WYDZIAŁ PODSTAWOWYCH PROBLEMÓW TECHNIKI
POLITECHNIKI WROCŁAWSKIEJ

Kontrola i statystyka obiektów

Projekt zespołowy

Autorzy:
Jakub Duda
Mikołaj Dukiel
Piotr Klepczyk
Mateusz Laskowski

Prowadzący:
dr inż. Łukasz Krzywiecki

WROCŁAW 2018

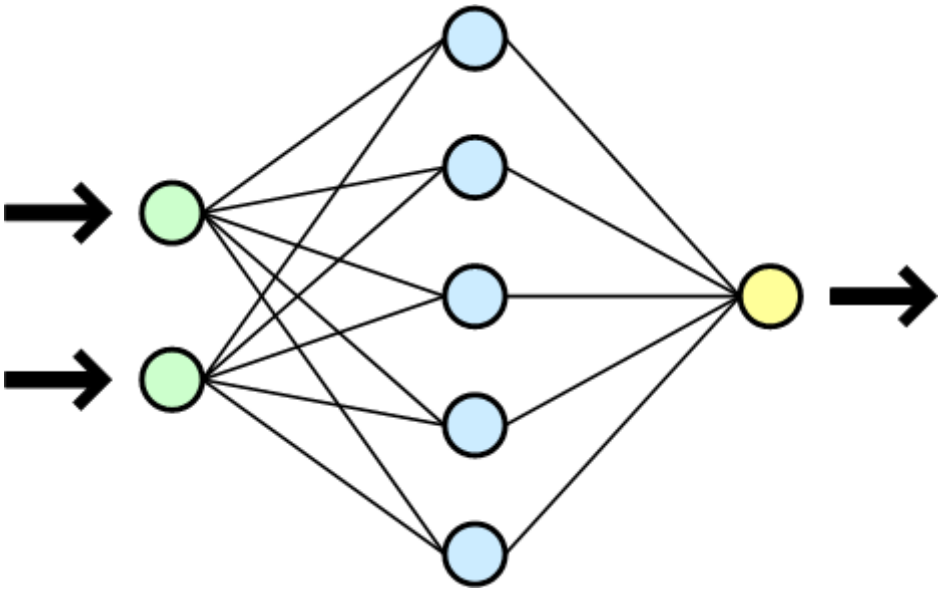
Spis treści

Wstęp

Celem projektu jest utworzenie programu do odczytu wideo, gdzie wideo wejściowe będzie analizowane i przetwarzane na dane statystyczne oraz wideo poglądowe, jako dane wyjściowe. Projekt będzie realizowany w małych krokach. Wpierw do danych wejściowych będziemy udostępniać obrazy, następnie nagrane wideo, a ostatnim oraz najtrudniejszym celem, będzie użytkowanie programu w czasie rzeczywistym. Oczywiście wideo będzie ograniczone jakościowo, jednak samo działanie powinno pozytywnie zaskoczyć. Celem oczywiście będzie również odpowiedni trening sieci neuronowych, które następnie pozwolą na rozpoznawanie cech wielu obiektów. Sieci neuronowe udostępnią nam dane, które my, twórcy będziemy chcieli odpowiednio opisać i wykorzystać do statystyki. Ostatnim celem będzie odtworzenie wideo po nałożonym filtrze, gdzie będzie można rozpoznać przypisane obiekty.

Technologie i zakres projektu. Naszym wspólnym pomysłem jest stworzenie programu, który będzie wykorzystywał jedną z nowszych technologii w świecie informatyki. Chcemy przedstawić oraz użyć w swojej pracy technologię z wykorzystaniem sieci głębokiego uczenia się przy przetwarzaniu dużych ilości danych. Technika ta opiera się na specjalnych sieciach neuronowych. Jednym z zastosowań tej techniki to możliwość rozpoznawania różnych obiektów w naturalnych scenach naszego życia. Wygodnym rozwiązaniem w tej technologii jest to, że nauczanie sieci rozpoznawania obiektów nie polega na wpajaniu mu szczególnych cech obiektów, lecz przedstawiamy mu przykładowe obiekty oraz jego nazwy, a w efekcie otrzymujemy sieć, która jest w stanie rozpoznawać dość szybko wiele różnych obiektów. Ten proces nazywa się treningiem sieci, gdzie wygląda to jak zwykła nauka dziecka. Przedstawia mu się przykłady, sieć je przetwarza i sama wychwytuje ważne cechy obiektu. Trening jest czasochłonny, ale bardzo opłacalny w porównaniu z manualną nauką poszczególnych cech. Technologia głębokiej sieci, czyta cały obraz, jaki jej się podaje do odczytu jako zbiór cech, które później w sieci neuronowej decyduje o istotnych cechach i odnajduje je w nim obiekty. Dzięki czemu można znaleźć wiele obiektów za jedną analizą obrazu. Znając już technologię, na której będziemy opierać cały swój projekt, chciałbym przedstawić jego zakres. Projekt będzie obejmował trzy wyróżniające się segmenty. Pierwszy to obróbka obrazu, a nawet udoskonalenie do możliwości odtwarzania obrazu live. Wideo live lub poprodukcyjne wideo będzie danymi wejściowymi do programu. Problemów w tym segmencie jest naprawdę sporo. Jednym z nich to rozdzielenie wideo na poszczególne klatki. Jednak nie celujemy w wideo najlepszej jakości, ponieważ wcześniej wspomniana technologia radzi sobie naprawdę dobrze w średniej jakości wideo. Następny segment to sieć głębokiego uczenia, która umożliwi nam z danych wejściowych odczytać obiekty, które sieć neuronowa rozpoznaje. Ostatnim segmentem będzie analiza i statystyka otrzymanych wyników z segmentu drugiego. Chcemy również, aby danymi wychodzącymi było wideo, na którym będą zaznaczone rozpoznane obiekty przez sieć neuronową.

Sieć neuronowa to połączenie elementów zwanych sztucznymi neuronami, które tworzą co najmniej trzy warstwy: warstwa wejściowa, warstwa ukryta oraz warstwa wyjściowa. Warto zaznaczyć, że warstw ukrytych może być wiele, w przeciwieństwie do warstw wejściowych i wyjściowych. Neurony sieci przetwarzają informacje dzięki temu, że ich połączeniom nadaje się parametry, zwane wagami, które modyfikuje się podczas działania całej sieci. Modyfikowanie wag w języku informatycznym zwane jest jako „uczeniem się sieci”.



Ryc.0.1 – przykładowy schemat warstw sieci neuronowych
(kolor: zielony – warstwa wejściowa, niebieski – warstwa ukryta, żółty – warstwa wyjściowa)

W procesie uczenia się sieci tkwi istota i rzeczywista wartość sieci neuronowej. W procesie uczenia się nie projektuje się algorytmu kolejnego przetwarzania danych wejściowych, lecz stawia się sieci przykładowe zadania (w naszym przypadku rozpoznawania kształtów/obiektów), a następnie zgodnie z założoną strategią uczenia modyfikuje się połączenia elementów sieci, a dokładniej jej współczynniki wagowe poszczególnych połączeń. Mamy dwie wyróżniające się strategie uczenia sieci neuronowej. Pierwsza to uczenie nadzorowane, zwane „uczeniem z nauczycielem", które polega na porównaniu sygnału wyjściowego ze znanymi prawidłowymi odpowiedziami. Drugą metodą nauki to uczenie bez nadzoru, zwane inaczej „uczeniem bez nauczyciela", które polega na pozwoleniu sieci samej określenia czy dane elementy są ważne, czy pomijalne, a następnie sama uogólnia wyniki i modernizuje wagi. Jest to dobry sposób, aby uogólniać wyniki, dzięki czemu sieć będzie w stanie rozpoznawać dane zbliżone do wzorca. Niestety ta metoda nie nadaje się na sam start nauki sieci neuronowej, ponieważ, taka sieć nie będzie w stanie określić dobrego wzorca przez co będzie popełniać mnóstwo błędów. Najlepszym sposobem nauki jest zmieszanie obu strategii, lecz pierwszą z nich powinna być mimo wszystko strategia „uczenia z nauczycielem". Problemem sieci neuronowych jest to, że nie potrafi wykonać obliczeń, w których wymagane są duże dokładności.

Rozwój projektu. Program jak najbardziej może wyliczać natężenie ruchu na skrzyżowaniach. Dzięki algorytmowi rozpoznawania obiektów będziemy w stanie określać różne obiekty na pliku wideo, a nawet wideo prosto z transmisji kamery. Projekt może być rozwijany na wiele sposobów. Jednym z nich, które my widzimy to dynamiczne zwiększanie i zmniejszanie przepustowości skrzyżowań w zależności od poziomu obiektów oczekujących na zjazd. Mając ten system na wielu skrzyżowaniach można przewidzieć, gdzie fala wielu samochodów następnie się pojawi i umożliwić tak zwaną „zieloną falę", gdzie naprawdę pozwoli to na błyskawiczne usunięcie korków z głównych tras. Kolejnym rozwinięciem to wykrywanie pojazdów MPK, które miałyby priorytet. System odczytuje z kamery autobus MPK, przesyła dane o zwiększeniu priorytetu danego pasa do zaświecenia zielonego światła. Byłby to na pewno bardzo złożone działanie, ale w informatyce nie ma rzeczy niemożliwych. Zapewne byłoby jeszcze wiele rozwiązań, których w tym momencie nie widzimy, ale to pokazuje jak dużą przyszłość ma ten projekt pod względem marketingowym. Warto zaznaczyć, że również można

rozwinąć naszą główną technologię, czyli sieci neuronowe. Nasze sieci można nauczyć rozpoznawać również i w nocy. Niestety cechy, które nauczyły się w dzień, nie będą w stanie dobrze funkcjonować w nocy. Wideo traci na jakości oraz w grę wchodzi mniejsza rozróżnialność obiektów (człowiek na czarno ubrany nie będzie się wiele różnił od cienia człowieka). Będzie można nauczyć naszą sieć neuronową chociaż najważniejszych dla nas obiektów (człowiek, samochód, autobus itd.), co umożliwi działanie nawet w nocy wcześniej wspomnianego przykładowego opisu możliwości wykorzystania programu (kontrolowany ruch w celu zmniejszenia zakorkowania na skrzyżowaniach). Jeżeli już jesteśmy przy rozwijaniu sieci, warto zauważyć, że można również, dzięki rozpoznawaniu szczególnych obiektów takich jak broń biała, niebezpieczne paczki, broń palna itd., być w stanie zwiększyć bezpieczeństwo, bądź zapobiec tragedią. A to wszystko mógłby robić dla nas program.

Środowisko działania: Przy realizacji przyjęto pewne założenia, które muszą zostać spełnione, by program funkcjonował poprawnie. Przyjęto następujące dane:

- Kąt kamery: 94^{o}
- Maksymalna prędkość autobusu: 50 km/h
- Odległość nagrywania: 25m

Powyższe dane dotyczą przypadku skrajnego, dla wartości przekraczających powyższe program nie będzie działał poprawnie.

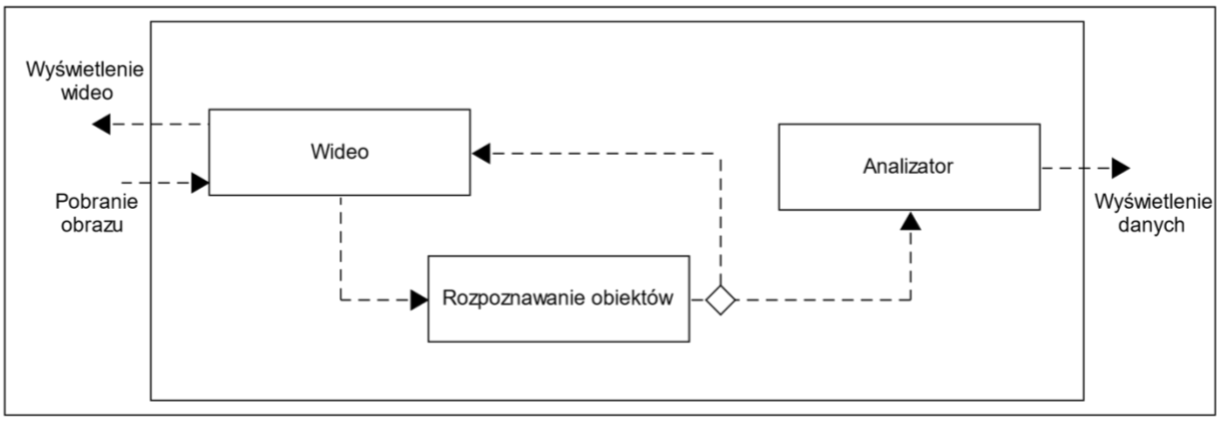
1. Analiza problemu

Opis zagadnień technicznych związanych z projektem:

Uczenie maszynowe (ang. *machine learning*) jest głównym celem samo-uczenia się maszyn. Ma to główne, praktyczne zastosowanie w dziedzinie sztucznej inteligencji do stworzenia automatycznego systemu potrafiącego doskonalić się przy pomocy zgromadzonego doświadczenia oraz nabywania na tej podstawie nowej wiedzy. Można zatem powiedzieć, że sprawiamy, że maszyny poznają świat w sposób empiryczny. Machine learning jest konsekwencją rozwoju idei sztucznej inteligencji i metod jej wdrażania praktycznego. Dotyczy rozwoju oprogramowania stosowanego zwłaszcza w innowacyjnych technologiach i przemyśle. Odpowiednie algorytmy mają pozwolić oprogramowaniu na zautomatyzowanie procesu pozyskiwania i analizy danych do ulepszania, a także rozwoju własnego systemu, sieć neuronowa, sieć nauczania, dostępne biblioteki i algorytmy spełniające funkcjonalność projektu, konwersja i obsługa wideo.

1.1. Komponenty programu

Projekt składa się z trzech głównych komponentów. Nazwy poszczególnych komponentów: `Wideo`, `Rozpoznanie`, `Analizator`. Poniżej przedstawiono graficzny opis zależności pomiędzy komponentami.



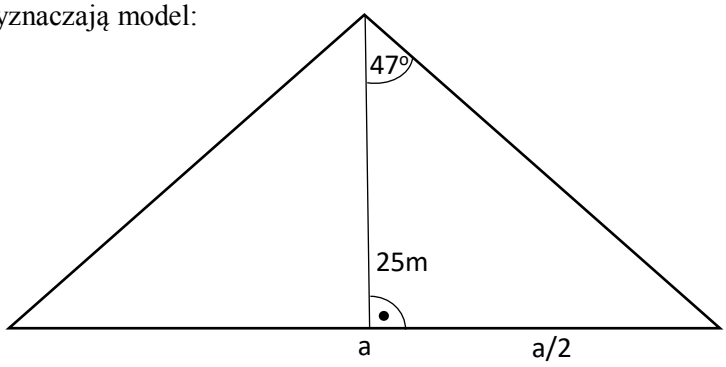
Ryc1.1.1 Diagram komponentów systemu

Komponent `Wideo` - Program będzie przechwytywał obraz z kamery w stałych odstępach czasowych. Przechwycony obraz zostanie przetworzony, poprawnie sformatowany do dalszego przetwarzania wideo, a następnie zostanie rozbity na pojedyncze klatki, na których będzie funkcjonował następny komponent. Po tych operacjach, wideo jest gotowe, aby komponent `Rozpoznanie`, mógł poprawnie funkcjonować. Ogólną funkcjonalnością tego komponentu jest przygotowanie obrazu w taki sposób, by spełniał wymagania kolejnego etapu działania programu. Komponent ten będzie odpowiadał również za wyświetlanie obrazu po wykryciu obiektów w taki sposób, że wykryty obiekt zostanie zaznaczony prostokątem z opisem co za typ obiektu został rozpoznany w tym miejscu. Dane, które nasz komponent otrzyma od komponentu `Rozpoznanie`, zostaną przetworzone, tak aby móc nanieść oczekiwane przez nas elementy na wideo wychodzącym z całego programu. Ten komponent odpowiada za dane wejściowe jak i dane wyjściowe, które użytkownik będzie w stanie zobaczyć jako wyniki pracy programu.

Komponent `Rozpoznawanie` – Komponent ten, wykorzystuje algorytm do identyfikacji obiektów z plików wideo lub z plików graficznych. Identyfikacja obiektów odbywa się na obrazie otrzymanym na wejściu. Po wykonaniu pracy zostanie zwrócone, ile obiektów zostało rozpoznanych, jakiego były one typu i z jakim procentem dopasowania zostały one rozpoznane, te dane zostaną przekazane do `Analizatora`. Będą dane przekazywane również z powrotem do komponentu odpowiedzialnego za obsługę wideo, dane te będą zawierały wielkość i umiejscowienie rozpoznanego obiektu oraz jego typ. Algorytm dzieli obraz na siatkę 13x13 małych komórek. Każda komórka jest odpowiedzialna za 5 obszarów. Obszar opisuje prostokąt, który otacza obiekt. YOLO zwraca wynik prawdopodobieństwa, że obszar coś zawiera. Wynik nie mówi, jaki obiekt się znajduje tylko czy może w nim coś być. Dla każdego obszaru komórka również przewiduje klasę. Działa to jak klasyfikator (daje rozkład prawdopodobieństwa na wszystkich możliwych klasach. Liczba klas zależy od wersji YOLO. Yolov3 (najnowsza wersja) ma ich najwięcej. My jednak używamy yolov3-tiny, aby przyśpieszyć proces rozpoznawania. Prawdopodobieństwo przewidzenia przez obszar i przewidywania klas są połączone w jeden końcowy wynik, który zwraca nam to, czy obszar zawiera jakiś typ obiektu. Na przykład żółty kontener zwraca nam wynik z prawdopodobieństwem 85%, że znajduje się w nim "dog". Ponieważ istnieje 13 x 13 = 169 komórek siatki i każda komórka przewiduje 5 kontenerów, mamy w sumie 845 ramek ograniczających. Wiele z tych kontenerów będzie miała niskie wyniki prawdopodobieństwa, więc zostawiamy tylko te, gdzie prawdopodobieństw jest większe od 30% (co można zmienić dodając flagę -thresh <val> podczas uruchamiania programu). W przykładzie z 845 wyników otrzymaliśmy tylko 3, ponieważ dały najlepsze wyniki. Należy pamiętać, że chociaż było 845 osobnych prognoz, wszystkie zostały wykonane w tym samym czasie - sieć neutronowa została uruchomiona tylko raz. I dlatego YOLO jest taki szybki.

Komponent `Analizator` - Komponent ma rozpoznać, czy na sąsiednich klatkach widzimy ten sam obiekt. Wykorzystuje on do tego dane, które otrzyma z poprzedniego komponentu - `Rozpoznanie`. Jeżeli na dwóch sąsiednich klatkach widzimy ten sam pojazd, czyli na następnej klatce w niewielkim od siebie odstępie, w stosunku do poprzedniej klatki, jest to określane na podstawie delty którą wyznaczono w następujący sposób:

Na podstawie tych danych skrajnych przedstawionych wcześniej, oraz danych które zostały wyznaczone przez algorytm wykrywający obiekty, a są to rozdzielczość obrazu wynosząca 280 pikseli oraz liczba klatek na sekundę, tu wynosi ona 2, pozwoliły wyliczyć deltę określającą maksymalne przesunięcie w pikselach dla pojazdu poruszającego się w kierunku prostopadłym do kamery z maksymalną przyjętą prędkością. Dane wyznaczają model:



Ryc1.1.2 Uproszczony schemat zakresu kamery

Do obliczenia wartości s zastosowano równanie: $tg(47^o) = \frac{a/2}{25}$.

Po jego obliczeniu utrzymano: $a = 53.62m$.

Następnie zostaje wyliczony dystans jaki pokona autobus z maksymalną prędkością w odstępie jednej klatki, czyli ½ s. Do tego celu wykonano następujące operacje:

$$50\frac{km}{h} = 13.9\frac{m}{s}$$

$$v = \frac{s}{t} \Rightarrow s = vt$$

$$s = 13.9 * \frac{1}{2} = 6.95m$$

Następnie została wyliczona zależność pomiędzy pokonanym dystansem, a szerokością kadru:

$$\frac{53.62}{6.95} = 7.72$$

Następnie tę wartość wykorzystano do obliczenia ilości pikseli przebytych w odstępie jednej klatki (d):

$$\frac{280}{d} = 7.72 \Rightarrow d = 36$$

Na podstawie tych obliczeń dla zadanych danych przyjęto deltę, czyli maksymalne przesunięcie autobusu w odstępie jednej klatki jako 36 pikseli.

Gdy mamy też obiekt spełniający dokładnie te same cechy, np. czerwony sportowy samochód, to mamy bardzo duże prawdopodobieństwo, że na obu klatkach mamy do czynienia z dokładnie tym samym pojazdem. Gdy już stwierdzimy, że na kolejnych dwóch klatkach jest ten sam obiekt, to liczmy jego prędkość, czyli o ile się on przesunął w czasie między dwoma klatkami, który w najgorszym wypadku wynosi 0,5 sekundy. Wykorzystamy do tego współrzędne prostokąta, który zaznacza rozpoznany z poprzedniego komponentu obiekt. Jego przesunięcie, odpowiednio przeliczone w skali, będzie wynikiem tej funkcjonalności. Komponent również zliczy natężenie ruchu, czyli sprawdzi, jak wiele pojazdów znajduje się w tym samym momencie na drodze oraz przepustowość, czyli jak dużo opuściło skrzyżowanie w ciągu jednego cyklu. Weźmie on również pod uwagę typ pojazdów, gdyż przykładowo jedna ciężarówka zajmuje zdecydowanie więcej miejsca od kilku motocykli. Na podstawie danych oraz ustawień własnych, określi on również, jakie działanie dla sygnalizacji świetlnej będzie najkorzystniejsze, czyli proporcjonalnie do natężenia danej strefy ruchu będzie dobierał długość trwania świecenie światła w cyklu świetlnym dla tych stref. Odpowiada on również za generowanie wniosków na podstawie wcześniej opisanych i wyuczonych danych. Do ich sformułowania będą potrzebne takie informacje, jak długość cyklu świateł, liczba pasów w danym kierunku i tym podobne. Ostatnią funkcjonalnością tego komponentu będzie prowadzenie statystyk na podstawie gromadzonych danych, które przyczynią się do rozwoju algorytmu i jeszcze większemu usprawnieniu ruchu drogowego. Sprawdzi on w jakim okresie natężenie jest największe, a w jakim najmniejsze. Porówna jakie ustawienie dało największą przepustowość, a także sprawdzi, który z samochodów najdłużej czasu spędził na skrzyżowaniu. To wszystko zostanie wypisane do pliku.

1.2 Scenariusz

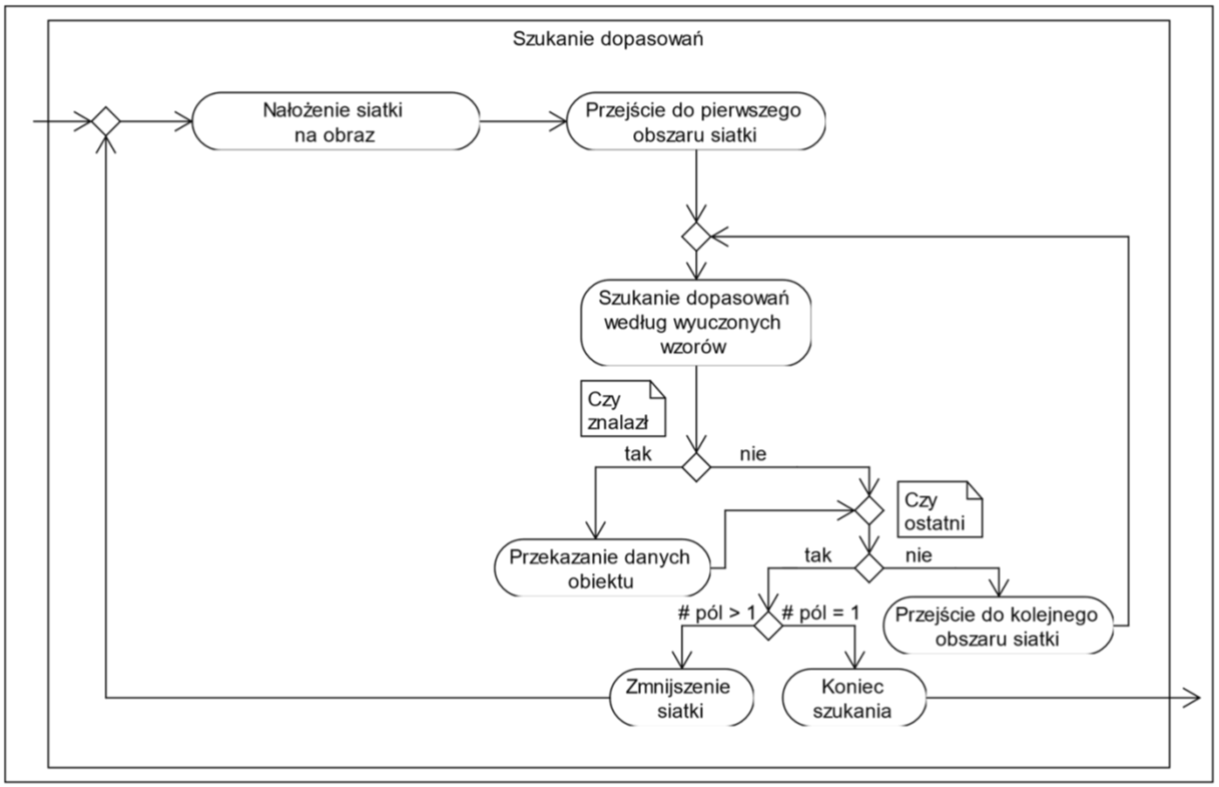
Użytkownik programu jest zobowiązany uruchomić program terminalowo, podając ścieżkę do pliku wideo lub nie podając argumentu co powoduje działanie w czasie rzeczywistym. Jeżeli użytkownik chce uruchomić program w czasie rzeczywistym to obowiązkowym narzędziem jest oczywiście podłączona kamera do komputera, na którym jest uruchamiany program oraz kamera musi być wykrywalna dla komputera, co może zmusić użytkownika do zainstalowania odpowiednich sterowników, aby poprawnie działała kamera. Warto zaznaczyć, że plik wideo podany jako argument wejściowy może być podany w dowolnym formacie wideo. Po wpisaniu komendy, program pobiera wideo i zmienia jego rozszerzenie do ujednolicenia działania, więc jest to ułatwienie do użytkownika, ponieważ może wprowadzić plik wideo w różnych. Pierwszym krokiem programu jest konwersja strumienia wideo do formatu AVI, gdzie następnie ze strumienia wideo co 30 klatek są wyciągane dwie klatki. Wydobyte klatki są formatowane na obrazki w formacie JPG, gdzie są przekierowane jako argumenty wejściowe do sieci neuronowej, czyli naszego algorytmu rozpoznawania obiektów. Sieć neuronowa statystycznie

rozwiązuje problem, czyli szuka występujących i znanych mu obiektów. Jeżeli algorytm nie będzie wstanie chociaż w 30% oszacować występujący obiekt, to znaczy, że jest słabo przedstawiony lub algorytm, nie wykrył w nim znaczących atrybutów. Jednakże po odpowiednim rozpoznaniu dane o obiekcie i jego występowaniu w jakiej klatce oraz na jakich pikselach się pojawił są danymi wyjściowymi z programu rozpoznawania obiektów. Uzyskane dane wyjściowe są potrzebne do dwóch następnym elementów w czasie działania programu. Jednym z nich to przekazanie dokładnych pikseli występowania obiektu co umożliwia określenia środka obiektu i nałożenie na niego ramki prostokątnej widocznej na wyjściu programu, które użytkownik widzi jako wideo z nałożonymi ramkami z rozpoznanymi wszystkimi obiektami. Zaś drugim elementem to przekazanie wiadomości o typie obiektu, długości życia, informacji na jakiej klatce został rozpoznany oraz wskazanie dokładnych pikseli. Analizator, czyli element drugi, tego rozwidlenia w programie ma brać pod uwagę tylko obiekty wcześniej określone – w naszym wypadku, zdefiniowane obiekty do rozpoznawania to autobusy miejskie – gdzie następnie brana jest pod uwagę kierunek poruszania się obiektu, rozpoznanie paru różnych obiektów tego samego typu oraz określenie czasu życia pojedynczego obiektu. Program kończy się w zależności od danych wejściowych. Jeżeli wejściem był plik wideo to działanie kończy się po skończeniu wideo, jeżeli zaś została wybrana druga opcja to program działa, dopóki program nie zostanie zatrzymany lub pojawi się jakiś błąd przesłania wideo ze źródła strumienia, czyli zostanie rozłączona kamerka lub jakieś inne niepoprawne jej działania, które nie będą udostępniać strumienia wideo.

1.3. Diagram czynności

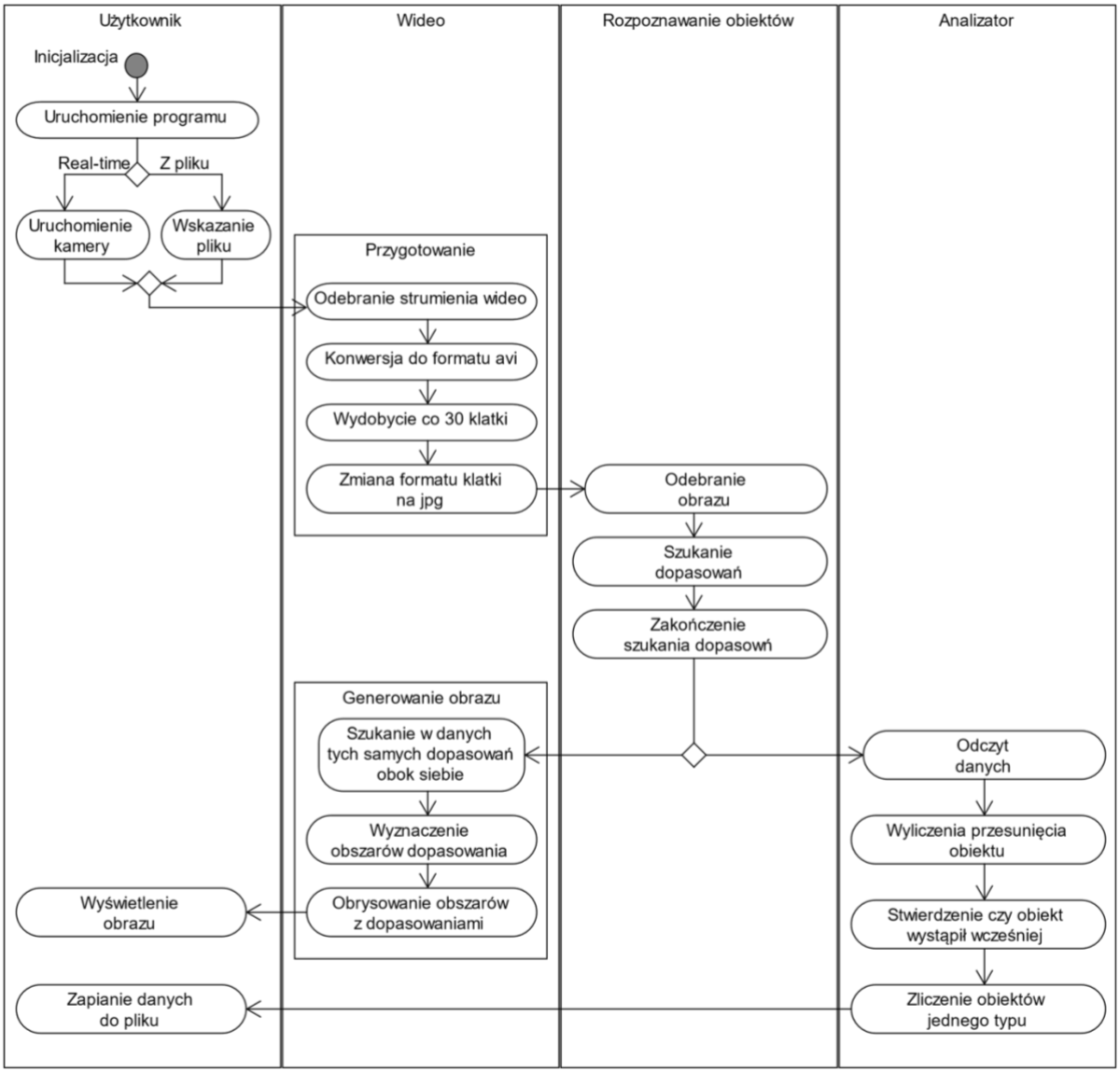
Poniżej przedstawiono diagramy obejmujące funkcjonalność programu z podziałem na komponenty oraz z operacjami przeprowadzanymi w obrębie komponentu. Na pierwszym diagramie opisano dokładny przebieg jednej z podstawowych funkcjonalności programu.

Dokładne działanie `Szukania dopasowań` z komponentu `Rozpoznawanie obiektów` zostało zaprezentowane na osobnym diagramie (Ryc1.3.2).



Ryc1.3.2 Diagram szukania dopasowań

Poniższy diagram obrazuje wszystkie funkcjonalności jakie składają się na poszczególne komponenty, a także na cały program.



Ryc1.3.1 Diagram czynności

1.4. Heurystyka rozwiązywania problemów

W Analizatorze dochodzi do rozwiązywania wielu problemów logicznych na temat rozpoznanych obiektów przez algorytm Rozpoznawania obiektów. Podczas testów dochodziło do niechcianych błędów, które zaburzały prawdziwą statystykę wyników pracy. Oprogramowaliśmy logikę rozwiązywania trzech błędów, które najczęściej występowały. Pierwszym błędem było chwilowe zgubienie obiektu przez algorytm YOLO. Musieliśmy wziąć pod uwagę, że obiekt może być chwilowo zasłonięty lub po prostu nie rozpoznany pomimo wcześniejszego rozpoznania. Tutaj z pomocą przychodzi nam czas życia obiektu. Uwzględniliśmy, że dany obiekt może „zniknąć” z oka algorytmu

na 80 jednostek życia co daje nieco ponad 2 sekundy nagrania. Dzięki temu obiekt, który nie został rozpoznany przez krótką chwilę nie jest od razu przypisywany jako nowy obiekt, lecz jako poprzedni. Kolejnym ulepszeniem unikania błędnych wyników to uwzględnienie, że algorytm mógł rozpoznać nie cały obiekt, lecz jego część, przez co prowadzi do przesunięcia środka obiektu pomimo innego ruchu. W naszym programie wykrywamy, obiekty poruszające się w lewą stronę od obiektywu kamery lub w przeciwną (prawą). Uśredniamy wyniki oraz różnice środków tych samych obiektów pomiędzy każdymi klatkami, co prowadzi do uwzględnienia wykonanie całego ruchu danego obiektu, a nie chwilowego. Dla ułatwienia założyliśmy, że obiekt nie może zawrócić. Trzecią ważną logiką do niwelowania błędów jest sprawdzanie, czy obiekty na siebie nie na chodzą. Mieliśmy przypadki, że dany obiekt rozstał rozpoznany, lecz w tym obiekcie rozpoznał jeszcze jeden, ten sam obiekt, co doprowadzało do generowania nowego obiektu w Analizatorze oraz obserwowanie jego ruchu na kamerze. Zniwelowaliśmy ten błąd poprzez sprawdzanie, czy pola dwóch takich samych obiektów nie nachodzą na siebie w 90%. Jeżeli taki przypadek występuje, to jest tworzony tylko jeden obiekt, ten który ma większe pole. Trzeba pamiętać, że jeżeli jakiś obiekt zostanie zasłonięty przez ten sam typ obiektu, i jego środek jest względnie blisko poprzedniego to zostanie zastąpiony. Algorytm nie będzie wiedział, że jest to inny, lecz w tym samych miejscu i takich samych parametrach. Można rozszerzyć tę heurystykę również o głębię obiektów, które mogłoby rozwiązać tego typu problemy. Przykładowo, wprowadzenia jeszcze jednej zmiennej do obiektu, która by miała sprawdzać odległość od obiektu, a kamery. I nagłe zmniejszenie tej odległości oznaczałoby, że obiekt albo się tak szybko poruszył w kierunku kamery, co dla samochodów jest nierealne, albo przed nim pojawił się jakiś inny obiekt. Jeżeli byłby to tego samego typu obiekt to wtedy wiedzielibyśmy, że trzeba wygenerować kolejny obiekt, a nie jest to ten poprzedni widziany przez kamerę.

1.5. Szybkość działania

Chcielibyśmy wziąć pod uwagę, że algorytm Rozpoznawania obiektów, zwany YOLO, bardzo mocno obciąża kartę graficzną. Im lepsza karta graficzna tym szybciej algorytm jest w stanie przejść przez węzły sieci neuronowej. Warto pamiętać, że są to operacje na wideo, więc tam również jest obciążana karta graficzna. Nasz algorytm YOLO działa tylko na kartach graficznych GeForce. Tak oprogramowana jest biblioteka, która została użyta w naszym programie. Udało nam się uzyskać działanie algorytmu Rozpoznawania obiektów na średnio dwóch klatkach na sekundę.

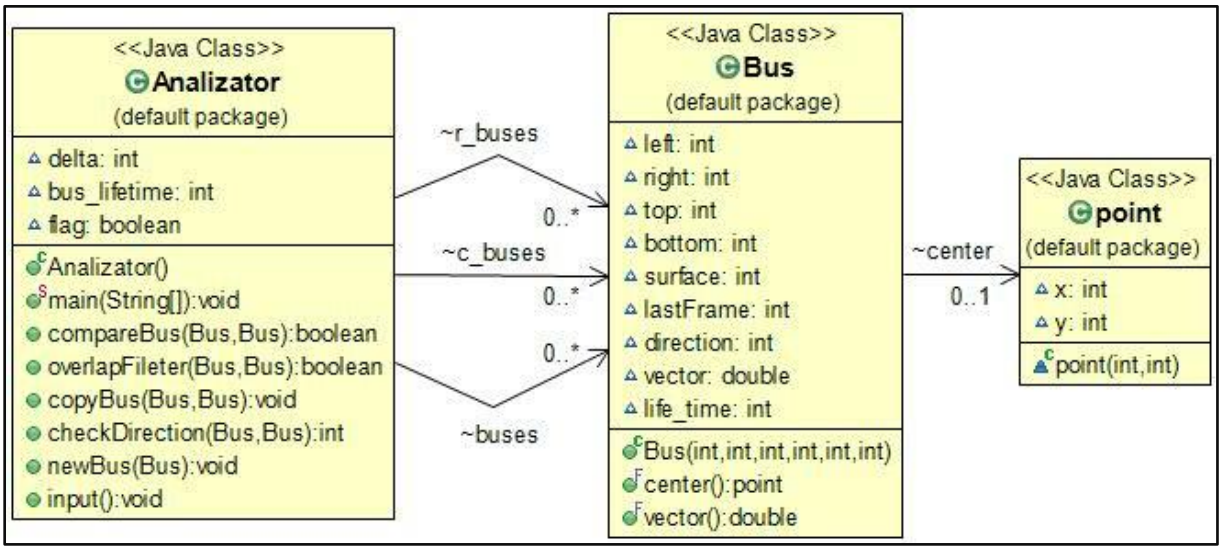
2. Dokumentacja kodu

2.1. Biblioteki

W naszym używamy następujących bibliotek:

1) "**darknet.h**" - Darknet jest open-source'ową strukturą sieci neuronowej napisaną w C i CUDA, która pozwala na implementację algorytmu YOLO w naszym projekcie. Biblioteka o tej samej nazwie daje nam możliwości uruchomienia i obsługi wymienionego wyżej algorytmu.
2) "**list.h**" - dzięki, której mamy kontener, będący listą dwukierunkową. Używamy jej dla tego, że lepiej w naszym przypadku sprawdza się od wektorów, gdyż często wprowadzamy zmiany w kontenerze. Jest to spowodowane tym, że dodajemy, co chwilę nowe elementy, więc listy będą szybszym rozwiązaniem. Z drugiej strony w większości przypadków przeprowadzamy operacje po kolei, zatem nie potrzebujemy dostępu do losowego elementu, co jest dosyć kosztowną operacją w przypadku listy.
3) "**tree.h**" - wykorzystywane są praktycznie we wszystkich dziedzinach informatyki, a zwłaszcza w uczeniu maszynowym i sieciach neuronowych. Zawiera ona struktury danych nazywane drzewami. Są to spójne i acykliczne grafy skierowane, których wierzchołki reprezentują konkretne dane. Dzięki nim algorytm, który używamy mógł uczyć się, aby coraz lepiej spełniać swoje zadanie.
4) "**image.h**" - jest to biblioteka, dzięki której jesteśmy w stanie zamieniać klatki filmu, na którym operujemy, w pliki graficzne, w formacie JPG, na których tworzymy macierze potrzebne później do podziału.
5) "**matrix.h**" - ta biblioteka zdecydowanie pomaga w operowaniu na macierzach, które są nam niezbędne, gdyż algorytm dzieli każdą klatę na kwadraty, gdzie każdy z nich jest właśnie elementem macierzy. W nich natomiast wyszukiwane są atrybuty naszej sieci neuronowej.

2.2. Diagram klas Analizatora



Ryc.2.2.1 Diagram klas Analizatora