Kamil Bolek
mgr inż.

Wyznaczanie dynamicznej mapy 2D na podstawie wielomodalnych danych rejestrowanych przez urządzenia mobilne

POLSKO-JAPOŃSKA
AKADEMIA TECHNIK
KOMPUTEROWYCH

Rozprawa doktorska
29 września 2021

Promotor:
dr hab. inż. Adam Świtoński prof. Pol. Śl.

Streszczenie

Praca poświęcona jest tematyce tworzenia mapy 2d, która przewyższy poziomem szczegółowości i aktualności wszystkie dostępne w danej chwili mapy. Metody dostarczania coraz dokładniejszych informacji o naszym otoczeniu nieustannie się zmieniają starając się nadążyć za szybko postępującymi zmianami otoczenia. Obecne moduły GPS w kilka sekund lokalizują użytkowników, a specjalne oprogramowanie dostarcza informacji o otoczeniu w którym się znajdują. Pozyskane w ten sposób dane dotyczą jednak obiektów, których czasowa dynamika zmian jest niewielka (takich jak lokalizacja budynku czy nazwa ulicy) co jest wystarczające do nawigacji, jednak nie pozwala to na określenie co dokładnie znajduje się na danym obszarze w danej chwili.

Niniejsza praca prezentuje oryginalne rozwiązanie służące mapowaniu obiektów będących zarówno elementami statycznymi, jak również dynamicznymi (tj. zmieniającymi swoje położenie np. rowery). W pracy zastosowano sieci neuronowe przetwarzające obraz na urządzeniach mobilnych i kamerach miejskich oraz zaimplementowano algorytm, który na podstawie danych pochodzących z wielu źródeł doprecyzowuje lokalizację wykrytych obiektów i nanosi ją na mapę.

Algorytm przetwarza, wielomodalne dane przestrzenne i czasowe, które są przesyłane przez urządzenia, w celu aktualizacji mapy dostępnej na serwerze. Dzięki takiemu podejściu, znajdują się na niej informacje o stanie otoczenia z wybranego okresu, a w szczególności chwile (przedziały czasowe) w których obiekt się pojawił lub zniknął z danego obszaru. Utworzone w ten sposób rozwiązanie realizuje funkcję wizualizacji zmian jakie zachodziły na wybranym terenie z możliwością dynamicznego odświeżania tych zmian w czasie rzeczywistym.

Opracowane rozwiązanie ma szerokie zastosowania. System może służyć do wytyczania nowych ścieżek rowerowych w miejscach, gdzie przemieszcza się najwięcej rowerzystów, wybrania lokalizacji dla przeprowadzenia kampanii reklamowej określonego produktu w obszarach gdzie go nie znaleziono lub też stworzenia gry terenowej dla turystów jako promocji miasta.

Stworzony system aktualnie przeszedł pomyślnie testy prawidłowości działania w środowisku laboratoryjnym i jest wdrażany w środowisku produkcyjnym. Pokazane w pracy badania eksperymentalne w warunkach zbliżonych do rzeczywistych pozwalają stwierdzić, że zaproponowane podejście jest skuteczne, a sam system działa według pierwotnych założeń. Jego pełne uruchomienie jest przewidziane na trzeci kwartał 2022 roku.

Słowa kluczowe: sieci neuronowe na urządzeniach mobilnych - TensorFlow Lite, mapowanie obiektów - OrientDB, estymacja lokalizacji, sensory IMU.

Abstract

The thesis is devoted to the project of creating a 2D map which would be more detailed than all the solutions available to date thanks to innovative methods of supplying information about a constantly changing surrounding. Existing GPS modules are able to find the location of users within a few seconds, combining it with information about their surroundings provided by a dedicated software. Visualized data usually concern objects which are relatively static (such as location of a building or the name of a street).

The thesis presents an innovative solution that allows for mapping of both static and dynamic objects (changing their location quickly, such as bicycles, for instance). The project is based on the neural network which processes images on mobile devices and the algorithm which, drawing on data derived from many different sources, makes the location of object more precise on the map.

The algorithm constantly processes temporal multi-modal spatial data that are sent by the users and used for updating the map. As a result, on the map there is information about the state of surroundings at a given point in time, especially the moment when a particular object appeared in the frame of the camera and disappeared. Employing this method, not only does the presented solution allow for the visualisation of changes that have taken place in a particular area, but also makes it possible to refresh the map dynamically in real time.

As the results of experiments presented in the thesis indicate, the designed system has wide applicability. For example, it can be used for establishing new cycle paths in the area where there are a lot of cyclists, choosing a place for an advertising campaign where a given product is barely present or creating a urban game for tourists that would promote a city.

The developed system has successfully passed regularity tests in the lab and is being implemented in the production environment. On the grounds of experiments carried out in the real-life conditions we may conclude that the system is effective and operates according to initial asssumptions. The application of the system is planned to be completed by third quarter of 2022.

Keywords: neural networks on mobile devices - TensorFlow Lite, mapping objects in space - OrientDB, distance estimation, IMU sensors.

Spis treści

Spis tabel

Spis rysunków

1

Wstęp

Ewolucja map na przestrzeni wieków, zmierzająca do uzyskania możliwie jak najwierniejszego i najbardziej aktualnego odzwierciedlenia terenu, doprowadziła do rozwoju dziedziny nauki poświęconej wyłącznie temu zagadnieniu. Kartografia jest nauką o metodach sporządzania oraz wykorzystywania map. Mapa jest to obraz powierzchni Ziemi wykonany na płaszczyźnie, w pomniejszeniu, z uwzględnieniem jej krzywizny. Współczesna mapa, dzięki wykorzystaniu sztucznych satelitów oferuje dokładność rzędu kilku centymetrów, która zmienia się nieznacznie w zależności od szerokości geograficznej w której wykonywane są pomiary. Wskazana różnica wynika z odległości jaka występuje pomiędzy południkami na globie dla różnych szerokości geograficznych. Mapy mają szerokie zastosowanie. Poza oczywistymi, takimi jak nawigacja czy wyznaczanie granic, są również wykorzystywane do wizualizacji danych i zjawisk występujących na danym obszarze.

Poza dokładnością map, równie istotna jest ich szczegółowość i aktualność. Spośród dostępnych map największą popularnością cieszą się mapy Google i Open Street Map (jako darmowa alternatywa do użytku komercyjnego). Oba wymienione rozwiązania cechują się dużą szczegółowością i aktualnością dzięki wsparciu ogromnej społeczności, która nieustannie zgłasza błędy oraz aktualizuje wizualizowane na mapie informacje. Wykorzystanie użytkowników jako źródła danych umożliwia ciągłą aktualizację mapy, realizowaną nieustannie na całym świecie. Napływające od użytkowników dane są etykietowane znacznikiem czasowym co umożliwia analizę zmian jakie zachodziły w wybranym regionie.

Przez ostatnie lata rozszerzono funkcjonalność map o możliwość podglądu widoku danego miejsca. Poza zdjęciami samych użytkowników, obecne rozwią-

zania oferują wizualizację widoku zarejestrowanego przez specjalnie przystosowane w tym celu samochody z osadzonymi na dachu kamerami 360°. Dzięki temu możliwe stało się pokazanie rzeczywistego widoku pochodzącego z dowolnego miejsca w którym znajdowali się użytkownicy lub wspomniane samochody. Wszystkie te rozwiązania uszczegóławiają dostępne na mapie informacje dodając aspekt temporalny. Brak w nich jednak informacji o elementach otoczenia cechujących się dynamiką rzędu kilku godzin. Nie istnieją rozwiązania, które dostarczałyby informacji o elementach, które w krótkim czasie zmieniają swoje położenie takie jak rowery czy samochody. Aspekt rejestracji na mapie miejsc w których się pojawiają i czasu dotychczas nie został uwzględniony z uwagi na dużą dynamikę, która takie elementy cechuje.

1.1 Motywacja oraz cel pracy

Nieustanne dążenie do kolekcjonowania oraz udostępniania szerokiemu gronu odbiorców coraz dokładniejszych informacji o nas i naszym otoczeniu przejawia się w wielu aspektach codziennego życia. W skali mikro, są to opaski na ręce monitorujące nasze zdrowie i parametry życiowe pomagając pogłębić w ten sposób wiedzę o nas samych. W skali makro przykładem są zdjęcia satelitarne oraz prowadzone przez lata pomiary temperatur, które pozwalają poznać i określić skutki efektu cieplarnianego. Oferowane obecnie mapy nie posiadają możliwości mapowania obiektów zmieniających swoje położenie (np. rower czy samochód). Oferują jedynie komplet informacji o elementach statycznych takich jak budynki, przystanki autobusowe czy stacje benzynowe uzupełniając dane o lokalizacji o informacje o godzinach otwarcia lub rodzaju oferowanych w danym miejscu usług.

Aspekt uchwycenia dynamiki obszaru, w szczególności miasta, zapoczątkował szereg działań mających na celu stworzenie systemu, który pozwoli na mapowanie z dużą dokładnością wszystkich obiektów (statycznych oraz dynamicznych) tworząc w ten sposób mapę świata realnego którego stan jest aktualizowany w czasie rzeczywistym. Pomimo ogólnej dostępności wiedzy o wszyst-

kich składowych docelowego systemu, nie znaleziono informacji, by powstała w wiodących ośrodkach naukowych czy firmach na świecie technologia tworzenia mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych. Zauważenie tej luki stało się impulsem który zapoczątkował realizację rozprawy doktorskiej. System stworzony w ramach rozprawy dostarczy całkowicie nowe narzędzie do analizy wielomodalnych danych przestrzenno czasowych i może mieć szerokie zastosowania praktyczne. Docelowe rozwiązanie będzie oferowało API do łatwej integracji z zewnętrznymi systemami oraz graficzny interfejs użytkownika (GUI) pozwalające na wizualizację przetwarzanych przez system danych.

Głównym dostawcą informacji o zmianach jakie zachodzą w otoczeniu mają być użytkownicy, którzy grając w gry terenowe będą przemieszczać się po mieście z telefonem, który nieustannie będzie dostarczał danych, umożliwiających, przy wykorzystaniu stworzonego algorytmu, doprecyzowania lokalizacji znalezionych w widoku kamery obiektów. Dokładność mapy i zebranych informacji z danego terenu będzie silnie uzależnione od liczby użytkowników, którzy się po nim przemieszczają. W celu popularyzacji systemu została dodatkowo opracowana innowacyjna platforma do tworzenia gier terenowych w oparciu o silnik gier AR (rozszerzonej rzeczywistości) wizualizująca mapę świata realnego w czasie rzeczywistym (w tym obiektów nie będących elementami krajobrazu), wraz z aplikacją umożliwiającą przesyłanie danych przez użytkowników. Stanowi ona główny rezultat podejmowanych prac, a działające wewnątrz algorytmy i moduły stanowiące jej elementy składowe pozwoliły, po dostosowaniu ich działania, na ich wykorzystanie również do innych zastosowań takich jak moduł estymacji długości kolejek na skrzyżowaniach w mieście Wrocław.

Prowadzone badania były realizowane w ramach dofinansowania uzyskanego ze środków NCBiR, na realizację projektu który zajął pierwsze miejsce spośród ponad 500 projektów zgłoszonych do konkursu GAMEINN w ramach Programu Operacyjnego Inteligentny Rozwój 2014- 2020 działanie 1.2, POIR.01.02.00-00-0045/19) Realizacja projektu doprowadziła do stworzenia systemu budującego mapę 2D na podstawie wielomodalnych danych przestrzennych i czasowych.

1.2 Teza i rezultat pracy

Zasadniczą tezę pracy można sformułować następująco: **Możliwe jest stworzenie mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych, przesyłanych z wielu źródeł, która dostarczy informacji o położeniu obiektów statycznych i dynamicznych (tj. zmieniających swoje położenie) znajdujących się na wybranym obszarze.** Postawiona teza, ze względu na swoją obszerność, wymagała w pierwszej kolejności potwierdzenia kilku pomocniczych tez obejmujących mniejszy zakres tematyczny tj:

- Fuzja danych o obiektach wykrywanych na obrazie, parametrach wewnętrznych kamery, informacji pochodzących z IMU oraz sygnału GPS pozwala na lokalizację obiektów 3D z dokładnością do 10m względem wartości referencyjnej.
- Integracja wielu pomiarów położenia obiektów 3D poprawia dokładność ich lokalizacji.
- Głębokie sieci neuronowe działające na urządzeniach mobilnych opartych o architektury ARMv8-A i nowsze (ARMv8.3-A, ARMv8.4-A) pozwalają na efektywną detekcję obiektów na obrazie z prędkością o wartości oczekiwanej nie mniejszej niż 3 klatki na sekundę.
- Zastąpienie systemu zarządzania bazą danych PostgreSQL w systemie Open Street Map na grafowy system zarządzania bazą danych w postaci OrientDB znacznie przyśpiesza proces wyszukiwania informacji niezbędnych do renderingu mapy.

W celu weryfikacji przedstawionych tez zostały opracowane i zaimplementowane algorytmy realizujące tworzenie mapy o wskazanych parametrach. Algorytmy stanowią element składowy zbudowanego systemu składającego się z oprogramowania działającego na serwerze do przetwarzania nadsyłanych danych, aplikacji mobilnej dla smartfonów z Android/iOS oraz oprogramowania dedykowanego dla mikrokontrolerów dostarczających dane z kamer umieszczonych w mieście.

Założeniem systemu jest zapewnienie dużej dostępności dla szerokiego grona użytkowników. W tym celu teza dotycząca głębokich sieci neuronowych określa wsparcie nie tylko najnowszych telefonów posiadających dedykowane wsparcie sprzętowe, lecz obejmuje także urządzenia starsze. W rozprawie skupiono się na urządzeniach z kilku kategorii: najnowsze smartfony, urządzenia ze średniej półki (kosztujące w okolicach 1000zł) oraz starsze urządzenia, które są oparte na architekturze ARMv8-A, która zaczęła być szeroko stosowana w urządzeniach mobilnych w 2015 roku (m.in. w Samsungu Galaxy S6). Wskazany zakres powinien objąć dostępnością zbudowanej aplikacji dla więcej niż 80% urządzeń na rynku. Z aktualnych raportów Apple i Google wynika, że aplikacja zasilająca system danymi jest kompatybilna z 84.9% urządzeń z systemem Android i 81% z iOS (stan na styczeń 2021). W przypadku mikrokontrolerów, zbudowane oprogramowanie jest dedykowane wyłącznie dla wąskiej grupy urządzeń tj. mikrokontrolerów STM32 w wersji Discovery F7 oraz Nvidia Jetson Nano. Obydwa ze wskazanych urządzeń są jednak szeroko wykorzystywane w ITS (Inteligentnym Systemie Transportowym) Wrocław gdzie prowadzono badania i uruchomiono prototyp systemu.

Wewnątrz systemu utworzono również moduł bazy danych noSQL przechowującej informacje niezbędne do renderingu map działający szybciej niż klasyczne Open Street Map, które bazuje na PostgreSQL. Zoptymalizowana struktura, bazująca na klasteryzacji zapisanych wewnątrz obiektów oraz sposób jej przeszukiwania doprowadził do zwiększenia szybkości serwera map co zostało potwierdzone na maszynie wirtualnej o niewielkich zasobach.

W celu potwierdzenia tezy dotyczącej scalania oraz interpretacji wielu pomiarów utworzono autorski algorytm, który doprecyzowuje lokalizację obiektów widzianych w obiektywie urządzenia brzegowego (kamery na mieście lub smartfonie). Algorytm, na podstawie wielu źródeł danych (rezultacie detekcji, parametrów wewnętrznych kamery, czujnika IMU i modułu GPS urządzenia), estymuje lokalizację danego obiektu, a kolejne iteracje działania przybliżają do coraz dokładniejszej estymacji względem wartości referencyjnej.

Istotnym aspektem na którym skupione będą opisywane prace badawcze jest

stworzenie prototypu systemu, który poza potwierdzeniem możliwości tworzenia dynamicznej mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych, będzie wydajny (umożliwiając obsługę powyżej tysiąca zapytań na sekundę), skalowalny oraz dostępny dla szerokiej gamy urządzeń końcowych.

1.3 Układ rozprawy

We wstępie przedstawiono wprowadzenie do zagadnienia tworzenia mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych. Przedstawiono również motywację do podjęcia wskazanego tematu. Ponadto określono cel, tezę i zakres rozprawy.

W rozdziale drugim przedstawiono obecny stan wiedzy dla zagadnień wykorzystywanych jako elementy składowe zbudowanego systemu. Wyszczególniono kilka problemów badawczych, które zostały pogłębione podczas prowadzonych prac w tym algorytmy wyznaczania głębi obrazu, algorytmy detekcji obiektów i metody wizualizacji danych przestrzenno-czasowych. Poza przeglądem literatury, zawarto w nim również wnioski oraz obserwacje wynikające z analizy aktualnie dostępnych rozwiązań.

Rozdział trzeci przedstawia założenia, sposób działania oraz szczegóły implementacyjne kluczowych elementów systemu będącego celem tej rozprawy. Zaprezentowano sposób działania każdej składowej systemu wraz ze szczegółowymi rezultatami testów przeprowadzanych w warunkach kontrolowanych na etapie badań.

Rozdział czwarty "Eksperymenty i weryfikacja działania systemu" zawiera testy utworzonego prototypu w warunkach produkcyjnych. Testy obejmowały szereg funkcjonalności i były wykonywane w terenie miejskim prezentując docelowy sposobu wykorzystania systemu.

Ostatni z rozdziałów „Podsumowanie i wnioski” przedstawia wnioski wynikające z otrzymanych wyników, sposoby zastosowania systemu w praktyce, a także określa potencjalne kierunki dalszych badań i możliwych udoskonaleń.

2

Omówienie istniejących rozwiązań

Stworzenie mapy na podstawie wielomodalnych danych przestrzennych i czasowych wymaga szeregu współpracujących ze sobą elementów począwszy od algorytmów zasilających system danymi na ich wizualizacji kończąc. Sposób działania opracowanego i zbudowanego w ramach pracy systemu można opisać w następujący sposób: i)urządzenia brzegowe (kamery na mieście i smartfony) wykorzystując sieci neuronowe do detekcji obiektów, algorytm śledzenia obiektów i badania głębi obrazu doprecyzowują położenie widzianego w obszarze kamery obiektu,ii)rezultaty są agregowane po stronie serwera i wizualizowane na żądanie użytkownika na platformie będącą aplikacją webową. Budowa każdego elementu została poprzedzona przeglądem najnowszych osiągnięć i rozwiązań podobnych problemów w literaturze co umożliwiło ich zastosowanie w systemie lub opracowanie własnych posiłkując się wiedzą zdobytą podczas analizy rozwiązań literaturowych. W dalszej części rozdziału przedstawiono krótki przegląd istniejących metod związanych z poruszanymi w pracy zagadnieniami.

2.1 Algorytmy badania głębi obrazu

Algorytmy służące do badania głębi obrazu umożliwiają estymację odległości pomiędzy ogniskową obiektywu, a obiektami występującymi na obrazie lub w skrajnym przypadku odległości mogą być wyznaczane dla pojedynczych pikseli obrazu. Są szeroko stosowane m.in. w systemach autonomicznych samochodów [26], automatyzacji tworzenia modeli 3D [38] i mechanizmach odtwarzania obrazów (np. na wskutek ich rozmycia) [6].

Ze względu na przyjęte założenia sprzętowe pominięto wyznaczenie głębi

z jawnym wykorzystaniem metod stereowizji i pokrewnych. Niejawnie elementy metody stereowizji takie mapy rozbieżności są jednak omawiane w grupie metod uczenia maszynowego.

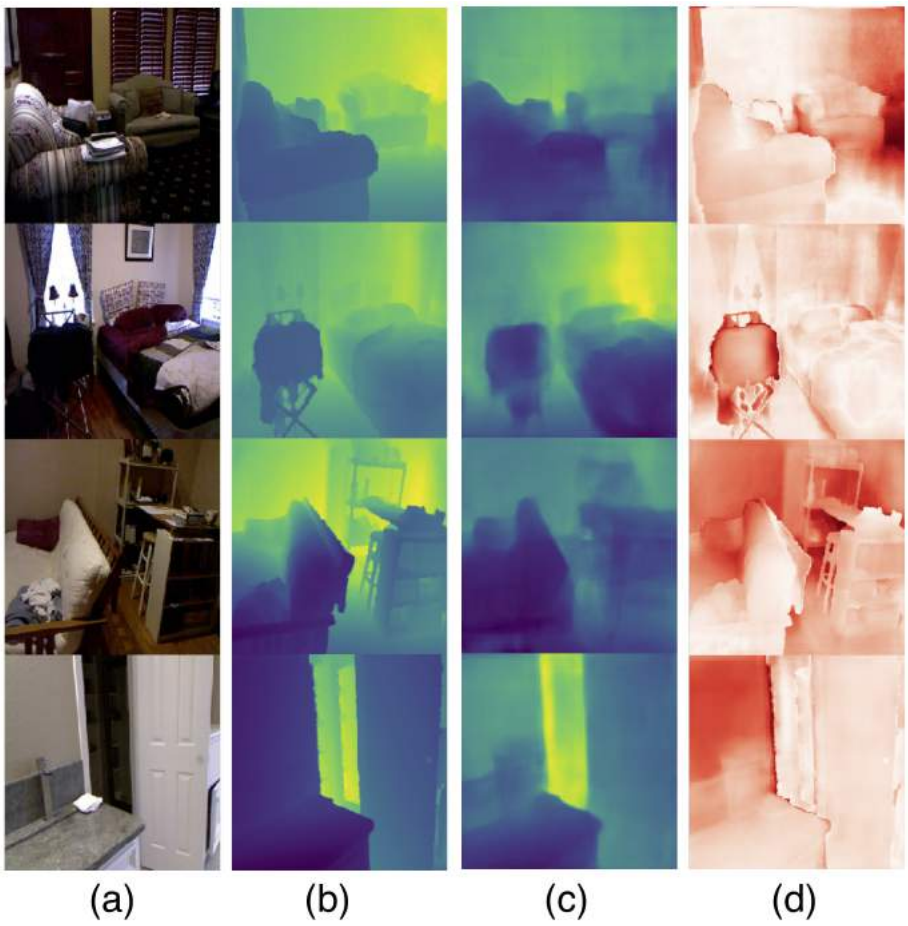
W literaturze można spotkać wiele rozwiązań problemu estymacji odległości z wykorzystaniem algorytmów AI. Część z nich opiera się na podejściu nadzorowanego i częściowo nadzorowanego uczenia maszynowego. Na wejściu, jest przekazywany obraz w skali szarości, a na wyjściu zwracany jest ten sam obraz z naniesionymi kolorami reprezentujący relacje odległości do obiektów znajdujących się na zdjęciu. Relacja odległości określa który z obiektów jest bliżej lub dalej bez podawania dokładnej odległości wyrażonej w jednostkach metrycznych.

Drugą grupą rozwiązań stanowią metody estymujące odległość od ogniskowej obiektywu do obiektów widocznych na zdjęciach wyrażoną w jednostkach metrycznych. W celu uzyskania dokładnych wyników konieczne jest posiadanie dodatkowych informacji np. o ogniskowej aparatu którym wykonano zdjęcie i wielkości fotografowanego obiektu również wyrażonej w jednostkach metrycznych co nie jest wymagane w metodach określających wyłącznie relację odległości.

2.1.1 Podejście oparte na nadzorowanym uczeniu maszynowym

Biorąc pod uwagę sukces głębokich splotowych sieci neuronowych w przetwarzaniu obrazu, zostało opracowanych wiele sieci przeznaczonych do estymacji głębokości. [94, 96, 52, 64, 75, 86, 70, 55, 44]. Dzięki wielopoziomowej informacji kontekstowej i strukturalnej z bardzo głębokich sieci (np. VGG [78] czy ResNet [31]) dokładność estymacji głębokości obrazu znacząco wzrosła [15, 24, 47, 49, 91]. Dalsze prace doprowadziły do powstania rozwiązań, które zapewniają zbliżoną dokładność, jednak przetwarzają obraz wielokrotnie szybciej na urządzeniach o ograniczonych zasobach [89]. Przykład działania modelu FastDepth pokazano na rysunku 2.1.

Zaproponowane w literaturze modele predykcyjne, bazując na danych te-



źródło: https://github.com/dwofk/fast-depth

Rysunek 2.1: Rezultat uzyskany przez model FastDepth, zoptymalizowany do działania na urządzeniach o ograniczonych zasobach (mikrokontrolerach i urządzeniach mobilnych). (a) kolorowy obraz wejściowy, (b) prawdziwa odległość naniesiona na obraz, (c) rezultat działania modelu, (d) mapa błędów pomiędzy rezultatem pracy modelu, a prawdziwą odległością. Bardziej intensywny kolor wskazuje na wyższy błąd

stowych i walidacyjnych składających się ze zdjęć wraz z skanami laserowymi wykonanymi dla danej scenerii określającymi dokładną odległość do widocznych na obrazie obiektów, w kolejnych iteracjach treningu dostosowują się, określając odległości na podstawie relacji pomiędzy różnicą w kolorach i zmieniającym się oświetleniu na obrazie.

Istnieje problem związany z wykorzystaniem głębokich sieci do ekstrakcji cech z obrazu. Wielokrotne operacje łączenia w tych głębokich ekstraktorach

cech szybko zmniejszają rozdzielczość przestrzenną map cech obrazu (ang. feature maps). W celu zapobiegania skutkom tego procesu w niektórych rozwiązaniach zastosowano sieci zwiększające rozdzielczości (ang. multi-scale NN), które etapowo udoskonalają oszacowaną mapę głębokości od niskiej do wysokiej rozdzielczości przestrzennej za pomocą innych sieci [16, 15]. Innym sposobem niwelowania negatywnych efektów wskazanego problemu jest dodanie połączeń skrótu (ang. skip connection) do łączenia mapy głębi o niskiej rozdzielczości przestrzennej w głębszych warstwach z mapą głębi o wysokiej rozdzielczości w niższych warstwach [91] lub zastosowanie wielowarstwowych sieci dekonwolucyjnych [24].

Modele łączące lokalne przewidywania [38, 74, 39, 48, 54, 8, 45, 1, 76, 69, 23, 30, 93] generują bryły geometryczne poprzez wyszukiwanie na obrazie wejściowym jednolicie oznakowanych regionów obliczając superpiksele (grupy połączonych ze sobą pikseli, które mają podobną reprezentację na obrazie). Po ich zgrupowaniu następuje ich przetworzenie w celu detekcji obszaru z danej klasy rozpoznawanej przez model (tj. ziemia, pion, niebo). Na podstawie rezultatu detekcji następuje dopasowanie sceny ze zdjęcia do estymacji odległości typowej dla danej etykiety.

Modele oparte o nieparametryczne próbkowanie scen [43, 65] działają wykorzystując algorytm dopasowywania obrazu wejściowego do najbardziej zbliżonego rezultatu zapisanego w bazie danych (wykorzystując np. algorytm SIFT). Po znalezieniu najbardziej zbliżonych kandydatów następuje interpolacja obrazu z bazy w taki sposób, by dopasować go do struktury obrazu wejściowego. Następnie, używając globalnej procedury optymalizacji do wykonanych interpolacji kandydatów uzyskuje się oszacowanie głębi dla każdego piksela obrazu wejściowego.

Pola losowe Markowa (ang. Markov Random Fields - MRF) i jego warianty również są wykorzystywane do estymacji odległości na obrazie 2D [74, 98, 56]. MRF (Markov Random Fields) są z powodzeniem wykorzystywane w wielu aplikacjach, w których cechy lokalne okazują się niewystarczające i konieczne jest wykorzystanie informacji kontekstowych obrazu. Do modelowania zależno-

ści przestrzennych w obrazach z wykorzystaniem MRF najlepsze rezultaty uzyskiwano po zastosowaniu algorytmu dyskryminacyjnych pól losowych Kumara i Heberta, który wykorzystywał rozkład Laplace'a do identyfikacji rozkładu struktur występujących na obrazach [73].

Badania wskazują, że wyższą dokładność uzyskują modele, które przetwarzają krótkie filmy niż pojedyncze obrazy wyjęte z kontekstu [21, 13]. Wynika to z faktu, że zapamiętując rezultat poprzedniej estymacji, kolejne interpolacje umożliwiają uzyskanie coraz dokładniejszych wyników dla punktów widzianych z różnej perspektywy podobnie jak w problemie SfM (Structure from Motion). Zaobserwowana zależność ukierunkowała dalsze badania i zaproponowanie własnego algorytmu estymacji lokalizacji, opisanego w kolejnym rozdziale rozprawy, wykorzystującego wskazane zjawisko.

2.1.2 Estymacja odległości poprzez wykorzystanie samo nadzorującego (ang. self-supervised) uczenia maszynowego

Pozyskiwanie dużej liczby map głębi jest procesem kosztownym. Często wymaga zupełnie innej platformy zbierania danych niż docelowa (np. skanerów laserowych). Metody uczenia samo nadzorującego okazały się sposobem na obejście tego ograniczenia. Sieci przekształcające przestrzeń (ang. Spatial Transformer Networks) [41] umożliwiły przetwarzanie scen symulując przepływ optyczny [92, 63], głębokość [25, 53] i pozycję kamery [97].

Bazując na rozbieżności pikseli pomiędzy parami zdjęć danej sceny, które zostały wykonane z dwóch pozycji kamery znajdujących się w niewielkiej odległości od siebie algorytmy głębokiego uczenia budują perspektywę bazując na rozbieżności pikseli pomiędzy dwoma zdjęciami [91]. Mapa głębi danej sceny jest szacowana na podstawie obrazu widoku wejściowego, a następnie algorytm DIBR (ang. Differentiable Interpolation-based Renderer) łączy mapę głębi z widokiem wejściowym w celu wygenerowania brakującego widoku pary stereo [19].

Inną metodą samo nadzorującego się uczenia maszynowego jest wykorzystanie filmów przedstawiających na obrazie scenę do której algorytm stara się

oszacować głębię. Podobnie jak w przypadku zdjęć obiektu, które zostały wykonane kamerą z punktów w niewielkiej odległości od siebie, również dla filmów algorytm stara się oszacować głębię poprzez rozbieżności pikseli pomiędzy kolejnymi klatkami. Cechą znacznie wyróżniającą trening sieci przy wykorzystaniu filmów od wykonanego na obrazach jest konieczność estymacji nie tylko głębokości, ale również położenia i orientacji kamery pomiędzy kolejnymi klatkami filmu. Estymacja położenia i orientacji kamery jest wykorzystywana wyłącznie na etapie treningu sieci, by umożliwić modelom reagowanie na zmianę położenia określonych regionów (np. na wskutek zbliżenia kamery.) [97].

2.1.3 Metody szacowania odległości wyrażonej w jednostkach metrycznych

W sytuacji, gdy informacja o relacjach położenia obiektów względem kamery (bliżej/dalej) jest niewystarczająca stosuje się metody szacowania odległości od obiektu wyrażonej w jednostkach metrycznych. Istnieje kilka metod takiego oszacowania z których najdokładniejsze są metody sprzętowe (tj. czujniki LIDAR). Przykład jego działania zaprezentowano na rysunku 2.2. Metody sprzętowe są często wykorzystywane w celu walidacji skuteczności i dokładności działania pozostałych technik.

Przez lata powstało wiele rozwiązań bazujących na kamerach RGB o niewielkiej rozdzielczości, stanowiących alternatywę dla czujników LIDAR.

Skrót LIDAR pochodzi od angielskiego "Light Detection and Ranging" i stanowi metodę pomiaru odległości poprzez pomiar czasu przelotu wiązki lasera odbitego od obiektu. Pomiary wykonane tą techniką cechują się wysoką dokładnością oraz niezależnością od warunków oświetleniowych i pogodowych (za wyjątkiem mgły). Dzięki temu możliwe jest określenie dokładnego położenia danego punktu w układzie emitera wiązki co jest wykorzystywane m.in. do tworzenia cyfrowych modeli rzeczywistych obiektów. Utworzona chmura punktów stanowi bazę dla dalszej obróbki umożliwiającej odtworzenie zeskanowanego obiektu z dużą wiernością. W zależności od długości emitowanej fali, czujnik jest w stanie określać odległości od obiektu znajdującego się nawet 200m od

emitera. Opisane rozwiązanie ma jednak wady. Wzrost odległości od czujnika LIDAR powoduje zmniejszenie rozdzielczość odbieranego sygnału przez co detekcja obiektów jest możliwa tylko w niewielkiej odległości od czujnika (zależnej od maksymalnej rozdzielczości skanowania) [32, 95].

źródło: https://sketchfab.com/3d-models/wat-chaiwatthanaram-scan-using-ipad-lidar-2358594916fa4abf9aadf6b20f886f12

Rysunek 2.2: Chmura punktów, która jest efektem działania skanera LIDAR, próbkującego poza położeniem również kolorystykę danego obszaru.

W przypadku fotografowania obiektu, który ma znane wymiary odległość do niego może zostać wyliczona wykorzystując podobieństwo trójkątów (w przypadku otworkowego modelu kamery). Na podstawie parametrów aparatu (tj. jego ogniskowej oraz wielkości sensora), informacji o wielkości obiektu w rzeczywistości oraz rozmiarze obiektu na zdjęciu można z dużą dokładnością oszacować odległość od ogniskowej obiektywu. Idealne warunki (najdokładniejsze oszacowanie) można uzyskać, gdy kamera jest zwrócona wprost na obiekt. Istnieją również metody, które cechuje wysoka dokładność, gdy kamera jest zwrócona pod pewnym kątem do fotografowanego obiektu [5]. Metoda jest skuteczna w przypadku określania odległości od obiektów o ustandaryzowanych lub nieznacznie różniących się rozmiarach (np. tablica rejestracyjna lub samochód typu sedan).

Metoda LBPA (Laser Beam Pixel Area) jest techniką mierzenia odległości od obiektu wykorzystującą kamerę oraz wskaźnik laserowy skierowany w cen-

tralny punkt pola widzenia kamery. Po skierowaniu wiązki lasera na obiekt do którego jest określana odległość następuje przetworzenie obrazu z kamery w celu wykrycia liczby pikseli na których znajduje się laser (im dany obiekt znajduje się bliżej tym wiązka lasera zajmuje większą część na obrazie). W ten sposób da się wyznaczyć zależność określającą liczbę pixeli na których jest widoczna wiązka lasera od odległości od obiektu. Rozwiązanie jest bardzo proste w budowie i kalibracji, a dodatkowo udowodniono jego skuteczność dla obiektów od których odległość wynosi do 10m z dokładnością +|- 3% zarówno wewnątrz jak i na zewnątrz budynku [77]. Wadą tej metody jest możliwość badania odległości do pojedynczego elementu na raz. W przypadku kamer lub czujników LIDAR skanowany jest cały obszar podczas pojedynczej iteracji.

Kolejna z metod określania odległości od obiektu jest wykorzystywana w kamerach umieszczanych w samochodach. Metoda wymaga znajomości parametrów wewnętrznych kamery w tym ogniskowej, oraz parametrów zewnętrznych jak kąt nachylenia osi kamery względem płaszczyzny drogi i rozdzielczość obrazu. Obraz przechwycony przez kamerę jest przetwarzany pod kątem detekcji obiektów znajdujących się wzdłuż osi optycznej kamery. Po określeniu współrzędnych dolnego punktu styku obiektu z drogą (np. poprzedzającego samochodu) następuje obliczenie kąta pomiędzy prostą łączącą ten punkt z centralnym punktem kamery, a osią optyczną. Później następuje przeliczenie uzyskanego kąta na odległość. Opisana metoda daje prawidłowe rezultaty z dokładnością do 1m nawet dla dystansu 70m od kamery [42] jednak jest skuteczna wyłącznie na płaskim obszarze (nachylenie drogi nie może znacznie różnić się od siebie), a dodatkowo kamera musi być umieszczona w ściśle określonym miejscu.

2.2 Detekcja obiektów, kwantyzacja modeli

Sieci neuronowe stanowią skuteczne rozwiązanie problemu detekcji obiektów na obrazach z kamery. Wykonane w ostatnich latach badania doprowadziły do znacznego ich rozwoju i stworzenia wielu frameworków znacznie różniących się od siebie stosowanymi rozwiązaniami. Zwiększanie złożoności sieci na pewnym

etapie spowodowało problem z ich uruchomieniem na urządzeniach o niewielkich zasobach, tak by dostarczały rezultatów swojego działania w akceptowalnym czasie. Rozwiązaniem okazała się kwantyzacja polegająca na zmniejszeniu precyzji wag połączeń wewnątrz sieci (z float32 na float16 lub int8). Po przeprowadzeniu takiego zabiegu sieć neuronowa wymaga mniej czasu na przetworzenie pojedynczej klatki obrazu pracując na procesorach CPU (ang. Central Processing Unit) [83].

2.2.1 Detekcja obiektów z wykorzystaniem głębokich sieci neuronowych

Modelami głębokimi nazywa się sieci neuronowe o głębokich strukturach (topologie z wieloma warstwami). Historia sieci neuronowych sięga lat 40-tych XX wieku [68], a ich pierwotną intencją było symulowanie działania ludzkiego mózgu w celu rozwiązywania ogólnych problemów mechanizmu nabywania wiedzy. Popularność zyskały w latach 80. i 90. XX wieku wraz z zaproponowaniem algorytmu wstecznej propagacji przez Hintona i innych [71]. Jednakże, z uwagi na częste przetrenowanie (zbytnie dostosowanie wag sieci do danych walidacyjnych) spowodowane małą ilością danych treningowych, ograniczoną moc obliczeniową i niewielką wydajnością w porównaniu z innymi narzędziami uczenia maszynowego, zaprzestano ich rozwoju na początku lat 2000. Ponowny rozwój sieci neuronowych zapoczątkował przełom jaki wprowadziły w rozpoznawaniu mowy w roku 2006 [37] [36]. Do rozwoju sieci neuronowych przyczynił się również dostęp do dużych i ogólnie dostępnych poetykietowanych/adnotowanych zbiorów danych jak ImageNet [14] oraz rozwój systemów obliczeń równoległych takich jak klastry GPU.

Jedną z miar weryfikacji skuteczności detekcji wykonywanych przez sieć jest średnia precyzja (ang. mean Avarage Precision - mAP) [34, 17, 72, 18]. Miara mAP dla zbioru detekcji jest średnią interpolowanych AP dla każdej klasy. Wartością AP nazywamy polem powierzchni pod krzywą precyzji dla uzyskanych detekcji. Krzywą precyzji konstruuje się, mapując najpierw każdą detekcję na najbardziej pokrywającą się z nią instancję obiektu zaznaczonego podczas pro-

cesu etykietowania (ang. ground true). Następnie, najwyżej oceniona detekcja jest uznawana za prawidłową, a wszystkie inne detekcje za fałszywie pozytywne. Następnie obliczamy wartości współczynników wycofania (ang. recall) i precyzji dla coraz większych podzbiorów detekcji, zaczynając od najwyżej ocenionej detekcji i dodając pozostałe w kolejności malejącej liczby uzyskanych przez nie punktów. Im wyższy wynik mAP, tym dokładniejsze detekcje generuje model.

Sieci neuronowe do detekcji obiektów składają się z warstw, których głównym zadaniem jest ekstrakcja niskopoziomowych cech obrazu tworząc w ten sposób mapę cech (ang. feature map), a następnie scalanie wyników ich lokalizacji. Stworzone w ten sposób obszary zawierają obiekt, którego model nauczył się rozpoznawać tj. wewnątrz sieci zapisano zestaw cech jakie są przyporządkowane danej klasie obiektu. W celu rozpoznawania różnych obiektów, sieć musi wydobywać ich cechy wizualne, które zapewniają semantyczną reprezentację. Ze względu na fakt mnogości warunków oświetleniowych i tła, sieci neuronowe potrzebują ogromnej liczby danych do treningu, by zbudować deskryptor cech. Poza ekstrakcją cech równie istotny jest klasyfikator, który przyporządkuje zlokalizowany obszar do jednej z wyuczonych kategorii obiektów. Można w tym celu zastosować warstwy w pełni połączone (ang. fully connected layers) maszynę wektorów podpierających (ang. Supported Vector Machine) [12], AdaBoost [22] lub model oparty na częściowej deformacji (Deformable Part-based model) [20]. Ostatni z wymienionych jest szczególnie pomocny w przypadku wyszukiwania obiektów, które mogą cechować się dużymi deformacjami.

Najczęstszym podejściem w wyszukiwaniu obiektów na obrazie jest przeszukiwanie całego obrazu za pomocą wieloskalowalnego okna przesuwnego. Takie podejście wynika z faktu, że wyszukiwane obiekty mogą znajdować się w dowolnym miejscu obrazu i mieć różne proporcje i rozmiary. Ogromną wadą takiego podejścia jest kosztowność takiego wyliczenia dla pojedynczego okna.

Konwolucyjna/splotowa sieć neuronowa (ang. Convolutional Neural Network - CNN) jest reprezentatywnym modelem głębokiego uczenia [50]. Każda warstwa CNN stanowi mapę cech. Mapa cech warstwy wejściowej jest trójwymiarową macierzą intensywności pikseli dla różnych kanałów wybranego modelu barw

(np. RGB). Mapa cech dowolnej warstwy wewnętrznej jest indukowanym wielokanałowym obrazem którego piksel lub ich zbiór może być postrzegany jako konkretna cecha. Na przetwarzanych wewnątrz sieci mapach cech są wykonywane różne rodzaje przekształceń takie jak filtrowanie i łączenie [46].

Współcześnie wykorzystywane CNN można podzielić na 2 zasadnicze typy. Pierwszy z nich determinuje położenie obiektu na podstawie zgrubnego przeskanowania całego obrazu, a następnie koncentrowanie się na regionach zawierających określone cechy (weryfikując tylko te obszary, które posiadają dane cechy z wyższym niż wymagany współczynnikiem pewności). Należąca do tego typu sieć R-CNN wykonuje selektywne wyszukiwanie [80] generując około 2k propozycji regionów dla każdego obrazu, a następnie opierając się na grupowaniu zwraca dokładne obszary występowania obiektów, redukując tym samym przestrzeń poszukiwań w ich wykrywaniu [20, 14].

Drugim typem współcześnie wykorzystywanych CNN są sieci, które określają położenie obiektów mapując bezpośrednio piksele obrazu na ramki ograniczające obiekty. Proces jest wykonywany na podstawie informacji zwracanych przez globalnie wykonaną klasyfikację, a następnie regresję danego obszaru. Takie działanie znacznie przyśpiesza prędkość przetwarzania pojedynczej klatki obrazu przez sieć eliminując konieczność przeglądania wszystkich obszarów. Wskazane podejście umożliwia zastosowanie sieci w aplikacjach czasu rzeczywistego na urządzeniach o ograniczonych zasobach. Przykładem frameworków będących reprezentantami tego typu sieci są YOLO (You Olny Look Once) [3] i SSD (Single Shot Multibox Detector) [57].

2.2.2 Kwantyzacja modeli do detekcji obiektów

Podczas, gdy sieci neuronowe wygrywają kolejne wyzwania w tematyce przetwarzania obrazu, narastającym problemem staje się rosnąca kosztowność obliczeniowa spowodowana nieustannym rozrastaniem się architektury kolejnych generacji sieci. Wiąże się to z koniecznością posiadania większej ilości pamięci RAM do ich uruchomienia. Trening sieci neuronowych na konwencjonalnym sprzęcie cyfrowym ogólnego przeznaczenia (architektura Von Neumanna) okazało się

wysoce nieefektywny z powodu ogromnej liczby operacji mnożenia i akumulacji (MAC) wymaganych do obliczenia ważonych sum wejść neuronów. Obecnie, DNN są trenowane prawie wyłącznie na jednej lub wielu posiadających odpowiednią moc obliczeniową jednostkach GPU (Graphic Processing Units) [9]. Uruchomienie DNN na docelowych urządzeniach o niskim poborze mocy jest dużym wyzwaniem, jeśli chcemy zapewnić przetwarzanie obrazu w czasie rzeczywistym zarówno na urządzeniach posiadających jak i pozbawionych wsparcia sprzętowego w postaci procesora graficznego.

W celu zastosowania CNN na urządzeniach mobilnych o ograniczonych zasobach, konieczne okazało się zmniejszenie wysokich wymagań dotyczących zasobów niezbędnych do wykorzystania takich modeli. W tym celu zaproponowano wiele technik kompresji i kwantyzacji sieci [40, 29, 90], które umożliwiają ich efektywne wykorzystanie na platformie o ograniczonych zasobach. Wskazane urządzenia charakteryzują się niewielką ilością dostępnego RAMu (1-2 GB), brakiem lub kartą graficzną o niskich parametrach i procesorem niskonapięciowym.

Alternatywą do jednostek GPU stały się układy FPGA. Stanowią one platformę dla implementacji już wyuczonej sieci DNN. Pozwala to na zastosowanie zoptymalizowanych jednostek arytmetycznych o niskiej precyzji w celu osiągnięcia znacznego wzrostu wydajności [85]. Istniejące akceleratory oparte na FPGA stworzyły konstrukcje sprzętowe, które znacznie usprawniają obliczenia wykonywane w jednolitej 16-bitowej precyzji [84, 60] z niewielkim wpływem na jej dokładność. Dodatkowo, w układach FPGA możliwe jest zastosowanie procesu binaryzacji sieci (ang. binarised networks) [81] co znacznie zwiększa szybkość przetwarzania danych przez sieć.

Jednym ze sposobów na redukcję rozmiarów sieci oraz liczby koniecznych obliczeń niezbędnych do przetworzenia pojedynczej klatki obrazu jest wykorzystanie mechanizmu grupowania połączeń sieci w taki sposób, by w danej grupie znalazły się wszystkie połączenia zawierające tą samą wagę. Następnie użycie funkcji haszującej sprawia, że każda z grup korzysta z pojedynczej wartości parametru [7]. Dzięki takiemu zabiegowi nie ma potrzeby alokowania wielu

zmiennych o tej samej wartości w pamięci programu. Zbliżoną metodą jest wykorzystanie algorytmu k-means dla wag sieci i ich grupowanie na podstawie rezultatu klasteryzacji. [27]. Obie te metody skupiają się jednak wyłącznie na warstwach w pełni połączonych (jako wymagających najwięcej pamięci). Dalszy rozwój badań doprowadził do wypracowania metod umożliwiających zmniejszenie rozmiaru sieci o rząd wielkości w stosunku do poprzednich metod. Obecnie najskuteczniejszą propozycją jest metoda, która optymalizuje sieć kilkuetapowo. W pierwszej kolejności sieć jest redukowana na etapie treningu podczas którego połączenia nie wnoszące nic w rezultat są usuwane podczas kolejnych iteracj. Później następuje klasteryzacja wytrenowanych wag, by w ostatnim kroku zastosować kodowanie Huffmana [62]. Kod Huffmana jest optymalnym kodem prefiksowym powszechnie stosowanym do bezstratnej kompresji danych. Wykorzystuje on słowa kodowe o zmiennej długości do kodowania symboli źródłowych. Tablica jest wyprowadzana z prawdopodobieństwa wystąpienia dla każdego symbolu. Bardziej powszechne symbole są reprezentowane przy użyciu mniejszej liczby bitów [82]. Pierwsze dwa kroki są wykonywane iteracyjnie aż do uzyskania możliwie najmniejszej liczby wymaganych połączeń dopuszczając, określoną w parametrze, niewielką utratę skuteczności.

2.3 Algorytmy śledzenia obiektów na obrazie z kamery

Większość dostępnych technik detekcji obiektów ruchomych została opracowana dla scen rejestrowanych przez nieruchomą kamerę. Metody te wykorzystują segmentację każdego obrazu na zbiór regionów reprezentujących poruszające się obiekty z wykorzystaniem algorytmu różnicowania tła [10]. Ze względu na charakterystykę działania, techniki wykorzystujące różnicowanie tła nie mogą być stosowane w urządzeniach mobilnych, które z definicji zmieniają swoje położenie, a tym samym widoczne w obrazie kamery tło.

Dostępność tanich sensorów wideo z możliwością obrotu, pochylenia i powiększenia montowanych w poruszających się platformach sprawiła, że uwaga

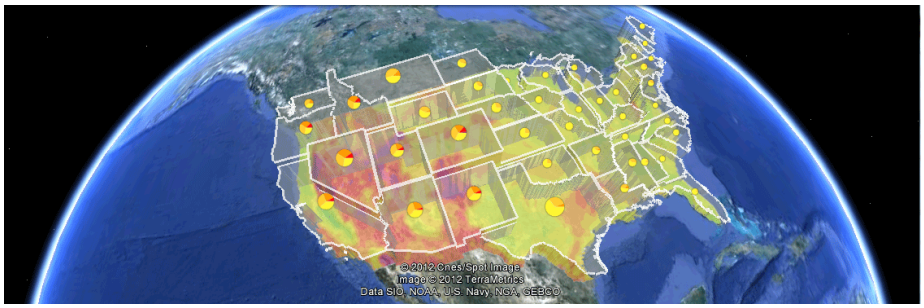
badaczy skupiła się na detekcji poruszających się obiektów w strumieniach wideo ze zmiennym tłem. Wysoką skutecznością cechują się metody wykorzystujące lokalne modelowanie tła za pomocą mieszanin K-Gaussowskich [28] pozwalając na przetwarzanie strumieni wideo ze zmiennym w czasie tłem. Metody te dają zadowalające rezultaty i mogą być zaimplementowane do przetwarzania obrazu w czasie rzeczywistym dla urządzeń o ograniczonych zasobach.

Metody wykorzystujące korelację filtrów do śledzenia obiektów można podzielić na dwie zasadnicze grupy różniące się uwzględnianiem lub nie translacji obrazu. Pierwsza z nich, nie uwzględniająca translacji, jest oparta na syntetycznej funkcji dyskryminacyjnej (SDF) [66]. Druga z kolei dopuszczająca translację opiera się na minimalizacji średniej korelacji [61] lub minimalizacji sumy błędu kwadratowego (MOSSE) [4]. Wskazane metody działają szybko jednak ich skuteczność jest mocno zależna od warunków obrazu np. występowania przeszkód pomiędzy obserwowanym obiektem lub szybkością przemieszczania się obiektu na obrazie.

W dalszej części rozprawy przeprowadzono eksperymenty podczas których testowano algorytmy śledzenia obiektów w celu wyłonienia najskuteczniejszego i najszybszego na urządzeniu o ograniczonych zasobach.

2.4 Wizualizacja danych przestrzenno-czasowych

Wizualizacja pozwala na prezentację danych w formie umożliwiającej człowiekowi łatwiejsze ich zrozumienie oraz wyciągnięcie wniosków z ich analizy. Językiem zorientowanym na wizualizację danych geograficznych jest KML (Keyhole Markup Language). Oparto go na popularnym formacie XML, jednak zawiera on wyłącznie argumenty niezbędne do wizualizacji oraz umożliwia dodawanie adnotacji do prezentowanych map i obrazów [88, 87]. Stanowi standard przyjęty przez Google i obecnie jest wykorzystywany w narzędziu Google Earth jako podstawowa forma wymiany danych. Ponieważ KML zapewnia różnorodność opcji prezentacji, stanowi atrakcyjne narzędzie do naukowej wizualizacji zjawisk geograficznych [35], a przykład jego użycia pokazano na rysunku 2.3.



źródło: https://developers.google.com/kml/

Rysunek 2.3: Przykład wizualizacji wykonanej wykorzystując mechanizmy dostępne w KML

Poza wspomnianym Google Earth, dostępne są również inne programy stanowiące mniej popularną alternatywę prezentującą dane przestrzenno-czasowe tj. Microsoft Bing Maps, AutoDesk LandXplorer, NASA World Wind, ESRI's ArcGlobe i Leica's Virtual Explorer. Inne przykłady map będące na licencji open source to Open Street Map, Marble, OSSIM Planet i Cesium.

Wizualizacja świata rzeczywistego za pomocą mapy ma wiele zastosowań. Może być wykorzystana do symulacji występujących zjawisk i trendów w mikroskali lub jako kanwa dla osadzenia abstrakcyjnych informacji o przestrzeni. Dzięki zapisywaniu poza danymi lokalizacyjnymi również danych czasowych, można przeprowadzać analizy zmian zachodzących w czasie lub analizując dane historyczne można opracować model predykcji dla wybranego wektora cech [2].

3

Koncepcja zaproponowanego rozwiązania i dobór jego składowych

3.1 Opis ogólny systemu

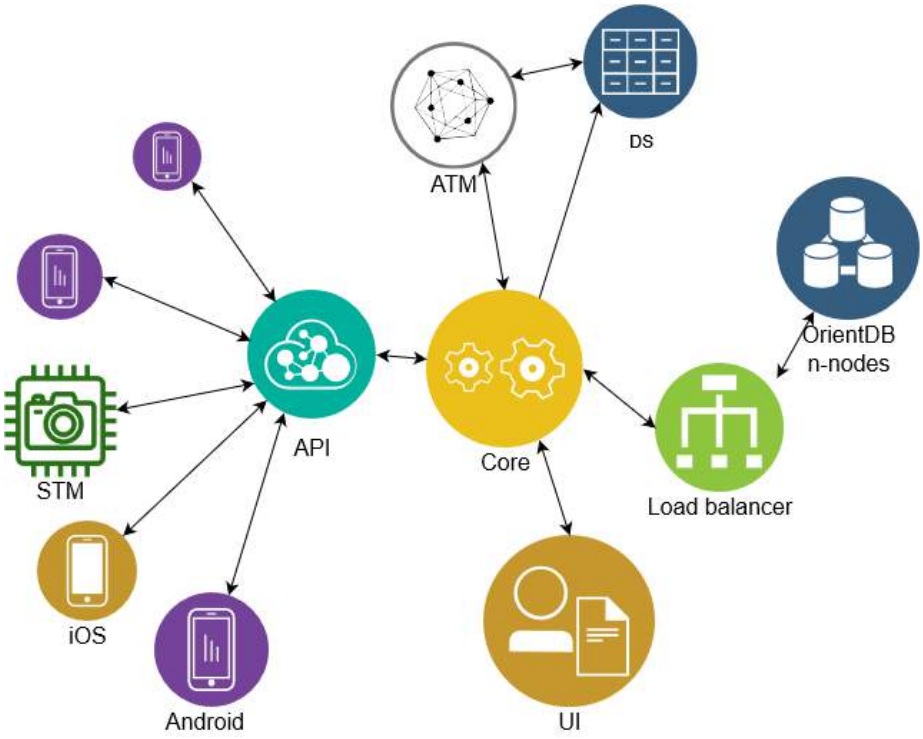
Obecnie wykorzystywane systemy wizualizacji map takie jak najpopularniejszy Google Maps czy darmowa alternatywa w postaci Open Street Map z powodzeniem realizują funkcję przetwarzania i wizualizacji danych terenowych, a także dostarczania użytkownikom dodatkowych informacji związanych z określonym miejscem np. lokalizacją najbliższego sklepu czy stacji benzynowej. Zawarte we wskazanych systemach mapy posiadają również zewnętrzne API umożliwiające tworzenie własnych systemów nawigacji i przetwarzanie dostępnych informacji. Niestety, dla analityków chcących badać mechanizmy dynamicznych zmian zachodzących w danym obszarze np. w mieście takie informacje są niewystarczające. System wizualizacji mapy na podstawie danych temporalnych i wielomodalnych stanowi rozwiązanie kierowane do szerokiego grona odbiorców. Wizualizacja obiektów zmieniających swoje położenie, w połączeniu z możliwością sięgnięcia do archiwalnych danych w celu uzyskania informacji co znajdowało się na określonym obszarze w danym czasie dostarczy wielu cennych informacji przeznaczonych do procesu analizy i wnioskowania. Przykładowo podmiot publiczny chcący wytyczyć nowe ścieżki lub stacje rowerowe odpowiadając na realne potrzeby mieszkańców będzie mógł to zrobić sięgając do archiwalnych danych systemu z okresu letniego dotyczących rowerzystów przemieszczających się w danym regionie.

System składa się z szeregu modułów, których prawidłowe działanie jest nie-

zbędne do osiągnięcia zamierzonego celu. Architekturę rozwiązania pokazano na schemacie 3.1. Urządzenia brzegowe zasilające system to jest urządzenia mobilne z systemem operacyjnym Android, iOS oraz kamery na mieście przetwarzające lokalnie obraz pod kątem detekcji oraz śledzenia obiektów. Dodatkowo określają estymację odległości pomiędzy ogniskową kamery, a wykrytym obiektem, i dostarczają te dane do systemu za pośrednictwem zaimplementowanego API serwera TCP. Dane te są kolejno przekazywane do serwera centralnego, który odpowiada za porównywanie danych archiwalnych z aktualnie nadsyłanymi co służy doprecyzowaniu lokalizacji obiektów. Przetworzone informacje są przekazywane do klastra bazy danych (OrientDB n-nodes) za pośrednictwem systemu zapewniającego równomierne obciążenie każdego z nodów (Load balancer). Uruchomienie wizualizacji o aktualnym stanie wybranego regionu powoduje sięgnięcie do bazy danych, a następnie przesyłanie do UI nowych elementów nadsyłanych do systemu (bezpośrednio, równolegle zapisując informacje do bazy w osobnym procesie). Budowana w ten sposób mapa cechuje się dużo wyższym poziomem szczegółowości i aktualności czasowej niż mapy dostępne obecnie. System dynamicznie mapuje i odświeża dane dotyczące nie tylko statycznych obiektów takich jak budynki lub znaki drogowe, ale również obiektów, które zmieniają swoje położenie (takich jak rowery pozostawione przed biblioteką czy samochód zaparkowany na pasie zieleni). Czas potrzebny na wizualizację na mapie informacji o obiektach na danym obszarze wynosi do 2s od chwili otrzymania informacji o położeniu obiektu przez serwer dla fragmentu mapy wynoszącego około $1km^2$.

W celu zapewnienia "narastającej w czasie" szczegółowości mapy, system posiada dodatkowy moduł automatycznego trenowania modeli (ATM), który tworzy modele sieci neuronowych dla urządzeń brzegowych bazując na danych zawartych w zbiorze danych (DS). Podczas treningu sieci wykorzystuje się aktywne uczenie metodą pobierania próbek z puli (active learning, Pool-Based Sampling), by zapewnić wysoką efektywność uczenia i minimalizować czas dużego obciążenia dedykowanego serwera wyposażonego w GPU.

Mapa obszaru jest budowana na podstawie informacji odbieranych z kilku



Rysunek 3.1: Schemat opracowanego w ramach badań systemu do tworzenia mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych. System składa się z: urządzeń brzegowych, API, rdzenia systemu (Core), modułu równoważenia obciążenia (load balancer), klastra bazy danych (OrientDB n-nodes), modułu automatycznego trenowania modeli (ATM), modułu zarządzania plikami nadesłanymi przez użytkowników, które zostaną wykorzystane do treningu (DS) oraz interfejsu użytkownika (UI).

rodzajów urządzeń końcowych. Dla każdego z nich wykorzystano dedykowane rozwiązania nieznacznie różniące się działaniem między sobą, jednak dostarczające identycznych funkcjonalności. System zbiera oraz przetwarza informacje ze smartfonów opartych na systemie iOS, Android oraz mikrokontrolerach z rodziny STM model F746G i Nvidia Jetson Nano. W przypadku smartfonów lokalizacja wykrytego obiektu jest określana na podstawie danych z kilku źródeł przetworzonych bezpośrednio na urządzeniu tj. informacji z żyroskopu, modułu GPS oraz rezultatów pracy algorytmu estymacji głębi obrazu i kwantyzowanej sieci neuronowej do detekcji obiektów. Agregacja wskazanych danych pozwala na uzyskanie wyższej dokładności w określaniu położenia obiektów w świecie rzeczywistym niż opieranie się wyłącznie na lokalizacji pozyskiwanej z modułu GPS. Przesłane na serwer informacje są porównywane z danymi aktualnie zapisanymi w bazie noSQL (OrientDB), pochodzącymi z innych urządzeń. Analizując rezultaty detekcji, czas oraz lokalizację z której pochodzą odebrane dane, lokalizacje obiektów na mapie są dynamicznie modyfikowane w tym usuwane lub dodawane. Podczas badań udało się uzyskać dokładność lokalizacji poniżej 10m względem wartości referencyjnej przy średniej jakości sygnału GPS (działającego z dokładnością około 15m).

3.2 Warstwa pozyskiwania danych od urządzeń brzegowych

Stworzenie mapy, która wizualizuje aktualny stan wybranego regionu, nie byłoby możliwe bez warstwy pozyskiwania danych, której zadaniem jest przetwarzanie oraz dostarczanie informacji do systemu. W celu optymalizacji działania serwera, wstępna estymacja lokalizacji wykonywana jest bezpośrednio na urządzeniu brzegowym. W kolejnym kroku lokalizacje obiektów z danego regionu są agregowane po stronie serwera w celu uspójnienia danych, a doprecyzowane lokalizacje są zapisywane w bazie noSQL. Największym wyzwaniem tej warstwy jest zapewnienie możliwie jak największej dokładności wykonywanych przez urządzenia brzegowe estymacji, by na etapie synchronizacji danych z łatwością wykryć

redundantne obiekty i scalać je, doprecyzowując w ten sposób ich położenie na mapie. W tym celu, poza badaniami nad szybkością i możliwościami przetwarzania obrazu na urządzeniach o niewielkich zasobach, wykonano również badania nad dokładnością pomiarów modułów GPS będących często standardowym wyposażeniem smartfonów.

3.2.1 Przetwarzanie danych po stronie urządzeń mobilnych z Android/iOS

Urządzenia posiadające system Android i iOS stanowią główne źródło danych przesyłanych do systemu. Z uwagi na fakt konieczności oszczędzania zasobów urządzenia, stworzone aplikacje korzystają w miarę możliwości z natywnych funkcji i bibliotek, by minimalizować obciążenie systemu. Aplikacja umożliwia wykonanie użytkownikowi następujących zadań:

- detekcja przy użyciu sieci neuronowej, która pracuje bezpośrednio na urządzeniu określonej klasy obiektów znajdujących się w wybranej lokalizacji
- segmentacja obrazu z kamery w celu przesłania na serwer jego fragmentu zawierającego przykład klasy obiektu, która nie jest aktualnie rozpoznawana przez sieć, a następnie przesłanie zdjęcia, wraz z zaznaczonym przez algorytm segmentacji obrazu ramkami ograniczającymi (ang. bounding boxy). Efekty działania algorytmu segmentacji mogą zostać poprawione przez użytkownika. Zebranie zdjęć nowych klas obiektów umożliwia tworzenie zbioru danych, który jest wykorzystywany do kolejnej iteracji treningu sieci neuronowej. W ten sposób zwiększa się liczba obiektów, które system potrafi znajdywać, a dodatkowo również zwiększa się dokładność działania systemu.

Najistotniejszym elementem stworzonej aplikacji mobilnej dla Android/iOS jest algorytm doprecyzowujący lokalizację widocznego w kamerze obiektu, który został znaleziony przez sieć neuronową . Wskazany algorytm bazuje na rezultatach badania głębi obrazu (w celu określenia relacji pomiędzy widocznymi obiektami), wysokości i szerokości ramki detekcji (ang. bounding box) wykry-

Precyzja danej długości geograficznej (format WGS84 i decymalny)		Odległość (pomijająca nieregularność powierzchni ziemi) pomiędzy sąsiednimi długościami geograficznymi (o wskazanej precyzji) wzdłuż równoleżnika w zależności od szerokości geograficznej [m]				
DD.dddddd°	**DD°MM'SS.sss°**	**0 ° (równik)**	**30°**	**45°**	**60°**	**75°**
1°	1°	111 120	96 233	78 574	55 560	28 760
0.1°	0°06'	11 112	9 623	7 857	5 556	2 876
0.01°	0°00'36"	1 111.2	962	785	555	288
0.001°	0°00'03.6"	111.12	96	78	55	29
0.000 1°	0°00'00.36"	11.112	9.6	7.8	5.5	2.9
0.000 01°	0°00'00.036"	1.111 2	0.96	0.78	0.55	0.29
0.000 001°	0°00'00.0036"	0.111 12	0.096	0.078	0.055	0.028

Tabela 3.1: Tabela przedstawia różnice w precyzji wyznaczanej lokalizacji, która wynika ze zróżnicowanej odległości pomiędzy południkami w różnych szerokościach geograficznych.

tego obiektu, średniej rzeczywistej wysokości i szerokości obiektu danej klasy (będącej wartością średnią), danych z modułu GPS (lokalizacji i siły sygnału, która bezpośrednio rzutuje na uzyskiwaną dokładność) i żyroskopu (określającego w którym kierunku jest zwrócony użytkownik i w jakiej orientacji względem płaszczyzny znajduje się jego urządzenie). Wszystkie te informacje są agregowane przez algorytm w celu określenia dokładnej lokalizacji obiektów, by w kolejnym kroku przesłać je na serwer, gdzie zostaną porównane z rezultatami przesłanymi przez innych użytkowników.

Dane pochodzące z modułu GPS również są wykorzystywane w procesie doprecyzowania dokładnej lokalizacji obiektów. Dokładność danych o lokalizacji dostarczanych przez moduł GPS zależy od szerokości geograficznej (ϕ) w której znajduje się urządzenie. Odległość pomiędzy południkami zależy od szerokości geograficznej i dla równika wynosi 111km, a dla równoleżnika 30°96km. Im wyższa szerokość geograficzna, tym południki są bliżej siebie np. w Niemczech (50°N) odległość pomiędzy południkami są o 1/3 mniejsze niż na równiku, a dokładność jest odpowiednio wyższa jak pokazano w tabeli 3.1. Informacja o możliwym błędzie danych dostarczanych przez moduł jest szczególnie istotna na etapie scalania redundantnych obiektów po stronie serwera. Dzięki znajomości możliwych odchyleń można stosować inną odległość dla której należy przeprowadzić proces przeszukiwania obiektów tej samej klasy celem ich scalenia.

W niektórych urządzeniach funkcja określenia błędu lokalizacji stanowi na-

tywny element systemu (Android/iOS). Moduł GPS urządzenia zwraca informacje o aktualnej lokalizacji w postaci np. λ= 22.020065218828933,
ϕ= 49.684429025371756. Oznacza to, że dokładność pozycji zwracanej przez moduł jest określana z precyzją przewyższającą 1cm (precyzja decymalna 10^{-6} to dokładność około 1cm dla λ Polski). Dodatkowo, istnieje możliwość pobrania precyzji, wyrażonej w metrach, jaką oferuje moduł w danej chwili (który mapuje aktualną siłę sygnału na daną wartość liczbową określającą margines błędu dla jego aktualnej pozycji).

Wykonano badania mające na celu oszacowanie marginesu błędu zwracanego przez moduł GPS w różnych środowiskach różniących się jakością sygnału (centrum miasta, przedmieścia, tereny pozamiejskie i las). Pomiary zostały przeprowadzane na terenie kilku lokalizacji. Badanie polegało na zainstalowaniu zaimplementowanej aplikacji, która na żądanie zwracała położenie urządzenia mobilnego na podstawie jego sygnału GPS, pobierając przy tym informację o aktualnym marginesie błędu. Następnie porównano te informacje z pomiarami referencyjnymi wykonanymi przy użyciu wysoce precyzyjnego geodezyjnego odbiornika GPS (Leica 1200 GPS). Wyniki cechują się dużą rozbieżnością, jednak margines błędu pochodzący z modułu GPS często był wyższy niż rzeczywisty błąd pomiaru, zwłaszcza na terenie miast. Jakość sygnału, a tym samym precyzja lokalizacji zależy od wielu czynników tj. materiału pobliskich budynków, poziomu zakłóceń pochodzących od innych urządzeń elektrycznych znajdujących się w pobliżu telefonu, aktualnej pogody czy odbicia fal od obiektów otaczających użytkownika. Również sam model modułu GPS w który wyposażony był telefon rzutował na dokładność lokalizacji. Pomiar lokalizacji GPSa smartfona był wykonywany co kilka sekund od uruchomienia zbudowanej aplikacji logującej lokalizację oraz margines błędu pomiarowego zwracanego przez sam moduł. Średnią z wykonanych (około 100) pomiarów pokazano w tabeli 3.2. W większości przypadków różnica pomiędzy wskazywaną lokalizacją, a wartością referencyjną pozyskaną z urządzenia Leica była niższa niż margines błędu dostarczanego z modułu GPS.

Rezultaty badań pokazały, że średnia odległość od wskazywanej przez mo-

Miasto	Leica 1200 GPS	Samsung A40		Samsung Galaxy S6	
		Margines błędu [m]	Różnica [m]	Margines błędu [m]	Różnica [m]
Katowice	50.25960137083932, 19.015848732869344	7,5	5	9	6,5
	50.25728404602706, 19.016505909784193	8,5	5,5	10	7
	50.25969817777004, 19.014285889730196	9,5	5,5	11	10
	50.255640191135065, 19.01481081683188	13	10	17,5	14,5
	50.25978715535597, 19.01372452342917	8	6	9	10
	50.27223161729429, 18.991788414302924	11	12	14	13,5
Gliwice	50.29688472598168, 18.668597814319458	9	7,5	11,5	6
	50.2985093584243, 18.670166447365943	12	10	12	4,5
	50.29421354355806, 18.665707717218215	11,5	6,5	13,5	11,5
Piekary Śl	50.390769881252396, 18.93053465476909	7,5	5,5	8	6,5
	50.37358076373249, 18.943691051061826	9	6,5	7,5	3,5
Klęczany	49.66863055537982, 20.622138678505415	18	21	25	22
	49.66670378265561, 20.625310675974067	25	20	19,5	14
	49.667385918786835, 20.624998763776485	33,5	39,5	40	31,5

Tabela 3.2: Rezultat pomiarów wykonanych przy użyciu specjalistycznego sprzętu geodezyjnego Leica (dostarczającego lokalizacji z dokładnością do 1cm) oraz pomiarów wykonanych przez aplikację zapisującą aktualną pozycję urządzenia (dla dwóch urządzeń tj. Samsung A40 oraz Galaxy S6). Wyniki zostały podane z dokładnością do 0.5m

duł GPS lokalizacji jest najczęściej o około 20% niższa niż wskazany margines precyzji pomiaru. W projekcie założono więc, że lokalizacja obiektów przesłanych od użytkownika znajduje się w kole o promieniu 0.8 wartości wskazanego podczas pomiaru marginesu błędu.

Kolejną z informacji przetwarzanych przez algorytm doprecyzowania lokalizacji obiektów jest rozmiar ramek ograniczających (bounding boxów) będących rezultatem działania sieci neuronowej realizującej detekcję obiektów. Wskazany parametr (rozmiar ramek) jest wykorzystywany podczas obliczania estymacji odległości według wzoru 3.1. Dobór topologii sieci neuronowej jest wynikiem badań jakie zostały przeprowadzone w celu wyłonienia sieci pracującej najszybciej spośród ogólnie dostępnych. Poza szybkością, podczas badań brano również pod uwagę dokładność uzyskiwanych detekcji (ocenianą poprzez porównanie wyliczanego mAP 2.2.1 dla każdego z modeli). Z uwagi na fakt, że detekcja obiektów stanowiła jeden z podstawowych elementów całego projektu, prac nad wyborem sieci zostały wykonana w początkowej fazie realizacji projektu.

Jednym z założeń projektu było wykorzystanie sieci, której uruchomienie będzie możliwe na szerokiej gamie urządzeń (opartych na architekturze ARMv8-A lub nowszej). Pierwszą fazą doboru sieci było przeprowadzenie badań nad do-

stępnymi frameworkami, które dostarczają rozwiązania dedykowane dla urządzeń mobilnych charakteryzujących się ograniczonymi zasobami. Spośród oczywistych wyborów takich jak modele pochodzące z TensorFlow Model Zoo oraz PyTorch Model Zoo, zweryfikowano i porównano skuteczność innych frameworków np. YOLO. W testowanej wersji PyTorcha (v1.3.1, release z listopada 2019 roku) wykryto problem z uruchomieniem sieci na urządzeniach mobilnych poza dwoma (ResNet18 i MobileNetv1), które zostały udostępnione do testów przez samych twórców PyTorch. Wskazana biblioteka działa prawidłowo w środowisku serwerowym, zarówno na Windows oraz Linux, jednak wsparcie dla urządzeń mobilnych w tym okresie było znikome co wykluczyło jej wykorzystanie w dalszej części projektu. TensorFlow w wersji Lite stanowi, po dziś dzień, jeden z najpopularniejszych frameworków do budowy modeli dedykowanych dla urządzeń mobilnych. Jego ogromne wsparcie, a także uniwersalność (działa zarówno dla urządzeń z iOS, Androidem jak i na mikrokontrolerach STM32) spowodowała, że został wybrany jako narzędzie do dalszych prac.

Pierwszym z badanych elementów wykorzystania sieci neuronowych do detekcji obiektów dla urządzeń mobilnych była szybkość przetwarzania pojedynczej klatki obrazu. Coraz więcej współczesnych urządzeń mobilnych posiada sprzętowe wsparcie dla sieci neuronowych w postaci dedykowanego układu znacznie przyśpieszającego wykonywanie operacji na macierzach oraz innych obliczeń jakie są wykonywane podczas przetwarzania obrazu przez sieci. W ostatnich latach obserwuje się wzrost popularności tych układów w nowych telefonach co wskazuje, że w najbliższych latach staną się one standardem takim jak aparaty czy moduły GPS, które dawniej były wyposażeniem dostępnym wyłącznie w modelach z najwyższej półki. Część smartfonów jest również wyposażonych w dedykowane GPU o niewielkich zasobach, by umożliwić uruchomienie wymagających gier lub lokalnej obróbki zdjęć. W ramach badań weryfikowano szybkość sieci przetwarzających dane na smartfonach, które są wyposażone w GPU oraz posiadających wyłącznie CPU. Zaimplementowana aplikacja zapewnia pełną elastyczność i uruchamia model na GPU jeśli wykryje, że telefon je posiada i dostępne zasoby na to pozwalają. Modele pracujące bezpośrednio na GPU nie

wymagają zmiany precyzji wag połączeń sieci z uwagi na fakt, że jednostki graficzne przetwarzają domyślnie liczby zmiennoprzecinkowe. Uruchomienie modeli bez potrzeby kwantyzacji ma szereg istotnych zalet. Przede wszystkim, możliwe jest uruchomienie modeli na smartfonach w takim samym formacie, w jakim zostały pierwotnie wytrenowane na serwerze przy użyciu standardowych bibliotek uczenia maszynowego. Dodatkowo, pozbawiona konwersji sieć uzyskuje taką samą dokładność jak w środowisku desktopowym lub serwerowym. Wykorzystanie GPU sprawia także, że czas potrzebny na przetworzenie pojedynczej klatki obrazu ulega istotnemu zmniejszeniu, a dodatkowo cały proces nie obciąża CPU. W tabeli 3.3 pokazano prędkość jaką osiągały wytrenowane modele na różnych urządzeniach mobilnych. Sprawdzono w ten sposób stabilność uzyskiwanych wyników na różnych modelach telefonów, a także prędkość jaką sieci neuronowe bez kwantyzacji osiągają na smartfonach nie posiadających GPU. Najszybszą z testowanych topologii sieci okazała się modiledet_cpu. Z uwagi na fakt, że jest to stosunkowo nowa sieć (stała się częścią TensorFlow ZOO w 2020 roku), postanowiono poszerzyć badania stabilności oraz czasu jej wykonania również na starszych urządzeniach mobilnych. Na smartfonie Samsung Galaxy S6, którego premiera miała miejsce w I kwartale 2015 roku wybrana sieć przetwarzała obraz w czasie przekraczającym 2 klatki na sekundę wykorzystując przy tym jedynie jeden rdzeń procesora urządzenia mobilnego co stanowiło satysfakcjonujący rezultat. Przy pełnym obciążeniu CPU telefon uzyskał ponad 3 FPS.

Dla urządzeń nie posiadających GPU przyśpieszenie pracy modeli osiąga się poprzez kwantyzację. Jest to proces polegający na zmniejszeniu precyzji wag jakie znajdują się wewnątrz sieci. Model jest konwertowany z 32-bitowego typu zmiennoprzecinkowego do formatu 16-bitowego lub całkowitego int-8. Zmniejsza to dwu lub czterokrotnie jego rozmiar, nieznacznie pogarsza jakość detekcji [67], wymagana jest mniejsza ilość pamięci RAM do uruchomienia oraz przyspiesza jego wykonanie kilkukrotnie (jest to zależne od rodzaju porównywanego CPU i GPU) [67]. Dodatkowo obliczenia na liczbach całkowitych zużywają mniej energii, co jest krytyczne w przypadku smartfonów i mikrokontrolerów dedykowanych do pracy na baterii. Wykorzystywana w środowisku produkcyjnym

aplikacja posiada możliwość zmniejszenia czasu pomiędzy pobraniem i przetworzeniem klatki obrazu z aparatu oraz zmiany rodzaju modelu na dedykowany do pracy na CPU jeśli pierwotny został uruchomiony na GPU co wydłuża czas pracy na baterii.

Do testów zostały wybrane sieci o topologiach pochodzących ze zbioru TensorFlow Zoo. Uruchomiono jedynie 5 topologii wykazujących najwyższą prędkość według oficjalnej dokumentacji g3doc [58] dla biblioteki TensorFlow. Celem eksperymentu było określenie średniej szybkości przetwarzania pojedynczej klatki obrazu o rozdzielczości 300x300 px. Na niektórych modelach przeprowadzono proces kwantyzacji polegający na zmniejszeniu precyzji wag wykorzystywanych wewnątrz sieci z float32 na int8. Standardowa procedura testów obejmowała instalację aplikacji na 3 modelach telefonów, której jedynym zadaniem było przechwytywanie obrazu z kamery telefonu oraz rejestracji czasu niezbędnego do jego przetworzenia przez sieć. Tabela 3.3 zawiera średnie czasu przetwarzania określone dla tysiąca przechwyconych obrazów z kamery. Sieć była uruchamiana kolejno na 1,2,3 i 4 wątkach CPU. Po wyselekcjonowaniu najszybszej topologii (mobiledet cpu) rozszerzono testy dla dodatkowych 3 urządzeń, by sprawdzić stabilność działania sieci i jej wsparcie dla większej gamy urządzeń przed podjęciem decyzji o jej wykorzystaniu w docelowym rozwiązaniu.

Architektura	**Rozmiar zdjęcia**	**Kwantyzowany**	**Smartfon**	**Czas przetwarzania 1 klatki obrazu (w ms- rozróżnienie na ilość wątków użytych podczas obliczeń)**				**Kwartyle czasu przetwarzania 1 klatki obrazu (dla 4 wątków)**			**Odchylenie standardowe czasu przetwarzania 1 klatki obrazu (dla 4 wątków)**
				1	**2**	**3**	**4**	**1/4**	**1/2**	**3/4**	
mobilenet oid	300x300	Nie	Pocophone F1	504	308	245	217	211	216	219	17
			Samsung A40	498	322	318	319	315	318	319	13
			iPhone XR	76	49	56	56	54	57	58	10
mobilenet mnasfpn	320x320	Nie	Pocophone F1	560	447	398	385	377	382	381	21
			Samsung A40	717	518	521	508	506	509	511	11
			iPhone XR	114	91	101	104	100	101	105	11
mobiledet dsp fp32	320x320	Nie	Pocophone F1	580	372	368	345	344	345	347	22
			Samsung A40	781	443	435	429	412	418	420	13
			iPhone XR	146	92	103	108	105	107	109	10
mobilenet ssd	300x300	Tak	Pocophone F1	236	221	200	203	199	201	206	13
			Samsung A40	349	256	260	263	261	262	263	17
			iPhone XR	64	47	67	93	91	93	97	9
mobiledet cpu	300x300	Tak	Pocophone F1	126	101	94	89	85	88	95	16
			Samsung A40	304	219	214	215	211	215	217	11
			iPhone XR	50	48	44	38	36	39	40	13
			Galaxy S6	470	294	290	291	285	292	299	16
			iPhone 6S	190	124	126	125	122	123	126	14
			iPhone 11 Pro	14	13	11	8	8	8	9	5

Tabela 3.3: Tabela przedstawia rezultaty przeprowadzonego eksperymentu polegającego na uruchomieniu ogólnie dostępnych, wytrenowanych na zbiorze COCO, topologii sieci na kilku urządzeniach mobilnych. Wskazany w tabeli czas jest średnią z tysiąca detekcji wykonanych na każdym z urządzeń.

W celu potwierdzenia trzeciej tezy niniejszej pracy, dotyczącej prędkości przetwarzania obrazu na urządzeniach o architekturze ARMv8-A (i nowszych) przeprowadzono testy statystyczne na bazie próby losowej. Weryfikacja normalności rozkładu czasu przetworzenia 1000 klatek z kamery na każdym z badanych urządzeń, przy użyciu testu Shapiro-Wilka, wykazała, że dane nie pochodzą z rozkładu normalnego. Z uwagi na powyższe zastosowano parametryczny test istotności dla wartości oczekiwanej w przypadku, gdy cecha ma dowolny rozkład o nieznanej ale skończonej wariancji i przy dużej próbie losowej (jej liczebność większa niż 100)[79].

Przyjęto następującą postać hipotez zerowej H0 i alternatywnej H1:

H_0: E(t)=m0=333ms (wartość oczekiwana rozkładu t czasu przetwarzania prędkości głębokich sieci neuronowych działających na urządzeniach mobilnych opartych o architekturę ARMv8-A i nowszych **wynosi** 333ms co odpowiada przetwarzaniu z prędkością 3 klatek na sekundę).

H_1: E(t)=m0<333ms (wartość oczekiwana rozkładu t czasu przetwarzania prędkości głębokich sieci neuronowych działających na urządzeniach mobilnych opartych o architekturę ARMv8-A i nowszych **jest mniejsza** niż 333ms co oznacza

prędkość większą niż 3 klatki na sekundę).

Następnie policzono statystykę U:

$U = \frac{\overline{X} - m_0}{S/\sqrt{n}}$

$\overline{X}$ - średnia arytmetyczna

m_0 - zakładana prędkość w milisekundach

S - odchylenie standardowe

n - liczba pomiarów

Gdzie:

$\overline{X} = 158.923$

$m_0 = 333$

$S = 73.9317289540596$

$n = 4000$

$U = -148.91571346618807$

W związku z przyjętą postacią hipotezy alternatywnej zbiór krytyczny ma postać:

$(-\infty, -u_{(1-\alpha)}\rangle$

tj. $(-\infty, -1,6449\rangle$

Gdzie u to kwantyl rzędu 1-alfa rozkładu normalnego, a alfa-0.05 to poziom istotności testu.

Ponieważ statystyka testowa ma wartość spoza obszaru krytycznego H_0 została odrzucona uznając H_1 za prawdziwą.Uzyskana wartość statystyki testowej wskazuje, że **wartość oczekiwana czasu przetwarzania prędkości głębokich sieci neuronowych** jest niższy niż określony w hipotezie. Średnia uzyskanych wyników wynosiła 158.923 ms z odchyleniem standardowym 73.932. Uzyskany rezultat wskazuje, że prędkość przetwarzania obrazu jest przeciętnie wyższa niż zakładana tj. 3 FPS co potwierdza tezę określoną we wstępie.

Dodatkową metodą na przetworzenia danego obrazu w możliwie najkrótszym czasie jest ograniczenie liczby klas wynikowych. Taki zabieg, dzięki zmniejszeniu liczby połączeń wewnątrz sieci, pozwala przyśpieszyć proces wnioskowania. Ponadto, czas potrzebny na interpretację wyników tj. pobranie wartości ostatniej

Architektura	**Liczba klas**	**Rozmiar zdjęcia**	**Kwantyzowany**	**Smartfon**	**Średni czas przetwarzania 1 klatki strumienia wideo [ms] (rozróżnienie na ilość wątków użytych podczas obliczeń)**			
					1	2	3	4
mobilenet_ssd	1005	300x300	Tak	Pocophone F1	444	404	376	371
				Samsung A40	600	467	458	461
				iPhone XR	110	86	99	110
mobilenet_ssd	5	300x300	Tak	Pocophone F1	236	221	200	203
				Samsung A40	349	256	260	263
				iPhone XR	64	47	67	93
YoloV4 Tiny	2000	416x416	Nie	Poco M3 Pro 5G	1281	1031	994	827
YoloV4 Tiny	80	416x416	Nie	Poco M3 Pro 5G	541	435	420	385
YoloV4 Tiny	3	416x416	Nie	Poco M3 Pro 5G	487	395	384	348

Tabela 3.4: Tabela przedstawiająca rezultaty przeprowadzonych testów polegających na wykorzystaniu procesu transfer learning do wytrenowania sieci do detekcji różnej liczby klas wraz z czasem potrzebnym do uzyskania wyników. Prezentowany w tabeli czas jest średnią arytmetyczną przetworzenia 1000 klatek obrazu.

warstwy i jej interpretacja, ulegają zmniejszeniu. Wykonano testy polegające na wykorzystaniu procesu transfer learning do wytrenowania sieci MobileNet SSD do detekcji wyłącznie pięciu klas obiektów oraz dla tysiąca pięciu, a także YoloV4 w wersji Tiny dla trzech, osiemdziesięciu i dwóch tysięcy klas (posiłkując się danymi z OpenImages). Po wytrenowaniu, sieci zostały kolejno uruchomione na kilku smartfonach przy użyciu zbudowanej aplikacji posiadającej funkcję do mierzenia i zapisywania czasu potrzebnego do przetworzenia pojedynczej klatki obrazu pochodzącej z widoku kamery. Zweryfikowano czas przy wykorzystaniu 1,2,3 i 4 wątków CPU. Rezultaty wykonanych testów 3.4 wskazują, że liczba klas wynikowych (determinująca rozmiar ostatniej warstwy sieci i liczbę filtrów wewnątrz) ma znaczny wpływ na czas przetwarzania obrazu przez sieć wydłużając go nawet kilkukrotnie dla kilku tysięcy klas w stosunku do tej samej topologii dla liczby klas mniejszej niż dziesięć.

W celu określenia odległości obiektu od ogniskowej aparatu, poza rezultatami pracy sieci w postaci ramki ograniczającej (ang. bounding box), rozmiarami obiektu w rzeczywistości jest również wymagana znajomość parametrów wewnętrznych kamery. Zarówno system Android jak i iOS posiadają funkcję zwracającą ogniskową oraz rozdzielczość aktualnie wykorzystywanego aparatu. Rdzeń (ang. core) systemu dostarcza wysokość i szerokość obiektu w rzeczy-

wistości (ustawionej na sztywno jako reprezentatywna wartość dla danej klasy obiektów). Posiadając wyżej wymienione dane algorytm wylicza orientacyjną odległość obiektu od kamery 3.2 wykorzystując wzór:

$$D = \frac{f \cdot H \cdot h}{o \cdot s}, \tag{3.1}$$

D – rzeczywisty dystans do obiektu [m]

f – ogniskowa [mm]

H – rzeczywista wysokość (albo szerokość) obiektu [m]

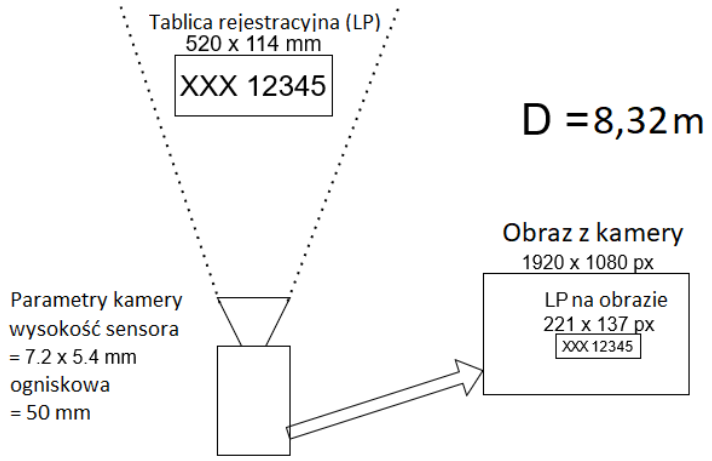
h – wysokość (albo szerokość) obrazu [px]

o – wysokość (albo szerokość) obiektu na obrazie [px]

s – wysokość (albo szerokość) sensora [mm]

Mechanizm działa prawidłowo jeśli obiekt znajduje się w centralnej części widoku kamery i cały mieści się w jej polu widzenia. Jeśli na zdjęciu widoczny jest jedynie fragment obiektu (np. tułów człowieka) następuje przetworzenie obrazu przez algorytm badania głębi obrazu. Bazując na informacjach czy dany obiekt jest bliżej lub dalej od innych obiektów do których można prawidłowo określić odległość, następuje estymacja odległości do wybranego obiektu bazując na gradiencie kolorów rezultatu algorytmu głębi. Znając dystans do minimum dwóch obiektów widocznych w widoku kamery oraz kolor (0-255) piksela znajdującego się w centralnym punkcie każdej detekcji można określić odległość do pozostałych obiektów.

Podczas badań weryfikowano działania kilku algorytmów głębi opisanych w przeglądzie literatury (m.in. PackNet-SfM, MonoDepth, FastDepth). Większość z nich nie zezwala na darmowe wykorzystanie do celów komercyjnych, a pozostałe nie są stworzone do pracy na urządzeniach o niewielkich zasobach. Istnieje jednak gotowe rozwiązanie, które spełnia powyższe warunki. Ma satysfakcjonujące rezultaty, a dodatkowo jest możliwe jego wykorzystanie zarówno dla urządzeń z Androidem, iOS jak i w systemach czasu rzeczywistego (mikrokontrolerach STM32). FastDepth stanowi rozwiązanie oparte na architekturze enkoder-dekoder, która została poddana procesowi przycinania (ang. network



Rysunek 3.2: Przykład działania modułu estymacji odległości dla sieci wytrenowanej dla detekcji tablic rejestracyjnych. Znając parametry kamery oraz rzeczywiste wymiary fotografowanego obiektu można z dużą dokładnością określić odległość w jakiej obiekt znajduje się od kamery.

pruning) na etapie treningu. Zmniejszyło to jej złożoność obliczeniową i czas niezbędny do przetworzenia pojedynczej klatki obrazu. Uzyskano wynik zbliżony do publikowanego przez autorów tj. na iPhone X sieć przetwarzała obraz z prędkością około 40 FPS. Wskazany algorytm jest wykorzystywany wyłącznie, gdy na obrazie są widoczne minimum 3 obiekty z których ramka ograniczająca (ang. bounding box) jednego z nich przylega do krawędzi obrazu (tylko fragment obiektu jest widoczny w kamerze).

3.2.2 Przetwarzanie danych po stronie urządzeń posiadających system czasu rzeczywistego

Informacje pochodzące ze smartfonów stanowią docelowo główne źródło danych zasilających system, jednak w jego początkowej fazie postanowiono wykorzystać aktualną infrastrukturę miasta taką jak kamery na skrzyżowaniach oraz kamery CCTV zainstalowane w parkach i miejscach użyteczności publicznej. Pobieranie danych z tych kamer wymaga uprzedniej ręcznej konfiguracji polegającej na uzupełnieniu kierunku w których zwrócona jest kamera (w formie kąta liczonego od kierunku północnego (będącego wartością 0°), jej położenia na mapie

w formie koordynat oraz określeniu ogniskowej i wielkość sensora. Dwie ostatnie informacje są dostępne w specyfikacji dostarczanej od producenta. Dla kamer obrotowych kierunek w którym jest zwrócona jest aktualizowany na bieżąco pobierając z API kamery jej aktualne położenie. Informacje o położeniu kamer firmy HIK (najczęściej wykorzystywana firma kamer w ITS Wrocław) są zwracane w formacie JSON i przechowują wartości dotyczące dokładnego kąta pod jakim patrzy kamera w pozycji wertykalnej i horyzontalnej. W celu ułatwienia konfiguracji dla kamer obrotowych, użytkownik musi ustawić kamerę, by byłą zwrócona na północ, a następnie wywołać w systemie procedurę odpytania kamery o jej aktualną pozycję i zapisanie jej położenie jako wartość 0°w systemie.

Wykorzystywane na mieście Wrocław kamery nie posiadały wystarczających zasobów, by osadzić na nich zarówno sieć neuronową do detekcji obiektów jak również sieć do estymacji głębi obrazu. Dodatkowo, producenci innych dostępnych kamer często blokują zasoby po osadzeniu na nich własnych algorytmów i usług, nie udostępniając pozostałego miejsca do użytku podmiotom zewnętrznym. Z tego powodu należało opracować moduł pracujący na przemysłowym urządzeniu (posiadającym zwiększoną wytrzymałość na wysokie i niskie temperatury), które będzie można zamontować w bezpośrednim otoczeniu kamery i móc przetwarzać z niej obraz na bieżąco. W tym celu zdecydowano się na przeprowadzenie badań nad możliwościami osadzenia i uruchomienia sieci neuronowych na mikrokontrolerach (mają wysoką tolerancję na wysokie temperatury, które czasem przekraczają 70°w szafach na mieście, a także cechują się dużą niezawodnością co redukuje liczbę akcji serwisowych).

W pierwszym etapie badań zdecydowano się na wykorzystanie Nvidia Jetson Nano jako tanią jednostkę wyposażoną w GPU, która posiada wszystkie niezbędne elementy do pobrania strumienia z kamery po RTSP i jego przetworzenia. Przeprowadzono testy polegające na uruchomieniu oprogramowania realizującego detekcję i po przetworzeniu 1000 obrazów zweryfikowano czas potrzebny na przetworzenie pojedynczej klatki 3.5. Podczas testów kilku sieci (m.in. YOLO, MobileNeta) stworzono rozwiązanie, które może przetwarzać obraz w zakresie detekcji obiektów i wyznaczania głębi obrazu dla 8 kamer równolegle uzyskując

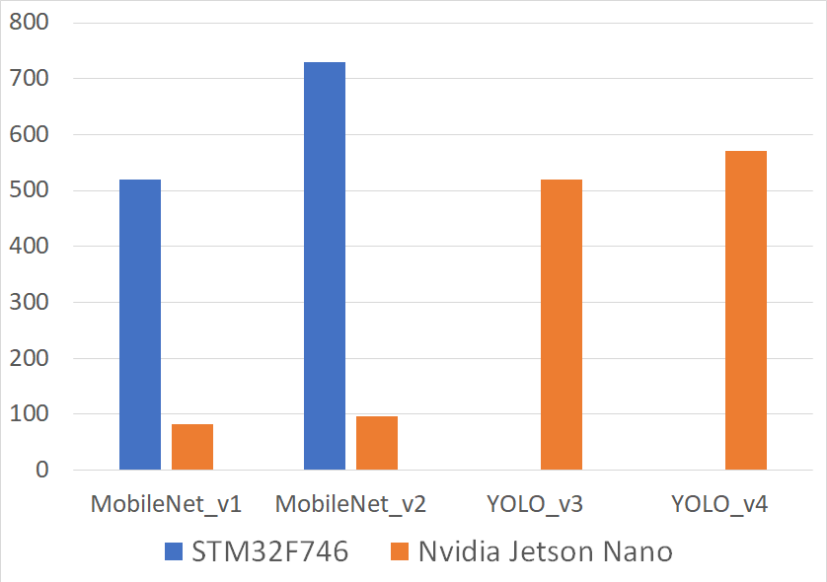
Architektura	NVIDIA Jetson Nano		DISCOVERY STM32F746	
	Średnia [ms]	Odchylenie standardowe	Średnia [ms]	Odchylenie standardowe
MobileNet_v1	88	4,472	518	0.002
MobileNet_v2	97	4,899	726	0.005
YOLO_v3	511	7,071	-	-
YOLO_v4	574	8,246	-	-

Tabela 3.5: Tabela przedstawia szybkość przetwarzania pojedynczej klatki strumienia dla różnych sieci neuronowych realizujących detekcję obiektów w obrazie . Ze względu na niewystarczającą ilość zasobów na STM nie udało się uruchomić topologii YOLO. Prezentowany czas jest średnią arytmetyczną z przetworzenia 1000 klatek obrazu.

dla każdej co najmniej 1 FPS. Podczas badań próbowano również uruchomić wskazane modele na tańszych urządzeniach tj. mikrokontrolerach STM32. Wygenerowane modele TFLite po kwantyzacji, pracujące na smartfonach bez GPU zostały przekonwertowane przy użyciu narzędzia X-CUBE-AI dedykowane do mikrokontrolerów STM32 i uruchomione na płytce DISCOVERY STM32F746. Niestety uzyskana prędkość była znacznie niższa niż w rezultatach Jeston Nano (mimo niższej ceny płytki STM stosunek liczby obsługiwanych kamer do ceny był wyższy). Ze względu na niewielkie zasoby (1MB pamięci Flash) sieci YOLO nie udało się uruchomić na wskazanym mikrokontrolerze, a prędkosć Mobilenet była znacznie wyższa niż na Nvidia stąd zaniechano dalszych badań nad przetwarzaniem obrazu przy użyciu STM i w docelowym rozwiązaniu wykorzystano Nvidia Jetson Nano 3.3.

3.3 Warstwa synchronizacji danych pozyskanych z urządzeń brzegowych

Urządzenia brzegowe pozyskują dane a następnie przesyłają je nieustannie na serwer w celu agregacji, synchronizacji i zapisania w bazie danych umożliwiając ich późniejsze przetwarzanie. Zapewnienie aktualizacji mapy w czasie rzeczywistym wymagało przeprowadzenia szeregu badań dotyczących doboru narzędzi umożliwiających szybkie odebranie i przetwarzanie danych przez serwer. Celem



Rysunek 3.3: Porównanie czasu potrzebnego na przetworzenie 1 klatki obrazu o rozdzielczości 300x300 px z wykorzystaniem kilku topologii sieci uruchomionych na Nvidia Jetson Nano i DISCOVERY STM32F746. Oś Y - czas podany w ms. Ze względu na rozmiar sieci YOLOv3 i YOLOv4 nie udało się ich uruchomić na wskazanym mikrokontrolerze (ze względu na niewystarczającą ilość zasobów w urządzeniu)

podjętych prac było zminimalizowanie czasu jaki musi upłynąć od chwili odebrania informacji o lokalizacji do umieszczenia go na wizualizacji użytkownika, będącego widokiem mapy 2D.

3.3.1 Odbiór danych przez serwer. Maksymalizacji szybkości obsługi nadsyłanych zapytań

W celu zapewnienia niskich opóźnień w odbiorze danych, w pierwszej kolejności przeprowadzono testy szybkości serwera odbierającego dane o lokalizacji obiektów znalezionych przez urządzenia brzegowe. Porównywane serwery zostały zaimplementowane jako HTTP REST API oraz serwer oparty na socketach. Testy przeprowadzano z wykorzystaniem oprogramowania Apache JMeter. Serwer osadzono na wirtualnej maszynie z systemem operacyjnym Linux Ubuntu z 2GB pamięci RAM oraz 1 rdzeniem procesora i7-10510U. Na rysunkach 3.4 pokazano raport z testów zawierający czas w milisekundach jaki upłynął od wysłania wiadomości (z 1000 wątków) do serwera i uzyskania od niego prawidłowej odpowiedzi (potwierdzenia odebrania danych i przekazania ich do dalszego przetworzenia). W przypadku socketów pierwsze odpowiedzi zostały odebrane po 4ms, a ostatnie po czasie 1252 ms. Średnia prędkość obsługi zapytania HTTP wynosiła 6ms, natomiast średnia prędkość obsługi zapytania TCP niespełna 2ms. Do uzyskania wskazanej prędkości wykorzystano framework dragon dla połączeń http oraz evpp dla socketów.

Evpp w ogólnie dostępnych porównaniach z innymi popularnymi bibliotekami takimi jak Boost Asio, libevent jest wydajniejszy od kilku, do nawet kilkudziesięciu procent w zależności od badanych parametrów, takich jak szybkość obsługi połączenia czy odbiór dużych plików [11]. Cały serwer jest napisany w języku C++, by wyeliminować opóźnienia wynikające z konieczności pracy interpreterów, a jego głównym zadaniem jest odbiór oraz kolejkowanie danych przeznaczonych do dalszej analizy i przekazywanie rezultatów do wizualizacji.

Poza szybkością pojedynczej instancji serwera, zapewniono jego pełną skalowalność, niezbędną w przypadku rosnącego zainteresowania i wzrostu liczby danych napływających do systemu. Zaimplementowany serwer osadzono na konte-

Sample #	Start Time	Thread Name	Label	Sample Tim... ↑	Status
1	12:15:03.222	Thread Group...	TCP Sampler	4	
3	12:15:03.222	Thread Group...	TCP Sampler	4	
59	12:15:03.311	Thread Group...	TCP Sampler	4	
72	12:15:03.329	Thread Group...	TCP Sampler	4	
2	12:15:03.221	Thread Group...	TCP Sampler	5	

Sample #	Start Time	Thread Name	Label	Sample Tim... ↓	Status
974	12:15:04.089	Thread Group...	TCP Sampler	1252	
975	12:15:04.090	Thread Group...	TCP Sampler	1252	
976	12:15:04.090	Thread Group...	TCP Sampler	1252	
973	12:15:04.089	Thread Group...	TCP Sampler	1251	
971	12:15:04.089	Thread Group...	TCP Sampler	1250	
972	12:15:04.089	Thread Group...	TCP Sampler	1250	

Rysunek 3.4: Szybkość serwera TCP zmierzona przy użyciu oprogramowania JMeter. Rysunki opracowano na podstawie wygenerowanego raportu. Na pierwszym rysunku posortowano czas uzyskania odpowiedzi rosnąco, a na kolejnym malejąco.

nerze Dockerowym. Umożliwia to uruchomienie wielu jego instancji oraz ich wirtualizowanie poprzez pojedynczy punkt końcowy (ang. endpoint) z osadzonym modułem równoważenia obciążenia (ang. load balancer). Ponadto wykorzystane rozwiązanie gwarantuje niezawodność i nieprzerwane działanie w przypadku awarii pojedynczej instancji serwera. W utworzonym środowisku przetestowano kilka współdziałających ze sobą kontenerów uzyskując liniowy wzrost możliwych do obsłużenia żądań. W przeprowadzonych testach potwierdzono, że pojedynczy kontener jest w stanie obsłużyć i utrzymywać do 10 tysięcy równoczesnych połączeń od urządzeń brzegowych.

W docelowym systemie zapewniono również bezpieczeństwo informacji przekazywanych do systemu. Z uwagi na fakt dynamicznych zmian w prawie jakie miały miejsce w ostatnich latach wdrożono metodę szyfrowania AES-256, aby zapewnić bezpieczeństwo danych jakie są wysyłane przez użytkowników. Pomimo braku wskazań przez ustawodawcę (brak przetwarzania danych sensytywnych, brak możliwości zidentyfikowania personaliów osoby przesyłającej), taki zabieg był konieczny, po to aby użytkownicy chętniej dzielili się danymi bez obaw, że ktoś niepowołany będzie mógł w relatywnie łatwy sposób je przechwycić. Dodatkowo, wykorzystany do budowy serwera framework evpp umożliwia zastosowanie tunelowania danych przez SSL, gdyby wskazane szyfrowanie okazało się w przyszłości niewystarczające.

3.3.2 Szybkość zapisu/odczytu z baz danych noSQL

Poza szybkością serwera, który odbiera strumieniowo napływające dane równie istotnym elementem mającym bezpośredni wpływ na wydajność całego systemu jest prędkość zapisu i odczytu rekordów bazy danych. Stanowi ona element kluczowy do tworzenia dynamicznej mapy otoczenia i największą wartość dodaną w stosunku do innych rozwiązań obecnych na rynku. Baza danych będzie przechowywała archiwalne położenie obiektów wykrytych w danym regionie dostarczając tym samym nieosiągalne dotychczas dane dotyczące dynamicznych zmian jakie zachodzą w okresie godzin, dni czy tygodni w danym regionie.

Przeprowadzono badania kilku silników baz danych, by zmaksymalizować szybkość z jaką obiekty będą do niej dodawane. W pierwszej kolejności odrzucono wszystkie silniki baz danych SQL. Z uwagi na specyfikę projektu zapytania kierowane do bazy będą dotyczyły obszarów dzięki czemu wydajniejszą metodą jest sięganie do konkretnych wierzchołków grafu niż przeszukiwanie i łączenie tabel.

Aktualnie jest dostępnych wiele systemów zarządzania dla grafowych baz danych m.in: DEX, InfiniteGraph, MongoDB, Neo4j, OrientDB, TinkerGraph i Titan. Dostępne porównania tych silników w literaturze dają zróżnicowane rezultaty. W pierwszej kolejności zweryfikowano koszty licencji każdego z nich, możliwość administracji i kontroli dostępu do zasobów oraz wsparcia dla wielu języków programowania, by w przyszłości zapewnić dostęp do danych dla wielu analityków niezależnie od języka programowania, którym biegle się posługują. Z uwagi na powyższe odrzucono kilka z nich na etapie wstępnej selekcji tj:

- DEX nie posiada kontroli nad dostępem do zasobów (nadawania uprawnień dla konkretnych użytkowników-każdy użytkownik ma dowolny dostęp do każdego rekordu).
- InfiniteGraph jest darmowy jedynie do 60 dni
- MongoDB w wersji darmowej nie posiada wbudowanego GUI do wizualizacji struktury danych i obiektów w nich zawartych, która jest niezwykle pomocna w procesie weryfikacji prawidłowego działania bazy oraz gene-

rowania raportów dla klientów końcowych bez potrzeby wykorzystywania zewnętrznych narzędzi

- TinkerGraph oraz Titan wspierają niewiele języków programowania (TinkerGraph - Java i Groovy, a Titan - Java, Python i Clojure), a dodatkowo nie posiadają RESTowego API co znacznie zawęża wybór technologii w przyszłości.

Dwie wybrane do badań bazy grafowe to: Neo4j i OrientDB ze względu na swoją popularność (duże wsparcie, częste aktualizacje, dobrze opisana dokumentacja), rozbudowane RESTowe API oraz możliwosć integracji z wieloma językami programowania.

Testy wybranych baz zostały przeprowadzone na fizycznej maszynie o następujących parametrach:

- System operacyjny: Windows 10 Pro
- Procesor: Intel Core i7-10510U
- RAM: 16GB, 2666 MHz SODIMM DDR4
- Dysk: SSD M.2 PCIe 512 GB

Procedura przeprowadzonych eksperymentów obejmowała nawiązywanie połączenia z bazą, a następnie sekwencyjne wykonywanie pojedynczego zapytania wykonującego operację zapisu i odczytu elementów wewnątrz. Testy zostały przeprowadzone wykorzystując język Python i rekomendowane przez dostawców OrientDB i Neo4j biblioteki do integracji. Dla OrientDB rekomendowaną w dokumentacji biblioteką jest pyorient, a dla Neo4j jest to pakiet neo4j. Testy obejmowały szereg różnych operacji wyróżnionych w tabeli 3.6. Bazy zostały przetestowane poprzez weryfikację czasu potrzebnego do zapisu nowych obiektów i relacji, a także szybkości wyszukiwania pojedynczego rekordu dla różnej liczby zapisanych wewnątrz rekordów. Zarówno OrientDB jak i Neo4j zostały uruchomione w sposób zapewniający reprezentatywność wyników (ustawiono identyczny rozmiar stosu i uruchomiono je na tej samej maszynie).

Ustawienia OrientDB/Neo4j:

Nazwa	Neo4j				
Wykorzystana biblioteka	neo4j				
Ilość operacji	Czas zapisu nodów [s]	Czas zapisu relacji [s]	Czas odczytu wszystkich elementów [s]	Czas znalezienia pojedynczego elementu [s]	Czas usunięcia wszystkich elementów [s]
10 tyś	11.313	11.023	0.001	0.001	0.001
50 tyś	59.340	57.828	0.001	0.003	0.004
100 tyś	120.959	116.533	0.068	0.009	0.011
250 tyś	224.129	217.434	0.401	0.016	0.015

Nazwa	OrientDB				
Wykorzystana biblioteka	pyorient				
Ilość operacji	Czas zapisu nodów [s]	Czas zapisu relacji [s]	Czas odczytu wszystkich elementów [s]	Czas znalezienia pojedynczego elementu [s]	Czas usunięcia wszystkich elementów [s]
10 tyś	2.949	4.615	1.068	0.025	0.341
50 tyś	16.045	27.480	4.597	0.124	1.389
100 tyś	37.022	49.774	8.878	0.249	2.700
250 tyś	81.166	155.355	23.779	0.616	7.638

Tabela 3.6: Tabela przedstawia średni czas (z 100 prób) wyrażony w sekundach potrzebny do zapisu określonej w wierszu liczby operacji (od 10 tyś do 250 tyś). Dane do tworzenia nodów i krawędzi były syntetyczne - zostały wygenerowane sztucznie wyłącznie w celu przeprowadzenia eksperymentu.

- rozmiar stosu po inicjalizacji: 1GB
- maksymalny rozmiar stosu: 2GB

Przeprowadzone testy szybkości miały w wierny sposób symulować działania jakie będą wykonywane w środowisku produkcyjnym systemu tzn. wielu klientów równocześnie podłączonych do BD przesyła informacje wymagające porównania oraz zapisania jako nowa encja lub aktualizacja istniejącej. Testy nie grupują transakcji celowo, by zweryfikować szybkość zapisu i aktualizacji w przypadku pesymistycznym (tj. osobne query dla każdego znalezionego obiektu). W środowisku produkcyjnym czas aktualizacji elementów w bazie danych stanowi element krytyczny. Zwłoka polegająca na długim buforowaniu danych i zapisanie ich po uprzednim zgrupowaniu podczas pojedynczej transakcji znacznie

zwiększyłaby wydajność bazy jednak system w założeniu ma za zadanie wizualizować mapę w czasie rzeczywistym co wymusza natychmiastową reakcję po uzyskaniu informacji o lokalizacji obiektu znalezionego przez użytkownika. W systemie przewidziano jedynie niewielkie opóźnienie w postaci przekazywania obiektów do zapisu z interwałem 0.3 s. Testy zapisują informacje bez wymienionego opóźnienia, po to by na wczesnym etapie projektu wyłonić bazę danych, która najlepiej radzi sobie z dużą liczbą transakcji wykonanych w krótkim czasie.

Rezultaty testów wykazały na znacznie niższy czas zapisu nodów i relacji do bazy uzyskał OrientDB. Neo4j zapisywał te same informacje (nody i relacje) kilkukrotnie dłużej jednak potrzebował znacznie mniej czasu na odczyt informacji o wybranym elemencie. W celu zniwelowania tej nierówności tj. skrócenie czasu odczytu pojedynczego elementu z bazy danych przewidziano klasteryzację w środowisku produkcyjnym jednak Neo4j nie umożliwia jej przeprowadzenia w wersji community (dostępnej za darmo do użytku komercyjnego). Umieszczając nody w klastrach, optymalizuje się query wyszukując wymaganych informacji jedynie w podzbiorze wszystkich elementów w bazie danych. Z uwagi na powyższe do docelowego projektu wybrano OrientDB.

Klasteryzacja bazy danych, poza zwiększeniem prędkości odczytu poszczególnych elementów z bazy (dzięki ich podziałowi na zbiory o mniejszej liczebności), posiada szereg innych zalet, które zostały wzięte pod uwagę na etapie projektowania całego systemu. Zapewnia redundancję danych tzn. awaria jednego z nodów w klastrze nie powoduje braku możliwości zapisu informacji do bazy co zabezpiecza przed ich utratą. Ponadto, dzięki mechanizmom rozpraszania ruchu sieciowego na poszczególne nody uzyskuje się wysoką dostępność co przekłada się na większą liczbę zapytań obsługiwanych równolegle. Klasteryzacja umożliwia również łatwe skalowanie docelowego systemu wynikające z rosnących potrzeb, bez przerw w jego działaniu, co eliminuje ograniczenia wynikające z osadzenia bazy danych na pojedynczym serwerze.

System zapisuje dane do bazy w sposób ustrukturyzowany. Cała baza danych została zaprojektowana w sposób zapewniający szybką obsługę najczęstszych zapytań jakie są do niej kierowane (tj. wyszukaj i zapisz obiekty w wybranym

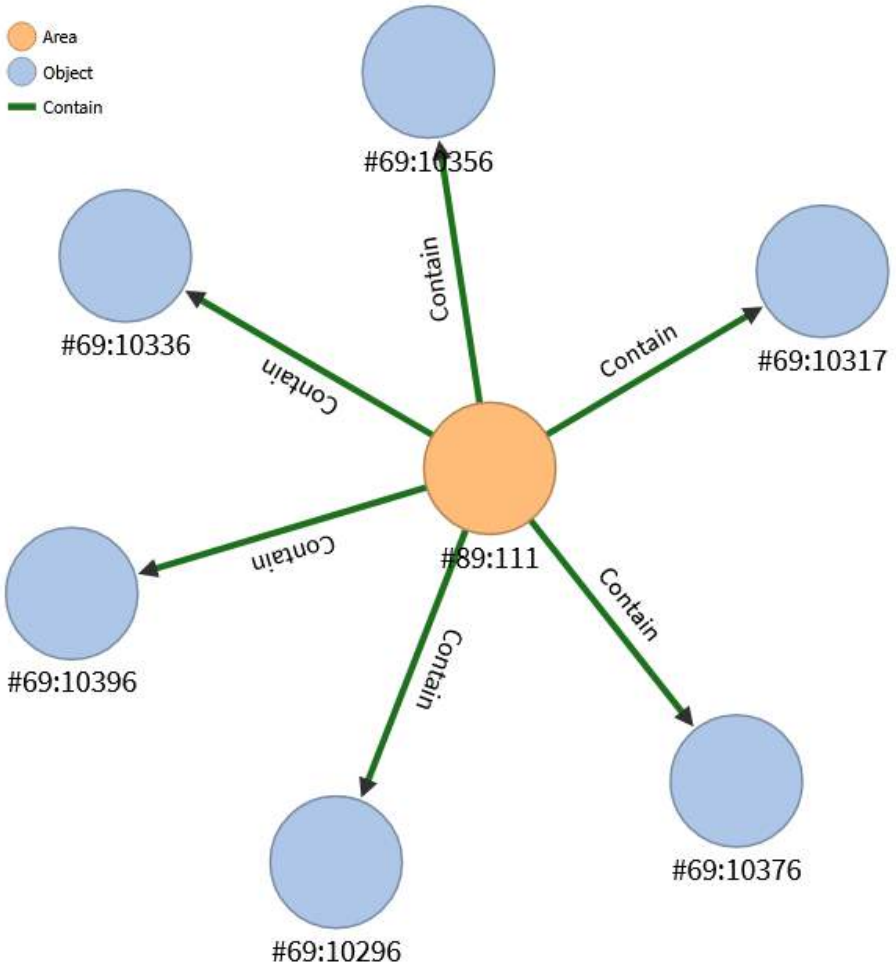
regionie). Nadesłane detekcje obiektów są połączone z daną strefą (o maksymalnym rozmiarze około 1.1km^2 dla Polski)3.5. W bazie przewidziano również możliwość dalszych podziałów, gdyby powyższe okazały się niewystarczające. Struktura danych jest drzewiasta co ogranicza listy wierzchołków wymagających przejrzenia. Obiekty (Object) są połączone z obszarami (Area), które posiadają identyczną długość (λ) i szerokość (ϕ) geograficzną po zaokrągleniu do części setnych (np. detekcja obiektu o ϕ=19.34051334 i λ=50.24892310 tworzy obiekt w bazie i łączy go z obszarem ϕ=19.34,λ=50.25). Podział mapy na niewielkie fragmenty ogranicza istotnie liczbę przeglądanych wierzchołków grafu w celu dotarcia do interesujących danych.

3.3.3 Algorytmy estymacji lokalizacji obiektów w świecie rzeczywistym

Głównym celem projektu jest utworzenie mapy na podstawie danych strukturalnych i temporalnych, która dostarczy informacji jak zmieniał się wybrany przez użytkownika region. Poza wizualizacją archiwalnego stanu wybranego regionu, system ma umożliwiać aktualizację lokalizacji obiektów w nim zawartych w czasie rzeczywistym (tj. do 2s od chwili odebrania informacji od urządzenia brzegowego o wystąpieniu danego obiektu w wybranej lokalizacji). Początkowo następuje pobranie informacji archiwalnych z bazy, a następnie każda nowa informacja przetworzona przez serwer jest od razu kierowana do widoku użytkownika (niezależnie od zapisu tej informacji w bazie danych). W ten sposób wizualizowane dane są prezentowane bez zbędnego opóźnienia wynikającego z potrzeby ich uprzedniego zapisu do bazy, a następnie oczekiwania na ich odczyt.

Estymacja lokalizacji po stronie urządzenia brzegowego

Synchronizacja danych przetwarzanych przez system rozpoczyna się na etapie detekcji obiektu na urządzeniu brzegowym. Zdefiniowano szereg reguł, które stanowią pierwszy filtr określający czy dany obiekt stanowi element, który już wcześniej był wykryty przez urządzenie użytkownika lub też czy jest to nowy obiekt, który wcześniej nie był rozpoznawany przez sieć neuronową osadzoną na



Rysunek 3.5: Wizualizacja struktury bazy danych przechowującej informacje o obiektach w wybranych obszarach. Na zaprezentowanym przykładzie kilka obiektów znajduje się w jednym obszarze.

urządzeniu.

Wyznaczenie kierunku w którym jest zwrócony smartphone jest jednym z argumentów doprecyzowywania fizycznej pozycji znalezionego obiektu (w formacie wysokości i szerokości geograficznej przesyłanej na serwer). Poza określeniem kierunku, na prawidłową estymację lokalizacji na wpływ dokładność działania algorytmu detekcji. Kierunek w którym znajduje się obiekt oddalony o estymowaną odległość jest wyznaczany poprzez dane dostarczane z czujnika IMU umieszczonego wewnątrz smartfona. Skrót IMU pochodzi od ang. Inertial Measurement Unit i jest to grupa sensorów odpowiedzialna za określenie pozycji te-

lefonu. Magnetometr, akcelerometr i żyroskop stanowią standardowe wyposażenie większości współczesnych smartfonów. Zaimplementowana aplikacja oblicza kąty orientacji 3.6, wykorzystując czujnik pola geomagnetycznego w połączeniu z akcelerometrem urządzenia. Korzystając z wskazanych czujników sprzętowych można określić orientację telefonu za pomocą następujących trzech kątów:

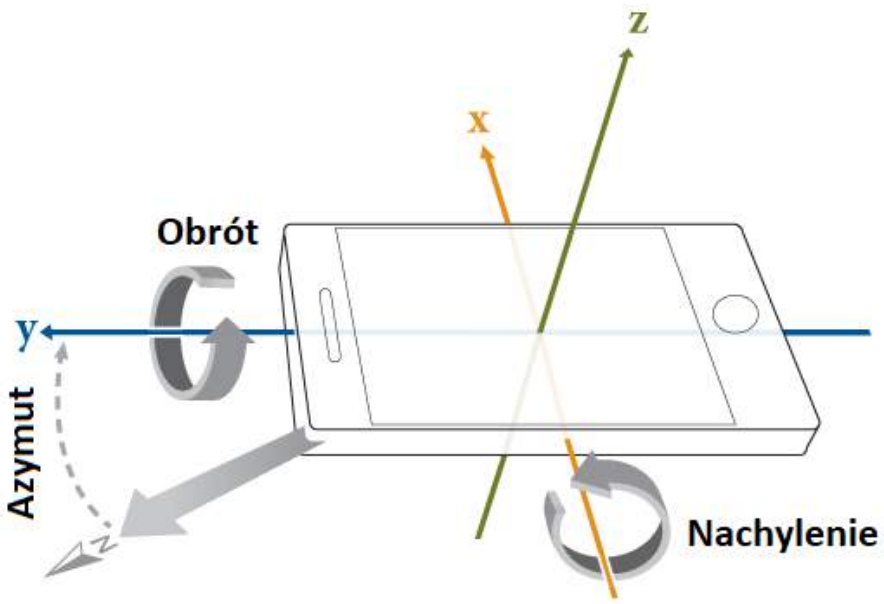
- Azymut (stopnie obrotu wokół osi z). Jest to kąt pomiędzy aktualnym kierunkiem kompasu urządzenia a północą magnetyczną mierzony zgodnie z kierunkiem wskazówek zegara. Azymut przyjmuje wartości od 0°do 359°. Jeśli górna krawędź telefonu jest skierowana na północ magnetyczną, azymut wynosi 0 stopni, jeśli na zachód to 270 stopni.

- Nachylenie (stopnie obrotu wokół osi x - ang. pitch). Jest to kąt pomiędzy płaszczyzną równoległą do ekranu urządzenia a płaszczyzną ziemi. Jeśli urządzenie jest trzymane równolegle do podłoża to wartość wynosi 0. Nachylenie przyjmuje wartości od -180°do 180°. Trzymanie telefonu w pozycji pionowej, dolną krawędzią skierowaną do podłoża zwraca wartość -90°.

- Przechylenie (stopnie obrotu wokół osi y - ang. roll). Jest to kąt pomiędzy płaszczyzną prostopadłą do ekranu urządzenia a płaszczyzną prostopadłą do podłoża. Jeśli urządzenie znajduje się równolegle do podłoża, a jego dolna krawędź znajduje się najbliżej użytkownika to przechylenie lewej krawędzi urządzenia w kierunku podłoża powoduje, że kąt przechylenia staje się dodatni. Zakres wartości wynosi od -180 stopni do 180 stopni.

Innym źródłem informacji wykorzystywanych w systemie budowy mapy są kamery umieszczone w mieście, które nie są wyposażone we wskazane czujniki. W ich przypadku należy skonfigurować kierunek w którym są zwrócone oraz ręcznie wpisać ogniskową i wielkość sensora (w systemie iOS oraz Android te dane są dostarczane przez samo urządzenie).

Liczba obiektów występujących w obrazie kamery oraz ich rozmieszczenie jest w pierwszej kolejności przetwarzane przez algorytm śledzenia obiektów na obrazie. Stanowi to pierwszy filtr określający czy kolejne detekcje obiektu na obrazie stanowią powtórzenie obiektu uprzednio znalezionego. Podczas badań



Źródło: publikacja "Measurement Level Experimental Test Result of GNSS/IMU Sensors in Commercial Smartphones"[51]

Rysunek 3.6: Schemat poglądowy wartości zwracanych przez czujnik pola magnetycznego i akcelerometr.

skupiano się na możliwie jak najmniejszym obciążeniu urządzenia w trakcie działania algorytmu śledzącego obiekty na obrazie. Osadzenie algorytmu detekcji obiektów na urządzeniu o niewielkich zasobach już znacznie zmniejszyło ilość dostępnego RAMu oraz obciążyło procesor. Zatem dodanie algorytmu śledzenia obiektów, będącego jedynie elementem wspomagającym decyzję o nieprzesyłaniu redundantnych detekcji na serwer, powinno w możliwie jak najmniejszym stopniu wpłynąć na wydajność pozostałych komponentów w aplikacji. Z uwagi na powyższe, badania skupiły się na wydajności algorytmów śledzących, odrzucając podczas wstępnej selekcji te, które wymagają dużej ilości zasobów na urządzeniu, bez względu na dokładność ich działania. Na podstawie ogólnie dostępnych w literaturze informacji wybrano oraz przetestowano kilka algorytmów śledzenia obiektów tj:

- KCF (ang. Kernelized Correlation Filters).

- CSRT[59] (ang. Discriminative Correlation Filter with Channel and Spatial Reliability)
- MOOSE (ang. Minimum Output Sum of Squared Error)
- GOTURN[33] (ang. Generic Object Tracking Using Regression Networks)

W ramach testów przygotowano 10 filmów charakteryzujących się różną dynamiką przemieszczania się obiektów na obrazie. Test polegał na weryfikacji działania określonego algorytmu na filmach o rozdzielczości 640x480px nagranych telefonem, które zawierały obiekt przemieszczający się z różnymi prędkościami. Zmieniano także wielkości śledzonych obiektów (przybliżano i oddalano obiekty od kamery), a także weryfikowano zachowanie algorytmu w przypadku częściowego i całkowitego zniknięcia, a następnie ponownego pojawienia się obiektu w polu widzenia kamery.

Śledzenie obiektów z wykorzystaniem algorytmu KCF ujawniło szereg wad jakie posiada ten algorytm wynikających ze sposobu jego działania. Gdy prędkość ruchu obiektów na obrazie jest bardzo duża, KCF nie jest w stanie precyzyjnie śledzić wybranego obiektu. Algorytm działa na zasadzie porównywania poprzedniej z bieżącą klatką obrazu przez co brak dokładności w śledzeniu w kolejnych klatkach kumuluje błąd polegający na śledzeniu tła zamiast samego obiektu. Sam proces opiera się na kilku warstwowych procesach filtrowania, gdzie algorytm tworzy pole ograniczające wokół obiektu, który ma być śledzony i porównuje cechy elementów wewnątrz pola ograniczającego w kolejnych klatkach. KCF ma wysoki współczynnik śledzenia, jeśli między kamerą a obszarem zainteresowania nie ma przeszkód, a obiekt przemieszcza się z relatywnie niewielką prędkością. Z subiektywnych obserwacji wynika, że jeśli jakaś przeszkoda znajdzie się w obszarze docelowym, algorytm gubi obiekt i śledzi błędny obszar (przeszkodę) 3.10. Przyczyną tego problemu jest fakt, że mapa śledzonego obiektu tworzona przez KCF usuwa inne obszary poza obszarem śledzenia (w którym znalazła się przeszkoda) i konstruuje mapę zawierającą niepożądany obszar.

Algorytm CSRT wykorzystuje mapę wiarygodności przestrzennej, by dopasować filtr do części wybranego regionu. Zapewnia to lepsze śledzenie niepro-

stokątnych regionów lub obiektów. Wykorzystuje tylko 2 standardowe funkcje (HoG i Colorname). HoG to skrót od ang. Histogram of Oriented Gradients i stanowi to deskryptor cech, który pozwala na wybór informacji użytecznej wewnątrz śledzonego obszaru. Funkcja colorname dodatkowo wzmacnia efekt wyboru informacji użytecznej poprzez porównywanie kolorów jakie zawiera śledzony obszar. CSRT zapewnia wyższą dokładność śledzenia obiektów niż KCF w przypadku śledzenia obiektów szybko przemieszczających się na obrazie z kamery. Niestety, również w przypadku tego algorytmu chwilowe zasłonięcie obiektu powoduje, że tło staje się częścią mapy śledzenia i algorytm przestaje podążać za prawidłowym obszarem.

MOOSE wykorzystuje adaptacyjną korelację cech fragmentu obrazu do śledzenia obiektów. Ponadto generuje filtry korelacji, gdy jest inicjalizowany przy użyciu pojedynczej klatki. Śledzenie z wykorzystaniem tego algorytmu jest odporne na zmiany oświetlenia, skali, pozy i niesztywne deformacje. Wykrywa on również sytuację w której obiekt jest chwilowo zasłonięty, co pozwala na wstrzymanie i wznowienie pracy w momencie ponownego pojawienia się obiektu. MOSSE podczas testów okazał się szybszy niż KCF i CSRT, wymagając przy tym zbliżonej ilości wykorzystywanych zasobów na urządzeniu.

Śledzenie obiektów z wykorzystaniem GOTURN zaimplementowanego w TensorFlow pozwoliło uzyskać najwyższą skuteczność przy prędkości nieco niższej niż MOOSE jednak samo uruchomienie na urządzeniu mobilnym nie było możliwe z uwagi na niewystarczające zasoby jakie pozostały po osadzeniu detektora obiektów stanowiącego najistotniejszy element działania aplikacji na urządzeniach brzegowych.

Spośród badanych algorytmów śledzenia obiektów wybrano i zaimplementowano MOOSE w docelowym systemie z uwagi na fakt szybkości działania i satysfakcjonujących rezultatów jakie uzyskał dla przygotowanych filmów testowych. Rezultaty szybkości działania algorytmów śledzących zostały pokazane w tabeli 3.7.

Poza informacją pochodzącą z algorytmu śledzenia obiektów na obrazie, podczas przeprowadzonych eksperymentów wyłoniono kilka reguł decyzyjnych, któ-

Rysunek 3.7: Algorytm KFC przestaje śledzić obiekt jeśli jego spora część znajdzie się poza widokiem kamery (na ostatnim obrazie nie ma ramki śledzącej obiekt)

Rysunek 3.8: Algorytm CSRT po chwilowym zniknięciu obiektu przestaje śledzić prawidłowy obszar

Rysunek 3.9: Algorytm MOOSE nadal z satysfakcjonującą dokładnością śledzi obiekt, który zniknął z obszaru kamery

Rysunek 3.10: Rysunki wskazują na różnice w działaniu algorytmów w przypadku przetwarzania filmu ze smartfona. Z uwagi na częste, szybkie zmiany widoku kamery smartfona szczególnie ważnym kryterium była możliwość śledzenia obiektów o dużym rozmyciu i znikających na chwilę z widoku kamery.

re dodatkowo wspomagają określenie czy dany obiekt jest powtórzeniem uprzednio wykrytego. W przypadku, gdy algorytm śledzenia obiektów zauważy nowy obiekt, który wcześniej nie pojawiał się na obrazie następuje wyszukiwanie danej klasy obiektu w pamięci telefonu (która działa jako pamięć cache przechowując informacje o stu ostatnich znalezionych obiektach). Po znalezieniu tożsamej klasy w pamięci, następuje porównanie estymacji lokalizacji aktualnie widzianego w widoku kamery obiektu z pozycją zapisaną w lokalnym archiwum. W zależności od klasy, której pozycja jest analizowana tzn. czy jest to klasa obiektów nieruchomych (takich jak ławka czy znak drogowy) czy ruchomych (rower,

L.p.	Tracker	FPS
1	GOTURN	6
2	CSRT	8
3	KCF	101
4	MOOSE	614

Tabela 3.7: Tabela przedstawiająca rezultaty testów szybkości algorytmów śledzenia wyrażone w średniej liczbie przetworzonych klatek na sekundę w ciągu testu trwającego 5 minut wykonanym na urządzeniu Samsung A40.

samochód) analiza przebiega na dwa różne sposoby. W przypadku obiektów nieruchomych, zbudowano zestaw reguł określających w jakiej odległości od wcześniejszej lokalizacji decyduje się, że dany obiekt jest z dużym prawdopodobieństwem tym, który znaleziono wcześniej. W przypadku obiektów ruchomych, poza aspektem odległości system określa czas jaki może upłynąć pomiędzy detekcjami. W systemie zapisano indywidualną odległość oraz czas w którym może nastąpić zmiana pozycji dla każdej klasy np. dla znaku drogowego jest to 5m bez względu na czas (jest to element niezmienny otoczenia), dla hydrantu jest to 15m bez względu na czas, dla rowerów jest to 20m w czasie do 10s od poprzedniej detekcji. Wskazane reguły są w pełni konfigurowalne po stronie serwera, a aplikacja pobiera je w chwili uzyskania informacji o ich zmianie nadpisując dotychczas przechowywaną lokalnie kopię konfiguracji (eliminującą konieczność odpytywania o konfigurację przy każdym starcie aplikacji).

Estymacja lokalizacji po stronie serwera

Po przetworzeniu wszystkich informacji z poziomu urządzenia końcowego następuje synchronizacja danych na poziomie serwera. Skupia się ona przede wszystkim na synchronizacji danych pozyskanych od wielu użytkowników równolegle dotyczących określonego obszaru. Potok danych odebranych od urządzeń brzegowych dotyczących detekcji i lokalnej estymacji lokalizacji obiektów jest kolejkowany i przetwarzany przez autorski algorytm doprecyzowania lokalizacji, który jest osadzony na serwerze.

Podstawą działania algorytmu estymacji lokalizacji po stronie serwera są informacje przechowywane w bazie danych określających położenie obiektów

w bezpośrednim otoczeniu urządzenia, a następnie uspójnienie ich z widokiem z kamery użytkownika. Serwer wykorzystuje następującą funkcję do obliczania odległości pomiędzy dwoma dowolnymi punktami na mapie (korzystając z ich wysokości i szerokości geograficznej jako dane wejściowe):

```
from math import radians, sin, cos, acos
#(...)
    #funkcja obliczająca odległości pomiędzy dwoma punktami na podstawie ich koordynatów
    def calculateDistanceBetweenLonLat(lat1, lon1, lat2, lon2):
        lat1 = radians(float(lat1))
        lon1 = radians(float(lon1))
        lat2 = radians(float(lat2))
        lon2 = radians(float(lon2))
        radius_of_the_earth= 6371.008
        dist = radius_of_the_earth * acos(sin(lat1)*sin(lat2)
            + cos(lat1)*cos(lat2)*cos(lon1 - lon2)) * 1000
        return dist
```

system wylicza odległość do obiektów tej samej klasy znajdujących się w promieniu do maksymalnie 50m (wynikających z maksymalnej detekcji obiektów na takiej odległości-powyżej niej detekcja nie jest skuteczna lub obiekty muszą być ogromne np. samoloty). Do wskazanej wartości dodaje się wartość 0.8 precyzji sygnału GPS od nadesłanej pozycji (bazując na poprzednich badaniach 3.2 nad zwracanym marginesem błędu określono, że faktyczny błąd wynosił z reguły maksymalnie 0.8 wartości możliwego błędu precyzji określonego przez moduł GPS), by zweryfikować czy znaleziony obiekt nie jest już zapisany w bazie danych. Ponadto, system weryfikuje jakie obiekty powinny znajdować się aktualnie w widoku kamery użytkownika. W przypadku stwierdzenia, że obiekty w bazie nie znajdują się już we wskazanej lokalizacji system zapisuje do bazy czas w którym nastąpiło wykrycie tej sytuacji. Ten mechanizm umożliwia usunięcie danego obiektu z widoku użytkownika oraz analizę zmian jakie zachodziły na danym obszarze. Jeśli obiekt stanowi powtórzenie i sygnał GPS jest silniejszy od poprzednio zebranych to następuje zmiana jego lokalizacji w bazie danych.

W przypadku obiektów nieruchomych, zaklasyfikowanych na urządzeniu jako powtórzenie, następuje przesłanie na serwer informacji o tym obiekcie oraz wskazanie, że jest to redundancja względem poprzednio wysłanego obiektu wraz z aktualizacją jego lokalizacji. Serwer odbierając te dane pobiera z bazy danych informację o poprzedniej lokalizacji oraz wylicza nową estymację położenia obiektu. Dla obiektów nieruchomych algorytm wykonuje następujące czynności:

1. Odbierz informację o obiekcie nieruchomym (jego estymowanej lokalizacji, orientacji telefonu (azymucie, obrocie i nachyleniu), lokalizacji GPS urządzenia/telefonu oraz dokładność modułu GPS jaką w chwili detekcji posiadało urządzenie).

2. Pobierz informację z pamięci cache oraz z bazy danych o obiektach, które znajdują się wokół użytkownika (w odległości maksymalnie 50m + 0.8 wartości błędu GPSa wyrażonego w metrach).

3. Określ czy nadesłany obiekt stanowi element aktualnie zapisany w bazie danych na podstawie zdefiniowanych reguł decyzyjnych (tj. reguły określającej jaka odległość pomiędzy obiektem zapisanym w bazie oraz nadesłanym powoduje jego scalenie np. dla hydrantu jest to 15m z uwagi na fakt, że w świecie rzeczywistym odległości pomiędzy nimi jest wyższa niż 15m). Jeśli na podstawie tej reguły określono, że obiekt jest tym samym co zapisany w bazie to następuje doprecyzowanie lokalizacji obiektu w bazie danych. Jeśli jest to obiekt nowy to utwórz niezbędne struktury (służące przechowywaniu coraz dokładniejszych lokalizacji o obiekcie) i zacznij kolekcjonować dane o obiekcie według dalszych kroków.

4. Oblicz na podstawie nadesłanych danych wagę doprecyzowania według wzoru:

$$W = 1 - \frac{E}{MD} \tag{3.2}$$

W – waga pomiaru zapisywana na serwerze. Im wyższa waga tym dokładniejszy jest pomiar. Pomiary, których waga jest ujemna nie są zapisywane
E – margines błędu jaki zwrócił moduł GPS w chwili detekcji obiektu

MD – wartość maksymalnego dystansu dla której zapisywana jest detekcja obiektu. Dystans skutecznej detekcji jest zależny od modelu. Obecnie zapisano w systemie wartość 50m, która ogranicza detekcję bardzo odległych obiektów (pomiary na takiej odległości nie będą dokładne dla wykorzystanej sieci stąd wymaga się skrócenia do niego dystansu).

5. Zweryfikuj czy nadesłany pomiar ma wagę wyższą niż najniższa waga w kolekcji lokalizacji dotyczących danego obiektu (przechowującej skończoną ilość pomiarów np. 50 najlepszych).

   (a) Jeśli nie to pomiń pomiar jako niewystarczająco precyzyjny (na serwerze są zapisane dokładniejsze pomiary od innych urządzeń)

   (b) Jeśli tak to usuń najniższą wartość i połączoną z nią lokalizację, a następnie dodaj nowy pomiar, oblicz i zapisz do bazy nową estymację lokalizacji dla wybranego obiektu według wzoru

$$L = \sum_{i=1}^{n} \frac{l_i * W_i}{\sum_{i=1}^{n} W_i} \tag{3.3}$$

   L – wysokość lub szerokość geograficzna (obie wartości są przeliczane osobno w taki sam sposób)
   l – pojedynczy pomiar wysokości lub szerokości geograficznej zapisany w tabeli najlepszych pomiarów danego obiektu (posiadających najwyższą wagę określającą dokładność)
   W – waga zapisana na serwerze określająca dokładność wybranego pomiaru
   n – liczba wszystkich pomiarów dotycząca lokalizacji wybranego obiektu zapisana na serwerze

6. Jeśli w kierunku w którym jest zwrócone urządzenie nie znajduje się obiekt nieruchomy zapisany w bazie to zapisz informację o jego zniknięciu. Jeśli brak detekcji obiektu we wskazanym miejscu powtórzy się 5 razy, wpisz informację o usunięciu obiektu z mapy do bazy z czasem ostatniej prawi-

dłowej detekcji.

Mechanizm przechowywania tylko najlepszych pomiarów służy ograniczeniu nagłych zmian położenia obiektu wynikającego z pojedynczego pomiaru o niskiej dokładności nadesłanego przez urządzenie brzegowe. Bazując na założeniu, że wraz z rozwojem infrastruktury sieci bezprzewodowych system będzie otrzymywał coraz dokładniejsze pomiary, kolejne nadesłane koordynaty powinny doprecyzować pozycję zapisaną na serwerze dając dokładną lokalizację w świecie rzeczywistym.

Dla obiektów ruchomych (nie będących stałymi elementami krajobrazu) opracowano inny zestaw reguł oraz działań, podejmowanych po stronie serwera, zapewniających ich odpowiednie odzwierciedlenie na wizualizacji użytkownika (na mapie). Obiekty typu rower lub samochód rozpoznane przez użytkownika z założenia często zmieniają swoje położenie o znaczne odległości. W przypadku stwierdzenia po stronie urządzenia, że widziany przedmiot stanowi powtórzenie (wykorzystując zdefiniowane reguły zależności zmiany położenia od czasu) następuje przesłanie na serwer informacji o nowej pozycji, która jest zmieniana niezależnie od wcześniej zapisanej. Takie działanie jest uzasadnione, eliminując konieczność pamiętania obiektu, którego pozycja może zmienić się znacznie w ciągu kilku sekund w sposób niedeterministyczny. Poza wskazaną zmianą lokalizacji system ma również możliwość określania czy dany obiekt ruchomy nadal znajduje się w miejscu jego poprzedniego wystąpienia. W tym celu serwer porównuje aktualną pozycję użytkownika i kierunek w którym zwrócony jest aparat z nadsyłanymi detekcjami. Jeśli w bazie znajduje się zapisany przedmiot reprezentujący obiekt ruchomy w miejscu w którym znajduje się użytkownik, a jego aparat nie wychwytuje w swoim otoczeniu tego obiektu następuje zapisanie znacznika czasu określającego czas, w którym dany obiekt już nie znajduje się w tej lokalizacji. Obiekt jest również usuwany z wizualizacji/mapy użytkownika. W ten sposób tworzona mapa jest dokładniejsza i tym częściej aktualizowana im więcej użytkowników porusza się po danym obszarze. Działanie algorytmu można scharakteryzować następująco (kroki od 1 do 3 są tożsame z sposobem przetwarzania informacji o obiektach nieruchomych):

1. Odbierz informację o obiekcie ruchomym (jego estymowanej lokalizacji, pozycji telefonu (azymucie, pochyleniu, przechyleniu), lokalizacji GPS urządzenia oraz dokładność modułu GPS jaką w chwili detekcji posiadało urządzenie).

2. Pobierz informację z pamięci cache oraz z bazy danych o obiektach, które znajdują się wokół użytkownika (w odległości maksymalnie 50m + 0.8 wartości błędu GPSa wyrażonego w metrach).

3. Określ na podstawie zdefiniowanych reguł decyzyjnych (tj. reguł określających jaka odległość pomiędzy obiektem zapisanym w bazie oraz nadesłanym powoduje jego scalenie np. dla roweru jest to odległość 30m w ciągu 10s od chwili zgłoszenia detekcji przez innego użytkownika w skanowanym obszarze).

4. Jeśli wskazany obiekt nie został wcześniej rozpoznany w danym regionie lub czas jest wyższy niż określony w regule decyzyjnej następuje zapisanie nowego obiektu do bazy wraz ze wskazaniem czasu pojawienia się obiektu w danej lokalizacji.

5. Zakwalifikowanie obiektu jako tożsamy powoduje nadpisanie jego poprzedniej lokalizacji bez względu na dokładność uzyskanej lokalizacji (o ile obliczony dystans do wykrytego obiekt nie przekracza maksymalnej odległości skutecznej detekcji zdefiniowanej dla wykorzystanego modelu)

6. Jeśli w kierunku w którym jest zwrócone urządzenie, w widoku kamery nie znajduje się obiekt ruchomy zapisany w bazie to informacja o wykryciu jego braku jest rejestrowana w postaci zapisania czasu kiedy obiekt przestał występować. W bazie zapisywany jest czas wykrycia braku obiektu w formie znacznika czasu.

W przypadku kamery w mieście, której pozycja jest nieruchoma, lokalizacja statycznego obiektu widzianego w jej obrazie jest w pierwszej kolejności wysyłana jednorazowo na serwer tuż po wykryciu (by możliwie szybko uzyskać informacje co znajduje się w widoku kamery), a następnie dokładna odległość

jest uśredniana bezpośrednio na urządzeniu (bierze się pod uwagę 24 pomiary wykonywane co godzinę) i ponownie przesyłana na serwer. Interwał dobowy został zaimplementowany, by zminimalizować ilość operacji aktualizacji obiektów statycznych na serwerze. W przypadku wykrywania obiektów ruchomych, kamera przesyła informację o ich lokalizacjach na bieżąco aż do ich zniknięcia z jej widoku. Algorytm śledzenia obiektów przetwarza obraz, a przemieszczenie obiektu ruchomego poza obszar widoku kamery powoduje automatyczne przesłanie na serwer informacji o jego zniknięciu wraz z aktualnym czasem kamery.

Poza mechanizmami opisanymi powyżej, uruchamianymi bezpośrednio po odebraniu informacji o nowym obiekcie wykrytym przez urządzenia brzegowe wymagającym zapisu lub aktualizacji obiektów aktualnie zapisanych w bazie, po stronie serwera opracowano jeszcze jeden proces, który wykonuje analizę zebranych danych w godzinach serwisowych. Polega on na przetwarzaniu danych strukturalnych przez heurystyczny algorytm klasyfikacji, którego zadaniem jest rozpoznawanie wzorców i na ich podstawie określenie czy na analizowanym terenie znajdują się wysypiska śmieci lub inne uprzednio zdefiniowane, poetykietowane i wytrenowane rodzaje obiektów/zdarzeń. Algorytm aktualnie wyszukuje następujące rodzaje zdarzeń:

- nielegalne parkingi
- miejsca postoju rowerów oraz ścieżki rowerowe
- wysypisko/sterta śmieci
- miejsca wymiany książek (ang. book sharing)

Algorytm otrzymuje jako dane wejściowe strukturę reprezentującą zagęszczenie danych obiektów tj. ilość oraz rozmieszczenie obiektów w danej lokalizacji. Jeśli średnia odległość obiektów od siebie jest niewielka, a ich liczba przekracza wartość progową to następuje wskazanie, że w danej lokalizacji znajduje się dany rodzaj zdarzenia. Klasyfikacja jest zależna od rodzaju analizowanego terenu np. w lasach liczba puszek i butelek jest sumowana. Jeśli ich suma przekracza 30 a średnia odległość pomiędzy nimi jest mniejsza niż 100m (z uwagi na fakt niskiej

jakości lokalizacji GPS obszar może być dodatkowo rozszerzany jednak nie więcej niż 200m) wtedy algorytm zwraca informację, że w danym punkcie (uśredniając przesłane lokalizacje puszek i butelek) najprawdopodobniej znajduje się nielegalne składowisko śmieci. Zbudowane klasyfikatory rozpoznające podane wyżej wzorce zostały stworzone w celach promocji stworzonego narzędzia wizualizacji danych jako mechanizmu wspomagającego wnioskowanie przetwarzającego dane surowych i wizualizującego rezultatu na mapie. Dane, wykorzystane do walidacji klasyfikatorów zostały wygenerowane na podstawie danych zebranych podczas testów manualnych systemu agregacji informacji z urządzeń brzegowych. Cały proces działa niezależnie, nie blokując przy tym możliwości równoczesnego zapisu do bazy nadsyłanych danych, jednak nie jest uruchamiany w szczytowych momentach wykorzystywania systemu, by nie wpływać negatywnie na wydajność tego procesu. Wskazane ograniczenie można wyeliminować poprzez kopiowanie bazy danych i pracę na archiwalnych informacjach (bez możliwości otrzymywania aktualizacji lokalizacji elementów w nim zawartych).

3.4 Optymalizacja procesu wyszukiwania danych pochodzących z map Open Street Map wykorzystywanych do renderingu

Aktualnie dostępne serwery Open Street Map są oparte na systemie zarządzania bazą danych PostgreSQL. Zapewnienie renderingu w czasie poniżej 0.2s wymaga wyrenderowania każdego poziomu mapy i na żądanie zwrotu interesującego użytkownika fragmentu. Takie podejście zapewnia szybkość działania jednak wymaga znacznych zasobów sprzętowych do przechowywania wszystkich wyrenderowanych map (dla całej planety jest to 1.2Tb przestrzeni dyskowej, rekomendowane są 32 rdzenie CPU oraz 128GB pamięci RAM).

Sposobem na obniżenie wymagań jest przeniesienie renderingu na urządzenia końcowe klientów, którzy z serwera pobierają jedynie informacje niezbędne do renderingu. W takim rozwiązaniu na serwerze baza danych jest odpytywana

o dane niezbędne do renderingu wybranego regionu wraz z poziomem szczegółowości jaki interesuje klienta. Najniższym poziomem szczegółowości są to kontury granicy państw, a najwyższym np. numery budynków.

Podczas badań nad minimalizacją czasu odpowiedzi serwera w postaci wyrenderowanego obszaru mapy zauważono jednak, że największym ograniczeniem systemu renderingu (w przypadku braku wcześniejszego renderingu wybranego obszaru) jest proces zwrócenia odpowiedzi przez PostgreSQL. Znajdują się w niej wszystkie informacje dostępne w OSM. Cała struktura bazy jest opisana w oficjalnej dokumentacji OSM. W ramach badań postanowiono zastąpić system zarządzania bazą danych SQL (w postaci PostgreSQL) na noSQL tj. bazy OrientDB, aby zmniejszyć czas niezbędny do uzyskania prawidłowej odpowiedzi.

Wskazana optymalizacja pojawiła się z uwagi na fakt, że istnieje grupa projektów nisko budżetowych gdzie wykorzystanie dedykowanej, wieloprocesorowej maszyny przekracza budżet klienta. Dla takiego przypadku przeprowadzono testy szybkości wskazanego systemu zarządzania bazą danych na maszynie wirtualnej o niewielkich zasobach (2 GB RAM, 30 GB SSD, 1 rdzeń procesora i7-10510U) inicjując ją danymi o województwie opolskim. Następnie zmieniono system zarządzania bazą danych z PostgreSQL na OrientDB i wykonano szereg zabiegów optymalizacyjnych. Między innymi zaproponowano całkowicie nową strukturę, zorientowaną na przetwarzanie struktur grafowych wewnątrz OrientDB. Zastąpiono strukturę bazy danych PostgreSQL strukturą grafu w OrientDB (dodając dodatkową klasę Area grupującą obiekty w wybranych obszarach) co zmniejsza obszar poszukiwań w bazie podczas wyszukiwań informacji o danym regionie. Wykonanie wskazanych zabiegów wymagało przeprowadzenia szeregu eksperymentów mających na celu potwierdzenie prawidłowości działania nowej bazy danych. Po pobraniu skompresowanych plików map województwa opolskiego i śląskiego wykorzystano ogólnie dostępny konwerter przetwarzający wskazane pliki do formatu .OSM (fragment pliku pokazano poniżej), co pozwoliło zapoznać się z ich formatem pomijając dostępne już po przetworzeniu dane zapisane w PostgreSQL. Umożliwiło to dowolność w projektowaniu bazy i uprościło etap przenoszenia, który został zastąpiony inicjalizacją

bazy danych bezpośrednio z skompresowanego pliku binarnego. Stworzono w ten sposób rozwiązanie szybciej dostarczające niezbędnych informacji do renderingu po stronie urządzenia klienta. Ponadto, zostało ono wykorzystane do narzędzia wizualizacji zmian występujących na danym obszarze w czasie rzeczywistym. Dostarczone z serwera informacje do renderingu stanowią podstawowy widok na którym dodaje się nowe informacje o obiektach jakie znajdują się w danym regionie. Dostarcza to użytkownikowi pełniejszych danych i nie wymusza nieustannego sięgania do zewnętrznych narzędzi celem zorientowania się jaki aktualnie obszar jest mu prezentowany lub co znajduje się w bezpośrednim sąsiedztwie znalezionego obiektu. Dynamiczne dodawanie oraz usuwanie obiektów z mapy jest procesem asynchronicznym. Mapa danego regionu jest pobierana w chwili uruchomienia wizualizacji, a dalsze zmiany nie wymagają jej przeładowywania. Fragment pliku .osm będącego wynikiem działania konwersji z formatu binarnego .pbf:

```
    <?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="osmconvert 0.8.8" timestamp="2020-06-28T20:59:02Z">
<bounds minlat="49.65101" minlon="16.9014999" maxlat="51.2014" maxlon="18.70219"/>
<node id="26860238" lat="50.9646912" lon="18.2789968" version="4"
timestamp="2014-10-07T00:31:18Z" changeset="0">
<tag k="name" v="Bąkow"/>
<tag k="email" v="otwbakow@osir.kluczbork.pl"/>
<tag k="phone" v="+48774180586"/>
<tag k="stars" v="2"/>
<tag k="tents" v="yes"/>
<tag k="tourism" v="camp_site"/>
<tag k="website" v="http://www.osir.kluczbork.pl"/>
<tag k="caravans" v="yes"/>
<tag k="operator" v="Ośrodek Sportu i Rekrecji w Kluczborku"/>
<tag k="ref:pfcc" v="23"/>
</node>
(...)
```

```
<way id="618242297" version="1" timestamp="2018-08-20T11:09:42Z" changeset="0">
<nd ref="3281660363"/>
<nd ref="5843725405"/>
<nd ref="5843725406"/>
<nd ref="5843725407"/>
<nd ref="5843725409"/>
<nd ref="5843725408"/>
<nd ref="3281660390"/>
<nd ref="3281660363"/>
<tag k="amenity" v="parking"/>
</way>
(...)
<relation id="11226792" version="1" timestamp="2020-06-22T14:45:13Z" changeset="0">
<member type="node" ref="7643764569" role="admin_centre"/>
<member type="way" ref="818486902" role="outer"/>
<tag k="admin_level" v="11"/>
<tag k="alt_name" v="Konstytucji 3 Maja"/>
<tag k="boundary" v="administrative"/>
<tag k="name" v="Osiedle Konstytucji 3 Maja"/>
<tag k="name:prefix" v="osiedle"/>
<tag k="type" v="boundary"/>
</relation>
</osm>
```

Pliki binarne po konwersji na format .osm zwiększyły swój rozmiar niemal 30 krotnie (dla porównania opolskie-latest.osm.pbf to plik binarny o rozmiarze około 35MB, a po konwersji na format .osm wynikowy plik ma ponad 810 MB). Zawartość pliku (niezależnie od formatu) wymagała przetworzenia oraz zapisania do bazy danych jako dane niezbędne do renderingu. Zwiększony rozmiar pliku to nie tylko większa ilość danych na dysku, ale również dłuższy czas parsowania, a tym samym opóźnienia w aktualizacji. Uniknięcie tego jest możliwe dzięki użyciu biblioteki obsługującej skompresowane pliki binarne bez potrze-

by ich konwersji. Do obsługi plików binarnych wykorzystano język Python oraz bibliotekę esy.osm.pbf. Wewnątrz bazy utworzono schemat, który odzwierciedla zawartość pliku .OSM wykorzystywanego do renderingu. Schemat bazy składa się z następujących klas:

- Node - punkt będący podstawową jednostką mapy. Każdy punkt przynależy do jakiejś klasy (np. droga, budynek) i posiada informacje o dokładnej szerokości i wysokości geograficznej w której się znajduje.
- Border - granica przyporządkowująca dany punkt na mapie jako krawędź ograniczająca dany obszar (np. lasu)
- Way - droga która jest wyznaczana poprzez łączenie kolejnych Nodów, posiada dodatkowe metadane takie jak nazwę w przypadku ulic lub numer dla dróg krajowych lub autostrad.
- Area - obszar grupujący w bazie dane elementy na mapie znajdujące się w danej lokalizacji. Wskazana klasa obiektów została utworzona w celach optymalizacji zapytań kierowanych do bazy. Klasa ta nie jest bezpośrednio wykorzystywana w pliku OSM, jednak stanowi klucz do szybkiego przeszukiwania bazy danych (tworząc indeks dla szerokości i wysokości geograficznej domyślne wyszukiwanie binarne w bazie danych umożliwia szybsze znalezienie elementów, które powinny znaleźć się na renderowanym obszarze).
- Level - poziom szczegółowości renderowanej mapy. W przypadku bardzo dużego powiększenia renderowane są wszystkie obiekty przynależące do danego obszaru. Im większy obszar jest renderowany tym poziom szczegółowości maleje (na pierwszym poziomie obejmującym cały kraj widać jedynie granice państw i duże zbiorniki wodne tj. morza i oceany.
- Reference - stanowi krawędź łączącą punkty na mapie z określonymi elementami grupującymi takimi jak obszar lub droga

W OrientDB poza powyższym schematem, wykorzystano mechanizm tworzenia klastrów grupujących w sobie Nody znajdujące się w danym obszarze.

OrientDB pozwala na utworzenie maksymalnie 32.767 klastrów w jednej bazie danych. Nody zostały przyporządkowane bazując na swojej lokalizacji do obszarów będących kwadratami o rozmiarach około 25 km^2. W przypadku, gdy renderowany obszar jest mniejszy niż wskazane okno, następuje wyszukiwanie wewnątrz z góry zdefiniowanego klastra (zawierającego podgrupę Nodów) elementów do renderingu. W przypadku pesymistycznym, gdy interesujący obszar znajduje się na przecięciu więcej niż jednego obszaru, następuje wyszukiwanie i filtrowanie Nodów, które znajdują się w każdym sąsiadującym klastrze. Mechanizm wyszukiwania informacji w klastrach jest wykorzystywany wyłącznie w przypadku dużego przybliżenia mapy więc przypadek pesymistyczny obejmuje maksymalnie wyszukiwanie w 4 sąsiadujących ze sobą grup Nodów w osobnych klastrach. W przypadku oddalenia mapy korzysta się z przyporządkowania elementów do klasy Area. Wraz z rozwojem bazy, pierwotnie zadeklarowane rozmiary obszarów można modyfikować zastępując obecne klastry, nowymi dla określonych miast lub wybierając inny rodzaj podziału w zależności od potrzeb.

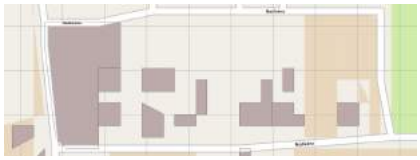
Na etapie badań poprawności danych niezbędnych do renderingu wykorzystano oprogramowanie Maperitive. Na podstawie danych z bazy w formacie OSM generuje ona mapę w postaci graficznej. W docelowym systemie serwer jest odpowiedzialny wyłącznie za wyszukiwanie wymaganych informacji w bazie danych oraz zwracania ich do urządzeń końcowych w formacie OSMa. Następnie, urządzenia końcowe renderują swoją mapę (gruby klient) wykorzystując OsmAnd dla urządzeń mobilnych lub JOSM dla desktopu. Przeprowadzone badania skupiały się na minimalizacji obciążenia serwera, czasu jego odpowiedzi, a także zmniejszenie wymaganych zasobów dla zapewnienia szybszych odpowiedzi niż w klasycznej mapie OpenStreetMap opartej na PstgreSQL.

Dalsza optymalizacja, poza zmianą systemu zarządzania bazą danych noSQL i zaproponowaniem nowego schematu bazy, polegała na zmniejszeniu ilości danych niezbędnych do prawidłowego renderingu wybranego obszaru. Dokładność granic obiektów renderowanych na mapie jest zależna od liczby nodów (wierzchołków wielokąta), które są wykorzystywane do ograniczania obszaru na którym obiekt się znajduje. Z obserwacji wynika, że zarówno PostgreSQL jak

i OrientDB zwracały dużą ilość nadmiarowych (znajdujących się bardzo blisko siebie) nodów, których braki jedynie nieznacznie pogarszały jakość renderowanych widoków 3.14. Dzięki optymalizacji polegającej na grupowaniu nodów znajdujących się blisko siebie (do 10m od siebie) uzyskano mniejszą ilość danych niezbędnych do renderingu mapy. Klasteryzacja nodów w bazie pozwoliła na zmniejszenie jej obojętności (liczba nodów jakie zostały w niej zapisane) o około 12%. Różnice w renderingu po klasteryzacji są widoczne najbardziej w dużym powiększeniu z uwagi na fakt scalania nodów znajdujących się w odległości do kilku metrów od siebie. Podczas klasteryzacji scalane punkty były usuwane, a na ich miejsce tworzono nowy nod, którego kordynaty były wyliczane jako średnia z koordynat wszystkich składowych. Nowy nod stawał się składową każdego kształtu do którego należały usunięte nody.

Wprowadzone działania optymalizacyjne pozwalają na osadzenie serwera zwracającego dane na zwykłych komputerach PC o niewielkiej ilości RAMu (min. 2 GB) i uzyskiwanie pomimo tego szybszych odpowiedzi niż klasyczny serwer OSM korzystający z PostgreSQL osadzony na wskazanym komputerze co pokazano podczas testów zaprezentowanych w kolejnym rozdziale. Jak pokazano na obrazku 3.13 rozwiązanie może stanowić szybszą i lżejszą alternatywę względem obecnie stosowanych rozwiązań w przypadku konieczności ich użycia na serwerach o niewielkich zasobach eliminując potrzebę kupowania lub opłacania zewnętrznego serwera obliczeniowego (optymalizacja kosztów dla projektów o niskim budżecie).

Rysunek 3.11: Wskazanie miejsca na mapie wykorzystanego jako przykład rezultatu zastosowanej klasteryzacji. Pozostałe rysunki zawierają ten wycinek mapy w dużym powiększeniu.



Rysunek 3.13: Render mapy korzystający z 1565 nodów (stan po klasteryzacji). Wizualizacja mapy z dużą wiernością.

Rysunek 3.12: Render niewielkiego fragmentu mapy korzystający z wszystkich tj. 1773 nodów (stan przed klasteryzacją).

Rysunek 3.14: Render mapy korzystający z zaledwie 1043 nodów (dalsze próby zwiększenia obszaru klasteryzacji- mapa prezentuje niemal wyłącznie ogólne przeznaczenie określonych obszarów. Niższy czas renderingu i zajętości pamięci osiągnięte kosztem szczegółowości).

4

Eksperymenty i weryfikacja działania systemu

W tym rozdziale, rozwiązania zaproponowane w rozprawie zostały zweryfikowane eksperymentalnie pod kątem prawidłowości i szybkości działania. Zakończenie prac nad elementami składowymi systemu umożliwiło rozpoczęcie integracji, czyli stworzenia prototypu systemu generującego mapy obszaru na podstawie wielomodalnych danych przestrzennych i czasowych. Uruchomienie prototypu było niezbędne do weryfikacji prawidłowości działania wszystkich modułów i potwierdzenia prawdziwości tez określonych w rozprawie. Proces integracji był iteracyjny i polegał na uruchamianiu komunikacji pomiędzy każdym elementem oraz przeprowadzenia gruntownych testów pod kątem spełnienia pierwotnych założeń. Głównym celem przeprowadzonych w tym rozdziale badań było porównanie działania części składowych opracowanego systemu z wybranymi obecnie stosowanymi rozwiązaniami.

Z uwagi na wąską specyfikę projektu rezultaty zostały poddane weryfikacji poprzez szereg eksperymentów wykonanych w środowisku testowym, a następnie produkcyjnym. Dokładność działania została określona w stosunku do pomiarów referencyjnych wykonanych w terenie. System, który powstał na potrzeby eksperymentów jest uruchamiany w komercyjnie wykorzystywanej platformie.

4.1 Algorytm do estymacji odległości na podstawie widoku z kamer

Algorytm estymacji odległości został przetestowany we Wrocławiu na losowo wybranym skrzyżowaniu (którego koordynaty to 51.09631717602039,

17.041466258291837). Model detekcji YoloV4, który został wykorzystany na mieście został wytrenowany wykorzystując transfer learning ogólnie dostępnego modelu do detekcji 80 klas COCO, a następnie wytrenowano go do rozpoznawania tablic rejestracyjnych. Znając wymiary tablic rejestracyjnych obowiązujących w Polsce (które wynoszą 520 x 114 mm), ogniskową kamery oraz szerokość ramki ograniczającej (bounding box), można obliczyć odległość pomiędzy ogniskową kamery a obiektem, wykorzystując zasadę podobieństwa trójkątów. Dzięki ustandaryzowanym rozmiarom tablic, rezultaty przeprowadzonego eksperymentu pozwalają na dokładne określenie błędu estymacji będącej rezultatem zaimplementowanego algorytmu (w przypadku mierzenia odległości do przedmiotów o niestandardowych wymiarach takich jak monitory komputerowe, wyniki mogą znacznie się od siebie różnić). Dzięki wykorzystaniu kamer zainstalowanych na skrzyżowaniu możliwe było określenie błędów pomiarowych jakie algorytm osiąga w środowisku produkcyjnym. Kamera umieszczona na jednym ze skrzyżowań (model HIKVISION DS-2CD5026G0-AP) posiadała przetwornik o rozmiarze 1/2,8" i ogniskową 50mm co zostało wprowadzone jako dane wejściowe do algorytmu. Dla testów prawidłowości działania pozostałych komponentów zbudowanego systemu, proces transfer learning przeprowadzono za pośrednictwem zaimplementowanego modułu automatycznego trenowania modeli (ATM). Do treningu i walidacji zebrano i poetykietowano ponad 200 obrazów pochodzących z kamery na skrzyżowaniu (na każdym z nich tablica rejestracyjna wystąpiła co najmniej 3 razy). Moduł ATM zwrócił model o najwyższym AP 2.2.1 jakie zostało uzyskane podczas treningu (78.98). Trening był wykonany z współczynnkiem uczenia 0.001 oraz wielkością kroku 64 z podziałem (ang. subdivision) na 8 wykorzystując kartę graficzną RTX 3090. Uzyskany rezultat porównano z ogólnie dostępną siecią, która rozpoznawała tablice rejestracyjne w celach odczytywania ich numerów poprzez OCR. Wyniki 4.1 wskazują, że proces przebiegł pomyślnie, zbudowany zbiór danych był wystarczający, a sieć stała się dokładniejsza uzyskując znacznie lepsze rezultaty detekcji.

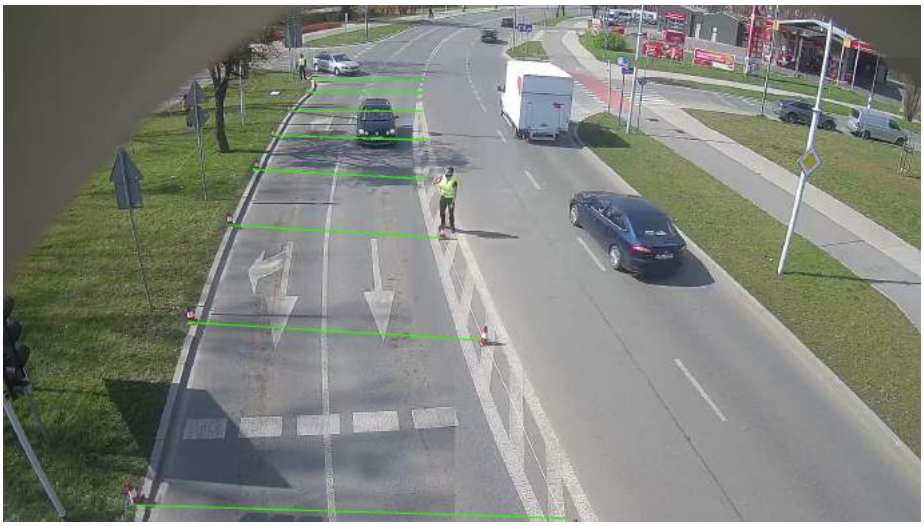
Opis	AP (in %)	IoU (in %)
Pretrenowane wagi do detekcji tablic rejestracyjnych dla OCR	56,30	51,42
Najlepszy rezultat po dotrenowaniu sieci własnymi danymi	78,98	55,65

Tabela 4.1: Porównanie rezultatu ogólnie dostępnej sieci wytrenowanej do detekcji tablic z własną wytrenowaną na stworzonym zbiorze pochodzącym z środowiska produkcyjnego.

W celach weryfikacji dokładności uzyskanych rezultatów z algorytmu estymacji odległości, przeprowadzono pomiary w terenie, by określić rzeczywistą odległość danych obiektów od kamery. Pomiary zostały naniesione na obraz w formie linii (co 5m), których koordynaty umożliwiały odniesienie do rezultatów pracy algorytmu. Znając wysokość na jakiej jest ustawiona kamera oraz jej parametry wewnętrzne, pochodzące z niej nagranie zostało przetworzone pod kątem detekcji tablic rejestracyjnych. Do weryfikacji prawidłowości estymacji zostały wzięte pod uwagę wyłącznie te detekcje, których centralny punkt bounding boxu był zbliżony do koordynat linii odległości zmierzonych w terenie i naniesionych na obraz (dopuszczono max +/- 5px różnicy) 4.1. Z przeprowadzonych w ten sposób eksperymentów wynika, że algorytm wykazywał wysoką dokładność estymacji (maks. 15% błąd przy określaniu odległości do obiektu oddalonego o 45m). Uzyskane rezultaty przedstawione w tabeli 4.2 wskazują, że w odległości 5m od kamery detekcja nie była możliwa z uwagi na duże zniekształcenie obrazu (obraz tablicy rejestracyjnej dla takiej odległości był mocno zniekształcony w wyniku rzutowania perspektywicznego kamery). Dla pozostałych odległości zebrano kilkanaście detekcji, a wyniki przestawione w tabeli zawierają średnią odległość jaka została wyznaczona przez algorytm dla danego obszaru.

Koordynaty linii px	Dystans [m]				Błąd (wyrażony w %)
	Min.	Maks.	Estymacja	Wartość referencyjna	
86	43.89	47.54	45	53	15
118	35.66	43.89	40	45	11
148	35.66	40.75	35	40	14
186	31.7	33.56	30	34	12
238	24.8	25.93	25	28	11
336	17.83	21.94	20	20	0
444	15.85	16.3	15	16	6
660	10.97	11.18	10	10	0
969	-	-	5	5,5	9

Tabela 4.2: Porównanie rezultatów pomiarów referencyjnych wykonanych w terenie z estymacją odległości dla danego obszaru wyznaczoną na podstawie algorytmu.



Rysunek 4.1: Pomiary odległości wykonane w terenie.

Sposób działania algorytmu estymacji odległości jest identyczny dla urządzeń mobilnych z systemem Android i iOS. Uzyskanie potwierdzenia prawidłowości działania wskazanego algorytmu w środowisku produkcyjnym, pozwoliło na prowadzenie dalszych badań nad algorytmem określania dokładnej lokalizacji na podstawie danych z kamery, czujnika IMU wewnątrz urządzenia mobilnego i możliwego błędu sygnału wskazywanego przez moduł GPS w chwili detekcji obiektu.

4.2 Prototyp aplikacji mobilnej dla systemu Android i iOS. Określenie lokalizacji obiektów na podstawie widoku kamery, modułu GPS oraz czujnika IMU.

Aplikacja mobilna przesyłała dane o wykrytych w widoku kamery obiektach na serwer wraz z informacją o ich estymowanym położeniu. Dokładna lokalizacja obiektów jest określana na podstawie podobieństwa trójkątów, ogniskowej kamery smartfona, wielkości sensora aparatu, położenia telefonu i średniej wielkości danej klasy obiektu jaka jest zapisana w pobranej z serwera konfiguracji (np. dla monitora jest to 40cm wysokości i 60cm szerokości). Przykład działania aplikacji w środowisku testowym wraz z przesłanymi danymi pokazano na rysunku 4.2.

Rysunek 4.2: Przykład detekcji obiektów z poziomu aplikacji uruchomionej na smartfonie z systemem iOS (uruchomionej w trybie Debug- użytkownik końcowy nie widzi (bounding boxów))

```
Obliczona przez serwer odległość pomiędzy obserwatorem,
a monitorem (w metrach): 1.1471090102514463


Dane na serwer:
{
    "device": {
        "longitude": 22.020109749038419,
        "latitude": 49.684358795315347,
        "gpsPrecisionInMetres": 12
        },
    "resultList": [{
        "class": "tvmonitor",
        "score": 0.99751055240631104,
        "duplicate": false,
        "object": {
            "longitude": 22.020114538072754,
            "latitude": 49.68434895561978
        },
        (...)
    }]
}
```

Rysunek 4.3: Przykład danych jakie zostały przesłane na serwer

Po wykonaniu testów w środowisku laboratoryjnym, uruchomiono również testy w środowisku produkcyjnym. Aplikacja została uruchomiona w trybie debug co umożliwiło prezentację ramki ograniczającej znajdywane obiekty (bounding boxy) na wyświetlaczu. Zeskanowano telefonem Samsung A40 szereg obiektów będących elementami nie zmieniającymi swojego położenia w mieście takich jak ławki w parkach czy hydranty. Celem była weryfikacja prawidłowości działania algorytmu doprecyzowywania lokalizacji obiektów działającego bezpośrednio na urządzeniu mobilnym. Poza estymacją lokalizacji, zweryfikowano również działanie kategoryzacji danego obiektu jako duplikatu bazując na przechowywanej lokalnie na urządzeniu liście ostatnich detekcji. Wykonany test polegał na kilkukrotnym zeskanowaniu obiektów w mieście, a następnie weryfikacja jakie dane urządzenie przesłało na serwer 4.4. Algorytm poprawnie wykrył, zaklasyfikował jako duplikat oraz doprecyzował lokalizację obiektu bazując wyłącznie na danych przechowywanych lokalnie na urządzeniu. Rezultat pracy wraz z surowymi danymi zaprezentowano na schemacie 4.5.

Rysunek 4.4: Przykład działania algorytmu doprecyzowywania lokalizacji na podstawie lokalnie przechowywanych przez urządzenie danych o ostatnich detekcjach.

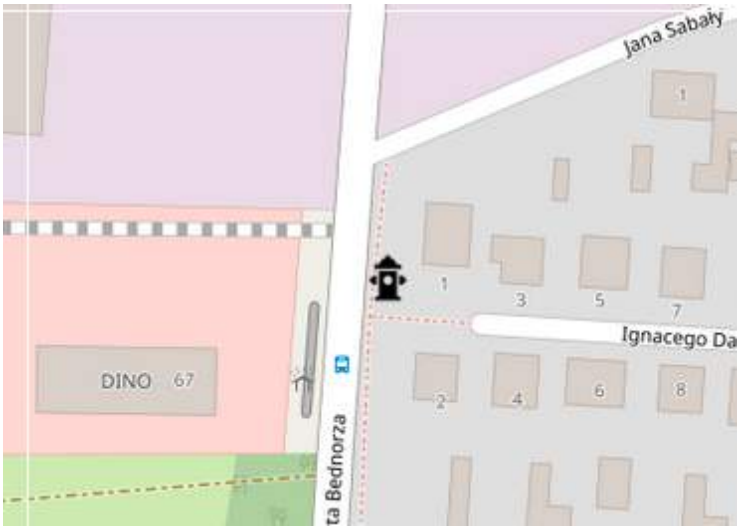

```
Dane przesłane na serwer:
(...)
"class": "fire hydrant",
"score": 0.6563,
"duplicate": false,
"object": {
    "longitude": 18.98132407106459,
    "latitude": 50.354586937464774
}
(...)
```


```
Dane przesłane na serwer:
(...)
"class": "fire hydrant",
"score": 0.7656,
"duplicate": true,
"object": {
    "longitude": 18.981318237210915,
    "latitude": 50.354568058520174

}
(...)
```


```
Dane przesłane na serwer:
(...)
"class": "fire hydrant",
"score": 0.6094,
"duplicate": true,
"object": {
    "longitude": 18.981320774182677,
    "latitude": 50.35457252059132
}
(...)
```

Rysunek 4.5: Surowe dane przetworzone przez aplikację, naniesione na mapę. Na schemacie pokazano również lokalizację doprecyzowaną bezpośrednio na urządzeniu. Odległość pomiędzy dwoma skrajnymi wartościami detekcji wynosi około 3m



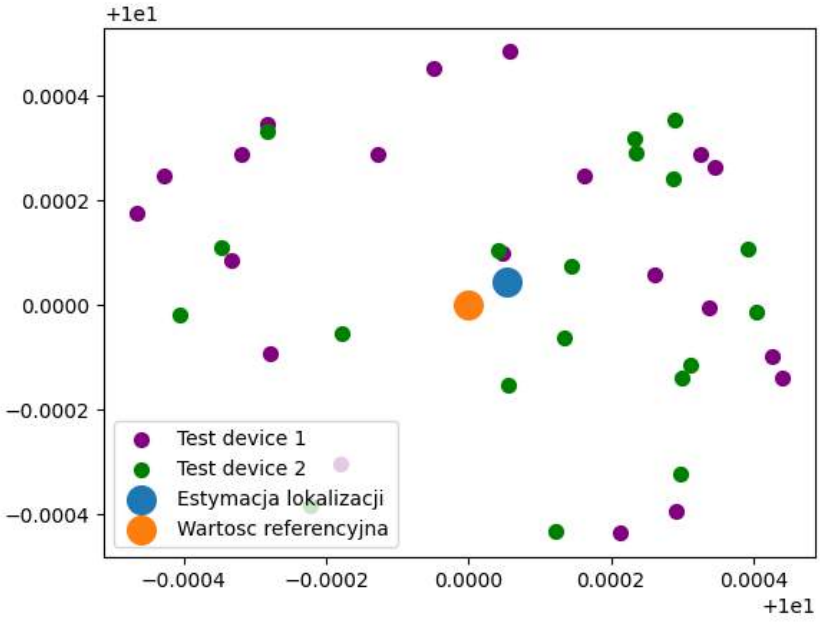
Rysunek 4.6: Wizualizacja pokazująca znaleziony obiekt na mapie.

4.3 Algorytm integrujący pomiary położenia obiektów z wielu źródeł

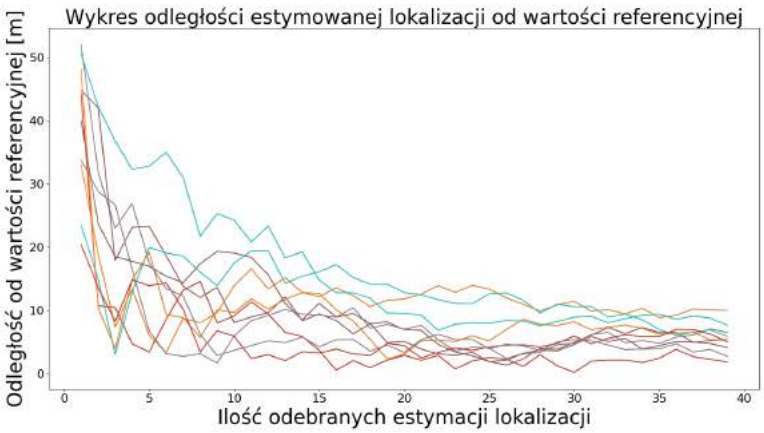
Podstawą do potwierdzenia poprawnego działania algorytmu doprecyzowującego położenie obiektów w przestrzeni było przeprowadzenie szeregu eksperymentów w warunkach kontrolowanych. Testy polegały na wybieraniu wartości referencyjnych (różnych szerokości i wysokości geograficznych), a następnie wylosowanie dokładności symulowanego modułu GPS (wartości od 2 do 20m) i koordynat położenia obiektu wskazywanego przez urządzenie (w odległości maksymalnej 50m + wartość wylosowanego błędu modułu GPS). Przeprowadzono w ten sposób kilkaset symulacji różniących się liczbą pomiarów (przetwarzając 5-500 detekcji). We wszystkich przypadkach odległość pomiędzy punktem referencyjnym, a estymacją odległości wyliczoną przez algorytm zmniejszała się wraz ze zwiększającą się ilością pomiarów 4.9. Przykład symulacji pokazano na schemacie 4.8.

Wartością referencyjną w prezentowanych testach były współrzędne 10 °N i 10 °E. Następnie wylosowano 40 lokalizacji obiektu i wraz z wylosowaną dokładnością GPS wysłano je na serwer celem przetworzenia przez algorytm doprecyzowania lokalizacji. Odległość od punktu referencyjnego malała wraz z kolejnymi nadsyłanymi pomiarami uzyskując dokładność wynoszącą poniżej 10m po przetworzeniu 33 pomiarów. Dla wszystkich testowanych przypadków dokładność poniżej 10m była uzyskiwana po przetworzeniu minimum 14 pomiarów (ze średnią wartością 36). Wykonane eksperymenty na danych syntetycznych wskazują, że teza *"fuzja danych o obiektach wykrywanych na obrazie, parametrach wewnętrznych kamery, informacji pochodzących z IMU oraz sygnału GPS pozwala na lokalizację obiektów 3D z dokładnością do 10m względem ich wartości referencyjnej"* jest prawdziwa po przetworzeniu przez system kolejnych pomiarów (ze średnią wartością 36) nadesłanych przez urządzenia brzegowe.

Rysunek 4.7: Wizualizacja estymacji odległości lokalizacji od wartości referencyjnej dla danych testowych podczas jednej z przeprowadzonych symulacji.



Rysunek 4.8: Wizualizacja testowych koordynat lokalizacji obiektu nadsyłanych do systemu. Wraz z koordynatami na serwer przesyłano dokładność GPSa dla każdego pomiaru.



Rysunek 4.9: Rezultaty 10 przykładowych symulacji. Odległość od punktu referencyjnego malała wraz z liczbą odebranych od urządzeń brzegowych estymacji lokalizacji.

Poza testami na danych syntetycznych, przeprowadzono testy statystyczne na bazie próby losowej na danych pochodzących z środowiska produkcyjnego. Weryfikacja normalności rozkładu uzyskiwanych błędów, rozumianych jako bezwzględna różnica lokalizacji referencyjnej i wyznaczonej przez algorytm działający lokalnie na urządzeniu, przy użyciu testu Shapiro-Wilka, wykazała, że dane nie pochodzą z rozkładu normalnego. Dane składały się z 1150 pomiarów (pochodzących z estymacji odległości dla 10 różnych klas obiektów). Z uwagi na powyższe zastosowano parametryczny test istotności dla wartości oczekiwanej w przypadku, gdy cecha ma dowolny rozkład o nieznanej ale skończonej wariancji i przy dużej próbie losowej (jej liczebność większa niż 100)[79].

Przyjęto następującą postać hipotez zerowej H0 i alternatywnej H1:

H_0: E(t)=m0=10m (wartość oczekiwana rozkładu błędu będącego odległością pomiędzy wartością referencyjną, a estymowaną **wynosi** 10m).

H_1: E(t)=m0<10m (wartość oczekiwana rozkładu błędu będącego odległością pomiędzy wartością referencyjną, a estymowaną **jest mniejszy** niż 10m)

Następnie policzono statystykę U:

$U = \frac{\overline{X} - m_0}{S/\sqrt{n}}$

$\overline{X}$ - średnia arytmetyczna

m_0 - zakładany błąd będący odległością pomiędzy wartością estymowaną na

urządzeniu, a wartością referencyjną

S - odchylenie standardowe

n - liczba pomiarów

Gdzie:

$\overline{X} = 6.704$

$m_0 = 10$

$S = 2.662$

$n = 1150$

$U = -15.761553307381021$

W związku z przyjętą postacią hipotezy alternatywnej zbiór krytyczny ma postać:

$\langle -\infty, -u_{(1-\alpha)} \rangle$

tj. $\langle -\infty, -1,6449 \rangle$

Gdzie u to kwantyl rzędu 1-alfa rozkładu normalnego, a alfa-0.05 to poziom istotności testu.

Ponieważ statystyka testowa ma wartość spoza obszaru krytycznego H_0 została odrzucona uznając H_1 za prawdziwą. Uzyskana wartość statystyki testowej potwierdziła prawdziwość tezy, że **algorytm wykonujący lokalnie na urządzeniu fuzję danych o obiektach wykrywanych na obrazie, parametrach wewnętrznych kamery, informacji pochodzących z IMU oraz sygnał GPS umożliwia lokalizację obiektów 3D z dokładnością do 10m względem wartości referencyjnej w warunkach produkcyjnych**.

Dalsze eksperymenty miały na celu określenie dokładności działania algorytmu integrującego pomiary nadsyłane przez wiele urządzeń i doprecyzowującego lokalizację obiektów w środowisku produkcyjnym. W tym celu opracowano scenariusz testów, który wyglądał następująco:

1. Uruchomienie na smartfonie prototypu aplikacji. Zeskanowanie obiektu na telefonie (skutkującego przesłaniem informacji o jego estymowanej lokalizacji na serwer).

2. Zasłonięcie obiektywu. Zmiana lokalizacji urządzenia oraz ponowne zeska-

nowanie obiektu. Powtórzenie tego kroku wielokrotnie.

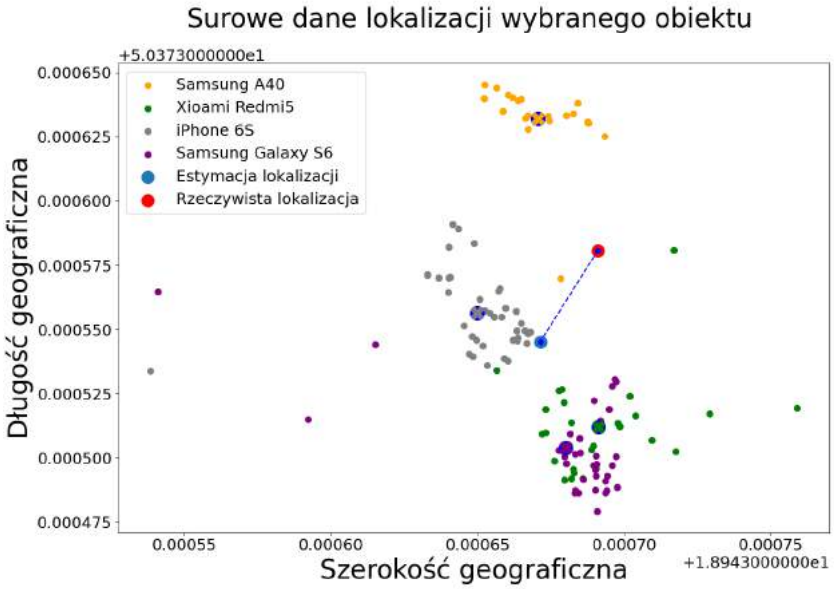
3. Powtórzenie detekcji z wykorzystaniem innego urządzenia.

4. Weryfikacja dokładności uzyskanego wyniku względem wartości referencyjnej.

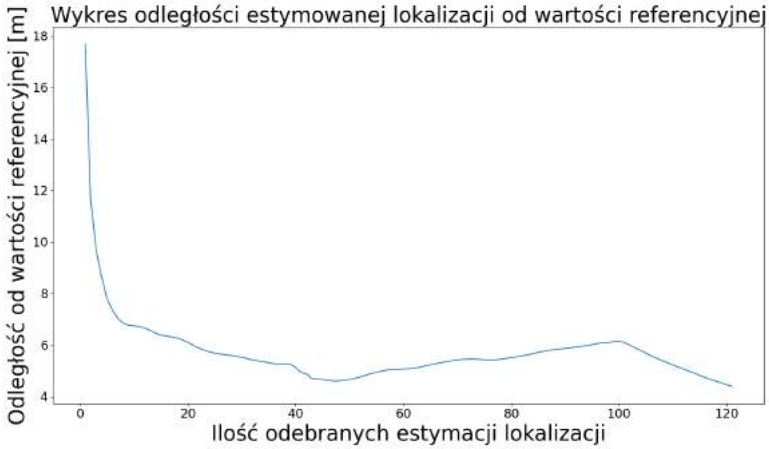
W pierwszej kolejności, dla celów testów, zmieniano klasyfikację skanowanych klas obiektów z ruchomych na nieruchome, by sprawdzić zbieżność dokonywanych pomiarów. Zabieg zmiany klasyfikacji był niezbędny z uwagi na fakt, że obiekty ruchome zmieniają swoje położenie w systemie niezależnie od lokalizacji w chwili wykrycia, że są duplikatami (po stronie urządzenia lub serwera). Wartością referencyjną dokonywanych pomiarów było umieszczenie obiektów w miejscu gdzie wcześniej wykonywano pomiary dokładności modułów GPS przy użyciu urządzenia Leica 1200 GPS. W jednej z lokalizacji w Piekarach Śląskich umieszczono kilka klas obiektów tj. laptopa, jabłko i butelkę soku oraz rozpoczęto ich detekcję z wielu stron z kilku urządzeń. Na schemacie pokazano estymowaną lokalizację obiektu na podstawie pomiarów z każdego z urządzeń osobno oraz wszystkich zebranych w systemie (będącym rezultatem działania algorytmu doprecyzowania lokalizacji) 4.10. Różnica pomiędzy estymacją lokalizacji (wyliczoną na podstawie wszystkich otrzymanych danych), a wartością referencyjną odległość wynosiła 4.33m.

Przeprowadzone eksperymenty algorytmu integrującego dane z urządzeń brzegowych wskazują, że doprecyzowywanie lokalizacji obiektów na mapie jest procesem zbieżnym. Zostało to potwierdzone zarówno podczas symulacji jak i faktycznych pomiarów wykonanych w warunkach rzeczywistych. Wraz z rosnącą liczbą nadesłanych lokalizacji estymowana odległość do punktu referencyjnego zmniejsza się. Przykład zbieżności zaobserwowanej w środowisku produkcyjnym pokazano na schemacie 4.11.

Na podstawie przeprowadzonych eksperymentów dokonano porównania otrzymanych błędów wyznaczania pozycji obiektów rozumianych jak bezwzględna różnica lokalizacji referencyjnej i wyznaczonej przez opracowywany system na bazie danych wielomodalnych w wariantach P10, P20, P30, P40, P50, P75 i



Rysunek 4.10: Schemat wizualizujący wszystkie detekcje pokazane na mapie. Pomiędzy estymacją lokalizacji (wyliczoną na podstawie wszystkich otrzymanych danych), różnica pomiędzy wartością wyznaczoną a referencyjną 4.33m



Rysunek 4.11: Wykres prezentujący odległość estymacji lokalizacji obiektu w środowisku produkcyjnym od punktu referencyjnego.

P100 uwzględniających odpowiednio 10, 20, 30, 40, 50, 75 i 100 pomiarów z urządzeń końcowych tego samego obiektu. Postawiono następujące hipotezy zerowe i alternatywne:

H_0: E(PN_1)=E(PN_2) – oczekiwane błędy wyznaczania pozycji obiektu dla wariantów uwzględniających N_1 i N_2 pomiarów z urządzeń końcowych są takie same.

H_1: E(PN_1)>E(PN_2): oczekiwany błąd wyznaczania pozycji obiektu dla wariantów uwzględniających N1 pomiarów jest większy niż dla wariantu z N_2 pomiarów.

Do weryfikacji postawionych hipotez i porównania obu populacji wybrano test U Manna-Whitneya. Podczas testu porównano kilka podzbiorów różnicy odległości pomiędzy wartością referencyjną, a estymacją lokalizacji wyliczaną po stronie serwera. Uzyskano następujące wyniki:

Liczba pomiarów w populacji		p-wartość
N_1	N_2	
10	20	6.602529956717721e-06
20	30	0.009598848737399742
30	40	0.0033702720190514194
40	50	0.03989005028283395
50	75	0.0034084938254031476
75	100	0.015017753124689283
10	100	4.63904559853132e-29

Tabela 4.3: Tabela z wynikami przeprowadzonego testu U Manna-Whitneya.

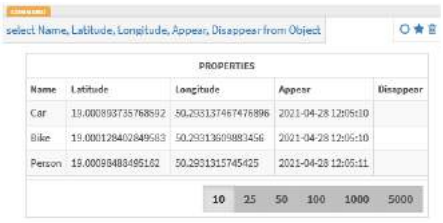
We wszystkich wariantach uzyskana p-wartość była mniejsza niż 0.05, a poza czwartym porównaniem N_1=40 i N_2=50 nawet mniejsza niż 0.02 co pokazano w tabeli 4.3, stąd na przyjętym poziomie istotności 0.05 odrzucamy wszystkie hipotezy zerowe i przyjmujemy alternatywne. W związku z przyjęciem hipotez alternatywnych dla rozpatrywanych wariantów postawiana teza o rosnącej dokładności wyznaczonej lokalizacji obiektów wraz ze wzrostem pomiarów urządzeń końcowych jest prawdziwa. Uzyskane wyniki potwierdzają prawdziwość tezy określonej we wstępie, że **integracja wielu pomiarów położenia obiektów 3D poprawia dokładność ich lokalizacji**.

Poza funkcją doprecyzowania lokalizacji na podstawie informacji nadsyłanych od użytkowników, algorytm działający po stronie serwera porównuje otrzymane z urządzenia dane z aktualnie dostępnymi na serwerze. W przypadku wykrycia, że użytkownik pomimo niewielkiej odległości od obiektu (zdefiniowanej indywidualnie dla każdej klasy obiektów osobno) nie znajduje wokół siebie danego obiektu otoczenia, serwer usuwa obiekt z wizualizacji użytkownika (zapewniając odświeżanie aktualnego stanu mapy w czasie rzeczywistym) jak również zapisuje do bazy danych informację o jego zniknięciu z danego obszaru.

Testy wskazanej funkcjonalności zostały potwierdzone w środowisku laboratoryjnym i produkcyjnym. Testy w warunkach kontrolowanych skupiały się na przesyłaniu kilku obiektów i odejmowanie z listy losowego z nich. Po potwierdzeniu prawidłowości działania algorytmu dla wygenerowanych danych, przeprowadzono również testy w środowisku produkcyjnym. Polegały one na kilkukrotnej detekcji obiektu w danej lokalizacji z poziomu pojedynczego telefonu, a następnie zasłonięciu obiektywu oraz przemieszczeniu obiektu poza widok kamery smartfona oraz rozpoczęcie ponownej detekcji (sieć neuronowa na telefonie nie wykonuje detekcji jeśli warunki oświetleniowe są nieodpowiednie wykorzystując funkcję określającą poziom jasności pixeli wymagany do przechwycenia obrazu kamery). Efekt działania algorytmu scalającego dane zweryfikowano uruchamiając wizualizację na prototypie platformy oraz analizując dane zapisane w bazie danych.

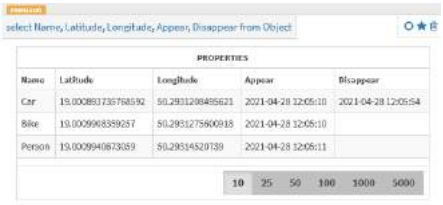
W pierwszej kolejności algorytm detekcji wykrył w obrazie kamery na urządzeniu osobę, samochód oraz rower 4.12. Po przesłaniu tych informacji na serwer, samochód odjechał przed ponowną detekcją obiektów Ponowna detekcja pozwoliła na wykrycie tylko dwóch klas obiektów (osoba, rower) z trzech klas (osobę, samochód oraz rower) Umożliwiło to wpisanie do bazy danych chwili zniknięcia samochodu oraz jego usunięcie z wizualizacji na mapie 4.14 potwierdzając tym samym prawidłowość działania mechanizmu w warunkach rzeczywistych.

Rysunek 4.12: Wizualizacja mapy z chwili otrzymania przez serwer informacji o detekcji trzech obiektów we wskazanych lokalizacjach



Rysunek 4.13: Zrzut obiektów zapisanych w bazie danych po detekcjach.

Rysunek 4.14: Wizualizacja mapy po wykryciu przez algorytm braku jednego z obiektów we wskazanej lokalizacji



Rysunek 4.15: Zrzut obiektów zapisanych w bazie po wykryciu zniknięcia jednego z nich.

4.4 Testy modułu zwracającego dane do renderingu mapy

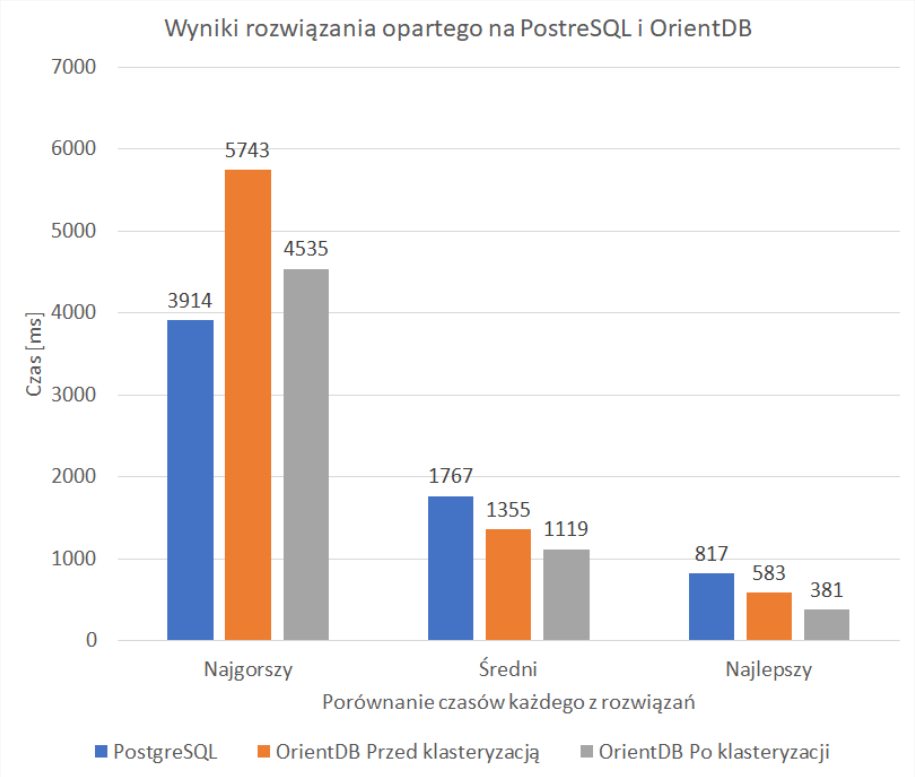
Podczas uprzednio przeprowadzonych badań potwierdzono prawidłowość działania systemu dostarczania informacji niezbędnych do renderingu mapy. Porównywano mapy wyrenderowane na podstawie oryginalnych danych dostępnych w PostgreSQL z mapami wyrenderowanymi na podstawie OrientDB, potwierdzając prawidłowość odwzorowania elementów na mapie. W ramach dalszych testów skupiono się więc na porównaniu szybkości oryginalnego rozwiązania

z rozwiązaniem opracowanym w rozprawie.

Testy polegały na pobraniu najnowszej mapy województwa opolskiego z oficjalnej dystrybucji OSM, a następnie zainicjalizowaniu baz danych (PostgreSQL oraz OrientDB) bezpośrednio z pobranego pliku binarnego. W pierwszym przypadku wykorzystano pakiet osm2pgsql, a w drugim własne oprogramowanie zaimplementowane w języku Python, bazujące na bibliotece pyorient do integracji z OrientDB oraz pakietu esy.osm.pbf do parsowania pobranego pliku. Po zainicjalizowaniu obydwu baz, zaczęto odpytywać je o losowe obszary o stałym rozmiarze 1x1 i 2x2 km^2 należące do województwa opolskiego weryfikują czas odpowiedzi obydwu rozwiązań. Wybór rozmiaru wynikał ze sposobu wdrożenia i wykorzystania opracowanego rozwiązania jako systemu wizualizacji obiektów na mapie w wybranym obszarze w czasie rzeczywistym. Większe obszary zostały zablokowane dla wskazanej funkcjonalności z uwagi na fakt rozmiaru danych jakie musiałaby być prezentowana równocześnie wewnątrz przeglądarki. Nieustanne dodawanie i usuwanie obiektów z większego obszaru wiązałoby się z koniecznością przetworzenia ogromnej liczby obiektów na sekundę. Wskazane ograniczenie zapewnia płynność działania zbudowanej wizualizacji i nie wymaga od użytkownika posiadania mocnej specyfikacji sprzętowej.

W celu zapewnienia miarodajności wyników, oba rozwiązania zostały uruchomione na maszynie wirtualnej dysponującej 2GB pamięci RAM, 30 GB SSD oraz 1 rdzeniem procesora i7-10510U. Przyjęte w eksperymenie wielkości zasobów miały za zadanie uwydatnić znaczne różnice w czasie działania obu rozwiązań w przypadku maszyn o niskich zasobach, bez dużej pamięci RAM i wielordzeniowego procesora. Testy działania zaproponowanego rozwiązania przeprowadzone na wirtualnej maszynie o niewielkich zasobach miały na celu także potwierdzenie możliwości jego wykorzystania w projektach o niskim budżecie, gdzie dedykowany serwer obliczeniowy stanowi dla klienta zbyt kosztowną inwestycję. Docelowe rozwiązanie miało możliwie szybko przetwarzać i zwracać dane, będąc przy tym możliwym do osadzenia na słabym komputerze (o wskazanych wyżej parametrach) klasy PC. Dodatkowo, czas odpowiedzi OrientDB został zmierzony przed oraz po klasteryzacji danych w nim zawartych.

Po odpytaniu obu rozwiązań o dane niezbędne do renderingu mapy dla kilkuset losowych koordynat i zmierzaniu czasu odpowiedzi zebrane dane zostały posortowane rosnąco i malejąco, określając czas najszybszej i najwolniejszej odpowiedzi systemu. Zapytania były wykonywane sekwencyjnie natychmiast po odebraniu prawidłowej odpowiedzi. Wyniki najgorszego, najlepszego oraz uzyskanego średniego czasu przedstawiono na wykresie 4.16.



Rysunek 4.16: Wykres porównujący czasy odpowiedzi [ms] rozwiązania opartego na PostgreSQL oraz OrientDB na zapytania dotyczące informacji niezbędnych do renderingu losowego fragmentu mapy (wykonano 1000 zapytań).

Wyniki wskazują, że największy czas był wyższy w zaproponowanym rozwiązaniu, jednak średni i najlepszy czas był niższy o około 25% dla bazy przed i około 35% po klasteryzacji na testowanej maszynie wirtualnej. Udowodniono w ten sposób prawdziwość ostatniej z tez wskazanych we wstępie tj. *"Zastąpienie systemu zarządzania bazą danych PostgreSQL w systemie Open Street Map na grafowy system zarządzania bazą danych w postaci OrientDB znacznie*

przyśpiesza proces wyszukiwania informacji niezbędnych do renderingu mapy". Osiągnięte wyniki pozwalają na skrócenie czasu obsługi pojedynczego zapytania o informacje dotyczące wybranego obszaru wymagającego wyrenderowania po stronie urządzenia końcowego.

5

Podsumowanie i wnioski

W niniejszej rozprawie zaproponowano własne podejście do budowania map 2D na podstawie temporalnych, wielomodalnych danych przestrzennych i czasowych. We wstępnie określono następujące tezy:

- Fuzja danych o obiektach wykrywanych na obrazie, parametrach wewnętrznych kamery, informacji pochodzących z IMU oraz sygnału GPS pozwala na lokalizację obiektów 3D z dokładnością do 10m względem wartości referencyjnej.

- Integracja wielu pomiarów położenia obiektów 3D poprawia dokładność ich lokalizacji.

- Głębokie sieci neuronowe działające na urządzeniach mobilnych opartych o architektury ARMv8-A i nowsze (ARMv8.3-A, ARMv8.4-A) pozwalają na efektywną detekcję obiektów na obrazie z prędkością o wartości oczekiwanej nie mniejszej niż 3 klatki na sekundę.

- Zastąpienie systemu zarządzania bazą danych PostgreSQL w systemie Open Street Map na grafowy system zarządzania bazą danych w postaci OrientDB znacznie przyśpiesza proces wyszukiwania informacji niezbędnych do renderingu mapy.

Zbudowana dla urządzeń mobilnych, opartych na systemach Android i iOS, aplikacja posiada zaimplementowany algorytm, który na podstawie fuzji informacji o obiektach wykrytych na obrazie, parametrach wewnętrznych kamery, sygnałów z czujnika IMU oraz lokalizacji i marginesie błędu modułu GPS estymuje

lokalizację obiektu widocznego w widoku kamery. Testy algorytmu estymacji odległości w środowisku produkcyjnym wykazały dokładność jego działania na poziomie 18% dla odległości 50m w przypadku wykrywania tablic rejestracyjnych, które posiadają ustandaryzowane rozmiary umożliwiające dokładne określenie błędu estymacji.

Przeprowadzone eksperymenty dokładności działania aplikacji pokazały, że wielokrotna detekcja oraz doprecyzowanie lokalizacji na urządzeniu pozwala na uzyskanie zakładanej dokładności tj. do 10m od wartości referencyjnej (rzeczywistej wartości zmierzonej urządzeniem Leica 1200 GPS) po przetworzeniu minimum 14 pomiarów. Badania zostały przeprowadzone w środowisku produkcyjnym, na terenie miasta Piekary Śląskie.

Poza określaniem lokalizacji obiektów na urządzeniu, po stronie serwera opracowano algorytm, który scalał nadsyłane przez użytkowników wyniki celem ich integracji i zapisywania czasu kiedy dane obiekty pojawiały się i znikały z danego obszaru. Integracja pomiarów po stronie serwera pozwala na uzyskanie lepszych rezultatów względem pojedynczych pomiarów przeprowadzonych wyłącznie z poziomu pojedynczego urządzenia co zostało udowodnione podczas testów w warunkach produkcyjnych. Dzięki mechanizmowi integracji nadsyłanych danych utworzono narzędzie, które poza danymi o lokalizacji dostarcza także dokładne dane o czasie pojawienia się danego obiektu, możliwej wizualizacji zmian jakie zachodziły na przestrzeni ostatnich minut, godzin, dni czy tygodni oraz jakie obiekty aktualnie znajdują się na wybranym obszarze (wizualizacja jest odświeżana w czasie rzeczywistym tj. do 2 sekund od chwili dotarcia informacji na serwer). Dostarcza to nowe narzędzie dla analityków chcących zgłębiać dynamiczne procesy zmian zachodzące w mieście i udowodniło, że *"możliwe jest stworzenie mapy 2D na podstawie wielomodalnych danych przestrzennych i czasowych, przesyłanych z wielu źródeł, która dostarczy informacji o położeniu obiektów statycznych i dynamicznych (tj. zmieniających swoje położenie) znajdujących się na wybranym obszarze"*.

W celu zwiększenia popularności rozwiązania, przebadano szybkość działania i możliwości wykorzystania sieci neuronowych do detekcji obiektów na urzą-

dzeniach mobilnych opartych o architekturę ARMv8-A i nowszych. Uzyskanie czasu około 290 ms na przetworzenie pojedynczej klatki obrazu o rozdzielczości 300x300 px na telefonie Samsung Galaxy S6 umożliwia wdrożenie aplikacji z zakładaną prędkością przetwarzania obrazu na ponad 80% wszystkich urządzeń dostępnych na rynku. Z aktualnych raportów Apple i Google wynika, że aplikacja jest kompatybilna z 84.9% dla Android i 81% dla systemu iOS (stan na styczeń 2021).

Prace nad optymalizacją procesu wyszukiwania danych do renderingu doprowadziły do zastąpienia systemu zarządzania bazą danych z PostgreSQL na OrientDB. Wskazane działanie doprowadziło do zmniejszenia czasu potrzebnego na wyszukanie informacji niezbędnych do renderingu map średnio o 35% w stosunku do oryginalnego rozwiązania, pozbawionego modyfikacji.

Wszystkie prace, poza aspektem badawczym miały na celu stworzenie komercyjnie wykorzystywanego produktu w postaci platformy i aplikacji kierowanej dla szerokiego grona odbiorców. Zbudowany system posiada szereg zastosowań np:

- wytyczania nowych ścieżek rowerowych w miejscach, gdzie z dużą częstotliwością następuje detekcja rowerzystów
- wybrania lokalizacji przeprowadzenia kampanii reklamowej określonego produktu w obszarach gdzie nie wystąpiła jego detekcja (produkt nie jest rozpowszechniony na danym obszarze)
- stworzenie gry terenowej dla turystów będącej promocją danego miasta, która oprowadzi ich po najważniejszych jego lokalizacjach i po wykryciu danego obiektu dostarczy szczegółowych informacji
- tworzenie własnych modeli dedykowanych dla małych firm tworzących oprogramownaie (software house) eliminując konieczność zatrudniania data scientist oraz kupowania lub wynajmowania kosztownej infrastruktury służącej trenowaniu własnych modeli
- opracowywanie rozwiązania, które po autoryzacji w systemie umożliwi uzyskanie nowych wniosków na podstawie zebranych przez system danych

- rendering map oraz wizualizacja stanu obszaru znajdującego się bezpośrednio wokół wybranej inwestycji (np. obszaru zbudowanego odcinka 4.6km drogi ekspresowej S2 wraz z monitoringiem odświeżanego w czasie rzeczywistym stanu wewnątrz 2.6km tunelu).

Wszystkie tezy określone we wstępie zostały potwierdzone i poparte poprzez wykonanie zaprezentowanych w pracy eksperymentów. Uzyskane wyniki pokazują również wyższość poziomu szczegółowości zbudowanej mapy nad ogólnie dostępnymi rozwiązaniami. Co więcej, dzięki implementacji dodatkowych modułów system ma możliwość pozyskiwania od użytkowników danych do tworzenia datasetów i automatycznego trenowania modeli, bez potrzeby ingerencji specjalisty z dziedziny uczenia maszynowego. Wskazane moduły zapewnią nieustanny rozwój projektu po jego wdrożeniu. Wytworzone z pozyskanych datasetów, nowe modele posłużą do mapowania coraz to nowszych obiektów dzięki czemu system będzie nieustannie poszerzał informacje o nowych klasach obiektów występujących w świecie rzeczywistym i określał ich lokalizację na mapie.

System posiada kilka niedoskonałości, które zostały zniwelowane poprzez zastosowanie działań naprawczych minimalizujących ich skutki:

- Brak możliwości określenia dokładnego rozmiaru każdej z rozpoznawanych przez system klas obiektów. Każda klasa uzyskała średnią wartość wysokości i szerokości na podstawie danych z zewnętrznego serwisu (https://www.dimensions.com/). Różne rozmiary np. drzew mogą generować błąd w pomiarze odległości pomiędzy urządzeniem, a danym obiektem. Wskazany błąd jest niwelowany poprzez agregację pomiarów z wielu źródeł i wraz z kolejnymi estymacjami odległości od obiektu wykrywanego z różnych stron błąd maleje co zostało udowodnione podczas jednego z wykonanych eksperymentów.

- Niska dokładność GPS w obszarach pozamiejskich i wewnątrz budynków może generować błąd w określaniu lokalizacji obiektów wykrytych w otoczeniu użytkownika. Wskazany problem jest niwelowany poprzez zapisywanie na serwerze dokładności modułu GPS w chwili detekcji, a następnie

określaniu położenia obiektów wykorzystując pomiary o najwyższej dokładności.

- Redundancja obiektów w bazie jest możliwa przy niskiej jakości sygnału GPS. Działaniem naprawczym niwelującym wskazany efekt jest określenie indywidualnego promienia scalania obiektów danej klasy ze sobą, a ponadto możliwe jest zastosowanie przetwarzania danych przez dodatkowy moduł osadzony w systemie po ich pozyskaniu (ang. post-processing). Wskazany mechanizm dotychczas został wykorzystany i zaprezentowany w pracy jako algorytm do klasyfikacji obszarów na podstawie informacji z bazy danych.

Prezentowane w pracy rezultaty zostały wdrożone w formie platformy umożliwiającej korzystanie z systemu osobie nietechnicznej. Aktualnie trwają prace mające na celu integrację mniej krytycznych elementów systemu (jak mechanizmy zarządzania użytkownikami i systemu monitorowania obciążenia platformy) jednak platforma w aktualnej wersji wraz z aplikacją została przetestowana w środowisku produkcyjnym i wdrożona w trzecim kwartale 2022 roku uzyskując IX poziom gotowości technologicznej.

Bibliografia

[1] M. H. Baig and L. Torresani. Coupled depth learning. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–10. IEEE, 2016.

[2] R. S. Bivand, E. J. Pebesma, V. Gómez-Rubio, and E. J. Pebesma. *Applied spatial data analysis with R*, volume 747248717. Springer, 2008.

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[4] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2544–2550. IEEE, 2010.

[5] M. T. Bui, R. Doskočil, V. Krivanek, T. H. Ha, Y. T. Bergeon, and P. Kutilek. Indirect method to estimate distance measurement based on single visual cameras. In *2017 International Conference on Military Technologies (ICMT)*, pages 695–700. IEEE, 2017.

[6] K. Cao, Y.-T. Peng, and P. C. Cosman. Underwater image restoration using deep networks to estimate background light and scene depth. In *2018 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, pages 1–4. IEEE, 2018.

[7] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015.

[8] S. Choi, D. Min, B. Ham, Y. Kim, C. Oh, and K. Sohn. Depth analogy: Data-driven approach for single image depth estimation using gradient samples. *IEEE Transactions on Image Processing*, 24(12):5953–5966, 2015.

[9] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep learning with cots hpc systems. In *International conference on machine learning*, pages 1337–1345. PMLR, 2013.

[10] I. Cohen and G. Medioni. Detecting and tracking moving objects for video surveillance. In *Proceedings. 1999 IEEE Computer Society Conference on*

Computer Vision and Pattern Recognition (Cat. No PR00149), volume 2, pages 319–325. IEEE, 1999.

[11] E. contributors. EVPP benchmark. `https://github.com/Qihoo360/evpp#benchmark`, 2017. [Online; accessed 4-December-2022].

[12] C. Cortes and V. Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[13] S. Das and N. Ahuja. Performance analysis of stereo, vergence, and focus as depth cues for active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1213–1219, 1995.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[15] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[16] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[17] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[18] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[19] C. Fehn. Depth-image-based rendering (dibr), compression, and transmission for a new approach on 3d-tv. In *Stereoscopic Displays and Virtual Reality Systems XI*, volume 5291, pages 93–104. International Society for Optics and Photonics, 2004.

[20] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[21] D. Forsyth and J. Ponce. *Computer vision: A modern approach.* Prentice hall, 2011.

[22] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[23] R. Furukawa, R. Sagawa, and H. Kawasaki. Depth estimation using structured light flow–analysis of projected pattern flow on an object's surface. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4640–4648, 2017.

[24] R. Garg, V. K. Bg, G. Carneiro, and I. Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European conference on computer vision*, pages 740–756. Springer, 2016.

[25] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[26] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019.

[27] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[28] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 22–29. IEEE, 1998.

[29] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[30] C. Hane, L. Ladicky, and M. Pollefeys. Direction matters: Depth estimation with a surface normal classifier. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 381–389, 2015.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[32] J. Hecht. Lidar for self-driving cars. *Optics and Photonics News*, 29(1):26–33, 2018.

[33] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference Computer Vision (ECCV)*, 2016.

[34] P. Henderson and V. Ferrari. End-to-end training of object class detectors for mean average precision. In *Asian Conference on Computer Vision*, pages 198–213. Springer, 2016.

[35] T. Hengl, P. Roudier, D. Beaudette, E. Pebesma, et al. plotkml: Scientific visualization of spatio-temporal data. *Journal of Statistical Software*, 63(5):1–25, 2015.

[36] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[37] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[38] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. In *ACM SIGGRAPH 2005 Papers*, pages 577–584. 2005.

[39] D. Hoiem, A. A. Efros, and M. Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75(1):151–172, 2007.

[40] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[41] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28:2017–2025, 2015.

[42] A. Joglekar, D. Joshi, R. Khemani, S. Nair, and S. Sahare. Depth estimation using monocular camera. *International journal of computer science and information technologies*, 2(4):1758–1763, 2011.

[43] K. Karsch, C. Liu, and S. B. Kang. Depth transfer: Depth extraction from video using non-parametric sampling. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2144–2158, 2014.

[44] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*, 2017.

[45] J. Konrad, M. Wang, P. Ishwar, C. Wu, and D. Mukherjee. Learning-based, automatic 2d-to-3d image and video conversion. *IEEE Transactions on Image Processing*, 22(9):3485–3496, 2013.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[47] Y. Kuznietsov, J. Stuckler, and B. Leibe. Semi-supervised deep learning for monocular depth map prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6647–6655, 2017.

[48] L. Ladicky, J. Shi, and M. Pollefeys. Pulling things out of perspective. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 89–96, 2014.

[49] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

[50] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[51] S. Lee, G.-H. Ji, and J.-H. Won. Measurement level experimental test result of gnss/imu sensors in commercial smartphones. *Journal of Positioning, Navigation, and Timing*, 9(3):273–284, 2020.

[52] J. Li, R. Klein, and A. Yao. A two-streamed network for estimating fine-scaled depth maps from single rgb images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3372–3380, 2017.

[53] R. Li, S. Wang, Z. Long, and D. Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7286–7291. IEEE, 2018.

[54] X. Li, H. Qin, Y. Wang, Y. Zhang, and Q. Dai. Dept: depth estimation by parameter transfer for single still images. In *Asian Conference on Computer Vision*, pages 45–58. Springer, 2014.

[55] F. Liu, C. Shen, G. Lin, and I. Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):2024–2039, 2015.

[56] M. Liu, M. Salzmann, and X. He. Discrete-continuous depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 716–723, 2014.

[57] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[58] G. LLC. TensorFlow g3doc documentation. `https://github.com/tensorflow/models/tree/master/research/object_detection/g3doc`, 2020. [Online; accessed 19-March-2020].

[59] A. Lukezic, T. Vojir, L. Cehovin Zajc, J. Matas, and M. Kristan. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6309–6318, 2017.

[60] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo. An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2017.

[61] A. Mahalanobis, B. V. Kumar, and D. Casasent. Minimum average correlation energy filters. *Applied Optics*, 26(17):3633–3640, 1987.

[62] S. H. Mao and J. Dally. Deep compression: Compressing deep neural networks with pruning trained quantization and huffman coding. In *International Conference of Learning Representations (ICLR)*, pages 1–14, 2016.

[63] S. Meister, J. Hur, and S. Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[64] T. Narihira, M. Maire, and S. X. Yu. Learning lightness from human judgement on relative reflectance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2965–2973, 2015.

[65] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[66] R. Patnaik and D. Casasent. Fast fft-based distortion-invariant kernel filters for general object recognition. In *Intelligent Robots and Computer Vision XXVI: Algorithms and Techniques*, volume 7252, page 725202. International Society for Optics and Photonics, 2009.

[67] K. Paupamah, S. James, and R. Klein. Quantisation and pruning for neural network compression and regularisation. In *2020 International SAUPEC/RobMech/PRASA Conference*, pages 1–6. IEEE, 2020.

[68] W. Pitts and W. S. McCulloch. How we know universals the perception of auditory and visual forms. *The Bulletin of mathematical biophysics*, 9(3):127–147, 1947.

[69] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun. Dense monocular depth estimation in complex dynamic scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4058–4066, 2016.

[70] A. Roy and S. Todorovic. Monocular depth estimation using neural regression forest. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5506–5514, 2016.

[71] D. Rumelhart. Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, 1, 1986.

[72] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[73] A. Saxena, S. H. Chung, A. Y. Ng, et al. Learning depth from single monocular images. In *NIPS*, volume 18, pages 1–8, 2005.

[74] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840, 2008.

[75] E. Shelhamer, J. T. Barron, and T. Darrell. Scene intrinsics and depth from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 37–44, 2015.

[76] J. Shi, X. Tao, L. Xu, and J. Jia. Break ames room illusion: depth from general single images. *ACM Transactions on Graphics (TOG)*, 34(6):1–11, 2015.

[77] N. Shukla and A. Trivedi. Image based distance measurement technique for robot vision using new lbpa approach. *International Journal of Novel Research in Electrical and Mechanical Engineering*, 2(1):22–35, 2015.

[78] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[79] K. Stąpor. *Wykłady z metod statystycznych dla informatyków z przykładami w języku R*. Wydawnictwo Politechniki Śląskiej, 2015.

[80] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[81] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[82] J. Van Leeuwen. On the construction of huffman trees. In *ICALP*, pages 382–410, 1976.

[83] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011.

[84] S. I. Venieris and C.-S. Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. IEEE, 2016.

[85] S. I. Venieris, A. Kouris, and C.-S. Bouganis. Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions. *arXiv preprint arXiv:1803.05900*, 2018.

[86] X. Wang, D. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 539–547, 2015.

[87] J. Wernecke. *The KML handbook: geographic visualization for the Web*. Pearson Education, 2008.

[88] T. Wilson. Ogc® kml. version 2.2. 0. 2008.

[89] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze. Fastdepth: Fast monocular depth estimation on embedded systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6101–6108. IEEE, 2019.

[90] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[91] J. Xie, R. Girshick, and A. Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European conference on computer vision*, pages 842–857. Springer, 2016.

[92] Z. Yin and J. Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1983–1992, 2018.

[93] X. You, Q. Li, D. Tao, W. Ou, and M. Gong. Local metric learning for exemplar-based object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(8):1265–1276, 2014.

[94] E. A. P. S. S. Zafeiriou, I. K. R. A. Güler, and G. Trigeorgis. Densereg: Fully convolutional dense shape regression in-the-wild. *arXiv preprint arXiv:1612.01202*, 2016.

[95] F. Zhang, D. Clarke, and A. Knoll. Vehicle detection based on lidar and camera fusion. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1620–1625. IEEE, 2014.

[96] Z. Zhang, A. G. Schwing, S. Fidler, and R. Urtasun. Monocular object instance segmentation and depth ordering with cnns. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2614–2622, 2015.

[97] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017.

[98] W. Zhuo, M. Salzmann, X. He, and M. Liu. Indoor scene structure analysis for single image depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern recognition*, pages 614–622, 2015.