Rok akademicki 2013/2014

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Kamil Gołofit

Rozpoznawanie tablic rejestracyjnych z sekwencji wideo

Opiekun pracy
mgr inż. Krzysztof Chabko

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 1991.08.20

Data rozpoczęcia studiów: 2010.10.01

Życiorys

Urodziłem się 20.08.1991r. w Puławach, w województwie lubelskim. Zainteresowanie informatyką przejawiło się podczas nauki w Publicznym Gimnazjum nr 3 w Puławach, gdzie pełniłem rolę administratora strony szkolnej w czasie nauki. W szkole średniej uczęszczałem do klasy o profilu matematyczno-fizyczno-informatycznym, gdzie dalej rozwijałem moje zainteresowanie tą dziedziną wiedzy. W kwietniu 2010 roku ukończyłem I Liceum Ogólnokształcące im. A. J. ks. Czartoryskiego w Puławach, w lipcu tego samego roku zostałem przyjęty na Wydział Elektroniki i Technik Informacyjnych Politechniki Warszawskiej, kierunek Informatyka, studia dzienne. Podczas piątego semestru studiów rozpocząłem pracę zawodową jako stażysta-programista w Quality Business Software z siedzibą w Warszawie, gdzie pracuję do chwili obecnej jako młodszy programista.

.....
Podpis studenta

EGZAMIN DYPLOMOWY

Złożył egzamin dyplomowy w dniu ..... 20__ r

z wynikiem .....

Ogólny wynik studiów: .....

Dodatkowe wnioski i uwagi Komisji: .....

.....

.....

STRESZCZENIE

Niniejsza praca inżynierska porusza zagadnienie rozpoznawania obrazów. Jej celem jest wykonanie aplikacji zdolnej do otwierania strumienia wideo, przechwytywania z niego klatek i przetwarzania ich, rozpoznając znajdujące się na obrazie znaki tablic rejestracyjnych. Praca dokumentuje przebieg prac nad realizacją tejże aplikacji w języku Java, zawierając opis całego procesu tworzenia aplikacji, od wymagań, przez projekt, po realizację.

W pierwszych rozdziałach zdefiniowany jest problem rozpoznawania, przedstawione są istniejące podobne systemy, oraz opisany jest typowy system rozpoznawania obrazów jako podłoże do realizacji programu. Dalsze rozdziały zawierają analizę wymagań aplikacji i projekt jej wykonania. Na końcu niniejszej pracy znajduje się opis realizacji systemu rozpoznawania, opis graficznego interfejsu użytkownika aplikacji powstałej w ramach tej pracy oraz podsumowanie tematu pracy.

Słowa kluczowe: rozpoznawanie obrazów, przetwarzanie obrazu, Java, rozpoznawanie tablic rejestracyjnych.

LICENSE PLATE FROM VIDEO STREAM RECOGNITION

Engineer's Degree Thesis touch the issue of pattern recognition. The aim of this study is to build an application, which is capable of opening a video stream, capturing single frames from it and processing them with recognizing vehicle's license plates contained in the analyzed single frame. This paper documents course of work on building the application in Java programming language, it contains description of entire creation process, from application requirements, through projecting an application, to implementation.

In the first chapters of this paper there is definition of the recognition problem in this study, analysis of the existing Polish systems similar to the topic of this study and a description of typical recognition system as an introduction to realization of the application. Further chapters of this paper contain requirements analysis of the application and implementation project. At the end of this paper, there is a description of the realization of the recognition system, description of a graphical user interface of existing application and the summary of this study.

Keywords: pattern recognition, image processing, Java, license plate recognition.

Pragnę podziękować

moim Rodzicom,
bez których nigdy nie osiągnąłbym wyznaczonego celu,

oraz Ewie,
za wsparcie i pomoc w edycji tej pracy.

Kamil Gołofit

Spis treści

1. WSTĘP 5

2. OPIS PROBLEMU 7

3. PROBLEM ROZPOZNAWANIA OBRAZÓW 8

3.1 Czym jest rozpoznawanie obrazów? 8
3.2 System rozpoznawania obrazów 11
3.3 Obliczanie niezmienników geometrycznych 14

4. SPRECYZOWANIE PROBLEMU 15

4.1. Przegląd istniejących systemów 15
4.2 Wymagania funkcjonalne 17
4.3 Wymagania niefunkcjonalne 19

5. PROJEKT ROZWIĄZANIA 20

5.1 Język programowania i środowisko programistyczne 20
5.2 Biblioteki i technologie użyte przy tworzeniu aplikacji 22
5.3 Przypadki użycia 26
5.4 Ogólny zarys diagramu klas 29
5.5 Wybrany diagram sekwencji 31

6. REALIZACJA SYSTEMU ROZPOZNAWANIA 33

6.1 Akwizycja i segmentacja 33
6.2 Wyznaczanie cech i rozpoznawanie 35

7. OPIS INTERFEJSU UŻYTKOWNIKA 36

8. PODSUMOWANIE DZIAŁANIA APLIKACJI 43

9. ZAKOŃCZENIE 45

10. BIBLIOGRAFIA 46

DODATEK A – WYBRANE WZORY MATEMATYCZNE 47

DODATEK B – OPIS WYBRANYCH KLAS PROGRAMU 48

1. Wstęp

Tematem pracy inżynierskiej jest utworzenie aplikacji zdolnej do analizowania filmów wideo, lokalizowania w nich tablic rejestracyjnych pojazdów i rozpoznawania znaków na zlokalizowanych tablicach. W rozwijającym się świecie technologii takie systemy mają wiele zastosowań – używa się ich między innymi do automatycznego rozpoznawania pojazdu wjeżdżającego i wyjeżdżającego z płatnego parkingu, celem naliczenia odpowiedniej opłaty, rozpoznawania numerów rejestracyjnych pojazdów przez różnego rodzaju służby, np. w poszukiwaniu skradzionych pojazdów, tworzenia statystyk ruchu (badanie miejsca zamieszkania kierowców i tras przez nich pokonywanych), wyszukiwania miejsc parkingowych na automatycznych parkingach czy kontroli granicznej. Są one coraz bardziej powszechne, gdyż zastępują pracę człowieka, uzyskują bardzo dobre wyniki rozpoznawania (niektóre systemy rozpoznają poprawnie ponad 95% znaków, co jest porównywalne z ilością jaką rozpoznaje ludzki operator), a także zapewniają szybsze przetwarzanie danych, co prowadzi do wzrostu wydajności organizacji, w których takie systemy są stosowane. Zastosowanie opisywanego systemu w codziennym życiu oraz jego zasada działania bardzo mnie zainteresowały, dlatego postanowiłem sam stworzyć aplikację o podstawowej funkcjonalności w tej dziedzinie – rozpoznawania znaków na tablicach rejestracyjnych.

Tematyka niniejszej pracy zaciekawiła mnie podczas studiów, kiedy w ramach jednego z projektów miałem przygotować system rozpoznawania pojedynczego symbolu. Podczas gdy większość studentów rozpoznawała znak drogowy czy logo firmy, ja zdecydowałem się na rozpoznawanie paru określonych znaków na tablicy rejestracyjnej. Tematyka ta wydała mi się interesująca, dzięki czemu postanowiłem wybrać temat pracy inżynierskiej związany z rozpoznawaniem obrazów.

Aplikacja, będąca wynikiem niniejszej pracy, powinna być w stanie otwierać strumień wideo, przechwytywać z niego pojedyncze klatki i poddawać je analizie oraz rozpoznawaniu znaków. Efektem pracy aplikacji ma być lista tablic znalezionych i rozpoznanych w klatkach strumienia wideo wraz z tekstową reprezentacją numeru tablicy rejestracyjnej. Program może popełniać błędy w klasyfikacji znaków (będzie się mylił w klasyfikacji poszczególnych znaków tablic rejestracyjnych) ale celem pracy jest stworzenie działającej aplikacji, będącej podstawą do dalszych badań nad skutecznością algorytmów segmentacji i rozpoznawania. Gotowy program będzie rozpoznawał znaki w określonych warunkach (oświetleniowych, pogodowych, przy statycznym pojeździe).

W ramach dalszych prac – na przykład pracy magisterskiej – możliwa będzie poprawa jakości klasyfikacji, poprzez testowanie różnych algorytmów i badanie ich działania dla problemu rozpoznawania tablic rejestracyjnych. Aplikacja powinna dać się uruchomić na współczesnym, przeciętnym komputerze stacjonarnym lub laptopie, bez instalacji specjalnych bibliotek czy podprogramów, powinna też posiadać przyjazny interfejs, umożliwiający użytkowanie aplikacji zwykłemu człowiekowi, niekoniecznie posiadającemu wykształcenie z dziedziny informatyki czy przetwarzania obrazów.

W pracy przedstawione będą kolejne kroki tworzenia opisanej pokrótce aplikacji od ogólnego pomysłu, poprzez wymagania i założenia, po opis obsługi poszczególnych jej modułów i widoków. Rozdział trzeci omawia problem rozpoznawania obrazów, skupiając się na poszczególnych elementach typowego systemu rozpoznawania obrazów, ich działaniu i zastosowaniu. W rozdziale czwartym przedstawiono wymagania aplikacji, oraz spis wybranych istniejących systemów rozpoznawania tablic na polskim rynku.

Rozdział piąty opisuje proces projektowania aplikacji. Opisane są w nim biblioteki i technologie użyte przy implementacji programu, przedstawiona jest także analiza przypadków użycia programu. Na ich podstawie przedstawiony został ogólny diagram klas programu oraz przykład diagramu sekwencji, prezentujący sposób obsługi przykładowego zdarzenia. W rozdziale szóstym zaprezentowany został sposób realizacji systemu rozpoznawania w aplikacji. Omówiony jest w nim wybór i zastosowanie poszczególnych elementów systemu rozpoznawania, przedstawionego wcześniej w rozdziale trzecim. Rozdział siódmy zawiera opis interfejsu użytkownika gotowej aplikacji, prezentując jej poszczególne ekrany i opisując jej działanie i poszczególne funkcjonalności.

Na koniec, w rozdziale ósmym, opisane zostało działanie aplikacji po jej zbudowaniu i przetestowaniu oraz konfrontacja gotowej aplikacji z założeniami projektowymi. Podane są także parametry, jakie gotowa aplikacja osiąga - czas przetwarzania jednej klatki, szybkość wyświetlania strumienia wideo z analizą i rozpoznawaniem „w locie”, a także wynikająca z testów jakość klasyfikacji. Rozdział dziewiąty zawiera podsumowanie projektu. Ponadto praca posiada także dodatek A – wybrane wzory służące do obliczania niezmienników geometrycznych, wykorzystywanych w projekcie oraz dodatek B – opis klas wykorzystywanych do procesu rozpoznawania znaków na tablicach.

2. Opis problemu

Na początku chcę określić zadanie, postawione przede mną w niniejszej pracy. Opisywana tu aplikacja powinna zostać ukończona, uruchomiona i przetestowana. Powinna także spełniać postawione przed nią w rozdziale 4. wymagania, które na początek chciałbym zdefiniować nieco mniej formalnie.

Docelowo aplikacja powinna umożliwiać odtwarzanie strumienia wideo oraz analizować go w poszukiwaniu tablic rejestracyjnych. W tym celu, w ramach niniejszej pracy, zaprojektuję i zaimplementuję system rozpoznawania obrazów oraz zbuduję przepływ danych, umożliwiający skierowanie strumienia wideo do systemu rozpoznawania.

Strumień wideo może pochodzić z dwóch podstawowych źródeł: może być nagrany wcześniej i ponownie odtworzony (otworzony z pliku wideo), lub też może być pozyskiwany w czasie rzeczywistym (przechwytywany bezpośrednio z kamery). W pierwszym wypadku aplikacja powinna umieć otworzyć plik wideo oraz wyodrębnić z niego poszczególne klatki. W drugim powinna umieć znaleźć kamerę podłączoną do komputera i przechwycić klatki przez nią rejestrowane. W obu przypadkach otrzymane klatki filmu zostaną poddane analizie i procesowi rozpoznawania obrazów w czasie rzeczywistym i wyświetlone użytkownikowi, wraz z informacją o znalezionych tablicach rejestracyjnych. Proces analizy i rozpoznawania będzie trwał jakiś czas, podczas którego nowe klatki będą wyodrębnianie z pliku czy przechwytywane przez kamerę. Aplikacja musi umieć odrzucić odpowiednią ilość klatek i przetwarzać oraz wyświetlać tylko takie, aby użytkownik miał wrażenie ciągłości filmu.

System rozpoznawania obrazów został opisany w rozdziale trzecim. Jego implementacja jest drugim kluczowym elementem aplikacji. Musi być on zaprojektowany i zaimplementowany w taki sposób, aby spełniać swoje zadanie (rozpoznawać tablice rejestracyjne) w możliwie krótkim czasie – tak, by zapewnić ciągłość odtwarzania wideo z punktu widzenia użytkownika. Rozpoznane tablice powinny być zapisywane, aby użytkownik mógł je przejrzeć, przeanalizować oraz wyświetlane w czasie rzeczywistym – użytkownik, oglądając strumień wideo, powinien widzieć położenie zlokalizowanych tablic i znaki na nich rozpoznane. Dzięki temu widoczna będzie praca w czasie rzeczywistym, a jednocześnie możliwa będzie analiza działania programu i algorytmów w nim zastosowanych.

Chciałbym, aby utworzona aplikacja oferowała jednak coś więcej niż samo rozpoznawanie tablic rejestracyjnych, zaprojektowane i wykonane na stałe – zakładam

wbudowanie w nią takich mechanizmów, aby użytkownik miał możliwość sterować w jakimś stopniu przebiegiem procesu rozpoznawania. Ponieważ każdy z użytkowników ma swoje preferencje, może używać aplikacji do różnych celów, chciałbym umożliwić personalizację działania programu. Użytkownik powinien otrzymać narzędzia do oceny działania aplikacji tak, aby mógł zmienić ustawienia i porównać pracę aplikacji z różnymi ustawieniami, dzięki czemu każdy z użytkowników będzie mógł dostosować program do swoich indywidualnych potrzeb i zastosowań.

3. Problem rozpoznawania obrazów

Problem rozpoznawania obrazów składa się z wielu etapów, ułożonych w odpowiedniej kolejności. Mają one na celu kolejno: poprawę jakości obrazu poddanego analizie, lokalizację potencjalnych obiektów do rozpoznawania, wyliczenie ich cech, rozpoznawanie i wyprowadzenie decyzji. Rozdział ten ma na celu przybliżyć czytelnikowi zarys systemu rozpoznawania obrazów, zasadę działania i cel istnienia poszczególnych etapów. Pozwoli to na głębsze zrozumienie problemu i ułatwi jego sprecyzowanie, przedstawione w następnym rozdziale. Na początku przedstawię sam problem rozpoznawania obrazów, a w kolejnym podrozdziale omówię budowę całego systemu rozpoznawania.

3.1 Czym jest rozpoznawanie obrazów?

Pojęcie rozpoznawania obrazów powstało z przetłumaczenia angielskiego zwrotu *pattern recognition*. Jest to więc rozpoznawanie wzorca, wyszukanego wcześniej w obrazie. Samo rozpoznawanie to prosta klasyfikacja obiektu, który został znaleziony, zakończona decyzją. Znalezienie obiektu w obrazie jest kluczowe dla rozpoznawania – jeśli obiekt nie zostanie znaleziony nie będzie czego rozpoznawać.

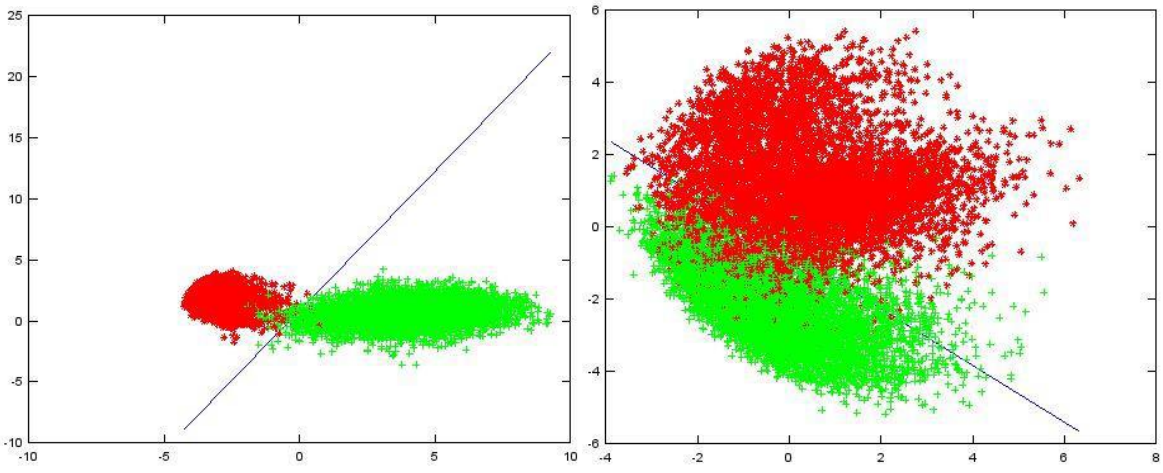
Obraz – bitmapa – jest reprezentowana w pamięci jako wartości kolejnych pikseli, zapisanych najczęściej na 3 lub 4 bajtach (po bajcie dla kanału R, G, B, czwarty bajt to kanał Alpha – kanał przezroczystości). Wartości kolejnych pikseli nie niosą same w sobie żadnej informacji, są jedynie wartościami kolorów w kolejnych punktach obrazu. Dopiero zebranie wszystkich pikseli (ich barw) występujących w obrazie i ich graficzne przedstawienie mają znaczenie dla człowieka – potrafi on rozpoznać kształty, kolory, postacie, obiekty. Z punktu widzenia algorytmu, który miałby za zadanie wydobyć z obrazu interesujący nas obiekt, sam obraz jest jedynie zbiorem wartości poszczególnych

pikseli – nie niesie ze sobą informacji o tym, gdzie znajduje się szukany obiekt. Z tego powodu najczęściej wydobywa się go stosując różne maski nakładane na obraz lub sterując kontrastem w nim. Proces wydobywania szukanego obiektu spośród tła (wszystkiego co szukanym obiektem nie jest) jest nazywany segmentacją. Ma ona za zadanie odróżnić obiekty, których potencjalnie szukamy, od tła. Na tym etapie może zdarzyć się tak, że wydzielony przez segmentację obiekt nie jest tym, który chcemy znaleźć. W takiej sytuacji dalsze etapy rozpoznawania powinny odrzucić znaleziony obiekt, nie klasyfikując go. Najczęściej w szukanym obiekcie znajduje się prosty kształt (koło, kwadrat, prostokąt o stałym stosunku szerokości do wysokości) lub stały kolor (niezmienny dla różnych obiektów). Poszukując w obrazie np. znaków zakazu (znaków drogowych), moglibyśmy szukać okrągłego obiektu, mającego czerwoną obwódkę o stałej grubości w stosunku do promienia koła. Projektując system rozpoznawania obrazu należy więc znaleźć takie atrybuty szukanych elementów, które będą niezmienne dla szukanej grupy obiektów. Na ich podstawie zaprojektować algorytm szukający obszarów obrazu zawierających obiekt do klasyfikacji. Segmentacja może odbywać się np. poprzez progowanie – porównywanie koloru piksela z wartością progową (najczęściej wyznaczoną doświadczalnie lub obliczoną na podstawie kolorów występujących w obrazie), gdzie wartość niższa niż progowa oznacza, że piksel należy do szukanego obiektu a wartość wyższa oznacza, że piksel należy do tła (lub odwrotnie, zależnie od przyjętej konwencji). Inną metodą segmentacji jest rozrost obszarów – wybór punktu (piksela) startowego, a następnie badanie sąsiadów punktu startowego, czy spełniają przyjęte kryterium jednorodności (np. czy nie różnią się barwą od sąsiada o więcej niż X). Istnieje wiele innych metod segmentacji, które można łączyć ze sobą, udoskonalać, dostosowywać do indywidualnych potrzeb danego projektu.

Po zlokalizowaniu obiektu w obrazie, musimy znaleźć cechy charakteryzujące dany obiekt. Jak już wspomniałem, z punktu widzenia algorytmu nie da się po prostu zaobserwować istnienia szukanego obiektu i sklasyfikować go. Algorytm potrzebuje wartości liczbowych, cech, które jednoznacznie określają dany kształt. Dzięki temu będzie w stanie sklasyfikować znaleziony obiekt. Cechami mogą być np. wszystkie piksele zawierające nasz szukany i zlokalizowany obszar ale w takim wypadku, ten sam obiekt na obrazie, mający inne rozmiary (powiększony/pomniejszony) lub też obrócony, osiągnie inne wartości. Cechy charakteryzujące szukany obiekt muszą więc być niezmienne względem trzech podstawowych operacji: translacji, rotacji oraz skalowania. Z racji tego, że najczęściej chcemy jak najszybciej uzyskać odpowiedź na pytanie „jaki obiekt znajduje się na obrazie?”, cechy powinny być możliwe do szybkiego obliczenia (ich policzenie nie

powinno zajmować wiele czasu). Zestaw cech spełniających podane kryteria jednoznacznie określa dany kształt i pozwala na rozróżnianie obiektów (np. różnych znaków zakazu od siebie). Przykładowymi cechami mogą być choćby stosunek kwadratu obwodu obiektu do jego powierzchni, wspomniane wartości pikseli obrazu czy wyliczone niezmienniki geometryczne[1].

Po określeniu systemu cech jednoznacznie określających klasyfikowane obiekty, znalezieniu szukanego obiektu w obrazie i policzeniu cech dla tego obiektu, możemy przystąpić do jego klasyfikacji. Istnieje wiele klasyfikatorów, z których chciałbym pokrótce omówić jedynie kilka. W tym miejscu należy wspomnieć, że jeśli chcemy klasyfikować obiekty, musimy mieć z czym je porównywać, potrzebujemy jakiejś bazy wiedzy, czyli zbioru cech wyliczonych dla różnych, możliwych do rozpoznania przez nasz system, obiektów. Na ich podstawie wybrany klasyfikator może np. stwierdzić do jakiego obiektu bazy wiedzy dany obiekt do klasyfikacji jest najbardziej podobny i udzielić odpowiedzi, jaki najbliższy obiekt znajduje się w bazie wiedzy. Opisany przykład to klasyfikator minimalno-odległościowy. Klasyfikator ten, po otrzymaniu zestawu cech policzonych z jakiegoś obiektu do klasyfikacji, liczy wcześniej zdefiniowaną odległość (np. odległość Euklidesową) do wszystkich wpisów w bazie wiedzy, znajduje wpis o najmniejszej odległości (najbliższy szukanemu obiektowi) i jako odpowiedź (wynik klasyfikacji) zwraca obiekt przyporządkowany do znalezionego wpisu o najmniejszej odległości.



Rys. 1.Przykład ilustrujący wyznaczone proste, rozdzielające dwie klasy obiektów. Wykres z lewej prezentuje prostą dość dobrze rozdzielającą klasy, z prawej widać klasy nachodzące na siebie.

Innym przykładem klasyfikatora jest klasyfikator liniowy. Jeśli określone są dwie cechy, którymi charakteryzujemy każdy potencjalny obiekt, możemy w oparciu

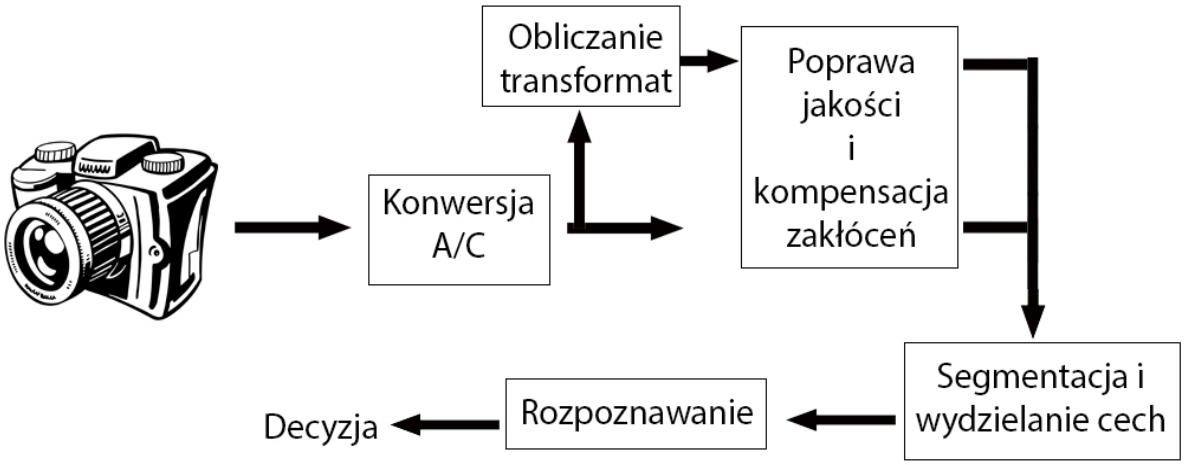
[1] Zob. rozdział 3.3 „Obliczanie niezmienników geometrycznych”.

o bazę wiedzy stworzyć wykres zależności cech i nanieść na niego wpisy dwóch różnych klas obiektów z bazy wiedzy jako punkty na płaszczyźnie (np. cechy znaków zakazu postoju oraz cechy znaków zakazu ruchu) – patrz rys. 1. Kolejnym krokiem jest wyznaczenie prostej na płaszczyźnie, najlepiej rozdzielającej dwie klasy obiektów. Postępując tak dla kolejnych par klas obiektów, uzyskujemy zbiór prostych, rozdzielających dwie klasy obiektów. Następnie przy klasyfikacji, wyznaczone proste oddają głosy, do której z dwóch oddzielanych przez prostą klas należy obiekt. Głosy są zliczane i jako odpowiedź zwracana jest klasa, na którą najczęściej głosowano.

Proces rozpoznawania obrazów jest procesem złożonym z wielu etapów. Jeśli ma być on prawidłowy, dający jak najlepsze wyniki, poszczególne etapy muszą pasować do siebie i być dobrze zaprojektowane. Uzyskanie dobrych wyników klasyfikacji nie jest więc rzeczą prostą, gdyż wymaga zgrania ze sobą kilku etapów procesu rozpoznawania.

3.2 System rozpoznawania obrazów

Tworząc system rozpoznawania obrazów, nie wystarczy jedynie zaimplementować proces rozpoznawania obrazów, omówiony przeze mnie w poprzednim podrozdziale. Jeszcze przed procesem rozpoznawania, należy obraz pozyskać, przygotować do późniejszego rozpoznawania i dopiero przejść przez samo rozpoznawanie. Schemat systemu rozpoznawania obrazów został przedstawiony poniżej, przejdę więc do omawiania jego poszczególnych elementów.



Rys. 2. Schemat pełnego procesu rozpoznawania obrazów – od akwizycji po otrzymanie decyzji klasyfikacyjnej.

Pierwszym krokiem do osiągnięcia decyzji klasyfikacyjnej (do rozpoznania czy w obrazie znajduje się poszukiwany obiekt i podjęcie decyzji do jakiej klasy obiektów

on należy) jest akwizycja obrazu. Jest to etap często pomijany i bagatelizowany ale niezmiernie ważny. Niestety, nie wystarczy samo zrobienie zdjęcia czy nakręcenie filmu. Należy pamiętać, że obraz, który pozyskamy, posłuży do procesu rozpoznawania – przejdzie więc proces segmentacji, wyliczania cech i klasyfikacji. Każde zakłócenia wprowadzone przy akwizycji zmniejszają szansę na poprawną klasyfikację, gdyż obliczone cechy zazwyczaj są na nie bardzo czułe. Wystarczy refleks światła w obiektywie, kurz na szkle obiektywu czy poruszenie aparatu podczas robienia zdjęcia, aby pozyskany obraz był pełen zakłóceń. Przyglądając się przedstawionemu schematowi procesu rozpoznawania obrazów można zauważyć, że znajduje się tam etap „poprawy jakości i kompensacji zakłóceń", więc nawet gdy wystąpią zakłócenia to będzie możliwość ich usunięcia, zniwelowania, lecz niestety nie jest to do końca prawda. Nie wszystkie typy zakłóceń są możliwe do usunięcia, ponadto kompensacja zakłóceń danego typu najczęściej wzmacnia zakłócenia innego typu. Przykładowo, jeśli otrzymano w procesie akwizycji obraz rozmyty, o nieostrych krawędziach, można próbować nałożyć na taki obraz filtr górnoprzepustowy celem wyostrzenia krawędzi. Efektem będą właśnie wyostrzone krawędzie i wzmocnione szumy, które z pewnością będą miały wpływ na wyliczone cechy szukanego obiektu. Zdarza się, że nawet jeśli aparat będzie stał nieruchomo, dobierzemy odpowiednie oświetlenie i zadbamy o to, aby pozyskany obraz był jak najlepszy, uzyskujemy obraz zaszumiony (z zakłóceniami typu sól/pieprz) na skutek niewystarczającej jakości matrycy (matryca złej jakości, niewystarczająca rozdzielczość matrycy). Pozyskując obraz należy więc zadbać o zniwelowanie zakłóceń: o dobre oświetlenie sceny, upewnienie się czy jakość sprzętu fotograficznego jest wystarczająca do naszych potrzeb.

Drugim etapem, zachodzącym zazwyczaj jeszcze w aparacie czy w kamerze, jest konwersja analogowo-cyfrowa. Jako, że zazwyczaj zachodzi ona w sprzęcie fotograficznym, nie mamy na nią wpływu. Jednak niektóre aparaty udostępniają możliwość zapisywania zdjęć w formacie RAW (surowym) – czyli zapisują bezpośrednie wartości zarejestrowane przez punkty światłoczułe na matrycy. Dzięki temu możemy sami zdecydować o tym, w jakim formacie ostatecznie zapiszemy zdjęcie, jakie kanały usuniemy tak, aby uzyskać jak najlepsze zdjęcie do procesu rozpoznawania.

Kolejnym etapem jest poprawa jakości i kompensacja zakłóceń. Szum i zakłócenia, powstałe w procesie akwizycji obrazu i jego konwersji na postać cyfrową, mogą zostać zredukowane. Tak jak pisałem wcześniej, redukcja szumu jednego typu zazwyczaj wzmaga inne. Na tym etapie należy więc zbadać jaki szum czy zakłócenia, których nie możemy wyeliminować, mogą występować w obrazach, a następnie wybrać

takie algorytmy poprawy jakości, aby skompensować zakłócenia przeszkadzające w rozpoznawaniu i zostawić te, które na rozpoznawanie nie mają dużego wpływu. Typowe zakłócenia to:

- zakłócenia typu sól/pieprz,
- spowodowane geometrią soczewek,
- wynikające ze złych warunków oświetleniowych sceny,
- obrazy prześwietlone/niedoświetlone,
- refleksy światła,
- rozmycie w efekcie poruszenia aparatem fotograficznym.

Zakłócenia typu sól/pieprz to zakłócenia punktowe, które powstają z wielu przyczyn, m.in. w wyniku punktowego zabrudzenia obiektywu, niskiej jakości matrycy światłoczułej, czy zbyt dużej jej czułości. Są to charakterystyczne, zbyt jasne lub zbyt ciemne punkty w obrazie, wyraźnie odróżniające się od barwy sąsiednich pikseli. Jako, że są to małe zakłócenia punktowe, mogą być usunięte poprzez filtrację medianową, filtr dolnoprzepustowy. Te operacje powodują rozmycie obrazu, niwelując punktowe zakłócenia, zabrudzenia. Przy dużej rozdzielczości obrazu wyeliminują zakłócenia i zbytnio go nie rozmyją, mogą więc być przydatne w poprawie jakości przed procesem rozpoznawania obrazów.

Zakłócenia wynikające ze złych warunków oświetleniowych, obrazy prześwietlone/niedoświetlone, występujące refleksy światła są częstym zjawiskiem przy akwizycji obrazu. Można je niwelować poprzez operowanie kontrastem – zdjęcia niedoświetlone należy poprawić poprzez podbicie kontrastu, przy zdjęciach prześwietlonych można natomiast obniżyć kontrast, co powinno poprawić jakość obrazów w sensie ich rozpoznawania. Zakłócenia spowodowane przez refleksy światła czy złe oświetlenie, także mogą być zniwelowane – przeprowadzenie operacji wyrównywania histogramu powinno zgasić miejsca prześwietlone oraz rozjaśnić niedoświetlone fragmenty obrazu. Operacje te są bardzo przydatne przy niwelowaniu tego typu zakłóceń i jeśli nie da się poprawić warunków oświetleniowych, warto rozważyć wbudowanie np. algorytmu wyrównywania histogramu do systemu rozpoznawania obrazów.

Kompensacja zakłóceń może być przeprowadzona także w dziedzinie częstotliwości – po policzeniu transformaty Fouriera o wiele szybciej można przeprowadzać poprawę jakości. Klasyczne algorytmy poprawy jakości, przedstawione pokrótce we wcześniejszych akapitach, wymagają „przejścia” po wszystkich pikselach obrazu

i wykonania dodatkowych obliczeń, co zabiera czas. Policzenie szybkiej transformaty Fouriera dla obrazu i przetwarzanie w dziedzinie częstotliwości, niejednokrotnie okazuje się szybsze niż działanie klasycznych algorytmów. W niniejszej pracy nie będę jednak przetwarzał obrazów w dziedzinie częstotliwości, dlatego nie będę opisywał głębiej tego procesu[2].

Kolejne etapy przedstawione na rys. 2. dotyczą już samego rozpoznawania i zostały przeze mnie omówione w poprzednim podrozdziale. W wyniku tych etapów, system powinien dawać odpowiedź na pytanie „jakie obiekty znajdują się w obrazie?", lokalizując potencjalne obiekty, licząc ich cechy i przekazując je do klasyfikatora. Ten, otrzymując same cechy obiektu, powinien klasyfikować obiekt i oznajmiać do jakiej klasy należy. Klasyfikator może także udzielać odpowiedzi wymijającej, kiedy różnica między kwalifikacją do dwóch lub więcej klas jest zbyt mała, aby nie popełniać błędów.

Podsumowując, system rozpoznawania obrazów to proces złożony z wielu etapów, a na każdym z nich mogą wystąpić zakłócenia i szumy, mające katastrofalny wpływ na liczenie cech obiektu i klasyfikację. Przy budowie systemu rozpoznawania należy przeanalizować wszystkie etapy tak, aby uniknąć niepotrzebnych zakłóceń, dzięki czemu segmentowany obraz, który zostanie poddany procesowi liczenia cech, będzie dobrej jakości i dostarczy odpowiednich cech do klasyfikacji.

3.3 Obliczanie niezmienników geometrycznych

W opisywanej w tej pracy aplikacji, jako systemu cech, użyłem niezmienników momentowych. Opiszę w tym podrozdziale sposób ich obliczania. Używane wzory znajdują się w dodatku A niniejszej pracy. Dwuwymiarowy moment geometryczny rzędu (p, q) uzyskuje się z definicji jako [1], przy dyskretnych współrzędnych obrazu przyjmuje postać [2]. Zakłada się przy tym, że kształt do rozpoznania jest barwy czarnej, tło jest białe lub odwrotnie, zależnie od przyjętej konwencji. Obraz, będący wynikiem segmentacji, powinien zawierać tylko obiekt oraz tło. Ma być sprogowany przed obliczaniem momentów geometrycznych tak, aby wyraźnie odróżniać tło od obiektu. Z użyciem momentów geometrycznych można obliczyć momenty centralne [4]. Przy ich obliczaniu stosuje się wzory na centrum obrazu [3]. Znając sposób obliczenia momentów centralnych, można przejść do obliczeń niezmienników momentowych, będących naszymi cechami. Wzory

[2] R. C. Gonzales, R. E. Woods, *Digital Image Processing*, New Jersey 2002, s. 148.

[5] i [6] ukazują przykładowe sposoby obliczenia niezmienników momentowych M1 i M2 - dwóch z dziesięciu cech[3] wykorzystywanych przy klasyfikacji obiektów.

Po przeanalizowaniu wzorów [2] i [4] łatwo zauważyć, że w celu wyliczenia zespołu cech, należy przejść przez obraz piksel po pikselu, wykonując przy tym niezbyt skomplikowane obliczenia. Na szczęście na tym etapie rozmiary fragmentów obrazu, dla których liczone są cechy, charakteryzują się niewielkimi rozmiarami. Cechy wyliczane są dla niewielkich fragmentów obrazu, w których znajduje się pojedynczy znak. Dzięki temu proces rozpoznawania cech nie jest procesem czasochłonnym i nie wydłuża zanadto czasu przetwarzania pojedynczej klatki.

4. Sprecyzowanie problemu

Przed projektowaniem i implementacją programu, przeprowadziłem poszukiwania istniejących na polskim runku systemów, w celu porównania różnych rozwiązań i znalezienia wskazówek oraz parametrów, które mógłbym zawrzeć w założeniach projektowych. Opisałem też wymagania, które wynikowa aplikacja powinna spełniać. Identyfikator użyty w spisie wymagań, służy jednoznacznemu określaniu wymagania, celem możliwości odwołania się do niego z późniejszych rozdziałów niniejszej pracy i składa się z oznaczenia kodowego WF lub WN (wymaganie funkcjonalne, wymaganie niefunkcjonalne) oraz z kolejnego dwucyfrowego numeru wymagania, co jednoznacznie określa dane wymaganie.

4.1. Przegląd istniejących systemów

W wyniku poszukiwań systemów wizyjnych rozpoznawania tablic rejestracyjnych na polskim rynku, znalazłem kilka interesujących pozycji. Poniższy spis prezentuje część znalezionych systemów, wyszczególniając charakterystyczne informacje dla danego systemu, mogące być pomocne w implementacji własnego rozwiązania problemu rozpoznawania tablic rejestracyjnych. Przy każdym systemie widnieje także link do strony jego producenta, pod którym można znaleźć aktualne informacje o cenach systemów, ich możliwościach i ograniczeniach.

1. System ARTR - http://www.metasoft.pl/pl/artr.html
   - Czas rozpoznawania pojedynczej tablicy < 100ms;
   - Skuteczność rozpoznawania (dla pojazdu statycznego) >96%;

[3] R. Tadeusiewicz, *Systemy wizyjne robotów przemysłowych*, Warszawa 1992, s. 157.

- Format wideo – PAL, NTSC;
- Szerokość obrazu w miejscu detekcji – ok. 2,5m;

2. System POLVISION - http://www.polvision.com.pl/offer-l.asp?lang=pl
   - Czas rozpoznawania pojedynczej tablicy – ok. 200ms;
   - Skuteczność rozpoznawania – ok. 95%;
   - Oparty o sieci neuronowe;
   - System pozwala na obsługę ponad 1000 kamer;
3. System LPR INNEX - http://innex.pl/28/system_odczytu_tablic_rejestracyjnych_lpr
   - Czas rozpoznawania pojedynczej tablicy – ok. 40ms;
   - Skuteczność rozpoznawania – 99,9% (potwierdzone laboratoryjnie i praktycznie);
   - Odporny na zabrudzenia tablicy i warunki atmosferyczne;
   - Umożliwia identyfikację pojazdów w ruchu;
4. System LuxriotLPR - http://adiglobal.pl/luxriot_lpr.html
   - Brak danych co do czasu rozpoznawania pojedynczej tablicy;
   - Skuteczność rozpoznawania – 99,8% (w aplikacji parkingowej);
   - Rozpoznawanie tablic w ruchu, do 60km/h;
   - Zawiera algorytmy rozpoznawania zaszumionych obrazów;
5. System GV-LPR-1 - http://www.geovision.pl/produkty/identyfikacja-tablic/gv-lpr-1.htm
   - Czas rozpoznawania pojedynczej tablicy < 200ms;
   - Skuteczność rozpoznawania – do 99% (z wielu kamer);
   - Może być sterowany sygnałami z czujników;
   - Obsługuje wideo 30kl/s;

Wszystkie znalezione systemy są komercyjne, według danych producentów charakteryzują się skutecznością rozpoznawania powyżej 95% oraz czasem analizy jednej tablicy poniżej 200ms (niektóre deklarują szybciej). Wszystkie systemy są reklamowane jako umożliwiające rozpoznawanie pojazdów w ruchu, ale skuteczność rozpoznawania podawana jest zazwyczaj dla pojazdu stojącego lub dla wolno poruszającego się, lecz analizowanego przez wiele kamer. Szczegóły dotyczące rozdzielczości kamery rejestrującej analizowany obraz, czy prędkości przesyłanych obrazów, w większości przypadków są niejasne, jedynie jeden z systemów deklaruje format wideo PAL/NTSC, a inny podaje prędkość rejestrowania klatek jako 30kl/s. Szczegóły systemu, jego budowy nie są podawane przez producentów, jedynie w większości znalezionych systemów producent deklaruje przechowywanie danych w relacyjnej bazie danych.

W oparciu o zgromadzone dane istniejących systemów i ich parametry, zdefiniowałem następnie wymagania dotyczące aplikacji powstałej przy niniejszej pracy. Przedstawione zostały one w poniższych podrozdziałach w formie tabelarycznej tak, aby były przejrzyste dla czytelnika.

4.2 Wymagania funkcjonalne

Aplikacja, będąca wynikiem niniejszej pracy, w pierwszej kolejności ma być funkcjonalna. Interfejs użytkownika ma być interfejsem graficznym. Użytkownik powinien za jego pomocą być w stanie przede wszystkim analizować i rozpoznawać znaki ze strumienia wideo. Powinien także móc bezproblemowo korzystać z poszczególnych funkcjonalności aplikacji. Ponadto aplikacja ma spełniać poniższe wymagania funkcjonalne:

Id	Nazwa wymagania	Opis wymagania funkcjonalnego
WF01	Otwieranie strumieni wideo	Aplikacja ma mieć możliwość otwierania strumienia wideo z pliku wideo zapisanego na nośniku lub z kamery podłączonej do komputera za pomocą portu USB (lub wbudowanej – np. laptop).
WF02	Analiza strumienia wideo	Po wybraniu strumienia wideo i otworzeniu go, program ma umożliwiać analizę strumienia wideo w celu poszukiwania tablic rejestracyjnych, a także rozpoznawanie znalezionych tablic.
WF03	Przeglądanie wyników analizy i rozpoznawania	Zlokalizowane i rozpoznane tablice mają być zebrane w kolekcji w pamięci, z możliwością wybrania i obejrzenia pozycji kolekcji przez użytkownika.
WF04	Usuwanie wyników analizy i rozpoznania	Użytkownik ma mieć możliwość usuwania pozycji z kolekcji rozpoznanych tablic po procesie rozpoznawania.
WF05	Wyodrębnianie znaków z rozpoznawanych tablic	Użytkownik ma mieć możliwość wyodrębnienia pojedynczych znaków z rozpoznanych tablic. Wyodrębnione znaki zostaną zaprezentowane użytkownikowi jako kolekcja.
WF06	Pomoc użytkownika	Aplikacja ma udostępniać wskazówki dla użytkownika, pomocne w użytkowaniu aplikacji i opisujące jej poszczególne funkcje.
WF07	Baza wiedzy	Program ma przechowywać bazę wiedzy – kolekcję faktów, które są podstawą do utworzenia klasyfikatora wykorzystywanego w aplikacji.
WF08	Spójność bazy wiedzy	Baza wiedzy ma być spójna – przy uruchamianiu programu baza ma być wczytywana do pamięci, przy zamykaniu programu baza ma być zapisywana z pamięci na dysku.

WF09	Wprowadzanie nowych faktów do bazy wiedzy	Użytkownik ma mieć możliwość, po wcześniejszym wyodrębnieniu znaków z rozpoznanych tablic (WF05), wprowadzenia wybranych znaków (ich cech) jako faktu do bazy wiedzy. Użytkownik nie może mieć możliwości wielokrotnego dodawania do bazy wiedzy jednego faktu, gdyż mogłoby to zaburzyć działanie klasyfikatora powstającego z bazy wiedzy. Próba dodania jeszcze raz faktu istniejącego w bazie wiedzy powinien być sygnalizowany komunikatem.
WF10	Interaktywne uczenie systemu	Aplikacja ma być interaktywna – powinna pozwalać na dodawanie i usuwanie próbek z bazy wiedzy przez użytkownika.
WF11	Opcje analizy i rozpoznawania tablic	Dostępne dla użytkownika mają być opcje, mające wpływ na lokalizację i rozpoznawanie tablic rejestracyjnych. Użytkownik ma mieć możliwość sterowania procesami lokalizacji i rozpoznawania, wybierania cech próbek, które biorą udział w klasyfikacji. Działanie opcji ma być wyjaśnione w pomocy użytkownika (WF06).
WF12	Interaktywny wykres zależności cech	Dostępny ma być wykres, prezentujący próbki z bazy wiedzy rzutowane na płaszczyznę. Użytkownik dzięki analizie rozmieszczenia próbek ma mieć możliwość włączania i wyłączania niektórych cech celem poprawienia jakości klasyfikacji.
WF13	Indywidualizacja opcji	Użytkownik ma mieć możliwość indywidualizacji niektórych parametrów, np. domyślnego katalogu otwierania plików wideo.
WF14	Informowanie o błędach	W wyniku wystąpienia sytuacji awaryjnej, błędu, program ma informować użytkownika zwięzłym i zrozumiałym komunikatem o zaistniałej sytuacji. Jeśli to potrzebne, ma informować krokach, jakie musi podjąć użytkownik, aby naprawić błąd.

4.3 Wymagania niefunkcjonalne

Obok funkcjonalności, które aplikacja powinna posiadać, stoją wymagania niefunkcjonalne, zebrane w poniższej tabeli:

Id	Nazwa wymagania	Opis wymagania niefunkcjonalnego
WN01	Obsługiwane formaty plików wideo	Aplikacja ma obsługiwać następujące formaty plików: 3GP, AVI, FLV, M4V, MJPEG, MOV, MP4, MPEG, OGG, RAWVIDEO, VOM, XWMA.
WN02	Czas analizy jednej klatki filmu	Czas analizy jednej klatki filmu musi być na tyle mały, aby użytkownik nie czekał zbyt długo na kolejną klatkę, niestety analiza i rozpoznawanie tablic przedłuża proces wyświetlania jednej klatki. Przyjąłem prędkość odtwarzania minimum 5 klatek na sekundę jako kompromis dający wrażenie płynności filmu (maksymalnie 200ms na przetwarzanie jednej klatki).
WN03	Przechowywanie bazy wiedzy na dysku	Zbiór faktów będący źródłem do tworzenia klasyfikatora ma być przechowywany na dysku, wybrałem format XML jako w miarę wygodny do ewentualnej edycji.
WN04	Przechowywanie opcji na dysku	Opcje użytkownika nie powinny resetować się podczas restartu aplikacji, ale być zapamiętywane – przyjęto zapisywanie opcji na dysk w formacie XML przy zamykaniu aplikacji.
WN05	Rozdzielczość przechwytywanego obrazu	Założyłem rozdzielczość 640x480px jako optymalną do rozpoznawania. Czas przetwarzania klatki o takim rozmiarze przez aplikację powinien mieścić się w wymaganiu WN02, jednocześnie taka rozdzielczość powinna dostarczać wystarczającej ilości informacji do rozpoznania znaków na tablicy przez program.
WN06	Rozpoznawane znaki	Rozpoznawane przez system znaki powinny mieścić się w zakresie A-Z;1-0 (duże litery oraz cyfry).
WN07	Postać przechowywania zlokalizowanej i rozpoznanej tablicy	Zgodnie z WF03 rozpoznana przez program tablica ma być przechowywana w kolekcji. Użytkownikowi ma być dostępny jej obraz, rozpoznane znaki w formie obrazów i w formie ciągu znaków, plik wideo (lub kamera) z jakiej pochodzi obraz i czas filmu (godzina dla kamery) rozpoznania tablicy. Przechowywane ma być także położenie środka tablicy jako punktu w obrazie.
WN08	Ograniczenia rozpoznawania tablic	Aplikacja nie będzie w stanie usunąć dużych zanieczyszczeń i zakłóceń z obrazu (takich jak

		refleksy światła, odbicie). Z tego powodu zalecane jest użycie stacjonarne aplikacji (rozpoznawanie obrazu nieruchomego pojazdu) lub użycie aplikacji dla pojazdów poruszających się z niewielką prędkością – taką, aby nie spowodować rozmycia w obrazie.
WN09	Przenośność aplikacji	Aplikacja pisana jest dla wszystkich komputerów z procesorem klasy Core 2 Duo lub wyższej. Chciałbym aby aplikacja działała na najbardziej popularnych systemach operacyjnych (aby była przenośna).
WN10	Środowisko programistyczne	Środowisko programistyczne użyte do utworzenia aplikacji powinno pozwalać na wygodną edycję i kompilację kodu, a także automatycznie pobierać i dołączać potrzebne biblioteki do projektu. Pożądana także byłaby możliwość automatycznego generowania dokumentacji z już powstałego kodu.
WN11	Kamera obsługiwana przez aplikacją	Aby kamera była obsługiwana przez aplikację musi umożliwiać przesyłanie rejestrowanego obrazu w czasie rzeczywistym (tak jak np. kamera internetowa), musi być podłączana przez port USB. Jak wynika z WN05, wskazana rozdzielczość obrazu to 640x480px, dlatego kamera powinna obsługiwać co najmniej taką rozdzielczość. Aplikacja powinna umożliwiać także wybór rozdzielczości kamery, jeśli ta jest zdolna do pracy przy różnych rozdzielczościach.

5. Projekt rozwiązania

W tym rozdziale przedstawię kolejne etapy projektowania aplikacji – od zdefiniowania języka i środowiska programistycznego użytego do tworzenia programu, przez przypadki użycia po ogólny diagram klas, składających się na aplikację. Powstały projekt aplikacji zakończy się procesem implementacji, w wyniku którego zbudowany zostanie prototyp programu.

5.1 Język programowania i środowisko programistyczne

W obecnym świecie informatyki, dostępnych jest wiele języków programowania, z czego każdy posiada wady i zalety oraz jest przeznaczony do wybranych zastosowań. Z powodu ich różnorodności i typowych przeznaczeń, wybranie języka

programowania dla mojej aplikacji nie było łatwe. Głównymi faworytami jest język C++ oraz Java.

C++ charakteryzuje się dość niskopoziomowym podejściem do programowania (w porównaniu do Javy) i posiada bibliotekę OpenCV, przeznaczoną do przetwarzania obrazów i grafiki komputerowej. Jest także językiem obiektowym, co jest bez wątpienia wygodne dla programisty. Biblioteka OpenCV jest wykorzystywana przy większości projektów z zakresu grafiki komputerowej, a dodatkowo uznaje się ją za jedną z najlepszych i najszybszych bibliotek do przetwarzania obrazu, co także jest niewątpliwym plusem wybrania C++ jako języka implementacji.

Java z kolei jest obiektowym językiem programowania wysokiego poziomu. Do uruchomienia wymaga wirtualnej maszyny Javy, która wczytuje pliki **.class* programu, kompiluje je do kodu maszynowego oraz optymalizuje z użyciem kompilacji dynamicznej. Takie rozwiązanie zapewnia przenośność, gdyż pliki **.class* są kompilowane do kodu maszynowego na każdej maszynie oddzielnie, ale z drugiej strony powoduje nieco wolniejsze działanie programu. W Javie istnieje wiele mniejszych bibliotek do tego celu, posiadających swoje plusy i minusy. Dodatkowo mamy w niej możliwość wykonywania bibliotek m. in. w języku C++ przez natywny interfejs Javy (Java Native Interface). Dzięki temu możliwe jest włączenie np. biblioteki OpenCV do projektu Javy. Ponadto, istnieje biblioteka JavaCV, będąca odpowiednikiem biblioteki OpenCV dla C++, posiadająca jednak nieco mniejsze możliwości niż jej odpowiednik dla C++. Java posiada także zestaw narzędzi Open Source, zwiększających wygodę programisty i umożliwiających automatyzację pewnych czynności, np. pobieranie i linkowanie bibliotek (Maven), przeprowadzających testy jednostkowe (JUnit) czy pomagającej w tworzeniu logów aplikacji (Log4J).

Na podstawie wymagań niefunkcjonalnych dla projektu będącego przedmiotem tej pracy WN09 i WN10 potrzebuję środowiska programistycznego umożliwiającego tworzenie przenośnych aplikacji, wygodnych dla programisty oraz posiadających odpowiednie biblioteki dla wybranego języka. Potrzebuję ponadto języka i środowiska, umożliwiającego automatyczną generację dokumentacji. Program w języku Java będzie przenośny, natomiast w C++ będzie szybszy. Jako że zamierzam sam zmierzyć się z algorytmami przetwarzania obrazu i nie używać gotowych klas i metod, wykonanie takiego kodu będzie wolniejsze. Mniej się dla mnie liczy argument szybkości C++, dlatego skłaniam się do wybrania Javy jako języka programowania dla tego projektu. Wybieram Javę, ponieważ jest ona przenośna, posiada wiele bibliotek Open Source, z których można dowolnie korzystać, jest wygodniejsza dla programisty i posiada możliwość uruchamiania

bibliotek z języka C++ z wykorzystaniem Java Native Interface. Spełnia ona wszystkie wymagania postawione na początku tego podrozdziału, ustępując językowi C++ jedynie szybkością działania. Jednak po głębszej analizie można zauważyć, że umożliwi ona także stworzenie systemu przetwarzania pojedynczej klatki w czasie mniejszym niż wynikający z WN02 czas 200ms/klatkę.

Popularnym środowiskiem programistycznym do Javy jest środowisko Eclipse. Jest ono środowiskiem Open Source, posiada szerokie wsparcie i integrację wielu bibliotek oraz umożliwia wygodną edycję, generację i refaktoryzację kodu. Pozwala również na integrację z narzędziem automatyzującym budowę oprogramowania Maven, które zamierzam wykorzystać w projekcie. Także większość narzędzi i bibliotek posiada swoje wtyczki do Eclipse, dzięki którym bez żmudnej konfiguracji możliwe jest szybkie włączenie wybranych narzędzi do projektu.

Na tym etapie celowo nie wspominam w jakiej wersji Javy będę pisał aplikację. Nie będę korzystał z mechanizmów wprowadzonych z niedawno udostępnioną wersją Javy 1.7, dlatego na potrzeby niniejszej pracy wystarczy Java 1.6. Jednak, jako że kolejne wersję Javy są kompatybilne wstecz, wynikiem tej pracy będzie aplikacja zbudowana dla obu wersji Javy, której kompilacja w dwóch wariantach, dzięki narzędziu do automatyzacji budowy aplikacji Maven, jest bardzo łatwa.

5.2 Biblioteki i technologie użyte przy tworzeniu aplikacji

Do napisania omawianej aplikacji będę potrzebował bibliotek umożliwiających otwieranie plików wideo i wyłuskiwanie z nich kolejnych klatek wideo, a także obsługujących kamery podłączone do komputera i również przechwytujące z nich kolejne klatki. Dodatkowo użyję wspomnianego już narzędzia do automatyzacji budowy aplikacji oraz bibliotek do tworzenia logów aplikacji i testów jednostkowych. W celu prezentacji rozłożenia faktów w przestrzeni cech, przydatna będzie także biblioteka umożliwiająca tworzenie interaktywnych wykresów, którą również wykorzystam przy tworzeniu aplikacji. W tym podrozdziale chcę opisać wykorzystywane biblioteki, narzędzia i mechanizmy.

Narzędzie do automatyzacji budowy oprogramowania w języku Java Maven (maven.apache.org/), o którym wspomniałem w poprzednim podrozdziale, jest produktem fundacji Apache i jest rozprowadzany na licencji Apache License. Użycie Mavena sprowadza się do utworzenia i skonfigurowania pliku *pom.xml,* który zawiera model obiektów projektu POM (Project Object Model). Określa on biblioteki, z których projekt ma

korzystać, sposób kompilacji, użytą to kompilacji wersję Javy, sposób generowania pliku META-INF i inne parametry projektu. W pliku POM określa się co ma zostać zbudowane i w jaki sposób, a następnie wywołuje jeden z celów lub jedną z wtyczek – Maven wykonuje resztę za programistę:

- automatycznie pobiera wymienione w pliku POM biblioteki z repozytoriów i włącza je do projektu,
- używa do kompilacji zadeklarowanej wersji Javy,
- przeprowadza napisane przez programistę testy jednostkowe w celu weryfikacji poprawności działania,
- buduje plik wynikowy programu (zawiera wykorzystywane biblioteki lub się do nich odwołuje, zależnie od ustawień).

Maven potrafi także wygenerować automatycznie dokumentację, utworzyć stronę internetową projektu, czy umieścić gotowy projekt w repozytorium zdalnym. Dodatkowo posiada wtyczkę do środowiska programistycznego Eclipse, dzięki czemu automatycznie aktualizuje ustawienia projektu, wynikające z pliku POM (m. in. automatycznie pobiera i dołącza do projektu biblioteki). Przykładowy wpis z pliku konfiguracyjnego POM, wystarczający do automatycznego pobrania biblioteki i dodania jej do projektu, przedstawiam poniżej.

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
    </dependency>
</dependencies>
```

Listing 1. Przykładowy fragment pliku POM.xml deklarujący użycie biblioteki JUnit w wersji 4.11.

Narzędzie kontroli wersji Marcurial (mercurial.selenic.com/) jest typowym międzyplatformowym systemem kontroli wersji, napisanym w większości w języku Python. Wspiera zarówno pliki tekstowe, jak i binarne, podstawowe sterowanie odbywa się z użyciem linii poleceń, ale dostępne są również wersje z graficznym interfejsem użytkownika. Posiada wtyczkę do Eclipse, dzięki czemu możliwe jest wygodne zarządzanie repozytorium z poziomu środowiska programistycznego. W ramach niniejszej pracy założyłem repozytorium zdalne w serwisie bitbucket.org/, który udostępnia prywatne i publiczne repozytoria. Niniejsza praca oraz aplikacja powstała w jej wyniku jest produktem indywidualnej pracy, dlatego nie istnieje konieczność zakładania repozytorium zdalnego

projektu, ale dzięki niemu w razie awarii komputera mogłem kontynuować pracę z innej maszyny.

Biblioteki użyte do utworzenia aplikacji to przede wszystkim biblioteki Xuggler (xuggle.com/xuggler) i Sarxos Webcam-Capture (webcam-capture.sarxos.pl/), użyte odpowiednio do otwierania plików wideo i do obsługiwania kamer podłączonych do komputera. Obie biblioteki są napisane w Javie, posiadają zdalne repozytorium, dzięki czemu ich pobranie i dołączenie do projektu było możliwe poprzez Mavena. Umożliwiają otwieranie strumienia wideo z wymienionych źródeł i przechwytywanie kolejnych klatek filmu celem przetwarzania przez aplikację. Są kluczowymi bibliotekami w opisywanej aplikacji.

Pisząc średniej wielkości aplikację, taką jak opisywana, dobrą praktyką jest pisanie także testów jednostkowych przynajmniej dla kluczowych klas. Pomagają one weryfikować działanie aplikacji i sprawdzać, czy zmiany przeprowadzone w aplikacji nie mają negatywnego wpływu na inne funkcjonalności ukończone wcześniej. W tym celu została wykorzystana biblioteka JUnit (junit.org/), która jest polecana przez wielu programistów i deweloperów aplikacji pisanych w Javie. Integruje się ona z Eclipsem oraz z Mavenem, przy każdej kompilacji sprawdzając czy przeprowadzone zmiany przechodzą napisane testy. Maven uniemożliwia zbudowania aplikacji, jeśli ta nie przeszła testów, co zapobiega pomyłkowemu utworzeniu niedziałającej aplikacji (np. w wyniku szybkich zmian w projekcie, przeprowadzonych przez wiele osób). JUnit wykorzystuje adnotacje języka Java, za pomocą których programista deklaruje przypadki testowe, które mogą być wykonane na życzenie programisty przez Eclipse czy automatycznie przez Mavena. Sposób tworzenia testów prezentuje poniższy przykład:

```
public class SampleTest {
	@Test
	public void addTest() {
		assertTrue(3 + 5 == 8);
	}
}
```

Listing 2. Przykładowy test jednostkowy z wykorzystaniem biblioteki JUnit.

Testując aplikację często znajduje się błędy, które powodują nieprawidłowe jej działanie lub zawieszenie i wypisują komunikat o zgłoszonym wyjątku na standardowe wyjście. Zdarzają się również sytuacje, kiedy śledząc błąd programista chciałby wypisać pewne komunikaty, celem upewnienia się o prawidłowym działaniu fragmentu programu. Wypisywanie komunikatów na konsolę nie jest dobrym pomysłem, ponieważ należy wtedy

zadbać o specjalne formatowanie komunikatu tak, aby był czytelny, a wyłowienie konkretnego komunikatu spośród setek innych informacji jest trudnym i czasochłonnym zadaniem. Z pomocą przychodzi biblioteka Log4J (logging.apache.org/log4j/), umożliwiająca tworzenie logu działania aplikacji i zapisywanie go do pliku. Log4J wymaga dołączenia do projektu (automatycznie przez Mavena) i utworzenia jednego pliku konfiguracyjnego. Użycie biblioteki do logowania usprawnia proces tworzenia logów, gdyż dba o formatowanie pliku wyjściowego (np. automatycznie wstawia datę i czas zdarzenia, klasę czy metodę) i umożliwia logowanie na kilku hierarchicznych poziomach. Dzięki temu możliwe jest ustawienie poziomu logowania, poniżej którego komunikaty logowania są ignorowane – np. przy tworzeniu aplikacji poziom logowania ustawiany jest na DEBUG, przez co większość wpisów zapisywana jest w pliku i możliwe jest śledzenie pracy programu. Po ukończeniu aplikacji przestawia się poziom np. na INFO (informacyjny), dzięki czemu użytkownik nie ogląda szczegółowych komunikatów o działaniu aplikacji, ale np. istotne z punktu widzenia informacje. Log4J jest bardzo wygodnym narzędziem, wspomagającym programistę w procesie programowania, a także po ukończeniu aplikacji w znajdowaniu błędów zgłoszonych przez użytkownika.

Chciałbym jeszcze wspomnieć o bibliotece JAXB (jaxb.java.net/) służącej do mapowania plików **.xml* do klas Javy. Od wersji 1.6 Java posiada wbudowaną bibliotekę JAXB. Mapowanie plików **.xml* do klas Javy stosuje się przy zapisie np. kolekcji obiektów do pliku **.xml*. Biblioteka konwertuje wtedy obiekt klasy mapowanej wraz z jego polami na tagi xml (lub odwrotnie) i umożliwia zapis kolekcji do/z pliku **.xml*. Aby JAXB poprawnie mapował obiekty klasy do reprezentacji xml, wystarczy dodanie do klasy Javy i jej pól specjalnych adnotacji. Poniżej przykład fragmentu klasy Javy z adnotacjami oraz pliku xml wygenerowanego po mapowaniu obiektu na plik **.xml*.

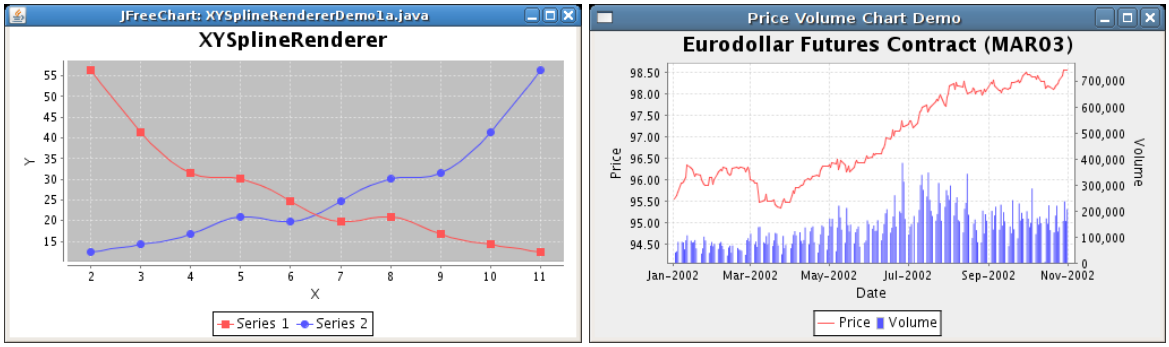
```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Options
{
	@XmlElement
	public int maxShift;

	@XmlElement
	public boolean learningMode;
}
```

```
<options>
<maxShift>10</ maxShift>
<learningMode>true</learningMode>
</options>
```

Listing 3. Prezentacja mapowania klasy i jej pól do reprezentacji xml z użyciem biblioteki JAXB.

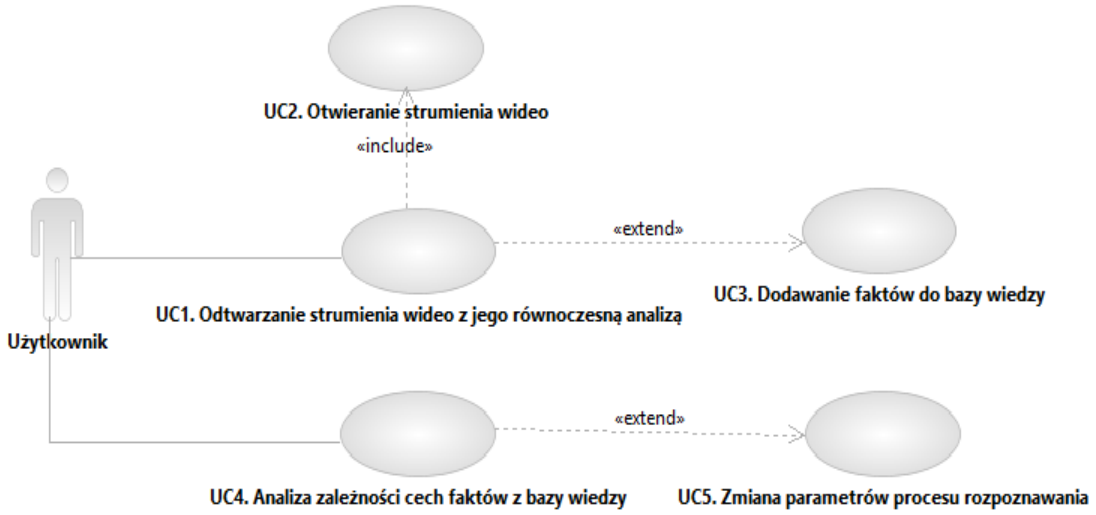
Ostatnią biblioteką, którą chciałbym opisać, jest biblioteka do tworzenia interaktywnych wykresów JFreeChart (jfree.org/jfreechart/). Użyję jej do graficznej

prezentacji zależności między parami cech tak, aby możliwe było ich porównanie i wybranie cech najlepiej rozróżniających rozpoznawane obiekty. Przedstawiona biblioteka jest także biblioteką Open Source, która po dostarczeniu jej danych umożliwia tworzenie różnych typów wykresów – od wykresów punktowych, liniowych, poprzez histogramy, wykresy słupkowe, na wykresach kołowych kończąc. Wykresy generowane przez bibliotekę obsługują zmianę skali, przybliżanie, oddalanie, zapisywanie wykresu jako obrazu oraz drukowanie, jako czynności wywoływane akcjami myszy komputerowej. Dzięki temu użytkownik ma możliwość obejrzeć prezentowane dane w sposób interaktywny, a także zmienić ich wygląd, zapisać czy wydrukować. Przykłady wykresów generowanych przez bibliotekę przedstawiam poniżej.



Rys. 3. Przykładowe wykresy wygenerowane z użyciem biblioteki JFreeChart (pobrane z oficjalnej strony biblioteki).

5.3 Przypadki użycia



Rys. 4. Diagram przypadków użycia dla projektowanej aplikacji.

Po analizie wymagań przeprowadzonej w poprzednim rozdziale, decyzji o języku programowania, środowisku programistycznym i bibliotekach użytych

do napisania aplikacji, przejdę do zdefiniowania przypadków użycia programu. Pomogą one usystematyzować wymagania funkcjonalne aplikacji oraz zaprojektować klasy, przedstawione w następnym podrozdziale. Użyte notacje odnoszą się do diagramu przypadków użycia, przedstawionego poniżej. Prezentuję tylko jeden diagram, ponieważ docelowo jedynym aktorem aplikacji jest użytkownik – nie ma logowania na konto i podziału na role.

UC1. Odtwarzanie strumienia wideo z jego równoczesną analizą.
Scenariusz główny:

1-5 jak w scenariuszu głównym UC2,
6. Użytkownik zatwierdza odtwarzanie strumienia wideo,
7. System odtwarza strumień, jednocześnie lokalizując i rozpoznając tablice,
8. System wyświetla informację o zakończeniu odtwarzania strumienia wideo.

Scenariusz alternatywny1 (przerwanie odtwarzania przez użytkownika):

1-7 jak w scenariuszu głównym,
8. Użytkownik przerywa odtwarzanie strumienia wideo,
Dalej jak w scenariuszu głównym, punkt 8.

UC2. Otwieranie strumienia wideo.
Scenariusz główny:

1. *System pokazuje wybór możliwych źródeł strumienia: z pliku lub z kamery,*
2. *Użytkownik wybiera źródło strumienia wideo,*
3. *System pokazuje parametry otwierania strumienia (plik wideo, kamera, rozdzielczość),*
4. *Użytkownik ustawia parametry i zatwierdza je,*
5. *System weryfikuje wprowadzone parametry i otwiera strumień wideo.*

Scenariusz alternatywny 1:

1-4 jak w scenariuszu głównym,
5. System stwierdza błąd w parametrach i sygnalizuje go stosownym komunikatem,
6. Użytkownik poprawia parametry i zatwierdza je,
Dalej od punktu 5. scenariusza głównego.

UC3. Dodawanie faktów do bazy wiedzy

1-8 jak w scenariuszu głównym UC1,
9. *Użytkownik przechodzi do listy zlokalizowanych tablic i dokonuje selekcji tablic pozostawiając na liście te, na których zostały wysegmentowane wyraźne znaki. Następnie przechodzi do sekcji wprowadzania faktów do bazy wiedzy i uruchamia zebranie znaków wysegmentowanych w procesie lokalizacji i rozpoznawania tablic,*
10. *System zbiera wysegmentowane znaki i umieszcza je na liście,*

11. *Użytkownik wybiera wyraźne znaki z listy i sygnalizuje chęć dodania ich do bazy wiedzy poprzez wciśnięcie stosownego przycisku,*
12. *System wyświetla okno dialogowe z prośbą o wpisanie jednego znaku, do którego zostaną przypisane cechy wysegmentowanego kształtu,*
13. *Użytkownik wprowadza znak i zatwierdza okno,*
14. *System akceptuje wpisany znak i wprowadza nowy fakt do bazy wiedzy.*

Scenariusz alternatywny1:

1-13 jak w scenariuszu głównym,

14. *System odrzuca wprowadzony znak, informując użytkownika stosownym komunikatem,*
15. *Użytkownik czyta komunikat i zamyka go,*
16. *System prosi ponownie o wpisanie jednego znaku,*

Dalej jak w scenariuszu głównym od punktu 13.

Scenariusz alternatywny2:

1-12 jak w scenariuszu głównym,

13. *Użytkownik anuluje wprowadzanie znaku dla wysegmentowanego znaku,*
14. *System zamyka okno dialogowe i ponownie wyświetla listę wysegmentowanych znaków.*

UC4. Analiza zależności cech faktów z bazy wiedzy

Scenariusz główny:

1. *Użytkownik przechodzi do sekcji wykresu zależności cech,*
2. *System wyświetla GUI sekcji wykresu zależności cech,*
3. *Użytkownik wybiera cechy do prezentacji na wykresie i wciska przycisk generacji wykresu,*
4. *System generuje wykres dla zadanych cech i prezentuje go użytkownikowi.*

UC5. Zmiana parametrów procesu rozpoznawania

Scenariusz główny:

1-4 jak w scenariuszu głównym UC4,

5. *Użytkownik po wysnuciu wniosków co do przydatności danych cech przechodzi do sekcji opcji,*
6. *System prezentuje sekcję opcji,*
7. *Użytkownik wybiera cechy używane w procesie rozpoznawania, dostosowuje parametry rozpoznawania do swoich potrzeb,*
8. *System zapisuje zmienione ustawienia.*

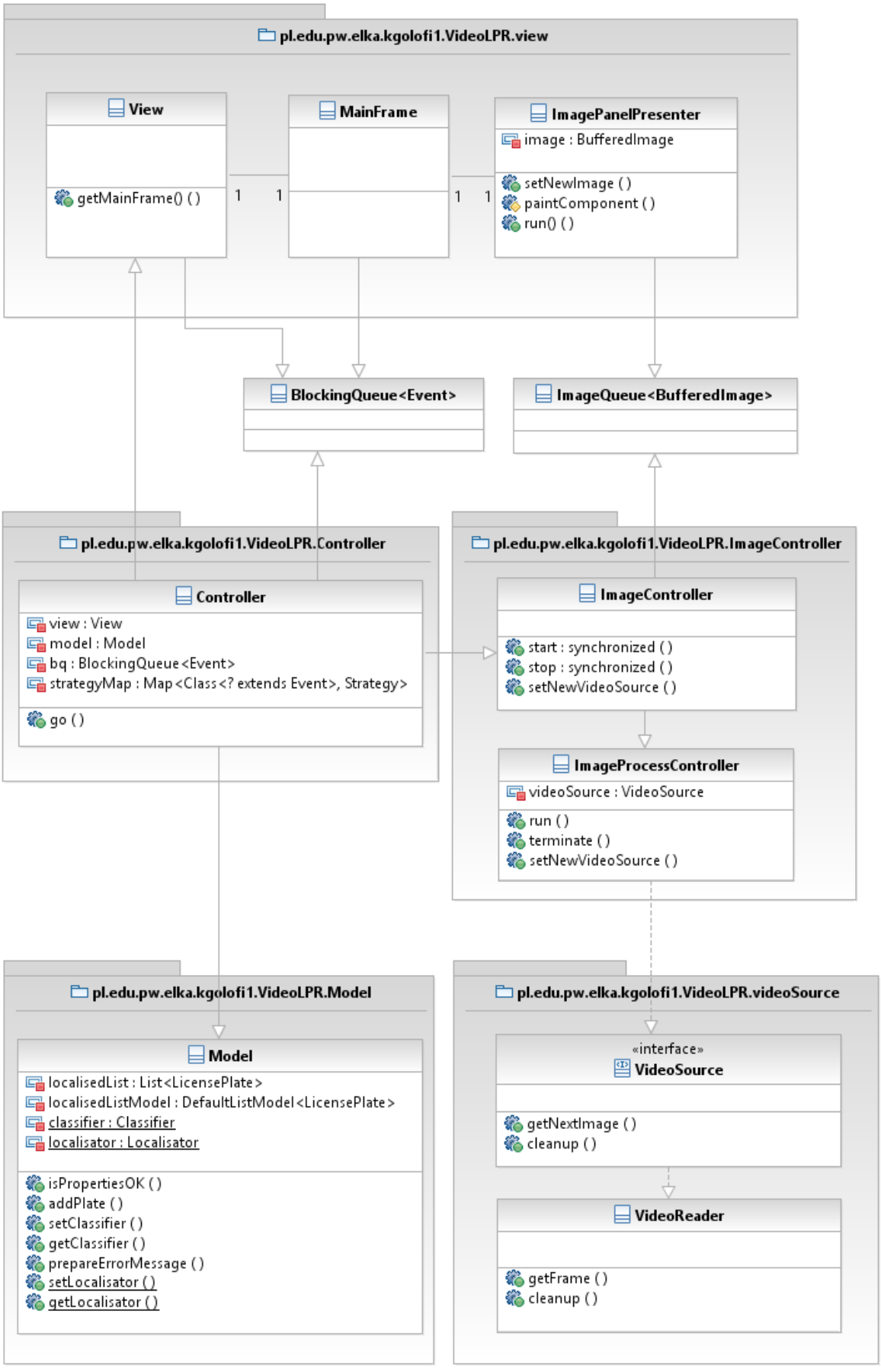
5.4 Ogólny zarys diagramu klas

Przy tworzeniu aplikacji zastosuję zmodyfikowany szablon Model–Widok–Kontroler (Model-View-Controller MVC). Pozwoli on na realizację graficznego interfejsu użytkownika w oddzielnym wątku, zajmującym się tylko nim. Kontroler będzie obsługiwał zdarzenia, a model będzie przechowywał zlokalizowane i rozpoznane tablice.

Ogólny diagram klas projektowanej aplikacji znajduje się na rys. 5. Zastosowanie szablonu MVC umożliwia zbudowanie responsywnego interfejsu użytkownika, który nawet podczas wykonywaniu długotrwałych obliczeń przez program będzie reagował na akcje użytkownika. Jest to możliwe dzięki zdarzeniom generowanym przez klasy widoku i umieszczanym w kolejce blokującej. Zdarzenie jest generowane przez komponenty widoku w odpowiedzi na akcję użytkownika, której wykonanie może długo trwać. Odpowiedni obiekt zdarzenia, po umieszczeniu go w kolejce blokującej, zostaje z niej wyjęty w kolejności FIFO przez kontroler. Ten z kolei, posiadając mapę strategii obsługi zdarzeń, odszukuje odpowiednią strategię i wykonuje ją (obsługuje żądanie użytkownika). Dzięki temu długotrwała akcja odbywa się w oddzielnym wątku, nie blokując interfejsu użytkownika. Krótkotrwałe akcje mogą być wykonywane bezpośrednio w wątku interfejsu, jeśli istnieje gwarancja krótkiego czasu wykonania.

Kontroler obsługuje zdarzenia przychodzące do niego z aplikacji, sterując pracą modelu oraz kontrolera obrazów. Ten z kolei obsługuje żądania kontrolera (głównego), otwierając zadane strumienie wideo i podając kolejne klatki do analizy, rozpoznawania i prezentacji użytkownikowi. Pracą kontrolera obrazów zarządza główny kontroler – to on inicjalizuje, uruchamia i zatrzymuje wątek. Jest on także automatycznie zatrzymywany w sytuacji skończenia się strumienia wideo (np. przeczytania całego pliku wideo).

W przychwytywaniu klatek filmu wykorzystywany jest odtwarzacz wideo (VideoReader). Jest to interfejs, posiadający dwie implementacje – dla przychwytywanie klatek z pliku wideo, oraz kamery. Jego pracę synchronizuje kontroler procesu przechwytywania obrazu (ImageProcessController) poprzez wyliczanie klatki, która powinna aktualnie być prezentowana użytkownikowi. Żądanie zadanej klatki jest przesyłane do otwartego źródła obrazu (VideoSource), który we współpracy z odtwarzaczem wideo (VideoReader) pozyskuje odpowiednią klatkę filmu i zwraca kolejno kontrolerowi obrazu. Ten analizuje ją, rozpoznaje tablice oraz wrzuca rozpoznaną klatkę filmu do kolejki obrazów i odświeża informację o aktualnie znalezionych tablicach.



Rys. 5. Ogólny diagram głównych klas dla projektowanej aplikacji.

Komponenty widoku zebrane są w głównym oknie aplikacji (MainFrame), który nimi zarządza. Jednym z komponentów jest panel prezentujący obrazy (ImagePanelPresenter), którego zadaniem jest sięganie do kolejki blokującej obrazów (ImageQueue) i uaktualnianie wyświetlanego obrazu kolejnymi obrazami wyjmowanymi z kolejki. Panel prezentujący obrazy działa w osobnym wątku, dzięki czemu po jego inicjalizacji nie potrzebna jest żadna ingerencja w jego pracę. Obrazy będą wyjmowane z kolejki i prezentowane przez panel graficzny co średni czas przetwarzania klatki. Klatka przetworzona zostanie natychmiast zaprezentowana, co stworzy złudzenie filmu (sekwencji obrazów). Jak wspomniałem, w pracy tej będę dążył do osiągnięcia czasu przetwarzania klatki rzędu 200ms (5 klatek na sekundę). Taka prędkość powinna być wystarczająca aby stworzyć złudzenie ciągłości filmu.

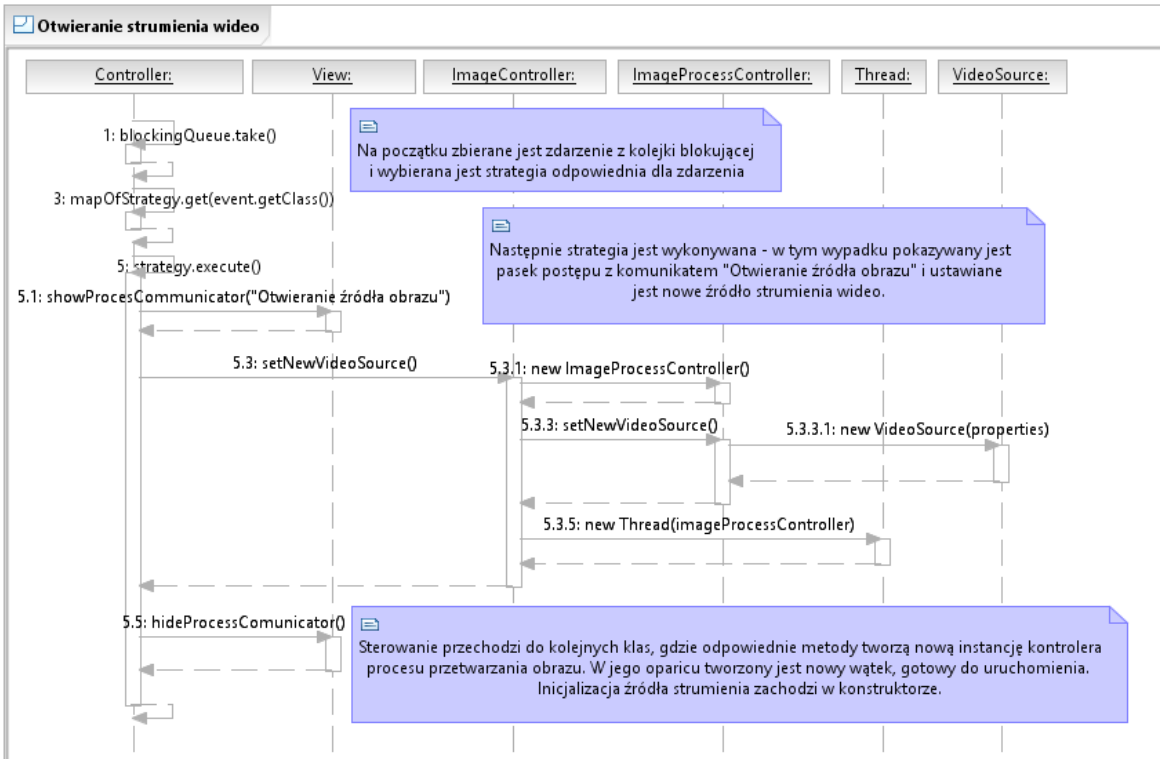
Przedstawiony diagram klas prezentuje ogólną architekturę tworzonej aplikacji. Widoczny jest ogólny przepływ danych i zależności między klasami. Dokładny diagram klas jest zbyt rozległy aby go prezentować w niniejszej pracy, zawiera bowiem informacje o wszystkich klasach użytych w aplikacji. Programowanie w języku programowania Java z użyciem biblioteki SWING do tworzenia graficznego interfejsu użytkownika charakteryzuje się dużą objętością kodu i ilością klas poświęconych interfejsowi użytkownika. Podobnie w niniejszej pracy, diagram klas samego interfejsu użytkownika nie mieści się na stronie A4, dlatego nie zostanie zaprezentowany.

Pełny spis klas i ich metod, wraz z opisem znajduje się na płycie z aplikacją. Dokumentacja została wygenerowana poprzez narzędzie Javadoc, dostępna jest jako lokalne pliki **.html*. Klasy odpowiedzialne za realizację systemu rozpoznawania obrazów, oraz implementujące struktury danych są opisane w dodatku B niniejszej pracy.

Przykładowy przepływ sterowania, omawiany w tym podrozdziale zostanie zaprezentowany w podrozdziale następnym, za pomocą diagramu sekwencji wybranej akcji.

5.5 Wybrany diagram sekwencji

Aby przedstawić przepływ sterowania zdarzeń interfejsu użytkownika, przedstawię i pokrótce opiszę diagram sekwencji procesu otwierania strumienia wideo. Przedstawia on działanie mechanizmu otwierania strumienia ze zdefiniowanych źródeł. Notatki widoczne na schemacie pomogą w zrozumieniu komunikacji wewnątrz programu.



Rys. 6. Diagram sekwencji otwierania strumienia wideo, wraz z opisem działania.

Opis diagramu widoczny jest na rys. 6. przedstawiającym go wraz z notatkami opisującymi działanie. Graficzny interfejs użytkownika, po akcji użytkownika tworzy nowy obiekt zdarzenia, w zależności od akcji użytkownika. Zdarzenie zostaje wrzucone do kolejki blokującej zdarzeń (BlockingQueue), a interfejs użytkownika znów oczekuje na akcje. Sam proces tworzenia obiektu i dodania go do kolejki jest bardzo szybki, dlatego interfejs pozostaje cały czas responsywny.

Dalszą kontrolę nad zdarzeniem przejmuje kontroler (Controller). Pobiera on obiekt zdarzenia z kolejki i z prywatnej kolekcji strategii zdarzeń wybiera właściwą strategię dla pobranego obiektu zdarzenia, a następnie wywołuje ją. W przedstawianym przykładzie następuje wywołanie metod klasy widoku i kontrolera obrazów, które kolejno wyświetlają komunikat pokazujący, że operacja jest w toku, tworzą nową instancję kontrolera procesu przetwarzania obrazu wraz z nowym źródłem strumienia wideo i po otworzeniu źródła chowają komunikat dla użytkownika wyświetlony na początku sekwencji. Jak już wspomniałem, podczas długotrwałego otwierania strumienia wideo (otworzenie strumienia kamery czy dekodowanie pliku wideo), interfejs jest cały czas responsywny, dzięki czemu użytkownik może przesuwać czy zmieniać rozmiar okna aplikacji, nie powodując jej zawieszenia.

6. Realizacja systemu rozpoznawania

System rozpoznawania obrazów, omówiony przeze mnie w rozdziale drugim, został zaimplementowany w języku Java. W tym podrozdziale opiszę jakie elementy systemu rozpoznawania i w jaki sposób zostały zrealizowane. Omówię podejmowane decyzje projektowe i uzasadnię je, opiszę zasadę działania zaprojektowanego systemu.

6.1 Akwizycja i segmentacja

W tworzonym systemie nie mamy większego wpływu na proces akwizycji obrazu. Pozyskiwany obraz pochodzi ze strumienia wideo, który został zapisany do pliku wideo lub jest aktualnie przechwytywany z kamery. Mogę jedynie opisać warunki, w których film będzie rejestrowany, niezbędne do uzyskania pożądanego efektu. Pozwoli to przyszłemu użytkownikowi uniknąć błędów podczas rejestracji sekwencji obrazów, a więc uzyskać obraz niezaszumiony. Rodzaje szumów i ich typowe przyczyny opisałem w rozdziale 3.2, dlatego nie będę ich tutaj powtarzał. Wspomnę jedynie, że obecne tablice rejestracyjne (biała farba, która pokrywa większość z ich powierzchni) bardzo mocno odbija światło, przy rejestracji filmu należy więc zadbać, aby światło nie odbijało się od tablicy i nie trafiało w obiektyw. Takie odbicie może spowodować prześwietlenie całego filmu, a więc wszystkich składowych klatek.

W rozpatrywanym problemie istotne jest zlokalizowane tablicy rejestracyjnej w obrazie. Należy znaleźć takie charakterystyczne cechy, aby pozwoliły one na łatwe znalezienie tablic rejestracyjnych i wykluczenie obiektów, które tablicami nie są. Po wstąpieniu Polski do Unii Europejskiej, krajowe tablice są wykonywane według wzoru unijnego, jednolitego dla całej Unii. Każda jednorzędowa samochodowa tablica rejestracyjna ma określony rozmiar 520x114mm. Składa się z czarnych znaków umieszczonych na białym, silnie odbijającym światło, tle. Po lewej stronie tablicy znajduje się niebieski pionowy pas, na który naniesiony jest symbol państwa, z którego pochodzi pojazd oraz symbol Unii Europejskiej. Zdecydowałem się na filtrację obrazu w poszukiwaniu opisanego niebieskiego pasa. Dodatkowym kryterium jest lokalizacja obszaru białego, który znajduje się zawsze po prawej stronie niebieskiego pasa (zakładam, że tablica nie będzie znajdować się w innym położeniu niż podobnym do poziomego, nie będzie też odwrócona o 180°). Obraz więc będzie segmentowany w następujący sposób: filtrowanie w poszukiwaniu niebieskiego pasa, posiadającego po swojej prawej stronie biały obszar. Jako, że wymiary jednorzędowych tablic samochodowych są jednakowe, dodałem

dodatkowe ograniczenie na stosunek szerokości do wysokości białego obszaru tablicy. Także ilość znaków na tablicy posiada ilość minimalną, która została dodana jako ograniczenie. Te wszystkie ograniczenia powodują, iż spośród bitmapy system rozpoznawania jest w stanie wyodrębnić obszar biały tablicy rejestracyjnej, na której znajdują się znaki. Po znalezieniu białego piksela obrazu, posiadającego po swojej lewej stronie piksel o barwie niebieskiej, obszar biały jest eksponowany z wykorzystaniem algorytmu wypełniania zalewowego, z implementacją kolejki. Początkowo do tego celu użyłem rekursywnej wersji algorytmu, niestety okazała się ona zbyt powolna i pochłaniała zbyt wiele pamięci. Tak wydobyty biały obszar, po spełnieniu opisanych wcześniej wymagań jest uznawany za tablicę rejestracyjną i poddawany segmentacji poszczególnych znaków.

Po wyodrębnieniu białego obszaru tablicy segmentacja znaków jest już prosta – wystarczy przeanalizować tablicę horyzontalnie w poszukiwaniu czarnych obszarów i wyodrębnić je. Czarne obszary na białym tle to oczywiście znaki, które w ten sposób zostaną wysegmentowane. W tym miejscu także wykorzystałem wspomniany algorytm wypełniania zalewowego. Początkowo, przez zabrudzenia tablic rejestracyjnych, segmentacja poszczególnych znaków nie działała tak dobrze jak się spodziewałem. Jeśli na tym etapie algorytm natrafi na tablicę zabrudzoną ciemnym kolorem (pokrytą warstwą błota, brudu) nastąpi błędna segmentacja – system rozpoznawania uzna cały obszar zabrudzenia za znak. Większość filmów testowych została przeze mnie nagrana w okresie jesiennym, przez co kurz w połączeniu z deszczem i kałużami powodował wszechobecne zabrudzenia tablic. W celu przeciwdziałania takim przypadkom został przeze mnie zastosowany algorytm wyrównywania histogramu w połączeniu z algorytmem progowania. Dla zabrudzonej tablicy algorytm wyrównania histogramu zwiększał różnicę między skrajnymi kolorami – w tym wypadku rozdzielał barwę białą od czarnej, tło od znaków – a następujące po nim progowanie pozwoliło na uzyskanie znaków wystarczającej jakości do wychwycenia przez segmentację. Jeśli zabrudzenie tablicy było bardzo duże (nagromadzony kurz, błoto), algorytm nie był w stanie odnaleźć tablicy, a co za tym idzie, rozpocząć proces segmentacji znaków. W niniejszej pracy pomijam jednak takie przypadki, broniąc się faktem, że polskie prawo nakazuje poruszanie się pojazdami posiadającymi czytelne i kontrastowe tablice rejestracyjne (bez zabrudzeń).

6.2 Wyznaczanie cech i rozpoznawanie

Po segmentacji znaków na tablicy rejestracyjnej otrzymaliśmy binarne (czarno-białe) fragmenty obrazu, w których obiekt (znak) jest barwy czarnej, natomiast tło jest białe. Pozostało jeszcze wyznaczyć cechy poszczególnych znaków i sklasyfikować je. Zdefiniuję najpierw zestaw cech, opisujących szukane obiekty.

Cechami systemu rozpoznawania mogą być, np. pole obszaru obiektu w pikselach, długość obwodu, średnica obszaru (wszystkie zależne od translacji, obrotu, skalowania) lub np. stosunek kwadratu obwodu do powierzchni, momenty geometryczne czy cały fragment obrazu (wartości pikseli składające się na obiekt). Wybranymi przeze mnie cechami są niezmienniki geometryczne obiektu[4], gdyż przyjmuje się, że dają dobre wyniki przy rozpoznawaniu, a ich obliczenie nie jest zbyt kosztowne. W podanych źródłach bibliograficznych znalazłem wzory obliczania dziesięciu niezmienników momentowych, które wykorzystałem w opisywanej aplikacji. Jak zapisałem w analizie wymagań aplikacji, chcę dać użytkownikowi możliwość włączania lub wyłączania wybranych przez niego cech (z dziesięciu dostarczonych) tak, aby możliwe było sterowanie procesem klasyfikacji.

Po wyznaczeniu cech zlokalizowanych i wysegmentowanych obiektów, muszę jeszcze sklasyfikować każdy z nich (każdy znak), używając jednego z klasyfikatorów. Jak opisałem w podrozdziale 3.1, istnieje wiele klasyfikatorów możliwych do zastosowania. W ramach tej pracy zdecydowałem się wykorzystać jeden z najprostszych – klasyfikator minimalno-odległościowy. Liczy on odległość do wszystkich sklasyfikowanych znaków w bazie wiedzy (do ich cech), wybiera najbliższy i jako odpowiedź zwraca znak przypisany do najbliższych cech. W aplikacji użyłem zwykłej odległości Euklidesowej do wyznaczania odległości między obiektami, klasyfikator nie bada sąsiedztwa najbliższego punktu – wyznacza tylko najbliższą odległość i zwraca odpowiedź. Jest to bardzo prosty klasyfikator, wymagający przejrzenia wszystkich pozycji w bazie wiedzy. W ramach bazy testowej, wprowadziłem ponad 200 sklasyfikowanych obiektów do bazy wiedzy. Nie jest to duża liczba, dlatego przejrzenie wszystkich wpisów nie jest zbyt czasochłonne i nie zabiera cennego czasu analizy jednej klatki filmu. Wprowadzenie ok. 1000 wpisów powinno wystarczyć do klasyfikacji cyfr i liter (36 znaki, ok 30 próbek na znak), jednocześnie nie powodując spadku wydajności przez konieczność przejrzenia bazy wiedzy przy każdej klasyfikacji.

[4] Zob. rozdział 3.3 „Obliczanie niezmienników geometrycznych”.

Omówiony w tym podrozdziale system rozpoznawania został zaimplementowany w ramach niniejszej pracy i wbudowany w schemat MVC opisany w podrozdziale 5.4. Tak jak wspominałem w wymaganiach funkcjonalnych, wprowadzanie próbek do bazy wiedzy jest udostępnione użytkownikowi, który ma możliwość rozpoznawać inne znaki, nie tylko występujące w Polsce (z zaznaczeniem, że tablice muszą być takiego formatu, jak obowiązujący w Polsce – niebieski pas na lewej krawędzi tablicy, czarne znaki na białym tle). Ma także możliwość ustawić kamerę w ustalony przez siebie sposób (np. pod kątem do poruszających się pojazdów) i wprowadzić do bazy wiedzy znaki widziane z innej perspektywy (kamery ustawionej pod kątem). Dzięki temu system spełniający swoje zadanie jest także elastyczny, co pozwala zastosować go do wielu celów. Po dobudowaniu odpowiednich funkcjonalności mógłby służyć jako system obsługi bramek i płatności w parkingu płatnym lub przy rejestracji wolno poruszających się pojazdów (np. na odcinku z ograniczeniem prędkości) celem zebrania danych o miejscach zamieszkania osób przejeżdżających w danym miejscu.

W kolejnym rozdziale przedstawię wygląd aplikacji, opiszę też pokrótce poszczególne ekrany i dostęp do zdefiniowanych w wymaganiach funkcjonalności.

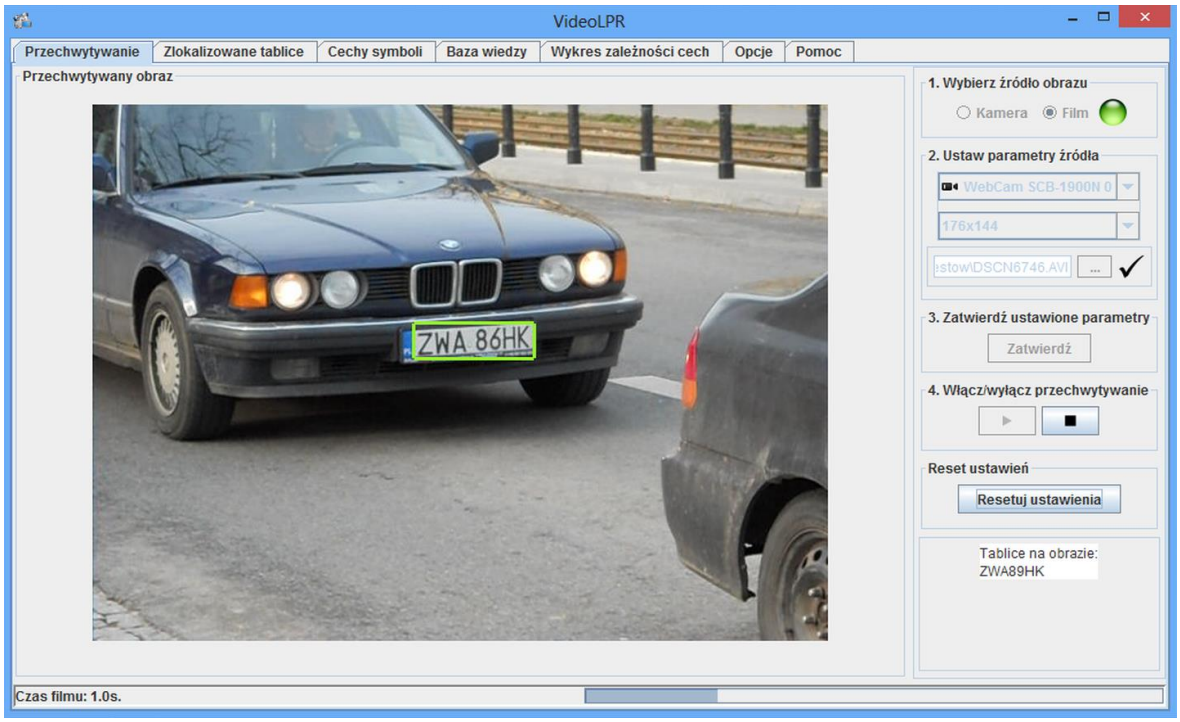
7. Opis interfejsu użytkownika

W tym rozdziale przedstawię wybrane ekrany graficznego interfejsu użytkownika, opiszę działanie ich elementów. Zaprezentuję także, w którym miejscu dostępne są funkcjonalności wyszczególnione wcześniej w wymaganiach funkcjonalnych. Poniższy opis jest niejako skróconą instrukcją użytkownika aplikacji, opisującą położenie i działanie dostępnych w niej akcji.

Rys. 7. przedstawia ekran startowy aplikacji. Umożliwia on wybranie rodzaju źródła strumienia wideo (plik wideo lub kamera), ustawienie jego parametrów (wskazanie pliku wideo, wybór kamery) i otworzenie go. Wszystkie te kroki są dostępne w panelu widocznym po prawej stronie ekranu. Są one ponumerowane, aby ułatwić użytkownikowi ich kolejne wykonywanie:

1. wybór rodzaju źródła strumienia wideo (film lub kamera) w sekcji 1.,
2. ustawienie parametrów źródła – wybór kamery i rozdzielczości lub wskazanie pliku filmowego oraz zatwierdzenie parametrów przyciskiem w sekcji drugiej,

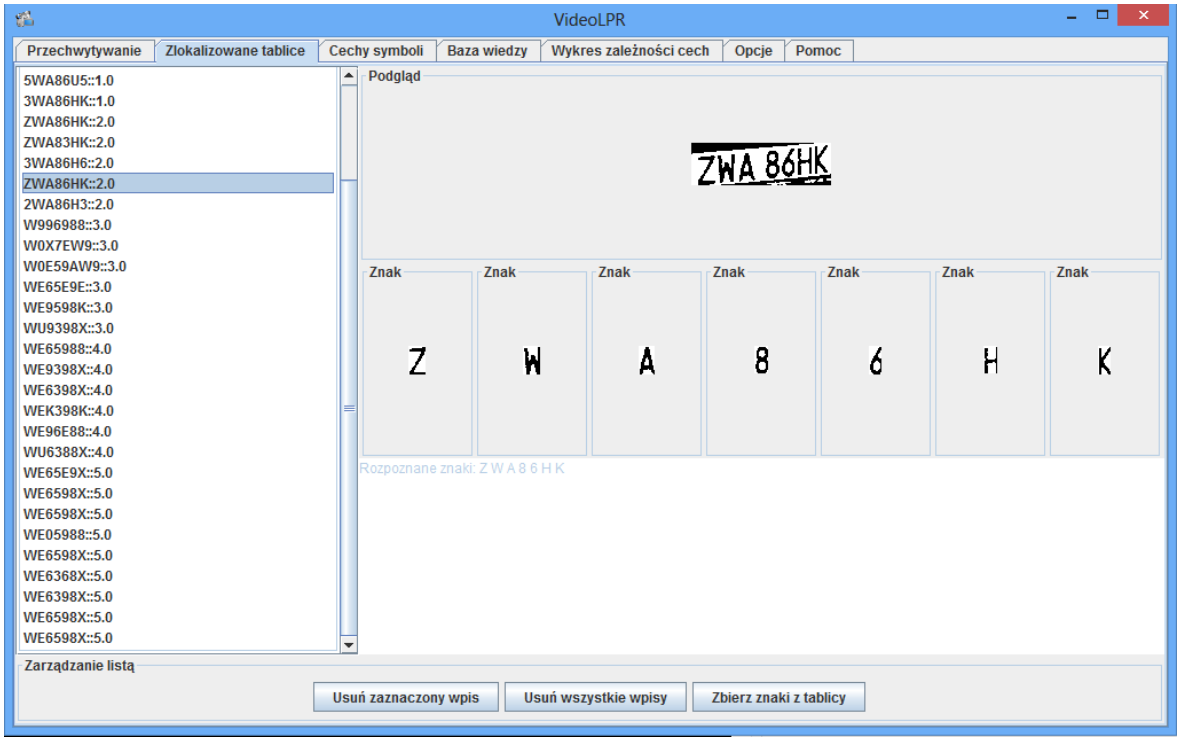
3. otworzenie strumienia wideo, następujące po zatwierdzeniu parametrów źródła - po wykonaniu tego punktu aktywują się przyciski do rozpoczęcia/zakończenia odtwarzania,
4. rozpoczęcie odtwarzania klatek ze strumienia (wraz z ich analizą) poprzez wciśnięcie przycisku „odtwarzaj”, symbolizowanego trójkątem,
5. zakończenie odtwarzania klatek ze strumienia poprzez wciśnięcie przycisku „stop”, symbolizowanego kwadratem,
6. [opcjonalnie] zresetowanie strumienia poprzez przycisk „Resetuj ustawienia” oraz odtworzenie i analiza kolejnych strumieni wideo (należy powtórzyć kroki 1-5),
7. [opcjonalnie] analiza rozpoznanych tablic w zakładce „Zlokalizowane tablice” – opis w kolejnym akapicie.



Rys. 7. Ekran startowy aplikacji, umożliwiający otwierania, analizę i wyświetlanie przeanalizowanego strumienia wideo.

Przy ustawianiu parametrów źródła następuje sprawdzenie poprawności wprowadzonych opcji - błędy w ustawieniach są przekazywane użytkownikowi w formie pokazującego się powiadomienia z komunikatem. Aktywowanie odtwarzania strumienia skutkuje uruchomieniem przechwytywania klatek ze strumienia wideo, analizę w poszukiwaniu tablic, rozpoznawanie i wyświetlenie efektu użytkownikowi. Wynik rozpoznawania każdej klatki jest widoczny poprzez zaznaczenie rozpoznanej tablicy (rozpoznanych tablic) poprzez prostokątny(e) zielony(e) kontrastowy(e) obrys(y)

na obrazie, oraz wyświetlenie w panelu w prawym dolnym rogu rozpoznanych znaków. Ponadto, podczas przechwytywania klatek z filmu wideo na pasku statusu w dolnej części okna pojawia się informacja o aktualnym czasie filmu oraz pasek postępu odtwarzania.



Rys. 8. Ekran aplikacji prezentujący rozpoznane tablice.

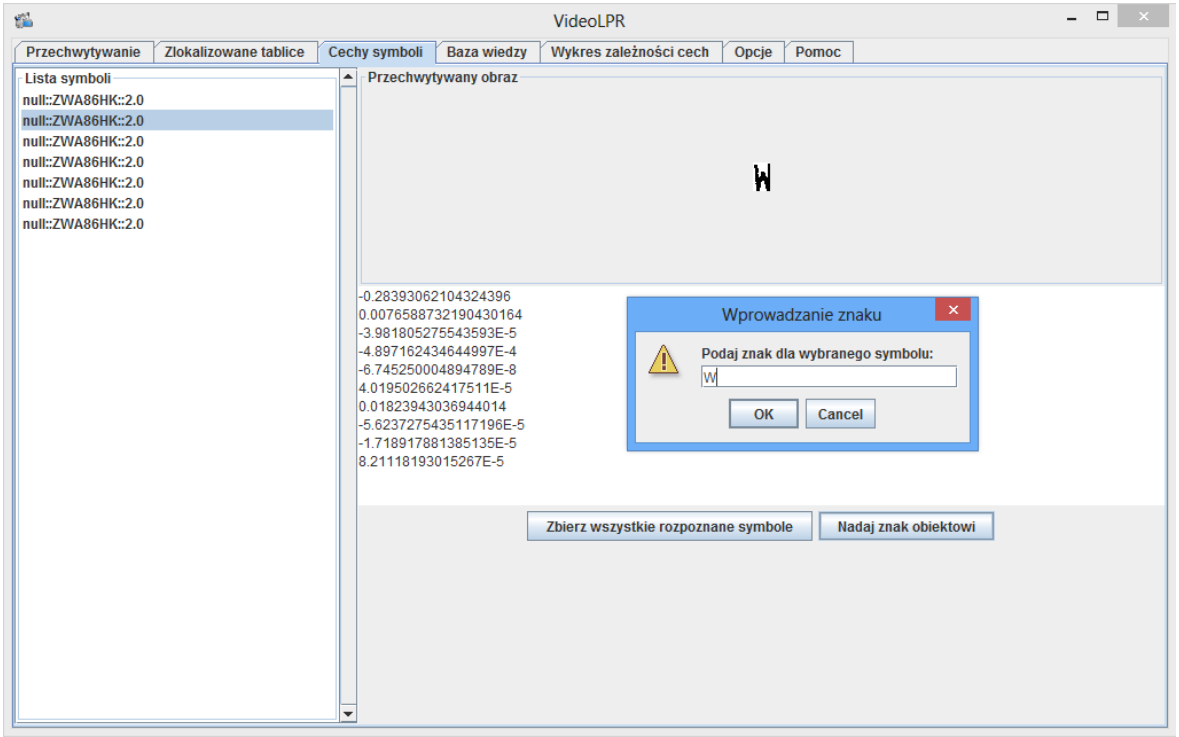
Drugi opisywany ekran jest widoczny na rys. 8. Prezentuje on zakładkę „Zlokalizowane tablice", w której rozpoznawane na poprzednio opisywanym ekranie tablice zebrane są w formie listy, widocznej po lewej stronie. Za pomocą tej zakładki użytkownik ma możliwość:

- obejrzenia dowolnej pozycji na liście, poprzez kliknięcie na nią myszą – skutkuje to otworzeniem podglądu zlokalizowanej tablicy, podglądu znaków na tablicy oraz rozpoznany numer rejestracyjny jako łańcuch znaków,
- zarządzania listą – możliwe jest usunięcie jednej lub wielu pozycji za pomocą przycisków u dołu ekranu,
- zebrania znaków z aktualnie wybranej tablicy i przekazania ich do listy w zakładce „Cechy symboli" poprzez przycisk „Zbierz znaki z tablicy".

Jeśli użytkownik chciałby wprowadzić nowe fakty do bazy wiedzy, służy do tego ekran zakładki „Cechy symboli", widoczny na rys. 9. W nim użytkownik ma możliwość:

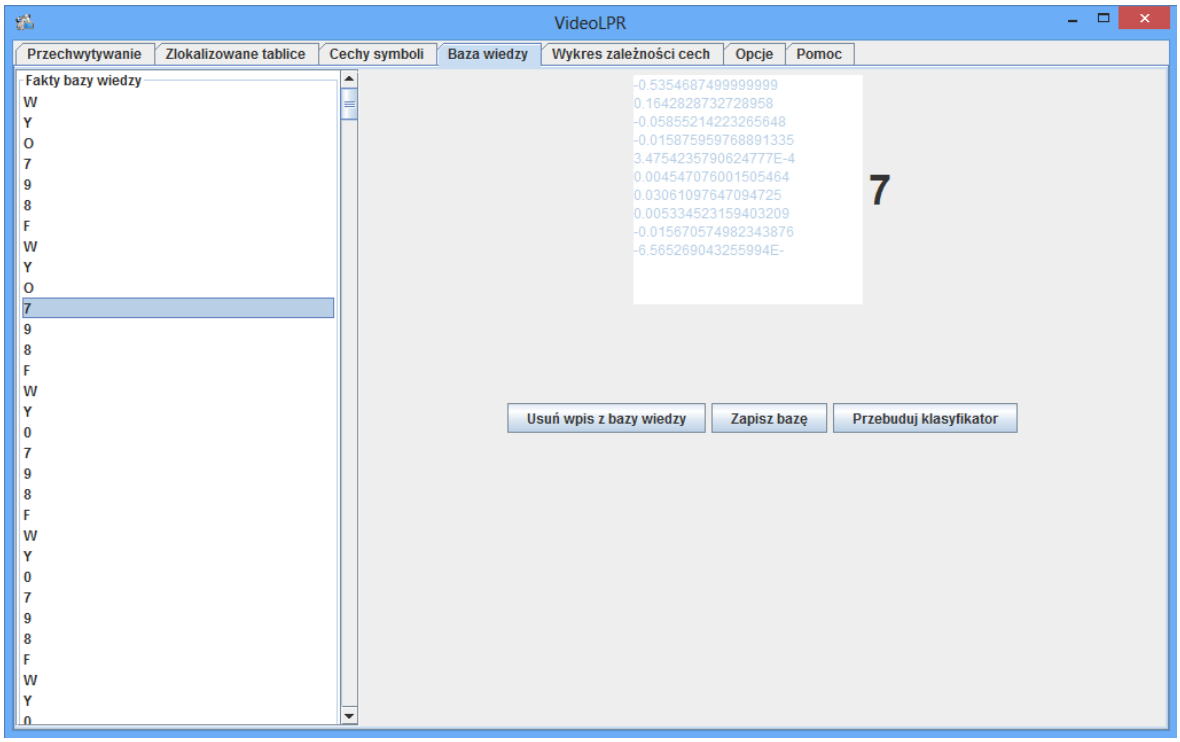
- przeglądać pozycje listy, na której znajdują się znaki pochodzące z segmentacji zlokalizowanych tablic rejestracyjnych – poprzez kliknięcie na pozycji listy,
- wprowadzić na listę w tej zakładce wszystkie znaki z wszystkich tablic znajdujących się na liście w zakładce „Zlokalizowane tablice” za pomocą przycisku „Zbierz wszystkie rozpoznane symbole”,
- wprowadzić zaznaczony znak (a właściwie wyliczone cechy) do bazy wiedzy poprzez przycisk „Nadaj znak obiektowi”.

Aktywacja przycisku „Nadaj znak obiektowi” wyświetla okno dialogowe z prośbą o wpisanie znaku, który zostanie przypisany policzonym cechom. Po wpisaniu znaku i zatwierdzeniu okna dialogowego, zbiór cech wraz z nadanym znakiem zostają zapisane do bazy wiedzy jako nowy fakt.

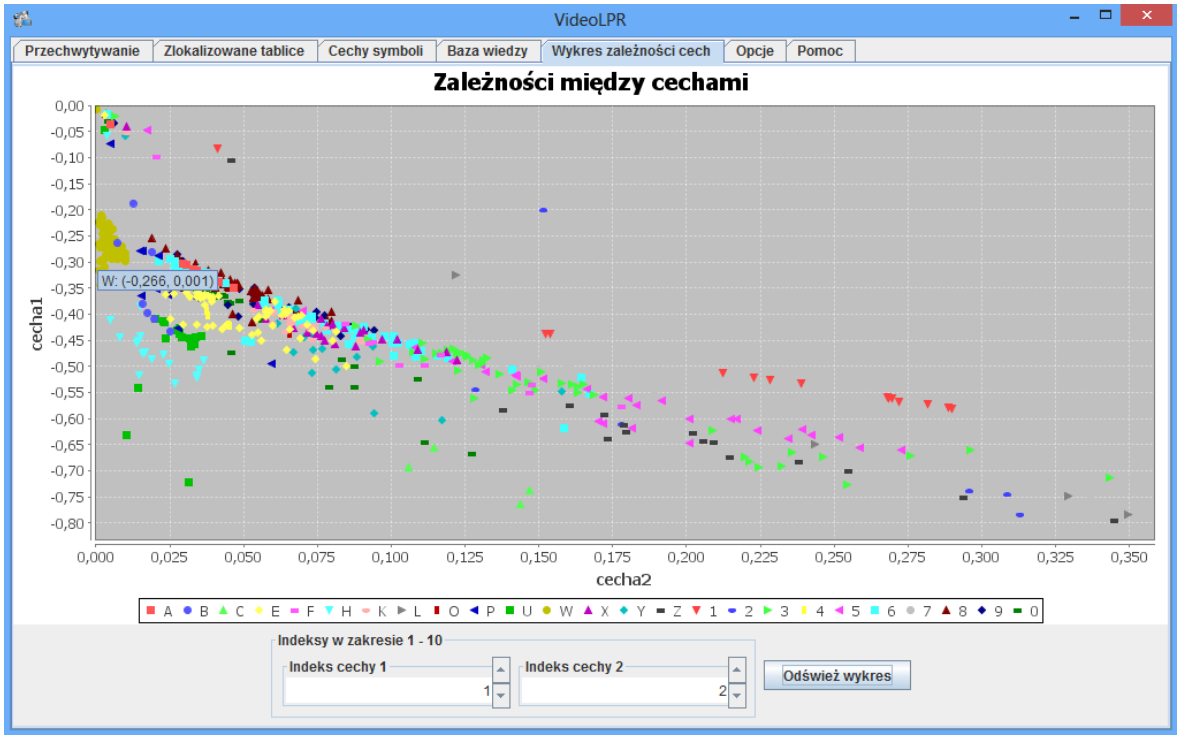


Rys. 9. Ekran aplikacji ukazujący wysegmentowane znaki i dialog dodawania znaku do bazy wiedzy jako faktu.

W kolejnej zakładce „Baza wiedzy” użytkownik posiada podgląd wszystkich pozycji w bazie wiedzy. Widok zakładki przedstawiony jest na rys. 10. Ostatnio dodane pozycje znajdują się na dole listy. Zaznaczenie pozycji na liście skutkuje wyświetleniem cech oraz skojarzonego z nimi znaku. Jeśli użytkownik omyłkowo dodał fakt do bazy wiedzy, może go usunąć poprzez zaznaczenie i przycisk „Usuń wpis z bazy wiedzy”. Przycisk „Zapisz bazę” przeprowadza zapis widocznej kolekcji do pliku **.xml* (zapis taki jest automatycznie przeprowadzany przy zamknięciu aplikacji).



Rys. 10. Ekran aplikacji przedstawiający bazę wiedzy – zbiór faktów wprowadzonych przez użytkownika



Rys. 11. Ekran aplikacji prezentujący wykres zależności cech – narzędzie do analizy przydatności cech w procesie rozpoznawania.

Kolejny opisywany widok to widok zakładki „Wykres zależności cech”, widoczny na rys. 11. Użytkownik ma możliwość generacji wykresu, poprzez wykonanie akcji:

1. wybranie pierwszej cechy do prezentacji na wykresie poprzez wybór wartości w polu „Indeks cechy 1”,
2. analogiczne wybranie drugiej cechy do prezentacji na wykresie,
3. wciśnięcie przycisku „Odśwież wykres”, powodującego generację wykresu dla wybranych cech.

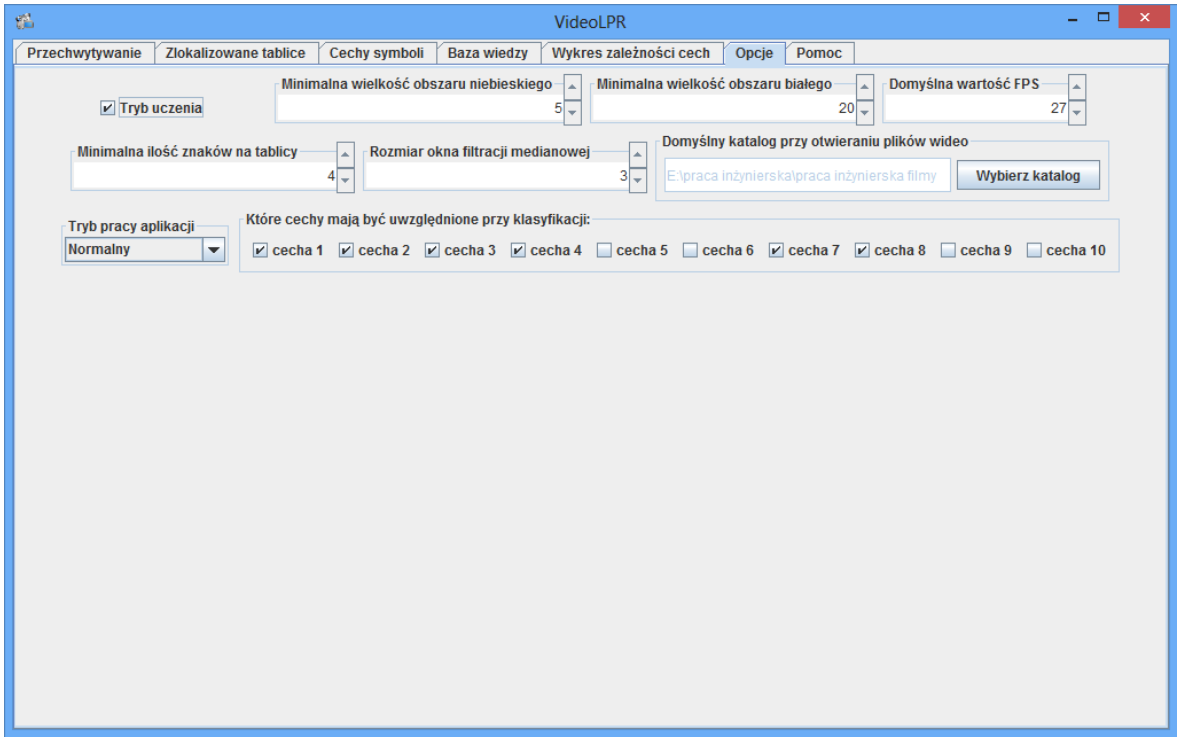
Prezentowany wykres jest interaktywny – użytkownik ma możliwość:

- przybliżenia fragmentu wykresu, poprzez zaznaczenie interesującego go obszaru (zaznaczenie fragmentu z przytrzymanym lewym przyciskiem myszy),
- oddalenia przybliżonego fragmentu wykresu do widoku obejmującego wszystkie punkty, poprzez przytrzymanie lewego przycisku myszy i pionowy ruch myszy ku górze wykresu,
- personalizacji wyglądu wykresu - dokonanie ustawień etykiet i kolorów na wykresie, poprze kliknięcie prawym przyciskiem myszy i wybór „Właściwości” z menu kontekstowego,
- drukowania lub zapisu do pliku wykresu, poprzez kliknięcie prawym przyciskiem myszy i wybór odpowiedniej opcji z menu kontekstowego.

Generowany wykres przedstawia bazę wiedzy, rzutowaną na przestrzeń dwuwymiarową na podstawie wybranych cech. Każdy znak jest zaznaczony innym kolorem i kształtem, opisanym w legendzie. Jest to narzędzie, pomagające użytkownikowi wybranie odpowiednich cech do procesu rozpoznawania znaków. Wybrane przeze mnie do klasyfikacji cechy niestety okazały się niezbyt dobrze dobrane, gdyż dla różnych znaków dają podobne wartości, co w efekcie daje duże prawdopodobieństwo błędnej klasyfikacji (szczególnie przy zastosowanym przeze mnie klasyfikatorze minimalno-odległościowym). Wykres prezentowany na rys. 11. jest jednym z lepiej rozróżnialnych wykresów (widoczne są „grupki” tych samych kolorów, które niestety na siebie zachodzą). Na podstawie porównania wykresów dla różnych cech, użytkownik może określić, które cechy lepiej rozróżniają klasyfikowane znaki (widoczne „grupki” tego samego koloru), a które nie spełniają swego zadania i nie pomagają w rozróżnianiu znaków (brak widocznych „grupek”, wszystkie punkty tworzą jedną, zbitą, grupę). Wybranie tego samego indeksu (porównanie cechy o indeksie X z cechą o indeksie X) prezentuje rozłożenie próbek z bazy wiedzy na prostej (na prostej $y=x$), co także jest pomocne przy określaniu przydatności danej cechy.

Te z cech, które okażą się nieprzydatne w klasyfikacji, użytkownik może wyłączyć w zakładce „Opcje”.

Pozostałe dwie zakładki – „Opcje” i „Pomoc” nie wymagają długiego opisu. Zakładka „Pomoc” zawiera informacje o użytkowaniu aplikacji, instrukcję wystarczającą do obsługi aplikacji przez użytkownika nie posiadającego wykształcenia w kierunku przetwarzania i rozpoznawania obrazów.



Rys. 12. Ekran aplikacji przedstawiający opcje programu.

Zakładka „Opcje” zaprezentowana jest na rys. 12. i zawiera ustawienia programu. Możliwe jest tu – w pewnym stopniu – sterowanie procesem rozpoznawania. Znajdują się tu także ustawienia używania cech – cechy określone jako nieprzydatne mogą zostać wyłączone w opisywanej zakładce. Ponadto, ustawienia aplikacji pozwalają na ustawienie trybu pracy programu (normalny, szczegółowy, debugowania), który ma wpływ na ilość informacji zapisywanych do logu oraz szybkość działania aplikacji. Tryb normalny zapisuje tylko najważniejsze wydarzenia do logu, tryb szczegółowy zapisuje wszystkie przewidziane informacje do logu, a tryb debugowania zapisuje wszystkie informacje do logu programu, oraz każdą klatkę przetwarzania zapisuje na dysk. W trybie debugowania z powodu zapisu klatek na dysk drastycznie spada szybkość analizy i rozpoznawania pojedynczej klatki – czas poświęcony jest na zapis do pliku.

8. Podsumowanie działania aplikacji

Aplikacja, której budowa była celem niniejszej pracy, została zaimplementowana i przetestowana. Oprócz bibliotek opisanych w podrozdziale 5.2 oraz klasy HistogramEQ[5], wszystkie klasy i mechanizmy zostały zaimplementowane przeze mnie. W tym rozdziale chciałbym krótko przedstawić wyniki testów programu.

Test	Rezultat
1. Test szybkości przetwarzania jednej klatki.	Test przeprowadzony dla obrazów przechwytywanych ze strumienia wido o rozdzielczości 640x480px. W zależności od rozmiarów tablicy występującej w obrazie, czas rozpoznawania wahał się. Jego średnia wartość to 115ms. Do testu użyto 13 filmów w formacie **.avi*, o długości od 10-50 sekund.
2. Test jakości rozpoznawania dla filmów biorących udział w uczeniu klasyfikatora.	Test przeprowadzono dla 5 filmów, zapisanych w formacie **.avi,* które były użyte do uczenia klasyfikatora – rozpoznane w nich znaki zostały wprowadzone do bazy wiedzy i na ich podstawie dokonywano klasyfikacji. W takiej konfiguracji jakość rozpoznawania wyniosła średnio około 67%.
3. Test jakości rozpoznawania dla filmów nie biorących udziału w uczeniu klasyfikatora.	Test przeprowadzono dla 5 filmów, zapisanych w formacie **.avi,* które nie były użyte do uczenia klasyfikatora. Przeciwnie niż w teście poprzednim, fakty w bazie wiedzy nie pochodziły z żadnego z testowanych filmów. Biorąc pod uwagę, że nie dla wszystkich znaków występujących w filmach istniały fakty w bazie wiedzy, ogólna jakość rozpoznawania wyniosła 23%. Ogólna jakość dla tego testu wyniosła 11% (zakładając, że wszystkie znaki alfanumeryczne występujące na tablicach mają minimum jeden wpis w bazie wiedzy).
4. Test odtwarzania różnych formatów plików.	Potwierdzone zostało prawidłowe odtwarzanie filmów w formatach *AVI, MP4, MPG*. Za odtwarzanie różnych formatów plików wideo odpowiada biblioteka Xuggler, opisana w rozdziale 5.2.
5. Test zapisywania i odtwarzania danych do plików **.xml*.	Sprawdzone zostało zapisywanie i odczytywanie danych z i do plików **.xml*. Przy każdym uruchomieniu programu opcje użytkownika oraz baza wiedzy są wczytywane z pliku **.xml*, przy zamykaniu programu dane te są prawidłowo zapisywane do plików.
6. Test działania interaktywnego wykresu zależności cech.	Na rys. 11. można zobaczyć efekt działania generatora wykresów. Tworzone wykresy są interaktywne, pozwalają na zmianę prezentowanych cech, przybliżanie/oddalanie prezentowanej przestrzeni, drukowanie czy zapis wykresu. Możliwe jest także personalizowanie wyglądu wykresu przez użytkownika.

[5] Zob. dodatek B – „Opis wybranych klas programu”.

Jak wynika z tabeli, program spełnia wszystkie wymagania funkcjonalne i niefunkcjonalne, przedstawione w rozdziale 4. Z pewnością nie udało mi się przetestować aplikacji w całości, lecz podczas testowania działania aplikacji na filmach testowych udało mi się wychwycić drobne błędy, które zostały przeze mnie poprawione w finalnej wersji. Nigdy nie zdarzyło się, aby aplikacja przestała odpowiadać, co nasuwa wniosek o dobrze zaimplementowanym wzorcu MVC.

Jak wspomniałem w rozdziale 7, zespół cech, użytych w programie do klasyfikacji, niezbyt dobrze spełnił swoje zadanie. Rys. 10. prezentuje wykres zależności cech 1 oraz 2. Klasyfikowane znaki są zbyt blisko siebie, aby możliwe było osiągnięcie dobrej jakości rozpoznawania przy użyciu klasyfikacji minimalno-odległościowej. Jeśli w przyszłości miałbym rozwijać i ulepszać opisywaną aplikację, zastanowiłbym się nad zmianą wyznaczanych cech na inne, lepiej rozróżniające klasyfikowane obiekty. Myślę, że zmiana klasyfikatora na np. klasyfikator liniowy lub sieć neuronową mogłaby korzystnie wpłynąć na jakość rozpoznawania. W obecnej postaci, program bardzo często myli się czy podobnych do siebie znakach, np. myli ‘B’ z ‘8’, ‘1’ z ‘I’, ‘O’ z ‘0’, czy ‘H’ z ‘K’. Poprawne rozpoznawanie znaków zachodzi wówczas, gdy do testowania procesu rozpoznawania zastosowałem te same filmy, które były źródłem tablic i segmentowanych cech wprowadzonych do bazy wiedzy. Niestety w praktyce obiekty, które zostały użyte do utworzenia bazy wiedzy, nie występują nigdy (lub prawie nigdy) jako obiekty do rozpoznania. Z tego powodu przydatność aplikacji maleje – w prawdziwym środowisku pracy aplikacja popełniałaby więcej błędów niż poprawnych decyzji (jakość klasyfikacji w teście 3. wyniosła tylko 11%, co jest bardzo niskim wynikiem).

Jakość klasyfikacji nie jest w tej pracy najważniejsza, gdyż jej głównym celem było utworzenie podstawy dla późniejszych badań i ulepszeń. Utworzona aplikacja została zaprojektowana i zbudowana w sposób umożliwiający modyfikacje poszczególnych modułów procesu rozpoznawania. Dzięki temu badanie działania algorytmów dla przedstawionego problemu będzie polegać na modyfikacji części segmentującej i klasyfikującej aplikacji. Możliwe także będzie utworzenie różnych implementacji części segmentującej i klasyfikującej, oraz zostawienie użytkownikowi wyboru, której z nich użyć.

9. Zakończenie

Niniejsza praca oraz opisywana aplikacja stanowi przedmiot do dalszej pracy. Jej rozwój i poprawa działania byłyby dobrym przedmiotem pracy magisterskiej, której tematem mogłoby być np. porównanie metod segmentacji, wyznaczania cech czy rozpoznawania i ich wpływ na jakość rozpoznawania. Możliwe byłoby również zaproponowanie nowatorskiego algorytmu klasyfikacji lub metody wyznaczania cech, które poprawiłyby jakość klasyfikacji osiąganą przez aplikację. Interesujące mogłoby być także badanie wpływu oświetlenia na jakość segmentacji i rozpoznawania.

Rozpoznawanie obrazu to stosunkowo młoda gałąź nauki, która cały czas ewoluuje, dostarczając nowych wyników i rozwiązań. Zbudowana aplikacja, której proces powstawania został udokumentowany w niniejszej pracy pozwala na rozpoznawanie znaków w samochodowych tablicach rejestracyjnych. Jest przystosowana do rozpoznawania tablic na terenie całej Unii Europejskiej, dzięki identycznym wymogom formalnym tablic dla jej państw członkowskich. Po dopracowaniu systemu cech i klasyfikatora, program może mieć dość szerokie zastosowanie: może rejestrować tablice celem zbierania danych statystycznych, po dobudowaniu odpowiedniego modułu może obsługiwać bramki parkingów strzeżonych, może także być wykorzystywany przez służby mundurowe w poszukiwaniu skradzionych lub nielegalnych pojazdów.

10. Bibliografia

1. Duda R.O., Hart P.E., Stork D.G., *Pattern Classification*, Wiley-Interscience, Wiley 2001.
2. Gonzales R.C., Woods R.E., *Digital Image Processing*, Prentice Hall, New Jersey 2002.
3. Jain A. K., *Fundamentals of Digital Image Processing*, Prentice-Hall International Editions, Engelwood Hills 1989.
4. Korohoda P., Tadeusiewicz R., *Komputerowa analiza i przetwarzanie obrazów*, Wydawnictwo Postępu Telekomunikacji, Kraków 1997.
5. Pavlidis T., *Grafika i przetwarzanie obrazów*, WNT, Warszawa 1987.
6. Press W. H., *Numerical Recipes in C*, Cambridge University Press, Cambridge 1992.
7. Tadeusiewicz R., *Systemy wizyjne robotów przemysłowych*, WNT, Warszawa 1992.

Dodatek A – Wybrane wzory matematyczne

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^p \cdot f(x,y) dx dy \qquad [1]$$

gdzie p,q – rząd momentu geometrycznego.

$$m_{pq} = \sum_{i=1}^{N} \sum_{j=1}^{M} i^p j^q \cdot f(i,j), f = \begin{cases} 1, & \text{gdy w obszarze} \\ 0, & \text{gdy poza obszarem} \end{cases} \qquad [2]$$

gdzie p,q – rząd momentu geometrycznego, i,j – współrzędne piksela.

$$\tilde{i} = \frac{m_{10}}{m_{00}} \qquad \tilde{j} = \frac{m_{01}}{m_{00}} \qquad [3]$$

gdzie ĩ, ĵ – centrum obrazu.

$$M_{pq} = \sum_{i=1}^{N} \sum_{j=1}^{N} (i - \tilde{i})^p \cdot (j - \tilde{j})^q \cdot x_{ij} \qquad [4]$$

gdzie M_{pq} – moment centralny rzędu p,q, x_{ij} – wartość piksela w punkcie i,j.

$$M1 = \frac{M_{20} + M_{02}}{m_{00}^2} \qquad [5]$$

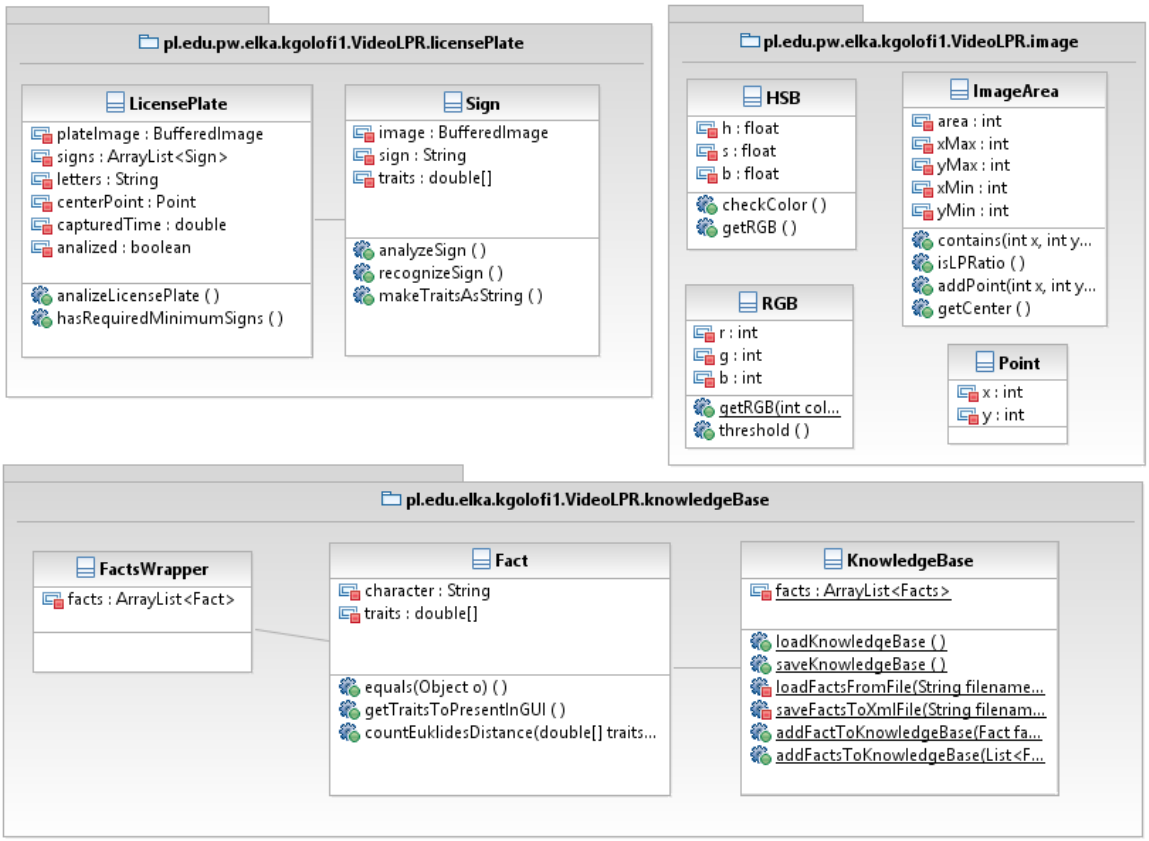
gdzie M1 – niezmiennik momentowy 1, M_{xy} – moment centralny rzędu x,y, m_{xy} – moment zwykły rzędu x,y.

$$M2 = \frac{(M_{20} - M_{02})^2 + 4M_{11}^2}{m_{00}^4} \qquad [6]$$

gdzie M1 – niezmiennik momentowy 1, M_{xy} – moment centralny rzędu x,y, m_{xy} – moment zwykły rzędu x,y.

Dodatek B – Opis wybranych klas programu

Przedstawiam wybrane klasy stworzonej aplikacji, skupiając się na klasach odpowiedzialnych za przetwarzania i rozpoznawanie obrazu oraz strukturze danych używanych w programie. Pełny opis klas programu w formacie plików **.html* znajduje się na płycie z aplikacją, w katalogu *„dokumentacja-javadoc”*.



Rys. 13. Diagram klas struktury danych wykorzystywanych w systemie rozpoznawania.

Rys. 13. przedstawia klasy implementujące użyte struktury danych. Ich opis znajduje się w poniższej tabeli:

Nazwa klasy	Nazwa metody	Opis
HSB		Reprezentuje model przestrzeni barw HSB. Przechowuje trzy składowe barwy, potrafi je przeliczyć na wartość całkowitą używaną w klasie BufferedReader.
	getRGB()	Przelicza kolor na RGB i zwraca instancję klasy RGB.
	checkColor()	Sprawdza kolor wśród rozpoznawanych kolorów (biały,

		czarny, niebieski), zwraca java.util.Color.
RGB		Reprezentuje model przestrzeni barw RGB. Przechowuje trzy składowe barwy, potrafi je zamienić na wartość całkowitą.
	getRGB(int colR, int colG, int colB)	Przyjmuje jako argumenty trzy składowe RGB, zwraca wartość całkowitą używaną w BufferedImage.
	threshold()	Proguje kolor, poprzez przeliczenie go na odcień szarości i sprawdzenie jasności. Jasność > 50% skutkuje zwróceniem wartości całkowitoliczbowej koloru białego, w przeciwnym wypadku zwraca kolor czarny – jako liczbę całkowitą.
ImageArea		Reprezentuje prostokątny obszar obrazu. Przechowuje współrzędne maksymalne i minimalne krawędzi prostokąta.
	contains(int x, int y)	Sprawdza, czy podany punkt zawiera się w obszarze. Przyjmuje współrzędne całkowitoliczbowe, zwraca wartość logiczną.
	isLPRatio()	Sprawdza, czy stosunek szerokości do wysokości obszaru mieści się w granicy stosunku wymiarów dla tablicy rejestracyjnej (zapamiętanej jako stała). Zwraca wartość logiczną.
	getCenter()	Liczy środek obszaru i zwraca go jako obiekt Point.
	addPoint(int x, int y)	Poszerza obszar tak, aby obejmował zadany w formie współrzędnych całkowitoliczbowych punkt.
Point		Przechowuje informacje o współrzędnych na płaszczyźnie, przechowuje dwie składowe całkowitoliczbowe (x, y).
Sign		Przechowuje informacje o pojedynczym znaku z tablicy rejestracyjnej. Przechowuje sprogowany obraz znaku, jego cechy oraz rozpoznany znak.

	analyzeSign()	Analizuje obraz znaku, licząc cechy i zapisując je.
	recognizeSign()	Rozpoznaje obraz znaku. Odwołuje się do klasyfikatora, dając mu cechy i otrzymując znak, który jest zapisywany w polu *sign*.
	makeTraitsAsString()	Zbiera wartości cech w sposób dogodny do prezentacji użytkownikowi i zwraca je jako String.
LicensePlate		Reprezentuje tablicę rejestracyjną. Przechowuje obraz tablicy, listę znaków uzyskanych w procesie segmentacji List<Sign>, ciąg rozpoznanych znaków jako String, oraz współrzędne centrum tablicy w przestrzeni rozdzielczości obrazu jako Point.
	analizeLicensePlate()	Analizuje obszar tablicy rejestracyjnej, wykorzystując algorytm wypełniania zalewowego wyznacza kolejne znaki i dodaje je do listy znaków.
	hasRequiredMinimumSigns()	Sprawdza czy tablica posiada minimalną zadaną liczbę znaków, przechowywaną jako stała definiowana przez użytkownika. Zwraca wartość logiczną.
Fact		Reprezentuje pojedynczy fakt bazy wiedzy. Zawiera informacje o cechach oraz znaku definiowanym przez cechy.
	equals(Object o)	Porównuje dwie instancje klasy Fact, zwraca wartość logiczną.
	getTraitsToPresentInGUI()	Zbiera wartości cech w sposób odpowiedni do prezentacji ich użytkownikowi. Zwraca jako łańcuch znaków String.
	countEuklidesDistance(double[] traits)	Przyjmuje jako argument zespół cech. Liczy odległość euklidesową do zadanych cech i zwraca ją jako liczbę zmiennoprzecinkową.

FactsWrapper		Przechowuje listę faktów. Klasa służy tylko jako opakowanie dla zbioru faktów, przy mapowaniu pliku *.*xml* na obiekty i odwrotnie.
KnowledgeBase		Reprezentuje bazę wiedzy. Przechowuje listę faktów, odczytywaną z pliku *.*xml*. Po zakończeniu aplikacji zapisuje bazę z pamięci do pliku *.*xml*.
	loadKnowledgeBase()	Metoda statyczna, ładuje bazę wiedzy z pliku, korzysta z metody loadFactsFromXmlFile.
	saveKnowledgeBase()	Metoda statyczna, zapisuje bazę wiedzy do pliku, korzysta z metody saveFactsToXmlFile.
	loadFactsFromXmlFile(String filename)	Zapisuje bazę wiedzy do pliku *.*xml*, mapując obiekty na reprezentację xml-ową. Jako argument przyjmuje nazwę pliku w katalogu bieżącym.
	saveFactsToXmlFile(String filename)	Wczytuje bazę wiedzy z pliku *.*xml*, mapując reprezentację xml-ową na obiekty. Jako argument przyjmuje nazwę pliku w katalogu bieżącym. Jeśli plik istnieje, jest nadpisywany.
	addFactToKnowledgeBase(Fact f)	Dodaje fakt, przekazany jako argument, do bazy wiedzy.
	addFactsToKnowledgeBase(List<Fact> facts)	Dodaje listę faktów, podają jako argument, do bazy wiedzy. Korzysta z metody addFactToKnowledgeBase.

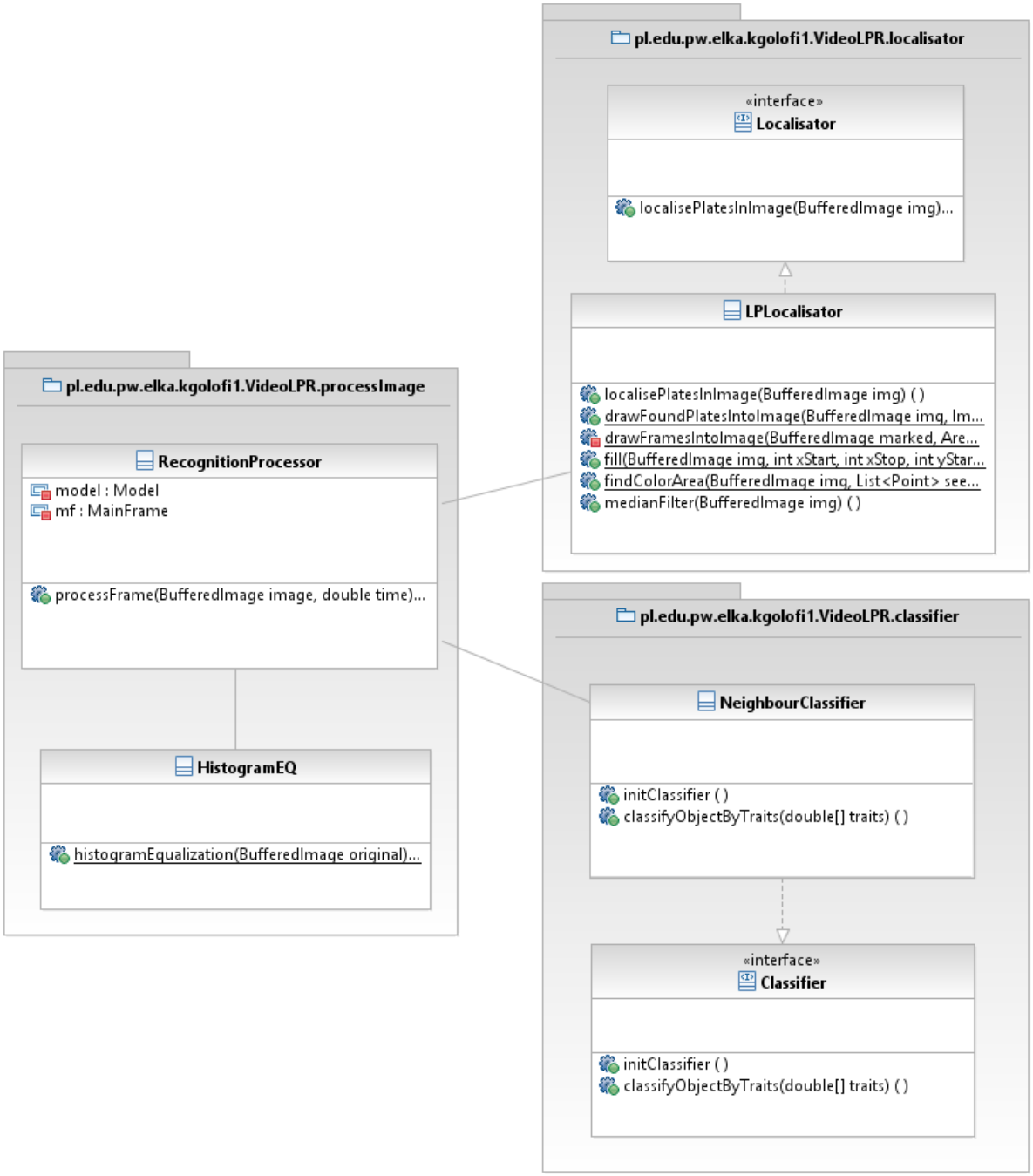
Po opisie klas reprezentujących strukturę danych, przedstawiam klasy odpowiedzialne za proces rozpoznawania. Diagram klas przedstawiony jest na rys. 14.

Opis klas i ich metod zamieszczony w tabeli:

Nazwa klasy	Nazwa metody	Opis
LPLocalisator		Implementuje interfejs Localisator. Służy do segmentacji obrazu i lokalizacji tablic w obrazie. Jego instancja przechowywana jest w Modelu.
	localisePlatesInImage(BufferedImage img, double time)	Główna metoda, realizuje lokalizację tablic w obrazie. Jako parametr przyjmuje obraz, w którym będzie lokalizować tablice.

		Zwraca listę obiektów klasy ImageArea, jako obszary zawierające potencjalne tablice rejestracyjne.
	drawFoundPlatesIntoImage(BufferedImage img, ImageAreaList plateList)	Statyczna metoda pomocnicza, zaznaczająca znalezione tablice w obrazie. Przyjmuje jako argument obraz oraz listę obszarów tablicy do zaznaczenia. Zwraca kopię obrazu, z narysowanymi obwódkami podanych obszarów.
	drawFramesIntoImage(BufferedImage img, ImageAreasList frames, int color, int thickness)	Prywatna statyczna metoda pomocnicza, wywoływana przez drawFoundPlatesIntoImage. Rysuje na podanym w parametrze obrazie podane w parametrze ramki. Jako parametr także przekazywana jest grubość ramki oraz jej kolor. Zwraca obraz z narysowanymi ramkami.
	fill(BufferedImage img, int xStart, int xStop, int yStart, int yStop, int color)	Wypełnia danym kolorem dany obszar obrazu. Otrzymuje w parametrach obraz, opis obszaru do wypełnienia i kolor. Zwraca obraz z wypełnionym obszarem.
	findColorArea(BufferedImage img, List<Point> seedPoints, Color color, int minArea)	Metoda implementująca algorytm wypełniania zalewowego, z wykorzystaniem kolejki. Otrzymuje obraz, punkty startowe oraz rozpoznawany kolor i minimalny obszar jako parametry. Wypełnianie zalewowe zbiera punkty należące do kształtu o podanym kolorze, zaczynając od punktów startowych. Zwraca obszar obrazu ImageArea.
	medianFilter(BufferedImage img)	Filtruje podany obraz z wykorzystaniem filtracji medianowej. Na wejściu otrzymuje obraz do filtracji, zwraca obraz po filtracji.
NeighbourClassifier		Implementuje interfejs Classifier. Służy do klasyfikacji znaków

		poprzez minimalną odległość do faktów z bazy wiedzy.
	initClassifier()	Inicjalizuje klasyfikator. Klasyfikator minimalno-odległościowy nie potrzebuje inicjalizacji, metoda ta jest utworzona z myślą o przyszłej rozbudowie aplikacji.
	classifyObiectByTraits(double[] traits)	Klasyfikuje znak na podstawie cech, podanych w parametrze. Znajduje fakt o najmniejszej odległości cech i zwraca jego etykietę jako decyzję klasyfikacyjną.
HistogramEQ		Klasa realizująca proces wyrównywania histogramu. Utworzona przez Bostiana Cigan (http://zerocool.is-a-geek.net).
	histogramEqualization(BufferedImage original)	Główna metoda statyczna klasy. Zmodyfikowana przeze mnie, realizuje algorytm rozszerzenia histogramu, oraz progowania obrazu do obrazu czarno-białego. Jako parametr przyjmuje obraz do przetworzenia, na wyjściu zwraca obraz czarno-biały po wyrównaniu histogramu.
RecognitionProcessor		Przeprowadza proces rozpoznawania, korzysta z klas implementujących interfejsy Classifier i Localisator.
	processFrame(BufferedImage img, double time)	Główna metoda klasy, rozpoznaje obraz. Otrzymuje jako parametr obraz do przetworzenia oraz czas uzyskania obrazu (czas filmu lub czas aktualny). Przeprowadza proces rozpoznawania, wyświetla aktualnie rozpoznane znaki w GUI, dodaje rozpoznane tablice do listy rozpoznanych tablic. Zwraca obraz z zaznaczonymi obwódką tablicami.



Rys. 14. Diagram klas tworzących system rozpoznawania obrazu.