocenie. W związku z wprowadzoną zasadą obliczania oceny rozwiązania, faworyzowane będą gesty z większą złożonością, a więc to litera "l" zostanie najprawdopodobniej rozpoznana (pod warunkiem występowania zbliżonych wartości współczynników korelacji dla innych wzorców).

## **5. Inteligencja systemu**

W tym rozdziale poruszone zostanie zagadnienie zdolności systemu do samouczenia się oraz dostosowywania się do zmiennych warunków. Do osiągnięcia takiego rezultatu zaproponowane zostały autorskie rozwiązania niektórych problemów. Stworzony system został zaprojektowany tak, aby podejmował interakcję z użytkownikiem w odpowiednich sytuacjach, oraz aby sam zmieniał swoje zachowanie na skutek poleceń wydawanych przez użytkownika.

Interfejs dotykowy został zaprojektowany tak, aby z początkowo pustą bazą wzorców w niedługim czasie użytkownika automatycznie uzupełnił ją wzorcami, dostosował się do charakteru pisma odręcznego użytkownika i uzyskał pełną funkcjonalność z zadowalającą skutecznością w rozpoznawaniu znaków.

System można nazwać inteligentnym również dlatego, że podczas identyfikacji gestów rozważa wiele możliwości, ocenia rozwiązania i wybiera to, które według niego jest najlepsze, a zatem w pewnym sensie dokonuje procesu wnioskowania.

### **5.1. Problem błędnych wzorców**

Zastosowana w rozpoznawaniu gestów metoda najbliższego sąsiada ma pewne wady. Jedną z nich jest wrażliwość na błędne dane w ciągu uczącym. Jeśli jeden obiekt zostanie źle zaklasyfikowany i zapamiętany jako wzorec, może to prowadzić do błędnego klasyfikowania całego jego otoczenia, przyczynić się do niepoprawnego podejmowania decyzji w pewnym obszarze przestrzeni cech. Jest to poważny problem, skutkiem zapamiętania błędnego wzorca może być znaczny spadek skuteczności rozpoznawania dla pewnych gestów. Trudne jest również zlokalizowanie takiej próbki i usunięcie przyczyny błędów.

Błędny wzorec może pojawić się w wyniku pomyłki użytkownika lub np. zbyt dużych zakłóceń i zniekształceń konturu. Inteligentny interfejs dotykowy będący tematem tej pracy podejmuje próby wykrywania takich wzorców i usuwania ich.

### **5.2. Rejestrowanie bilansu identyfikacji**

Aby móc stwierdzić, które wzorce w bazie aplikacji są najczęściej podstawą do identyfikacji innych gestów, a które prowadzą do błędnych decyzji, potrzebne są dodatkowe informacje. W tym celu dla każdego wzorca rejestrowane jest każde jego poprawne rozpoznanie (gdy w procesie rozpoznania wzorec ten został wybrany jako najlepsze rozwiązanie) oraz każde błędne rozpoznanie. Liczebności tych zdarzeń są przechowywane w dwóch liczbach całkowitych, co pozwala również na łatwe wyznaczenie sumy liczby

wszystkich rozpoznań oraz obliczenie tzw. bilansu identyfikacji, który wyraża się jako różnica liczby poprawnych oraz niepoprawnych rozpoznań. Ujemna wartość takiego bilansu pozwala wnioskować, że wzorzec może być nieprawidłowy i powinien zostać usunięty (gdyż częściej prowadzi do błędnych identyfikacji).

Stwierdzanie poprawnego lub błędnego rozpoznania dla wzorca odbywa się automatycznie. Każda identyfikacja wzorca zawsze początkowo uważana jest za poprawną, a liczba poprawnych rozpoznań zwiększana o 1, gdy tylko wzorzec stanie się podstawą do rozpoznania innego gestu. Użytkownik oprócz wpisywania znaków ma do dyspozycji również możliwość wykonania akcji poprzez naciśnięcie przycisków "Cofnij" lub "Popraw" (opis przycisków oraz funkcjonalności aplikacji znajduje się w kolejnych rozdziałach). Wykonanie jednej z tych akcji oznacza, że wzorzec posłużył do błędnej identyfikacji, a więc należy cofnąć początkowe, błędne założenie (liczbę poprawnych rozpoznań zmniejszyć o 1), zaś należy dokonać inkrementacji liczby błędnych rozpoznań. Takie podejście pozwala automatycznie aktualizować dodatkowe informacje o statystykach wzorców oraz stwierdzać, jaka jest przydatność danego wzorca.

### **5.3. Automatyczne usuwanie błędnych wzorców**

Usuwanie błędnych wzorców w aplikacji odbywa się automatycznie. Jako błędny wzorzec uznawany jest ten, dla którego wartość bilansu identyfikacji osiągnie pewien ustalony próg. Póg ten został wybrany doświadczalnie a jego wartość wynosi  $-3$ . W praktyce oznacza to, że z bazy aplikacji usuwane są te wzorce, których liczba błędnych rozpoznań jest większa o 3 od liczby poprawnych rozpoznań. Sprawdzenie takie odbywa się po każdym usunięciu znaku lub poprawieniu go przez użytkownika. Wymagane jest również przechowywanie informacji o tym, przez który wzorzec został rozpoznany każdy z wprowadzonych znaków (nie tylko ten ostatni znak).

### **5.4. Automatyczne dodawanie wzorców**

Aby system uczył się w trakcie działania, i aby zwiększał swoją skuteczność w rozpoznawaniu, zastosowano automatyczne dodawanie wzorców. Polega to na tym, że nowo wprowadzony gest, który został poprawnie rozpoznany, automatycznie staje się nowym wzorcem. Aby lista wzorców nie rozrastała się zbyt szybko i niepotrzebnie, proces automatycznego dodawania wzorców odbywa się tylko dla gestów, które zostały rozpoznane ze stosunkowo niskim współczynnikiem korelacji. Wymaga to zdefiniowania wartości progu decydującego o dodaniu nowego wzorca lub nie. Wartość tego progu wynosi w aplikacji 0,96.



Dzięki automatycznemu dodawaniu wzorców kolejne próby rozpoznania tych samych znaków będą jeszcze skuteczniejsze a cały ten proces jest ukryty dla użytkownika i odbywa się bez jego udziału.

Podjęcie przedstawione w poprzednim akapicie wymaga nieco innego zachowania aplikacji w przypadku poprawiania lub usuwania gestów przez użytkownika. Źródłem błędów może stać się niepoprawne rozpoznanie wzorca, a następnie dodanie kolejnego gestu (również błędnego) w wyniku automatycznego dodawania wzorców. Aby tego uniknąć, należy zapamiętywać informację o tym, które wzorce zostały dodane automatycznie i usuwać je natychmiastowo w przypadku poprawienia lub usunięcia wpisanego gestu przez użytkownika.

### **5.5. Optymalizator liczby wzorców**

Wprowadzenie automatycznego dodawania wszystkich wzorców mogłoby wpłynąć na znaczny rozrost bazy wzorców a w efekcie przyczynić się do spadku wydajności całej aplikacji z powodu konieczności wykonywania większej liczby obliczeń oraz przechowywania większej liczby danych.

Aby temu zapobiec wprowadzono mechanizm, który czuwa nad ograniczaniem liczebności wzorców dla każdej klasy obiektów. W przypadku rozpoznawania znaków, klasami są poszczególne znaki, natomiast obiektami są przechowywane wzorce oraz wprowadzane gesty. Do każdej klasy przynależeć może wiele obiektów (każdy znak może posiadać wiele wzorców). Liczba wzorców przynależących do jednego znaku ograniczona jest maksymalną wartością, która wynosi 25. W takiej sytuacji pojawia się problem podjęcia decyzji, które wzorce należy usuwać, a które pozostawiać w bazie. Otóż usuwane są te nadmiarowe wzorce, które mają najmniejszy bilans identyfikacji. W drugiej kolejności spośród tych wzorców, które miały taki sam bilans identyfikacji, pozostawiane są wzorce nowsze (można to stwierdzić, gdyż zapamiętywany jest czas dodania wzorca).

Efektom takiego podejścia jest usuwanie nadmiarowych obiektów ciągu uczącego, a więc wzorców, które są rzadko podstawą do poprawnego rozpoznawania gestu, lub które stają się przyczyną błędnych rozpoznań. Wprowadzony mechanizm optymalizacji liczby wzorców zawsze pozwala na dodanie kolejnych wzorców w trakcie działania aplikacji (nawet, gdy przekroczy to maksymalną liczbę wzorców dla klasy). Natomiast samo usuwanie nadmiarowych obiektów odbywa się dopiero przy wyjściu z aplikacji, kiedy to zapisywana jest aktualna lista wzorców. Pozostawianie w bazie próbek najlepszych (lub najnowszych) sprawia, że system uczy się na bieżąco i dostosowuje się do aktualnego pisma użytkownika. Nawet w

przypadku zmiany użytkownika korzystającego z aplikacji interfejs dotykowy powinien po pewnym czasie przestawić się na nowy charakter pisma.

## **5.6. Interakcja z użytkownikiem**

Użytkownik oprócz możliwości wprowadzania konturów na ekranie ma do dyspozycji dwa przyciski pozwalające wydawać pewnego rodzaju polecenia uczące dla aplikacji. Są to przyciski "Cofnij" oraz "Popraw". Przycisk "Cofnij" usuwa ostatnio rozpoznany gest i zapisuje fakt błędnego rozpoznania gestu w odpowiednim wzorcu, ale nie dodaje nowego wzorca do bazy. Zaś przycisk "Popraw" zastępuje ostatnio rozpoznany znak nowym i dodaje nowy wzorec odpowiadający podanemu znakowi (wymaga to wyświetlenia użytkownikowi odpowiedniego ekranu).

W przypadku, gdy algorytm rozpoznawania złożonych gestów nie rozpozna żadnego obiektu, system podejmie interakcję z użytkownikiem i wyświetli ekran z pytaniem o to, jaki gest został ostatnio wpisany (jaki znak reprezentował i jaką miał złożoność). Podanie i zatwierdzenie danych skutkuje zapamiętaniem odpowiedniej liczby wprowadzonych konturów, przypisaniem im etykiety znaku i dopisaniem kolejnego wzorca do wewnętrznej bazy aplikacji. Można powiedzieć, że w ten sposób system uczy się poprzez zadawanie pytań użytkownikowi w sytuacjach, w których jest jeszcze niezdolny do samodzielnego podjęcia decyzji.

Wygląd ekranów widocznych dla użytkownika zostanie przedstawiony w kolejnych rozdziałach.

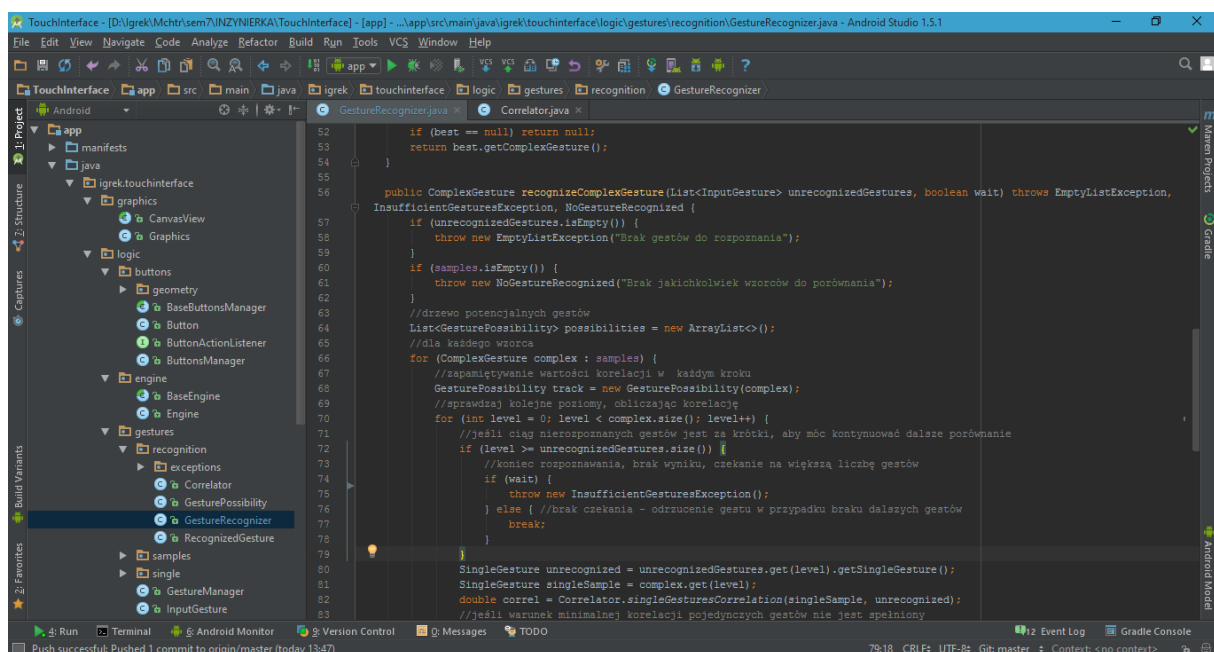
## 6. Implementacja oprogramowania na urządzenie mobilne

Podczas implementacji oprogramowania zastosowano przedstawioną w poprzednich rozdziałach metodę rozpoznawania gestów złożonych oraz wprowadzono opisane wcześniej inteligentne zachowania systemu.

### 6.1. Urządzenie i środowisko programowania

Urządzeniem, na którym została zainstalowana aplikacja jest smartphone Samsung Galaxy Grand Prime (model G531F). Posiada ekran dotykowy z wyświetlaczem o rozdzielczości  $540\text{ px} \times 960\text{ px}$  z zagęszczeniem pikseli równym 220 PPI. Urządzenie mobilne posiada 1 GB pamięci RAM, natomiast jednostką obliczeniową jest czterordzeniowy procesor z częstotliwością taktowania 1,2 GHz. Taki sprzęt pozwala na wykonywanie dużej liczby obliczeń w krótkim czasie. Systemem operacyjnym, na którym pracuje urządzenie jest system Android w wersji 5.1.1.

Aplikacja została napisana w obiektowym języku programowania Java na platformę Android. Do tego celu posłużono się darmowym środowiskiem programistycznym Android Studio w wersji 1.5.1. Środowisko to pozwala na testowanie aplikacji na wirtualnym urządzeniu z systemem Android oraz na instalację i uruchamianie aplikacji na zewnętrznym urządzeniu podłączonym do komputera przez kabel USB. Przykładowy zrzut ekranu środowiska programistycznego podczas pracy nad aplikacją znajduje się poniżej.



Rysunek 6.1. Zrzut ekranu środowiska programistycznego Android Studio

## 6.2. Struktura aplikacji

W strukturze powstałej aplikacji można wydzielić pewne moduły odpowiedzialne za poszczególne elementy aplikacji, które z kolei zawierają w sobie klasy lub dzielą się na mniejsze podmoduły. Poniżej przedstawiono hierarchiczną strukturę najważniejszych modułów (paczek Javy) oraz **klas** w aplikacji razem z opisem ich odpowiedzialności za poszczególne elementy.

- moduł logiki aplikacji:
  - moduł gestów:
    - moduł rozpoznawania:
      - **GestureRecognizer** - jest odpowiedzialny za zarządzanie procesem rozpoznawania gestów, uruchamianie algorytmu rozpoznawania złożonych gestów oraz interpretację zwracanych przez niego wartości;
      - **Correlator** - wyznacza poszczególne współczynniki korelacji dla pojedynczych gestów;
      - **GesturePossibility** - klasa reprezentująca tymczasowe potencjalne rozwiązanie w procesie identyfikacji złożonych gestów. Każdy taki obiekt przechowuje referencje do wzorca oraz wyznaczone współczynniki korelacji dla poszczególnych gestów składowych.
      - **RecognizedGesture** - klasa obiektu będącego rozpoznanym gestem, zawiera informacje o tym, jaki wzorzec posłużył do jego rozpoznania, jakie kontury zostały wprowadzone przez użytkownika, a także czy gest został dodany automatycznie;
    - moduł wzorców:
      - **ComplexGesture** - klasa wzorca, przechowuje listę pojedynczych gestów, z których składa się gest złożony, znak, jaki jest mu przyporządkowany, identyfikator w bazie (w celu odróżnienia go spośród wielu), czas dodania wzorca oraz liczbę poprawnie i błędnie sklasyfikowanych gestów na jego podstawie.
      - **SamplesContainer** - kontener zawierający listę wzorców, obiekt tej klasy jest serializowany i służy głównie do zapisu danych o wzorcach w systemie plików.
      - **SamplesCollector** - Optymalizator liczby próbek, jest odpowiedzialny za ograniczanie liczebności wzorców w bazie i usuwanie nadmiarowych próbek.

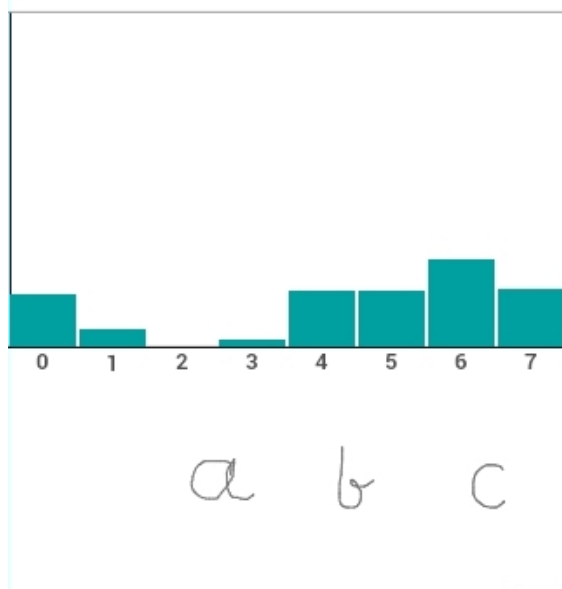
- moduł pojedynczych gestów:
  - **SingleGesture** - reprezentacja wydobytych cech z pojedynczego konturu, zawiera współrzędne punktu startowego, długość oraz histogram kodów łańcuchowych Freemana;
  - **Track** - nieprzetworzony kontur, lista punktów pochodzących z ekranu dotykowego;
  - **Point** - klasa reprezentacji punktu na płaszczyźnie, posiada dwie współrzędne:  $x$  i  $y$ ;
  - **FreemanHistogram** - klasa przedstawiająca rozkład kodów łańcuchowych Freemana na histogramie, jest reprezentowana przez jednowymiarową tablicę. Klasa posiada również metody do obliczania histogramu na podstawie konturu oraz do normalizacji histogramu.
- **GestureManager** - klasa zarządza listą wzorców aplikacji, jest odpowiedzialna m.in. za automatyczne dodawanie, usuwanie wzorców oraz odczyt i zapis do bazy aplikacji. Przechowuje listę rozpoznanych i nierozpoznanych gestów.
- **InputGesture** - reprezentuje wpisany przez użytkownika elementarny gest, zawiera informację o konturze oraz flagę oznaczającą, czy obiekt został poddany analizie przez algorytm identyfikacji.
- **TouchPanel** - kontroler panelu dotykowego, monitoruje na bieżąco położenie rysika lub palca na ekranie i tworzy listę punktów konturu. Rejestruje zdarzenia przyciśnięcia, przesunięcia po panelu oraz oderwania od ekranu.
- **Engine** - główny silnik aplikacji, obiekt decyzyjny. Zarządza logiką aplikacji na najwyższym poziomie. Definiuje zachowanie interfejsu oraz zarządza elementami z niższego poziomu.
- **ButtonsManager** - obiekt zarządzający przyciskami oraz wykonujący odpowiednie akcje w wyniku naciśnięcia przez użytkownika.
- **Statistics** - odpowiada za zbieranie danych statystycznych dotyczących procesu rozpoznawania gestów w celu wyznaczenia m.in. skuteczności całego interfejsu dotykowego.
- **Graphics** - główny element odpowiedzialny za grafikę aplikacji. Jego zadaniem jest wyświetlanie na ekranie odpowiednich ekranów interfejsu graficznego.
- moduł funkcji systemowych:
  - **Files** - służy do odczytu i zapisu w systemie plików;

- **InputManager** - odpowiada za wyświetlanie i obsługę klawiatury ekranowej oraz przygotowanie ekranów formularzy do wprowadzania danych przez użytkownika (liczb lub znaków)
- **Output** - klasa służy do raportowania błędów, wyświetlania komunikatów na ekranie i ich zapisu do dziennika zdarzeń, odpowiedzialna jest również za zgłaszanie wystąpienia błędów i wyjątków oraz wysyłanie odpowiednich komunikatów do środowiska programistycznego w celu zlokalizowania przyczyny niepoprawnego działania aplikacji.
- **TimerManager** - klasa odpowiadająca za odmierzanie czasu rzeczywistego, używana jest do uruchamiania czasomierzy i odmierzania ustalonego czasu oczekiwania.
- moduł ustawień:
  - **Preferences** - klasa odpowiedzialna za odczyt i zapis własnych ustawień użytkownika na urządzeniu (np. lokalizacji pliku z bazą wzorców)
  - **Config** - klasa przechowująca wartości większości parametrów przyjętych w aplikacji. W przypadku zaistnienia potrzeby modyfikacji jakiegoś parametru, wystarczy zmienić wartość odpowiedniej zmiennej w tej klasie.
- **MainActivity** - główna aktywność aplikacji, odpowiada za uruchomienie aplikacji oraz wykonanie akcji w odpowiedzi na komunikaty pochodzące z systemu operacyjnego.

### 6.3. Obsługa aplikacji

System posiada dwa główne tryby aplikacji, są to tryb menedżera gestów oraz tryb szybkiego pisania. Tryb menedżera gestów służy do zarządzania bazą wzorców (dodawania nowych wzorców, usuwania, czyszczenia całej listy wzorców) oraz wyświetlania wykresu histogramu łańcuchów Freemana ostatnio wpisanych konturów. Drugi tryb - szybkiego pisania wykorzystywany jest do ciągłego rozpoznawania gestów i zamiany ich na tekst (ciąg znaków). Umożliwia również poprawianie wstawionych znaków, kopiowanie rozpoznanego tekstu do schowka systemowego lub wyczyszczenie całego rozpoznanego tekstu. Przykładowy wygląd interfejsu użytkownika dla każdego trybu zamieszczono na poszczególnych zrzutach ekranu.

Czyść komunikaty	Zakończ
Szybkie Pisanie	Zapisz wzorzec
Lista wzorców	Lokalizacja wzorców
Usuń wzorzec	Wyczyść wzorce



Rysunek 6.2. Zrzut ekranu aplikacji w trybie menedżera gestów

Czyść komunikaty	Zakończ
Manager gestów	Zapisz wzorzec
Cofnij	Popraw
Kopiuj do schowka	Wyczyść tekst

Rozpoznany tekst:  
rozpoznane gesty  
tekst



Rysunek 6.3. Zrzut ekranu aplikacji w trybie szybkiego pisania

### 6.3.1. Interfejs użytkownika

W każdym z dwóch trybów na górze ekranu znajduje się lista przycisków, która różni się dostępnością niektórych elementów w zależności od trybu.

Przyciski w aplikacji wykonują następujące funkcje:

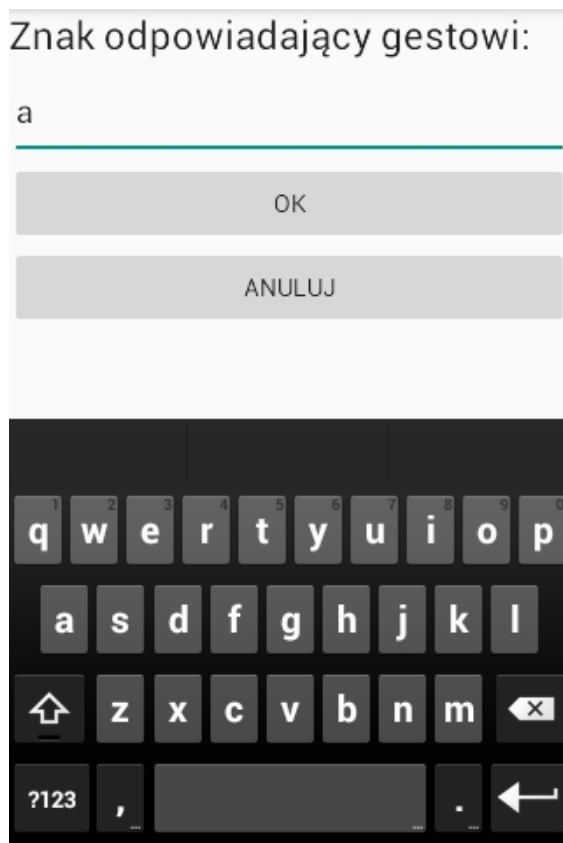
- **Zakończ** - zapisuje aktualną listę wzorców do pliku i zamyka aplikację;
- **Czyść komunikaty** - Usuwa z ekranu widoczne komunikaty;
- **Szybkie pisanie / Manager gestów** - przełącza między trybami aplikacji;
- **Zapisz wzorzec** - wyświetla odpowiedni ekran służący do wprowadzenia znaku reprezentowanego przez gest oraz złożoności zapisywanego gestu (liczby ostatnio wprowadzonych konturów)
- **Lista wzorców** - wyświetla komunikat o liczbie wszystkich wzorców, ich nazwach oraz przyporządkowanym znakom;
- **Lokalizacja wzorców** - wyświetla odpowiedni ekran pozwalający modyfikować ścieżkę położenia pliku zawierającego bazę wzorców;

- **Usuń wzorzec** - pozwala usunąć wybrany gest z bazy poprzez wskazanie jego nazwy;
- **Wyczyść wzorce** - usuwa wszystkie wzorce z listy;
- **Cofnij** - usuwa ostatni gest, zapamiętuje wystąpienie błędnego rozpoznania przez dany wzorzec;
- **Popraw** - usuwa ostatni gest (lub więcej w zależności od złożoności poprawionego gestu), zapamiętuje wystąpienie błędnego rozpoznania przez dany wzorzec oraz wyświetla ekran dodawania nowego wzorca;
- **Kopiuj do schowka** - kopiuje aktualnie rozpoznany łańcuch znaków do systemowego schowka, co pozwala na wykorzystanie wpisanego tekstu w innych aplikacjach na urządzeniu;
- **Wyczyść tekst** - Usuwa zawartość aktualnie rozpoznanego tekstu.

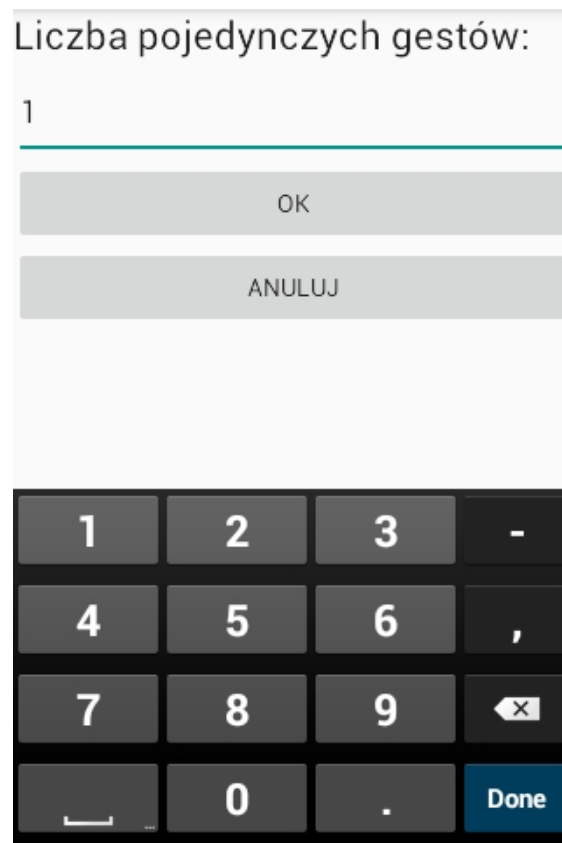
Oprócz przycisków użytkownik ma oczywiście możliwość pisania po ekranie (w obszarze, gdzie nie występują przyciski). W trakcie rysowania kontur jest na bieżąco aktualizowany i wyświetlany. Po zakończeniu rysowania konturu gestu, wykonuje się odpowiednia akcja. W przypadku trybu zarządzania gestami będzie to przetworzenie i wykreślenie histogramu łańcuchów Freemana na ekranie, natomiast w trybie szybkiego pisania będzie to próba dokonania analizy wprowadzonych konturów i dopisania znaku do rozpoznanego tekstu.

Naciśnięcie niektórych przycisków skutkuje wyświetleniem użytkownikowi ekranu wprowadzania danych za pomocą klawiatury ekranowej. Na przykład wciśnięcie przycisku "Zapisz wzorzec" powoduje wyświetlenie ekranu z rysunku 6.4. Po wpisaniu znaku i zatwierdzeniu go, pokazany zostanie ekran wyboru liczby gestów elementarnych (z numeryczną klawiaturą ekranową). Dopiero zatwierdzenie tego ekranu z poprawnymi danymi spowoduje dodanie gestu do bazy wzorców.





Rysunek 6.4. Ekran wyboru znaku dla nowego wzorca



Rysunek 6.5. Ekran wyboru liczby pojedynczych gestów dla nowego wzorca

Wprowadzenie przez użytkownika gestu, który nie został rozpoznany przez algorytm skutkuje tym, że również zostanie wyświetlone podobne okno dodawania nowego gestu, lecz tym razem z pytaniem o to, jaki znak ma reprezentować nierozpoznany gest.

## 6.4. Wykorzystane metody

### 6.4.1. Wykorzystanie biblioteki OpenCV

Do wyznaczania histogramów i obliczania współczynnika korelacji Pearsona dla histogramów wykorzystano funkcje z biblioteki OpenCV służącej do przetwarzania obrazów. Użycie biblioteki wymaga jej inicjalizacji podczas startu aplikacji.

Za wyznaczanie histogramu odpowiedzialna jest funkcja o nazwie "calcHist". Wynik zapisuje w obiektach klasy "Mat" (pochodzących również z biblioteki OpenCV). Natomiast do samego obliczenia współczynnika korelacji używana jest funkcja "compareHist" z pakietu "Imgproc". Jej parametrami są dwa histogramy wejściowe o takich samych rozmiarach oraz indeks metody porównywania, jaka ma zostać użyta. Biblioteka OpenCV udostępnia 4 metody porównywania:

- CV\_COMP\_CORREL - korelacja Pearsona,
- CV\_COMP\_CHISQR - metoda Chi-kwadrat,
- CV\_COMP\_INTERSECT - metoda przecięć,
- CV\_COMP\_BHATTACHARYYA - metoda odległości Bhattacharyya

Funkcja "compareHist" zwraca w wyniku liczbę zmiennoprzecinkową typu *double*. Na metodę porównywania histogramów wybrano korelację Pearsona, której to użycie podczas testów dawało najlepsze wyniki w rozpoznawaniu.

### 6.4.2. Mechanizm serializacji

Do zapisu i odczytu bazy wzorców w systemie plików wykorzystano mechanizm serializacji obiektów występujący w języku programowania Java. Serializacja polega na przekształceniu obiektów do postaci strumienia bajtów z zachowaniem aktualnego stanu. Procesem odwrotnym jest deserializacja, polegająca na odczycie strumienia bajtów z pliku i odtworzeniu obiektu wraz z jego stanem. Serializowanym obiektem w programie jest obiekt klasy "SamplesContainer". Zawiera on listę wszystkich wzorców jako atrybut klasy. Mechanizm serializacji wymaga, aby wszystkie klasy, które będą zapisywane (a więc również te, które są agregowane wewnątrz innych obiektów) implementowały interfejs *Serializable*. Dodatkowo zalecane jest, aby serializowane klasy miały zdefiniowane statyczne pole "serialVersionUID" z numerem wersji klasy. W przypadku zmiany struktury klasy, należy zwiększyć numer jej wersji. Wtedy pojawienie się niezgodności wersji przy odczycie obiektów uniemożliwi proces deserializacji, wymuszając stosowanie najnowszej wersji.

## **6.5. Moduł wprowadzania znaków pisma odręcznego**

Głównym celem implementacji oprogramowania na urządzenie mobilne było utworzenie modułu wprowadzania znaków pisma odręcznego jako alternatywnej metody wpisywania tekstu. System Android pozwala na tworzenie aplikacji działających w tle jako usługi. Zastosowanie takiego rozwiązania wraz z przyznaniem aplikacji pełnego dostępu do ekranu dotykowego mogłoby rozszerzyć funkcjonalność modułu i pozwolić na bezpośrednie wprowadzanie tekstu za pomocą gestów w innych aplikacjach uruchomionych w systemie. W utworzonym module użytkownik ma do dyspozycji możliwość kopiowania wpisanego tekstu do schowka, co również pozwala na wykorzystanie go w innych aplikacjach.

Projekt skupia się na rozpoznawaniu znaków tekstowych, natomiast łatwo można rozszerzyć tę funkcjonalność o obsługę gestów takich jak: gest usuwania ostatniego znaku, bądź też zmiany aktualnego położenia kursora tekstu. W aplikacji możliwe jest natomiast wstawianie białych znaków (spacji lub załamania wiersza) poprzez gesty. Nie istnieje ograniczenie dotyczące długości tekstu przypisanego do gestu. Oznacza to, że jednemu gestowi może zostać przypisanych wiele znaków (np. całe wyrazy), co jeszcze bardziej może zwiększyć wygodę i szybkość pisanie poprzez stosowanie inteligentnego interfejsu dotykowego przy wprowadzaniu tekstu.

## **6.6. Cechy systemu**

System pozwala na ciągle wpisywanie tekstu przez użytkownika. Nie jest konieczne odczekanie pewnego czasu po wpisaniu gestu, gdyż analiza odbywa się na bieżąco. Mimo to, można w pewnym stopniu wpłynąć na działanie interfejsu dotykowego poprzez odczekanie pewnego czasu po wpisaniu gestu. System zinterpretuje to jako zakończenie pisanie znaku i nie będzie rozważał gestów z większą złożonością niż liczba wprowadzonych konturów wejściowych. Takie rozwiązanie sprawia, że użytkownik zawsze powinien mieć wrażenie, że tekst jest rozpoznawany na bieżąco, bez zbędnych opóźnień.

System jest praktycznie gotowy do użytkowania nawet w przypadku pustej bazy wzorców. Braki wzorców zostaną uzupełnione w wyniku zadawania użytkownikowi pytań o to, jakie znaki zostały wprowadzone. Z czasem takich pytań będzie coraz mniej, skuteczność w rozpoznawaniu powinna wzrastać, a system powinien stawać się coraz bardziej "samodzielny", wzbogacając swoją bazę o coraz to kolejne wzorce.

## 7. Testy aplikacji

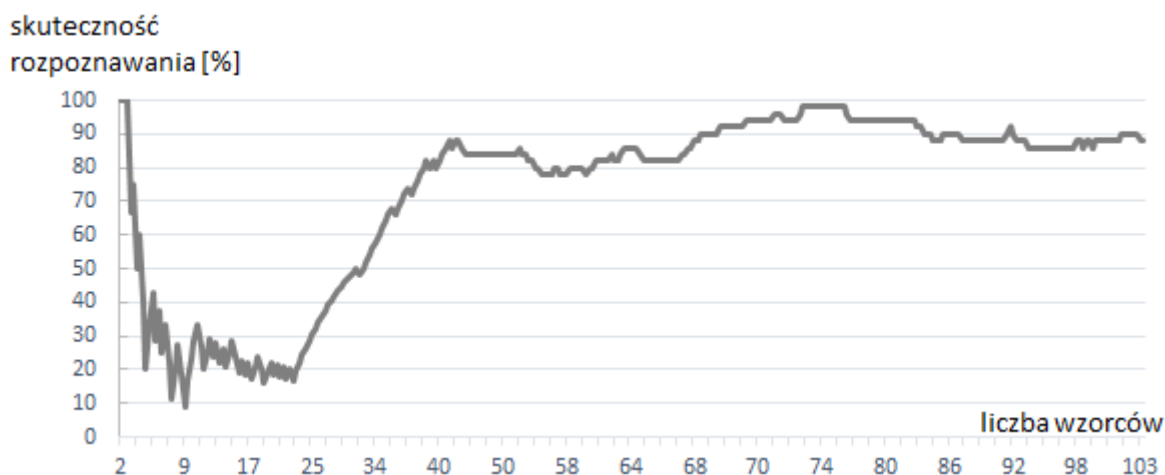
### 7.1. Dane statystyczne

W celu wyznaczenia skuteczności rozpoznawania gestów (procentowej części poprawnych wyników spośród wszystkich) zaimplementowano w aplikacji osobny moduł do zbierania danych statystycznych. Odpowiednie testy zostały przeprowadzone na urządzeniu mobilnym podłączonym do komputera przez kabel USB w trybie debugowania, co pozwalało na przesyłanie zbiorczego wyniku w odpowiedniej formie do środowiska programistycznego.

Moduł statystyk zbiera dane w postaci listy zdarzeń. Zdarzeniami są np. poprawne rozpoznanie gestu lub błędna identyfikacja. Lista zdarzeń ma ograniczony rozmiar, w przypadku przepełnienia przechowywane są zawsze najnowsze zdarzenia. Dzięki takiemu podejściu można łatwo odczytać, jaką część spośród wszystkich ostatnich zdarzeń stanowią np. poprawne rozpoznania gestu, a więc jaka jest skuteczność metody wprowadzania znaków.

### 7.2. Test uczenia się systemu

Aby zademonstrować, jak działa inteligencja systemu, i czy rzeczywiście prowadzi do poprawienia skuteczności, przeprowadzono pewien eksperyment. Polegał na tym, że usunięto w aplikacji wszystkie wzorce, pozostawiając pustą bazę, a następnie zaczęto wprowadzać kontury znaków w celu rozpoznania ich. Oczywiście początkowo system nie znał znaków i wyświetlał ekran z pytaniem o to, jaki gest został wprowadzony. Jednak w stosunkowo krótkim czasie, system zaczął wykorzystywać zdobytą wiedzę o gestach i poprawił skuteczność. W trakcie trwania testu zapisywane oraz przesyłane były dane statystyczne, które posłużyły do stworzenia poniższego wykresu. Przedstawia on zależność skuteczności rozpoznawania gestów od liczby wszystkich wzorców znajdujących się w bazie aplikacji.



Rysunek 7.1. Wykres zależności skuteczności rozpoznawania gestów od liczby wszystkich wzorców

Test aplikacji przeprowadzono na wszystkich małych literach alfabetu polskiego, które wprowadzane były w kolejności losowej. W sytuacji nierozpoznania żadnego gestu i wyświetlenia ekranu z pytaniem, gest nie był traktowany ani jako poprawnie, ani jako błędnie rozpoznany.

Na początku liczba znanych klas była równa 0. Następnie w wyniku braku zdefiniowanych gestów, niektóre kolejne kontury były błędnie klasyfikowane jako inne gesty, przez co zaczęła spadać skuteczność. Wzrost skuteczności nastąpił po zdefiniowaniu wszystkich wzorców. Można zauważyć, że już przy liczbie wzorców równej około 45, skuteczność rozpoznawania zaczęła oscylować wokół wartości 90 %.

### **7.3. Dobór parametrów**

W implementacji systemu wiele wartości współczynników zostało dobranych doświadczalnie na skutek przeprowadzenia testów. W początkowej fazie powstawania programu wartości te były przyjmowane intuicyjnie, gdyż nie było możliwości przeprowadzenia testów skuteczności, natomiast kiedy system rozpoznawania osiągnął podstawową funkcjonalność, można było przeprowadzić testy i stwierdzić, jakie wartości współczynników poprawiają parametry interfejsu dotykowego.

Poniższa tabela przedstawia zbiorcze zestawienie najważniejszych wartości parametrów programu istotnych dla algorytmu rozpoznawania gestów.

**Tabela 7.2. Zbiorcze zestawienie wartości parametrów wykorzystanych w aplikacji**

Parametr	Wartość
czas braku aktywności użytkownika, po jakim wprowadzanie gestu zostaje uznane za zakończone	800 ms
liczba pikseli do uśredniania przy filtrowaniu	20
minimalna liczba pikseli konturu (by nie został uznany za kropkę)	20
liczba kierunków łańcuchów Freemana	8
waga współczynnika korelacji histogramów (przy obliczaniu sumarycznego współczynnika korelacji)	0,668
waga współczynnika korelacji punktów startowych konturów	0,166
waga współczynnika korelacji długości konturów	0,166
minimalny współczynnik korelacji dla pojedynczego gestu	0,85
maksymalna liczba wzorców przechowywana dla jednego znaku	25
maksymalna wartość współczynnika korelacji dla automatycznego dodawania wzorców	0,96
graniczna wartość bilansu identyfikacji, dla której wzorzec zostaje usunięty	-3

#### 7.4. Testy końcowe działającej aplikacji

Na działającej aplikacji posiadającej już stosunkowo dużą liczbę wzorców dodanych przez użytkownika również przeprowadzono testy skuteczności. Testy te dotyczyły rozpoznawania małych liter polskich, dużych liter, cyfr oraz niektórych znaków specjalnych. Liczba wszystkich wzorców znajdujących się w bazie aplikacji przed wykonaniem testów wyniosła 286. Warto dodać, że przy takiej stosunkowo dużej liczbie wzorców, rozmiar pliku przechowującego bazę wzorców wynosił jedynie 42,4 kB. W wyniku testu uzyskano średnią skuteczność rozpoznawania gestów równą około 92,1 %. Zmierzony został także czas analizy po wpisaniu konturu. Jest to czas porównywania wpisanego gestu ze wszystkimi wzorcami. Na badanym urządzeniu czas ten mieścił się w przedziale od 3 ms do 98 ms, zaś jego średnia wartość wyniosła 43 ms. Jest to bardzo dobrym wynikiem i można stwierdzić, że opóźnienia wynikające z etapu analizy praktycznie nie występują i są zupełnie niezauważalne z perspektywy użytkownika. Stwierdzono także, że czas potrzebny do klasyfikacji jest w przybliżeniu proporcjonalny do liczby wszystkich wzorców w bazie, zaś wpływ złożoności przechowywanych gestów jest znikomy.

## 8. Wnioski

Celem niniejszej pracy było stworzenie inteligentnego interfejsu dotykowego, który umożliwiałby obsługę złożonych gestów. W szczególności powinien służyć do rozpoznawania znaków pisma odręcznego wprowadzanych przez użytkownika poprzez rysowanie na ekranie dotykowym. Zastosowanie własnych, uczących się algorytmów sprawia, że system stara się uogólniać dane przekazane w ciągu uczącym i samodzielnie podejmować decyzje o przynależności nieznanymi obiektów. Wszystkie cele pracy zostały osiągnięte a założenia systemu zrealizowane.

Elementy odpowiedzialne za inteligencję systemu wpływają na poprawę "samodzielności" oraz wzrost skuteczności rozpoznawania wraz z dłuższym czasem użytkowania aplikacji. Stworzony interfejs dotykowy potrafi także dostosowywać się do zmiennych warunków, w jakich pracuje.

Podczas użytkowania interfejsu dotykowego zaobserwowano, że aplikacja bardzo dobrze radzi sobie z rozpoznawaniem podłużnych konturów (lub składających się z wielu podłużnych odcinków, np. litery "L" lub "Z"), natomiast gorsze wyniki osiąga podczas rozpoznawania zaokrąglonych kształtów (np. litery "o"). Może to być spowodowane większą wrażliwością na zakłócenia podczas rysowania zaokrąglonych konturów i w efekcie utworzenia w reprezentacji obiektu zniekształconego histogramu kodów łańcuchowych. Rozwiązaniem tego problemu może być zastosowanie innej metody filtracji konturów, wybór innej metody obliczania współczynnika korelacji histogramów lub ponowny dobór parametrów aplikacji pod kątem zwiększenia skuteczności rozpoznawania tych kształtów. Lepsze rezultaty mogłoby zapewnić także wprowadzenie rozpoznawania liter w kontekście całego słowa.

Kolejnym etapem rozwoju systemu może być rozszerzenie funkcjonalności aplikacji o obsługę akcji gestów, wykonujących określone czynności w systemie operacyjnym (np. skrótowe uruchamianie innych aplikacji). Można tego dokonać poprzez uruchamianie aplikacji w trybie usługi działającej w tle oraz przechwytywanie wszystkich gestów wykonywanych na ekranie urządzenia. Wymaga to jednak nadania odpowiednich uprawnień aplikacji oraz głębokiej integracji z systemem.

Dobór niektórych, istotnych dla aplikacji parametrów często był możliwy jedynie na drodze empirycznej. Dlatego też dalszą możliwością rozwoju interfejsu dotykowego jest przeprowadzenie większej liczby testów pod kątem wyboru korzystniejszych parametrów aplikacji w celu poprawy skuteczności całego systemu.

## Literatura

- [1] Baggio D.: Mastering OpenCV with Practical Computer Vision Projects, 2012, 148 - 171.
- [2] Bradski G., Kaehler A.: Learning OpenCV, 2008.
- [3] <http://opencv.org/>
- [4] <http://developer.android.com/develop/>
- [5] Tadeusiewicz R., Korohoda P.: Komputerowa analiza i przetwarzanie obrazów. FPT, Kraków 1997.
- [6] Tadeusiewicz R.: Rozpoznawanie obrazów, PWN, Warszawa 1991.



## **Załączniki**

Do pracy dołączona została płyta CD, zawierająca:

- kod źródłowy aplikacji,
- pracę inżynierską w formacie "pdf",
- życiorys autora w formacie "pdf",
- plakat przedstawiający wyniki pracy w formacie "pdf".