Rok akademicki 2014/2015

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Przemysław Onaszkiewicz

Rozpoznawanie znaków drogowych w sekwencjach wideo

Opiekun pracy
mgr inż. Krzysztof Chabko

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Kierunek: Informatyka

Specjalność: Inżynieria Systemów Informatycznych

Data urodzenia: 1991.01.10

Data rozpoczęcia studiów: 2010.10.01

Życiorys

Urodziłem się 10 stycznia 1991 roku w Warszawie, lecz moją rodzinną miejscowością jest Miedzna w powiecie węgrowskim. W roku 2004 ukończyłem szkołę podstawową im. Tadeusza Kościuszki w Miedznie. W latach 2004 – 2007 uczęszczałem do gimnazjum w Miedznie. W latach 2007 – 2010 uczęszczałem do I Liceum Ogólnokształcącego im. Adama Mickiewicza w Węgrowie do klasy o profilu matematyczno-fizycznym. W 2010 roku zdałem maturę i zostałem przyjęty na studia dzienne na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. W trakcie piątego semestru wybrałem specjalizację Inżynieria Systemów Informatycznych.

.....
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 dotyczy zagadnienia rozpoznawania obrazów. Jej celem jest wykonanie aplikacji która będzie w stanie otwierać strumienie wideo, wydzielać z nich ramki i przetwarzać je. Owo przetwarzanie ma polegać na rozpoznawaniu przynależności do czterech głównych grup znaków drogowych, jak również próbie implementacji rozpoznawania kilku wybranych konkretnych znaków. Praca dokumentuje przebieg prac nad realizacją tejże aplikacji napisanej w języku C++. Zawiera ona opis wymagań, projekt jak i opis jego realizacji.

W pierwszych rozdziałach omówione jest zagadnienie rozpoznawania obrazów w dzisiejszych czasach i przedstawiony jest typowy system rozpoznawania obrazów. Jest również przedstawiony przegląd istniejących rozwiązań. Dalsze rozdziały przedstawiają analizę wymagań aplikacji i projekt jej wykonania. W dalszej części następuje opis przeprowadzonych testów i podsumowanie pracy wraz z wnioskami i sugestiami dalszego rozwoju. Na końcu jako dodatek dołączona została instrukcja użytkownika.

Słowa kluczowe: rozpoznawanie znaków drogowych, C++, rozpoznawanie obrazów, przetwarzanie obrazów, znaki drogowe.

ROAD SIGNS RECOGNITION IN VIDEO STREAMS

This Engineer's Degree thesis touch the issue of pattern recognition. It's objective is making an application which is able to open video streams, capture single frames from it and process them. This processing is about recognition of belonging to one of four main signs groups and also about an attempt of recognition of a few concrete ones. This paper documents course of works on building the application in C++ language. It contains description of application requirements, application project and description of its realization.

In the first chapters of this papers a definition of pattern recognition nowadays and typical structure of pattern recognition system is described. This chapters also contain research about existing solutions of road sign recognition problem. The next part of the paper describes application requirements analysis and process of making it. Further chapters touch the testing issue alongside with conclusions and suggestions of improvement and contains a summary of this study. The last chapter contains user's guide.

Keywords: road sign recognition, C++, pattern recognition, image processing, road signs

1 Wstęp ... 3
1.1 Wprowadzenie ... 3
1.2 Cel pracy ... 4
1.3 Plan pracy ... 5
2 Systemy wsparcia kierowcy ... 6
2.1 Komercyjne systemy rozpoznawania znaków drogowych ... 6
2.1.1 Mobileye ... 8
2.1.2 Continental ... 9
2.2 Rozwiązania akademickie ... 9
2.3 Podsumowanie przeglądu. ... 11
3 Koncepcja systemu ... 13
3.1 Wymagania funkcjonalne ... 13
3.2 Wymagania niefunkcjonalne ... 15
3.3 Przygotowanie do realizacji (research) ... 16
3.4 Przypadki użycia ... 16
3.5 Użytkowanie systemu ... 20
4 Znaki drogowe ... 21
4.1 Znaki ostrzegawcze ... 21
4.2 Znaki zakazu ... 22
4.3 Znaki nakazu ... 23
4.4 Znaki informacyjne ... 23
4.5 Umiejscowienie znaków w klatkach obrazu (wpływ tła na zlewanie) ... 24
5 Analiza problemu i zastosowane rozwiązania ... 25
5.1 Omówienie wymagań ... 25
5.2 Komunikacja podzespołów ... 25
5.3 Ograniczenia projektu i zakres jego realizacji ... 26
5.4 Implementacja systemu ... 26
5.4.1 Narzędzia ... 26
5.4.1.1 CMake ... 27
5.4.2 Język programowania i system operacyjny. ... 29
5.4.3 Zastosowane biblioteki i narzędzia ... 30
5.4.3.1 QtWidgets (GUI) ... 30
5.4.3.2 OpenCV ... 31
5.4.4 Komponenty ... 32
5.4.4.1 Wejście ... 35
5.4.4.2 Poprawa jakości ... 35
5.4.4.3 Segmentacja ... 37

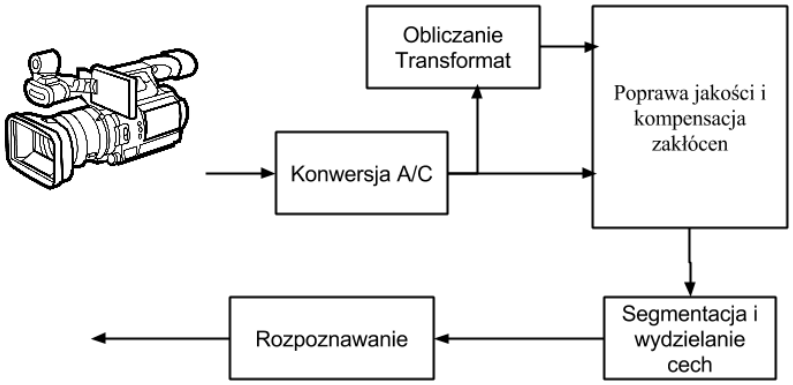
5.4.4.4 Identyfikacja 39
5.4.4.5 Rozpoznawanie 41
5.4.4.6 Przechowywanie danych.(struktura, zapis, odczyt) 43
5.4.5 Interfejsy 45
5.4.5.1 Mechanizm sygnałów slotów w Qt 45
6 Testy 47
6.1 Przeprowadzone Testy Poboczne 47
6.2 Statystyki wykrywanych obiektów 48
6.3 Fałszywie wykrywane obiekty 50
6.4 Zestaw uruchomieniowy 51
7 Podsumowanie 52
7.1 Podsumowanie wykorzystanych bibliotek 52
7.2 Podsumowanie wykonanych prac 53
7.3 Sugerowane drogi dalszego rozwoju aplikacji 53
8 Bibliografia. 54
9 Załącznik 1. Zawartość płyty CD 55
10 Załącznik 2. Instrukcja użytkownika. 56
10.1 Ładowanie plików z danymi 56
10.2 Przetwarzanie sekwencji wideo w trybie manualnym 57
10.3 Przetwarzanie sekwencji wideo w trybie automatycznym. 62

1 Wstęp

Niniejszy rozdział zawiera wprowadzenie do tematyki pracy. Przedstawia on cel jej powstania, jak również plan rozmieszczenia treści w niej zawartych.

1.1 Wprowadzenie

Rozpoznawanie obrazów jest to najogólniej mówiąc proces przetwarzania i analizowania obrazu, który pozwala stwierdzić jakiego rodzaju obiekty znajdują się na obrazie, a następnie na stwierdzenie czy wśród danych obiektów znajdują się takie, które nas w jakiś sposób interesują lub których szukamy. Proces ten składa się z kilku podstawowych etapów, które zostały przedstawione na rysunku poniżej.



Rysunek 1.1 Ogólny schemat systemu przetwarzania obrazów.

Pierwszym etapem przetwarzania obrazu jest jego akwizycja. Ów etap jest bardzo ważny, ale często pomijany w rozważaniach. Na szczęście większość urządzeń nagrywających sekwencje wideo w dzisiejszych czasach jest przystosowana do zapisywania wyników swojej pracy w formie cyfrowej.

Niezmiernie ważnym jest również poprawa jakości obrazu przed próbą jego segmentacji. Wystarczy poruszyć aparatem w momencie robienia zdjęcia albo uchwycić rozbłysk światła na obiektywie, aby całe zdjęcie uczynić niemożliwym do rozpoznania poprzez zakłócenia na nim występujące. Dzieje się tak, ponieważ wszelkie zakłócenia, których system nie zredukuje będą propagować dalej i zniekształcać wyniki działania dalszych etapów procesu. Niestety nie wszystkie zakłócenia są możliwe do usunięcia, a ponadto próba usunięcia jednych najczęściej powoduje wzmocnienie efektu innych.

Kolejne etapy są już związane z rozpoznawaniem obrazu. Pierwszym z nich jest segmentacja która wyznacza obiekty mogące być potencjalnymi obiektami

zainteresowania systemu. Po ich wydzieleniu następuje dla każdego z nich obliczenie wartości cech przez które są one reprezentowane. W następnym etapie owe cechy są używane do sklasyfikowania obiektu. Jedną z metod klasyfikacji jest metoda najbliższej odległości do obiektu wzorcowego jednej z dostępnych klas. Wynikiem działania tego etapu jest odpowiedź na pytanie „Czy na obrazie znajdują cię obiekty należące do którejś z grup zainteresowania?”.

W dzisiejszych czasach z racji na rosnącą moc obliczeniową zarówno komputerów stacjonarnych jak i urządzeń mobilnych przetwarzanie obrazów znajduje zastosowanie w wielu dziedzinach nie tylko przemysłu, ale i życia codziennego.

Wzrost mocy obliczeniowej urządzeń pozwala wdrażać bardziej wyrafinowane algorytmy. Pozwala to znieść ograniczenie mocno sprecyzowanych i w niewielkim stopniu zmiennych warunków sceny, a jednocześnie możliwe staje się zanalizowanie coraz bardziej skomplikowanych scen w czasie rzeczywistym albo do niego zbliżonym. Powszechnymi stają się systemy wykorzystujące dane wizyjne w celu rozpoznawania np. tablic rejestracyjnych samochodów, śledzenia postaci ludzkich oczami kamer monitoringu w celu ustalenia trasy przemarszu danej osoby.

Rozpoznawanie obrazów znajduje swoje zastosowania często w miejscach gdzie kończy się przydatność wielu sensorów ze względu na dokładność danych przez nie dostarczanych (ale również ich charakter). Przykładem może być automatyczny system monitorowania posesji i możliwość rozpoznania sylwetki osoby bądź zwierzęcia, która wtargnęła na posesję i w zależności od jego wyniku owego rozpoznawania podejmowania odpowiednich działań.

1.2 Cel pracy

Celem pracy jest stworzenie bazowej aplikacji z możliwością późniejszego jej rozszerzania. Podstawowymi zadaniami owej aplikacji będą:

- Przetwarzanie sekwencji wideo w celu zbierania danych numerycznych ręcznie wysegmentowanych obiektów będących niezbędnymi do tworzenia baz wiedzy potrzebnych do późniejszego ich rozpoznawania.
- Automatyczne przetwarzanie strumieni wideo w celu rozpoznawania znaków drogowych i klasyfikacji ich do konkretnych grup. W późniejszym etapie aplikacja ma zostać udoskonalona w celu dokładniejszego rozpoznawania znaków i przypisywania ich nie tylko do określonych grup znaków, ale utożsamienie z konkretnym ich członkiem.

1.3 Plan pracy

Niniejsza praca dzieli się na pięć zasadniczych części . Pierwsza część to rozdział drugi dokonujący przeglądu dziedziny i dostępnych na rynku gotowych rozwiązań. Wyjaśnione zostały w nim podstawowe zagadnienia i podział rozwiązań w celu wprowadzenia czytelnika w tematykę pracy.

Drugim, i zarazem głównym elementem pracy są rozdziały 3-5. Rozdział 3 zawiera koncepcję systemu wraz z przedstawieniem jego wymagań i przypadków jego użycia. Rozdział 4 zawiera omówienie charakterystyki obiektów, których rozpoznanie jest celem projektu. Rozdział 5 zawiera analizę problemu rozpoznawania obiektów, analizę wymagań postawionych przed twórcą i opis dobranych bibliotek, języków i narzędzi. Zasadniczą jego część stanowi jednak opis komponentów programu wraz ze sposobem ich działania.

Część trzecia obejmuje rozdziały 6-7. Rozdział 6 traktuje na temat przeprowadzonych testów aplikacji, wykrytych za ich pomocą problemów i wysnutych z nich wniosków. Rozdział 7 podsumowuje daną pracę i podsuwa możliwości dalszego usprawniania i rozwijania systemu.

Część czwarta to Bibliografia.

Część piąta to załączniki 1 i 2. Załącznik 1 mówi o zawartości płyty CD dołączonej do pracy, a załącznik 2 przedstawia instrukcję obsługi aplikacji.

2 Systemy wsparcia kierowcy

Systemy wsparcia kierowcy (ang. Driver Support Systems) dzielą się na kilka kategorii. Są to między innymi:

- General Object Detection – System wykrywający obiekty na drodze które mogą być zagrożeniem dla pojazdu gospodarza albo dla których to pojazd gospodarza może być zagrożeniem.
- Pedestrian Collision Warning – System wykrywający możliwość potencjalnego zderzenia z pieszym lub wejścia na kurs kolizyjny pieszego i pojazdu gospodarza.
- Trafic Sign Detection - System wykrywający znaki drogowe. Głównie ograniczenia prędkości, ale również inne znaki zgodne z konwencją Vienneńską.

W niniejszej pracy będą rozważane głównie te ostatnie, czyli systemy wykrywające znaki drogowe. Rozróżnia się wśród nich kilka grup ze względu na sposób działania. Są to:

- Systemy bazujące na współrzędnych GPS - współrzędne GPS są porównywane z zawartością bazy danych zawierającej rodzaj dróg i obowiązujące na nich ograniczenia prędkości.
- Systemy bazujące na rozpoznawaniu obrazów – systemy które opierają się na interpretacji obrazu przechwytywanego z kamery samochodu.
- Systemy hybrydowe – systemy w których wynik wstępnego rozpoznawania obrazu z kamery zostaje porównany z bazą danych, a po otrzymaniu potwierdzenia zgodności jest wyświetlany kierowcy.

W poniższych podrozdziałach niektóre z nich zostaną przedstawione.

2.1 Komercyjne systemy rozpoznawania znaków drogowych

Niestety na tym etapie systemy TSR (ang. Trafic Sign Recognition) są jeszcze na stosunkowo wczesnym etapie rozwoju. Skupiają się one głównie na rozpoznawaniu znaków krytycznych na zachowanie bezpieczeństwa, takich jak ograniczenia prędkości lub zakazy wyprzedzania. Wady i zalety każdego z tych typów zostały przedstawione w poniższej tabeli poniżej (Tabela 2.1)

Rodzaj	Zalety	Wady
Systemy bazujące na współrzędnych GPS	- deterministyczne – zależą tylko od danych w bazie danych i zasięgu GPS, Jeżeli oba te elementy działają to zawsze zostanie zwrócony aktualny wynik znajdujący się w bazie danych - nie jest zależna od warunków oświetlenia	- zależna od zasięgu GPS i od ukształtowania otoczenia - poprawność działania zależy również od aktualności bazy danych
Systemy bazujące na rozpoznawaniu obrazów	- zawsze analizowane dane są aktualne	- silnie zależne od warunków pogodowych i oświetleniowych - wrażliwe na zakłócenia związane z oświetleniem
Systemy hybrydowe	- dwuetapowe rozpoznawania gwarantuje mniejszy procent błędów. Błędy powodowane błędnym bądź niedokładnym rozpoznawaniem obrazu mogą zostać skorygowane poprzez dane otrzymane na podstawie położenia. Nieaktualne dane otrzymane na podstawie współrzędnych GPS mogą również zostać skorygowane na podstawie przeanalizowanych danych z kamery.	- problemy z wyważeniem który etap rozpoznawania jest ważniejszy

Tabela 2.1 Wady i zalety systemów rozpoznawania znaków drogowych.

Istnieje kilka firm, które udostępniają swoje rozwiązania w swoich autach. Niestety te wbudowane dostępne są tylko w autach luksusowych, których ceny oscylują wokół kilkuset tysięcy złotych. Istnieją jednak również rozwiązania zewnętrzne które są dostępne w niższej cenie.

2.1.1 Mobileye

Mobileye [1] jest technologiczną firmą która zajmuje się aplikacjami klienckimi opartymi na przetwarzaniu informacji wizualnych przeznaczonych na rynek DAS (ang. Driver Assistance Systems). Oprogramowanie Mobileye ogranicza ryzyko wypadków drogowych i ma potencjał zrewolucjonizowania rynku poprzez umożliwienie autonomicznego kierowania samochodem (wraz z innymi systemami DAS).

Urządzenia i oprogramowanie Mobieye dostarcza szczegółowej interpretacji pola widzenia w celu przewidzenia możliwych kolizji z innymi pojazdami, pieszymi zwierzętami i innymi przeszkodami.

Funkcja TSR (ang. Traffic Signs Recognition) jest używana do informowania i ostrzegania o ograniczeniach występujących na danym odcinku drogi. Rozpoznaje i interpretuje różne znaki drogowe, zarówno znaki stojące z boku drogi, jak i znaki wyświetlane na tablicach ponad drogą. Usługa obejmuje również rozpoznawanie znaków ograniczenia prędkości, która informuje kierowcę o przekroczeniu dopuszczalnej prędkości.

Kolejna funkcja nosi nazwę Line Departure Warning (LDR). Ostrzega kierowcę sygnałami wizualnymi i dźwiękowymi przed nieplanowanym i niekontrolowanym zjechaniem z pasa ruchu, czyli przed potencjalnie jednym z najniebezpieczniejszych zagrożeń w ruchu drogowym. System Mobileye mierzy odległość kół pojazdu od linii na drodze z obu stron. Badania przeprowadzone na flotach pojazdów firmowych wykazały 65% spadek ilości wypadków tego typu w pojazdach wyposażonych w system wykrywania linii pasa ruchu. Funkcja jest aktywowana tylko jeżeli prędkość pojazdu przekroczy 55 km/h i podczas zbaczania z obecnie zajmowanego pasa nie jest włączony kierunkowskaz.

Urządzenia Mobileye były integrowane w nowych modelach samochodów od 2007 roku. Pierwszy milion czipów został sprzedany do 2012 roku Następne 1.3 miliona czipów zostało sprzedane między 2012 i 2013 rokiem.

Obecnie Mobileye pracuje nad następną odmianą technologii DAS którą planuje wprowadzić w 2016 roku a która będzie umożliwiała prowadzenie pojazdu bez użycia rąk, po autostradach i w sytuacjach przeciążonego (gęstego) ruchu ulicznego. Technologia jest wspierana przez kilka patrzących do przodu kamer i kilka tanich radarów.

2.1.2 Continental

System rozpoznawania znaków stworzony przez firmę Continental [2] przypomina kierowcom o obecnie obowiązującym limicie prędkości za pośrednictwem wyświetlacza na głównym panelu samochodu. Rozpoznaje on wszystkie znaki, które są dostosowane do wymogów zawartych w konwencjach w Viennie, czyli znaków z białym tłem i czerwoną obwódką naokoło czarnych lub ciemno granatowych cyfr. Rozpoznaje on również sygnały widoczne na zmiennych elektronicznych wyświetlaczach pochodzące z systemu kontroli ruchu.

Poza danymi wizyjnymi system korzysta również z danych dostępnych w mapach GPS dostępnych w samochodzie na podstawie których jest on w stanie wyświetlić domyślne ograniczenie obowiązujące na danym odcinku drogi. Opcja ta jest bardzo przydatna w razie wykrycia znaku końca ograniczenia, lub kiedy kierowca wjeżdża w jakiś obszar zabudowany (Zazwyczaj obszar zabudowany nie jest poprzedzony znakiem ograniczenia prędkości, a wtedy na danym obszarze obowiązuje ograniczenie domyślne).

W niedalekiej przyszłości system będzie również zdolny do odczytywania tekstu z tabliczek umieszczonych pod znakami, w rodzaju: „Tylko dla ciężarówek” .

System rozpoznawania znaków wyprodukowany przez Continental składa się z systemu nawigacji i z kamery, z której dane mogą być analizowane zarówno w skorelowaniu z danymi systemu nawigacji, jak i bez jego udziału. Pozwala to systemowi ciągle informować kierowcę o obowiązującym ograniczeniu prędkości. Kamera rozpoznaje standardowe charakterystyki znaków drogowych w zasięgu swojego widzenia, a te rozpoznane są porównywane ze znakami w nawigacji i jeżeli występuje ich zgodność to są one wyświetlane kierowcy.

2.2 Rozwiązania akademickie

Temat rozpoznawania znaków drogowych jest szeroko eksploatowany w świecie akademickim, ponieważ jest on świetnym przykładem użytecznego wykorzystania metod rozpoznawania obrazu i jednocześnie przetestowania ich sposobu działania w zmiennym i różnorodnym środowisku. Od zmienności tła poprzez zmienne warunki naświetlenia, na zmiennej dynamice obrazu kończąc.

Jednym z największych wyzwań w tego typu oprogramowaniu jest wybranie metody detekcji, która pozwoli na zminimalizowanie współczynników fałszywych wykryć i jednoczesne zmaksymalizowanie współczynnika prawdziwych wykryć. Najczęściej stosowane metody detekcji znaków to [3]:

- Detekcja na podstawie barw.
- Detekcja na podstawie wykrywania kształtów.
- Detekcja za pomocą algorytmów uczenia maszynowego.

Każda z wymienionych powyżej metod ma określone wady i zalety , które są przedstawione w poniższej tabeli (Tabela 2.2).

Rodzaj detekcji	Zalety	Wady
Detekcja na podstawie barw	- podstawowy algorytm prosty w implementacji - zgodny z intuicją człowieka	- przy stosowaniu tylko kryterium koloru występuje zbyt duży współczynnik fałszywych wykryć - kolor jest zmienny i zależny od wielu czynników takich jak pora dnia i warunki pogodowe
Detekcja na podstawie kształtów	- mniej zależny od warunków oświetlenia i pogodowych,	- algorytm składający się z większej ilości etapów (ustalenie kształtu musi być poprzedzone jakimś rodzajem segmentacji która wyodrębni obiekt)
Detekcja na podstawie uczenia maszynowego	- stosunkowo szybkie działanie - przy dużym zbiorze danych uczących wykazuje duży współczynnik prawdziwych trafień.	- do poprawnego działania wymagany duży zbiór danych uczących - konieczność przeprowadzania fazy uczenia

Tabela 2.2 Wady i zalety metod detekcji znaków drogowych.

Jak można przypuszczać powyższe metody nie nadają się do samodzielnego zastosowania w systemach rozpoznawania obrazów , ale mogą zostać z powodzeniem użyte jako elementy składowe do bardziej skomplikowanych algorytmów rozpoznawania.

2.3 Podsumowanie przeglądu.

W ostatnich czasach wzrost mocy obliczeniowej komputerów sprawił że rozpoznawanie obrazów przeszła z dziedziny czysto naukowej do dziedziny dostępnej przez szarego użytkownika komputera. Wraz z postępem technologii również możliwości rozpoznawania znaków drogowych w czasie rzeczywistym zaczęły znajdować się w zasięgu. Niektóre z nowych modeli wysokiej klasy samochodów już są wyposażone w systemy asystujące kierowy, wśród których znajduje się system detekcji znaków drogowych, który umożliwia wykrycie i rozpoznanie niektórych grup znaków.

Wbrew pozorom takie systemy przydają się nie tylko w luksusowych samochodach jako wsparcie kierowcy ale mogą również odgrywać znaczącą rolę w konserwacji dróg. Każda droga musi być okresowo sprawdzana w celu wykrycia ewentualnego braku lub zniszczenia niektórych znaków jako, że te braki powodują niebezpieczeństwo wypadku w ruchu drogowym. Aplikacje rozpoznające znaki w połączeniu z danymi z modułów GPS mogą zautomatyzować ten proces, poprzez tworzenie map znaków wraz z ich umiejscowieniem na mapie. Przy kolejnej kontroli odcinka mogą wykrywać braki w sieci znaków i informować o nich konserwatora co pozwoli przyspieszyć proces konserwacji i uniknąć błędu ludzkiego.

Na pierwszy rzut oka rozpoznawanie znaków drogowych może się wydawać prostym zadaniem. Znaki drogowe są prostymi, prawidłowymi figurami ze ściśle określonymi właściwościami kolorystycznymi. Jednak im bliżej badania zbliżają się do zastosowań komercyjnych, tym bardziej ograniczenia problemu ulegają zmianie. W systemach DAS albo systemach inwentaryzacji dróg problemem nie jest już jak efektywnie wykryć i rozpoznać znak drogowy w pojedynczym obrazie, ale jak niezawodnie wykryć go w setkach tysięcy ramek wideo bez jakichkolwiek fałszywych alarmów, często używając niskiej jakości, tanich sensorów (kamer) dostępnych w masowej produkcji.

W tym podrozdziale przedstawiony został podział metod rozpoznawania znaków drogowych. Rodzajami owych metod są:

- metody bazujące na kolorach
- metody bazujące na wykrywaniu kształtów
- metody bazujące na uczeniu maszynowym

Ciężko jest porównać owe metody ponieważ badacze często stosują inne kryteria podczas ich tworzenia lub adaptowania, a owe metody są mocno zależne od rodzaju przyjętych założeń.

Czynnikami najczęściej branymi pod uwagę przy wyborze metody są :

- Typ wejście. Wideo czy statyczne obrazy?
- Zasięg metody. Ma ona rozpoznawać określony znak, ich grupę, a może kilka grup?
- Warunki nagrywania sekwencji/obrazów. Czy dane są uzyskiwane w świetle dziennym, nocnym czy w obu? Warunki pogodowe są stałe czy zmienne?
- Typ sensora. Wysoka czy niska rozdzielczość? Kolorowa czy w odcieniach szarości?
- Wymagania szybkości przetwarzania. Czy przetwarzanie musi następować w czasie rzeczywistym czy nie musi?
- Akceptowalny współczynnik prawdziwych i fałszywych wykryć.

Natura problemu, dostępność sensorów i docelowe zastosowanie zazwyczaj determinuje której metody użyć.

Faktem jest, że rozwiązania komercyjne posiadają dużo większe wymagania zarówno w dziedzinie wydajności (przetwarzanie w czasie rzeczywistym) jak i w zakresie współczynników prawdziwych i fałszywych trafień. Owe współczynniki są o tyle ważne, że mimo iż owe systemy powinny tylko wspierać człowieka, po pewnym czasie człowiek ma tendencje do zbytniego zawierzania takim systemom i polegania na wynikach ich działania, co w razie pomyłki może prowadzić do wypadków. Przykłady takich zachowań kierowców można było zaobserwować podczas wchodzenia na rynek detaliczny systemów GPS, kiedy to kierowcy zbytnio ufający wskazaniom pilota nawigacji powodowali kolizje drogowe.

3 Koncepcja systemu

Celem niniejszej pracy jest stworzenie podstawowej aplikacji, z możliwością późniejszego rozwijania, której zadaniem będzie najogólniej mówiąc rozpoznawanie znaków drogowych w sekwencjach wideo. Aplikacja powinna posiadać dwa tryby działania. Pierwszy tryb pracy będzie udostępniał możliwość kolekcjonowania danych potrzebnych do rozpoznania interesujących użytkownika obiektów. Drugi tryb pracy będzie trybem automatycznym, w którym użytkownik będzie miał możliwość prześledzenia przebiegu sekwencji wideo wraz z zaznaczonymi na niej rozpoznanymi obiektami.

3.1 Wymagania funkcjonalne

ID	Nazwa	Opis
WF 1.	Odtwarzanie danych obrazowych z plików	System powinien umożliwiać odtwarzanie danych obrazowych z plików
WF 1.1.	Przetwarzanie obrazu	Przetwarzanie obrazu odczytanego z pliku w popularnych formatach (.jpg, .png)
WF 1.2.	Przetwarzanie sekwencji wideo	Przetwarzanie kolejnych ramek wyodrębnionych z sekwencji wideo
WF 2.	Przetwarzanie w trybie Auto	Aplikacja powinna posiadać tryb automatyczny, w którym system będzie próbował sam, na podstawie danych domyślnych bądź wczytywanych z pliku konfiguracyjnego, rozpoznać znaki drogowe w sekwencji wideo lub obrazie
WF 2.1.	Rozpoznawanie grup znaków	Program powinien rozpoznawać cztery główne grupy znaków: ostrzegawcze, informacyjne, nakazu, i zakazu
WF 2.2	Możliwość rozszerzenia funkcji o rozpoznawanie konkretnych znaków	Program powinien również posiadać opcje rozszerzenia o możliwości identyfikacji konkretnych znaków.

WF 3.	Przetwarzanie w trybie manualnym	Aplikacja powinna posiadać tryb automatyczny który będzie umożliwiał ustawienie własnych parametrów i prześledzenie przebiegu rozpoznawania, jak również wyświetlenie danych liczbowych i statystyk kolorów danego obiektu.
WF 3.1.	Ustawianie parametrów segmentacji	Użytkownik będzie miał możliwość ręcznego ustawienia parametrów segmentacji oddzielnie dla każdej z barw. W segmentacji każdej z barw będzie miał możliwość ustawienia parametrów progowania każdego kanału.
WF 3.2.	Ustawienie parametrów przekształceń morfologicznych	Aplikacja powinna umożliwiać stosowanie przekształceń morfologicznych po segmentacji, takich jak otwarcie lub zamknięcie
WF 3.3.	Zapisywanie bieżącej ramki przetwarzanego pliku	System powinien umożliwić zapisanie bieżącej ramki w celu odtworzenia jej w aplikacji zewnętrznej (jeżeli zaistnieje konieczność sprawdzenia danych obrazu których system nie udostępnia).
WF 4.	Wczytywanie danych konfiguracyjnych z pliku	Program powinien umożliwić wczytanie danych konfiguracyjnych z pliku które zostaną wykorzystane do klasyfikowania obiektów.
WF 5.	Wyodrębnianie kształtów wewnątrz znaków	Wyodrębnianie kształtów znajdujących się wewnątrz znalezionych konturów jeżeli takie istnieją.
WN 6.	Rodzaje rozpoznawanych znaków	Rozpoznawane przez program znaki powinny należeć do 4 podstawowych (a co za tym idzie najliczniejszych) grup znaków: ostrzegawczych, informacyjnych, zakazu, nakazu. Tylko w sytuacjach, kiedy znaki nie stykają się (przykładem takiej sytuacji są dwa znaki informacyjne umieszczone jeden zaraz pod drugim).

Tabela 3.1 Wymagania funkcjonalne aplikacji.

3.2 Wymagania niefunkcjonalne

ID	Nazwa	Opis
WN 1.	System operacyjny	Wymagany system operacyjny to Linux, najlepiej Ubuntu.
WN 2.	Interfejs użytkownika	Interfejs użytkownika stworzony na platformie Qt 5.2.0.
WN 3.	Możliwości rozbudowy	System powinien mieć budowę, która umożliwia łatwe dodawanie kolejnych modułów bądź funkcjonalności
WN 4.	Wydajność	System powinien umożliwiać przetwarzanie znaków w prędkości interakcyjnej. Nie musi być ono wykonane w czasie rzeczywistym ale również nie powinno być konieczne zostawianie kilkosekundowej sekwencji na kilka godzin w celu jej zanalizowania.
WN 5.	Bezpieczeństwo	System w obecnej wersji nie jest przeznaczony do użytkowania w samochodach, a w związku z tym jego działanie nie jest krytyczne dla użytkownika. Z tego powodu w razie błędnego wyniku nie ma poważnych konsekwencji. Nie są w nim również przetwarzane dane wrażliwe dla bezpieczeństwa użytkownika (np. hasła, numery kont) więc nie istnieje konieczność zabezpieczania przetwarzanych danych w jakikolwiek sposób, poza zabezpieczeniem przed uszkodzeniem używanych danych (sekwencji).
WN6	Obsługiwane formaty plików.	Aplikacja powinna obsługiwać następujące formaty plików: MP4, MOV, AVI

WN7	Przechowywanie pliku z danymi używanymi do stworzenia klasyfikatora	Jako że głównymi przechowywanymi danymi będą dane liczbowe niezmienników momentowych i statystyk kolorów do przechowywania ich wybrałem format CSV, co pozwoli na wygodne odpalanie danego pliku przez zewnętrzną aplikację (np. Libre Calc) i dokonywanie na nim pomocniczych operacji.

Tabela 3.2 Wymagania niefunkcjonalne aplikacji.

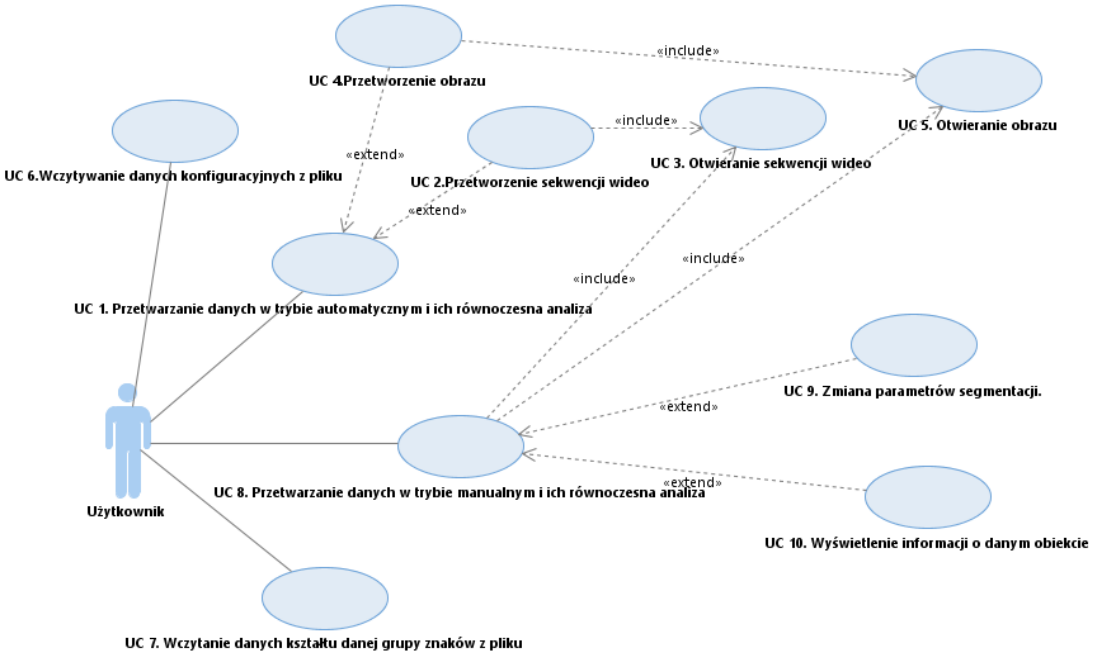
3.3 Przygotowanie do realizacji (research)

Na wstępie nastąpiło poszukiwanie rozwiązań dostępnych na rynku w celu ustalenia zakresu ich funkcjonalności, którego wynik przedstawiony jest w podrozdziałach 2.1.1 i 2.1.2 i ewentualnie sposobu ich działania. Niestety szczegóły implementacji są strzeżone przez firmy. Na podstawie danych zawartych na stronach internetowych producentów [1][2], udało się ustalić jedynie, że do celów rozpoznawania używane są „skomplikowane klasyfikatory".

Następnym etapem było znalezienie informacji o próbach implementacji podobnych systemów na innych uczelniach i problemach na jakie się podczas tej implementacji natykano. Cel tego przedsięwzięcia był dwojaki. Po pierwsze, uniknięcie powtarzania błędów poprzedników, a po drugie, pomoc w dobraniu kolejnych etapów algorytmu.

3.4 Przypadki użycia

Po analizie wymagań przeprowadzonej w podrozdziałach „Wymagania funkcjonalne" i „Wymagania niefunkcjonalne" należało przemyśleć przypadki użycia które opisują sposób działania aplikacji. Poniżej zaprezentowany został jeden diagram (Schemat 3.1) ponieważ jedynym aktorem aplikacji jest użytkownik. A skoro jest on jedynym aktorem, to nie ma podziału na role.



Schemat 3.1 Diagram przypadków użycia.

Przypadki użycia:

UC 1. Przetwarzanie danych w trybie automatycznym i ich równoczesna analiza.

Scenariusz główny (wideo)

Jak w scenariuszu UC 2.

Scenariusz alternatywny 1

Jak w scenariuszu UC 4.

UC 2. Przetwarzanie sekwencji wideo.

Scenariusz główny:

1-5 Jak w przypadku użycia UC 3.

6. Pobieranie kolejnej klatki.

7. Rozpoznawanie kolejnej klatki.

8. Wyświetlenie rozpoznanych obiektów na odpowiednim panelu.

9. Zaznaczenie rozpoznanych obiektów na ramce docelowej.

10. Powrót do punktu 6.

Scenariusz alternatywny 1 (brak rozpoznanych obiektów):

1-7. Jak w scenariuszu głównym.

8. Powrót do punktu 6.

Scenariusz alternatywny 2(koniec filmu):

1-9 Jak w Scenariuszu Głównym.

10. Zamknięcie sekwencji.

UC 3. Otwieranie sekwencji wideo.

Scenariusz główny:

1. System pokazuje okno w którym użytkownik wybiera plik do otworzenia.

2. Sprawdzane jest rozszerzenie pliku.

3. System wykrywa rozszerzenie obsługiwanego typu sekwencji wideo.

4. System zamyka okno wyboru pliku.

5 . System otwiera strumień wideo.

Scenariusz alternatywny 1:

1-2. Jak w scenariuszu głównym.

3. Wykryte błędne rozszerzenie.

4. System zwraca wiadomość o niepowodzeniu otwarcia pliku.

UC 4. Przetwarzanie statycznego obrazu.

Scenariusz główny:

1-5 Jak w przypadku użycia UC 5.

6. Rozpoznawanie klatki.

7. Wyświetlenie rozpoznanych obiektów na odpowiednim panelu.

8. Zaznaczenie rozpoznanych obiektów na ramce docelowej.

Scenariusz alternatywny 1(brak rozpoznanych obiektów):

1-6. Jak w scenariuszu głównym.

UC 5. Otwieranie statycznego obrazu.

Scenariusz główny:

1. System pokazuje okno w którym użytkownik wybiera plik do otworzenia.

2. Sprawdzane jest rozszerzenie pliku.

3. System wykrywa rozszerzenie obsługiwanego typu pliku.

4. System zamyka okno wyboru pliku.

5 . System otwiera plik.

Scenariusz alternatywny 1:

1-2. Jak w scenariuszu głównym.

3. Wykryte błędne rozszerzenie.

4. System wyświetla wiadomość o niepowodzeniu otwarcia pliku.

UC 6.Wczytywanie danych konfiguracyjnych z pliku.

Scenariusz główny:

1. System otwiera okienko do wybrania pliku konfiguracyjnego.

2. System wczytuje dane z pliku i aktualizuje swoje dane

Scenariusz alternatywny:

1. System otwiera okienko do wybrania pliku konfiguracyjnego.

2. Następuje błąd wczytywania i system zostaje przy danych domyślnych.

UC 7. Wczytanie danych kształtu danej grupy znaków z pliku.

Scenariusz główny:

1. System otwiera okno w którym użytkownik wybiera plik CSV do wczytania.

2. System wczytuje plik i aktualizuje dane potrzebne do rozpoznawania kształtów.

Scenariusz alternatywny:

1. System otwiera okno w którym użytkownik wybiera plik CSV do wczytania.

2. Błąd parsowania. System Wyświetla wiadomość o błędzie i zostaje przy danych domyślnych.

UC 8. Przetwarzanie danych w trybie manualnym i ich równoczesna analiza.

Scenariusz główny:

1. Pobranie z Odpowiedniego panelu wartości parametrów segmentacji.

2. Przetworzenie ramki.

3. Wyświetlenie przetworzonej ramki wraz z maskami.

Scenariusz alternatywny(wyświetlenie danych obiektu):

1-3. Jak w Scenariuszu głównym.

4-7. Jak 1.4.Scenariusza głównego UC 10.

UC9. Zmiana parametrów segmentacji.

Scenariusz główny:

1. Zmiana opcji na odpowiednim panelu.

2. Odświeżenie obecnej ramki.

UC 10. Wyświetlenie informacji o danym obiekcie.

Scenariusz główny:

1. System wykrywa kliknięcie na jednym z obszarów wysegmentowanych.

2. System odnajduje obiekt do którego należy zaznaczony piksel.

3. System wyszukuje dane które zgromadził na temat obiektu.

4. System Wyświetla okno z danymi obiektu.

Scenariusz alternatywny (kliknięty piksel nie należy do żadnego potencjalnego obiektu):

1. Jak w Scenariuszu głównym.

3.5 Użytkowanie systemu

System będzie przeznaczony do użytkowania go na komputerze stacjonarnym bądź laptopie. W celu badania właściwości sekwencji wideo i zawartych na niej znaków będzie można używać trybu manualnego a w celu oglądania automatycznego rozpoznawania sekwencji będzie można używać trybu automatycznego.

Będą się one różniły tym, że w trybie manualnym będą wyświetlane maski obrazu na których znajdują się interesujące z punktu widzenia znaków drogowych obiekty. W trybie automatycznym nie będzie widocznych masek dla każdej z barw, ale będą widoczne tylko ramka źródłowa i docelowa, z zaznaczonymi na niej rozpoznanymi obiektami. Nie będzie również widoczny panel z opcjami segmentacji które w trybie manualnym są możliwe do ustawienia. Do tego będzie również widoczny panel z umieszczonymi na nim rozpoznanymi obiektami z obecnej ramki.

Bardziej szczegółowy opis użytkowania można przeczytać w rozdziale zatytułowanym: Załącznik 2. Instrukcja użytkownika.

4 Znaki drogowe

Znaki drogowe w większości krajów zostały ustandaryzowane konwencjami w Viennie. Podsumowanie owej konwencji [4] dzieli znaki na kilka podstawowych podgrup, takich jak:

- Znaki ostrzegawcze
- Znaki regulujące
    - Znaki pierwszeństwa
    - Znaki zakazu
    - Znaki nakazu
    - Znaki regulujące specjalne
- Znaki informacyjne
    - Znaki informacyjne, informujące o budynkach specjalnych, albo znaki serwisowe
    - Znaki kierunku, pozycji i wskazujące
    - Dodatkowe panele

W niniejszej pracy zostaną uwzględnione tylko 4 najliczniejsze podgrupy znaków. Są nimi znaki ostrzegawcze, zakazu nakazu i informacyjne. Umiejscowienie owych grup w hierarchii jest pokazane powyżej.

4.1 Znaki ostrzegawcze

Istnieją dwa modele znaków Ostrzegawczych A^a i A^b [4].

Znaki zgodne z modelem A^a (Rysunek 4.1) powinny być z trójkątami równobocznymi mającymi jedną krawędź poziomą i przeciwległy wierzchołek nad nią. Tło może być białe lub żółte a granica znaku (obwódka) powinna być czerwona. Zaś znaki zgodne z modelem A^b (Rysunek 4.2)są kwadratowe z jedną przekątną pionową tło jest żółte a granica znaku(obwódka) która jest tylko obręczą jest czarna.

Dopóki opis nie będzie mówił inaczej symbole wyświetlane na tych znakach powinny być czarne albo ciemno niebieskie, granatowe. Rozróżnia się znaki normalne i małe. Wielkość znaków, w modelu A^a dla rozmiaru małego musi być większa od 0.60m a dla rozmiaru normalnego powinna być w granicach 0.90m. Wymiary modelu A^b dla znaku małego muszą być większe niż 0.40m a dla znaku normalnego wynoszą w granicach 0.60m.

System który jest przedstawiony w tej pracy zajmuje tylko znaki w wersji polskiej, czyli trójkąty z żółtym tłem.

Rysunek 4.1 Przykładowe znaki ostrzegawcze według modelu A[a].

Rysunek 4.2 Przykładowe znaki ostrzegawcze według modelu A[b].

4.2 Znaki zakazu

Wykonuje się je według modelu B. Znaki zakazu [4] powinny być okrągłe. Ich średnica powinna być nie mniejsza niż 0.60m lub 0.40m z wyjątkiem zakazujących bądź ograniczających stawanie i parkowanie w obszarach zabudowanych.

Dopóki specyfikacja nie mówi inaczej znaki zakazu i ograniczające powinny mieć białe, żółte albo niebieskie tło dla znaków zakazujących bądź ograniczających prawo do parkowania bądź postoju. Symbole i inskrypcje powinny być czarne albo granatowe, a ukośne przekreślenia, jeżeli takie na danym znaku są, powinny być czerwone i opadać od lewej do prawej.

System przedstawiony w niniejszej pracy zajmuje się tylko znakami bez czerwonych przekreśleń(wyłączając zakaz zatrzymywania).

Rysunek 4.3 Przykładowe znaki nakazu według modelu C.

4.3 Znaki nakazu

Znaki nakazu [4] powinny być okrągłe a ich średnica powinna być nie mniejsza niż 0.60 metra poza obszarem zabudowanym i 0.40 metra wewnątrz niego. Jednakże znaki ze średnicą powyżej 0.30 metra mogą zostać użyte w połączeniu ze światłami drogowymi albo na pachołkach na wysepkach wyłączonych z ruchu.

Dopóki nie jest powiedziane inaczej znaki nakazu powinny być niebieskie a znajdujące się na nich symbole powinny być białe, albo jasnego koloru, albo alternatywnie, znaki powinny być białe z czerwoną obręczą a symbole na nich będące czarne.

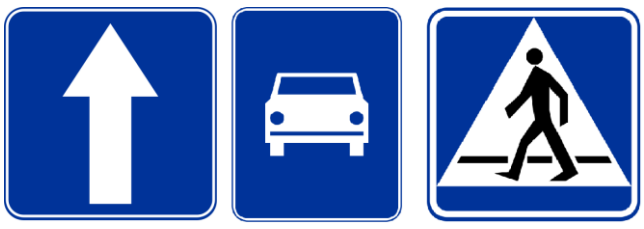
System przedstawiony w niniejszej pracy zajmuje się tylko niebieskimi znakami bez czerwonych przekreśleń.

Rysunek 4.4 Przykładowe znaki nakazy według wzorca C.

4.4 Znaki informacyjne

Znaki informacyjne skonstruowane są według modelu F, który mówi o tym, że znaki z nim zgodne powinny mieć niebieskie tło. Powinny one również nosić na tym tle biały lub żółty prostokąt, na którym będzie narysowany symbol, który z kolei powinien być czarny lub granatowy. Jeżeli znaki posiadają przekreślenie to powinno być ono czerwone i opadać od prawego górnego do lewego dolnego rogu.

System przedstawiony w niniejszej pracy zajmuje się tylko znakami bez czerwonych przekreśleń.



Rysunek 4.5 Przykładowe znaki informacyjne według wzorca F.

4.5 Umiejscowienie znaków w klatkach obrazu (wpływ tła na zlewanie)

Pionowe znaki drogowe najczęściej pojawiają się po prawej stronie drogi lub po nad jej powierzchnią przypięte do rusztowań lub w jakiś inny sposób. W związku z tym można zaobserwować, że w miarę przybliżania się znaku przesuwa się on coraz bardziej w prawo i do góry. Można z tego wywnioskować że obszarem o największym natężeniu występowania znaków jest górna – prawa ćwiartka ramki.

W związku z tym znaki często znajdują się na tle tego krajobrazu, który mijamy z prawej strony, ewentualnie na tle nieba. Rodzajem tła, które sprawia wiele kłopotów jest niebieskie niebo w bezchmurny dzień. Znaki nakazu i informacyjne które znajdą się na jego tle zleją się z tłem ze względu na jego barwę i nasycenie bardzo podobne do barwy i nasycenia tła.

Następnym nagminnie powtarzającym się problemem jest zlewanie się barwy zielonej z żółtą, (na przykład gdy znak ostrzegawczy znajduje się na tle lasu.), jako że w przestrzeni barw HSV znajdują się one zaraz obok siebie.

5 Analiza problemu i zastosowane rozwiązania

System powinien rozpoznawać cztery najliczniejsze grupy znaków. Wszystkie znaki znajdujące się w kręgu zainteresowania są znakami stojącymi. Ich kształty są w większości kształtami wielokątów prostych. Zostały one zaprojektowane w ten sposób aby ich składowe były wysoko kontrastowe i aby ich barwy były możliwie rzadko występującymi w przyrodzie kolorami o dosyć wysokim nasyceniu.

Z tego powodu w projekcie opisanym tutaj autor zdecydował się użyć segmentacji przez progowanie w przestrzeni HSV, co pozwoliło mu w łatwy sposób wyodrębnić barwę, Niestety w trakcie tworzenia projektu okazało się że sama barwa nie wystarcza i zaszła konieczność wyodrębnienia obszarów o większej od pewnego ustalonego na podstawie badań progu składowej Saturacji.

Z powodu mocnego ustandaryzowania kształtu interesujących autora obiektów rozważano użycie współczynników kształtu lub niezmienników momentowych. Finalnie odrzucono możliwość używania współczynników kształtu ze względu na zbyt dużą ich wrażliwość na zmianę skali[5]. Konsekwencją tej wrażliwości mogłaby być sytuacja, w której dwie figury o podobnych rozmiarach ale różnych kształtach mogłyby być bardziej podobne (patrząc na wartości współczynników kształtu) niż dwie figury o identycznym kształcie ale różnych rozmiarach.

5.1 Omówienie wymagań

Po ukończeniu etapu projektowania konceptu aplikacji, który został przedstawiony w poprzednich rozdziałach należy omówić wymagania implementacji systemu.

5.2 Komunikacja podzespołów

System przedstawiony w tej pracy jest skonstruowany na podstawie wzorca projektowego Model-Widok-Kontroler. Odpowiednikiem widoku jest w omawianym systemie interfejs napisany w Qt w wersji 5.2.0 [6]. Zdegenerowanym kontrolerem jest klasa Model, która przechwytuje sygnały otrzymywane od widoku i na podstawie danych przesłanych za pośrednictwem ich argumentów podejmuje określone akcje. Modelem jest zbiór klas odpowiadających za struktury danych i przetwarzanie obrazów.

Głównym mechanizmem który jest używany do komunikacji pomiędzy modelem a widokiem jest mechanizm slotów zaczerpnięty z Qt, który zostanie opisany w podrozdziale 5.4.5.

5.3 Ograniczenia projektu i zakres jego realizacji

Jak zostało opisane wcześniej projekt zakrojony został na dosyć dużą skalę. Mnogość detali koniecznych do zaimplementowania i analiz do przeprowadzenia powoduje, że niemożliwym stało się zaimplementowanie tego rozwiązania w pełnym zakresie w tak krótkim czasie.

Z związku z tym autor zdecydował się na pominięcie implementacji elementów aplikacji wymienionych poniżej:

- szczegółowego rozpoznawania wszystkich znaków w grupach zainteresowania wymienionych wcześniej. Autor ograniczył się do rozpoznawania po jednym znaku w każdej grupie ponieważ wystarcza to do względów demonstracyjnych. Dalsza implementacja wymaga zebrania większej ilości danych testowych jak również przeprowadzenia dalszych analiz kształtów których wykrywanie musiałoby zostać zaimplementowane.
- Struktura plików przechowujących dane została bardzo uproszczona, przechowywane są w nich tylko dane zewnętrznych konturów obiektów.
- Zapisywanie i odczytywanie danych z plików zostało uproszczone, obsługiwane są tylko zewnętrzne dane konturów.

5.4 Implementacja systemu

W niniejszym podrozdziale zostanie opisany sposób implementacji systemu, w tym:

- Użyte narzędzia,
- Użyty język programowania i System operacyjny
- Zastosowane biblioteki i narzędzia
- Komponenty systemu.
- Interfejsy służce do komunikacji między komponentami systemu.

5.4.1 Narzędzia

W początkowym etapie rozwoju projektu autor używał środowiska programowania Eclipse CDT (ang. C/C++ Deevelopment Tooling). Projekt CDT dostarcza w pełni funkcjonalnego zintegrowanego środowiska programowania opartego na Platformie Eclipse. Niestety pomimo wielu zalet Eclipse nie wspiera projektów łączonych z Qt co utrudnia wykorzystanie jego pełnych możliwości. Nie wspiera ono również automatycznego uzupełniania funkcji Qt, co bardzo utrudnia pisanie kodu.

W związku z tym autor zdecydował się zmienić środowisko pracy na Qt Creator, który jest środowiskiem dedykowanym do pisania kodu wykorzystującego cały potencjał Qt.

5.4.1.1 CMake

Kolejnym narzędziem użytym przez twórcę jest narzędzie CMake [7]. Jest to wieloplatformowe narzędzie służące do automatycznego zarządzania procesem kompilacji programu. Jego główne cechy to:

- Niezależność od używanego kompilatora.
- Niezależność platformy sprzętowej.
- Niezależność od systemu operacyjnego.
- Obsługa złożonych drzew katalogów.
- Kompilacja skrośna.
- Możliwość kompilowania w oddzielnym pod-folderze. Skutkiem tego jest porządek, gdyż pliki wynikowe kompilacji nie mieszają się z plikami źródłowymi.
- Mechanizm wykrywania zależności i bibliotek zewnętrznych.

Świetnym przykładem możliwości tego narzędzia jest, kilka linijek kodu które pozwalają zbudować projekt zawierający zarówno elementy Qt , zewnętrzną bibliotekę OpenCV i standardowe biblioteki C++.

```
01 cmake_minimum_required(VERSION 2.8)
02
03 SET(CMAKE_BUILD_TYPE Debug)
04
05 project(RoadSignsRecognition)
06
07 find_package(OpenCV REQUIRED)
08 find_package(Qt5Widgets REQUIRED)
09
10 SET(CMAKE_INCLUDE_CURRENT_DIR ON)
11
12 SET(CMAKE_AUTOMOC ON)
13
14 include_directories("src/" "input/" "qualityImprover/" "recognition/")
15
16 SET(     RoadSignsRecognition_SOURCES src/RoadSignsRecognition.cpp
17          src/Model.cpp  input/VideoReader.cpp
18          input/InputReader.cpp qualityImprover/QualityImprover.cpp
19          qualityImprover/Segmenter.cpp  recognition/Identifier.cpp
20          qualityImprover/DataCollector.cpp GUI/MainWindow.cpp
21          GUI/OptionsPane.cpp src/ModelController.cpp
22          GUI/PicturesPane.cpp GUI/PictureLabel.cpp GUI/ObjectDialog.cpp)
23
24 include_directories(${Qt5Widgets_INCLUDES})
25 include_directories(${Qt5Widgets_INCLUDE_DIRS})
26 add_definitions(${Qt5Widgets_DEFINITIONS})
27
28 ADD_DEFINITIONS(${QT_DEFINITIONS})
29 ADD_DEFINITIONS(${Qt5Widgets_DEFINITIONS})
30
31 SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
31 ${Qt5Widgets_EXECUTABLE_COMPILE_FLAGS}")
32
33 qt5_generate_moc(RoadSignsRecognition.cpp RoadSignsRecognition.moc)
34
35 ADD_EXECUTABLE(RoadSignsRecognition ${RoadSignsRecognition_SOURCES} )
36 TARGET_LINK_LIBRARIES(RoadSignsRecognition  ${Qt5Widgets_LIBRARIES}
37                                             ${OpenCV_LIBS})
```

Listing 5.1 Zawartość głównego pliku CMakeLists.txt

Linia 03 ustawia typ budowania kodu. W trakcie implementacji był on ustawiony w tryb debug, aby było możliwe śledzenie kodu linijka po linijce. W liniach 07 i 08 CMake skanuje system w celu znalezienia bibliotek potrzebnych do zbudowania aplikacji. Linia 12 instruuje CMake aby uruchomił MOC(ang. Meta Object Compiler) automatycznie kiedy będzie to potrzebne. Linia 25 dodaje katalogi nagłówków modułu Qt5 Widgets. Linia 31 i 32 Dodaje wymagane flagi kompilacji podczas budowania plików wykonywalnych. Linia 33 generuje plik o rozszerzeniu .moc dla głównego pliku źródłowego programu. Linia 35 mówi aby CMake stworzył plik wykonywalny o nazwie *RoadSignsRecognition*. Następnie łączy plik wykonywalny z wymaganymi bibliotekami Qt i OpenCV.

Przedstawiony powyżej przykład pomimo swojej małej objętości pozwala na zbudowanie aplikacji w której zachodzi konieczność integrowania bibliotek zewnętrznych.

5.4.2 Język programowania i system operacyjny.

Jako, że system przeznaczony w niniejszej pracy został przeznaczony na komputery stacjonarne bądź mobilne (laptopy), istnieje możliwość wyboru spośród wielu języków programowania i systemów operacyjnych. Autor zdecydował się na wybór systemu operacyjnego Linux i języka programowania C++ w połączeniu z narzędziem do budowania aplikacji CMake. Pobudki kierujące jego poczynaniami zostały przedstawione poniżej.

Do problemu dobrania systemu, narzędzi i języka programowania można podejść dwojako, a mianowicie można tą decyzje rozpatrywać z perspektywy końcowego użytkownika albo z perspektywy autora. Każda z tych perspektyw powinna zostać wzięta pod uwagę.

Z perspektywy końcowego użytkownika pewną rolę odgrywają:

- Funkcjonalność interfejsu użytkownika
- Dostępność systemu operacyjnego na którym aplikacja zostanie uruchomiona
- Interakcyjność (szybkość odpowiedzi)

Z perspektywy autora, czyli osoby która bierze udział w wytwarzaniu oprogramowania znaczenie mają zupełnie inne aspekty:

- Szybka i bezproblemowa instalacja bibliotek które będą wykorzystywane podczas wytwarzania oprogramowania.
- Środowisko programistyczne które wspiera wytwarzanie kodu programu w wybranym języku
- Język programowania powinien być znany programiście
- Założony język programowania musi być zgodny z API wykorzystywanych bibliotek zewnętrznych

Autor aplikacji posiada niejakie doświadczenie w programowaniu w języku Java, ale dużo większe obycie ma on z językiem C++. Wybranie znanego programiście języka jest niezmiernie ważne, gdyż w przeciwnej sytuacji przed implementacją nastąpi faza uczenia składni nowego języka co spowoduje niepotrzebny narzut czasowy.

Zarówno Qt jak i OpenCV posiadają API w języku C++ więc będzie istniała możliwość sprzężenia tych bibliotek bez nieprzewidzianych komplikacji.

Drugą braną pod uwagę opcją było wybranie języka programowania Java. Z uwagi na to że jest on wykonywany na maszynie wirtualnej JVM (Java Virtual Machine) istniało przypuszczenie że aplikacja napisana w Javie będzie działać wolniej,

a przy tych narzutach obliczeniowych, które prawdopodobnie będzie miała aplikacja podczas przetwarzania może to mieć znaczący wpływ na szybkość jej działania (interakcyjność). Poza tym istnieje również problem z brakiem interfejsu Javy dla Qt , co zmusiłoby programistę do wykorzystania innej biblioteki umożliwiającej stworzenie interfejsu użytkownika.

Kolejną decyzją do podjęcia było wybranie systemu operacyjnego. W dzisiejszych czasach zarówno systemy Windows jak i systemy Linux są szeroko dostępne. Przeciwko systemom Windows przemawia ich płatna licencja.

W związku z wątpliwościami co do wydajności rozwiązania wytworzonego za pomocą technologii Java autor postanowił pozostać przy języku C++. Zaś to z kolei pociągnęło za sobą wybór systemu z rodziny Linux, ponieważ owe systemy znacznie bardziej sprawdzają się przy wytwarzaniu kodu w języku C-++.

Jak zostało wcześniej wspomniane narzędzie CMake jest niezależne od platformy więc jako takie nie nakłada ograniczeń na wybór systemu. Za to po dokonaniu zmian w głównym pliku C++ powinno być możliwe zbudowanie kodu programu w systemie Windows.

5.4.3 Zastosowane biblioteki i narzędzia

W niniejszym rozdziale zostaną omówione biblioteki zastosowane podczas wytwarzania kodu aplikacji.

- Qt Widgets – Moduł Qt, który został wykorzystamy do wytworzenia interfejsu użytkownika.
- OpenCV – biblioteka która została wykorzystana do zaimplementowania wielu funkcjonalności modelu związanych z przetwarzaniem obrazów.

5.4.3.1 QtWidgets (GUI)

Moduł Qt Widgets dostarcza zbioru elementów do tworzenia klasycznych interfejsów użytkownika w stylu odpowiadającym aplikacjom desktopowym. Widżety (ang. Widget – najogólniej mówiąc jest to element widoku) integrują się świetnie z platformą na której są wyświetlane dopasowując się do jej natywnego wyglądu. Widżety służą do przedstawiania różnorodnych elementów interfejsów użytkownika odpowiednich głównie do statycznych interfejsów. Są one dobrym wyborem dla aplikacji z tradycyjnym desktopowym centrycznym UI (ang. User Interface) występującym głównie w aplikacjach typu offline.

Elementy wchodzące w skład tego modułu posiadają zaimplementowane metody które pozwalają w bardzo łatwy sposób powiadamiać jedne z nich o akcjach wywołanych na innych. Mechanizmem służącym w tym celu jest mechanizm sygnałów i slotów opisany w podrozdziale „Mechanizm sygnałów slotów w Qt”.

Nawet jeżeli istniejące domyślnie zaimplementowane sygnały i sloty w danym widżecie nie spełniają dokładnie wymagań jakie stawia przed nimi ich użytkownik (programista) można w bardzo łatwy sposób dostosować dany widżet poprzez dziedziczenie. Tworząc własną klasę i dziedzicząc ją po widżecie który w największym stopniu spełnia wymagania projektanta, a następnie dopisując do niej własne sloty i sygnały można w dowolny sposób rozszerzać funkcjonalności widżetów.

Bogactwo widżetów i ich elastyczność sprawiają, że są one bardzo wygodną technologią do tworzenia interfejsów użytkownika i było jedną z głównych przyczyn wybrania tego frameworka do tworzenia interfejsu użytkownika aplikacji omawianej w tej pracy.

5.4.3.2 OpenCV

OpenCV jest biblioteką wydawaną na licencji BSD a stąd wynika, że jest ono darmowe dla zastosowań zarówno akademickich jak i komercyjnych. Posiada ono interfejsy w językach C, C++, Java i Python oraz wspiera systemy operacyjne takie jak Windows, Linux, MacOS, iOS and Android .

OpenCV zostało zaprojektowane w celu efektywności obliczeniowej i z mocnym naciskiem na aplikacje czasu rzeczywistego. Napisana w zoptymalizowanym C/C++ biblioteka może czerpać korzyści płynące z wielowątkowego przetwarzania.

Zasięg jej zastosowań jest bardzo szeroki, zaczynając się na interaktywnej sztuce poprzez łączenie fragmentów map a na zaawansowanej robotyce kończąc.

Jest ona podzielona na moduły z których każdy posiada pewne zaimplementowane funkcjonalności. Modułami które były najczęściej używane podczas tworzenia kodu opisywanego programu są:

- *Core*
    - bazowe struktury danych
    - zaimplementowane operacje na macierzach
    - funkcje rysujące proste kształty
- *Imgproc*
    - filtracja obrazów,

- analiza strukturalna i deskryptory kształtu

- *Highgui*
  - wczytywanie i zapisywanie obrazów
  - wczytywanie i zapisywanie sekwencji wideo
  - prosty interfejs użytkownika (używany głównie do celów wyświetlania pomocniczych obrazów)

Istnieją również inne biblioteki wspomagające przetwarzanie obrazów jednak OpenCV w pełni zaspokaja wymagania projektu. Posiada ono obszerną dokumentację, która mimo wszystko czasami w sposób mocno niewystarczający opisuje działanie niektórych funkcji. W tym miejscu zaletą okazuje się popularność owej biblioteki i duża społeczność użytkowników, która jest bardzo pomocna w rozwiązywaniu problemów takich jak nie sprecyzowane argumenty funkcji, czy niedokładnie określone znaczenie zwracanych danych.

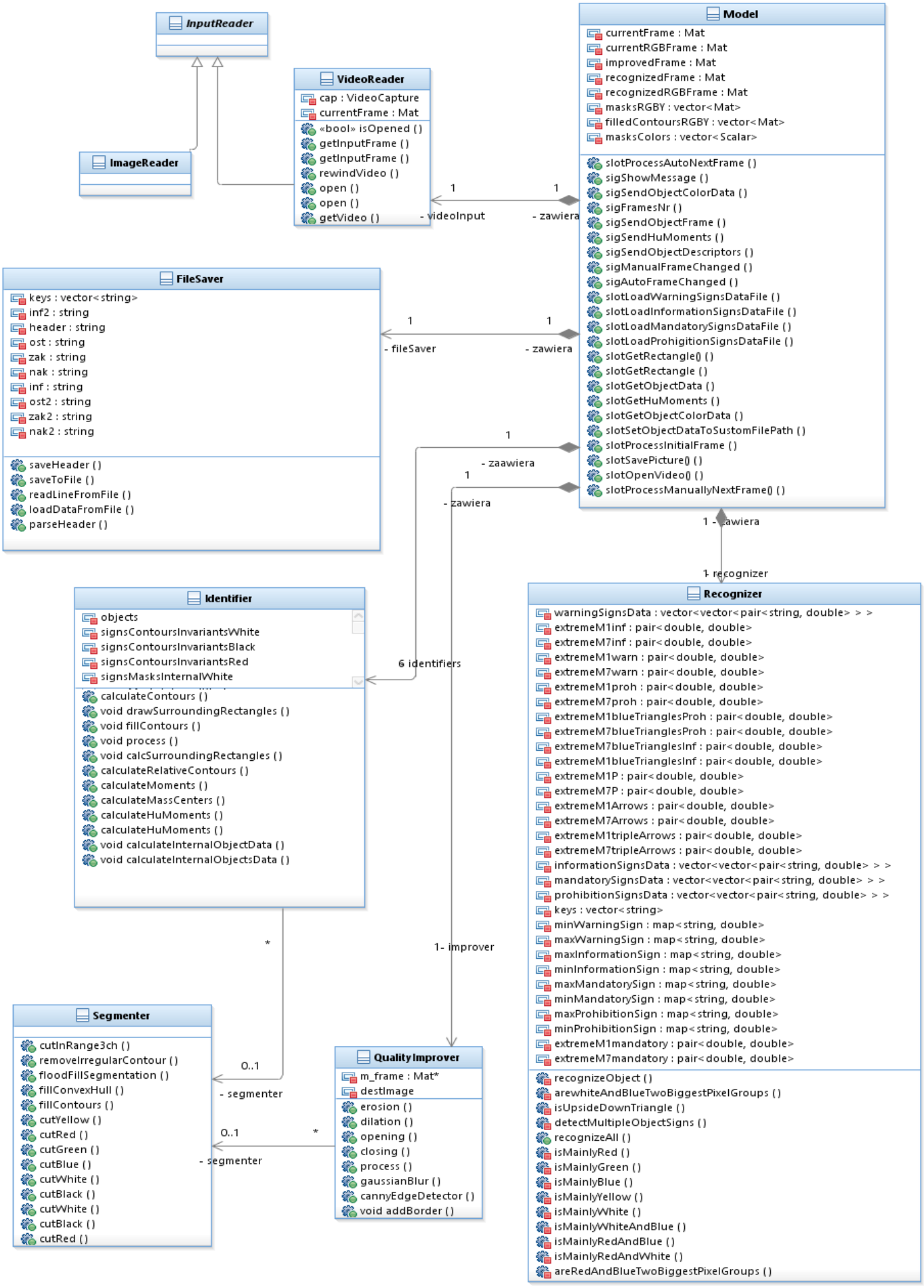
5.4.4 Komponenty

Jednym z założeń projektowych było aby możliwe było rozbudowywanie aplikacji w przyszłości. Z tego powodu została ona podzielona na różne komponenty, według kryterium funkcjonalności. Zostały one przedstawione w podrozdziałach (od 5.4.4.1 do 5.4.4.6).

Każdy komponent jest reprezentowany przez jedną klasę, a diagram poniżej (Schemat 5.2) przedstawia relacje między nimi. Algorytm przetworzenia jednej klatki przedstawiono poniżej.

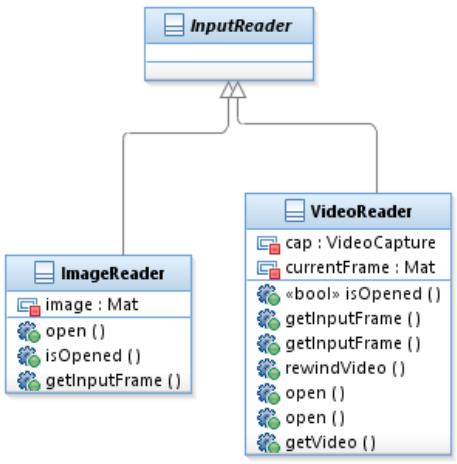
1. Pobranie następnej ramki.
2. Poprawa jakości ramki.
3. Segmentacja obrazu
4. Ewentualne zastosowanie przekształceń morfologicznych (otwarcie, zamknięcie).
5. Przekazanie odpowiednich masek i obrazów źródłowych do obiektów identyfikujących (6 masek = 6 identyfikatorów)
6. Wyznaczenie danych potrzebnych do identyfikacji.
7. Przekazanie danych potencjalnych obiektów do obiektu rozpoznającego.
8. Próba rozpoznania obiektów hierarchicznie (najpierw kształt zewnętrzny i kolor, potem obiekty wewnątrz).

9. Próba rozpoznania obiektów złożonych z kilku podobiektów (np. cztery „trójkąty ” wchodzące w skład zakazu zatrzymywania).
10. Zwrócenie danych potrzebnych do zaznaczenia rozpoznawanych obiektów.
11. Jeżeli aplikacja w trybie automatycznym to przesłanie informacji o rozpoznanych obiektach do widoku.
12. Zaktualizowanie widoku.
13. Powrót do punktu pierwszego.



Schemat 5.1 Diagram klas modelu aplikacji (uproszczony).

5.4.4.1 Wejście



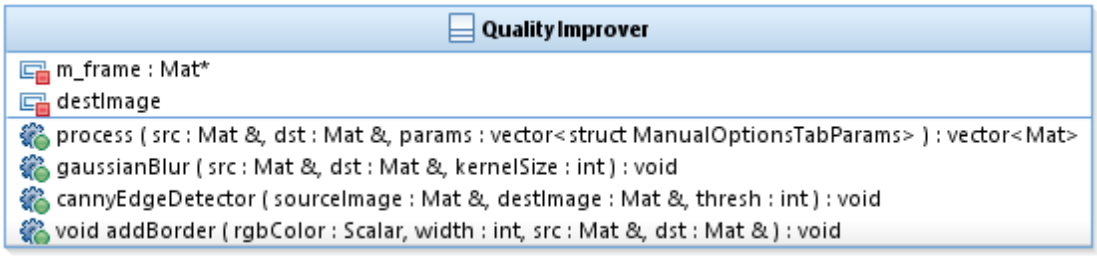
Schemat 5.2 Schemat klas modułu wejściowego

Moduł wejściowy aplikacji składa się z klasy wirtualnej *InputReader* i dwóch dziedziczonych po niej typów.

Jeden z nich odpowiada za otwieranie i odczytywanie sekwencji wideo. Istnieje również możliwość przewijania sekwencji wideo i oczywiście możliwość wyłuskiwania ze strumienia ramek.

Kolejny odpowiada za otwieranie i przechowywanie statycznego obrazka jeżeli zajdzie taka możliwość, i ewentualne przekazywanie go dalej.

5.4.4.2 Poprawa jakości



Schemat 5.3 Schemat klasy QualityImprover.

Moduł opisany w tej części pracy odpowiada za poprawę jakości obecnie przetwarzanej ramki. Główną metodą owej klasy jest metoda

```
vector<Mat> process(Mat & src, Mat & dst,
          vector<struct ManualOptionsTabParams> params);
```

która jako parametry przyjmuje referencje na obrazek źródłowy, referencje do obiektu w którym zostanie zapisany obrazek po poprawieniu jakości i wektor struktur w których są zapisane parametry dla segmentacji. Metoda zwraca wektor masek. Każdy element wektora zawiera maskę zawierającą obiekty o odpowiedniej barwie (w kolejności: czerwona, zielona, niebieska, żółta, biała).

W pierwszej kolejności na obrazie jest stosowane rozmycie gaussowskie a następnie na jego wynik nakładany jest filtr medianowy. W późniejszym etapie wzmacniane są krawędzie obiektów poprzez zastosowanie na obrazie filtru Canny'ego w następnie splecenie jego wyniku z każdym z trzech kanałów obrazu przefiltrowanego. Odbywa się to według wzoru:

$$dst[i] = 0.2 * cannyOutput + 0.8 * src[i]$$

gdzie

i – numer kanału

dst – obrazek docelowy

dst[i] – i-ty kanał obrazka docelowego

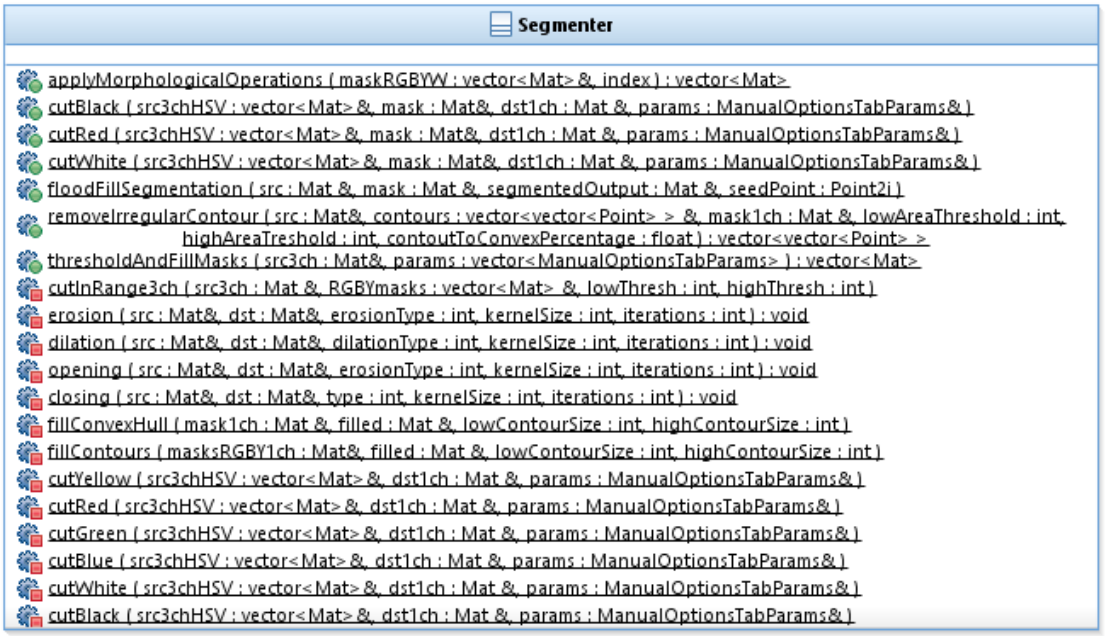
cannyOutput – maska zawierająca krawędzie

src – obrazek źródłowy

src[i] – i-ty kanał obrazka źródłowego

Następnie podnoszona zostaje nieco jasność całego obrazu i funkcja kończy swoje działanie zapisując jego wynik w miejsce wskazywane przed argument `Mat & dst`.

5.4.4.3 Segmentacja



Schemat 5.4 Schemat klasy Segmenter

Zadaniem owej klasy jest wydzielenie obiektów zainteresowania od tła. Realizuje ona to poprzez wywołanie metody

```
static vector<Mat> thresholdAndFillMasks(Mat & src3ch,
                    vector<ManualOptionsTabParams> params);
```

w której odbywa się segmentacja obrazu. Metoda zwraca wektor masek w którym każdy element wektora jest maską odpowiadającą za oddzielną barwę. Metoda zwraca tyle masek ile struktur *ManualOptionsParams* zostało przekazane w parametrze *params*.

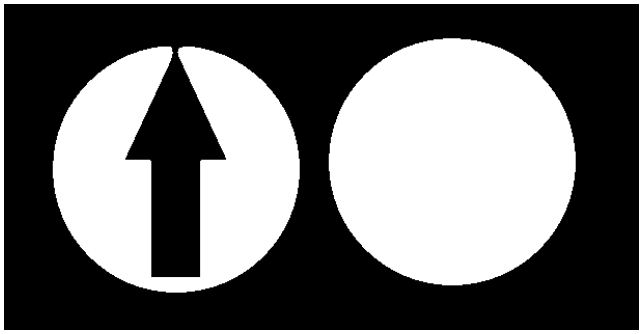
W początkowym etapie metody obraz źródłowy jest konwertowany do przestrzeni barw HSV a następnie rozdzielany na oddzielne kanały. Kanał Hue jest wycinany w zakresie przekazanym w odpowiednim parametrze *params*. Następnie na tej samej zasadzie wycinane są kanały Saturation i Value. W końcowym etapie maski uzyskane ze wszystkich trzech kanałów są poddawane działaniu bitowego mnożenia. W ten sposób uzyskiwana jest ich część wspólna.

W kolejnym etapie uzyskane maski są poddawane działaniu metod

```
static void fillConvexHull(Mat & mask1ch, Mat & filled, int
                lowContourSize, int highContourSize);
static void fillContours(Mat& masksRGBY1ch, Mat & filled,
                int lowContourSize, int highContourSize);
```

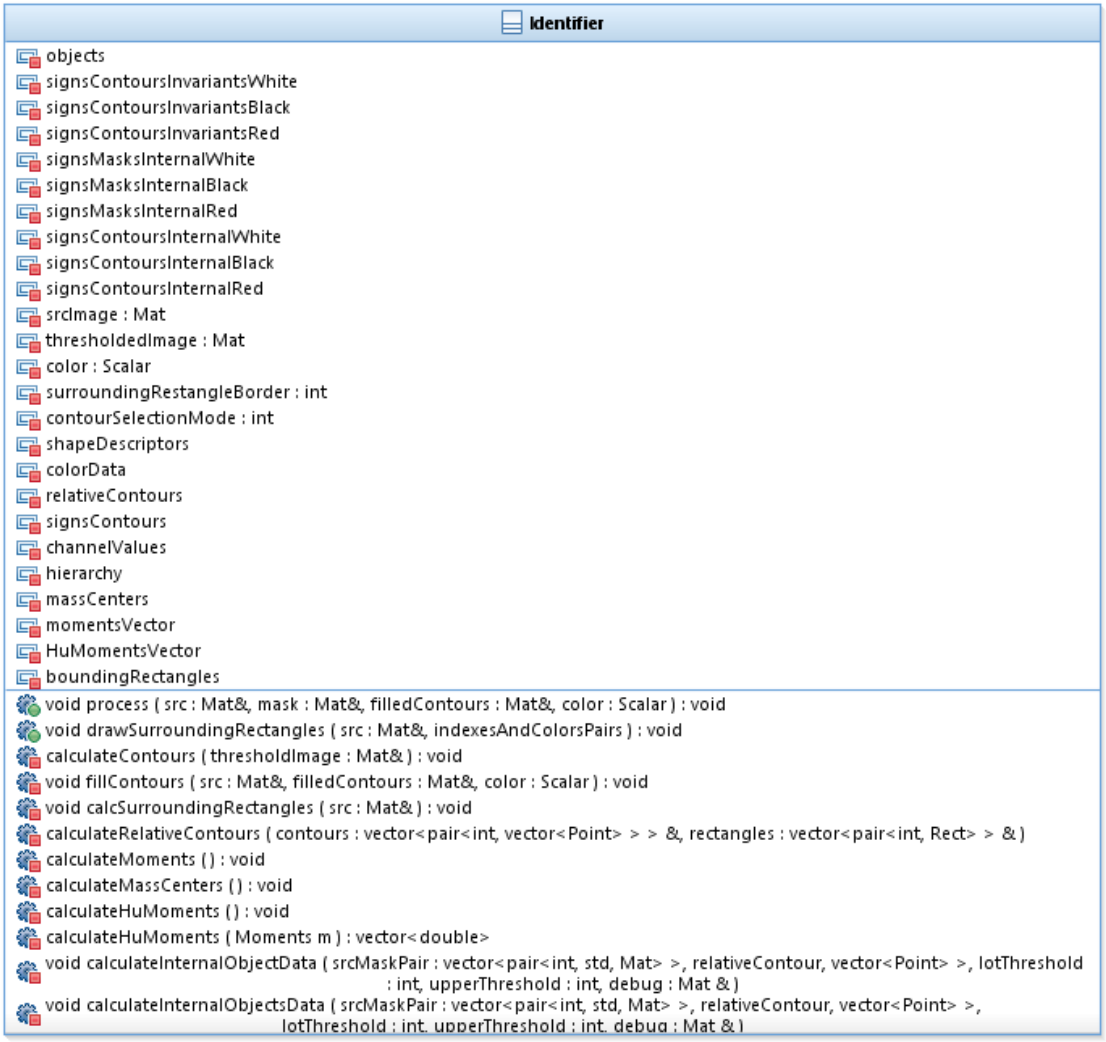
Pierwsza z nich odpowiada za wyznaczenie krzywych wypukłych obiektów maski a następnie ich wypełnieniu. Jest to pomocne w celu wypełnienia konturu który poprzez niedokładną segmentację został w jakiś sposób przerwany a co za tym idzie nie został do końca wypełniony. Działanie metody zostało przedstawione na rysunku „Rysunek 5.1”. Druga z metod jest prostsza, ponieważ odpowiada tylko za wypełnienie obiektów widocznych na masce w takiej postaci w jakiej się tam znajdują.

Finalnie główna metoda zwraca maski zawierające potencjalne obiekty zainteresowania. Na każdej z masek znajdują się obiekty innego koloru.



Rysunek 5.1 Ilustracja sposobu działania metody Segmenter::fillConvexHull().

5.4.4.4 Identyfikacja



Schemat 5.5 Schemat klasy Identifier

Klasa *Identifier* jest jednym z najważniejszych elementów programu. To właśnie w niej są obliczane wszelkie dane liczbowe potrzebne do późniejszego identyfikowania obiektów. Główną metodą odpowiadającą za uzyskanie wszystkich danych jest metoda

```
void process(  Mat & srd, Mat & mask,  Mat&filledContours,
               Scalar color);
```

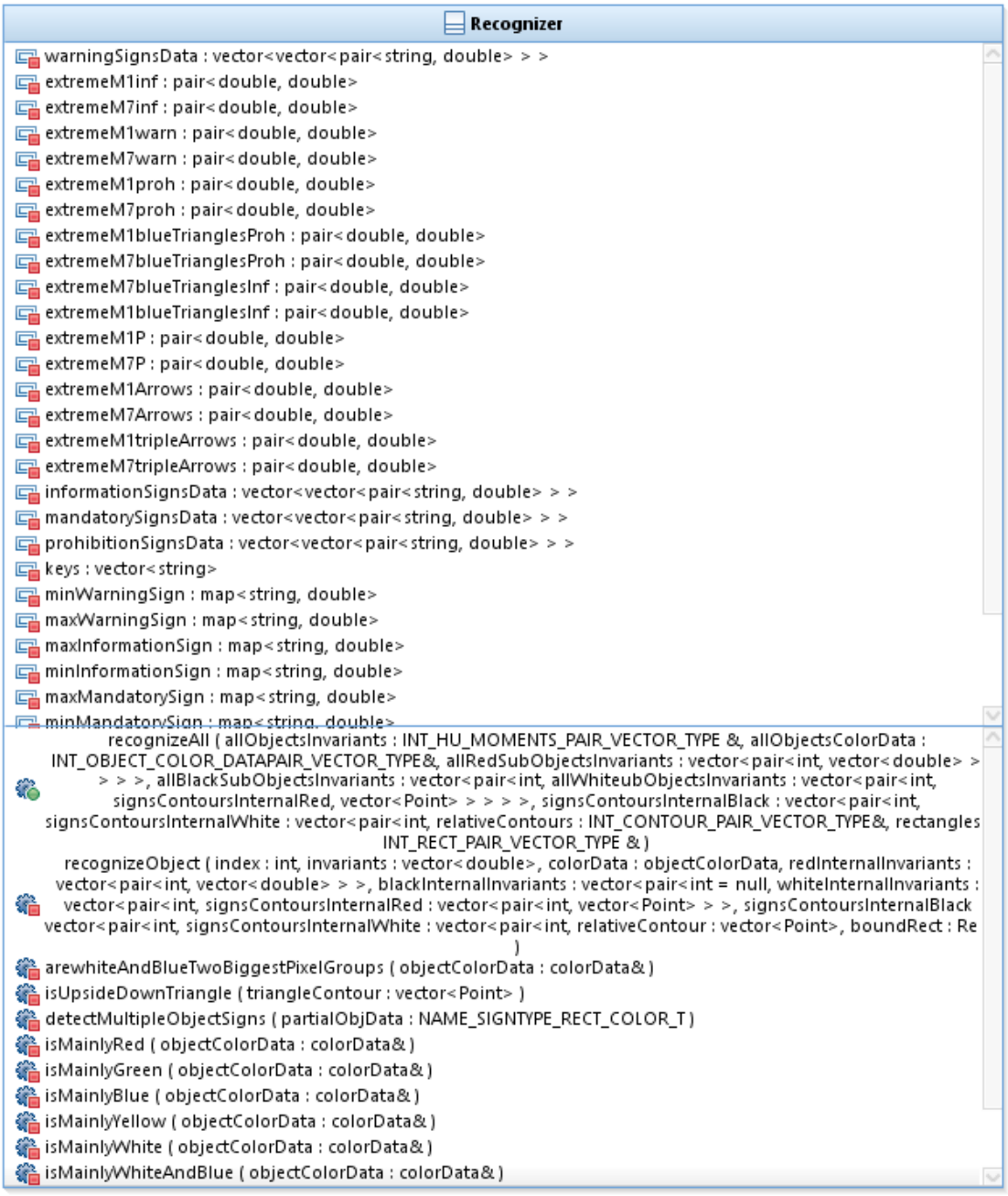
której proces działania można przedstawić w następujących punktach:

1. Na wstępie ustawiony zostaje kolor, który został wysegmentowany na wcześniejszym etapie działania programu przez *Segmenter.*
2. Następnie wyliczane są kontury wszystkich obiektów znajdujących się na podanej w argumencie masce.

3. Na podstawie wyliczonych konturów przechowywanych wewnątrz obiektu wyliczane są momenty geometryczne obiektów.
4. Na podstawie otrzymanych momentów geometrycznych obliczane są środki masy obiektów.
5. Korzystając z obliczonych wcześniej momentów geometrycznych obliczane są niezmienniki momentowe dla każdego obiektu.
6. Kontury znajdujące się na masce są po raz kolejny wypełniane w celu zalania ewentualnych znajdujących się w nich dziur. (Ten podpunkt ma walory tylko estetyczne, nie zmienia on w żaden sposób obliczeń).
7. Obliczane są prostokąty opisane na wszystkich obiektach znajdujących się na masce.
8. Obliczane są względne kontury obiektów (współrzędne punktów konturów obliczonych w punkcie 2. jako punkt odniesienia traktowały początek całego obrazu, a współrzędne konturów względnych są obliczane względem lewego górnego rogu prostokąta opisanego na danym obiekcie).
9. Tworzone są obiekty ROI każdego obiektu znajdującego się na masce, jak również ich odpowiedniki na obrazie źródłowym.
10. Dla każdego obiektu obliczone są dane dotyczące gabarytów (obwód, pole) i rozłożenia ilościowego kolorów w danym obiekcie.
11. Dla każdego obiektu zewnętrznego wyznaczane są obiekty wewnętrzne koloru białego, czarnego i czerwonego i wszystkie dane ich dotyczące.
    a. Obliczane są kontury elementów
    b. Jeżeli dany kontur jest mniejszy od 20 pikseli jest usuwany z listy obiektów wewnętrznych danego koloru.
    c. Następnie dla każdego pozostałego konturu zostają obliczone momenty geometryczne a na ich podstawie niezmienniki momentowe

Po przetworzeniu danej ramki wszystkie dane jej dotyczące są przechowywane wewnątrz danego obiektu.

5.4.4.5 Rozpoznawanie



Schemat 5.6 Schemat klasy Recognizer.

Klasa *Recognizer* odpowiada za rozpoznawanie obiektów. Może być ona zainicjowana na kilka sposobów:

- Danymi z pliku CSV, w którym są przechowywane niezmienniki momentowe zewnętrznych konturów obiektów – w takim przypadku wartości potrzebne do rozpoznawania wewnętrznych obiektów są inicjowane wartościami domyślnymi zapisanymi bezpośrednio w kodzie
- Danymi z pliku konfiguracyjnego XML //TODO

- Nie inicjowana w ogóle – używane są dane domyślne zapisane bezpośrednio w kodzie

Główna metoda funkcji, *recognizeAll*, przyjmuje jako argumenty dane zaczerpnięte z obiektu *Identifier*. Dal każdego obiektu przesłanego do owej funkcji odnajduje ona odpowiadające mu kontury, dane kolorów i dane o wszystkich obiektach znajdujących się we wnętrzu danego obiektu. Następnie dla tych danych wywoływana jest metoda *recognizeObject* i następuje przetwarzanie pojedynczego obiektu.

Pierwszym etapem jest rozpoznawanie kształtu zewnętrznego i sprawdzanie jego koloru. Pozwala to na wczesnym etapie wyeliminować obiekty skrajnie niezgodne z wzorcami o których rozpoznawaniu traktuje ta praca. Na przykład już na wstępie jesteśmy w stanie odrzucić zielone trójkąty albo żółte koła.

Następnie rozpatrywana jest ilość obiektów wewnętrznych jakie zostały wykryte w obiekcie zewnętrznym i ich kolor ponieważ jak zostało zauważone w rozdziale 4 określone grupy znaków posiadają obiekty wewnętrzne o ściśle sprecyzowanej kolorystyce.

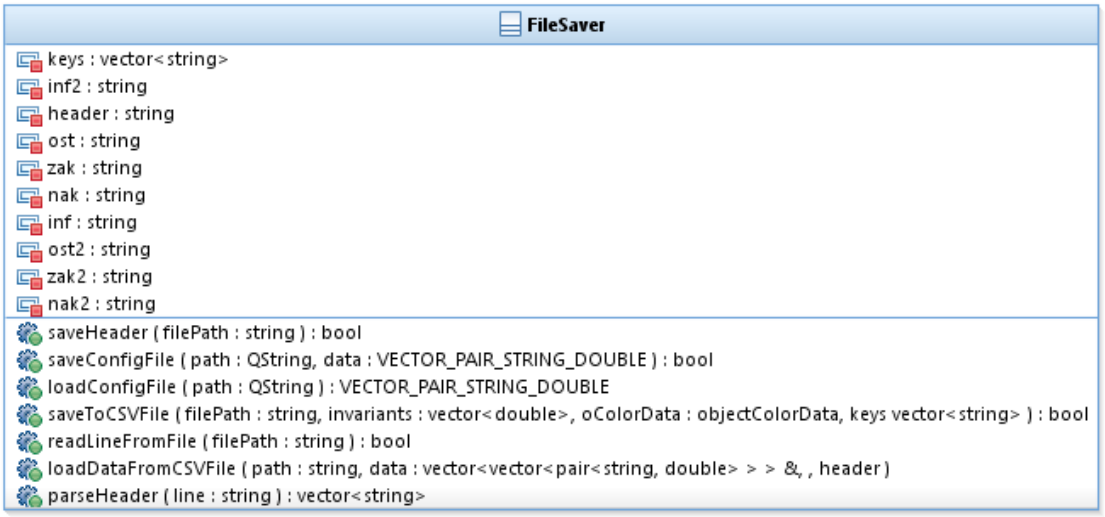
W razie gdy kształt i kolorystyka obiektu zewnętrznego nie pasują do żadnej z głównych grup istnieje również możliwość, że obiekt ów jest częścią większego znaku drogowego. Przykładem takich obiektów mogą zostać niebieskie elementy zakazu zatrzymywania albo dwa czerwone półkola zakazu wjazdu przedstawione na rysunku poniżej (Rysunek 5.2).

Rysunek 5.2 Rysunek przedstawiający przyklady znaków złożonych (od lewej: a) zakaz zatrzymywania, b)informacyjny przejście dla pieszych)

W takim wypadku należy pogrupować obiekty jako kryterium stosując odległość ich środków masy od siebie. Jest to zasadne ponieważ znaki drogowe są konstrukcjami zwartymi, a co za tym idzie ich elementy znajdują się w niedalekim oddaleniu od siebie. Na przykład jeżeli dany kontur odpowiada kształtem ćwiartce znaku zakaz

zatrzymywania (Rysunek 5.2 a) należy sprawdzić czy w jego sąsiedztwie nie znajduje się jeszcze trzy inne obiekty o takim samym kształcie. Jeżeli zostanie dopasowana odpowiednia ilość obiektów należy sprawdzić czy środki ich ciężkości znajdują się w równej odległości od siebie, czyli czy tworzą kwadrat.

5.4.4.6 Przechowywanie danych.(struktura, zapis, odczyt)



Schemat 5.7 Schemat klasy FileSaver

Klasa *FileSaver* jest klasą odpowiadającą za zapisywanie i odczyt danych obiektów z plików. Jej głównymi metodami są

```
bool saveToFile(    std::string path,
                    std::vector<double> inveriants,
                    struct objectColorData & colorData,
                    std::vector<std::string> keys =
                    std::vector<std::string>(),
                    std::string sequencePath =
                         std::string());

std::vector<std::vector<std::pair<std::string,double> > >
loadDataFromFile(   std::string path,
                    vector<vector<pair<string, double> > >
                         & data,
                    vector<string> header);
```

Pierwsza z nich odpowiada za zapisywanie danych do pliku CSV w dwojaki sposób. Pierwszy z nich jest to zapisanie nagłówka z nazwami kolejnych kolumn a w każdym następnym wierszu zapisywane zostają dane liczbowe jednego obiektu w odpowiednim porządku. Drugi sposób nie zapisuje nagłówka ale w każdej komórce pliku CSV zapisuje się para klucz i wartość przedzielone znakiem „=" (n.p. M0=0.19345).

Wybór metody zapisywania jest dokonywany poprzez przekazanie (bądź nie) wektora z kluczami jako argumentu funkcji. Program domyślnie zapisuje dane w plikach obu formatów. Forma zapisu „key=value" jest bardziej czytelna dla ludzkiego oka, ale pierwszy format był częściej używany przez twórcę w celu badania fluktuacji wartości współczynników dla obserwowanych obiektów. Jako, że można go bezpośrednio otworzyć w każdym arkuszu kalkulacyjnym umożliwia to bardzo łatwe rysowanie wykresów i wyznaczanie zakresów zmienności współczynników dla danych grup obiektów.

Druga z głównych funkcji odpowiada za wczytywanie danych z plików CSV o formatowaniu takim, jak opisane wyżej. Zwraca ona dane wszystkich elementów które znajdowały się w pliku.

Kolejne dwie funkcje odpowiadają za obsługę pliku komunikacyjnego który można wykorzystywać do przechowywania parametrów rozpoznawania obrazów. Owymi funkcjami są:

```
bool saveConfigFile(QString path,
                    vector<pair<string, double> > params);

vector<pair<string, double> > loadConfigFile(QString path);
```

Do ich zaimplementowania użyte zostały obiekty *QXMLStreamReader* i *QXMLStreamWriter* z biblioteki Qt. Struktura pliku założona przez autora wygląda następująco.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <minWarningM1 minWarningM1="xx.xxxxxxxx"/>
    <maxWarningM1 maxWarningM1="xx.xxxxxxxx"/>
    <minWarningM7 minWarningM7="xx.xxxxxxxx"/>
    <maxWarningM7 maxWarningM7="xx.xxxxxxxx"/>
    <minInformationM1 minInformationM1="xx.xxxxxxxx"/>
    <maxInformationM1 maxInformationM1="xx.xxxxxxxx"/>
    <minInformationM7 minInformationM7="xx.xxxxxxxx"/>
    <maxInformationM7 maxInformationM7="xx.xxxxxxxx"/>
    .
    .
    .
    <minArrowsM1 minArrowsM1="xx.xxxxxxxx"/>
    <maxArrowsM1 maxArrowsM1="xx.xxxxxxxx"/>
    <minArrowsM7 minArrowsM7="xx.xxxxxxxx"/>
    <maxArrowsM7 maxArrowsM7="xx.xxxxxxxx"/>
    <minTripleArrowsM1 minTripleArrowsM1="xx.xxxxxxxx"/>
    <maxTripleArrowsM1 maxTripleArrowsM1="xx.xxxxxxxx"/>
    <minTripleArrowsM7 minTripleArrowsM7="xx.xxxxxxxx"/>
    <maxTripleArrowsM7 maxTripleArrowsM7="xx.xxxxxxxx"/>
</config>
```

Listing 5.2 Struktura pliku konfiguracyjnego XML.

5.4.5 Interfejsy

W niniejszym rozdziale zostanie opisany sposób łączenia się widoku aplikacji z jej modelem. Mechanizmem, który był w tym celu wykorzystany jest mechanizm slotów dostarczony przez Qt. Jest to rozwiązanie pozwalające w bardzo wygodny sposób przesyłać komunikaty między tymi modułami kodu zarówno w jedną jak i w druga stronę.

5.4.5.1 Mechanizm sygnałów slotów w Qt

Sygnały i sloty są używane do komunikacji między obiektami. Podczas zmiany stanu jednego obiektu często chcemy aby inne obiekty zostały o tym powiadomione i reagując na tą zmianę statusu podjęły określone działania. Mechanizm sygnałów i slotów pozwala na zrealizowanie takiej komunikacji przez połączenie sygnału jednego obiektu z sygnałem (bądź wieloma sygnałami) innych obiektów.

- Sygnał - jest emitowany kiedy w obiekcie zajdzie jakieś szczególne zdarzenie.
- Slot - jest funkcją, która jest wywoływana w odpowiedzi na nadanie szczególnego sygnału

W momencie nadejścia zdarzenia – wywołania sygnału – wszystkie sloty które zostały z nim wcześniej połączone są wykonywane (zazwyczaj natychmiast). Kod który znajduje się po klauzuli *emit* (odpowie ona za emitowanie sygnału) zostaje wykonany dopiero po zakończeniu się wykonywania wszystkich slotów. Definiowanie sygnałów i slotów dostępne jest w klasach z dostępnym mechanizmem introspekcji, czyli takich które na początku swojego ciała posiadają makro Q_OBJECT i dziedziczą po klasie

QObject. Mechanizm sygnałów i slotów jest bezpieczny pod względem typów – sygnatura sygnału i slotu muszą się ze sobą pokrywać.

Sloty deklaruje się w sekcji *slots* a sygnały w sekcji *signals*. Są one łączone za pomocą metody *connect*, rozłączane za pomocą metody *disconnect*. W Qt sygnały są typu multicast, czyli każdy sygnał może mieć ze sobą połączone dowolnie wiele slotów.

6 Testy

W niniejszym rozdziale zostanie opisany sposób testowania powstałego programu. Przedstawiony zostanie spis testowych przypadków a następnie wyniki testów. Testy będą obejmować kilka aspektów działania aplikacji takich jak:

- odtwarzanie sekwencji wideo w różnych formatach
- odtwarzanie obrazów statycznych w różnych formatach
- wczytywanie danych konfiguracyjnych z plików
- działanie trybu auto
- działanie trybu manualnego.

Wszystkie dane znajdujące się w zbiorze uczącym zostały pobrane ze statycznych obrazów. Zdjęcia te były zrobione w innych miejscach niż filmy użyte do testowania aplikacji.

6.1 Przeprowadzone Testy Poboczne

Poza testami funkcjonalności głównych zostały również przeprowadzone testy funkcjonalności pobocznych. Wyniki i wnioski z ich przeprowadzenia przedstawione są w tabeli poniżej.

ID	Opis testu	Wynik testu
Test 1	Test wczytywania i przetwarzania obrazów statycznych	Wczytywanie obrazów statycznych, ich segmentacja i odświeżanie masek działała be zarzutu. Testowanie obejmowało tylko manualną segmentacje ponieważ tryb wczytania obrazu statycznego nie obejmuje jego rozpoznania. Ta funkcjonalność została zaprojektowana tylko w celu gromadzenia zestawu danych uczących.

Test 2	Wczytywanie plików konfiguracyjnych CSV	Wczytywanie plików konfiguracyjnych CSV działa pod warunkiem że wczytywane są pliki przygotowane wcześniej w tej samej aplikacji. Przy wczytywaniu dwudziestu różnych rozmiarów plików przygotowanych przez aplikację w trakcie prowadzenia badań nad doborem współczynników nie zostały zauważone żadne przekłamania ani luki we wprowadzonych danych.
Test 3	Wczytywanie plików konfiguracyjnych XML	Pliki mają ściśle ustaloną strukturę a w związku z tym muszą być tworzone przez aplikacje opisaną w niniejszej pracy. Jeżeli to wymaganie jest spełnione problemy nie występują. W razie błędu parsowania pliku program korzysta z danych domyślnych.

Tabela 6.1 Zestawienie testów pobocznych funkcjonalności.

Do testowania głównych scenariuszy działania aplikacji zostało wybrane 11 sekwencji testowych, które zostały uruchomione z domyślnymi, ustalonymi na podstawie eksperymentów, parametrami segmentacji i z wartościami niezmienników momentowych ustalonymi na podstawie danych uzyskanych ze zbioru uczącego.

6.2 Statystyki wykrywanych obiektów

Nazwa sekwencji	Ilości znaków	Ostrzegawcze	Informacyjne	Nakazu	Zakazu
1.mp4	Il. znak. obecnych	2	0	2	0
	Il. znak. rozp.	2	0	2	0
2.mp4	Il. znak. obecnych	0	1	1	2
	Il. znak. rozp.	0	1	0	1
3.mp4	Il. znak. obecnych	0	1	1	2
	Il. znak. rozp.	0	1	1	2
4.mp4	Il. znak. obecnych	2	3	0	1
	Il. znak. rozp.	1	3	0	1

5.mp4	Il. znak. obecnych	0	1	0	2
	Il. znak. rozp.	0	1	0	2
6.mp4	Il. znak. obecnych	1	2	3	0
	Il. znak. rozp.	1	3	2	0
7.mp4	Il. znak. obecnych	3	1	2	0
	Il. znak. rozp.	3	0	0	0
8.mp4	Il. znak. obecnych	5	0	1	0
	Il. znak. rozp.	3	0	1	0
9.mp4	Il. znak. obecnych	1	0	0	1
	Il. znak. rozp.	1	0	0	1
10.mp4	Il. znak. obecnych	2	1	1	1
	Il. znak. rozp.	2	1	1	0
11.mp4	Il. znak. obecnych	1	0	0	0
	Il. znak. rozp.	1	0	0	0

Tabela 6.2 Wyniki przedstawionych testów.

Dane testowe przedstawione w powyższej tabeli są poglądowe i służą do wykazania że aplikacja spełnia wyznaczone jej funkcje. Są one jednak zbyt małe aby obliczyć statystyczne prawdopodobieństwa wykrycia znaków określonych grup. Tym bardziej są one zbyt małe aby statystycznie obliczyć prawdopodobieństwo przyporządkowania znaku do jakiegoś konkretnego reprezentanta danej grupy. Poprawianie i testowanie aplikacji na większym zbiorze danych będzie przedmiotem dalszych badań podczas procesu tworzenia pracy magisterskiej.

Jednocześnie zestaw sekwencji testowych dobrano tak, aby zawierał on znaki, które są obsługiwane przez aplikację nie tylko w zakresie określenia przynależności do grupy znaków, ale również dokładnego rozpoznawania znaku. W grę wchodziło po jednym znaku z każdej grupy:

- Ostrzegawcze – ustąp pierwszeństwa
- Informacyjne – droga jednokierunkowa
- Nakazu – skrzyżowanie o ruchu okrężnym
- Zakazu – zakaz zatrzymywania się

W tabeli poniżej przedstawiono wyniki badań tych określonych znaków. W tym przypadku próbka testowa jest również zbyt mała, aby móc skutecznie, statystycznie przebadać skuteczność aplikacji. Jednak na podstawie rozpoznawania znaku ustąp pierwszeństwa można stwierdzić, że aplikacja ma potencjał aby rozpoznawać znaczny

odsetek znaków, jeżeli zostaną prawidłowo wykryte ich zewnętrzne kształty. W celu przeprowadzenia takich testów należy zaimplementować rozpoznawanie wszystkich konkretnych przedstawicieli rozpatrywanych grup. Ułatwi to także zbieranie danych testowych, gdyż procentowo większa ilość pozyskanych sekwencji będzie zawierała znaki możliwe do dokładnego rozpoznania przez aplikację.

Nazwa sekwencji		Ustąp pierwszeństwa	Droga jednokierunkowa	Skrzyżowanie o ruchu okrężnym	Zakaz zatrzymywania
1.mp4	Il. znak. obecnych	2	0	2	0
	Il. znak. rozp.	2	0	1	0
2.mp4	Il. znak. obecnych	0	1	0	1
	Il. znak. rozp.	0	1	0	1
3.mp4	Il. znak. obecnych	0	1	0	2
	Il. znak. rozp.	0	1	0	2
4.mp4	Il. znak. obecnych	2	1	0	1
	Il. znak. rozp.	2	1	0	1
5.mp4	Il. znak. obecnych	0	1	0	1
	Il. znak. rozp.	0	1	0	1
6.mp4	Il. znak. obecnych	1	0	0	0
	Il. znak. rozp.	1	0	0	0
7.mp4	Il. znak. obecnych	3	0	2	0
	Il. znak. rozp.	3	0	0	0
8.mp4	Il. znak. obecnych	5	0	0	0
	Il. znak. rozp.	3	0	0	0
9.mp4	Il. znak. obecnych	0	0	0	0
	Il. znak. rozp.	0	0	0	0
10.mp4	Il. znak. obecnych	1	0	1	0
	Il. znak. rozp.	1	0	1	0
11.mp4	Il. znak. obecnych	1	0	0	0
	Il. znak. rozp.	1	0	0	0

Tabela 6.3 Wyniki testów dokładnego rozpoznawania konkretnych znaków.

6.3 Fałszywie wykrywane obiekty

Podczas testowania aplikacji założono, że znak jest uznany za wykryty jeżeli notorycznie powtarza się on w kolejno występującej po sobie sekwencji ramek. Jest to

potrzebne ponieważ w rozpoznawaniu występuje dosyć duży odsetek rozpoznań obiektów które nie są znakami, a jedynie losowymi obiektami odpowiadającymi kształtem i kolorystyką rozpoznawanym znakom drogowym. Jest to problem, który nie został przewidziany podczas tworzenia aplikacji, sposób jego rozwiązania został przedstawiono w rozdziale 7.3.

6.4 Zestaw uruchomieniowy

Podczas tworzenia, aplikacja była uruchamiana i testowana na dwóch zestawach uruchomieniowych, które przedstawiono w tabelach poniżej:

System operacyjny	Ubuntu 14.04
Wersja OpenCV	2.4.9
Wersja Qt	5.2.0
Procesor:	Intel Core i5 M430 2.27GHz
Pamięć RAM	4 GB

Tabela 6.4 Zestaw uruchomieniowy 1

System operacyjny	Ubuntu 14.04
Wersja OpenCV	2.4.9
Wersja Qt	5.2.0
Procesor:	Intel Core i7 4790 3.60GHz
Pamięć RAM	8 GB

Tabela 6.5 Zestaw uruchomieniowy 2

7 Podsumowanie

W powyższym rozdziale przedstawiony jest zakres wykonanych prac i opinia na temat narzędzi i bibliotek używanych podczas jej tworzenia. Sugerowane są w nim również drogi dalszego rozwoju aplikacji.

7.1 Podsumowanie wykorzystanych bibliotek

Używanie przez okres kilku miesięcy pewnego zbioru bibliotek i narzędzi pozwala wysnuć daleko idące wnioski dotyczące ich przydatności przy tworzeniu aplikacji.

Moduł biblioteki Qt o nazwie Qt Widgets jest doskonałym narzędziem do tworzenia interfejsu użytkownika. Posiada do tego celu dedykowane środowisko graficzne. Zaletą tego środowiska (ale również pisanego kodu) jest prędkość tworzenia podstawowych interfejsów graficznych. Sprawa komplikuje się wraz ze wzrostem skomplikowania interfejsu. Środowisko graficzne przestaje być przydatne, ponieważ umożliwia ono korzystanie tylko z dedykowanych elementów GUI znajdujących się w bibliotece, a nie pozwala dostosowywać owych elementów poprzez dodawanie własnych slotów i modyfikowanie działania niektórych metod. Za to owe operacje dosyć gładko i bezproblemowo wykonuje się poprzez dziedziczenie swoich własnych elementów UI i przeciążanie metod których zmianę konstruktor uzna za stosowną.

OpenCV jest również bardzo przydatną biblioteką, zarówno ze względu na zaimplementowane w niej algorytmy jak i przemyślane i zoptymalizowane struktury danych, które twórca może wykorzystać do implementacji własnych algorytmów. Jednak ilość dostarczanych przez bibliotekę algorytmów jest na tyle duża, a ich zastosowanie na tyle szerokie, że z nawiązką wystarcza do podstawowych zadań przetwarzania obrazów.

Nie bez znaczenia jest również modułowość wspomnianych bibliotek, zarówno Qt jak i OpenCV. Pozwala to na dołączanie do projektu tylko części z nich które faktycznie są używane do realizacji projektu, bez konieczności włączania całych bibliotek w skład projektu.

7.2 Podsumowanie wykonanych prac

Pomimo faktu, iż aplikacji brakuje implementacji rozpoznawania konkretnych znaków większość wymagań funkcjonalnych została zrealizowana.

Występują problemy z wydajnością algorytmu spowodowane głównie zbyt dużym narzutem obliczeniowym na wyznaczenie danych obiektów znajdujących się wewnątrz wysegmentowanych regionów zainteresowania.

Kolejnym kłopotem okazało się zakładane na początku bezkontekstowe analizowanie klatek sekwencji wideo. W połączeniu z brakiem implementacji rozpoznawania wszystkich konkretnych znaków powoduje to wzrost ilości fałszywych rozpoznań znaków drogowych. W związku z tymi problemami w dalszej części pracy zostały przedstawione dalsze drogi rozwoju aplikacji.

7.3 Sugerowane drogi dalszego rozwoju aplikacji

Aplikacja przedstawiona w niniejszej pracy posiada potencjał to stania się w pełni funkcjonalnym systemem rozpoznawania znaków drogowych. Jednak aby tak się stało należy się zastanowić w jaki sposób należy wyeliminować jej ograniczenia.

W celu wyeliminowania problemów z wydajnością należy przede wszystkim podjąć próbę zrównoleglenia wykonywania obliczeń dotyczących obiektów. Obecnie są one wykonywane sekwencyjnie dla obiektów każdego koloru (regionów zainteresowania każdego koloru). Rozdzielność danych wejściowych potrzebnych do wykonania obliczeń pozwala przypuszczać że będzie to w pełni realizowalne.

Należy również wprowadzić mechanizmy przetwarzania kontekstowego sekwencji wideo. Jednym z nich są filtry czasowe sprawdzających czy w sekwencji ramek pojawia się ten sam obiekt i od tego uzależniających proces kwalifikacji do jakiejkolwiek grupy.

Kolejnym elementem przetwarzania kontekstowego jest śledzenie trajektorii ruchu znaku i na jej podstawie aproksymowanie miejsca jego pojawienia się w następnej ramce. Pozwoli to uniknąć przetwarzania za każdym razem całych ramek obrazu, a jedynie śledzić już wykryte znaki. Tylko raz na jakiś czas przeszukana zostanie cała ramka w celu sprawdzenia czy nie pojawiły się nowe obiekty.

8 Bibliografia.

[1] Mobileye, „Mobileye,” [Online]. Available: http://www.mobileye.com.pl/.

[2] Continental, „Continental,” [Online]. Available: http://www.conti-online.com/www/automotive_de_en/general/chassis/business_units/adas_en.html.

[3] K. Brkić, „An overview of traffic sign detection methods,” Zagreb.

[4] A. Nieznany, „Convention on Road Signs And Signals Report,” Vienna, United Nations Publication, 2006, pp. 31-57.

[5] R. Tadeusiewicz, w *Systemy Wizyjne Robotów Przemysłowych*, Warszawa, Państwowe Wydawnictwo Naukowe, 1991, pp. 150-157.

[6] „Qt Project,” [Online]. Available: http://qt-project.org/.

[7] „CMake,” [Online]. Available: http://www.cmake.org/.

[8] T. Ryszard i F. Mariusz, Rozpoznawanie obrazów, Warszawa: Państwowe Wydawnictwo naukowe, 1991.

[9] U.Zakir, A.N.J.Leonce i E.A.Edirisinghe, „Road Sign Segmentation Based on Color Spaces: A comparative study,” Insbruck, 2010.

9 Załącznik 1. Zawartość płyty CD

Na załączonej płycie znajdują się:

- tekst pracy w pliku *Rozpoznawanie znaków drogowych w sekwencjach wideo.pdf*,
- kod źródłowy programu wraz z instrukcją jego zbudowania w katalogu *RoadSignsRecognition*,
- pliki wideo wykorzystywane do testowania aplikacji w katalogu *WideoTest*,
- zbiór zdjęć wykorzystany do stworzenia bazy wiedzy w katalogu *ZdjBazaWiedzy*.

10 Załącznik 2. Instrukcja użytkownika.

Niniejszy rozdział zawiera przejrzystą instrukcję użytkownika. Opisuje ona czynności jakie należy wykonać aby korzystać z najważniejszych funkcjonalności programu. Omówione zostało korzystanie z dwóch trybów:

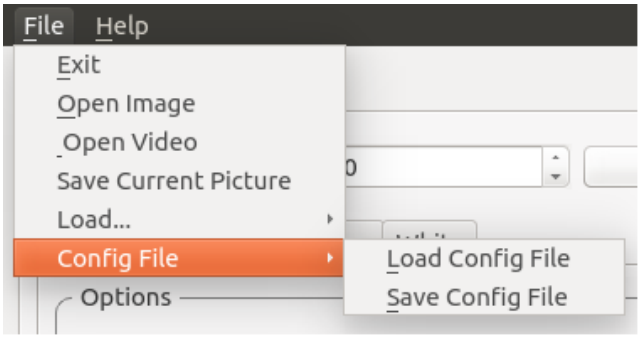
- Automatycznego – znajdując się w nim aplikacja wykorzystuje dane ze zbioru uczącego do rozpoznawania znaków
- Manualnego – w tym trybie działania aplikacja umożliwia zbieranie danych potrzebnych do kolekcjonowania zbiorów uczących. Oraz podglądanie wyników segmentacji. Można w nim również manipulować parametrami segmentacji.

Poruszony został również proces ładowania danych konfiguracyjnych z plików testowych.

10.1 Ładowanie plików z danymi

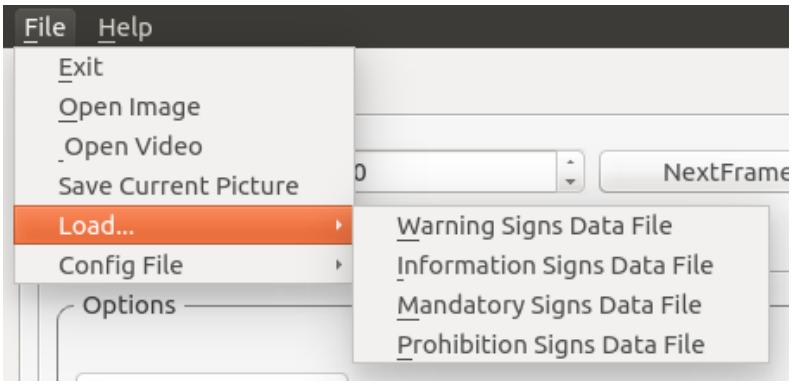
W aplikacji zostały na sztywno zapisane pewne dane pozwalające na rozpoznawanie obiektów, jednak użytkownik posiada możliwość załadowania innych danych które sam wyznaczył i zapisał wcześniej.

Aby wczytać dane konfiguracyjne z pliku XML należy z menu programu wybrać opcje *Config File -> Load Config File*. W oknie, które się pojawi użytkownik powinien wybrać plik, który chce wczytać. Aby mieć pewność że plik CSV jest zgodny z przyjętym formatem należy stworzyć do za pomocą niniejszej aplikacji.



Rysunek 10.1 Ścieżka prowadząca po menu programu w celu zapisania bądź odczytania pliku konfiguracyjnego.

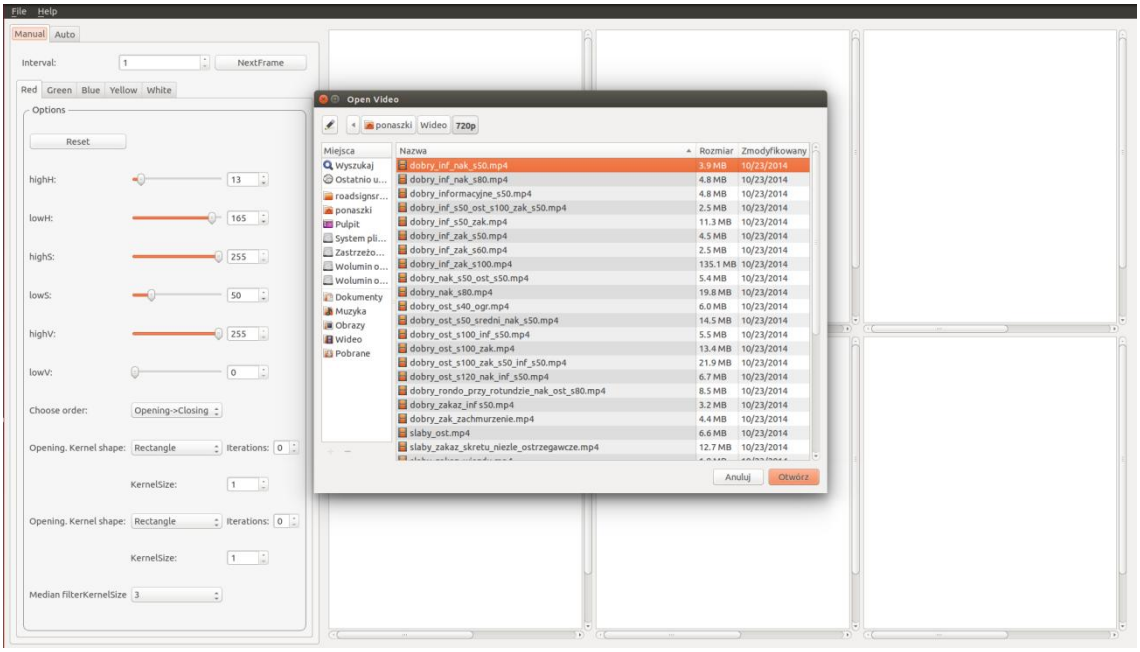
W tym celu należy z menu wybrać opcje *File->Config File->Save ConfigFile*. Następnie należy wypełnić formularz uzupełniając go odpowiednimi danymi, wskazać miejsce jego zapisania i potwierdzić klawiszem *Save*.



Rysunek 10.2 Ścieżka prowadząca po menu programu w celu wczytania plików CSV z danymi konturów zewnętrznych.

Również w przypadku plików CSV polecanym rozwiązaniem jest wczytywanie tych, które zostały utworzone za pomocą omawianej aplikacji. Również w tym przypadku jest to niezbędne w celu zachowania zgodności. Aby wczytać plik CSV należy wybrać opcję *File->Load…* a następnie jedną z opcji która pojawi się w rozwiniętym menu, Wybór opcji zależy od grupy danych dla których chcemy wczytać dane.

10.2 Przetwarzanie sekwencji wideo w trybie manualnym



Rysunek 10.3 Rysunek przedstawiający okno wyboru pliku na tle głównego okna aplikacji.

Pierwszą czynnością jaką należy wykonać aby rozpocząć pracę z aplikacją jest wybranie z menu *Plik* opcji otworzenia pliku wideo. Kiedy użytkownik kliknie na odpowiednią opcję ukaże mu się okno pozwalające na przemieszczanie się po całym systemie plików i wybranie odpowiadającego mu pliku wideo. Po zaznaczeniu

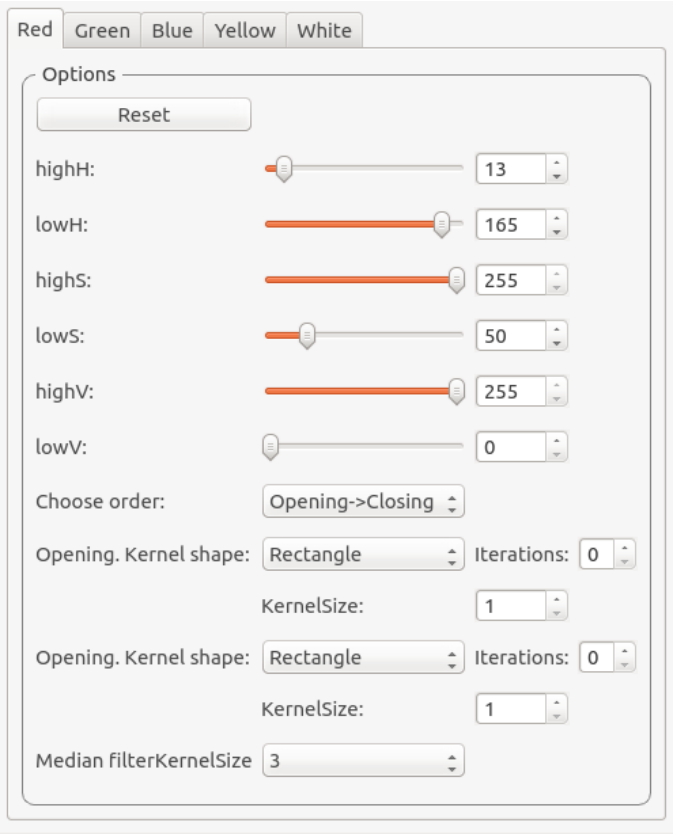
wybranego pliku należy zaakceptować swoją decyzję poprzez kliknięcie na przycisk *Otwórz* znajdujący się w prawym dolnym rogu aktywnego okna.

Po otwarciu pliku wideo nastąpi wstępne przetworzenie wybranego pliku i wczytanie pierwszej ramki i masek. Następnie użytkownik ma możliwość przetwarzania kolejnych ramek naciskając na klawisz *NextFrame* (Rysunek 10.4). Każde naciśnięcie na klawisz powoduje przetworzenie jednej ramki i zatrzymanie przetwarzania. Istnieje możliwość przekazania aplikacji ile ramek powinna ominąć przy wyborze kolejnej ramki. Realizuje się to poprzez wpisanie w pole poprzedzone etykietą *Interval:* (Rysunek 10.4) odpowiedniej wartości. Pozwala to przyspieszyć proces przetwarzania jeżeli użytkownik nie jest zainteresowany przetwarzaniem każdej ramki.

Rysunek 10.4 Rysunek przedstawiający górną część panelu opcji z przyciskami Next Frame, Reset i miejscem do wpisania interwału pomiędzy ramkami(na lewo od Next Frame).

Kolejną możliwością jest ponowne przetworzenie obecnie wyświetlanej ramki. Realizuje się ją poprzez kliknięcie przycisku *Reset* (Rysunek 10.4). Opcja przydatna gdy potrzeba obserwować jedną ramkę przy zmiennych parametrach segmentacji.

W czasie przetwarzania istnieje możliwość zmiany parametrów segmentacji dla następnych ramek. Można tego dokonać za pomocą panelu przedstawionego na rysunku niżej (Rysunek 10.5). Wszystkie parametry można ustawić niezależnie dla każdej z masek które chcemy segmentować. Panele odpowiadające za parametry segmentacji barw czerwonej, zielonej, niebieskiej i żółtej są jednakowe. Użytkownik ma możliwość ustawienia za ich pomocą zasięgu progowania każdego z kanałów HSV oddzielnie. Kolejnymi możliwymi do zastosowania przez użytkownika przekształceniami są przekształcenia morfologiczne takie jak otwarcie i zamknięcie.

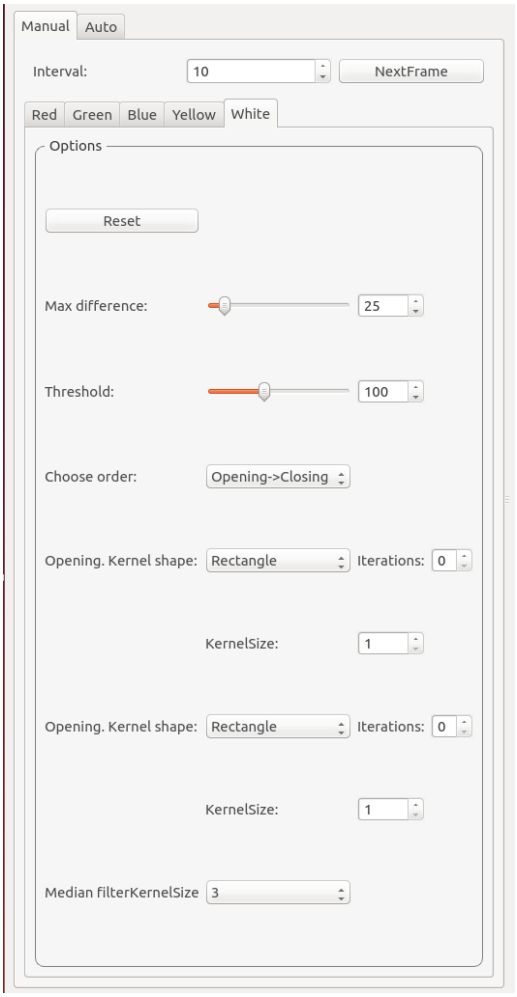


Rysunek 10.5 Część panelu opcji umożliwiająca ustawienie opcji segmentacji.

Użytkownik jest w stanie ustawić kilka parametrów ich dotyczących takich jak

- Kształt jądra które odpowiada za przekształcenie
- Ilość iteracji która jest na aplikowania na obiekty
- Rozmiar jądra które odpowiada za przekształcenie

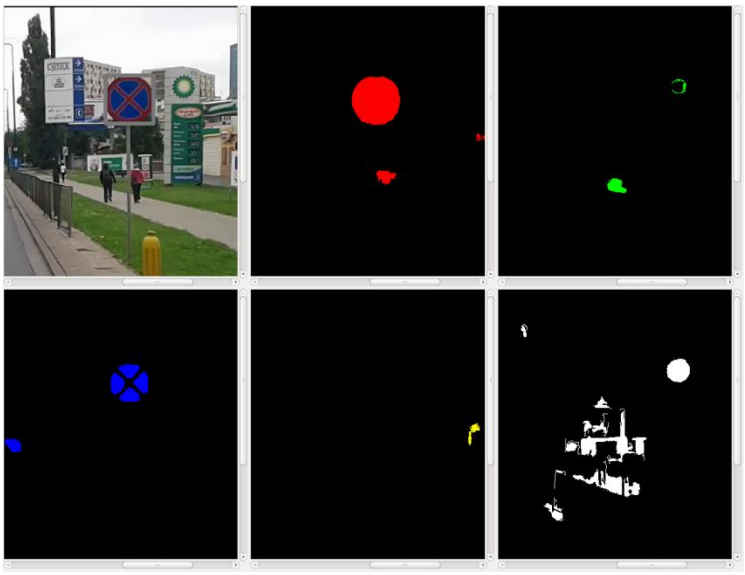
Nieco odmiennym przypadkiem jest panel dotyczący segmentacji białej barwy. Przedstawiony jest on na rysunku poniżej.



Rysunek 10.6 Rysunek przedstawiający wygląd panelu odpowiadającego za ustawianie parametrów białej barwy.

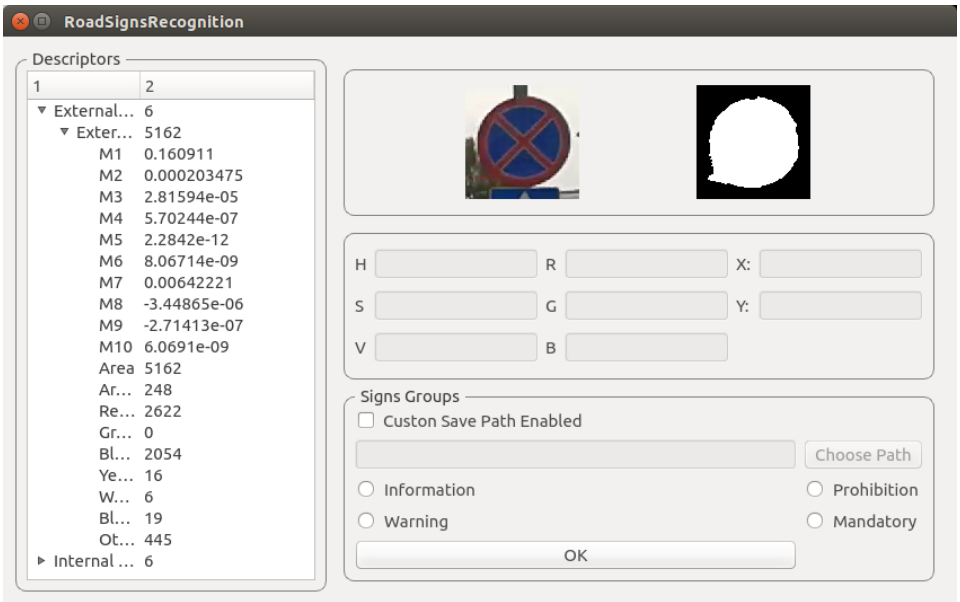
Segmentacja białego koloru odbywa się w przestrzeni barw RGB. Parametr *Max Difference* odpowiada za maksymalną różnicę między wartościami składowych piksela. Zaleca się ustawianie parametru na niewielkie wartości w celu wykrycia odcieni szarości. Kolejną wartością możliwą do ustawienia jest wartość progowa jasności (*threshold*) która ma być traktowana jako biała. Zaleca się ustawienie parametru w połowie zakresu.

Wyświetlanie kolejnych przetworzonych ramek odbywa się na panelu składającym się z sześciu obrazów, jednego źródłowego i pięciu masek (Rysunek 10.7). Gdy obrazy nie mieszczą się w całości na wyznaczonych panelach istnieje możliwość przewijania ich. Użytkownik może przewijać oddzielnie każdą z masek albo przewijać synchronicznie wszystkie naraz używając suwaki znajdujące się przy obrazie źródłowym. W trybie manualnym nie istnieje panel wyświetlający wykryte znaki. Są one tylko otaczane ramkami w celu zgrubnego ustalenia które obiekty spełniają warunki klasyfikacji do jednej z grup wedle obowiązujących obecnie zasad klasyfikacji.



Rysunek 10.7 Rysunek przedstawiający panel wyświetlający obraz wraz z maskami

Gdy ramka jest już przetworzona użytkownik ma możliwość kliknięcia na wykryte kontury w celu podejrzenia parametrów liczbowych obliczonych dla tego obiektu. Okno które zostaje wtedy wyświetlone zostało pokazane na rysunku niżej (Rysunek 10.8).



Rysunek 10.8 Okno pomocnicze zawierające dane wysegmentowanych obiektów.

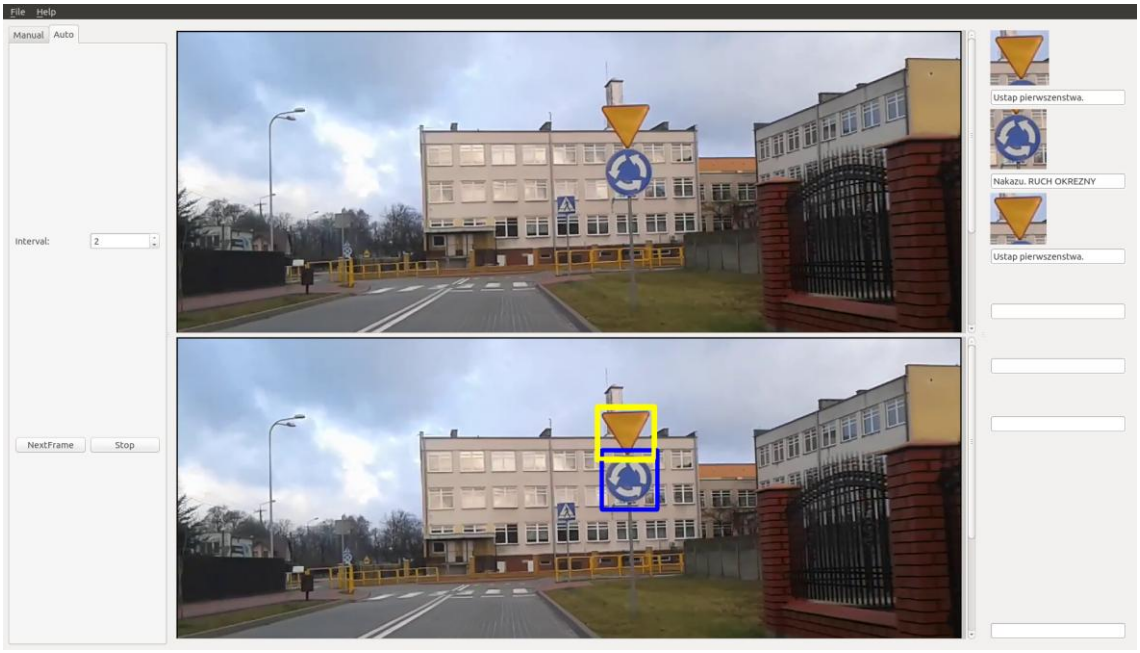
Za pomocą owego okna użytkownik ma możliwość dynamicznego podglądania wartość pikseli obrazu na które wskazuje myszką. Wyświetlane wartości pikseli są podawane w dwóch przestrzeniach barw, RGB i HSV. Ma on również możliwość zapisania danych obiektu do pliku CSV. Użytkownik ma możliwość wyboru czy dany

plik powinien być zapisany przez użytkownika w lokalizacji domyślnej czy w wybranej przez niego samego. Do zapisywania w lokalizacji domyślnej wystarczy wybór jednej z grup znaków do której mają zostać przyporządkowane dane liczbowe. Jeżeli użytkownik chce wybrać własną lokalizację dla pliku ze znakami wystarczy, że zaznaczy pole mówiące o zapisie do niestandardowej ścieżki, następnie wybierze ją klikając na przycisk *Choose Path* i zatwierdzi klikając na przycisk *OK*.

Kolejną rzeczą jaką użytkownik może zobaczyć po lewej stronie okna jest drzewo z danymi obiektu zewnętrznego, którego kontur przedstawiony jest w oknie (dane na drzewie danych znajdują się w gałęzi *External Object*), jak również z danymi liczbowymi wyznaczonymi dla wszystkich jego podobiektów (na drzewie danych gałąź *Internal Objects*).

10.3 Przetwarzanie sekwencji wideo w trybie automatycznym.

Tryb automatyczny jest dla użytkownika o wiele prostszy w obsłudze niż tryb manualny. Widok trybu automatycznego posiada również nieco inną budowę. Zamiast panelu z sześcioma obrazami pokazywany jest panel z dwoma. W jednym z nich jest pokazywana ramka źródłowa a w drugim jej kopia z zaznaczonymi obiektami które zostały przez system rozpoznane.



Rysunek 10.9 Widok głównego okna aplikacji działającej w trybie automatycznym.

Kolejną różnicą jest obecność panelu który wyświetla rozpoznane znaki wraz z ich nazwami. Użytkownik nie ma możliwości manipulowania żadnymi parametrami poza interwałem między ramkami. Jak przedstawiono na rysunku wyżej (Rysunek 10.9)

w trybie automatycznym panel z opcjami segmentacji pozostaje ukryty.

Działalność użytkownika jest bardzo ograniczona. Może on tylko wystartować automatyczne przetwarzanie i ewentualnie zastopować je w dowolnym momencie.