Politechnika Poznańska

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ROZPRAWA DOKTORSKA

ALEKSANDER KOSTUSIAK

Metody doboru struktury oprogramowania i optymalizacji parametrów w zadaniu lokalizacji wizyjnej na podstawie danych RGB-D

Methods for the software structure determination and optimization of the parameters in the visual localization task utilizing RGB-D data.

PROMOTOR:
prof. dr hab. inż. Piotr Skrzypczyński

Poznań 2024

Serdecznie dziękuję wszystkim, którzy wspierali mnie podczas pisania tej pracy.

Panu prof. dr hab. Piotrowi Skrzypczyńskiemu, prof. PP za okazaną życzliwość, cierpliwość, wyrozumiałość i za każdą chwilę poświęconego mi czasu. Za merytoryczne wsparcie i pomoc w trakcie realizacji niniejszej pracy, a także za bycie moim mentorem w pozostałych pracach: inżynierskiej i magisterskiej.

Rodzicom: Markowi oraz Mari i bratu Kamilowi dziękuję za okazane mi wsparcie, motywację i wskazówki.

Tobie kochana Haniu za zaufanie i motywację.

Przyjaciołom, kolegom i całemu zespołowi z uczelni.

Streszczenie

W rozprawie przedstawiono metody optymalizacji struktury i parametrów systemu odometrii wizyjnej RGB-D, ze szczególnym uwzględnieniem lokalnego ruchu sensora oraz wykorzystania algorytmów optymalizacji populacyjnej i uczenia maszynowego. Omówiono rolę sensorów wizyjnych w pomiarze odległości w kontekście robotyki mobilnej oraz przeanalizowano wpływ decyzji projektowych dotyczących architektury systemu na jakość odtwarzanej trajektorii. Zaproponowano prosty system odometrii wizyjnej RGB-D, który pozwala zbadać, w jaki sposób poszczególne elementy systemu oraz dobór parametrów wpływają na końcowy wynik.

W rozprawie szczególną uwagę poświęcono wybranym detektorom i deskryptorom cech fotometrycznych oraz porównano metody n-punktowe estymacji ruchu kamery z metodami uwzględniającymi dane o głębi, oceniając ich wpływ na jakość estymowanej trajektorii. Następnie skoncentrowano się na populacyjnych algorytmach optymalizacji, dążąc do znalezienia optymalnych parametrów systemu odometrii wizyjnej RGB-D, które pozostają stabilne w trakcie całego eksperymentu, oraz przeprowadzono próbę optymalizacji detektora cech punktowych w trybie online.

Przedstawiono również metody uzupełniania braków w danych pomiarowych głębi, zarówno klasyczne, jak i oparte na sztucznych sieciach neuronowych. W celu poprawy jakości estymacji trajektorii zmodyfikowano istniejącą sieć neuronową działającą na danych RGB, aby mogła także przetwarzać dane RGB-D. Zbadano wpływ uzupełnionych za pomocą inferencji map głębi na dokładność oszacowania przebytej trajektorii, zarówno w oparciu o dane RGB, jak i RGB-D.

Wszystkie zaproponowane metody i algorytmy zostały zweryfikowane w eksperymentach z wykorzystaniem rzeczywistych danych benchmarkowych, zebranych przy pomocy różnych robotów mobilnych lub ręcznie trzymanej kamery RGB-D.

Abstract

This dissertation presents methods for optimising the structure and parameters of an RGB-D visual odometry system, with particular emphasis on the local movement of the sensor and the use of population-based optimization algorithms and machine learning. The role of visual sensors in distance measurement within the context of mobile robotics is discussed, and the impact of design decisions related to system architecture on the quality of the reconstructed trajectory is analysed. A simple RGB-D visual odometry system is proposed, allowing for an examination of how individual system components and parameter selection affect the final outcome.

Special attention is given to selected photometric feature detectors and descriptors, and a comparison is made between n-point motion estimation methods and those that incorporate depth data, evaluating their impact on the quality of the estimated trajectory. The dissertation then focuses on population-based optimization algorithms, aiming to find optimal parameters for the RGB-D visual odometry system that remain stable throughout the entire experiment, and attempts to optimise the feature point detector in an online mode.

Methods for filling gaps in depth measurement data are also presented, including both classical approaches and those based on artificial neural networks. To improve trajectory estimation quality, an existing neural network operating on RGB data was modified to also process RGB-D data. The impact of depth maps, filled through inference based on RGB and RGB-D data, on the accuracy of the estimated trajectory is investigated, as well as the results that can be achieved based on maps derived solely from RGB-D inference.

All proposed methods and algorithms have been validated through experiments using real-world benchmark data collected with various mobile robots or a handheld RGB-D camera.

Spis treści

Lista symboli i skrótów

c_x, c_y	Offsets of the principal point, Optical center	Położenie środka obrazu względem współrzędnych x, y
$d_{E,1}$, $d_{E,1}$	Initial Euclidean error threshold	Początkowe progi odległości euklidesowej
E	Essential matrix	Macierz zasadnicza
f_x, f_y	Focal lengths	Ogniskowe kamery w osiach x i y
F	Fundamental matrix	Macierz fundamentalna
$\mathbf{g}_{\text{best}}$	Set of globally best found parameter	Zestaw globalnie najlepszych znalezionych parametrów
$\Gamma_{o,1}$, $\Gamma_{o,2}$	Satisfying ratio of inliers to outliers	Zadowalające stosunki między liczbą akceptowalnych (*inliers*) i nieakceptowalnych (*outliers*) dopasowań
k_i	Radial distortion coefficients	Współczynniki zniekształceń radialnych
K	Calibration matrix	Macierz kalibracji
$\mathbf{l}_{\text{best}}$	Set of locally best found parameter	Zestaw lokalnie najlepszych znalezionych parametrów
lr	Learning Rate	Szybkość uczenia sieci
p_i	Tangential distortion coefficients	Współczynniki zniekształceń tangensowych (decentrycznych)
τ_A	AKAZE detector threshold value	Próg detekcji detektora AKAZE
R	Rotation matrix	Macierz rotacji
t	Translation vector	Wektor translacji

$\mathbf{u}$, $\mathbf{u}'$	Points form the first image and matched from the second	Punkty z jednego ujęcia i odpowiadające im punkty z drugiego
u, v	Pixel coordinates	Współrzędne piksela
wd	Weight Decay	Waga Ograniczania Wag Sieci
$\mathbf{v}_m^i$	Velocity of the m-th particle at the i-th time	Prędkość m-tej cząstki w i-tej chwili

AR	Augmented Reality	Rozszerzona Rzeczywistość
AMCW	Amplitude-Modulated Continuous Wave	Ciągła Fala o Modulowanej Amplitudzie
AOS	Additive Operator Splitting	Schemat Dodatniego Operatora dzielenia
ATE	Absulute Trajectory Error	Absolutny Błąd Trajektorii
BA	Bundle Adjustment	Metoda regulacji wiązki
BN	Batch Normalization	Normalizacja wsadowa
BoW	Bag of Words	Worek Słów
CFL	Courant-Friedrichs-Lewy condition	warunek Couranta-Friedrichsa-Lewy'ego
CNN	Convolutional Neural Networks	Neuronowe Sieci Splotowe
EA	Evolutionary Algorithm	Algorytm Ewolucyjny
EKF	Extended Kalman Filter	Rozszerzony Filtr Kalmana
ELU	Exponential Linear Unit	Wykładnicza Jednostka Liniowa
ES	Evolution Strategies	Strategie Ewolucyjne
FFN	FeedForward Networks	Sieci Jednokierunkowe
FLANN	Fast Library for Approximate Nearest Neighbors	Szybka biblioteka do poszukiwania najbliższych sąsiadów
FMM	Fast Marching Method	Metoda Szybkiego Marszu
GA	Genetic Algorithms	Algorytmy Genetyczne
GP	Genetic Programming	Programowanie Genetyczne
GPU	Graphics Processing Unit	Procesor graficzny
GPGPU	General Purpose Graphic Processing Unit	Procesor graficzny ogólnego przeznaczenia
IMU	Inertial Measurements Unit	Jednostka do nawigacji Inercyjnej
LiDAR	Light Detection and Ranging	Skaner laserowy

LSTM	Long Short Term Memory	Długoterminowa Pamięć Krótkoterminowa (typ sieci RNN)
MSE	Mean Squar Error	Błąd średniokwadratowy
MSS	Minimal Sample Set	Model zbudowany na podstawie minimalnej liczby danych
ONXX	Open Neural Network Exchange	Otwarta Wymiana Sieci neuronowych
PCG	Preconditioned Conjugate Gradient	Sprzężony gradient z poprawianiem uwarunkowania
PNP	solver Perspective-n-Point	Algorytm n-punktowy
PF	Particle Filter	Filtr Cząsteczkowy
ROS	Robot Operating System	System Operacyjny Robotów
RELU	Rectified Linear Unit	Rektyfikowana jednostka liniowa
RPE	Relative Pose Error	Względny błąd pozycji
RMSE	Root Mean Squar Error	Pierwiastek błędu średniokwadratowego
RCNN	Recurrent Convolutional Neural Network	Rekurencyjna, neuronowa sieć konwolucyjna
RNN	Recurrent Neural Networks	Sieci Rekurencyjne
StM	Structure-from-Motion	Struktura sceny uzyskana z ruchu kamery
SGD	Stochastic Gradient Descent	Stochastyczna aproksymacja gradientu prostego
SLAM	Simultaneous Localization and Mapping	Jednoczesna samolokalizacja i mapowanie
SVD	Singular Value Decomposition	Rozkład na wartości własne
SSIM	Structural Similarity Index	Indeks Podobieństwa Strukturalnego
SVM	Support Vector Machine	Maszyna wektorów wsparcia
ToF	Time of Flight	Czas Przelotu
UQI	Universal Quality Index	Indeks Uniwersalnej Jakości

Rozdział 1

Wstęp

1.1 Motywacja do podjęcia badań nad systemem lokalizacji wizyjnej RGB-D

Jednym z klasycznych, a zarazem najtrudniejszych i kluczowych zagadnień robotyki jest dokładne oszacowanie przebytej przez robota drogi, trajektorii–szeregu występujących po sobie pozycji i orientacji względem pewnego, arbitralnego układu współrzędnych. Ponieważ to zagadnienie jest w centrum uwagi od wielu lat, doprowadziło to do powstania wielu rozwiązań, których różnorodność doddatkowo zwiększa rodzaj wykorzystywanych danych: sygnał GPS, ultradźwięki, światło podczerwone, światło strukturalne, czy też światło widzialne.

System GPS zapewnia wysoką precyzję, jednakże takie podejście obarczone jest dość znacznymi wadami. Tracimy dokładność, jeśli ma działać w czasie rzeczywistym, online, ponieważ nie jesteśmy w stanie pobrać danych ze stacji referencyjnych na czas, aby porównać je z sygnałem nadawanym z satelit. Taki system działa dobrze tylko na zewnątrz i to tylko w przypadku, gdy dochodzi sygnał z co najmniej 3 satelit. Zatem istnieją okienka czasowe, w których to rozwiązanie nie działa.

Wraz z pojawieniem się nowego sensora, jakim była kamera, rozpoczęto prace nad wykorzystaniem dostarczanych przez niego informacji do innych celów, niż pierwotnie zakładana rejestracja obrazu na potrzeby historyczne, osobiste czy też komercyjne.

Ponieważ wiele robotów mobilnych działa również wewnątrz budynków, jak również w jaskiniach czy innych miejscach, gdzie nie dochodzi sygnał nadawany z satelit, to nie możemy polegać na systemie GPS. W celu rozwiązania tego pro-

blemu powstało wiele różnych rozwiązań, do których można zaliczyć odometrię kołową, odometrię wizyjną VO (ang. *Visual Odometry*) [207], a wreszcie pełny system jednoczesnej samolokalizacji i tworzenia mapy SLAM (ang. *Simultaneous Localization And Mapping*) [184, 219]. Do najlepszych rozwiązań należą te korzystające z danych wizyjnych.

Rozwiązania wykorzystujące tylko ten typ informacji mogą być podzielone na dwa główne podejścia: „gęste" oraz „rzadkie" czyli bazujące na cechach charakterystycznych. Do najlepszych podejść należą te „gęste" wykorzystujące wszystkie informacje dostarczane przez kamerę. Jednak jest to bardzo kosztowne obliczeniowo [157] i wymaga wspomagania się kartą graficzną. Sprawia to, że wykorzystanie takich rozwiązań na większą skalę w robotyce mobilnej jest niemal niemożliwe. Dlatego w tej rozprawie skupiono się na zastosowaniu „rzadkich" metod.

Początkowo możliwości techniczne pozwalały na zbieranie danych tylko w przestrzeniach zamkniętych. Natomiast nowsze wersje, takie jak Kinect v2 pozwalają również na wykorzystanie ich na zewnątrz. Dane ze starszego sensora były też mniej dokładne i cechowały się większymi brakami. W związku z tym podjęto próbę wykorzystania nowszego, lepszego sensora do ulepszenia danych otrzymanych z starszego sensora za pomocą klasycznych metod, oraz sztucznych sieci neuronowych. To ulepszanie danych ma kluczowe zastosowanie w nisko–budżetowych rozwiązaniach.

Metody lokalizacji wizyjnej dowiodły już swej przydatności w wielu zadaniach, w tym w zadaniach poszukiwawczo–ratowniczych [104]. Moduł lokalizacji jest też niezwykle istotny w planowaniu ruchu autonomicznych robotów kroczących. Zapewnienie dokładności lokalizacji nie przekraczającej wielkości stopy robota jest niezbędne do zapewnienia odpowiedniej rejestracji danych sensorycznych [101].

Problem VO/SLAM można rozwiązać na wiele różnych sposobów i dla tego trudno jest ocenić wpływ, jaki mają implementacje poszczególnych części tych systemów [130, 99] na ostateczną wydajność i wyniki. Szczególnie żmudny jest wybór takich bloków VO/SLAM jak detektor i deskryptor cech kluczowych, czy sposób dopasowywania tych cech [165]. Wybór bloków jest kwestią architektoniczną systemu i trudną w automatyzacji, natomiast znalezienie optymalnych parametrów można powierzyć odpowiednim algorytmom.

Seghal i inni [212] wykorzystali algorytm genetyczny do optymalizacji parametrów monokularowego systemu odometrii wizyjnej VO korzystającego również z LiDAR. Natomiast Wang i inni [233] wykorzystali metodę

rojową PSO do ulepszenia dokładności dopasowywania skanów, poprzez pokrycie i dopasowanie wydobytych cech z globalną mapą, w bazującym na chmurze punktów SLAMie $3D$, uzyskując poprawę estymacji pozycji. Natomiast w systemach SLAM można wykonać taką optymalizację w innych celach np. poprawie skuteczności wykrywania i zamykania pętli [95, 136], czy poprawie próbkowania i unikania lokalnych ekstremów w bazującym na filtrze cząsteczkowym SLAMie [247].

Dokładną lokalizację oraz mapę można wykorzystać w celach handlowych i np. militarnych poprzez dokładne planowania trasy, jaką ma przemierzać robot by przemieścić się z punktu A do punktu B. Z bardzo dokładnej mapy świata $3D$, albo dużego miasta można by wyciąć odpowiedni kawałek dla drona dostawczego, by dostarczył odpowiednią paczkę, czy zaopatrzenie w miejsce docelowe.

1.2 Sformułowanie problemu naukowego–teza pracy

1.2.1 Hipoteza badawcza

Zastosowanie populacyjnych metod optymalizacji oraz wybranych algorytmów uczenia maszynowego pozwala zoptymalizować strukturę i parametry systemu lokalizacji RGB-D, ze szczególnym uwzględnieniem lokalnego ruchu sensora.

Zastosowanie populacyjnych metod optymalizacji oraz wybranych algorytmów uczenia maszynowego pozwala zoptymalizować strukturę i parametry systemu lokalizacji RGB-D, ze szczególnym uwzględnieniem lokalnego ruchu sensora.

1.2.2 Hipotezy pomocnicze

1. Populacyjne metody optymalizacji pozwalają istotnie zmniejszyć nakład pracy związany z wyborem najlepszych parametrów systemu odometrii wizyjnej RGB-D, a otrzymane parametry są odpowiednie dla wielu sekwencji testowych zebranych w podobnym środowisku.

2. Algorytm uczenia maszynowego oparty na sieciach neuronowych i wykorzystujący dane RGB-D umożliwia estymację głębi sceny w obszarach, w których brakuje pomiarów głębi z rzeczywistego sensora

używającego oświetlenia strukturalnego, co poprawia jakość trajektorii generowanych przez system odometrii wizyjnej.

1.3 Plan badań

Rozprawa dotyczy nowych rozwiązań systemów wizyjnych wykorzystujących kombinację pasywnej kamery RGB i aktywnej kamery głębi. Z uwagi na złożoność stereowizji, a co za tym idzie wymagania obliczeniowe, obecnie niezbyt często stosuje się w robotyce mobilnej systemy złożone z więcej niż jednej kamery RGB. Od ukazania się na rynku sensorów typu Kinect, pozwalających w przystępnej cenie na akwizycję kolorowych obrazów wraz z odpowiadającymi im pomiarami głębi, popularne stały się systemy korzystające z tak dostarczonych map głębi. Niestety, ze względu na sposób pomiaru, w wielu przypadkach dane dotyczące głębi sceny nie są kompletne i nie zawsze odzwierciedlają faktyczną odległość dla poszczególnych pikseli. W tej sytuacji określenie pozycji robota na podstawie danych wizyjnych jest utrudnione i wymaga stosowania wyrafinowanych oraz złożonych obliczeniowo metod opartych na zasadach filtracji optymalnej lub optymalizacji nieliniowej.

W rozprawie proponowany jest relatywnie prosty system odometrii wizyjnej RGB-D, który pozwala jednak na wierne odwzorowanie przebytej przez robota/sensor trajektorii na podstawie zebranych klatka po klatce danych. Główny nacisk został położony na taką jego konstrukcję, która pozwoli na uzyskanie powtarzalnych wyników, a optymalizacji (w szerokim sensie) podlega głównie procedura wyznaczania przekształcenia (roto–translacji) między kolejnymi klatkami danych z sensora.

Zaproponowane rozwiązanie opiera się na algorytmach w sposób umożliwiający badanie wpływu poszczególnych składowych na jakość oszacowanej trajektorii. Z uwagi na mnogość dostępnych rozwiązań, przeanalizowane zostaną w sposób szczegółowy jedynie algorytmy detekcji i deskrypcji punktów kluczowych. W procesie estymacji przemieszczenia nie biorą bowiem udziału wszystkie punkty widziane przez kamerę, a jedynie te które zdają się nieść więcej informacji niż piksele je otaczające. Wzięcie pod uwagę wszystkich punktów wydłużyłoby wielokrotnie obliczenia, uniemożliwiając zastosowanie tego systemu w praktyce.

Poddane analizie zostaną dostępne systemy odometrii wizyjnej, jak i jednoczesnej samolokalizacji i budowy mapy (SLAM). Dostępne rozwią-

zania zostaną porównane w odniesieniu do powszechnie używanych zbiorów danych testowych (ang. *benchmarks*), jak i w odniesieniu do zaproponowanego rozwiązania.

W trakcie badań porównywane będą różne populacyjne metody optymalizacji parametrów w kontekście ich skuteczności w odnajdowaniu najlepszych parametrów systemu lokalizacji wizyjnej. Metody będą porównywane pod względem jakości oszacowanej trajektorii z zastosowaniem ogólnie przyjętych miar absolutnego błędu trajektorii ATE i względnego błędu pozycji RPE [225], a także pod względem czasu potrzebnego na znalezienie przybliżonego rozwiązania. Z uwagi na złożoność zadania w eksperymentach wykorzystywane będą publicznie dostępne zestawy danych i sprawdzane będzie na jakim typie danych można znaleźć uniwersalne parametry.

Z uwagi na sposób działania systemu, z klatki na klatkę, w sposób szczególny zostaną przeanalizowane składowe względnego błędu trajektorii RPE. Pozwoli to na wysnucie wniosków co do dopasowania obrazu widzianego przez kamerę RGB i głębi, a także na korekcję parametrów kalibracyjnych sensora.

Istotne w przypadku działania odometrii wizyjnej jest prawidłowe dopasowanie widzianych na kolejnych ujęciach punktów kluczowych. W wielu przypadkach niedopuszczalne jest, by punkt kluczowy widziany na jednym ujęciu był bezzasadnie przesunięty na drugim ujęciu względem prawidłowej lokalizacji. Wprowadza to dość znaczny błąd w obliczeniach. Kolejnym bardzo ważnym aspektem jest to, by ten punkt był prawidłowo rozpoznany na kolejnym obrazie, ponieważ źle sparowane punkty wprowadzają największe błędy w obliczeniach.

Nie wszystkie wykryte punkty kluczowe posiadają odpowiadającą ich lokalizacji informację o głębi. Powoduje to odrzucenie części charakterystycznych i dobrze umiejscowionych cech, a to negatywnie wpływa na jakość estymowanej trajektorii. W związku z tym, planowane jest wykorzystanie metod uczenia maszynowego do uzupełniania brakujących danych. Spośród metod uczenia maszynowego planowane jest testowanie przede wszystkim podejścia opartego na głębokich sieciach neuronowych (ang. *deep learning*).

Oczekuje się, że realizacja proponowanego planu badań doprowadzi do eksperymentalnego zweryfikowania hipotez cząstkowych, dotyczących możliwości optymalizacji sposobu działania systemu lokalizacji wizyjnej RGB-D. System taki powinien być dobrą podstawą do stworzenia bardziej

zaawansowanych aplikacji, takich jak system jednoczesnej lokalizacji i mapowania, metody planowania ruchu robota oraz metody rekonstrukcji obiektów widzianych na scenie.

Dodatkowymi rezultatami prowadzonych prac będzie implementacja oraz porównanie: populacyjnych metod optymalizacji dla zadania optymalizacji parametrów systemu odometrii wizyjnej, wyników badań wpływu optymalizacji na jakość odzyskanej trajektorii, wyników badań metod wyboru różnych par detektor-deskryptor cech, metod uczenia odtwarzania mapy głębi, oraz wyników badań eksperymentalnych, porównujących całościowo zaproponowane rozwiązanie z dostępnymi, reprezentatywnymi, otwarto–źródłowymi systemami.

Koncepcje nowych rozwiązań zawarte w planie badań dotyczą głównie nowych zastosowań znanych algorytmów optymalizacji oraz wykorzystania i badania metod uczenia maszynowego. Oczekuje się, że użycie zarówno metod optymalizacji, jak i uczenia pozwoli wybrać odpowiednie elementy struktury systemu nawigacji, a także automatycznie dobrać ich optymalne parametry.

1.4 Wprowadzenie do treści rozprawy

W rozdziale 2 skupiono się na opisie aktualnego stanu wiedzy w obrębie metod pomiaru odległości, roli systemów wizyjnych w robotach mobilnych, oraz otwarto–źródłowych systemach SLAM.

Następnie w rozdziale 3 opisano dostępne metody detekcji i deskrypcji cech fotometrycznych oraz sposoby ich dopasowywania. Przedstawiono metody estymacji przebytej trajektorii i zaproponowano prosty system odometrii wizyjnej RGB-D.

Rozdział 4 poświęcono wybranym metodom optymalizacji parametrów a rozdział 5 uczeniu maszynowemu bazującemu na wykorzystaniu głębokiego uczenia sieci neuronowych.

Przeprowadzone badania eksperymentalne opisano w rozdziale 6.

Rozdział 7 jest podsumowaniem całości rozprawy. Z przeprowadzonych badań wyciągnięto wnioski i zaproponowano dalszy przebieg badań.

Rozdział 2

Stan wiedzy w obszarze systemów lokalizacji wizyjnych w robotyce

2.1 Metody pomiaru odległości wykorzystujące aktywne i pasywne sensory obrazu

Niniejsza praca bazuje na danych wizyjnych, w związku z tym należy omówić czujniki dostarczające te informacje. Ogólnie sensory wizyjne można podzielić na aktywne, czyli takie które mają własne źródło światła i pasywne, które go nie posiadają.

2.1.1 Aktywne sensory

Najogólniej rzecz ujmując, są to wszelkiego rodzaju sensory wypromieniowujące energię do otoczenia. W dalszej części ograniczymy się do stosowania tego pojęcia w odniesieniu do tych urządzeń, które emitują energię specjalnie w celach pomiarowych, a nie jako efekt uboczny ich działania. W skład tej grupy sensorów wchodzą dalmierze, kamery z oświetleniem, sonary i urządzenia służące do radio-lokalizacji. W tej rozprawie skupimy się w szczególności na rozwiązaniach wykorzystujących triangulację, bezpośredni i bezpośredni pomiar czas przelotu ToF (ang. *Time of Flight*) oraz światło strukturalne.

Bezpośredni pomiar czasu przelotu

Na tej zasadzie działają zarówno czujniki korzystające ze światła jak i te wykorzystujące ultradźwięki. Główna zasada jest ta sama: znając prędkość v (w przypadku światła $v = c$) rozchodzenia się fali w danym ośrodku, można obliczyć odległość d do odbijającego przedmiotu, wykorzystując równanie (2.1):

$$d = \frac{vt}{2} \tag{2.1}$$

gdzie t to łączny czas przelotu wiązki w obie strony.

Bezpośredni pomiar–właściwości laserów

Wykorzystanie światła do mierzenia odległości pozwala na uzyskanie bardzo dokładnych pomiarów odległości. Przy korzystaniu z laserów nie trzeba kompensować temperatury, ale należy pamiętać, że nie wysyłają one jednej fali, a wiązkę fal o zbliżonych częstotliwościach (szczególnie te półprzewodnikowe, można to kompensować np. detekcją progową), więc uzyskuje się pomiary nie punktu, a pewnego zbioru, plamki. Część fal jest absorbowana i rozpraszana, więc do pomiarów dalszych odległości niezbędny jest mocniejszy laser. Kąt padania również ma znaczenie, bo zmienia kształt odbieranej plamki. W związku z tym wybierając konkretny sensor do danego problemu należy rozważyć wszystkie zalety i wady, jak np. pobór mocy, bezpieczeństwo (może np. uszkodzić wzrok), maksymalny zasięg, szybkość działania, masa czy cena. Większość czujników w rozsądnej cenie nie nadaje się do użytku w warunkach silnego, naturalnego oświetlenia, jakie występuje w ciągu dnia na zewnątrz budynków.

Prędkość światła minimalnie się zmienia wraz z ośrodkiem, więc trzeba uważać, czy na scenie nie ma obiektów wpływających w ten sposób na pomiary (np. akwarium). Dla $\lambda = 589nm$ w próżni wynosi $c = 299792458\frac{m}{s}$ [1], zaś w powietrzu $c = 299702547\frac{m}{s}$. W wodzie współczynnik załamania nieznacznie się zmienia wraz z długością fali. Prędkość w tym ośrodku można policzyć jako (2.2):

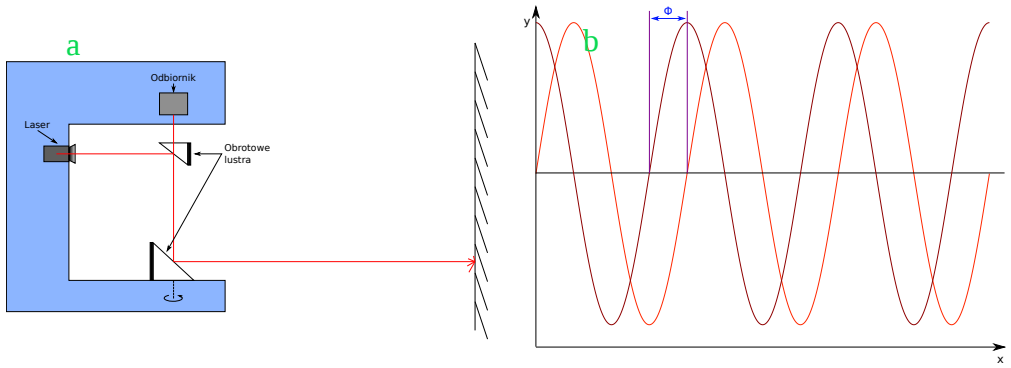
$$v = c \cdot n = 299792458 \div 1,33 = 225407863 \left[\frac{m}{s}\right] \tag{2.2}$$

gdzie v to prędkość w nowym ośrodku a n to współczynnik załamania.

Do niedawna, pomiar krótkich odległości z dokładnością co do cm lub mm stanowił problem technologiczny, bo oznaczało to mierzenie czasu z dokładnością co do ps, a to było bardzo drogie. Dokładność pomiaru upływu czasu wyznacza również minimalną odległość, od której można zacząć wykonywanie pomiarów.

Bezpośredni pomiar–skaner LiDAR

Skanery laserowe LiDAR (ang. *Light Detection and Ranging*). Zwykle polega to na wykorzystaniu obrotowego lustra do odbijania światła na scenę pod różnymi, znanymi katami tworząc w ten sposób jej reprezentację w postaci chmury punktów $3-D$. Na rys. 2.1a zaprezentowano prosty schemat skanera $2D$.



Rysunek 2.1: a) Skaner laserowy, b) Zasada pomiaru przesunięcia fazy

Pośredni pomiar czasu przelotu–przesunięcie fazowe

Metoda AMCW (ang. *Amplitude-Modulated Continuous Wave*), w odróżnieniu od pomiarów ToF, nie wymaga przerywania nadawania fali podczas dokonywania pomiarów. Wysyłana jest ciągła fala o sinusoidalnie modulowanej amplitudzie, co pokazano na rys. 2.1b.

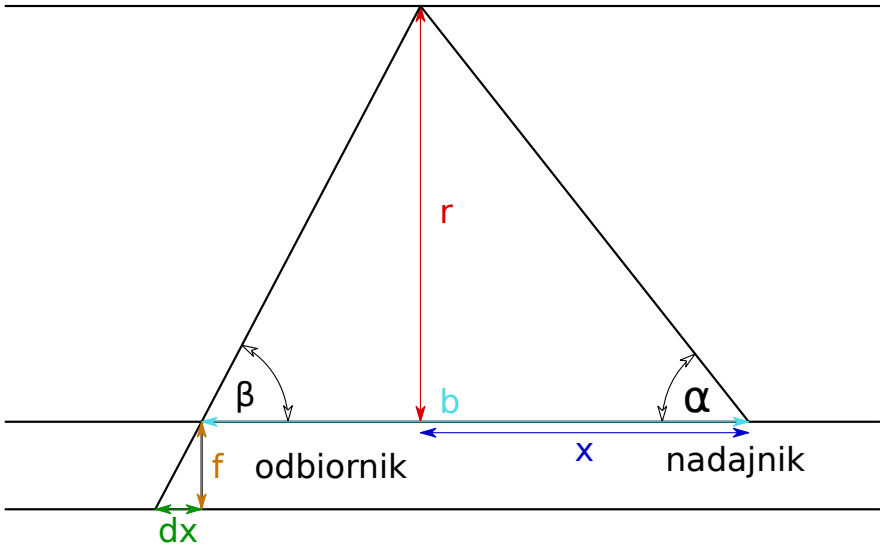
Przesunięcie fazowe jest proporcjonalne do odległości obiektu od nadajnika/odbiornika (2.3):

$$r = \frac{c}{2\omega}\Phi + n\frac{c\pi}{\omega} \tag{2.3}$$

Ponieważ modulujemy tą falę, to częstotliwość ω jest znana–dobrana przez nas, c to prędkość światła, a Φ to mierzone przesunięcie. Tak wykonany pomiar jest niejednoznaczny, bo możemy mierzyć w zakresie kąta pełnego, Φ będzie się powtarzało. By dokładniej ustalić odległość można mierzyć ją wykorzystując dwie różne częstotliwości modulacji.

Triangulacja

Jest to jedna z najstarszych i najprostszych metod pomiaru odległości. Rys. 2.2 przedstawia ogólny schemat pomiaru.



Rysunek 2.2: Schemat triangulacyjnej metody pomiaru odległości

Znając ogniskową f oraz odległość b pomiędzy nadającym wiązkę, pod znanym kątem α, laserem a odbierającym ją, pod nieznanym kątem β, odbiornikiem można wyznaczyć (2.4) odległość r do obiektu, korzystając z przesunięcia dx odbieranej plamki względem środka układu pomiarowego.

$$\begin{array}{lcll} \frac{f}{dx} = \frac{r}{b-x} & & & x = \frac{r}{tg(\alpha)} \\ f \cdot b = dx \cdot r & + & \frac{f \cdot r}{tg(\alpha)} & = \quad r\left(dx + \frac{f}{tg(\alpha)}\right) \\ r = \frac{f \cdot b}{dx + \frac{f}{tg(\alpha)}} & = & \frac{b}{\frac{dx}{f} + \frac{1}{tg(\alpha)}} & = \quad \frac{b}{\frac{dx}{f} + ctg(\alpha)} \end{array} \tag{2.4}$$

Gdy baza b jest stosunkowo duża do mierzonej odległości, to pomiar jest dokładny, ale potrzeba wtedy dużej matrycy. Wraz ze wzrostem odległości maleje dokładność a duża baza może sprawić, że nie otrzymamy obrazu, ponieważ istnieje ryzyko, że coś znajduje się na drodze wiązki między odbiornikiem, przedmiotem

a nadajnikiem. Odbita plamka musi być jak najjaśniejsza a jej kontrast duży. Takie pomiary nie są odporne na silne światło w otoczeniu, np. słoneczne, które redukuje kontrast.

2.1.2 Systemy wizyjne

Do tego celu służą obecnie kamery–pasywne sensory matrycowe rejestrujące dwuwymiarowy rzut perspektywiczny oświetlonego, trójwymiarowej fragmentu otoczenia na zdyskretyzowaną przestrzennie płaszczyznę matrycy obrazowej [217]. Kiedyś w robotyce mobilnej dominowały kamery wykorzystujące zewnętrzne oświetlenie, dziś popularne stało się również emitowanie światła do tworzenia map głębi–obrazu reprezentującego odległość sensora do najbliższej przeszkody.

Światło strukturalne

Ogólna zasada polega na emitowaniu znanego wzorca na scenę i rejestrowaniu zniekształconego wzorca przez kamerę [2] ustawionej w znanej odległości od projektora i pod znanym kątem. Na podstawie różnic pomiędzy wysyłanym wzorcem a odebranym odtwarza się głębię sceny.

Wzorcem zwykle jest to zestaw pewnych kropek (jak np. w sensorze Kinect v1), pasków, czy nawet zmiennych w czasie wzorów. Ta metoda nie nadaje się do stosowania, w środowiskach w których znajduje się dużo przejrzystych obiektów. Jest również nieodpowiednia do pomiaru dużych odległości albo przy silnym zewnętrznym (szczególnie naturalnym) oświetleniu. Pomiary mogą być również zakłócone przez inne źródła światła strukturalnego wysyłające wzory na tej samej długości fali, co odbierana przez kamerę pomiarową. Stosowanie światła strukturalnego pozwala na uzyskanie wysokiej dokładności pomiarów dla krótkich odległości za rozsądną cenę (w porównaniu do innych technologi).

Wykorzystywany w badaniach sensor Kinect v1 został opracowany przez firmę PrimeSense na zlecenie Microsoftu. Jest to urządzenie przeznaczona do gier we współpracy z konsolą Xbox 360 firmy Microsoft składające się z: promiennika i odbiornika podczerwieni CMOS o rozdzielczości 320×240 pikseli interpolowanej do 640×480, kamery RGB (także CMOS) o rozdzielczości 640×480, 4 mikrofonów oraz z silnika regulującego nachylenie sensora. Dane o obrazie i głębi są przesyłane z prędkością 30 FPS [198]. Pomimo swojego przeznaczenia Kinect okazał się bardzo atrakcyjną alternatywą dla wielu drogich skanerów laserowych i właśnie dla tego został z radością powitany w środowisku robotyki mobilnej, jak i innych. Rys. 2.3 przedstawia budowę Kinecta.

Rysunek 2.3: a) Kinect_v1 w obudowie [3] b) Schemat urządzenia [149]

Zasadnicza część patentu [4] chroniącego Kinecta dotyczy budowy i sposobu wytwarzania, z niedostępną wcześniej dokładnością, dyfrakcyjnego elementu optycznego. Konkurencyjnymi odpowiednikami Kinecta są sensory takie jak Asus Xtion czy Occipital Structure [5].

Kinect jednak nie był pierwszym urządzeniem tego typu [4, 6]. Prawdopodobnie poprzednikiem była kamera ZCam firmy 3DV Systems (działała na zasadzie pomiaru czasu przelotu ToF), która nie doczekała się premiery gdyż firma została kupiona przez Microsoft, rok przed premierą Kinecta.

Firma PrimeSense zaproponowała nieco inne podejście do sposobu działania skanera laserowego: pomiaru odległości dokonywany jest na podstawie analizy wzoru odbieranych, niewidocznych dla ludzkiego oka, podczerwonych punktów z fabrycznie zakodowanym wzorem, na zasadzie analizy światła strukturalnego [4]. Do estymacji głębi wykorzystywany jest algorytm z dziedziny stereowizji: dla wybranej plamki ze skalibrowanego wzorca poszukuje się jej odpowiednika w scenie razem z ośmioma unikatowymi, otaczającymi ją pikselami. Dzięki temu można obliczyć obraz dysparycji a przy wykorzystaniu informacji o długości ogniskowej kamery podczerwieni i odległości–bazy pomiędzy nadajnikiem a odbiornikiem można także zdeterminować jak daleko na scenie znajduje się punkt. Tę czynność powtarza się dla każdego punktu ze wzorca.

Po założeniu noktowizora na standardową kamerę można zarejestrować emitowany przez Kinecta wzór. Przykład takiego obrazu przedstawiony jest na rys. 2.4.

Emitowane przez Kinecta punkty mają trzy rozmiary odpowiednio dobrane dla różnych odległości pozwalając na pracę urządzenia od $1m$ do $8m$ (w nowszych wersjach zasięg jest krótszy [4]), przedstawia to schematycznie rys. 2.5a). Inne źródła podają zasięg $0,4m - 6,5m$ a sam Microsoft podaje, że minimalny zasięg to $0,4m$ a maksymalny $4m$ (w trybie pomiaru bliskich odległości jest to zasięg $0,5m$ do $3m$ [3]). Najpierw rozpoznawana jest wstępna odległość do obiektu a

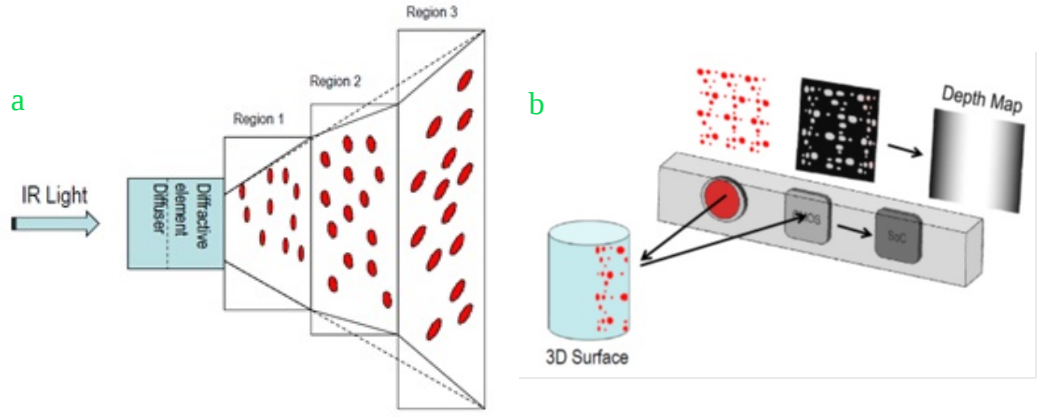
Rysunek 2.4: Emitowane przez Kinecta punkty widziane za pomocą noktowizora [7]

następnie do obliczenia głębi korzysta się z tylko tych punktów, które są optymalne dla danej odległości. Dokładność pomiaru maleje ze wzrostem odległości. Kinect wysyła mniej kropek i robi to rzadziej. Porównywane są także zmiany wielkości i kształtu danej plamki z wzorcową, co prezentuje rys. 2.5.

Jednak temu rozwiązaniu towarzyszą duże obszary bez danych głębi, szczególnie blisko krawędzi widzianych na scenie obiektów. W nowszej wersji zastosowano podejście ToF. Zapewnia ono uzyskanie spójniejszej mapy głębi dla tych samych punktów obserwacji. Poruszono to w podrozdziale 6.2.2 i ukazano na rys. 6.3. Do obsługi Kinecta można skorzystać z następujących bibliotek: OpenNI, PCL czy systemu ROS. Są one opisane w rozdziale czwartym.

Stereowizja

Podejścia wykorzystujące stereowizję z własnym, sztucznym oświetleniem światłem strukturalnym, bądź losowym wzorcem, niewiele się różnią od tych wykorzystujących światła zewnętrzne. W związku z tym opis stereowizji został umieszczony w podrozdziale o pasywnych sensorach. Głównym celem oświetlenia jest dodanie pewnych cech ułatwiających dopasowywanie punktów i poprawiających dokładność oszacowywanej odległości oraz uodpornienie systemu na zmienne wa-



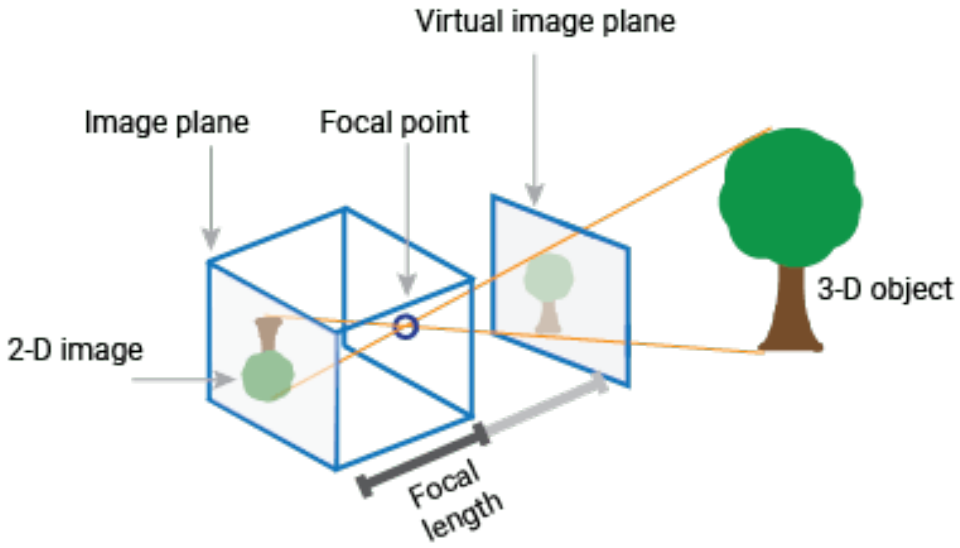
Rysunek 2.5: a) Schematyczna ilustracja działania Kinecta_v1 [4] b) Schemat tworzenia mapy głębi [4]

runki oświetleniowe.

2.1.3 Kalibracja kamery

Kalibracja kamery polega na oszacowaniu parametrów soczewek oraz sensora obrazu, by móc poprawić zniekształcenia w rejestrowanym obrazie. Oprócz tego można je wykorzystać do rzeczywistego rozmiaru obiektu, ustalenia pozycji sensora względem sceny, rekonstrukcji $3D$ sceny czy w systemie nawigacji robota. Ponieważ można odwzorować obiekty widziane w trójwymiarowej przestrzeni na dwuwymiarowej płaszczyźnie obrazu na wiele sposobów, to skupiono się na otworkowym modelu kamery (ang. *pinhole*), w którym każdej z prostych rzutujących odpowiada jeden punkt–przedstawiono go na rys. 2.6.

Parametry dotyczące kamery można podzielić na trzy grupy: parametry zewnętrzne (rotacja i translacja lokalnego układu współrzędnych kamery względem globalnego), wewnętrzne oraz współczynniki zniekształceń radialnych k_i oraz tangensowych p_i (decentrycznych)–są one obecne ze względu na wykorzystanie soczewek (których idealna kamera otworkowa nie posiada). Parametry wewnętrzne to: położenie środka (c_x, c_y) obrazu względem współrzędnych x i y, ogniskowe kamey w osiach x i y (współczynniki skali) f_x i f_y, parametr γ opisujący skręt osi układu współrzędnych kamery. Te parametry można zapisać łącznie w macierzy



Rysunek 2.6: Otworkowy model kamery (ang. *pinhole*) [8]

kalibracji **K** (2.5):

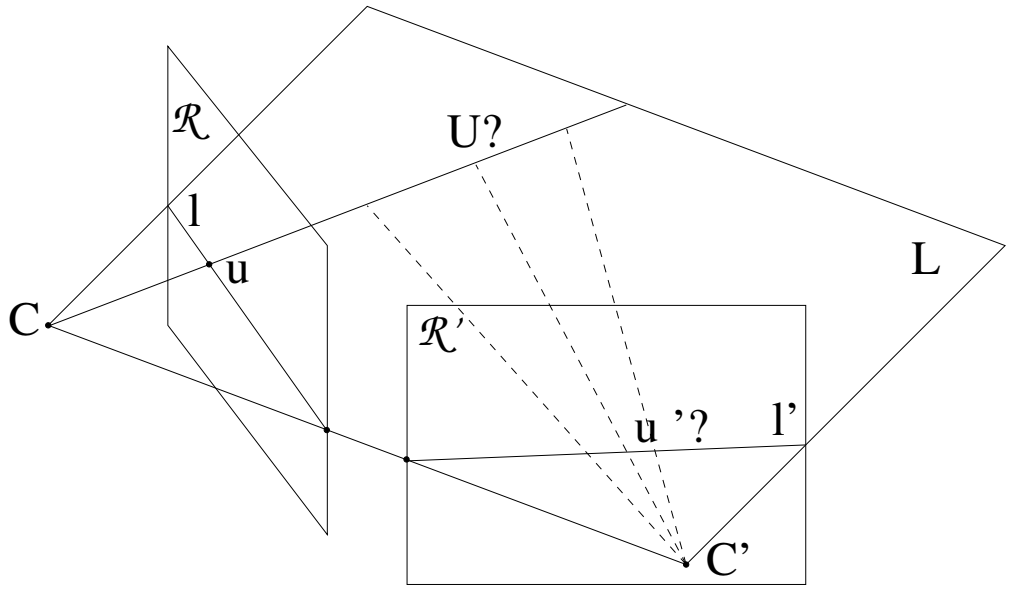
$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

By móc estymować te parametry niezbędne jest posiadanie dopasowanych punktów w przestrzeni $3D$ z odpowiadającymi im na obrazie punktami $2D$, które można uzyskać posługując się odpowiednim, znanym wzorcem kalibracyjnym (np. szachownicą) [8].

2.1.4 Geometria Epipolarna

Obserwacje sceny, z więcej niż jednego miejsca, pozwalają na powiązanie zależnościami geometrycznymi, widzianych na niej i odpowiadających sobie punktów. Zakładając, że znane są nam pozycje kamer C i C' (rys. 2.7) jak i płaszczyzn $\mathcal{R}$ i $\mathcal{R}'$, na które rzutowany jest obraz, to mimo, że nie wiadomo gdzie dokładnie leży punkt U w przestrzeni, to wiadomo, że powinien się znajdować na lini l wychodzącej ze środka kamery C i przechodzącej przez punkt u [201]. Tą linię można rzutować na inną płaszczyznę (tworzącą inny obraz), więc odpowiadający punktowi u punkt u' musi leżeć na lini l'. Każda płaszczyzna L przechodząca przez środki

projekcji C i C' pociąga za sobą takie powiązane linie. Wszystkie one przechodzą przez szczególny punkt e, bądź e' na drugim obrazie, zwany epipolarnym. Są one rzutem środka kamery drugiego obrazu.



Rysunek 2.7: Plan epipolarny [201]

Matematycznym odzwierciedleniem tej geometri, łączącej punkty **u** z **u**′ jest, posiadająca 8 stopni swobody, macierz fundamentalna **F** spełniająca równanie (2.6):

$$\mathbf{u}'^{T}\mathbf{F}\mathbf{u} = 0 \tag{2.6}$$

Uwzględniając parametry kalibracyjne kamer uzyskuje się macierz zasadniczą **E** (2.7):

$$\mathbf{E} = \mathbf{K}'^{T}\mathbf{F}\mathbf{K} \tag{2.7}$$

gdzie **K** oraz **K**′ są odpowiednimi macierzami kalibracyjnymi kamer. Macierz zasadnicza posiada 5 stopni swobody: 3 ze względu na rotację i 2 ze względu na pozbawiony skali wektor łączący środki obydwu kamer.

2.1.5 Pasywne sensory–stereowizja

W metrologi uznaje się za pasywne sensory takie, które działają bez zasilania. W naszym przypadku uznajemy za nie również kamery. Początkowo do odwzorowania głębi w obrazie korzystano z stereografii–tą samą scenę obserwowano dwoma kamerami.

Obecnie do pomiaru głębi również służy stereowizja [108]. Z dwóch ujęć tej samej sceny można stworzyć jej przestrzenną reprezentację. Do tego celu niezbędne jest posiadanie informacjii na temat wzajemnego położenia punktów na zdjęciach wykonanych przez te kamery. Tą zależność opisuje geometria epipolarna, którą matematycznie wyraża macierz zasadnicza **E**. Dokładniejszy opis geometri epipolarnej można znaleźć w podrozdziale 2.1.4.

Obrazy mogą pochodzić z jednej kamery przesuniętej w znany sposób. W związku z tym źródłem błędu może być błąd wyznaczonego przekształcenia układu współrzędnych ujęcia A do ujęcia B. Dlatego zwykle korzysta się z systemu dwóch kamer o dokładnie wyznaczonej transformacji. Zwykle ich osie są równoległe a kamery są ustawione w poziomie, bo wtedy krawędzie pionowe będą przesunięte a poziome będą się nakładać, a to w przyrodzie występuje najczęściej. Można również ustawić kamery pionowo. Wówczas wystąpi sytuacja odwrotna. Na podobnej zasadzie działają systemy korzystające z trzech (trinokularowe) i więcej kamer, czerpiące korzyści z obu rodzajów ustawień.

Kąt widzenia steroskopowego zależy od odległości B, zwanej bazą, pomiędzy kamerami. Bliższym punktom odpowiada większa zmiana obserwowanego położenia na matrycach kamer. Dla każdego piksela z ujęcia A poszukuje się odpowiadającego mu na ujęciu B. Następnie usuwa się zniekształcenia z obydwu ujęć. Do tego celu wykorzystuje się kalibrację, której procedury opisano w 2.1.3. Istotną cechą całego procesu jest zapewnienie rejestracji obiektów sceny, w dokładnie tym samym stanie i położeniu, dla obydwu ujęć. Oznacza to wymuszenie synchronizacji akwizycji ujęć przez obie kamery, bądź niezmiennego obserwowanego środowiska.

Matryce obydwu kamer ustawia się równolegle do siebie w celu ułatwienia zadania dopasowywania punktów, ponieważ wtedy linie epipolarne powinny być horyzontalne. W praktyce idealne ustawienie kamer jest niemożliwe. W związku z tym obrazy są poddawane procesowi rektyfikacji. Dzięki temu odpowiadające sobie punkty leżą na tej samej lini horyzontalnej. Różnice w ich położeniu na niej prowadzą do powstania mapy dysparycji, którą wykorzystuje się do obliczenia odległość przy wykorzystaniu triangulacji.

2.2 Rola systemów wizyjnych w robotach mobilnych

Zdecydowana większość zwierząt wykorzystuje informacje niesione przez światło do wszelako rozumianego widzenia. Taki sposób działania został zapewne nie tylko dogłębnie sprawdzony przez naturę, ale i we wszelaki sposób podważany i kwestionowany poprzez alternatywne podejścia postrzegania rzeczywistości. Jest to też dla nas, ludzi, naturalny sposób patrzenia na świat i o wiele łatwiej jest nam zrozumieć kogoś albo coś co widzi czy działa podobnie. Nic więc dziwnego, że w budowanych przez ludzi robotach również chce się wykorzystać ten typu informacji.

2.2.1 Robot

W zasadzie od zawsze ludzie dążyli do urzeczywistniania własnych marzeń oraz zwiększenia wpływów.

Wraz z rozwojem techniki zaczęto tworzyć urządzenia o coraz większych możliwościach aż stworzono skomplikowane maszyny zdolne do wykonywania określonego zadania dzięki wyposażeniu ich w komputer z programem sterującym jego peryferiami na podstawie danych dostarczanych przez sensory. Choć dziś nie zawsze jest to tożsame, a definicja bardzo szeroka, to części systemów przypisywano posiadanie sztucznej inteligencji i nazwano robotem [9].

Słowo to wywodzi się od czeskiego „robota“ oznaczającego ciężką pracę, wysiłek. Po raz pierwszy w tym znaczeniu zostało użyte przez Karela Čapka w dziele *Rossumovi Univerzální Roboti*, odnosząc się do sztucznych istot żywych, jednak wymyślenie tego słowa przypisuje on swemu bratu [10], z którym wspólnie tworzył, Josefowi.

Wśród robotów [11] rozróżnia się te, które nie są mobilne i te które są, czyli potrafią zmienić swoje położenie w przestrzeni, pływając, jeżdżąc lub latając. Można je dodatkowo podzielić na autonomiczne, których ruch nic nie ogranicza (np. przewody zasilające czy sterujące), i nieautonomiczne.

2.3 Lokalizacja robota mobilnego na podstawie informacji wizyjnej

Jeśli chce się planować jakieś działania i ruchy, to oprócz znajomości otoczenia wymagana jest znajomość dokładnej pozycji robota. Przez lokaliza-

cję rozumie się ustalenie pozycji i orientacji robota względem arbitralnie przyjętego układu współrzędnych. Jeśli odbywa się to tylko na podstawie pomiarów sensorów własnych robota to wówczas takie działanie nazywane jest samolokalizacją [217]. Początkowo starano się zlokalizować robota w globalnym układzie współrzędnych, porównując to co widzi z dostarczoną mu wcześniej reprezentacją świata–mapą, czyli wykonywano lokalizację absolutną. Lokalizacja lokalna (inkrementalna) polega na ustaleniu położeniu robota względem poprzedniego położenia, do czego wykorzystuje się informacje o jego przemieszczeniu. Należy pamiętać, że takie zadanie cechuje się nakładaniem błędów powstałych w obliczaniu poszczególnych przemieszczeń.

Mapa może przyjąć różne formy. W przypadku lokalizacji na podstawie informacji wizyjnej, może to być zbiór landmarków–punktów orientacyjnych, lub ogólniej rzecz ujmując zestaw zdjęć z przypisanym do nich położeniem (ustalonym np. za pomocą systemu GPS), lub bazująca na strukturze mapa $3D$, składająca się np. z chmury punktów albo siatki zajętości w przestrzeni.

Oprócz wymienionych podejść pozwalających lokalizować robota, istnieje również wiele innych, także tych hybrydowych, starających się łączyć zalety wybranych metod z różnym skutkiem. Od niedawna wykorzystuje się sieci neuronowe do dopasowywania cech znalezionych na obrazie do mapy $3D$. Jednak tak nauczony model działa dobrze tylko w środowisku, w którym był uczony i zazwyczaj nie uogólnia się dobrze na inne [234, 244].

2.3.1 Lokalizacja na podstawie mapy złożonej z landmarków

By ułatwić robotowi zadanie pozycjonowania, w środowisku pracy często umieszcza się specjalne sztuczne znaczniki (landmarki). Mogą być aktywne (laserowe, radiowe) lub pasywne (odbłyśniki lub znaki graficzne). Choć ostatecznym celem jest eliminacja konieczności rozmieszczania takich znaczników, to w wielu aplikacjach robotów nadal się je wykorzystuje. Przyjmują one różne formy, dla przykładu w dokładnym ustalaniu pozycji za pomocą systemu GPS wykorzystuje się stacje referencyjne.

Jeśli skupimy się na ostatnim przypadku, znacznikach graficznych, to taka lokalizacja polega na wybraniu (wedle przyjętego kryterium) z zestawu zdjęć, stanowiących mapę, tych najbardziej podobnych do tego co

widzi teraz robot, opcjonalnie przypisując im różną rangę czy wagę. Za podobne zdjęcia uważane są te ukazujące ten sam obszar przestrzeni. Wtedy mówi się o odnajdywaniu landmarku. Można również wybrać zdjęcia znajdujące się w sąsiedztwie, co postrzegane jest jako geolokalizacja, rozpoznawanie miejsca [106]. Istnieją również podejścia wykorzystujące metody podobne do worka słów BoW (ang. *Bag of Words*) [148].

Taka lokalizacja jest wykorzystywana do rozpoznawania miejsc, w którym dane zdjęcie mogło zostać zrobione. W tym celu można próbować wykorzystać wewnętrzne parametry kamery do oszacowania względnego położenia robota względem wykrytego landmarku, wybrać adekwatne miejsca obok i na ich podstawie ustalić pozycję robota albo interpolować pozycję kilku najbardziej podobnych podobnych zdjęć.

Oczywiście można tą metodę wykorzystać do ograniczenia istniejącej mapy w zadaniu lokalizacji na podstawie mapy $3D$. Można też na podstawie wybranego z mapy zestawu zdjęć stworzyć lokalną mapę $3D$, ale to nie zawsze musi się udać (bo np. będzie zbyt mało zdjęć interesującego nas fragmentu przestrzeni, sceny).

2.3.2 Lokalizacja na podstawie mapy $3D$, bazująca na strukturze

Inną reprezentacją przestrzeni może być pełna mapa $3D$, odtwarzająca, rekonstruującą (stąd w literaturze anglojęzycznej te metody nazywane są bazującymi na strukturze) daną przestrzeń. Na taką mapę może składać się pewien zbiór istotniejszych, kluczowych punktów posiadających znane, dokładne położenie i opis pozwalający na ich późniejsze zidentyfikowanie i sparowanie z punktami widzianymi przez robota na obrazie.

W takim przypadku lokalizacja polega na próbie określenia miejsca z którego kamera robota widzi ten obraz.

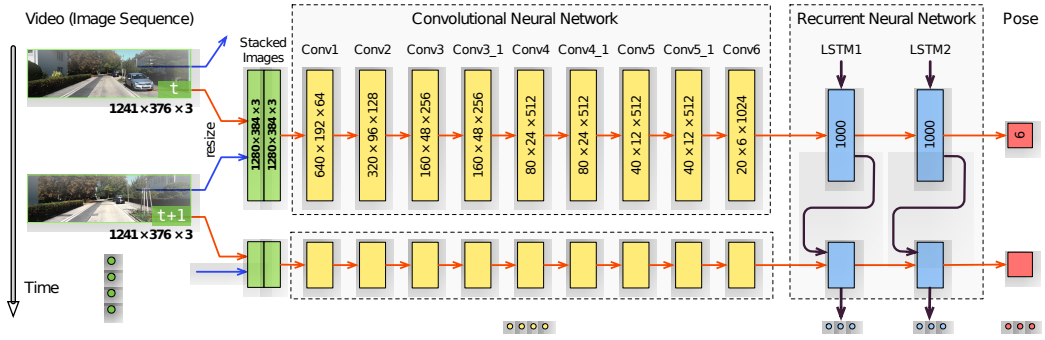
Zarówno wadą, jak i zaletą takiego rozwiązania jest w zasadzie nieograniczony rozmiar tak powstałej mapy. Duża mapa będzie zajmowała dużo miejsca w pamięci. To z jednej strony pozwoli na dokładniejszą lokalizację jeśli jest szczegółowa na małym obszarze, bądź na ustalanie pozycji na większym (w przeciwnym przypadku). Z drugiej zaś, wymusi wyposażenie robota w odpowiednią ilość pamięci co przełoży się na jego zwiększony koszt a czasem też masę. Zwiększy to również nakłady obliczeniowe oraz wydłuży proces dopasowywania, który w skrajnych przypadkach może się to okazać niemożliwy do zrealizowania–bo w bardzo dużej mapie

może istnieć kilka punktów o tym samym, albo mocno zbliżonym opisie. Ten problem można rozwiązać wymuszając najpierw rozpoznanie konkretnego miejsca, a następnie wycinając z niego mniejszą, lokalną mapę.

2.3.3 Lokalizacja na podstawie metod głębokiego uczenia

Istnieją również próby wykorzystania głębokiego uczenia i w tym kontekście. Jedna z koncepcji–PoseNet [155] opiera się na sieci CNN (uczonej do zadań klasyfikacji) i zamianie w niej warstw softmax na warstwy w pełni połączone, służące do oszacowania pozycji oraz orientacji kamery. Do zalet należy rezygnacja z ręcznie tworzonych cech typu SIFT, co czyni je odporniejszymi na typowe wady tradycyjnego podejścia. Tak jak w każdej sieci, można przyjąć stały czas inferencji, a zapisana w niej mapa zajmuje mniej miejsca. W dodatku, metody douczania sieci pozwalają na jej efektywne uczenie do pracy w innym miejscu.

Innym przykładem sieci działającej na podstawie sekwencji zdjęć jest DeepVO [234], której architekturę przedstawiono na rys. 2.8. Każde zdjęcie z sekwencji jest wstępnie przetworzone poprzez odjęcie średnich wartości RGB występujących w danych uczących z opcjonaną zmianą ich rozmiaru do wielokrotności 64 pikseli. Z dwóch, występujących po sobie ujęć tworzy się tensor. By wydobyć informacje o ruchu i oszacować pozycję jest on podawany na wejście rekurencyjnej, neuronowej sieci konwolucyjnej RCNN (ang. *Recurrent Convolutional Neural Network*)–wydobyte przez sieć CNN cechy są dalej wykorzystywane przez sieć rekurencyjną RNN.



Rysunek 2.8: Architektura RCNN monokularowego systemu VO [234]. Wymiary tensorów są przykładowe

Jednakże takie sieci osiągają znacząco gorsze wyniki niż obecnie najlepsze i tradycyjne metody oparte o mapę $3D$. Pewną przesłanką tego stanu rzeczy może być to, że na sieć neuronową można spojrzeć jak na pewnego rodzaju skompresowaną mapę otoczenia (która w dodatku ma maksymalny rozmiar wpisany w swoją architekturę), a to ogranicza siłę uogólniania sieci [148].

2.4 Lokalizacja przy braku znanej mapy otoczenia

Dla robotów nieco trudniejszym zagadnieniem jest lokalizacja bez znanej mapy otoczenia. Do określenia miejsca, w którym się znajdujemy można podejść na dwa sposoby: bez jednoczesnej budowy mapy (odometria wizyjna VO [ang. *Visual Odometry*]) oraz z jej budową (jednoczesna samolokalizacja i tworzenie mapy SLAM [ang. *Simultaneous localization and mapping*]). Oba te podejścia były i nadal są intensywnie rozwijane, gdyż niezawodna lokalizacja ma ogromne znaczenie w wielu zastosowaniach. Pierwsze podejście zwykle wykorzystuje większą liczbę, pomiędzy występującymi po sobie klatkami obrazu, pomiarów.

Odometria wizyjna może stanowić podstawę do budowy systemu SLAM opartego na optymalizacji pozycji (ang. *pose-based SLAM*) [99, 130]. W dodatku te metody można dość łatwo łączyć z innymi metodami służącymi do estymacji pozycji robota [124]. W swojej pracy Schmidt i inni [210] podjęli próbę połączenia obu podejść (SLAM i VO)–rozszerzyli metodę SLAM poprzez zastąpienie zmiany rotacji co kilka kroków predykcji EKF poprzez tą proponowaną przez oddzielnie działający system odometrii wizyjnej.

Dość obszerne porównanie metod odometrii wizyjnej VO można znaleźć w pracach [207, 130] jak i pierwszych stronach [185], gdzie je dość ładnie i krótko opisano VO i SLAM. Inne porównanie tych metod przedstawiono w [99].

2.4.1 Odometria wizyjna VO

Ogólną ideą tego podejścia jest ustalenie własnej pozycji i orientacji względem poprzedniej. Ciąg takich ustaleń (w stylu klatka po klatce) prowadzi do powstania tak zwanej samolokalizacji zliczeniowej. Jeśli do tego celu wykorzystywane są dane pochodzące z kamery, to wówczas mówimy o

odometri wizyjnej VO (ang. *Visual Odometry*) [12, 207]. Może być monokularowa bądź korzystająca z wielu kamer (w przypadku dwóch nazywa się ją stereowizyjną). Połączenie danych RGB wraz z głębią (RGB-D) poprawia dokładność oraz odporność rozwiązania VO [130]. Jednak ze względu na cenę i złożoność obliczeniową systemów wielokamerowych zwykle stosuje się rozwiązania bazujące na tylko jednej kamerze. W zamian powstają problemy związane z ustaleniem skali przesunięcia. Do rozwiązania tego typu zadania stosuje się algorytmy n-punktowe, które są opisane w rozdziale 3.5.1.

Poza zwykłymi, pasywnymi kamerami (utrwalającymi tylko dane RGB), na rynku znajdują się również czujniki dodatkowo dostarczające mapy głębi D–pomiary odległości do najbliższego widzianego obiektu dla danego piksela. Akwizycja tych informacji najczęściej dokonywana jest aktywne– albo wysyłany jest specjalny wzór, albo jest to kamera typu ToF. Powoduje to konieczność odpowiedniej korekcji odczytanych odległości, bo nie robi tego kamera RGB tylko odbiornik znajdujący się fizycznie obok. Zwykle jest to zrobione przez bibliotekę podczas pobierania danych albo jest to już zrobione w benchmarkowych zestawach danych. W ten sposób jesteśmy w stanie dokładnie wyliczyć o ile przemieścił się robot, bez stosowania stałej skali. Można to zrobić stosując np. algorytm Kabscha [154].

Zwykle nie wszystkie widziane przez kamerę piksele są wykorzystywane (bez względu na korzystanie z danych głębi) a jedynie ich część (choć istnieją podejścia wykorzystujące wszystkie), posiadająca jakieś istotne, szczególne informacje. Taki piksel nazywany jest cechą (ang. *feature*) lub punktem kluczowym (ang. *keypoint*). Spowodowane jest min. to złożonością obliczeniową, wymagającą posiłkowania się kartą graficzną, lecz bez gwarancji dokonania obliczeń w rozsądnym czasie (co w takim zadaniu jest niemal niemożliwe). Oznacza to zwiększony pobór energii a więc wymóg wyposażenia robota w większy akumulator. To pociąga za sobą większe koszty jak i wzrost masy co, z punktu widzenia robotyki mobilnej, stanowi znaczną przeszkodę. Natomiast wyekstrahowane informacje mogą być dalej wykorzystane w dalszych zadaniach takich jak rozpoznawanie miejsc i obiektów, tworzenia mapy środowiska i wielu innych, co jest też przyczyną ich niezwykłej popularności.

Istnieje wiele sposobów wykrywania tego typu punktów. Następnie trzeba dopasować cechy z jednego ujęcia z tymi widzianymi na drugim–a więc konieczna jest możliwość opisu punktu. Dalej odrzuca się te odstające (ang. *outliers*). Te problemy oraz sposób wyznaczania na podstawie

punktów przebytej trajektorii stanowią bloki, z których buduje się systemy odometrii wizyjnej–na każdym etapie można skorzystać z różnych algorytmów. Każdy z nich posiada parametry, które należy odpowiednio dobrać. Trudno jest stwierdzić jak dobrać te parametry oraz to z jakich dokładnie bloków powinny się składać tego typu systemy [130, 99]. Niektóre z istniejących podejść lokalizacji RGB-D [122] dobierały parametry do konkretnego eksperymentu poprzez zupełne przeszukanie przestrzeni możliwych rozwiązań dla danej sekwencji RGB-D. Jednakże jest to nieefektywne czasowo, a ponadto, nie daje ono wglądu w zależności między parametrami a jakością otrzymywanych estymat [99].

W tego typu podejściach nie jest tworzona mapa, a estymowana trajektoria robota/sensora niekoniecznie jest optymalna w świetle wszystkich zgromadzonych danych–zazwyczaj widoczny jest dryft wraz z przebytą trajektorią, którego nie da się skompensować np. za pomocą metod grafowych. Z uwagi na problemy z dokładnością pomiarów, powstałe błędy nakładają się w przypadku wyznaczania przebytej trasy. W odróżnieniu od tradycyjnej odometrii bazującej na odczytach np. z enkoderów silników, pomija się tu problemy robotów kołowych czy kroczących takich jak poślizg powodujące niejednorodny ruch. Równania wykorzystujące informacje o prędkości, z uwagi na całkowanie, są bardzo wrażliwe na błędy, szumy i inne niedokładności.

Murphy i inni [185] przedstawili porównanie trzech technik VO: bazującej na łatach, odometrii wizyjnej wykorzystującej wyznaczanie struktury uzyskanej z ruchu SfM, monokularową wersję podejścia [237]), oraz fuzji odometrii kołowej i wizyjnej. Ostatnie podejście oblicza średnią ważoną oszacowań przebytej trajektorii, jaką obliczają te dwa podejścia. Waga odpowiada estymacie błędu obliczonej przez odometrię wizyjną. Później każda uzyskana przez system VO estymata trajektorii jest przeskalowywana poprzez wykorzystanie odometrii kołowej w tym samym kroku czasowym.

2.4.2 Jednoczesna samolokalizacja i tworzenie mapy–SLAM

Najogólniej rzecz ujmując chcielibyśmy, by nasz robot mógł się dobrze poruszać i wykonywać powierzone mu zadania bez względu na to, czy będzie to znane już środowisko, czy też nie. W związku z tym autonomiczny robot powinien być w stanie budować mapę, na podstawie widzianego

środowiska, móc ją poprawiać oraz lokalizować swoją pozycję (w odniesieniu do niej). Innymi słowy, chcemy, by nasz robot był w stanie podołać zadaniu typu SLAM, to jest dokonywać jednoczesnej lokalizacji i tworzenia mapy (ang. *Simultaneous localization and mapping*) [13].

W związku z tym, gdy robot rozpoczyna swoje działanie w nieznanym środowisku, bez dostępnej mapy powinien rozpocząć jej tworzenie na podstawie odczytów z sensorów określających położenie obiektów czy charakterystycznych cech względem układu odniesienia robota. Ponieważ robi to od początku, to ten szczególny układ nazywa się początkowym układem współrzędnych. By powstała mapa była użyteczna, to wyniki wszystkich pomiarów muszą być umieszczone we wspólnym, zewnętrznym względem robota, układzie współrzędnych, co pociąga za sobą konieczność znania położenia w nim robota. Z tego powodu zakłada się że początkowe położenie robota jest w centrum zewnętrznego układu odniesienia (choć może być arbitralne to nie ma to większego znaczenia).

Następnie w procesie samolokalizacji odwołuje się do informacji przestrzennej opisującej otoczenie–mapy, przy czym zakładamy statyczne środowisko. Wykryte przez robota w środowisku cechy, stanowiące mapę lokalną, można skojarzyć z ich odpowiednikami zapisanymi w mapie. Dodatkowo, wykorzystując odometrię, można ograniczyć obszar mapy globalnej wykorzystywany do poszukiwania dopasowania. Dalej wykryte na obrazie punkty kluczowe można umiejscowić w określonym układzie odniesienia porównując je z cechami, których się spodziewamy wykryć na danym fragmencie mapy w danym momencie. Dzięki temu jesteśmy w stanie ustalić pozycję robota. Natomiast pozostałe, nieskojarzone jeszcze cechy można dodać do mapy.

Oba zagadnienia są ze sobą ściśle związane, ponieważ wykorzystują te same, niepewne (ze względu na błędy pomiarowe) informacje o otoczeniu, a zbudowana na ich podstawie mapa i ustalona na niej pozycja robota również będą obarczone niepewnościami.

Do tradycyjnych podejść można zaliczyć te stosujące filtr Kalmana bądź nieco bardziej niezawodny filtr cząsteczkowy [96]. W obydwu rozwiązaniach zakłada się poprawne kojarzenie danych pomiarowych z mapą (obserwacji i predykcji). Następnie zaadoptowano nieliniowe techniki optymalizacji. Doprowadziło to do rozwoju podejść bazujących na grafach [134]. Grafowe podejścia SLAM obliczają pozycję sensora, a następnie tworzą graf, którego wierzchołki stanowią pozycje, a krawędzie ograniczenia związane z ruchem [99]. Ten graf jest następnie optymalizowany, by zmini-

malizować rozbieżności (błędy) pomiędzy oszacowaną a mierzoną pozycją wierzchołków. Gdy robot ponownie odwiedza znany teren, następuje zamknięcie pętli wprowadzające ograniczenia pomiędzy pozycjami niwelujące dryft trajektorii. W przypadku obrazów RGB-D, ograniczenia ruchu pomiędzy klatkami da się obliczyć stosując bezpośrednie metody (np. [156]), albo poprzez dopasowywanie rzadkich cech za pomocą deskryptorów [121]. Podejścia SLAM bazujące na pozycjach pomijają ograniczenia pomiędzy cechami a pozycjami, powstałe z częstej obserwacji tych samych cech [224]. Te ograniczenia są wykorzystywane w metody regulacji wiązki BA (ang. *Bundle Adjustment*) [230], by łącznie optymalizować pozycje sensora i cech. Tą metodę zastosowano po raz pierwszy w systemie PTAM [159] i nadal jest stosowana we współczesnych systemach nawigacji monokularowej czy stereo, np. ORB-SLAM [183, 184, 109].

EKF SLAM

Filtrowanie przypomina nieco rozumowanie, które łączy wcześniejszą wiedzę i zaobserwowane dowody, aby wywnioskować cechy charakterystyczne interesującego zjawiska [222]. Jest to poprawa wiedzy którą posiadamy o stanie systemu (chcemy uzyskać jego jak najlepsze oszacowanie w chwili obecnej) na podstawie informacji uzyskanych z pomiarów na nim dokonywanych. Można do tego wykorzystać całą wiedzę o pomiarach albo tylko poprawiać ostatnie oszacowanie bazując na najświeższych pomiarach (a więc działamy rekursywnie). Skupiamy się na tym późniejszym wariancie, gdyż może działać online.

Stan systemu w czasie k można odnieść do $k-1$ poprzez dwa mechanizmy. Najpierw, w kroku predykcji, propaguje się zaobserwowane (posteriori), w czasie $k-1$, gęstości prawdopodobieństwa stanu systemu jako a priori dla czasu k, poprzez wykorzystanie wiedzy o dynamice systemu i perturbacjach (modyfikuje się rozkład prawdopodobieństwa z poprzedniego kroku).

Następnie krok korekcji, poprawy (albo aktualizacja pomiarów) pozwala na poprawienie wcześniejszych (a priori) przewidywań na końcowe oszacowanie poprzez wykorzystanie przesłanek jakie dostarczają nowe pomiary wykorzystując w zasadzie regułę *Bayes'a*. Na tym etapie integrujemy informacje pochodzące z obserwacji. Można to robić osobno dla każdego landmarku, po czym trzeba odpowiednio to uwzględnić w głównych macierzach.

W praktyce stosowany jest rozszerzony filtr Kalmana, bo na ogół większość rzeczywistych funkcji jest nieliniowa. Zmiana polega na tym, że w każdym kroku dokonuje się linearyzacji (wyliczając odpowiednie jakobiany) nieliniowych funkcji wokół najświeższej, najlepszej estymaty, a dalej stosuje się równania filtru Kalmana.

Roboty mobilne zwykle posiadają dostęp do odometrii–lepszego oszacowania przemieszczenia niż matematyczny model pojazdu. W związku z tym Jednoczesna samolokalizacja i tworzenie mapy z wykorzystaniem rozszerzonego filtru Kalmana zmienia krok predykcji poprzez wykorzystanie pomiaru obecnego położenia po skończonym ruch zamiast sterowania z poprzedniej $(k-1)$ chwili.

W tym podejściu mapa przybiera formę dużego wektora, zbudowanego ze stanów sensora i landmarków (punktów orientacyjnych), modelowanego jako zmienna losowa o rozkładzie gaussowskim. Utrzymuje się ją podczas procesu przewidywania (gdy sensor się porusza) jak i poprawy (gdzie sensor widzi landmarki wcześniej dodane do mapy). Końcowe rozkłady prawdopodobieństw są przybliżane za pomocą rozkładu normalnego reprezentowanego przez macierz wartości średnich μ i macierz kowariancji Σ. Macierz taką w poglądowy sposób przedstawia rys.2.9–na żółto zaznaczono części dotyczące pozycji robota R, niebiesko landmarków M a zielono mieszane $RM = MR$ (równe z uwagi na symetryczność macierzy kowariancji), dotyczące korelacji między pozycją robota a landmarkami.

$$\underbrace{\begin{pmatrix} x \\ y \\ \theta \\ m_{1,x} \\ m_{1,y} \\ \vdots \\ m_{n,x} \\ m_{n,y} \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{n,x}} & \sigma_{xm_{n,y}} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{m_{n,x}} & \sigma_{m_{n,y}} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta m_{1,x}} & \sigma_{\theta m_{1,y}} & \cdots & \sigma_{\theta m_{n,x}} & \sigma_{\theta m_{n,y}} \\ \sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{\theta} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{n,x}} & \sigma_{m_{1,x}m_{n,y}} \\ \sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{\theta} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{n,x}} & \sigma_{m_{1,y}m_{n,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{n,x}x} & \sigma_{m_{n,x}y} & \sigma_{\theta} & \sigma_{m_{n,x}m_{1,x}} & \sigma_{m_{n,x}m_{1,y}} & \cdots & \sigma_{m_{n,x}m_{n,x}} & \sigma_{m_{n,x}m_{n,y}} \\ \sigma_{m_{n,y}x} & \sigma_{m_{n,y}y} & \sigma_{\theta} & \sigma_{m_{n,y}m_{1,x}} & \sigma_{m_{n,y}m_{1,y}} & \cdots & \sigma_{m_{n,y}m_{n,x}} & \sigma_{m_{n,y}m_{n,y}} \end{pmatrix}}_{\Sigma}$$

Rysunek 2.9: Macierze opisujące pozycję i mapę robota w EKF SLAM

Celem jest dbanie by ten rozkład był aktualny w sytuacji, gdy:

1. robot się poruszy,
2. zostanie dostrzeżony obecny w mapie landmark,

3. zostanie wykryty nowy landmark i robot będzie chciał dodać go do mapy.

W dwóch pierwszych przypadkach wykonywany jest odpowiednio krok predykcji (ruch robota) oraz krok korekcji (dostrzeżenie obecnego w mapie landmarku). Gdy wykryty zostanie nowy landmark, wówczas jest on dodawany do mapy poprzez wykorzystanie odwróconej funkcji obserwacji, wraz z jej Jakobianem, do wyliczenia z pozycji i pomiarów sensora stanu landmarka wraz z niezbędnymi kowariancjami i wariancjami skośnymi z pozostałą częścią mapy. Te relacje są następnie dodawane do wektora stanu i macierzy kowariancji–podsumowuje to równanie (2.8):

$$p(\mathbf{x}_{R(k)}, \mathbf{m} \mid \mathbf{z}^k, \mathbf{u}^k) = \tag{2.8}$$

$$= \eta \overbrace{\left(\int p(\mathbf{x}_{R(k)} \,|\, \mathbf{x}_{R(k-1)}, \mathbf{u}_k) p(\mathbf{x}_{R(k-1)}, \mathbf{m} \,|\, \mathbf{z}^{k-1}, \mathbf{u}^{k-1}) d\mathbf{x}_{R(k-1)} \right)}^{\text{etap predykcji}} \overbrace{p(\mathbf{z}^k \,|\, \mathbf{x}_{R(k)}, \mathbf{m})}^{\text{etap korekcji}}$$

gdzie η jest stałym czynnikiem normalizującym, $\mathbf{x}_{R(k)}$ oznacza pozycję robota, $\mathbf{z}^k$ jest obserwacją, a $\mathbf{u}^k$ sterowaniem w k-tej chwili.

EKF SLAM ma kilka właściwości związanych ze zbieżnością. Gdy liczba obserwacji dąży do nieskończoności, to cechy występujące na mapie stochastycznej stają się całkowicie skorelowane. Dodatkowo minimalna kowariancja dowolnej cechy jest określona przez początkową niepewność pozycji robota.

Cały algorytm nie musi działać ze stałym krokiem, a predykcję i korekcję można wykonywać niezależnie. Może wystąpić sytuacja, że wykonano kilka kroków predykcji (bo np. w chwili k nie zaobserwowano żadnych cech) albo korekcji.

W rzeczywistości to proste podejście ma kilka problemów, do których można zaliczyć:

- mijające się ze stanem faktycznym wyidealizowane założenia co do rozkładów prawdopodobieństw (nie zawsze są normalne);
- modelowane procesy obserwacji i ruchu robota mają wiele nieliniowych składników i nie są aż tak liniowe jakbyśmy tego chcieli,
- estymata stanu, wokół której dokonywana jest linearyzacja, nie jest idealna.

Prowadzi to do złych dopasowań obserwowanych cech z mapą bądź dodawaniem nieistniejących cech. Ponieważ dysponujemy tylko jedną mapą, to tych błędów nie da się dalej naprawić. Nie ma również możliwości reprezentowania niejednoznacznej pozycji robota. W efekcie tak „uszkodzona" mapa utrudnia dopasowanie kolejnych cech. Natomiast dość poważną wadą wynikającą z natury tego rozwiązania jest fakt, że nie nadaje się ono do tworzenia zbyt dużych map–wraz z jej rozrostem drastycznie rośnie koszt obliczeniowy. Pierwszy taki działający w czasie rzeczywistym system zaproponowali Davison i Murray [116].

FastSLAM

To podejście [131] jest podobne do EKF SLAM. Wektor stanu dzieli się na dwie oddzielne grupy (np. stan robota i położenia landmarków). Z wektorem stanu robota związane są główne problemy z nieliniowościami, więc jego rozkład prawdopodobieństwa przybliża się cząstkami wykorzystując do tego filtr cząsteczkowy. Położenia landmakrów są zwykle modelowane rozkładem normalnym, a dodawanie nowych, nie widzianych wcześniej punktów orientacyjnych odbywa się tak jak w EKF SLAM.

Z uwagi na to, że połączenia między robotem a landmarkami powstają tylko i wyłącznie na podstawie pomiarów robota to, gdy jego rozkład jest cząstką, pozwala na dekorelację zestawów robot-landmark dając w wyniku pojedynczą pozycję robota (cząstkę) i zestaw niezależnych Gaussowych estymat landmarków. Podczas redystrybucji położenia cząstek można wykorzystać obecne, ostatnie pomiary by losować z bliższego do rzeczywistego, docelowego rozkładu prawdopodobieństwa. Dzięki temu można użyć mniejszej liczby cząstek uzyskując dokładniejsze rozwiązanie.

Zakładając, że dysponujemy dopasowanymi cechami, działanie algorytmu można ująć w następujących krokach:

- Dla każdej cząstki wyznaczamy nowe położenie na podstawie poprzedniego położenia i wydanego polecenia sterującego (korzystamy z dynamiki systemu) albo odczytu z odometrii.
- Dla każdej obliczamy jej wagę–czyli jak dobrze pasują do siebie obecna obserwacja i ta spodziewana.
- Niewidziane wcześniej cechy dodajemy do mapy, inicjalizując je odpowiednio.

- Aktualizujemy ufność co do zaobserwowanych landmarki, które są już obecne w mapie–wykonujemy krok korekcji EKF.
- Niezaobserwowane cechy pozostawiamy bez zmian.
- Redystrybucja cząstek.

Choć by takie podejście działało powinno zapisać się całą przebytą trajektorię robota, to w praktyce korzysta się tylko z ostatniej pozycji. Dla każdej cząstki robota ustanawia się zbiór rozszerzonych filtrów Kalmana, po jednym dla każdego landmarku. Odznacza się to stałym czasem aktualizacji, bo ogranicza to liczbę jednoczesnych pomiarów landmarków. W związku z tym całkowity rozkład prawdopodobieństwa tego systemu SLAM jest modelowany przez zbiór cząstek (delt Diraca) i zbiorem zbiorów EKF.

Zaletą tego podejście jest prostota reprezentacji wielu hipotetycznych skojarzeń pomiędzy obserwacjami cechy a jej występowaniem w mapie oraz nieco lepsze ujęcie występujących nieliniowości. Ceną jaką się płaci jest utrata deterministycznego, zamkniętego charakteru rozwiązania i wprowadzenie czynnika losowego (otrzymujemy kilka rozwiązań). Nie każda cząstka „widzi" te same i tyle samo landmarków.

2.5 Dostępne otwarto–źródłowe systemy SLAM

Obecnie istnieje wiele różnych metod optymalizacji wykorzystywanych w systemach odometrii wizyjnej VO czy jednoczesnej samolokalizacji i mapowania SLAM. Niniejszy podrozdział prezentuje ich wybrany podzbiór.

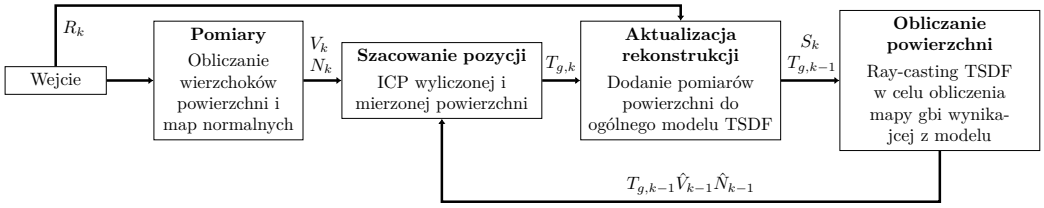
2.5.1 KinFu Large Scale: algorytm bazujący na KinectFusion i Kintinuous

Pojawienie się tego algorytmu jest konsekwencją wprowadzenia nowego, taniego typu sensora jakim jest Kinect i tak jak on proponuje nowe podejście do zagadnienia typu SLAM [187]. Dotychczas istniały algorytmy tego typu działające w czasie rzeczywistym ale tylko na danych z kamery RGB takie jak MonoSLAM [116, 115] czy system Parallel Tracking and Mapping

(PTAM) [159]. Skupiały się one głównie na śledzeniu kamery pozostawiając z boku zagadnienia związane z rekonstrukcją sceny, otoczenia.

W przeciwieństwie do nich, KinectFusion korzysta skupia się na dokładnej rekonstrukcji otoczenia (buduje globalną, wolumetryczną mapę) przy jednoczesnym śledzeniu pozycji sensora, w 6 stopniach swobody i wykorzystując wszystkie aktualne dane dostarczane z Kinecta, a nie podzbiór wybranych punktów–cech (np. opisanych keypointów) jak ma to miejsce w innych podejściach [115, 159, 122].

To podejście może działać bez przeszkód w kompletnych ciemnościach ponieważ nie korzysta z kamery RGB, a wyłącznie z emitera i odbiornika podczerwonego światła strukturalnego. By móc skorzystać z opisywanego algorytmu należy posiadać dedykowaną, kompatybilną z CUDA kartę graficzną. Rys. 2.10 przedstawia ogólny schemat działania omawianego algorytmu.

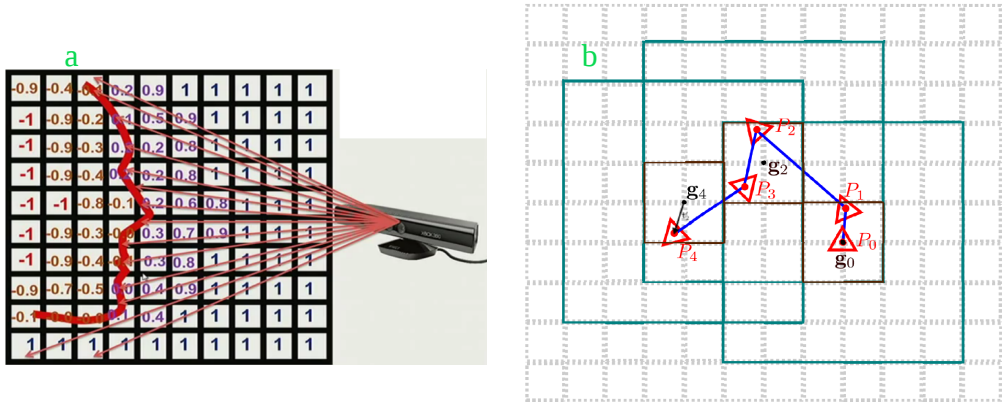


Rysunek 2.10: Schemat pracy algorytmu KinectFusion

Najpierw na podstawie dostarczanych przez Kinecta map głębi tworzy się wolumetryczną reprezentację przestrzeni TSDF (ang. *Truncated Signed Distance Function*). To nic innego jak przypisanie każdemu wokselowi, w modelowanym sześcianie, odległości do najbliższej powierzchni (z nasyceniem) i wagą, związaną z niepewnością. Po osiągnięciu przez funkcję dystansu ze znakiem pewnej arbitralnie określonej wartości należy obciąć dalsze pomiary by nie dopuścić do nakładania się pomiarów odległości pochodzących z kilku różnych powierzchni. Tak by np. tylna, niewidoczna część jakiegoś obiektu nie wpływała na pomiary dokonywane z przodu [114]. Poniższy rys. 2.11a ilustruje zasadę działanie funkcji TSDF.

Istnieją dwa główne parametry służące do określenia wolumetrycznej reprezentacji sceny–rozmiar w wokselach v_s i metrach d. Wymiar [238] pojedynczego woksela v_m oblicza się z zależności (2.9):

$$v_m = \frac{d}{v_s} \tag{2.9}$$



Rysunek 2.11: a) Idea funkcji TSDF [173] b) Schematyczne przedstawienie idei przemieszczania TSDF (w $2D$) [238]

Na podstawie tego modelu otoczenia tworzy się mapę wierzchołków i mapy normalnych o różnych rozdzielczościach–każda kolejna mapa zawiera mniej szczegółów.

Kolejnym krokiem jest estymacja pozycji sensora osiągana przez śledzenie urządzenia poprzez wielo-skalowe dopasowanie iteracyjne najbliższego punktu ICP pomiędzy przewidzianą wcześniej powierzchnią, a tą otrzymaną z aktualnych pomiarów sensora. W przypadku, gdy algorytm dopiero rozpoczyna swoją pracę, obecna powierzchnia traktowana jest jednocześnie jako modelowa.

Oszacowana pozycja Kinecta wykorzystywana jest do lokalizacji położeń punktów z dostarczonej, nowej mapy głębi w tworzonym modelu sceny. Następnie konstruuje się lokalną reprezentację przestrzeni i łączy się ją z obowiązującą, ogólną reprezentacją przestrzeni TSDF na zasadach sumy ważonej [198].

Ostatnim krokiem jest stworzenie mapy głębi, wynikającej ze zbudowanego modelu, na podstawie zaktualizowanego modelu i informacji o pozycji kamery (oznacza to śledzenie aktualnej klatki głębi, dostarczonej przez kamerę, w globalnym modelu). Tak stworzoną mapę wykorzystuje się następnie przy estymacji pozycji kamery dla nowych danych pobranych z sensora.

Algorytm KinectFusion (a więc i KinFu) cechował się licznymi ograniczeniami takimi jak nieelastyczny model powierzchni, który nie może w odpowiedni sposób modelować jej odkształceń, ograniczenie przestrze-

ni roboczej do $8m^3$ czy błędy śledzenia w prostym, nieskomplikowanym geometrycznie środowisku. Postanowiono je wyeliminować i udoskonalić istniejące podejście [238].

Śledzenie i mapowanie poprawiono w Kintinuous–nie są tu ograniczane do punktu zaczepienia w przestrzeni środka funkcji TSDF bo pozwala się na dynamiczne przemieszczanie się z czasem obszarów mapowanych przez wolumetryczną reprezentację przestrzeni. Ekstrahuje się regiony, które wychodzą z TSDF i zapisuje się je za pomocą oszczędniejszej siatki trójkątów.

By zmniejszyć błędy śledzenia w ubogim geometrycznie środowisku zaproponowano zastąpienie predykcyjnego podejścia ICP algorytmem odometrii wizyjnej FOVIS. Najpierw kolorowy obraz jest przetwarzany na czarno-biały i wygładzany filtrem Gaussa o $\sigma = 0,85$ a następnie, na podstawie uzyskanego w ten sposób obrazu, budowana jest „piramida" (każdy kolejny obraz jest czterokrotnie mniejszy i ponownie wygładzany filtrem Gaussa). Ma to na celu polepszenie detekcji cech kluczowych w różnych skalach. Do ekstrakcji cech wykorzystuje się detektor FAST dbając o wykrycie wystarczającej liczby cech kluczowych przy czym wyklucza się te bez informacji o głębi.

Każdy poziom piramidy jest dyskretyzowany do rozdzielczości 80×80 pikseli a następnie tylko 25 punktów kluczowych, z najlepszym wynikiem, jest branych pod uwagę. Kolejnym krokiem jest obliczenie wstępnej macierzy rotacji, obliczanie specjalnego deskryptora (zawiera poziomy jasności 9×9 punktów wokół opisywanej cechy, znormalizowane wokół 0 i pomijające prawy dolny piksel w celu korzystania z instrukcji bazujących na wektorach). Następnie dopasowuje się ze sobą cechy z nowej klatki ze starą, odrzuca złe powiązania lecz nie za pomocą algorytmu typu RANSAC lecz kierując się podejściem zaproponowanym przez Howarda [144]–liczenia grafu dobrych dopasowań i jego dalszym wykorzystaniu [140]. Dokładnie jest to opisane w [147].

Oszacowanie ruchu dokonywane jest w trzech etapach. Najpierw otrzymuje się wstępną odometrię przy wykorzystaniu algorytmu Horna [142], który minimalizuje euklidesowy dystans pomiędzy dopasowanymi wcześniej cechami kluczowymi. Tak uzyskany wynik poprawia się minimalizując błąd reprojekcji. Ostatnim krokiem jest odrzucenie tych dopasowań, dla których błąd reprojekcji przekracza ustalony próg, po czym powtarza się opisaną procedurę by uzyskać lepsze wyniki. W dodatku nową klatkę nie zawsze porównuje się z otrzymaną bezpośrednio przed nią ale naj-

pierw próbuje się ją porównać z kluczową klatką–taką dla której udało się poprzednio wyznaczyć przemieszczenie i rotację (jeśli się to nie uda to porównuje się ją z bezpośrednio ją poprzedzającą).

Jedną z wad, wynikającą ze stosowania metody najbliższego punktu jest możliwość utknięcia w minimum lokalnym w przypadku niefortunnego, początkowego dopasowania. Kolejną jest brak informacji o jakości dokonanego dopasowania

W tym celu dynamicznie, gdy kamera oddali się wystarczająco od środka TSDF (przekroczy konfigurowalny próg), zmienia się region wolumetrycznej reprezentacji przestrzeni–nowe regiony przestrzeni inicjalizuje się jako niezmapowane a poprzednie wyodrębnia się w postaci chmur punktów i zmienia na oszczędniejszą, zajmującą mniej miejsca w pamięci, siatką trójkątów według [174]. Wolumetryczna reprezentacja sceny jest przechowywana w pamięci karty graficznej podczas gdy siatka trójkątów zapisywana jest w pamięci operacyjnej.

Rys. 2.11b w poglądowy sposób przedstawia dynamiczne przemieszczanie się zmapowanych obszarów.

Globalna pozycja TSDF oznaczona jest przez g_i, pozycja kamery oznaczona jest przez P_i (Translacja i Rotacja względem początku układu współrzędnych), aktualne okno TSDF jest zaznaczone na seledynowo a granica po przekroczeniu której TSDF ulegnie przesunięciu na brązowo. Linia przerywana oznacza granice pojedynczego woksela. Gdy kamera przemieści się o określoną liczbę wokseli, w tym przypadku o 2, przesunięciu ulega cały TSDF.

By po przesunięciu TSDF system mógł funkcjonować tak jak oryginalny algorytm KinectFusion należy obliczyć lokalną translację $t'i$, to jest od środka aktualnej wolumetrycznej reprezentacji przestrzeni g_i.

Algorytm KinFu Large Scale jest otwarto-źródłowym, dostępnym w bibliotece PCL [199], rozszerzeniem algorytmu KinFu. Dla tych podejść nie zostały napisane żadne artykuły naukowe–ich twórcy dokonali implementacji algorytmów KinectFusion i Kintinunous [239]. KinFu Large Scale został jednak w minimalnym stopniu opisany na blogu developerów PCL [200]. To podejście wykorzystuje tylko dane o głębi D i wymaga do działania wsparcia GPGPU. Do określania pozycji wykorzystywany jest zmodyfikowany algorytm ICP–wykorzystywana jest przewidywana, za pomocą funkcji TSDF, powierzchnia jak i aktualna klatka głębi. Głównym celem algorytmów ICP jest wyznaczenie takiej rotacji i translacji, która lokalnie najlepiej dopasowuje punkty z dwóch kolejnych ujęć, zwykle w sensie nor-

my euklidesowej–minimalizuje sumę błędów dopasowań.

Najpierw dokonuje się wstępnego dopasowania, po czym powtarza się tą procedurę dla inaczej dopasowanych punktów, tak długo aż znajdzie się transformację o odpowiednio niskim błędzie, przerywając poszukiwania w dwóch przypadkach:

1. Kilka kolejnych prób znalezienia lepszej transformacji nie przynosi efektów. Przerwanie pracy w tej sytuacji przyspiesza działanie algorytmu.

2. Gdy przekroczona zostanie maksymalna liczba prób znalezienia translacji i rotacji spełniającej poprzednie warunki. Ma to na celu nie dopuszczenie do sytuacji, w której algorytm nie jest w stanie wyjść z pętli poszukiwania transformacji lub trwa to zbyt długo.

Jakość wyników dostarczanych przez ten algorytm silnie zależy od początkowego dopasowania jak i rodzaju środowiska w którym porusza się robot–np. w obszarach, gdzie powtarza się wiele elementów (korytarze) algorytm może nie znaleźć prawidłowej transformacji.

W algorytmie KinFu, a więc i KinFu Large Scale, dopasowania są wykonywane w różnych rozdzielczościach i zakłada się, że zmiany pomiędzy nową chmurą punktów a poprzednią są niewielkie, więc nie potrzeba dokonywać wstępnego dopasowania.

Pozwala to wszystko na zrównoleglenie algorytmu i wykorzystanie do pracy karty graficznej co przekłada się na przyspieszenie działania algorytmu i umożliwienie jego działania w czasie rzeczywistym.

Główny nacisk został położony na dokonaniu takiej modyfikacji algorytmu KinFu, która zniosłaby ograniczenia co do maksymalnego rozmiaru przestrzeni roboczej. Dokonano tego poprzez kompresję (z uwagi na niską przepustowość połączenia PCIe) i wysłanie do procesora części informacji z karty graficznej o zmapowanym sześcianie przestrzeni ($8m^3$) gdy kamera zbliży się do jego granicy. Dane zapisywane w pamięci RAM obejmują tylko wierzchołki trójkątów (tworzących siatkę) znajdujące się tuż przed powierzchnią, bądź za nią, czyli tym wokselom, którym funkcja TSDF przypisuje wartości z przedziału $[-1, 0.98]$. Woksele z wartością „1" są omijane gdyż nie niosą informacji o odległości do najbliższej powierzchni–jest to pusta przestrzeń. Dalej następuje obliczenie wymaganego przesunięcia, a to powoduje wyczyszczenie pamięci reprezentującej usuwaną z sześciana

części przestrzeni, bo odtąd ten sam obszar w pamięci będzie odpowiedzialny za nowe obszaryi–korzysta się z cyklicznego bufora $3D$. Ostatecznie aktualizuje się lokalny układ współrzędnych TSDF, tak by kamera była w jego środku–przesuwa się go o odpowiednią liczbę wokseli. Współrzędne kamery są globalne i nie są zmieniane, są wyrażone w metrach. Kolejny rekonstruowany obszar (sześcian) przestrzeni pokrywa się częściowo z poprzednim.

Do odtworzenia reprezentacji wolumetrycznej, gdy sensor ponownie wejdzie w dany obszar, wykorzystuje się tak zapisane dane. Wyekstrahowane woksele posiadają koordynaty w odniesieniu do ogólnego układu współrzędnych w wokselach. Ogólny model otoczenia jest przechowywany w pamięci procesora.

By KinFu Large Scale poprawnie działał, wymaga detekcji odpowiedniej liczby punktów kluczowych w otoczeniu i dbałości o właściwości geometryczne przestrzeni w której się porusza sensor. Algorytm ICP nie daje sobie rady z powierzchniami leżącymi na tej samej płaszczyźnie lub dużymi płaskimi powierzchniami takimi jak ściany czy podłogi. Z uwagi na brak optymalizacji algorytmu i większą liczbę operacji wykonywanych pomiędzy klatkami w (stosunku do pierwotnego KinFu) nie należy wykonywać chaotycznych, szybkich ruchów kamerą w szczególności przy przesuwaniu TSDF.

2.5.2 CCNY_RGBD

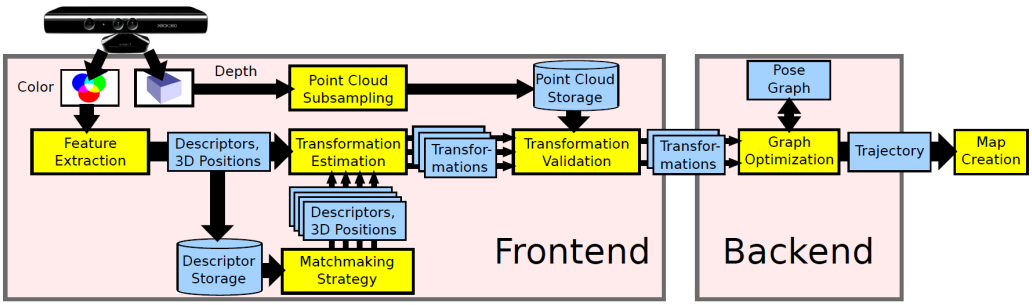
Algorytm zaprezentowany w [118] (implementacja bazująca na ROS jest dostępna pod adresem [14]) wykorzystuje informacje fotometryczne tylko do detekcji punktów kluczowych, nie tworząc ich deskryptorów (można wybrać różne detektory: ORB, SURF, STAR oraz GFT. W prezentowanych badaniach użyto detektora ORB). Na podstawie punktów kluczowych oraz informacji o głębi sceny tworzona jest globalna mapa punktowych cech $3D$. Każda cecha ma przypisaną niepewność swojej pozycji wyrażoną przez macierz kowariancji. Uzyskiwane z kolejnych ramek punkty kluczowe są dopasowywane do cech w mapie za pomocą zmodyfikowanego, rzadkiego algorytmu ICP korzystającego z (podobnej do Mahalonobisa) funkcji odległości. Pozwala to na wyznaczenie przemieszczenia sensora. By poprawić jakość dopasowania zaproponowano metodę szacowania niepewności mapy głębi z sensora Kinect. Dopasowania punktów do

mapy są wykorzystywane do aktualizacji pozycji cech w mapie przy wykorzystaniu filtra Kalmana, a niedopasowane, nowe cechy są dodawane do mapy. W przypadku, gdy mapa osiągnie zbyt duży rozmiar (w badaniach przyjęto za limit 10000 cech), najnowsze cechy zastępują te najstarsze. W systemie CCNY_RGBD możliwa jest dodatkowa optymalizacja offline grafu pozycji sensora, przy wykorzystaniu g^2o [168]. Przy założeniu, że wszystkie badane systemy działają w czasie rzeczywistym z możliwości tej nie skorzystano, demonstrując wynik użycia filtru Kalmana w zadaniu RGB-D SLAM.

2.5.3 RGB-D SLAM

W algorytmach typu KinectFusion każdy pomiar jest bezpośrednio integrowany z wokselowym modelem co zmniejsza dryft w stosunku do porównania klatki do klatki–metody stosowanej w RGB-D SLAM. Jednakże podejściom opartym na woksleowym modelu brakuje możliwości redukcji nagromadzonego dryftu po wykryciu zamknięcia pętli [121].

RGB-D SLAM to rozwiązanie (którego kod, bazujący na systemie ROS, dostępny jest pod adresem [15]) korzysta zarówno z danych RGB jak głębi D. Cechy punktowe oraz ich deskryptory (można wybrać SIFT, SURF albo ORB), są wykorzystywane do reprezentacji mapy jako grafu pozycji, ale dalej nie są jej częścią. Cały system algorytmu RGB-D SLAM można podzielić na trzy zasadnicze części: frontend, backend i końcową reprezentację mapy [121]. Na rys. 2.12 przedstawiono schemat działania algorytmu.



Rysunek 2.12: Poglądowy schemat działania algorytmu RGB-D SLAM [121]

Frontend ma za zadanie wydobyć zależności, powiązania geometrycz-

ne na podstawie danych z sensora i ustalić nową pozycję kamery (odometria typu klatka po klatce). W przypadku opisywanego algorytmu wykorzystywane są dane o głębi i kolorze otrzymywane z Kinecta. Praca zaczyna się od oznaczania landmarków: każdemu przypisuje się $64-$wymiarowy deskryptor $\boldsymbol{d} \in \mathbb{R}^{64}$, wraz z jego pozycją w przestrzeni $\boldsymbol{y} \in \mathbb{R}^3$ i jego położeniem w odniesieniu do aktualnej pozycji kamery $\boldsymbol{x} \in \mathbb{R}^6$. Dalej system stara się wykryć zamknięcie pętli by zredukować dryft.

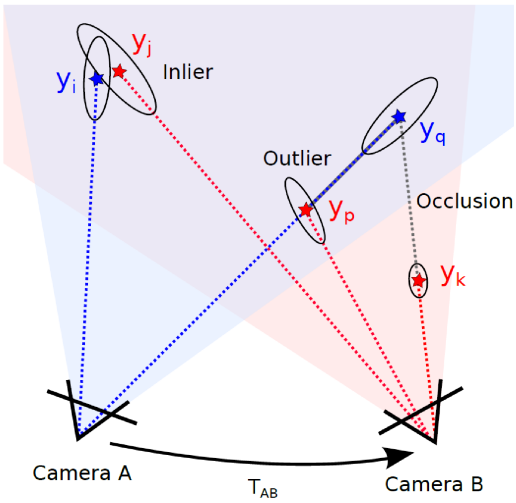
Zadaniem backendu jest konstrukcja grafu reprezentującego relacje geometryczne z niepewnościami co do nich i radzenie sobie z nimi. Tą strukturę wykorzystuje się następnie do wyznaczenia najbardziej prawdopodobnej trajektorii a dzięki niej tworzy się wokselową mapę zajętości poprzez projekcje danych z sensora do używanego układu współrzędnych.

Ponieważ zarówno algorytmy ICP jak i RANSAC nie zapewniają wykrywania wszystkich błędów, w RGB-D SLAM stworzono metodę weryfikacji estymowanej trajektorii niezależnie od metody użytej do jej oszacowania. Bazuje ona na wiązkowym modelu pomiaru środowiska EMM (ang. *Environment Measurement Model*). Odrzucane są te estymaty dla których pomiary sensorem są nieprawdopodobne–np. jakieś punkty z mapy głębi nie powinny być widoczne z danej pozycji bo powinny być zasłonięte przez inne, o których wiemy z innych pomiarów otoczenia. Głównym założeniem jest to, że po zastosowaniu oszacowanego przemieszczenia odpowiadające sobie pomiary głębi wynikają z położenia powierzchni w tym samym miejscu. Tą ideę można zobaczyć na rys. 2.13.

By udoskonalić odrzucanie złych sparowań stosuje się następnie algorytm RANSAC kilka razy ze zmiennym, malejącym progiem liczby dobrych dopasowań (ta metoda została zaproponowana w [112]). W celach weryfikacji korzysta się także z cech bez informacji o głębi. Polepsza to działanie algorytmu w pomieszczeniach i lokacjach gdzie występuje wiele jednakowych obiektów (ale za cenę wydajności).

Estymację ruchu oblicza się z ustalonych relacji $3D$ podczas każdej iteracji RANSACa przy wykorzystaniu metody najmniejszych kwadratów [231]. Estymację można jeszcze polepszyć minimalizując kwadrat dystansu Mahalanobisa zamiast Euklidesowego. To drugie podejście jest stosowane podczas optymalizacji grafu, już po estymacji ruchu, składającego się z tylko dwóch pozycji kamery i wcześniej określonymi *inlierami*–przynosi to niewielką poprawę jakości trajektorii.

W tym algorytmie zaimplementowane zostały następujące metody wyznaczania odległości dla dopasowania deskryptorów:



Rysunek 2.13: Schematyczne przedstawienie zasady zaliczania punktu jako *inlier* ($\boldsymbol{y}_i$ widziany na dwóch ujęciach i $\boldsymbol{y}_k$ tylko na jednym); *outlier* ($\boldsymbol{y}_p$ pomylony z punktem $\boldsymbol{y}_q$ przez kamerę A) i *przysłoniętego* ($\boldsymbol{y}_q$–kamerze B przysłania go punkt $\boldsymbol{y}_p$)

– dla SIFT i SURF można wybrać pomiędzy odległością Euklidesową a dystansem Hellingera [92]. W artykule [121] autorzy polecają wykorzystanie tego drugiego–dla niektórych zestawów danych zauważyli znaczną poprawę dopasowań (dla większości nieznaczną) bez zauważalnego obciążania systemu;

– dla ORB dystans Hamminga.

Takie podejście do każdej kolejnej klaki, prowadzi do otrzymania transformat cząstkowych a ich dodanie do odometrii wizyjnej. Efektem tego jest gromadzenia się z czasem dryftu, gdyż poszczególne transformaty wciąż zawierają dużo zakłóceń, tym więcej im mniej jest znalezionych punktów kluczowych i są one dalej–w szczególności gdy są poza zasięgiem.

By zredukować kumulowanie się zakłóceń stosuje się metodę heurystyczną: porównuje się aktualną klatkę z poprzednimi, poszukując możliwie najstarszej klatki dla której da się obliczyć transformację do aktualnej klatki. Dryft zostanie zredukowany, tym bardziej im więcej klatek uda się pominąć przy szacowaniu trajektorii.

Ponieważ takie podejście jest niewydajne, twórcy [121] proponują za-

stosowanie algorytmu wybierającego ujęcia najlepiej nadające się do porównania. Proponowane klatki pochodzą z trzech różnych grup:

- n bezpośrednio poprzednich klatek,
- k losowo klatek z tych leżących geodezyjnie blisko bezpośrednio poprzedniej, bez n ostatnich (by sztucznie nie dublować zbioru). Preferowane są te starsze.
- l losowo wybranych klatek ze zbioru klatek kluczowych. Ujęcie dołącza do zbioru kluczowych gdy nie da się go sparować z wcześniejszym ujęciem kluczowym.

Dwustronne przybliżenia transformacji (progresywnych i regresywnych) pomiędzy pozycjami sensora, obliczanymi przez frontend SLAMu (po zamknięciu pętli, jeśli nastąpiło), tworzą krawędzie grafu pozycji ale z uwagi na błędy oszacowań nie tworzą one ogólnie spójnej trajektorii. Aby ją policzyć należy dokonać optymalizacji za pomocą złożonego algorytmu (ang. *framework*) g^{2}o [168], przeznaczonego do minimalizacji nieliniowej funkcji błędu (która może być reprezentowana pod postacią grafu). Do tego celu wykorzystywane są trzy solvery. Dwa bazują na rozkładzie Choleskiego (CHOLMOD, CSpare) a ostatni implementuje (domyślnie wykorzystywaną) metodę sprzężonego gradientu z poprawianiem uwarunkowania PCG (ang. *Preconditioned Conjugate Ggradient*). Precyzując, minimalizowana jest funkcja błędu w następującej postaci:

$$\boldsymbol{F}(\boldsymbol{X}) = \sum_{\langle i,j\rangle \in \mathcal{C}} \boldsymbol{e}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij})^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij}) \tag{2.10}$$

Obliczona wcześniej trajektoria służy do rzutowania oryginalnych pomiarów punktów do ogólnego układu współrzędnych, czyli reprezentacji świata w postaci chmury punktów. By stworzyć mapę elementów powierzchni należy dostarczyć informacje o punkcie widzenia kamery. Takie modele są w dużym stopniu nadmiarowe i wymagają zarówno dużej mocy obliczeniowej jak i pamięci. W związku z tym często grupuje się w zbiory punkty z chmury i tworzy się ich reprezentację wokselową.

By pozbyć się ograniczeń wynikających z reprezentowana środowiska za pomocą chmury punktów, wykorzystano mapy zajętości $3D$. Skorzystano ze złożonego algorytmu (ang. *framework*) OctoMap [143], w którym

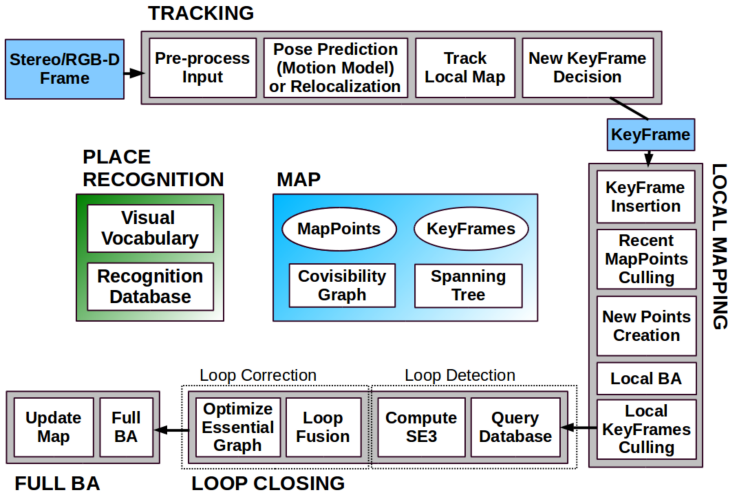
woksele utrzymywane są w drzewie. Ma to na celu zmniejszenie wykorzystania pamięci i zezwolenie na tworzenie map w różnych rozdzielczościach (wynika to ze struktury drzewa). Wykorzystanie probabilistycznego szacowania zajętości pozwala na kopiowanie zakłóconych pomiarów wraz z błędami w oszacowaniu pozycji co dalej umożliwia wyraźną reprezentację wolnej przestrzeni i niezmapowanych obszarów z zadaną rozdzielczością i sposobem jej zapisu (binarny w bardzo kompaktowy sposób zapisuje np. czy dane miejsce jest zajęte lub wolne).

2.5.4 ORB-SLAM2

ORB-SLAM2 jest drugim wariantem wizyjnego systemu SLAM przedstawionego w pracy [183]. Powstał jako monokularowy SLAM wykorzystujący koncepcję mapy cech punktowych i optymalizację niezależnych wiązek BA (ang. *Bundle Adjustment*) [230], do optymalizacji zarówno trajektorii kamery jak i mapy punktów kluczowych. Wersja druga [184], dostępna na stronie www twórców systemu [193] może jednak wykorzystywać dane RGB-D (albo pasywnych kamer stereo), dzięki czemu unika się problemu właściwej inicjalizacji mapy cech fotometrycznych (pochodzących z kamery monokularowej), dla których nie jest dostępna informacja o głębi. Graf ograniczeń traktuje zarówno pozycje sensora jak i pozycje cech jako wierzchołki, a jego optymalizacja realizowana jest w ORB-SLAM z użyciem biblioteki g^{2}o.

Architektura podzielona na front-end oraz przechowujący mapę i optymalizujący back-end. ORB-SLAM zachowuje wszystkie cechy monokularowego SLAM, uwzględniając także cechy pozbawione informacji o głębi i optymalizując mapę jedynie na podstawie błędu reprojekcji cech na obraz (nawet dla danych RGB-D, dających pozycję punktów $3D$). Pozwala to istotnie zwiększyć liczbę cech wykorzystywanych do dopasowań między mapą, a bieżącą percepcją, np. uwzględniając cechy leżące poza zasięgiem pomiaru odległości sensora RGB-D. Ponadto w ORB-SLAM zaimplementowano zamykanie pętli na podstawie rozpoznawania widoków sceny z użyciem algorytmu słownikowego (ang. *bag-of-words*). Umożliwia to zamykanie pętli o dowolnej wielkości, nie tylko lokalnych, choć proces ten dla bardzo dużych pętli nie odbywa się w czasie rzeczywistym. W ORB-SLAM zmodyfikowane, w pełni wieloskalowe cechy ORB (które dzieli na bliskie i dalekie) wykorzystywane są w całym systemie do śledzenia ru-

chu sensora, dopasowywania bieżącej percepcji do mapy (tworzenie ograniczeń) oraz zamykania pętli. Śledzenie wykorzystuje prosty model ruchu kamery zakładający ruch ze stałą prędkością, powodując wrażliwość tego systemu na nagłe zmiany ruchu. Schemat tego systemu przedstawiono na rys. 2.14.



Rysunek 2.14: Poglądowy schemat działania algorytmu ORB-SLAM2 [184]

2.6 Metody uzupełniania brakujących danych na obrazie

Techniki inpaintingu służą do cyfrowego uzupełniania (ang. *inpainting*) niewielkiej ilości brakujących danych obrazowych (również tych przeznaczonych do zastąpienia np. z powodu uznawania ich za uszkodzone, bądź niechciany tekst czy logo). W obecnej literaturze istnieje wiele podejść służących temu celowi.

Większość technik inpaintingu działa na podobnej zasadzie. Najpierw wybierane są części obrazu przeznaczone do uzupełniania–należy je zaznaczyć za pomocą odpowiednich masek. Następnie znane informacje o wartości funkcji obrazowej z obszarów przy nich (tych blisko granicy) są propagowane do wnętrza uzupełnianego obszaru. W trakcie uzupełniania powinno się przeciągać izofoty (linie na których wartości szarości–

intensywności jest ta sama), by tak przetworzony obraz wyglądał prawdopodobnie. Czyli by uzupełniona wartość w skali szarości i jej gradient ekstrapolowały te dwie rzeczy poza tym obszarem. Sprowadza się to do rozwiązania pewnego dobranego heurystycznie równania różniczkowego cząstkowego. Perfekcyjne osiągnięcie tego celu nie jest możliwe z uwagi na dyskretny charakter operacji–powstają numeryczne rozmycia.

Istnieją również rozwiązania, które wielokrotnie splatają prosty filtr 3×3 nad uzupełnianą częścią obrazu by rozmyć znaną informację do tych pikseli. Jako wadę takiego rozwiązania zalicza się brak gwarancji zachowania kierunku izofot co powoduje konieczność ręcznego zaznaczenia obszarów o dużym gradiencie wartości pikseli i ich specjalnego traktowania.

Spośród tych klasycznych rozwiązań wybrano dwa podejścia: algorytm Telei oraz Naviera-Stokesa. Oprócz tego posłużono się również metodą korzystającą z głębokiego uczenia, co zostało opisane w rozdziale 5.3.1.

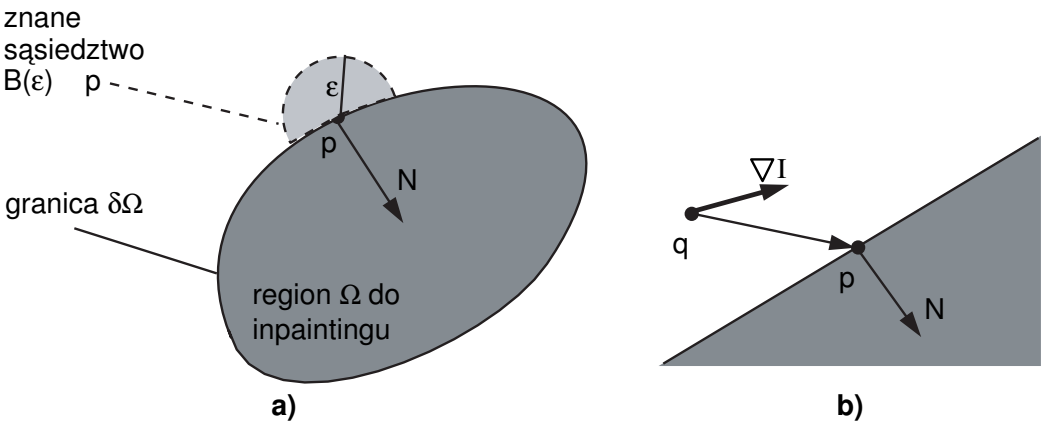
2.6.1 Algorytm Telei

Ta metoda inpaintingu [229] bazuje na propagacji estymatora gładkości, będącego średnią ważoną wartości znanych sąsiednich pikseli (dla tego podlegającego uzupełnianiu), wzdłuż gradientu obrazu. Brakujące obszary są traktowane jak poziomice. Do propagacji informacji z obrazu wykorzystywana jest metoda szybkiego marszu FMM (ang. *fast marching method*) [214] Sethian'a, która z kolei jest wersją metody zbiorów poziomicowych.

Zasadę działania tej metody prezentuje rys. 2.15, gdzie uzupełniamy punkt p znajdujący się na granicy $\partial\Omega$ regionu do uzupełniania Ω. W tym celu należy wziąć wartości pikseli znajdujących się w małym sąsiedztwie (znajdujących się w okręgu o promieniu o długości wyrażonej w pikselach, wybranym przez użytkownika) $B_\epsilon(p)$, o wielkości ϵ, znanego obszaru wokół punktu p (rys. 2.15a). Dalsze rozważania dotyczą obrazów szarych. Dla dostatecznie małego ϵ dokonuje się aproksymacji (2.11) $I_q(p)$ pierwszego rzędu w okół punktu p, przy wykorzystaniu danego obrazu $I(q)$ oraz gradientu $\nabla I(q)$ wartości punktu p (rys. 2.15b).

$$I_q(p) = I(q) + \nabla I(q)(p - q) \tag{2.11}$$

Następnie uzupełnia się punkt p jako wynik działania wszystkich punktów q w $B_\epsilon(p)$ jako ich zważoną sumę poprzez znormalizowaną funkcję ważącą $w(p, q)$, która zaprojektowana jest w taki sposób, że uzupełnianie p



Rysunek 2.15: Zasada inpaintingu Telei

propaguje wartości szarości oraz detale ostrości obrazu przez $B_\epsilon(p)$. Przedstawia to równanie (2.12):

$$I_p = \frac{\sum_{1 \in B_\epsilon(p)} w(p,q)\left[I(q) + \nabla I(q)(p-q)\right]}{\sum_{1 \in B_\epsilon(p)} w(p,q)} \tag{2.12}$$

Wykorzystanie metody szybkiego marszu FMM

By uzupełnić cały region Ω należy iteracyjnie stosować równanie (2.12) do wszystkich dyskretnych pikseli $\partial\Omega$ zwiększając dystans $\partial\Omega$ od początkowego, inicjowanego $\partial\Omega_i$ i przesuwać granicę wgłąb Ω aż uzupełni się cały ten obszar. Prezentuje to pseudokod 1. Takie postępowanie zapewnia, że punkty bliżej granicy $\partial\Omega$ są uzupełniane najpierw, co naśladuje ręczne techniki uzupełniania. W tym celu wykorzystywana jest metoda szybkiego marszu FMM. Zdecydowano się na nią, bo w przeciwieństwie do innych metod jawnie utrzymuje *cienką granicę* (którą jest tu $\partial\Omega$) pomiędzy znanym, a nie znanym obszarem 2.13:

$$|\nabla T| = 1 \quad \text{dla } \Omega, \quad \text{gdzie } T = 0 \text{ dla } \partial\Omega. \tag{2.13}$$

Rozwiązanie T jest mapą dystansów pikseli Ω do $\partial\Omega$. Te izolinie T to dokładnie kolejne granice $\partial\Omega$ kurczącego się obszaru Ω. Normalna N do $\partial\Omega$ to dokładnie ∇T. FMM gwarantuje, że piksele $\partial\Omega$ będą zawsze przetwarzane w narastającym porządku dystansów do granicy T, czyli, że najpierw uzupełniane są punkty będące najbliżej znanych części obrazu.

Algorytm 1: Algorytm inpaintingu Telei

$\partial\Omega_i =$ granica regionu do uzupełnienia
$\partial\Omega = \partial\Omega_i$
while *($\partial\Omega$ nie pusty)* **do**
    p = piksel z $\partial\Omega$ najbliższy do $\partial\Omega_i$
    uzupełnij p korzystając z równania (2.12)
    przesuń $\partial\Omega$ w kierunku Ω

Do każdego piksela będącego częścią obrazu przypisuje się 3 wartości: odległość T, jego wartość szarości I (obydwie są zmiennoprzecinkowe typu `float`) oraz flagę f, która także może przyjąć 3 wartości:

- Obwódka: piksel leży na granicy $\partial\Omega$. Jego wartość T jest aktualizowana.
- Znany: piksel jest poza granicą $\partial\Omega$, w znanym obszarze obrazu.
- Wewnątrz: piksle znajduje się wewnątrz $\partial\Omega$, w obszarze do uzupełniania. Jego wartości T oraz I są jeszcze nieznane.

Algorytm FMM inicjalizuje się poprzez ustawienie wartości T na 0 dla wszystkich pikseli poza granicą $\partial\Omega$ do uzupełniania a dla tych wewnątrz na jakąś bardzo dużą wartość (w praktyce 10^6), a następnie przypisywane są im adekwatne flagi f. Wszystkie punkty leżące na *obwodzie* są wkładane na stertę *cienkiej granicy*, która jest ułożona według ich rosnących wartości T. Następnie wartości T, f oraz I są propagowane, tak jak pokazuje to algorytm 2.

Najpierw (krok 1) wyciągane są punkty należące do *cienkiej granicy* o najmniejszej wartości T. W kroku 2 przesuwa się granicę do wewnątrz poprzez dodanie do niej nowych punktów. Krok 3 wykonuje operację uzupełniania, a 4 propaguje wartości T punktu (i, j) do ich sąsiadów będące rozwiązaniem skończonej różnicy zdyskretyzowanego równania (2.13), do przedstawia równanie (2.14):

$$max(D^{-x}T, -D^{+x}T, 0)^2 + max(D^{-y}T, -D^{+y}T, 0)^2 = 1. \tag{2.14}$$

gdzie $D^{-x}T(i,j) = T(i,j) - T(i-1,j)$, a $D^{+x}T(i,j) = T(i+1,j) - T(i,j)$ oraz podobnie dla y. Następnie idzie się „pod wiatr" (jak zaproponował [214])

rozwiązując równanie (2.14) dla 4 kwadrantów (k, l) zachowując najmniejszy wynik. W ostatnim (5) kroku k, l jest ponownie wstawiany z nową wartością T na stos.

Uzupełnianie pojedynczego punktu

By uzupełnić nowy punkt (k, l) jako funkcję znanych już punktów wokół niego, iteruje się po tych znajdujących się w sąsiedztwie B_ϵ (gdzie ϵ to odległość w pikselach) bieżącego punktu (i, j) i oblicza $I(i, j)$, wykorzystując równanie (2.12). Gradient obrazu ∇I szacuje się jako różnice centralne. Znak funkcji ważącej $w(p, q)$ odgrywa kluczową rolę w propagacji ostrych oraz gładkich detali obrazu do uzupełnianego obszaru i zostało zaprojektowane by składała się z 3 składników $w(p, q) = dir(p, q) \cdot dst(p, q) \cdot lev(p, q)$ (2.15). Ma to dzielić na składowe model heurystyczny.

$$
\begin{aligned}
dir(p, q) &= \frac{p - q}{\|p - q\|} \cdot N(p) \\
dst(p, q) &= \frac{d_0^2}{\|p - q\|^2} \\
lev(p, q) &= \frac{T_0}{1 + |T(p) - T(q)|}
\end{aligned}
\tag{2.15}
$$

gdzie komponent $dir(p, q)$ kierunkowy ma zapewniać większy wkład pikseli bliskich normalnemu kierunkowi $N = \nabla T$ propagacji informacji w FMM niż tych dalej od normalnej. Składowa geometrycznego dystansu $dst(p, q)$ zmniejsza udział pikseli znajdujących się dalej od (p). Składowa odpowiadająca za dystans od poziomu $lev(p, q)$ dba, by te piksele będące blisko przechodzącego przez p konturu miały większy wkład, od tych znajdujących się dalej. dir oraz dst są składowymi względnymi w odniesieniu do odległości d_0 i T_0 (które ze względów praktycznych są ustawiane jako dystans między pikselami np. 1).

FMM liczone jest najpierw na zewnątrz początkowej granicy uzupełniania $\partial\Omega$ by obliczyć dystanse T_{out}. W tym celu są wykorzystywane punkty, które znajdują się nie dalej niż o ϵ od granicy $\partial\Omega$ do momentu gdy $T > \epsilon$. Następnie wykonuje się obliczenia wewnątrz obszaru i otrzymuje T_{in}. To

Algorytm 2: Algorytm FMM wykorzystany do uzupełniania przez Teleę

```
while zbiór cienkiej granicy nie jest pusty do
    pobierz P(i,j) = head(cienka granica)
    f(i,j) = ZNANE;
    for (k,l) in (i1,j),(i,j1),(i+1,j),(i,j+1) do
        if (f(k,l)! = ZNANE) then
            if (f(k,l) == WEWNĄTRZ) then
                (f(k,l) = granica);
                inpaint(k,l);
            T(k,l) = min(solve(k1,l,k,l1),
                         solve(k+1,l,k,l1),
                         solve(k1,l,k,l+1),
                         solve(k+1,l,k,l+1));
            wstaw(k,l) do cienkiej granicy;

float solve(int i1, int j1, int i2, int j2)
    float sol = 1.0e6;
    if f(i1,j1) == ZNANE then
        if f(i2,j2) == ZNANE then
            float r = sqrt(2(T(i1,j1)T(i2,j2)) * (T(i1,j1)T(i2,j2)));
            float s = (T(i1,j1) + T(i2,j2)r)/2;
            if (s >= T(i1,j1) && s >= T(i2,j2)) then
                sol = s;
            else
                s += r;
                if (s >= T(i1,j1) && s >= T(i2,j2)) then
                    sol = s;
        else
            sol = 1 + T(i1,j1);
    else if (f(i2,j2) == ZNANE) then
        sol = 1 + T(i1,j2);
    return sol;
```

powoduje uzyskanie pola T (2.16):

$$T(p) = \begin{cases} T_{in}(p) & \text{jeśli } p \in \Omega \\ -T_{out}(p) & \text{jeśli } p \notin \Omega \end{cases} \tag{2.16}$$

Całe pole T wygładzane jest filtrem 3×3 by ostatecznie obliczyć ∇T różnicami centralnymi. ϵ odpowiada on niejako za „grubość", wielkość obszaru B do uzupełnienia. Zwykle jest mniejszy od 15 i mieści się w zakresie od 3 do 10 pikseli. Większe wartości rozmywają obszar podlegający uzupełnianiu, ale są przydatne jeśli te obszary są duże. Dla obrazów kolorowych, opisana wyżej metoda stosowana jest oddzielnie dla każdego kanału.

Szczegóły matematyczne opisano w [186, 214, 215, 123].

2.6.2 Algorytm Naviera-Stokesa

To podejście [105] również stara się naśladować profesjonalnych restauratorów obrazów. W celu odwzorowania tego procesu wykorzystywane są równania Naviera-Stokesa i osiągnięcia dynamiki płynów. Algorytm propaguje Laplasjan w kierunku linii o tym samym poziomie–izofot.

W tym celu dokonuje się projekcji gradientu gładkości intensywności obrazu w kierunku izofot (2.17):

$$I_t = \nabla^{\perp} I \cdot \nabla \Delta I \tag{2.17}$$

gdzie I oznacza intensywność obrazu, $\nabla^{\perp}$ gradient prostopadły $(-\partial_y, \partial_x)$ a Δ Laplacjan $\partial_x^2 + \partial_y^2$. Jeśli dodamy anizotropową dyfuzję, to otrzymamy (2.18):

$$I_t = \nabla^{\perp} I \cdot \nabla \Delta I + \nu \nabla \cdot (g(\mid \nabla I \mid) \nabla I) \tag{2.18}$$

Celem jest taka ewolucja równania (2.17) albo (2.18), by otrzymać stabilne rozwiązanie spełniające warunki uzupełniania regionu którym jest to, by linie izofot w kierunku $\nabla^{\perp} I$ były równoległe do konturów poziomów gładkości ΔI intensywności obrazu, które dla $\nu = 0$ stają się (2.19):

$$\nabla^{\perp} I \cdot \nabla \Delta I = 0 \tag{2.19}$$

Można to rozumieć jako przemieszczanie intensywności obrazu wzdłuż linii izofot. Można również spojrzeć na ten problem z punktu widzenia dynamiki, w którym równanie (2.17) jest równaniem transportu intensywności obrazu I wzdłuż krzywych gładkości ΔI. Jest to równoważne $DI/Dt =$

0, gdzie D/Dt to matematyczna pochodna $\partial/\partial t + v \cdot \nabla$ dla wektora prędkości pola $v = \nabla^{\perp}\Delta I$. W szczególności I jest transportowane przez pole prędkości v, które jest kierunkiem poziomów krzywych gładkości ΔI.

Zamiast rozwiązywać równanie transportu (2.18) dla I rozwiązuje się równanie transportu (2.23) wirowości względem $w = \Delta I$, co prowadzi do (2.20):

$$\partial w/\partial t + v \cdot \nabla w = \nu \nabla \cdot (g(\mid \nabla w \mid)\nabla w) \tag{2.20}$$

gdzie funkcja g pozwala na anizotropowe rozchodzenie się gładkości w. Intensywność I, która definiuje pole prędkości $v = \nabla^{\perp} I$ odtwarza się poprzez równoczesne rozwiązanie problemu Poissona (2.21):

$$\Delta I = w,\ I \mid_{\partial\Omega} = I_0 \tag{2.21}$$

dla $g = 1$ klasycznym podejściem jest bezpośrednie numeryczne rozwiązanie (2.20–2.21) równań dynamiki płynów i ewolucji dynamiki w kierunku stabilnego stanu. Dla zagadnień płynów o małej lepkości ν powyższa dynamika może wymagać dłuższego czasu by zbiec się do stabilnego stanu. W związku z tym można skorzystać z pseudo stabilnych metod, które zastępują równanie Poissona (2.21) dynamiczną relaksacją (2.22):

$$I_t - \alpha[\Delta I + w\ = 0,\ \alpha > 0,\ I \mid_{\partial\Omega} = I_0 \tag{2.22}$$

gdzie parametr α decyduje o stopniu relaksacji. Dyfuzja może powodować rozmycie ostrych połączeń, gradientów I w uzupełnianym obszarze, a to sprawia, że warto dodać taką anizotropową dyfuzję do rozwiązania I. Można ją zarówno dodać do (2.22) albo jako kolejny krok połączony z krokiem Poissona (2.21).

Osiągnięcie stabilnego stanu oznacza rozwiązanie (2.19) dla intensywności I.

Problem ciągłości izofot na granicy uzupełnianego regionu rozwiązuje analogia równań Naviera-Stokesa. By rozwiązać (2.20) potrzebne są warunki brzegowe dla funkcji przepływu oraz wirowości ω. W przypadku płynów ω nie jest zwykle znana poza granicą, ale w przypadku obrazów posiadamy już tą informację ΔI i możemy stworzyć dokładny warunek brzegowy dla $\omega = \Delta I$.

W celu rozwiązania (2.20) wykorzystywana jest jednokrokowa metoda Eulera, prostokątów z centralnymi różnicami dla dyfuzji (która jest tu

anizotropowa) oraz min-mod [194] dla członu dotyczącego konwekcji. Następnie po jednym kroku rozwiązywane jest równanie Poissona (2.21) przy wykorzystaniu iteracyjnej metody Jacobiego. Dla tak zaktualizowanych intensywności obrazu I liczy się ponownie gładkość w i zaczyna od nowa. Co kilka kroków wykonywana jest anizotropowa dyfuzja by zaostrzyć krawędzie I. Stan stabilny uzyskiwany jest po N iteracjach (zwykle 300), bądź gdy obraz nie zmienia się znacznie.

Transport wirowości nieściśliwych płynów

Nieściśliwe Newtonowskie płyny są rządzone poprzez równania Naviera-Stokesa, wiążących wektor prędkości polowej $overrightarrow v$ ze skalarnym ciśnieniem p (2.20):

$$v_t + v \cdot \nabla v = -p\nabla + \nu\Delta v,\ \nabla \cdot v = 0 \tag{2.23}$$

W przypadku dwu–wymiarowym, pozbawione dywergencji pole prędkości v posiada funkcję strumienia Ψ (taką że $\nabla^{\perp}\Psi = v$). W $2D$ wirowość $\omega = nabla \times v$, spełnia podobne równanie adwekcji, które można otrzymać stosując rotację do pierwszego członu równania (2.23) i wykorzystując podstawowe właściwości geometri $2D$. Otrzymujemy więc (2.24):

$$\omega_t + v \cdot \nabla\omega = \nu\Delta\omega \tag{2.24}$$

W dwu–wymiarowym przypadku wirowość jest skalarem, który odnosi się do funkcji strumienia poprzez Laplasjan $\Delta\Psi = \omega$. W przypadku braku lepkości ($\nu = 0$) otrzymuje się równanie Eulera dla nieściśliwego przepływu. W związku z tym równanie (2.24) pociąga za sobą, że w stanie stabilnym nielepki przepływ musi spełniać (2.25):

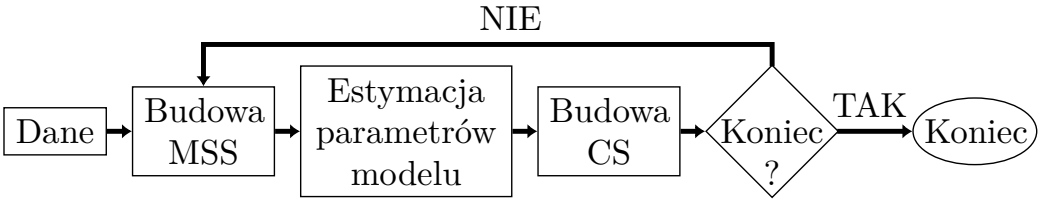
$$\omega_t + v \cdot \nabla\omega = \nu\Delta\omega \tag{2.25}$$

Oznacza to, że Laplasjan funkcji strumienia (a więc i wirowość) muszą mieć te same poziomy krzywych jak funkcja strumienia. Funkcja strumienia spełnia te same równania co równania intensywności obrazu w stanie ustalonym (2.19).

2.7 RANSAC

W wielu rzeczywistych przypadkach zebrane dane pomiarowe nie są idealne i można je podzielić na część, których rozkład da się w jakiś sposób wyjaśnić (*inliery*), np. za pomocą jakiegoś modelu z parametrami, oraz te znacznie odstające (*outliery*).

Procedura RANdom SAmple Consensus [16, 129] ma na celu ustalenie takiego spójnego zbioru danych, który (według przyjętego modelu) nie zawiera znacznie odbiegających od reszty danych. To podejście jest w stanie oczyścić zbiór danych z odstających danych, outlierów, nawet, gdy jest ich w nim więcej niż 50%–czyli praktycznego progu granicznego po przekroczeniu którego większość technik zawodzi. Mimo wielu wersji i modyfikacji, podejście RANSAC (rys. 2.16), zawsze składa się z dwóch etapów: postawienia hipotezy oraz jej sprawdzenia.



Rysunek 2.16: Schemat działania procedury RANSAC

W tym celu tworzy się minimalny model próbny MSS (ang. *Minimal Sample Set*) wykorzystując do tylko i wyłącznie minimalną liczbę danych (wystarczającą do jego utworzenia) wylosowanych ze zbioru danych $\mathcal{D}$.

Następnie sprawdza się które elementy zestawu danych są spójne z tak utworzonym modelem. Jeśli na jego podstawie odpowiednia część Γ_o, % danych pozwala osiągnąć odpowiednio dobre wyniki (te dane stanowią zestaw zgodny $\mathcal{CS}$ ang. *Consensus Set*). Oznacza to że operująca na pojedynczej danej (która może składać się z kilku różnych pomiarów) przyjęta funkcja (uwzględniająca parametry modelu) nie przyjmuje większych wartości od pewnego, obranego progu d_E. Próg definiuje maksymalne akceptowalne odchylenie spowodowane zakłóceniami.

Poszukiwania można skończyć jeśli znaleziony zostanie odpowiednio dobry zestaw zgodny albo gdy prawdopodobieństwo znalezienia lepszego (np. o większej liczbie elementów) jest małe, spadnie poniżej ustalonego progu.

Przykładowo, jeśli zadaniem byłoby dopasowanie łuku koła do zbioru punktów $2D$, to wówczas minimalna liczba punktów do stworzenia modelu wynosiłaby 3 (bo tyle potrzeba do policzenia pozycji środka okręgu oraz jego promienia). Następnie liczono by liczbę punktów, które mieszczą się w promieniu tego koła.

W przypadku, gdy dany model potwierdza zbyt mała liczba danych, wówczas się go odrzuca i tworzy nowy na podstawie nowo wylosowanych punktów. Jeśli przez zbyt wiele prób nie uda się znaleźć takiego modelu, to wówczas można zakończyć zadanie z niepowodzeniem, wykorzystać najlepszy znaleziony model albo spróbować obniżyć wymagania.

Jeśli model próbny jest zadowalający, wówczas wykorzystuje się wszystkie potwierdzające go dane, do stworzenia nowego, ostatecznego już modelu.

O ile stosunek *inlierów* do *outlierów* Γ_o oraz d_E zależą od specyfiki zadania i można je ustalić na podstawie wiedzy eksperckiej czy przeprowadzonych badań, to liczbę losowań modelu można ustalić na podstawie pożądanego prawdopodobieństwa p sukcesu. Za taki uznamy, że procedura RANSAC chociaż raz wybrała tylko *inliery* do stworzenia próbnego modelu. Przez $w = liczba\ inlierw/liczba\ elementw$ oznaczymy prawdopodobieństwo wyboru *inliera* podczas losowań danych do próbnego modelu. Na ogół ten stosunek nie jest pierwotnie znany, ale można przyjąć pewną przybliżoną wartość.

Zakładając, że niezależnie losujemy n punktów, to w^n oznacza prawdopodobieństwo, że wszystkie n punktów jest *inlierami*, a $1-w^n$, że co najmniej jeden jest *outlierem*, powodując wyznaczenie złego próbnego modelu z tego zestawu punktów. Jeśli stworzymy k takich modeli (czyli wykonaliśmy k iteracji algorytmu RANSAC, k losowań minimalnych zestawów parametrów) to prawdopodobieństwo, że algorytm nigdy nie wylosuje n punktów, które wszystkie byłyby *inlierami*, musi być równe $1-p$ (2.26):

$$1 - p = (1 - w^n)^k \tag{2.26}$$

To prawdopodobieństw dąży do 0, gdy $k \to \infty$–czyli za którymś razem zostaną wylosowane odpowiednie próbki danych. Po obustronnym zlogarytmowaniu równania (2.26) otrzymujemy k jako (2.27):

$$k = \frac{log(1-p)}{log(1-w^n)} \tag{2.27}$$

Równanie (2.27) zapewnia, że n punktów jest losowana niezależnie i ten sam punkt może zostać wybrany ponownie, co zwykle nie jest uzasadnione i wartość parametru k powinna być wzięta jako górna granica, w przypadku gdy punkty są wybierane bez powtórzeń.

Rozdział 3

Koncepcja prostego systemu odometrii wizyjnej

3.1 Wykorzystywany typ sensora i biblioteki

W rozprawie wykorzystywane były dane z sensorów takich jak Kinect v1 (w tym podobne do niego np. firmy Asus Xion). W późniejszej fazie skupiono się na wykorzystaniu danych z Kinecta v2 w eksperymentach związanych ze sztuczną inteligencją.

Do przeprowadzenia badań skorzystano głównie, poza standardowymi bibliotekami, z następującego oprogramowania: OpenCV, Eigen, PyTorch, Fastai. Do instalacji Pytorcha z Fastai wykorzystano Anaconde [17]. Do interaktywnego douczania sieci wykorzystano jupyter notebook [18]. Bibliotekę OpenGV wykorzystano tylko do badania metod n-punktowych.

3.2 Dostępne algorytmy detekcji i deskrypcji cech

3.2.1 SURF

To podejście wykonuje te same założenia co znany SIFT, ale w nico inny sposób [97].

By przyspieszyć cały proces, zaczyna się od transformacji oryginalnego obrazu do postaci całkowej. Do budowy przestrzeni skali nie trzeba próbkować oryginalnego obrazu. Zamiast tego, przybliża się wyznacznik Hes-

sianu, poprzez splatanie obrazu całkowego z przybliżonymi jądrami przekształcenia Gaussa o różnych rozmiarach. Ostatnim krokiem do wykrycia punktów charakterystycznych jest tłumienie nie maksymalnych wartości w otoczeniu $3 \times 3 \times 3$. Następnie maksyma macierzy wyznaczników Hessianu są interpolowane w przestrzeni skali i obrazu.

Dalej, oblicza się falkowe odpowiedzi Haara w kierunkach x oraz y w próbkowym, kołowym sąsiedztwie o promieniu odpowiadającym skali, w której dany punkt kluczowy został wykryty. Następnie reprezentuje się te wartości jako punkt w przestrzeni wektorowej i dodawane wewnątrz przesuwanego fragmentu koła. Najdłuższy wektor oznacza orientację.

Następnie buduje się kwadratowy region wyśrodkowany na punkcie kluczowym. Dzieli się go na mniejsze 4×4 obszary, w których liczy się falkowe odpowiedzi Haara w kierunkach x i y dla regularnie, 5×5, oddalonych punktów, interpolowane odpowiednio do orientacji i ważone za pomocą jądra przekształcenia Gaussa. Potem w każdym podregionie falkowe odpowiedzi Haara są dodawane (3.1):

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|) \tag{3.1}$$

Ostateczny deskryptor składa się z wektorów pochodzących z wszystkich podregionów.

3.2.2 ORB

Głównym pomysłem stojącym za częścią dotyczącą detektora tego algorytmu [205] jest efektywne wykorzystanie detektora FAST [204] razem z miarą „narożnikowości“ Harrisa, by zachować N najlepszych punktów kluczowych (najpierw ustawia się próg detektora tak, by znaleźć co najmniej N punktów) i piramidę skali obrazu. By osiągnąć odporność na zmianę skali–widzenie obrazu z różnych odległości, budowana jest piramida skali–tworzone są coraz mniejsze obrazy mające reprezentować widzenie tego samego obrazu z różnych odległości. Otrzymuje się w ten sposób cechy FAST. Odporność na zmianę rotacji otrzymuje się poprzez wykorzystanie jako miary orientacji centroidu intensywności.

Deskryptor tworzony jest za pomocą serii testów binarnych wykonywanych w sposób określony za pomocą wzorca uzyskanego uczeniem maszynowym, obróconego zgodnie z dominującą orientacja opisywanego punktu kluczowego. Test dwóch sparowanych punktów wygląda następująco

(3.2):

$$\tau(p; x, y) := \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases} \tag{3.2}$$

gdzie $p(x)$ jest intensywnością p w punkcie x.

3.2.3 BRISK

Podobnie do innych algorytmów, BRISK [169] tworzy przestrzeń skali, poprzez zmniejszanie liczby próbek oryginalnego obrazu. Na każdym poziomie tej przestrzeni skali, szuka się punktów kluczowych za pomocą detektora FAST [204]. W związku z tym 9 z 16 pikseli wokół kandydującego na punkt kluczowy punktu musi być od niego wystarczająco jaśniejszych bądź ciemniejszych, by dalej go rozpatrywać.

Następnie tłumi się nie maksymalne wartości. Po pierwsze kandydujący punkt musi mieć lepszy wynik FAST niż sąsiednie 8 pikseli. Po drugie musi mieć lepsze wyniki niż piksele w łatach oraz skalach, nad i pod nim, ze względu na dyskretyzację te wycinki są interpolowane. Następnie dla każdego wykrytego maksimum wykonuje się podpikselowe udoskonalanie, by uzyskać końcowy punkt kluczowy. Ostatecznie ponownie interpoluje się współrzędne punktu kluczowego.

Dysponując poprawionym położeniem punktu kluczowego na obrazie oraz jego skalą, określaną za pomocą liczb zmiennoprzecinkowych, deskryptor BRISK wygładza najpierw obraz jądrem przekształcenia Gaussa i skaluje odpowiednio wzorzec, by następnie określić główną rotację, którą określa się na podstawie binarnych testów pomiędzy odległymi parami punktów. Ostatecznie obraca się ten wzorzec i wykonuje serię testów jasności, ale tym razem bazując na bliskich parach punktów.

3.2.4 KAZE

Zaproponowane w [90] podejście (KAZE) jest podobne do algorytmu SIFT [172, 170]. Do zbudowania przestrzeni skali wykorzystuje się schemat dodatniego operatora dzielenia AOS (ang. *additive operator splitting*) oraz rozmywanie o zmiennym przewodzeniu, by zredukować szumy, jednocześnie zachowując ostrość. Nie zmienia się rozdzielczości obrazów. By zna-

leźć punkty kluczowe poszukuje się maksimów, w przestrzeni i na różnych skalach, znormalizowanego względem danej skali wyznacznika Hessianu funkcji obrazowej.

By uzyskać odporny na obroty deskryptor orientację główną oblicza się podobnie jak w SURF–waży się pierwsze pochodne L_x oraz L_y za pomocą jądra przekształcenia Gaussa wycentrowanego na opisywanym punkcie. Następnie te wartości reprezentuje się jako punkt w przestrzeni wektorowej i sumuje z przesuwanym fragmentem koła. Za orientację przyjmuje się najdłuższy powstały w ten sposób wektor.

Deskryptor jest inspirowany SURFem, dla znalezionej cechy kluczowej liczy się pochodne pierwszego rzędu w, podzielonym na 4×4 nakładające się podobszary, prostokątnym wycinku. Każda próbkę, jak powiązane z nią pochodne, będącą w wycinku obraca się zgodnie z orientacją główną. Po ważeniu dodaje się deskryptory każdego z regionów do podwektora deskryptora, które są dalej ważone i dodawane, by stworzyć ostateczny deskryptor.

3.2.5 AKAZE

To podejście [195] jest dalszym rozwinięciem podejścia KAZE, kontynuowanym przez tych samych autorów, [90] które to z kolei jest podobne do dobrze znanego SIFT [172, 170, 210], (jego patent wygasł w 2020 r.), ale o wiele szybszym do obliczenia.

Pozwala na uzyskanie wyników podobnych do popularnego detektora/deskryptora SURF, jednak wykorzystuje binarny deskryptor, którego dopasowywanie jest mniej kosztowne obliczeniowo, a jego licencja pozwala na darmowe korzystanie, nawet w komercyjnych zastosowaniach.

Detektor AKAZE działa podobnie do SIFT, z tą różnicą, że korzysta z algorytmu *fast explicit diffusion*, a nie jądra przekształcenia Gaussa, w celu utworzenia wieloskalowej piramidy obrazów. By znaleźć punkty kluczowe w różnych miejscach i na różnych poziomach skali, szukane są maksima znormalizowanego względem nieliniowej skali, przybliżonego wyznacznika Hessianu funkcji obrazowej, które porównywane są do przyjętej wartości progowej τ_A. Wyznaczniki Hessianu liczone są dla każdego przefiltrowanego filtrem Sharra obrazu, należącego do nieliniowej przestrzeni skali (3.3):

$$L^i_{\text{Hessian}} = \sigma^2_{i,\text{norm}} / 2^{\sigma^i} (L^i_{xx} L^i_{yy} - L^i_{xy} L^i_{xy}) \tag{3.3}$$

gdzie L oznacza iluminację obrazu, i jest indeksem filtrowanego obrazu, o indeksem oktawy. Współczynnik normalizujący $\sigma_{i,\text{norm}}$ liczony jest jako $\sigma_{i,\text{norm}} = \sigma_i/2^{o^i}$, przy czym $\sigma_i(o,s) = 2^{o+s/S}$, gdzie s jest indeksem podpoziomu, a S liczbą podpoziomów. Parametrem kontrolującym działanie detektora jest wartość progowa τ_A. Jeśli przybliżony wyznacznik Hessianu jest większy od τ_A, to następnie sprawdza się, czy jest to również lokalne maksimum w oknie 3×3 pikseli. Dopiero wówczas taki punkt rozważany jest w kategorii kandydata na punkt kluczowy i porównuje się go z innymi kandydatami z poziomów bezpośrednio nad i pod nim w oknie $\sigma \times \sigma$.

Autorski deskryptor AKAZE (zwany przez autorów M–LDB) działa bardzo podobnie do deskryptora BRIEF, z tą różnicą,że porównuje się średnią intensywność regionów (zamiast poszczególnych wartości pikseli) oraz wartości średnie pionowych i poziomych pochodnych funkcji obrazowej. Obliczenia deskryptora zaczynają się od oszacowania orientacji metodą histogramową, tak jak w KAZE, po czym odpowiednio obraca się wzorzec. Ostatecznie generowany jest wektor deskryptora poprzez wykonanie binarnych testów średnich obszarów oraz wartości średnich pionowych i poziomych pochodnych w danym obszarze.

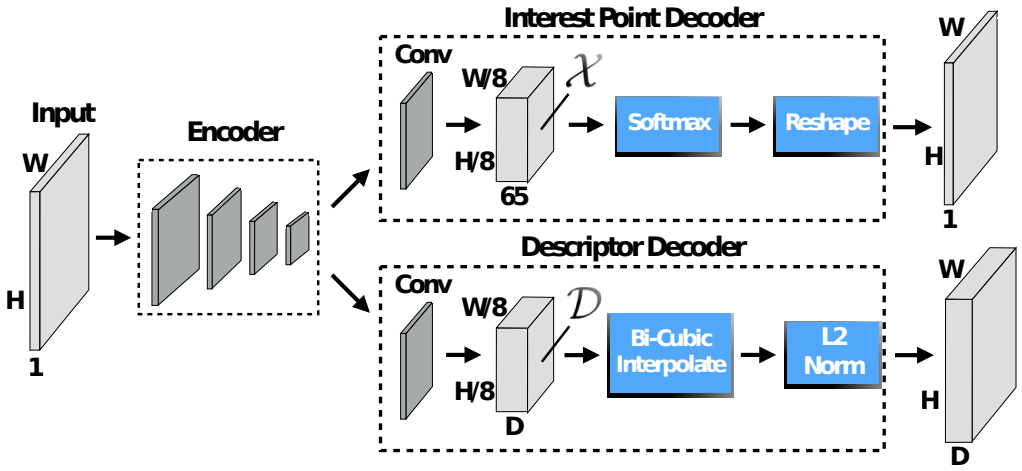
3.2.6 SuperPoint

Jest to w pełni konwolucyjna sieć neuronowa działająca jako detektor i deskryptor o stałej długości. Model [117] dzieli wspólny enkoder, który zmniejsza rozmiar wejściowego obrazu. Następnie architektura (rys. 3.1) rozdziela się na dwa dekodery. Jeden służy do znajdowania punktów kluczowych, a drugi do ich opisu

Wartość każdego wyjściowego piksela odpowiada prawdopodobieństwu, że jest to cecha charakterystyczna. Odpowiada jej gęsta mapa, znormalizowanych według normy L2 (odległość Euklidesowa) deskryptorów, których większość jest interpolowana, ponieważ sieć dostarcza pół-gęstą mapę.

3.3 Metody dopasowywania par deskryptorów

By powiązać z sobą parę deskryptorów ($\mathbf{d}_i$, $\mathbf{d}_j$) można obliczyć dystans pomiędzy nimi w przestrzeni deskryptorów i dopasować je na podstawie



Rysunek 3.1: Architektura SuperPoint

odległości, obliczaną różnymi miarami, odpowiednio dobranymi do typu danego deskryptora.

Generalnie rozróżnia się dwa typy deskryptorów: te bazujące na lokalnych gradientach oraz na bazujące na intensywności obrazu. Do tych pierwszych (takich jak SIFT czy SURF) stosuje się miary dystansu takie jak L2 (odległość Euklidesowa), czy L1 (odległość taksówkowa, manhattan). Do drugiego rodzaju korzysta się z dystansu Hamminga.

Pierwsza z metod siłowo (ang. *BruteForce*) wyznacza dystans dla każdego deskryptora z zestawu $\mathbb{A}$ pomiędzy nim, a każdym deskryptorem z zestawu $\mathbb{B}$. Następnie za dopasowany można uznać ten najbliższy punkt. Opcjonalnie zachowując informacje o k (ta liczba jest dobierana arbitralnie) kolejnych najbliższych deskryptorach.

Kolejnym sposobem jest skorzystanie z odpowiednich metod do szybszego poszukiwania najlepszego, najbardziej podobnego deskryptora do aktualnie rozważanego. Celowi temu może posłużyć biblioteka szybkiego poszukiwania najbliższych sąsiadów FLANN (amg. *Fast Library for Approximate Nearest Neighbors*) [182, 19], korzystająca z drzew *kd*, w której został zawarty zbiór algorytmów zoptymalizowanych do szybkiego poszukiwania najbliższych, w przybliżeniu, sąsiadów w dużych zbiorach danych i wielowymiarowych cech.

3.4 Metody filtracji błędnych dopasowań

Analiza obrazu z kamer jest wrażliwa zarówno na czynniki zewnętrzne (zmiany oświetlenia, cienie, tekstura powierzchni) jak i wewnętrzne (głównie koszty obliczeniowe, zasilanie). Wprowadza to do obrazu zakłócenia i jest przyczyną niepewności co do przedmiotu pomiaru, która przejawia się występowaniem w obrazie cech, które nie są odwzorowaniem nieciągłości geometrycznych, fizycznych czy fotometrycznych charakterystyk obserwowanych obiektów [217]. Ta niepewność jest jednym ze źródeł trudności w wykrywaniu punktów kluczowych/cech (ang. *feature*), a dalej ich dopasowaniu–opis (deskryptor) tej samej cechy bazuje na innych obrazach, więc się różni. Kolejnym źródłem problemów jest sama wielowymiarowość deskryptora.

Powstały więc metody radzenia sobie z tym problemem–filtracji błędnych dopasowań.

Test stosunku Lowego: Lowe'y [171] zaproponował, by zastosować próg będący stosunkiem odległości pomiędzy keypointem, a najbliższym odpowiadającym mu punktem na drugim obrazie, do odległości do drugiej najbliższej cechy na drugim obrazie, która powinna być o wiele dalej. Jest to podejście typu BruteForce.

Dopasowywanie w obydwu kierunkach: Zachowuje się tylko te pary deskryptorów (i, j), które nie tylko do i-tego deskryptora z zestawu $\mathbb{A}$ posiadają przypisany j-ty z zestawu $\mathbb{B}$ jako najlepszy, ale dzieje się tak również i na odwrót. Nie zawsze tak musi być, może się okazać że w zestawie $\mathbb{A}$ jest większy, wówczas może istnieć lepiej dopasowany punkt.

Zastosowanie procedury RANSAC: Spośród dopasowanych par deskryptorów wybiera się najmniejszą liczbę par punktów niezbędną do wykonania zadania poprzez wyznaczenie pewnych parametrów jakiejś transformacji bądź modelu. Następnie sprawdza się jaki stosunek pozostałych punktów również mieści się w tak powstałym modelu albo jaki jest stosunek punktów akceptowalnie dobrze spełnia daną transformację do tych przekraczających dany próg. Geneza i procedury zostały dokładnie opisane w podrozdziale RANSAC 2.7.

3.5 Metody estymacji przebytej trajektorii

W zależności od wykorzystywanego czujnika korzysta się z różnych podejść do oszacowania translacji i rotacji występującej pomiędzy dwoma ujęciami. W przypadku wykorzystania do tego celu danych wizyjnych, można wyróżnić metody wykorzystujące tylko i wyłącznie dane RGB oraz te mające dostęp także do pomiarów odległości dla każdego piksela–map głębi.

3.5.1 Wykorzystujące tylko dane RGB

Obecnie istnieje wiele rozwiązań i bibliotek, które można wykorzystać do odtworzenia występującej pomiędzy dwoma obrazami RGB rotacji: OpenCV [107], OpenMVG [181] oraz OpenGV [160]. W przypadku skalibrowanych kamer uzyskamy macierz zasadniczą posiadającą 5 stopni swobody (zawiera w sobie rotację i translację bez skali), a dla nieskalibrowanych fundamentalną o 7 stopniach (zapisane są w niej dane kalibracyjne).

Wszystkie z bibliotek, poza OpenCV, zawierają 5-punktowy algorytm Nistera [189], Stewéniusa [223], a także podejście 7-punktowe czy 8-punktowe pozwalające znaleźć macierz zasadniczą na podstawie powiązań $2D-2D$. W OpenCV dwie ostatnie metody zwracają zamiast macierzy zasadniczej macierz fundamentalną. Podczas gdy OpenMVG zawiera wiele algorytmów związanych z wizją komputerową (w tym algorytmy 5-punktowe Nistera i Stewéniusa), wyznaczania struktury uzyskanej z ruchu SfM (ang. *Structure from Motion*) czy odtwarzania roto-translacji pomiędzy dwoma ujęciami (czyli tego co nas najbardziej interesuje), to OpenGV jest biblioteką zawierającą najwięcej rozwiązań potrzebnych do przeprowadzanych badań.

Biblioteka OpenGV [160] zawiera algorytmy do: obliczania pozycji kamery na podstawie powiązań punktów $2D-3D$ pomiędzy układem odniesienia świata, a wektorami wodzącymi układu kamery (to jest znane jako problem centralnej pozycji absolutnej), wieloma układami odniesienia kamery (niecentralnej pozycji absolutnej), znajdowania pozycji kamery w odniesieniu do innej kamery na podstawie $2D-2D$ powiązań wektorów wodzących układu kamer (centralnej pozycji względnej) oraz na podstawie wielu ujęć kamer (niecentralnej pozycji względnej). W wyniku, w zależności od wykorzystanego algorytmu, otrzymujemy tylko rotację, tylko

translację, kilka prawdopodobnych rotacji, rotację i translację, kilka możliwych rotacji i translacji, macierz zasadniczą, kilka możliwych macierzy zasadniczych oraz kilka możliwych zespolonych macierzy zasadniczych w przypadku 5-punktowego podejścia Stewéniusa.

Algorytm 8-punktowy

Jest to najstarszy algorytm służący do obliczania macierzy fundamentalnej **F**, bądź zasadniczej **E** (jeśli przeprowadzono kalibrację). Wykorzystuje on osiem bądź więcej dopasowanych punktów. W zależności czy wykonano kalibrację, macierze **E** i **F** można zdefiniować jako (3.4):

$$\mathbf{u}'\mathbf{E}\mathbf{u} = 0 \quad a) \qquad lub \qquad \mathbf{u}'\mathbf{F}\mathbf{u} \quad b) \tag{3.4}$$

gdzie $\mathbf{u}'$ oraz $\mathbf{u}'$ to dopasowane, jednorodne punkty obrazu (z tego względu ostatni element każdego punktu wynosi 1). Wszystkie punkty wejściowe służące do obliczenia tych macierzy są normalizowane, by zapewnić (w przybliżeniu) ich równe traktowanie. Równania macierzowe (3.4) można przedstawić w formie wektorowej otrzymując zbiór równań liniowych (3.5):

$$\mathbf{A}\mathbf{e} = 0 \quad a) \qquad lub \qquad \mathbf{A}\mathbf{f} \quad b) \tag{3.5}$$

gdzie **A** jest macierzą równania, a **e** i **f** są 9-elementowymi wektorami zawierającymi wartości macierzy **E** i **F**. Ponieważ kamera mierzy zasadniczo kąty, to ruch możemy określić tylko z dokładnością do pewnego wspólnego, niezerowego współczynnika skali–macierz A musi mieć rząd nie większy niż 8. By uniknąć rozwiązania trywialnego (tzn. zerowego) narzuca się dodatkowe ograniczenie, tj. by normy wektorów były jednostkowe: $\|e\| = 1$ i $\|f\| = 1$ (Alternatywnym rozwiązaniem jest przyjęcie $F_{33} = 1$).

Dalej wykorzystuje się właściwość macierzy fundamentalnej, czyli to że jest ona osobliwa i ma rząd wielkości 2. Ponieważ nie zawsze mamy do czynienia z idealnymi danymi, to ta własność jest wymuszana poprzez wyzerowanie najmniejszej wartości osobliwej albo zamianę jej poprzez macierz $\mathbf{F}'$ minimalizującą normę Frobeniusa $\mathbf{F} - \mathbf{F}'$ pod warunkiem że macierzy $\mathbf{F}'$ jest równy 0 ($det(\mathbf{F}') = 0$).

Istnieją przypadki, w których nie da się odtworzyć macierzy fundamentalnej. Dzieje się tak, gdy wszystkie punkty leżą na tej samej płaszczyźnie, bądź występuje tylko rotacja, nie ma translacji.

Algorytm 5-punktowy Nistera i algorytm Stewéniusa

Oba podejścia [189, 223] są dość podobne i obliczają macierz zasadniczą **E** wykorzystując 5 dopasowanych par punktów. Przyjmując, że $[t]_\times$ oznacza macierz skośno–symetryczną (3.6):

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{3.6}$$

Dzięki temu $[t]_\times = t \times x$. Wtedy macierz fundamentalną można zapisać jako (3.7):

$$\mathbf{F} \equiv \mathbf{K}_2^{-T}[t]_\times \mathbf{R}\mathbf{K}_1^{-1} \tag{3.7}$$

Macierz fundamentalna koduje w sobie ograniczenie epipolarne/koplanarne (3.8):

$$\mathbf{u}'^T\mathbf{F}\mathbf{u} = 0 \tag{3.8}$$

Kamerę uważa się za skalibrowaną, gdy zarówno $\mathbf{K}_1$ jak i $\mathbf{K}_2$ są znane. W takim przypadku zawsze można założyć, że punkty **u** i **u**′ zostały uprzednio odpowiednio przemnożone przez $\mathbf{K}_1^{-1}$, jak i $\mathbf{K}_2^{-1}$, tak że ograniczenie epipolarne upraszcza się do (3.9):

$$\mathbf{u}'^T\mathbf{E}\mathbf{u} = 0 \tag{3.9}$$

gdzie $\mathbf{E} \equiv [t]_\times \mathbf{R}$ nazywana jest macierzą zasadniczą. Każda macierz macierz o rzędzie 2 mogłaby być macierzą fundamentalną ale macierzą zasadniczą jest tylko taka, której dwie niezerowe wartości osobliwe są sobie równe, i w związku z tym otrzymuje się sześcienne ograniczenie (3.10):

$$\mathbf{E}\mathbf{E}^T\mathbf{E} - \frac{1}{2}(\mathbf{E}\mathbf{E}^T)\mathbf{E} = 0 \tag{3.10}$$

Każda para powiązanych par punktów musi spełniać równanie (3.9), a je można przedstawić za pomocą iloczynu dwóch wektorów (3.11):

$$\widetilde{u}^T\widetilde{E} = 0 \tag{3.11}$$

gdzie:

$$\widetilde{\mathbf{u}} \equiv \begin{bmatrix} u_1u'_1 & u_2u'_1 & u_3u'_1 & u_1u'_2 & u_2u'_2 & u_3u'_2 & u_1u'_3 & u_2u'_3 & u_3u'_3 \end{bmatrix}^T$$
$$\widetilde{\mathbf{E}} \equiv \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^T \tag{3.12}$$

Poprzez złożenie równań dla (3.11) w formie wektorowej to dla 5 powiązanych par punktów otrzyma się zbiór równań liniowych, który można opisać macierzą o wymiarach 5×9. Następnie wylicza się cztery wektory $\widetilde{X}$, $\widetilde{Y}$, $\widetilde{Z}$ i $\widetilde{W}$ które rozciągają prawą przestrzeń zerową tej macierzy, tworzą jądro przekształcenia (bo macierz $\widetilde{u}^T$ ma rząd równy 5 a 9 kolumn). Te 4 wektory odpowiadają wprost macierzom 3×3 i w związku z tym macierz zasadnicza musi przyjmować formę (3.13):

$$\mathbf{E} = x\mathbf{X} + y\mathbf{Y} + z\mathbf{Z} + w\mathbf{W} \tag{3.13}$$

dla pewnych wartości skalarnych x, y, z i w. Są one zdefiniowane tylko co do wspólnego współczynnika skali, w związku z tym zakłada się $w = 1$. Dalej wstawia się (3.13) do 9 sześciennych ograniczeń (3.10) i otrzymuje układ równań, który następnie redukuje się poprzez wykorzystanie eliminacji Gaussa-Jordana (w podejściu Nistera) albo za pomocą bazy Gröbnera (w wersji Stewéniusa). Jak zauważono w [162], okazuje się, że pierwsze podejście jest równoważne do obliczenia leksykograficznej bazy Gröbnera, zamiast tej porządkującej jednomiany w sposób leksykograficzny odwrotnego stopnia (albo z odwrotną gradacją, w zależności od przyjętego tłumaczenia ang. *graded-reverse lexicographic*) jak to robi Stewénius.

Ostatecznie otrzymuje się wielomian 10 stopnia, którego rozwiązanie dostarcza nam 10 możliwych rozwiązań dla zmiennej z, które to z kolei wykorzystuje się do znalezienia x, y i ostatecznie macierzy zasadniczej poprzez wykorzystanie (3.13).

Ten algorytm można rozszerzyć, by wykorzystywał więcej niż 5 punktów w taki sam sposób co nieskalibrowane 7 i 8-punktowe metody. Wtedy korzysta się z 4 nadokreślonych (ang. *overdetermined*) wektorów osobliwych X, Y, Z, W, którym odpowiadają najmniejsze wartości własne.

Algorytm 5-punktowy Kneipa

Ta bezpośrednia metoda [162] odnajduje właściwą rotację pomiędzy dwoma obrazami niezależnie od występującej translacji. Wynika to z faktu, że

pływ optyczny wynikający z obrotu istotnie różni się od tego wynikającego z translacji. Jeśli przez **R** oznaczy się rotację z 2 punktu widzenia do 1, to skompensowane rotacją obserwacje z punktu 2 przyjmują formę (3.14):

$$c_{\mathbf{R}}(\mathbf{f}_i') = \mathbf{R}\mathbf{f}_i' \tag{3.14}$$

gdzie **R** oznacza macierz rotacji, a f_n i f_n' n-te jednostkowe wektory cech opisujące położenie cechy z pierwotnego i kolejnego punktu widzenia. To oznacza, że musimy mieć czysto translacyjny pływ optyczny, więc wszystkie wektory $\mathbf{f}_i \times \mathbf{R}\mathbf{f}_i'$ muszą leżeć na jednej płaszczyźnie, być koplanarne. Najmniejsza liczba wektorów opisująca koplanarność to trzy, co najprościej ująć poprzez przyrównanie wyznacznika tych trzech wektorów do zera. W związku z tym na rotację można nałożyć, niezależne względem przesunięcia, ograniczenie (3.15):

$$|(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}_1')(\mathbf{f}_2 \times \mathbf{R}\mathbf{f}_2')(\mathbf{f}_3 \times \mathbf{R}\mathbf{f}_3')| = 0 \tag{3.15}$$

Ponieważ rotacja posiada trzy stopnie swobody, to potrzeba co najmniej trzech ograniczeń koplanarnych epipolarnej płaszczyzny prostopadłej do pełnego ograniczenia rotacji. Wykorzystując dwa dodatkowe punkty f_4 i f_5 można zbudować następujący układ równań do obliczenia rotacji (3.16):

$$\begin{cases} |(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}_1')(\mathbf{f}_2 \times \mathbf{R}\mathbf{f}_2')(\mathbf{f}_3 \times \mathbf{R}\mathbf{f}_3')| = 0 \\ |(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}_1')(\mathbf{f}_2 \times \mathbf{R}\mathbf{f}_2')(\mathbf{f}_4 \times \mathbf{R}\mathbf{f}_4')| = 0 \\ |(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}_1')(\mathbf{f}_2 \times \mathbf{R}\mathbf{f}_2')(\mathbf{f}_5 \times \mathbf{R}\mathbf{f}_5')| = 0 \end{cases} \tag{3.16}$$

Do ograniczeń (3.16) można dołożyć wymogi, w zależności od wybranej parametryzacji, by macierz rotacji była faktyczną, właściwą macierzą rotacji. Wybór sposobu parametryzacji ma duży wpływ na złożoność rozwiązania.

Równanie (3.16) rozwiązywane jest przy wykorzystaniu bazy Gröbnera w wyniku czego otrzymuje się do 20 różnych macierzy rotacji, wynikających z 10 macierzy zasadniczych. Część z nich jest wewnętrznie odrzucana, jeśli:

a) ich urojone części rozkładu na wartości własne są zbyt duże, przekraczają obrany próg,

b) nie są spójne, czyli ich wyznacznik nie jest równy jeden,

c) po podstawieniu ich do pierwotnych ograniczeń powstały błąd przekracza obrany próg,

d) rozstrzygnięcie niejednoznaczności reprezentacji rotacji i translacji przez macierz zasadniczą.

Rozstrzyganie niejednoznaczności przebiega nieco inaczej niż klasyczne podejście np. Hartleya i Zissermana [138]. Sprawdza się czy wektor cechy leży również na promieniu zdefiniowanym przez wektor cechy. Poprzez $\mathbf{f}$ oznacza się wektor kierunkowy, który jest równy wektorowi translacji z dokładnością do znaku. Da się go otrzymać jako iloczyn wektorowy dwóch normalnych do epipolarnych płaszczyzn (3.17):

$$\mathbf{d} = \pm\mathbf{t} = \mathbf{n}_1 \times \mathbf{n}_2 = (\mathbf{f}_1 \times \mathbf{R}\mathbf{f}'_1) \times (\mathbf{f}_2 \times \mathbf{R}\mathbf{f}'_2) \tag{3.17}$$

Dla poprawnej rotacji można zaobserwować, że iloczyn wektorowy pomiędzy wektorem kierunkowym $\mathbf{d}$ oraz $\mathbf{f}_1$ musi wskazywać ten sam kierunek co iloczyn wektorowy wektora kierunkowego $\mathbf{d}$ i $\mathbf{R}\mathbf{f}'_1$. Innymi słowy iloczyn skalarny obydwu iloczynów wektorowych musi być dodatni. To pozwala na zapisanie (3.18) nierówności ograniczającej każdą rotację:

$$\begin{gathered}(\mathbf{d} \times \mathbf{f}_1) \cdot (\mathbf{d} \times \mathbf{R}\mathbf{f}'_1) > 0 \implies \\ \implies \{(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}'_1) \times (\mathbf{f}_2 \times \mathbf{R}\mathbf{f}'_2) \times \mathbf{f}_1\} \cdot \{(\mathbf{f}_1 \times \mathbf{R}\mathbf{f}'_1) \times (\mathbf{f}_2 \times \mathbf{R}\mathbf{f}'_2) \times \mathbf{R}\mathbf{f}'_1) > 0\}\end{gathered} \tag{3.18}$$

Dzięki temu ograniczeniu (3.18) jesteśmy w stanie ujednoznacznić macierze rotacji bez wyprowadzania translacji czy informacji przestrzennej ($3D$) punktu. Do jednoznacznego określenia macierzy można wybrać każdą parę punktów.

To podejście pozwala na obliczenie poprawnej rotacji, także w przypadku braku translacji. Zawodzi jedynie w zdegenerowanych przypadkach. Może to być w sytuacji, gdy wszystkie punkty znajdują się wzdłuż równika (ang. *equator*), jednej linii. Wtedy pływ optyczny można by wytłumaczyć zarówno jako czysto rotacyjny albo czysto translacyjny.

Mimo to obliczenia się pogarszają w przypadku małego, dążącego do 0, przesunięcia. W tedy 20 rozwiązań algorytmu 5-punktowego zbiega się do pojedynczego rozwiązania, bo taką rotację (przy braku translacji) da się ustalić za pomocą 2 punktów. Efektem ubocznym jest to, że w wyniku

podążania stałym śladem obliczeń bazy Gröbnera, otrzymuje się podobne wielomiany i redukcje skutkują otrzymaniem niemal zerowych współczynników w środku obliczeń. Sparametryzowane rozwiązanie ma w tym wypadku zbyt dużą złożoność skutkując niestabilnymi numerycznie obliczeniami. W związku z tym autorzy [162] zalecają wykorzystać algorytm 2-punktowy.

Solver wartości własnych–metoda Eigensolver

To podejście wykorzystuje fakt, że płaszczyzna epipolarna powiązanej cechy zawiera tę zaobserwowaną cechę, a więc pozycję $3D$, oraz położenia dwóch środków obiektywu z której został ten punkt zaobserwowany. W efekcie otrzymujemy rodzinę płaszczyzn, które przecinają się w linii translacji. Innymi słowy wektory normalne do tych płaszczyzn muszą być współliniowe, co wyraża równanie (3.19):

$$\mathbf{n}_i = \mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i \tag{3.19}$$

gdzie $\mathbf{R}$ jest macierzą transformującą i-ty wektor wodzący $\mathbf{f}_i$ do odpowiadającego mu wektora wodzącego widzianego w drugim ujęciu $\mathbf{f}'_i$.

Główną ideą algorytmu jest wymuszenie współpłaszczysnowości n wektorów normalnych do płaszczyzn epipolarnych poprzez potraktowanie wektorów normalnych jak chmurę punktów i anulowanie drugich momentów, dylatacji w jednym z kierunków. Otrzymuje się to poprzez zebranie wszystkich wektorów normalnych w macierz $\mathbf{N} = [\mathbf{n}_1 \dots \mathbf{n}_n]$ (o wymiarach $3 \times n$) i zminimalizowanie (3.20) najmniejszej wartości własnej $\mathbf{N}\mathbf{N}^T = \sum_{i=1}^{n} \mathbf{n}_i\mathbf{n}_i^T = \sum_{i=1}^{n}(\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i)(\mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i)^T = \mathbf{M}$:

$$\mathbf{R} = argmin_{\mathbf{R}} \lambda_{\mathbf{M},min} \tag{3.20}$$

W najlepszym przypadku trzeba przeprowadzić nieliniową optymalizację (metodą gradientu prostego) dla trzech parametrów, bo rotacja posiada 3 stopnie swobody. W idealnym świecie $\mathbf{M}$ ma rząd co najwyżej równy 2 i jest rzeczywistą, symetryczną i dodatnio określoną macierzą. Wynika to z faktu, że jeśli wektory normalne są współliniowe, to stworzona z nich macierz nie może rozpinać przestrzenni trójwymiarowej–nie może mieć rzędu 3 tylko 2. Niestety ale w rzeczywistości nie musi tak się stać z uwagi na różnego rodzaju niedoskonałości przetwarzanych danych (dyskretyzacja pomiarów, szumy i inne problemy).

By rozwiązać ten problem Kneip i inni [161] wymuszają zerowe pochodnych cząstkowych pierwszego stopnia na $\lambda_{\mathbf{M},min}$ a potem wykorzystują schemat Levenberg-Marquardta do dalszych obliczeń. Ten algorytm także oblicza kierunek przemieszczenia (ale jawnie go nie zwraca) jako automatycznie dany poprzez wektor własny odpowiadający najmniejszej wartości własnej.

Chociaż można by minimalizować wartość bezwzględną otrzymaną z $SVD(\mathbf{N}) = \mathbf{UDV}^T$ to wykorzystywany jest jej kwadrat otrzymany jako $SVD(\mathbf{NN}^T = \mathbf{UDV}^T\mathbf{VDU}^T = \mathbf{UDDU}^T = \mathbf{UD}^2\mathbf{U}^T)$. Otrzymuje się w ten sposób macież o wymiarach 3×3. Posiada ona rozwiązanie w zamkniętej formie analitycznej, a więc jego wyznaczenie jest znacznie efektywniejsze numerycznie. Macierz rotacji można rozłożyć na czynniki, a poszczególne sumy wynikające z i-tej cechy da się odpowiednio wyodrębnić na potrzeby obliczeń elementów macierzy **M**. Dzięki temu zabiegowi wszystkie człony odpowiedzialne za sumowanie wystarczy policzyć tylko raz, a więc można je stale wykorzystywać w całym procesie minimalizacji rządu macierzy (powodując optymalizację macierzy rotacji **R**) przy liniowej złożoności obliczeniowej.

Wykorzystanie większej liczby punktów

Większość z wymienionych algorytmów może działać z większą, niż minimalna, liczbą punktów. Osiąga się to metodą najmniejszych kwadratów przez znalezienie przybliżonego wektora zerowego z dekompozycji własnej. Niektórzy [223] uważają jednak, że nie jest pewne, czy dodatkowa liczba punktów jest efektywnie wykorzystywana.

3.5.2 Wykorzystujące dane RGB oraz dane głębi D

Algorytm Kabscha

To podejście [154, 153] zaimplementowano wspierając się pracą [141]. Oblicza ono transformację pomiędzy dwoma wyrównanymi/wycentrowanymi zbiorami, w sensie metody najmniejszych kwadratów, punktów. Warto podkreślić, że przedstawiony algorytm znajduje najlepsze oszacowanie rotacji i translacji z zestawu $\mathbb{A}$ do $\mathbb{B}$ a nie robota/kamery. W związku z tym przemieszczenie kamery jest odwrotne, co realizujemy poprzez zamianę danych, tak by punkty widziane w najnowszym ujęciu traktować jako te ze

starszego i wice wersa (choć można transponować macierz rotacji i zmienić znak translacji).

Algorytm składa się z trzech kroków. Najpierw, dla każdego zestawu punktów kluczowych $\mathbb{A}$ oraz $\mathbb{B}$ oblicza się centroidy i przesuwa się położenie $3D$ każdej cechy tak by oba zestawy miały środek masy w początku układu współrzędnych. Następnie rotację oblicza się za pomocą rozkładu na wartości własne SVD (ang. *Singular Value Decomposition*) macierzy kowariancji macierzy zawierających położenia punktów zestawów $\mathbb{A}$ i $\mathbb{B}$. Po odtworzeniu rotacji można obliczyć przemieszczenie za pomocą centroidów.

Za centroid rozumie się średnie położenie punkt wszystkich punktów w danym zbiorze (3.21):

$$\mathbf{p}_c = \frac{1}{N} \sum_{i=1}^{N} \mathbf{p}_i \tag{3.21}$$

gdzie $\mathbf{p}_c$ to średnie położenie punkt wszystkich punktów w danym zbiorze, $\mathbf{p}_i$ to i-ty punkt punkt w zbiorze, a N to liczba punktów występujących w tym zbiorze.

Dalej trzeba przesunąć wszystkie punkty ze zbioru $\mathbb{P}$, tak, by średnie położenie punkt wszystkich punktów pokrywało się z początkiem układu współrzędnych.

W ten sposób otrzymujemy wycentrowany zbiór $\mathbb{P}'$.

By obliczyć rotację należy:

1. Stworzyć $3 \times N$ macierze **A** oraz **B** zawierające uprzednio dopasowane i wycentrowane punkty ze zbiorów $\mathbb{A}$ i $\mathbb{B}$.
2. Obliczyć macierz kowariancji $\mathbf{C} = \mathbf{A}\mathbf{B}^T$
3. Rozłożyć macierz kowariancji **C** na wartości własne (SVD) $\mathbf{C} = \mathbf{V}\mathbf{S}\mathbf{W}^T$
4. obliczyć znak $d = sign(det(\mathbf{C}))$
5. obliczyć macierz optymalnej rotacji jako $\mathbf{R} = \mathbf{W} \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \mathbf{V}^T$

Posiadając obliczone punkty centralne obydwu zestawów $\mathbb{A}$ i $\mathbb{B}$ punktów (centroidy) oraz rotację **R** można obliczyć translację jako (3.22):

$$\mathbf{t} = -\mathbf{R}\mathbf{p}_{cA} + \mathbf{p}_{cB} \tag{3.22}$$

gdzie $\mathbf{t}$ oznacza trnalsację, $\mathbf{R}$ rotację z zestawu $\mathbb{A}$ do $\mathbb{B}$, a $\mathbf{p}_{cA}$ oraz $\mathbf{p}_{cA}$ to centroidy, średnie położenie punktów zestawów $\mathbb{A}$ i $\mathbb{B}$.

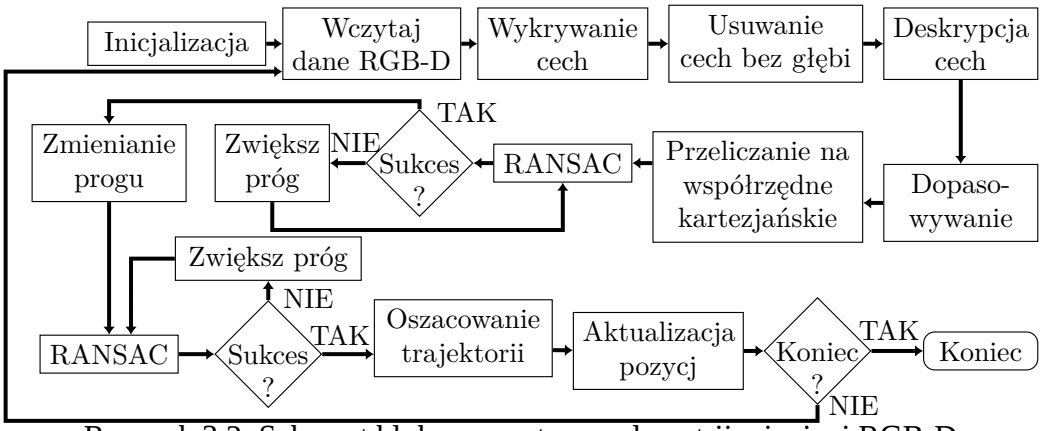
3.6 Struktura systemu

Wykorzystywany do badań system RGB-D VO (3.2) jest bazującym na cechach podejściem do śledzenia kamery [207], który oparto na procedurach biblioteki OpenCV [107]. Prostota tego podejścia zapewnia, że bardziej zaawansowane technikik, takie jak lokalna optymalizacja w oknie [130, 99], nie przysłoni wpływu uzupełniania głębi bądź optymalizacji parametrów na ostateczny wynik.

Stworzono osobne wersje dla różnych par detektorów i deskryptorów cech. W trakcie inicjalizacji algorytm wybiera parametry badanej cząstki/osobnika (mogą być dobrane ręcznie), które były stałe podczas obliczania całej sekwencji (wyjątek stanowiły badania online nad detektorem AKAZE, gdzie zmieniano próg detekcji oraz dodatkowy parametr odpowiedzialny za percentyl–parametry RANSAC nadal pozostawały bez zmian).

Wczytywany jest również pierwszy kolorowy obraz (RGB), na którym są wykrywane punkty kluczowe (cechy). W wersji podstawowej usuwane są cechy bez odpowiadających im danych głębi. Wszystkie pozostałe cechy są opisywane przez deskryptory. Dalej dopasowuje się punkty kluczowe z nowo wczytanego obrazu z cechami widzianymi poprzednio. Stosowane jest dopasowywanie deskryptorów między obrazami w obu kierunkach, aby pozbyć się błędnych dopasowań możliwie szybko. Wykorzystujemy do tego celu metodę siłową (ang. *BruteForce*), ponieważ zastosowanie przybliżonego sposobu dopasowywania (np. popularnej biblioteki FLANN) mogłoby zwiększyć losowość wyników. Teraz po raz kolejny wykorzystuje się informacje dostarczone przez mapy głębi do obliczenia pozycji w przestrzeni $3D$ tych dwóch dopasowanych zestawów punktów otrzymując ich kartezjańskie współrzędne.

W celu usunięcia pozostałych błędnych dopasowań, negatywnie wpływających na obliczanie transformacji, stosowana jest procedura RANSAC [129], którą wykonujemy dwukrotnie [163]. W każdej z tych procedur losujemy 3 pary odpowiadających sobie punktów–minimalną liczbę niezbędną



Rysunek 3.2: Schemat blokowy systemu odometrii wizyjnej RGB-D

do wyznaczenia transformacji miedzy dwoma ujęciami za pomocą algorytmu Kabscha [154, 153] (w przypadku badań nad metodami n-punktowymi stosowano odpowiednie algorytmy). W ten sposób wyznaczamy roto-translację ($[\mathbf{R}, \mathbf{t}]^T \in SE(3)$) pomiędzy zestawem punktów z poprzedniego ujęcia, a obecnym.

W kolejnych iteracjach procedury RANSAC dopasowane cechy z poprzedniej klatki przesuwa się i obraca zgodnie z wyznaczonym minimalnym modelem transformacji, a następnie sprawdza się jak daleko znajdują się one od cech z kolejnej klatki RGB-D. Jeśli są zbyt daleko, to uznaje się je za punkty odstające (ang. *outliers*). Jeżeli punktów odstających jest zbyt wiele w stosunku do poprawnie dopasowanych, to taką wstępną propozycję rotacji i translacji odrzucamy i losujemy kolejną parę 3 punktów. Gdy tych losowań jest zbyt wiele, zwiększamy próg akceptowalnej odległości, po którego przekroczeniu daną parę punktów uznajemy za źle dopasowaną i kontynuujemy poszukiwania. Te 2 progi, występujące w 2, następujących po sobie procedurach RANSAC, są jedynymi (poza jednym wyjątkowym badaniem) parametrami, które zmieniamy w trakcie szacowania przebytej przez kamerę trajektorii dla danej sekwencji. Nie zmieniamy w trakcie tego procesu progu detekcji τ_A–jest on stały. W związku z tym nasze podejście nie jest w stanie zareagować na zmiany sceny, jakie mogą nastąpić wzdłuż trajektorii. Wyjątek stanowi jedynie próba optymalizacji parametrów online.

Wszystkie pary punktów, które przeszły przez filtrację RANSAC (ang. *inliers*) wykorzystujemy do obliczenia przesunięcia i obrotu pomiędzy rozpatrywanymi dwoma klatkami RGB-D, również korzystając z algorytmu

Kabscha. Następnie zapisywane są dane benchamrkowe (takie jak czasy trwania poszczególnych operacji oraz liczby punktów kluczowych, inlierów i outlierów) oraz dokonywana jest aktualizacja pozycji sensora poprzez dołączenie wyznaczonej roto-translacji do poprzednio wyznaczonej pozycji sensora i cyklicznie powtarza się tę operację dla kolejno wczytanych klatek.

Rozdział 4

Wybrane metody optymalizacji parametrów systemu lokalizacji

Przyroda od wieków w doskonały sposób radziła sobie z przystosowaniem do środowiska, opanowała najlepsze techniki przetrwania i optymalizacji. Potencjał ten dostrzeżono w ubiegłych stuleciach tworząc różnego rodzaju maszyny naśladujące żywe organizmy. Wraz z pojawieniem się maszyn liczących zaczęto przeprowadzać również różnego rodzaju symulacje i podjęto próby stworzenia systemów zawierających niezbędne aspekty życia, z daleko idącym zamiarem stworzenia sztucznego życia (ang. *Artificial Life, ALife*). Tym zagadnieniem zajmuje się wiele dziedzin naukowych–zarówno nauki ścisłe, przyrodnicze jak i humanistyczne. Do tej pory powstało wiele metod inspirowanych naturą. Zaliczyć do nich można np. algorytmy genetyczne, ewolucyjne, rojowe czy, sztuczne sieci neuronowe. W niniejszej rozprawie główny nacisk położono na techniczną stronę tego zagadnienia.

W naszych badaniach nad optymalizacją parametrów skupiliśmy się na metodzie roju cząstek PSO (ang. *Particle Swarm Optimization*) oraz algorytmie ewolucyjnym EA (ang. *Evolutionary Algorithm*), gdyż oba posiadają, uzupełniające się, mocne strony. PSO znany jest ze swojej szybkiej konwergencji i prostoty, podczas gdy adaptacyjny EA odznacza się wydajnością, wymagając w zamian minimalnego dostrajania parametrów i lepiej utrzymując zróżnicowaną populację. W celach porównawczych, zastosowano również również bardzo prostą metodę przeszukania logarytmicznego.

4.1 Metoda przeszukania logarytmicznego

Nim zostaną opisane bardziej zaawansowane metody, na początku wypada opisać prostsze podejście, jakim jest chociażby prosta metoda przeszukania logarytmicznego. To podejście pozwala na szybkie i skuteczne poszukiwanie optymalnej wartości parametru w przypadku prostych problemów.

Całość sprowadza się do wyczerpującego przeszukania zakresu danego parametru z logarytmicznym krokiem, tak jak zrobiono to w [191]. Zakres możliwych nastaw parametrów zostaje logarytmicznie spróbowany, tj. poczynając od najmniejszej dozwolonej wartości parametru, kolejną wartość otrzymuje się poprzez pomnożenie poprzedniej przez arbitralnie dobrany współczynnik k do momentu gdy osiągnięta w ten sposób wartość przekroczyłaby górną granicę.

Po znalezieniu najodpowiedniej nastawy, pozwalającego uzyskać najlepsze wyniki, takie poszukiwania mogą być dalej poprawiane w jego okolicy w ten sam sposób ale dla mniejszego zakresu–od wartości poprzedzającej (to jest nowa minimalna wartość parametru) do następującej po niej (nowej maksymalnej wartości parametru). W ten sposób poprzednio najlepszy parametr zajmuje środkowy numer w liście badanych rozwiązań. Oczywiście można zmienić liczbę kroków (zwykle się ją zwiększa). Całość podsumowuje algorytm 3 gdzie p_{min}, p_{max} oraz p_{opt} to początkowo najmniejsza, największa i optymalna wartość parametru p w tym przeszukaniu, a k to krok logarytmiczny.

W trakcie naszych badań cały proces zajmował 2 iteracje za optymalny parametr przeszukania przestrzeni obierano ten znaleziony po drugiej iteracji. Wartości parametrów wykorzystywane były do oszacowania przebytej przez kamerę trajektorii, którą dalej oceniano według przyjętej miary–u nas RPE. Współczynnik k dobrano tak, by liczba badanych parametrów była równa ilości dostępnych wątków (12). W związku z tym podczas drugiej iteracji dokładniej spróbkowano badany parametr, bo wykorzystano wyniki dla minimalnego, maksymalnego i środkowego parametru poprzez odpowiednie dobranie współczynnika k by znów korzystać z 12 wątków.

Algorytm 3: Metoda przeszukania logarytmicznego

1. Ustaw początkową wartość parametru $p = p_{min}$,
2. Sprawdź działanie dla parametru o wartości p,
3. Ustaw $p_{opt} = p_{min}$,
4. Zmień wartość $p = p \cdot k$,

while $p < p_{max}$ **do**
    Zmień wartość $p = p \cdot k$,
    Sprawdź działanie dla parametru o wartości p,
    Jeśli p lepsze od p_{opt} ustaw $p_{opt} = p$.

W celu poprawy rozwiązania ustaw $p_{min} = p_{opt}/k$ oraz $p_{max} = p_{opt} * k$ i rozpocznij algorytm od 1 kroku.

4.2 Metoda roju cząstek PSO

Ten algorytm rojowy PSO (ang. *Particle Swarm Optimization*) nadaje się do optymalizacji parametrów ciągłych i nieliniowych funkcji wielowymiarowych. Podstawę do stworzenia tego algorytmu była obserwacja E.O. Wilsona (socjologa), mówiące że przynajmniej w teorii, pojedyncze jednostki ławicy, podczas poszukiwania pokarmu, mogą osiągnąć korzyści z odkryć i wcześniejszych doświadczeń wszystkich innych jej członków. Te korzyści mogą być decydujące, przeważając niekorzyści konkurowania o pokarm, ilekroć fragmenty zasobów są nieprzewidywalnie rozmieszczone. Zasugerowało to autorom algorytmu, że społeczne dzielenie informacji poprzez członków roju daje im ewolucyjną przewagę.

To podejście symuluje np. stado ptaków poszukujące pola kukurydzy [119]. Zostało to przedstawione na schemacie 4.2. Jednakże w odróżnienia od rzeczywistego stada, ławicy czy roju, tu, na cząstki nie są narzucane fizyczne ograniczenia. W n-wymiarowej przestrzeni jedno miejsce może zajmować więcej niż jedna cząstka. Nasze „zwierzęta“ są odporne na kolizje.

Podczas inicjalizacji losuje się m cząstek (zwykle $20 - 40$) składających się z n parametrów i prędkości zmian tych parametrów. Te prędkości nie mających fizycznego uzasadnienia ale kontrolują przeszukiwanie przestrze-

ni rozwiązań–im prędkość jest większa, tym większe zmiany są dokonywane pomiędzy kolejnymi iteracjami. Następnie sprawdza się, jak dobrze, względem przyjętej miary, zestawy parametrów poszczególnych cząstek są przystosowane do środowiska. W nieniejszej pracy przystosowanie cząstek zależy od RMSE części translacyjnej miary RPE lub miary ATE dokładności lokalizacji, w zależności od wybranego wariantu optymalizacji.

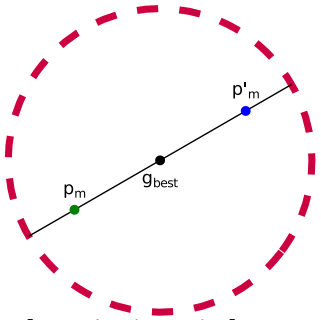
Dla m-tej cząstki określa się zestaw najlepszych dotychczas znalezionych przez nią parametrów, jak również najlepszy zestaw parametrów znalezionych przez którąkolwiek z cząstek (globalny). Na tej podstawie zmienia się prędkość i położenie każdej cząstki w przestrzeni parametrów, co przedstawiają równania (4.1):

$$\begin{aligned} \mathbf{v}_m^{i+1} &= \mathbf{v}_m^i + c_1 * \text{rand}() * (\mathbf{l}_{\text{best}} - \mathbf{p}_m^i) + c_2 * \text{rand}()(\mathbf{g}_{\text{best}} - \mathbf{p}_m^i) \\ \mathbf{p}_m^{i+1} &= \mathbf{p}_m^i + \mathbf{v}_m^{i+1} \end{aligned} \tag{4.1}$$

gdzie $\mathbf{v}_m^i$ to obecna prędkość m-tej cząstki $\mathbf{p}_m^i$ (niektórzy dodają do tego parametru wagę), c_1 i c_2 są stałymi wartościami, $\mathbf{l}_{\text{best}}$ to wektor najlepszych parametrów znalezionych przez badaną cząstkę, $\mathbf{g}_{\text{best}}$ to zestaw globalnie najlepszych znalezionych parametrów, $\mathbf{p}_m^{i+1}$ i $\mathbf{v}_m^{i+1}$ to kolejno nowe położenie i prędkość m-tej cząstki, a $\text{rand}()$ oznacza funkcję generującą liczby pseudo–losowe. Oczywiście prędkości i położenia są arbitralnie ograniczane. Niekiedy bierze się pod uwagę najbliższe cząstki–wówczas $\mathbf{g}_{\text{best}}$ zastępuję się $\mathbf{l}_{\text{best}}$–zestawem lokalnie najlepszych parametrów.

W początkowych badaniach nie zmieniano prędkości na podstawie różnic położeń, a o pewną losową stałą przemnożoną przez odpowiedni współczynnik. Powstałe tak zbyt mocne dążenie do najlepszych lokalnie parametrów, w stosunku do globalnych nastaw, powodowało nadmierne błądzenie odizolowanych cząstek w przestrzeni problemu. W przypadku odwrotnym powodowało to przedwczesny dążenie cząstek do lokalnego minimum.

Najefektywniejsze okazało się w równe ustawienie dążeń do wartości tych parametrów. Parametrom $\mathbf{l}_{\text{best}}$ przypisuje się historycznie najlepsze ustawienia pozwalające danej jednostce osiągnąć najlepsze wyniki. Natomiast $\mathbf{g}_{\text{best}}$ utożsamia się z dostępną publicznie wiedzą. Stałe c ustawiono na 2 (takie też wartości wykorzystujemy w naszych badaniach) by nadać jednostkową wartość oczekiwaną części losowej, Ten człon pozwala cząstce na przyjęcie wszystkie wartości pomiędzy p_m a p'_m. Poglądowo przedstawiono to na rys. 4.1. Niektórzy badacze dobierają inaczej wartości parametrów c_1 i c_2, ale zwykle jako równe i będące w przedziale $0 - 4$.

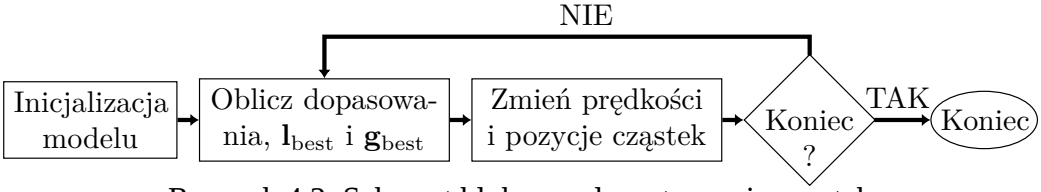


Rysunek 4.1: Obrazowe przedstawienie pojedynczego członu zmiany prędkości cząstki

W początkowych fazach prac nie istniały człony odpowiedzialne za lokalnie i globalnie najlepsze parametry. Dodano je, ponieważ całe stado ptaków szybko zaczęło poruszać się w jednomyślnym, niezmiennym kierunku. Wymuszano także dopasowywanie prędkości danego członka stada do prędkości innego mu najbliżej położonego. Po ich usunięciu z algorytmu symulacje przestały przypominać zachowanie stada, a bardziej roju, realizują wszystkie 5 zasad paradygmatu Millonasa [179].

Pierwsza zasada to bliskość, sąsiedztwo: populacja wykonuje obliczenia w n-wymiarowej przestrzeni w seri kroków w czasie. Następna to odpowiadanie na miarę jakości, wpływa na to $\mathbf{l}_{\text{best}}$ i $\mathbf{g}_{\text{best}}$. Dalej populacja powinna zróżnicowanie odpowiadać: zapewnia to ulokowajnie odpowiedzi gdzieś pomiędzy $\mathbf{l}_{\text{best}}$ i $\mathbf{g}_{\text{best}}$. Po czwarte powinna być stabilna–nie powinna zmieniać swojego zachowania z każdą zmianą środowiska. W przypadku PSO dzieje się tak, gdy $\mathbf{g}_{\text{best}}$ jest stałe. Ostatnia zasada to taka, że musi być w stanie zmieniać swoje zachowanie jeśli jest to warte nakładu obliczeniowego- czyli gdy $\mathbf{g}_{\text{best}}$ ulegnie zmiani.

Operacje (4.1) są wykonywane tak długo, aż spełniony zostanie warunek stopu, którym jest satysfakcjonujący poziom dopasowania do środowiska, brak poprawy wyniku dopasowania przez kilka kolejnych iteracji, bądź też przekroczenie maksymalnej liczby iteracji.



Rysunek 4.2: Schemat blokowy algorytmu roju cząstek

Ten algorytm cechuje się dużą prędkością obliczeń zmian dla poszcze-

gólnych cząstek oraz szybką zbieżnością, która niestety może prowadzić do utraty różnorodności cząstek, czego efektem może być przedwczesna zbieżność. Jednym z istniejących rozwiązań tego problemu jest algorytm PSO z perturbacją [243], wprowadzającą perturbację do globalnie najlepszego rozwiązania by zachować różnorodność.

4.3 Algorytm ewolucyjny

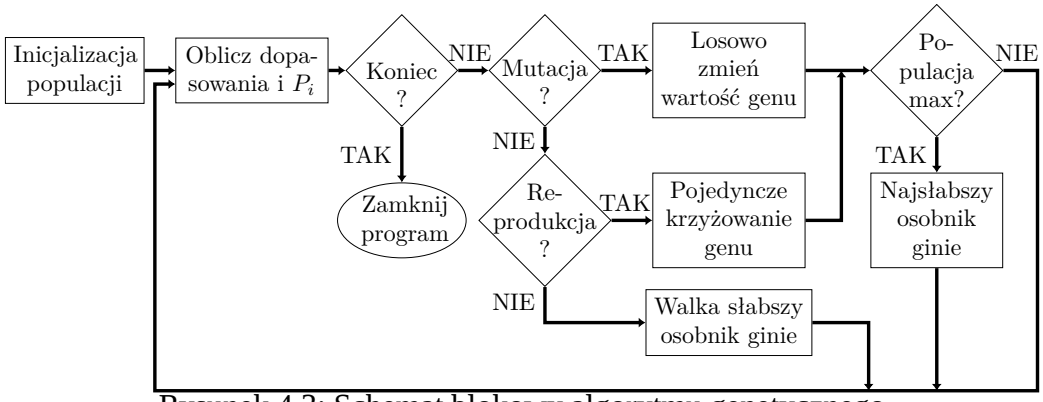
Istnieje kilka podejść wykorzystujących ewolucyjny paradygmat [100]: algorytmy genetyczne GA (ang. *Genetic Algorithms*), programowanie genetyczne GP (ang. *Genetic Programming*) czy strategie ewolucyjne ES (ang. *Evolution Strategies*), które różnią się zarówno sposobem kodowania problemu jako populacji jednostek, jak i wykorzystywanymi operacjami genetycznymi.

Z tego powodu trudno jest dobrać odpowiednie podejście do danego problemu badawczego. Trzeba wziąć pod uwagę właściwości samego problemu jak i parametry wykorzystywanej symulacji. W naszym przypadku trzeba przeszukać dużą przestrzeń rozwiązań. Badane nastawy parametrów to wartości rzeczywiste, natomiast klasyczne GA używają jednostek reprezentowanych poprzez binarnie kodowane genotypy i wymagają mapowania z genotypu do reprezentacji problemu za pomocą liczb rzeczywistych. Jednocześnie nie ma dostępnych metod heurystycznych które pozwoliłyby na ustalenie rozmiaru populacji, liczby pokoleń, czy poszczególnych operacji genetycznych. W rozważanych w rozprawie zagadnieniach większość czasu badań pochłaniają obliczenia przebytej przez kamerę trajektorii, a nie dobory parametrów algorytmu RANSAC czy detektora.

W związku z tym preferujemy algorytm ewolucyjny, który pozwala na bezpośrednie reprezentowanie genów za pomocą rzeczywistych wartości. Kolejnym problemem związanym z klasycznym GA jest parametryzacja. Mimo, że GA dowiodła, że jest wydajną techniką heurystycznego poszukiwania w wielu zastosowaniach, to nie jest do końca jasne jak GA powinno być sparametryzowane i skonfigurowane by pasować do konkretnego problemu. Dobór odpowiedniej strategii selekcji, operacji rekombinacji (ang. *crossover*), czy prawdopodobieństwa mutacji dla bazującego na GA systemu doboru parametrów do zadania odometrii wizyjnej prawdopodobnie byłoby trudnym zadaniem. Mogłoby to wymagać wielu symulacji i zająć dużo czasu, a jeśli wzięto by odpowiednie ustawienia to można by otrzy-

mać wyraźnie gorsze wyniki. Dobór tych parametrów to kolejny, oprócz rozważanego, problem optymalizacyjny. W związku z tym, algorytm który potrafi samemu adoptować swoje parametry podczas poszukiwań, jest interesującym i atrakcyjnym wyborem. Naśladuje to naturalne strategie ewolucji społeczeństw, żyjących w środowisku z ograniczonymi zasobami. Ich rozmiary, stopień reprodukcji czy rywalizacji jest zmienny w czasie.

Spełniającym te wymagania jest „samoregulujący“ się [91] algorytm ewolucyjny 4.3 EA (ang. *Evolutionary Algorithm*) wzorowany na naturalnych ekosystemach. Przenosi on genetyczne postrzeganie zagadnienia w stronę sztucznej ewolucji społeczeństw. Taki algorytm pomyślnie zastosowano w nauce chodu sześcionożnego robota [100]. Pomimo, że rozważane tu zagadnienie jest nieco inne, to problemy w jego rozwiązaniu zdają się być podobne, więc zastosowanie tego podejścia budzi nadzieje na powodzenie.



Rysunek 4.3: Schemat blokowy algorytmu genetycznego

Wybrane podejście charakteryzuje się minimalną liczbą parametrów, które dobierane są przez użytkownika. Każdy osobnik reprezentuje indywidualne rozwiązanie problemu, którego genotyp kodowany jest za pomocą liczb rzeczywistych (typu `float`), a nie za pomocą ciągu binarnego (jak w oryginalnym podejściu). Na wstępie określany jest maksymalny rozmiar populacji r, obrazujący skończone zasoby środowiska, oraz liczba osobników w pierwszym pokoleniu (choć oryginalnie jest ona losowana), dla których losowane są parametry początkowe (genomy). Rzeczywisty rozmiar populacji jest wypadkową operacji rywalizacji pomiędzy osobnikami i reprodukcji, zależnymi od stopnia zagęszczenia populacji.

Kolejne pokolenia osobników są tworzone i ewaluowane w pętli aż do

spełnienia jednego z warunków stopu, które są identyczne z opisanymi dla algorytmu PSO (oryginalnie algorytm [91] kończył działanie po wykonaniu obliczeń dla zadanej liczby pokoleń/iteracji). Osobniki są oceniane względem części translacyjnej miary RPE lub miary ATE. Następnie, z prawdopodobieństwem P_i (gdzie i jest numerem pokolenia-iteracji) równym stosunkowi początkowej liczby osobników w danym pokoleniu do ich maksymalnej liczby, rozpatruje się czy i-ty osobnik wchodzi w interakcję (zastępuje to selekcję z GA) z innym losowym osobnikiem, a w przeciwnym razie mutuje (oryginalnie z prawdopodobieństwem równym reprodukcji, u nas zawsze). W pierwszym przypadku z tym samym prawdopodobieństwem P_i losujemy czy będzie to walka–jeśli tak to słabszy (o mniejszej wartości funkcji dopasowania) osobnik ginie. Jeżeli nie, to dochodzi do rozmnażania, które realizowane jest jako wymiana części genomu między krzyżowanymi osobnikami. Punkt krzyżowania jest jeden, znajduje się pomiędzy dwoma wartościami rodzicielskich genomów i jest losowanym za pomocą rozkładu normalnego. W odróżnieniu od standardowego GA, w tym przypadku tak powstały potomek nie zastępuje swoich rodziców. Mutację realizujemy jako inicjowanie nowego osobnika o losowo zmienionej wartości jednego genu względem osobnika macierzystego, przy czym oryginalny osobnik zostaje zachowany.

Jeśli takie działanie powoduje wzrost populacji ponad jej maksymalną wartość, to ginie tylko gorzej dopasowany osobnik. Może to być jeden z rodziców (bądź jeden rodzic w przypadku mutacji), bądź dziecko. Ten mechanizm kontroluje charakter poszukiwań globalnego optimum jednocześnie dbając o szybką zbieżność. Krzyżowanie i mutacja postrzegane są jako dwie różne drogi reprodukcji.

Parametry algorytmu ewolucyjnego zostały tak dobrane, aby operował na populacji o liczebności zbliżonej do użytej w algorytmie PSO. Początkowa liczba osobników w populacji wyniosła 10, a maksymalną ich liczbę ustalono na 40. Maksymalną liczbę iteracji ustalono na 20.

4.4 Porównanie metod

Obydwa podejścia posiadają cechy wspólne. Zaczynają z losowo inicjowaną populacją o określonej liczbie członków, dla których liczona jest miara dopasowania. Obydwie zmieniają parametry populacji i poszukują optimum korzystając z technik losowych. Nie gwarantują sukcesu–znalezienia

optimum.

Algorytm PSO nie posiada operacji genetycznych, za to cząstki posiadają wewnętrzny parametr–prędkość i związaną z nim pewnego rodzaju pamięć. Członkowie populacji inaczej dzielą informacje–w pierwszym zbiór najlepszych parametrów znaleziony przez cząstkę rozpowszechniany jest jednorodnie dla wszystkich cząstek. W EA informację wymieniają z sobą osobniki biorące udział w krzyżowaniu.

4.5 Miary dopasowania

W zastosowaniach metod populacyjnych istotny jest właściwy wybór miary dopasowania osobników/cząstek do środowiska. W rozważanym przypadku miara dopasowania musi wiązać się z jakością estymowanej trajektorii. Dlatego jako miary dopasowania wykorzystano bezwzględny błąd trajektorii ATE (ang. *Absolute Trajectory Error*) oraz względny błąd translacji RPE (ang. *Relative Pose Error*). Są to miary jakości zaproponowane w [225] i powszechnie stosowane w robotyce do ewaluacji systemów SLAM oraz VO. Są to te same miary, które opisano dokładniej w podrozdziale 6.5.

Rozdział 5

Wybrane metody uczenia maszynowego

Uczenie maszynowe, a ostatnio głębokie sieci neuronowe pozwoliły drastycznie przesunąć poziom rozwiązań w wizji komputerowej, rozpoznawaniu mowy, tłumaczeniu i wielu wielu innych dziedzinach zagadnieniach.

By te cele osiągnąć bardzo ważne jest precyzyjne określenie zadania jakie ma realizować sieć, do czego jest to porównywane, jakimi środkami oraz jak są oceniane uzyskane efekty, jak mierzony jest błąd, różnica pomiędzy tym co chcieliśmy uzyskać a tym co faktycznie otrzymujemy. Dlatego w tym rozdziale zostaną opisane motywacje stojące za wyborem platformy programistycznej służącej do uczenia sztucznych sieci neuronowych, wybrane architektury sieci neuronowych, oraz przygotowanie zestawu uczącego. Nie opisano natomiast funkcji aktywacji, uznając je za powszechnie znane.

5.1 Podstawowe pojęcia

By ułatwić przegląd pracy, w niniejszym podrozdziale zostanie przypomniana definicja kilku powszechnie wykorzystywanych elementów i pojęć. Wiele pojęć zwięźle wyjaśnia również leksykon [226].

5.1.1 Zestawy danych–rodzaje

Zestawem danych nazywa się zbiór wykorzystywanych przez sieć danych wejściowych oraz odpowiadającym im etykietom–czyli tym co powinno być na wyjściu sieci. Zwykle dzieli się je na 3 kategorie: uczący, walidacyjny i testowy. Zestaw uczący służy do uczenia sieci–na jego podstawie zmienia się wagi zapisane w sieci. Zbiór walidacyjny nie jest wykorzystywany do uczenia a do oceny jak stworzony model radzi sobie z nowymi, niewidzianymi podczas uczenia danymi. Zbiór testowy służy do niezależnej oceny stworzonego rozwiązania. Zwykle nie jest on publicznie udostępniany by nie dopuścić do nadużyć i potencjalnego wykorzystania go w procesie uczenia w celu np. wygrania konkursu.

5.1.2 Błędy

Błąd to odpowiednio potraktowana, przez wybraną metrykę, różnica pomiędzy generowaną przez sieć odpowiedzią a wzorcową. Zwykle mówiąc o błędzie ma się na myśli średni błąd, jest generowany przez poszczególne pakiety w epoce. Dla poszczególnych zestawów, tj,:treningowego, walidacyjnego i testowego, mają adekwatne nazwy, tj.: błąd treningowy, błąd walidacyjny i błąd testowy.

Innym rodzajem błędu jest błąd generalizacji, rozumiany jako różnica pomiędzy błędem błędem uczenia a błędem testowym.

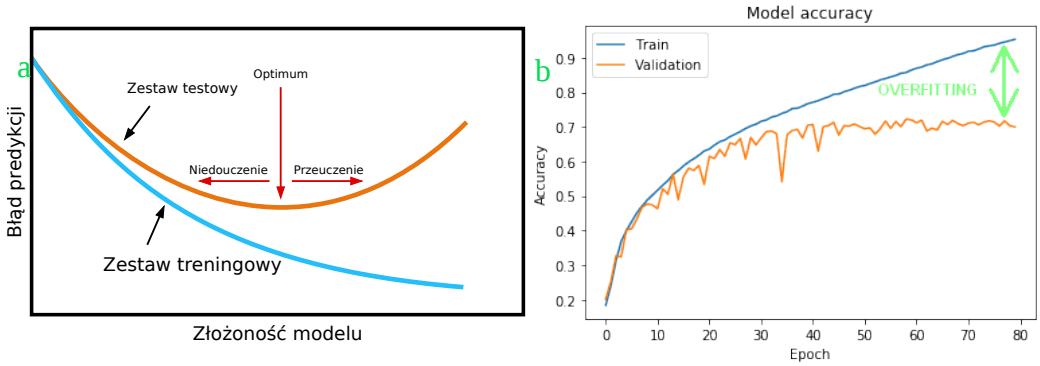
5.1.3 Uczenie

Celem uczenia sieci, czyli dopasowywania modelu do zbioru uczącego, jest taki dobór jego wag Θ, by dla zestawu testowego różnica między odpowiedzią sieci a wzorcową (czyli błąd) była jak najmniejsza, by uzyskane wyniki były optymalne.

$$\Theta = arg \min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} l(x_i, \Theta) \tag{5.1}$$

gdzie $x_{1,\ldots,N}$ to zestaw uczący, a N liczba to przykładów.

Jest to również znalezienie balansu pomiędzy niedouczeniem a przeuczeniem–takiego, by dla zestawu walidacyjnego, w końcowej fazie wykres funkcji błędu przebiegał horyzontalnie. Pokazuje to rys. 5.1a.



Rysunek 5.1: a) Obrazowe przedstawienie kompromisu pomiędzy niedouczeniem a przeuczeniem. Złożoność modelu na osi x odnosi się do pojemności, siły modelu. b) Przeuczenie sieci neuronowej [20]

Niedouczenie

Bardziej dosłownym tłumaczeniem angielskiego *underfittig* byłoby niedopasowanie. Jest to sytuacja w której uczony maszynowo model nie jest w stanie zmniejszyć błędu ani dla zestawu testowego ani uczącego. Wynika to ze zbyt małej pojemności modelu, by mógł on się dopasować do występujących w rozkładzie danych złożoności.

Przeuczenie

W trakcie uczenia sieci neuronowej może się zdarzyć, że dla zbioru uczącego uzyskujemy coraz lepsze wyniki, choć nie widać już takiej poprawy dla zbioru walidacyjnego. Oznacza to, że model zbyt mocno dopasował się do zestawu uczącego i nie jest już w stanie tak dobrze uogólniać wyników, czyli rośnie błąd generalizacji. Pokazuje to rys. 5.1b.

5.1.4 Epoka

Kompletne, jednorazowe przejrzenie wszystkich przypadków dostępnych w zestawie treningowym nazywane jest epoką. W każdej epoce kolejność przykładów jest losowa.

5.1.5 Nastawianie hiperparametrów

Celem jest taki dobór parametrów decydujących o przebiegu uczenia, aby było ono jak najszybsze i najlepsze. Dzieje się tak, gdy wykres funkcji błędu zestawu walidacyjnego (do testowego zbioru zwykle nie ma się dostępu) na początku opadał a w końcowej fazie przebiegał horyzontalnie.

5.1.6 Douczanie

W sieci internetowej można znaleźć, w tak zwanym zoo, wiele popularnych architektur sieci neuronowych wraz z wyuczonymi już wagami. W większości przypadków nie są one najlepsze do nowego zagadnienia ale jeśli jest ono podobne, zamiast zaczynać uczenie od nowa można starać się poprawić te wagi i dzięki temu zyskać na czasie. Proces ten nazywa się douczaniem sieci neuronowej (ang. *transfer learning*).

W ostatnich czasach dość modne jest zamrażanie wszystkich warstw (czyli ignorowanie zmian ich parametrów) z wyjątkiem ostatniej, zwykle będącej klasyfikatorem, łącznie z jej modyfikacją, wymianą, bo nie musi ona pozostać taka jak w oryginale. Proponowane jest to np. w bibliotece FastAi czy Kerasie. Wówczas pierwszym etapem jest nauczenie wag tego klasyfikatora, a dopiero później odmrażanie pozostałych (zezwolenie na ich zmianę).

5.1.7 Stochastyczny gradient prosty

Metoda ta służy do uczenia sieci–rozwiązania równania (5.1). Obecnie wykorzystywana jest stochastyczna aproksymacja gradientu prostego (ang. *Stochastic Gradient Descent*). Sposób postępowania przedstawia poniższe równanie (5.2):

$$\Theta_{i+1} = \Theta_i - \epsilon\delta L(F(x, \Theta_i)\Theta_i) \tag{5.2}$$

Dla przyspieszenia obliczeń, krok uczący wykonuje się po przetworzeniu kolejnego mini-pakietu zawierającego m przykładów uczących $x_{1,\ldots,m}$, a nie całego zestawu. Można na to patrzeć jako na pewnego rodzaju estymatę Monte Carlo naszych oczekiwań co do gradientu i funkcji błędu. Θ reprezentuje wszystkie parametry sieci, ϵ to waga ucząca (ang. *learning*

rate), a $\delta L(F, x, \Theta_i)\Theta_i)$ to gradient. Wówczas gradient funkcji strat można przybliżyć za pomocą gradientów wyliczonych w ramach mini-pakietu poprzez ich uśrednienie:

$$\frac{1}{m}\frac{\partial l(x_i, \Theta)}{\Theta} \tag{5.3}$$

Można patrzeć na to jako na pewnego rodzaju estymatę Monte Carlo naszych oczekiwań co do gradientu.

5.1.8 Momentum

Sposób liczenia ulega drobnej zmianie w przypadku uczenia z momentum (5.4):

$$\begin{aligned} \upsilon_{i+1} &= \alpha\upsilon_i - \epsilon\delta L(F(x, \Theta_i)\Theta_i) \\ \Theta_{i+1} &= \Theta_i - \upsilon) \end{aligned} \tag{5.4}$$

gdzie υ to prędkość a α to współczynnik momentum. Z tych równań wynika, że momentum ma podobny wpływ na zmianę wag jak waga ucząca. Prędkość jest średnią ruchomą gradientu.

Test progów uczenia

Zwykle, by znaleźć optymalną wagę uczenia, korzystano z metody przeszukiwania siatki (ang. *grid search*) bądź przeszukiwania losowego (ang. *random search*). Leslie N. Smith w [221] zaproponował przeprowadzenie testu w celu odnalezienia maksymalnego i minimalnego progu uczenia sieci. W swych badaniach autor zauważył, że zbyt mała waga ucząca ma tendencje do przeuczania sieci.

Polega on na rozpoczęciu nauki sieci, ale w procesie poprzedzającym właściwe uczenie sieci, małym progiem uczenia, który jest następnie powoli, liniowo zwiększany. Następnie tworzony jest wykres, w którym na jednej osi są błędy a na drugiej wartość progu uczenia. Za maksymalną wartością progu uczenia należy obrać taką, przy której błąd odpowiedzi sieci jeszcze drastycznie nie rośnie. Jeśli chodzi o minimalną wartość to autor zasugerował 3 podejścia, z czego w rozprawie wykorzystywano wartość $10 - 20$ mniejszą od maksymalnej. W przypadku uczenia ze stałym progiem uczenia, nie należy brać maksymalnego progu uczenia a zdecydować się na $2 - 3$ mniejszą wartość.

Uczenie w cyklu

W swoim raporcie [220] Leslie Smith zaproponował, by przed przystąpieniem do uczenia sieci, sprawdzić jak zmienia się błąd predykcji sieci w zależności od zwiększanego współczynnika uczenia.

Sieć należy uczyć najpierw zwiększając współczynnik uczenia z wartości kilku-krotnie mniejszej (zwykle 10-krotnie) od największej, przy której sieć się jeszcze uczy przez trochę mniej niż połowę całkowitej liczby iteracji przeznaczonej na naukę. Następnie należy przez taką samą liczbę iteracji zmniejszać współczynnik uczenia. Przez pozostałą część iteracji należy zmniejszać współczynnik uczenia do o kilka rzędów mniejszego niż początkowy najniższy.

Istnieje maksymalna prędkość, z jaką można zwiększać i zmniejszać wagę uczenia, po przekroczeniu której uczenie staje się niestabilne. By temu zaradzić można zwiększyć liczbę iteracji.

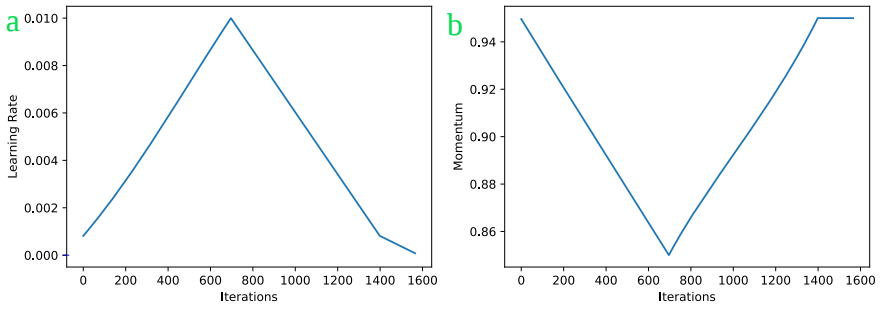
W swej poprzedniej pracy [221] (praca pochodzi z 2015 r. ale była następnie edytowana) autor badał różne sposoby zwiększania i zmniejszania progu uczenia i nie znalazł znaczących różnic pomiędzy doborem krzywej a finalnym rezultatem. W związku z czym proponuje najprostszą metodę, czyli liniowe zwiększanie i zmniejszanie.

Jeżeli w trakcie uczenia zmieniamy również momentum, to na początku powinniśmy je zmniejszać, by progu uczenia mógł odgrywać większą rolę w początkowej i środkowej fazie uczenia, a gdy zaczniemy zmniejszać wagę uczenia, to wtedy momentum należy zwiększać. Poszukiwania optymalnej wartości maksymalnej można zacząć od $0,99$ do $0,90$ a minimalną obrać jako $0,85$ albo $0,80$. Dobór minimalnej wagi momentum nie ma aż takiego znaczenia jak jego maksymalnej wartości.

Poniższy rys. 5.2 przedstawia jak się zmieniają wagi uczenia i momentum w trakcie cyklu.

5.1.9 Regularyzacja

Celem regularyzacji jest takie wpływanie na naukę sieci, by posiadała ona zdolność do generalizacji, czyli by osiągała dobre wyniki również z danymi, które nie były jej udostępnione, pokazane w procesie uczenia. Istnieje wiele jej form takie jak: duża waga ucząca, małe pakiety, weight decay czy dropout–losowo wybrane 50% aktywacji jest ignorowane i zamieniane na 0.Pozwala to na uodpornienie sieci na przeuczenie. Ich dobrego przeglądu



Rysunek 5.2: Zaproponowane przez Leslie N. Smith [220] krzywe zmian a) progu uczenia, b) współczynnika momentum

dokonano w [167]. W [220] postawiono hipotezę, że złożone dane zawierają już w sobie regularyzację i inne jej formy powinny być ograniczone. Pokazuje min. że w przypadku wielowymiarowych danych, redukowanie bądź rezygnacja z ograniczania wag (ang. *weight decay*) pozwala na użycie większych progów uczenia, przyspiesza je i pozwala na uzyskanie większej dokładności.

Z przeprowadzonych badań w [220] wynika, że do nauki konkretnego zadania wybranej architektury trzeba zbadać jaka jest optymalna wielkość pakietu. Przy czym nie powinno się tego robić przy stałej liczbie epok, bo nie jest brane pod uwagę zwiększenie wydajności obliczeniowej co skutkuje preferowaniem mniejszych pakietów. Z kolei branie pod uwagę samych iteracji zbyt mocno faworyzuje duże pakiety. Należy odpowiednio dobrać liczbę epok i iteracji do wielkości pakietu. W uczeniu w cyklu zbyt duże pakiety nie pozwalają na uczenie z dużym progiem uczenia, a głównym celem jest szybkie uczenie przy zachowaniu wysokiej jakości inferencji. Tym niemniej, jeśli nie posiada się możliwości sprzętowych do zwiększania ilości przykładów w pakiecie, należy liczyć z taką jaka się mieści w GPU i wykorzystać większą wagę uczenia.

Technika ograniczania wag sieci

Technika ta zmienia funkcję celu tak, by wagi nauczonej sieci nie były zbyt wielkie, co przedstawia poniższe równianie (5.5):

$$E = F + wd \sum (w^2) \tag{5.5}$$

Z tego równania widać, że ostateczny błąd E składa się z wybranej funkcji błędu F oraz przemnożonej przez wagę wd sumy kwadratów wszystkich wag w. Można to porównać z rozkładem (ang. *decay*) i stąd wywodzi się też angielska nazwa tej techniki *weight decay*.

Według badań przeprowadzonych przez Smitha [220] wartość tego współczynnika powinnapozostać stała podczas uczenia i warto skorzystać z metody przeszukiwania siatki. Tyczy się to również innych badanych tam metod regularyzacji. Według autora warto sprawdzić jako współczynnik ograniczania wag, wartości takie jak 10^{-3}, 10^{-4}, 10^{-5} czy 0. Jeśli natomiast podejrzewa się wstępnie, że np. wartość 10^{-4} sprawdziłaby się dobrze, to wówczas należałoby również sprawdzić 3×10^{-5} i 3×10^{-4}. Proces ten można powtórzyć następnie dla dwóch najlepszych wartości. Liczba 3 przewija się obecnie w wielu propozycjach, zamiast 5, ponieważ jest przybliżeniem połowy wykładnika potęgi a nie wartości (np. połową wykładnika pomiędzy 10^{-4} a 10^{-3} jest $10^{-3.5} = 3.16 \times 10^{-4}$). Przy czym z jego badań wynikło, że wystarczy tylko 1 liczba cyfra znacząca. W raporcie jest jeszcze jedna propozycja sposobu poszukiwania optymalnej wartości tego hiperparametru. Ten parametr ma inną wartość w zależności od tego czy waga ucząca jest stała czy nie.

5.1.10 Internal covariate shift

Jest to zmiana rozkładu aktywacji sieci spowodowana zmianą parametrów sieci podczas uczenia [150]. Każdy zbiór danych cechuje się pewnym rozkładem- np. pewną wartością średnią i wariancją dla każdego z kanałów RGB. Przy podaniu nowego zestawu może okazać się, że dane wejściowe mają przesunięty rozkład danych względem zestawu treningowego [21], którego nauczyła się sieć. W związku z tym wyniki będą gorsze, a do tego sieć będzie ciągle musieć się adoptować do nowego rozkładu. Mamy wtedy do czynienia z wewnętrznym przesunięciem zmiennej niezależnej (ang. *Internal covariate shift*).

5.1.11 Normalizacja wsadowa

W uczeniu sieci neuronowej często na wejście zamiast 1 przykładu podaje się kilka–tak zwany mini pakiet (ang. *mini-batch*). Oprócz zapewne pierwotnego celu, jakim było przyspieszenie obliczeń, z uwagi na możliwość

zrównoleglenia obliczeń przy wykorzystaniu GPU, taki zabieg pozwala również potraktować gradient błędu obliczony dla mini-pakietu jako estymata dla całego zbioru uczącego.

Mimo to, każdy taki pakiet może mieć inny rozkład, bo nie jest likwidowane wewnętrzne przesunięcie współzmiennej. By temu zaradzić można wymusić, poprzez normalizację (czyli by wartość średnia $\mu = 0$ i wariancja $\sigma^2 = 1$), stały rozkład danych wejściowych do sieci [150]. Można pójść nawet o krok dalej i normalizować wejścia do każdej z warstw sieci, bo to co było wyjściem z poprzedniej można potraktować jako wejście do kolejnej. Likwiduje to problemy z saturacją nieliniowych funkcji aktywacji, np. sigmoidalnej, a dalej z zanikającym gradientem błędu przy jego wstecznej propagacji. Dzięki temu można agresywniej uczyć sieć, bez stosowania takich technik jak np. Dropout, bo normalizacje jednocześnie redukuje zależność od wartości początkowych czy skali parametrów. W przeprowadzonych tam badaniach poprawę przyniosło również ograniczenie zniekształceń obrazu.

5.1.12 Indeks podobieństwa strukturalnego SSIM

Jest to następca Indeksu uniwersalnej jakości UQI. Powstał do oceny postrzeganej jakości obrazu. Wzorowano go na sposobie postrzegania rzeczywistości przez ludzi, gdzie zakłada się, że jest on bardzo dobrze przystosowany do wydobywania informacji strukturalnych ze sceny. Dla poszczególnego piksela obrazu indeks sformułowany jest następująco (5.6):

$$SSIM(x, y) = (l(x, y))^{\alpha} + (c(x, y))^{\beta} + (s(x, y))^{\gamma} \tag{5.6}$$

gdzie $l(x, y)$, $c(x, y)$, $s(x, y)$ oznaczają składową luminacji, kontrastu i struktury. Są one liczone w lokalnych oknach, których piksele były uprzednio odpowiednio zwarzone, wygładzone za pomocą filtru Gaussa. Autorzy [236] wykorzystują jądro przekształcenia Gaussa o wymiarach 11 na 11 pikseli i o odchyleniu standardowym równym $1, 5$ piksela, a następnie pro-

ponują takie wyliczenie poszczególnych komponentów miary (5.7):

$$
\begin{aligned}
l(x,y) &= \frac{2\mu_A\mu_B + C_1}{\mu_A^2 + \mu_B^2 + C_1} \quad \text{przy} \quad \mu_x = \sum_{i=1}^{N} w_i x_i \\
c(x,y) &= \frac{2\sigma_A\sigma_B + C_2}{\sigma_A^2 + \sigma_B^2 + C_2} \qquad\quad \sigma_x = (\sum_{i=1}^{N} w_i (x_i - \mu_x)^2)^{\frac{1}{2}} \\
s(x,y) &= \frac{\sigma_{AB} + C_3}{\sigma_A + \sigma_B + C_3} \quad \text{czym} \quad \sigma_{xy} = (\sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y))
\end{aligned} \tag{5.7}
$$

gdzie μ i σ oznaczają odpowiednio wartość średnią, wariancję i kowariancję w zależności od lokalnych, zwarzonych części obrazów wejściowych A i B. C_1, C_2 i C_3 to współczynniki dodane by uniknąć problemów ze stabilnością numeryczną, w sytuacjach gdy odpowiednie mianowniki byłyby bliskie zeru. Oryginalny artykuł proponuje następujący ich dobór: $C_1 = (K_1 L)^2, C_2 = (K_2 L)^2, C_3 = C_2/2$, przy czym $K_1 = 0.01, K_2 = 0.03$ a $L = 255$ będący zakresem wartości jakie piksel może przyjąć, tu dla 8-bitowego obrazu. α, β, γ to współczynniki > 0, choć dla otrzymania uproszczonej wersji można przyjąć $\alpha = \beta = \gamma = 1$.

W efekcie otrzymuje się nieco mniejsze mapy cieplne od obrazów wejściowych. Końcowe rozmiary tej mapy można policzyć jako $wymiar\ obrazu - wymiar\ okna + 1$. w proponowanym przypadku te wymiary są obcięte o 5 pikseli z każdej strony.

Zwykle wykorzystywana jest wartość średnia wszystkich miar SSIM danej pary obrazów, tj. $MSSIM(\mathbf{A}, \mathbf{B}) = \frac{1}{wh} \sum_x \sum_y SSIM(x, y)$

5.1.13 Krytyka SSIM

Do największych wad indeksu SSIM zarzuca się między innymi to, że może kierować uczenie sieci neuronowej w złym, niezamierzonym kierunku. W artykule [188] przebadano bardzo dokładnie omawianą funkcję i osoby zainteresowane dokładniejszą analizą powinny go przeczytać. W niniejszej rozprawie zostanie przybliżonych tylko kilka najistotniejszych, zdaniem autora, jej elementów.

Przede wszystkim zaczęto od zbadania matematycznych właściwości SSIM. Dwie pierwsze jej składowe posiadają min. powyżej 0 a ostatnia powyżej -1, przy czym dla każdej składowej wartość $max = 1$. Oznacza to,

że ten indeks przyjmuje wartości z zakresu $(0, 1]$. Minima oznaczają niską jakość, choć nietrudno znaleźć przykłady takich obrazów, to mogą być one nierozróżnialne dla człowieka patrzącego na nie na monitorze z normalnej odległości.

Współczynnik dotyczący struktury, jeśli przyjmie się $C_3 = 0$, staje się współczynnikiem korelacji Pearsona i nie znaleziono tam powiązania tego z postrzeganiem struktury przez człowieka. W dodatku tylko σ_{AB} może być ujemne. Wtedy przy podnoszeniu do parzystej ale niecałkowitej potęgi γ otrzyma się w wyniku liczę zespoloną, która nie jest dla większości języków programowanie zdefiniowana, a taki obrót sprawy nie powinien mieć miejsca. W dodatku, gdyby SSIM dopuszczał do swej dziedziny liczby zespolone, to nadal pozostaje problem ich interpretacji, bo nie jest on określony.

Zauważono również, że ten indeks ma zmienną czułość w zależności od intensywności porównywanego obrazu a wzorcowego–gdy ten jest czarny SSIM a porównywany prawie, to mała zmiana w luminacji tego obrazu powoduje przesadnie dużą zmianę części l. W dodatku ta składowa ma bardzo znikomą zmienność dla obrazów o luminacji $> 0,2$ (przy przyjęciu jej znormalizowanego zakresu $[0, 1]$ oraz, że ma jednolitą intensywność). W 90% przypadkach MSSIM jest bliskie 0. Gdy wzorcowy obraz jest biały ta zależność nie jest już tak widoczna ale znów gdy porównywany obraz jest bliski białości te zmiany stają się niewielkie.

W rozważanym w pracy przypadku oznacza to, że można uzyskać inne wyniki w zależności od odległości obiektu od kamery–$0,2$ natężenia przypada na ok $2,62[m]$ przy przyjęciu 5000 jednostek na $[m]$ czyli mniej więcej środek widzianego zakresu. Jeśli zestaw danych byłby zebrany przy obraniu stałej 1000 na $[m]$ to $0,2$ natężenia przypada na $0,52[m]$, czyli niebezpiecznie i nieakceptowalnie zbyt blisko minimalnego, wynoszącego $0,50[m]$, zakresu wykrywania odległości przez Kinect v2. Kolejnym problemem jaki się naturalnie pojawia jest możliwość uzyskania różnych wyników przy zmianie współczynnika głębi. W końcu nie bez znaczenia okazać się może już sam charakter sceny biorącej udział w badaniu–tj. średnia odległość przedmiotów od kamery.

W dodatku obiekty które będą dalej będą obarczone mniejszą karą podczas uczenia. Spowoduje to przesunięcie punktu środkowego widzianej przez kamerę chmury punktów, a co za tym idzie, będzie mieć wpływ nie tylko na wyliczenia dotyczące przebytej przez kamerę drogi, ale przede wszystkim na kąty jej obrotów, co jest kluczowe w odometrii wizyjnej.

Innym problemem tej miary jest to, że przed obliczeniami, transformuje obrazy kolorowe do skali szarości. Istnieje bardzo dużo różnych kolorów o jednakowej intensywności. Oznacza to, że ta miara może uznać za podobne zdjęcia o różnych barwach. Nie należy więc jej stosować do obrazów kolorowych. W niektórych przypadkach rozdzielczość zdjęcia również ma istotny wpływ na otrzymywaną wartość indeksu.

Wspomniane problemy mogą stać się się tym istotniejsze, im zbiór danych wejściowych do uczenia sieci jest mocniej sztucznie powiększany z użyciem różnych technik wpływających na kontrast, intensywność, drgania (ang. *jitter*) itd. Sam arbitralny dobór parametrów K, α, β czy γ, z perspektywy wykorzystania uczenia maszynowego w tej rozprawie, budzi pewne wątpliwości. Można by się pokusić o próbę wykorzystania jakiegoś algorytmu optymalizacyjnego, np. roju czy ewolucyjnego.

5.1.14 Krytyka RMSE

Błąd średniokwadratowy również posiada wady–opisano je w publikacji proponującej indeks SSIM [236]. Zarzuca się jej brak wrażliwości na strukturę obrazu–na możliwość uzyskania takiego samego błędu dla różnie zniekształconych, transformowanych wersji oryginalnego obrazu. Oznacza to, że sieć może nauczyć się zwracać mocno zniekształcone obrazy.

5.2 Wybór platformy programistycznej

Wraz ze wzrostem popularności metod głębokiego uczenia powstało mnóstwo platform programistycznych (ang. *framework*). W związku, z tym przed przystąpieniem do badań, dokonano rozpoznania obecnie najpopularniejszych bibliotek, by wybrać tę najodpowiedniejszą, dającą nie tylko możliwość uzyskania najlepszych wyników ale i największą nadzieję na to, że będą w przyszłości rozwijane i wspierane. Dzięki temu łatwe będzie nie tylko zweryfikowanie i odtworzenie przeprowadzonych badań ale i ponowne wykorzystanie kodu do innych celów.

Jednym z najważniejszych kryteriów była dostępność gotowych rozwiązań, architektur, narzędzi do uczenia oraz popularność–czyli zarówno możliwość uzyskania wsparcia w razie problemów, jak i zabezpieczenie się przed sytuacją, w której napisana praca i oprogramowanie staną się

przedwcześnie bezużyteczne. Platformy programistyczne i tak się rozwiną ale ważne, żeby wybrać tą z przyszłością.

Przyjrzano się stronom producentów kart graficznych, gdyż obecnie większość oprogramowania poświęconego sieciom neuronowym wykorzystuje te podzespoły a nie CPU. Na swoich stronach zarówno AMD jak i NVidia podają, i krótko charakteryzują najbardziej znane platformy programistyczne do głębokich sieci neuronowych. Niestety obecna sytuacja wygląda tak, że dla kart NVidi samoistnie powstały platformy programistyczne, z których firma upatrzyła sobie na forka (czyli niezależne rozwijanie projektu od wersji bazowej) Cafee, a dla kart AMD nie. W związku z tym ta druga firma tworzy własne wersje tych oprogramowań, by na ich kartach również były możliwe obliczenia.

Dodatkowo należy przyjrzeć się nowej tensorowej jednostce obliczeniowej TCU (ang. *tensor processing unit*). Nie nadaje się ona do zwykłych obliczeń ale za to bardzo szybko realizuje operacje macierzowe [22]. Te procesory zostały wynalezione przez firmę Google i póki co ta firma nie zamierza ich sprzedawać a jedynie umożliwiać korzystanie z nich w chmurze. Na dzień dzisiejszy tylko TensorFlow współpracuje z tego typu rozwiązaniem, choć istnieją pogłoski, że PyTorch wkrótce także może działać na tym sprzęcie [23]. Zważywszy na fakt, że TensorFlow pochodzi od Googla, zdecydowanie się na to oprogramowanie może oznaczać częściowe uzależnienie się od tej organizacji a z uwagi na powstałą dedykowaną platformę może to doprowadzić w przyszłości do sytuacji, w której by w pełni skorzystać z najnowszej wersji, trzeba będzie wykupić płatny dostęp do platformy obliczeniowej.

Z uwagi na obecne wykorzystywanie biblioteki OpenCV w rozprawie, należy sprawdzić które platformy programistyczne z nią współpracują. Aktualnie są to TensorFlow, Torch/PyTorch i Caffe. Nie oznacza to jednak, że nie należy przyjrzeć się również innym rozwiązaniom, gdyż wyćwiczone sieci można potem konwertować do innego, otwartego formatu wymiany sieci neuronowych ONNX (ang. *Open Neural Network Exchange*) stworzonego przez Microsoft we współpracy z Facebookiem, NVidiom i Qualcomem. Na stronie pytań OpenCV jest obecnie bardzo mało zapytań odnośnie PyTorch.

5.2.1 Caffe

Ta platforma programistyczna powstała na uniwersytecie w Berkeley [24]. W internecie można przeczytać różne pogłoski i pytania czy ta platforma programistyczna jest nadal rozwijana–ilość kontrybucji do tego oprogramowania jest od jakiegoś już czasu bardzo znikoma [25]. Powstały nawet wersje Intel Caffe, Windows Caffe i OpenCL, a więc działająca także na innym sprzęcie niż ten od NVidi. Nvidia stworzyła własną wersję–NVCaffe [26] ale nie cieszyła się ona dużą popularnością, a ilość wprowadzanych zmian była niewielka i bardzo szybko spadła [27].

5.2.2 Caffe2

Facebook, razem z NVidią, Qualcommem, Intelem, Amazonem, Microsoftem oraz z twórcą oryginalnego Caffe [28] również zainteresował się platformą programistyczną z Berkeley i stworzył Caffe2 [29], kładąc nacisk na jego optymalizację. W maju 2018 r. ta platforma programistyczna została wchłonięta przez PyTorch i nie są wprowadzane już żadne zmiany–od tego czasu pojawiła się tylko 1 kontrybucja usuwająca kod [30].

5.2.3 Torch

Torch to platforma programistyczna [31] napisana w języku skryptowym LuaJIT. Jest przeznaczony do zagadnień inteligencji maszynowej, zawiera wiele procedur do obliczeń na N-wymiarowych tablicach, cięcia, transponowania, indeksowania, algebry liniowej. Wspiera sztuczne sieci neuronowe, modele energetyczne i optymalizację numeryczną. Oficjalnie nie jest już aktywnie rozwijany [32].

5.2.4 PyTorch

Drugą najpopularniejszą platformą programistyczną, po TensoFlow, jest PyTorch [196] posiadający wsparcie min.: Facebooka, NVidii, Intela, Microsoftu. Nie bez powodu jest też na pierwszym miejscu (nie są one alfabetyczne ułożenie) w omawianych przez NVidię platformach programistycznych [33]. Stworzenie przez Googla własnej otwartej (na wzór Androida) platformy programistycznej i nowej jednostki obliczeniowej TPU

zostało potraktowane przez pozostałe firmy, posiadające wcześniej płatne, komercyjne i zamknięte rozwiązania, albo jako zaostrzenie trwającej już rywalizacji (np. z Microsoftem przez wprowadzenie Androida) albo wręcz jako wypowiedzenie wojny–zdecydowana większość obliczeń była wykonywana na GPU NVidii, czyniąc ją niemalże monopolistą w tym segmencie.

To podejście [34] ma pozwalać na gładkie przejście pomiędzy prototypowym, pozwalającym na łatwe wyszukiwania błędów, podejściem a w pełni grafowym, czyli szybko działającym. Wykorzystują wsparcie dla asynchronicznych operacji wynikające z Pythona i C++. Przy czym twórcy dość znacznie zaznaczają, że ich kody są w minimalnym stopniu platformą programistyczną a raczej są biblioteką. Całość jest pisana jako ścisła integracja z językiem Python–nie jest to oprawa dla monolitycznego oprogramowania napisanego w C++ (choć posiada do niego interfejs by umożliwić wykorzystanie stworzonych sieci w przemyśle). W związku z tym, można bez przeszkód i dodatkowych komplikacji korzystać z najpopularniejszych bibliotek Pythona i łatwo integrować projekt z innym oprogramowaniem. Co W drugiej wersji postanowiono w nim w dużej mierze zasymilować platformę programistyczną Cafee.

Stworzone modele z łatwością można eksportować do standardu ONNX. Twórcy [35] uważają, że ta biblioteka jest częściowo równoważnikiem biblioteki NumPy [36], potrafiącym wykorzystać kartę graficzną. Całość jest aktywnie rozwijana–w ostatnich latach znacznie wzrosła liczba wprowadzanych zmian. Google [37] chce ułatwić integrację TensorBoard z Pytorchem (nawiązał jakaś współpracę z Facebookiem) i dodać możliwość korzystania z jednostek TPU (ich układów). W między czasie powstała też nowa platforma programistyczna: FastAi–czyli próba stworzenia nakładki dla PyTorcha na wzór Kerasa. To rozwiązanie zostało wybrane pośrednio, poprzez korzystanie z platformy programistycznej FastAI.

5.2.5 FastAI

Projekt był wspierany przez Amazon Web Services, Salamander [38] i bazuje na PyTorch, skąd także otrzymuje wsparcie. Pierwotnie była to nazwa kursów bazujących na Kerasie i Tensorflow ale jesienią 2017 r. zdecydowano się przejść na Pytocha ponieważ większość z najlepszych rozwiązań w konkursie Kaggle korzystała właśnie z niej. Ma być pierwszą biblioteką

do uczenia głębokich sieci neuronowych dostarczającą prosty i jednolity interfejs do wszystkich powszechnie wykorzystywanych aplikacji w zagadnieniach wizji, tekstu, danych tabelarycznych, analizy/przewidywania szeregów czasowych (szumów, pogody) i filtrowania kolaboratywnego CF (tworzenie rekomendacji). Na dzień przeglądu to oprogramowanie było bardzo nowe i bardzo aktywnie rozwijane [39]–być może będzie to odpowiednik Kerasa dla PyTorcha. Właśnie to rozwiązanie wybrano. Na szczęście, pomimo niepokojących sygnałów (takich jak brak zmian, bądź ich bardzo niewielka liczba), projekt nadal jest rozwijany. Dużo zmian jest wprowadzanych na raz.

5.2.6 TensorFlow

Podczas wyboru, jak i pod koniec pisania rozprawy, najpopularniejszą platformą programistyczną był TensorFlow [40] od Googla. Jest to oprogramowanie wspierające obliczenia inteligencji maszynowej i głębokich sieci neuronowych a także inne dziedziny, które mogą korzystać z jego elastycznego rdzenia obliczeń numerycznych. Posiada on szereg rozbudowanych narzędzi diagnostycznych i ułatwiających uczenie. Pozwala na obliczenia z wykorzystaniem CPU, GPU i TPU. Ta platforma programistyczna domyślnie najpierw zapisuje sieć w postaci grafowej a następnie na nim pracuje. Później dodano (podczas przeglądu dostępnych rozwiązań było to nowością), tak jak w Chainerze czy Pytorchu (bezpośrednim konkurencie), możliwość wymuszenia natychmiastowego wykonania operacji bez budowy grafu. Część kodu pochodzi od YangqingJia [28], twórcy Caffe, gdyż pracował dla Googla (następnie dla Facebooka). Niektórzy uważają [41], że ta platforma programistyczna nie jest do końca przejrzysta i cześć kodu jest specjalnie komplikowana i nie można znaleźć kilku rzeczy takich jak: poszukiwanie współczynnika uczenia, współczynnik szybkości uczenia dyskryminującego, zamrażania znormalizowanych paczek danych, klasyfikacji NLP na dużych dokumentach, wykorzystanie kategorycznych i ciągłych kolumn w danych tabelarycznych czy dostęp do plików CSV. Całość jest bardzo aktywnie rozwijana [42]. Stworzono do niego również nakładkę–Keras, mającą na celu ułatwienie korzystania z tej platformy programistycznej, przy czym dodano kilka niestandardowych funkcji.

5.2.7 Keras

Ta platforma programistyczna to ponoć nakładka i przyjazny interfejs do TensorFlow (w przeszłości także dla CNTK i Theano [43] ale z uwagi na to, że nie są już tak rozwijane, to jest to raczej nieaktualne), stworzony przez pracownika Googla. Ostatnio zyskał jego uznanie i wsparcie. W trakcie badań rozpoczęto jego integrację z „główną" platformą programistyczną– TensorFlowem [44]. Dzięki modularnej budowie, elastyczności i przyjaznemu dla użytkownika podejściu ma pomóc w szybkim eksperymentowaniu i prototypowaniu. Może korzystać zarówno z CPU jak i GPU. Całość jest aktywnie rozwijana. Choć z początku Keras nie cieszył się zbyt dużą popularnością wśród osób wprowadzających zmiany, istnieją momenty, w których dodawana jest spora liczba nowego kodu [45]. Zdaje się, że wiele osób aktualnie z tego korzysta.

5.2.8 Inne platformy programistyczne

Oprócz wymienionych platform programistycznych istnieje więcej narzędzi przeznaczonego do uczenia sieci neuronowych, jak np. bardzo mocno, aktywnie rozwijany i wspierany przez chińskie Baidu [46] PaddlePaddle (niektórzy [47] twierdzą, że działa lepiej od TensorFlow–duża część komunikacji nad projektem odbywa się po chińsku [48]), napisany w Javie i scali Deeplearning4j [49, 50], czy najpopularniejszy kiedyś (ale płatny) Matlab [51, 52]. H2o [53] (ang. *Deep water*) stworzone przez 0xdata (obecnie H2o.ai) jest to jedna ze starszych platform programistycznych, VowpalWabbit [54], scikit-learn [197] (choć raczej nie dotycz uczenia sieci neuronowych tylko uczenia maszynowego), Spark [55] i wolno rozwijany bigDL Intela [56].

Część z platform po utracie zainteresowania zdecydowała się nawet na udostępnienie kodu źródłowego. Jednak z uwagi na ich bardzo niszowy charakter lub zakończony żywot w wyniku ich porzucenia przez autorów, zostaną one tylko wymienione bez zbędnego, dłuższego opisu.

Już nie rozwijane (albo w znikomym stopniu): CNTK [57, 58]–choć Microsoft nadal wspiera swoje rozwiązanie, to zdecydował się pomóc w rozwoju PyTorcha [59]. PlaidML zapoczątkowanego przez przejęte przez Intela Vertex.AI [60, 61, 62], Theano [63] (twórcy zapowiedzieli wspieranie i rozwijanie przedsięwzięcia ale jako silnik dla PyMC [64, 65]), Chainer wykorzystujący dynamiczne grafy obliczeniowe [66, 67]. Natomiast MXNet [68, 69, 70] zapisuje tworzone warstwy w języku C++, działa z Gluonem

[71] (miał stanowić konkurencję dla Kerasa, z którym również przez pewien czas działało MXNet) czy ONNX. DSSTNE specjalizował się w obliczeniach na rzadkich macierzach i wspierał tylko w pełni połączone warstwy [72]. Konkurencji platform programistycznych nie wytrzymała również ta należąca do NVIDIi: DIGITS [73, 74], Sonnet [75] (wysokopoziomowa nakładka na TensorFlow, stworzona przez Googla), bardzo wolno rozwijany Horovod Ubera [76], nierozwijany Neon [77], Deepdist [78], Deep Learning Pipelines [79], SparkNet [80], Leaf [81].

Na uwagę zasługują również platformy programistyczne automatycznie poszukujące strukturę sieci (AutoML). Najpopularniejszym z tej grupy jest AutoKeras [82, 152], choć nie korzysta z niego wiele osób i nie jest także szczególnie szybko rozwijany. Istnieją również wersje dla Tensorflowa [83] i Pytorcha [84] ale one nie są już rozwijane. Oprócz wymienionego oprogramowania można znaleźć jeszcze wiele innego, także nowego, co świadczy o tym, że jest to bardzo prężnie rozwijająca się dziedzina badań.

5.2.9 Wybrana platforma programistyczna

Do badań zdecydowano się posłużyć FastAi i PyTorchem. Podczas gdy TensorFlow uważany jest prawie że jako język sam w sobie, to PyTorch jest w bardzo dużym stopniu zbieżny z językiem Python i to w głównej mierze stanowiło o jego wyborze.

5.3 Zastosowanie sieci neuronowych do estymacji głębi

Estymacja głębi na podstawie danych z pojedynczej kamery (system monokularowy) w środowisku pomieszczeń zamkniętych był przedmiotem licznych prac badawczych ze względu na praktyczne zastosowania w robotyce, rozszerzonej rzeczywistości oraz rekonstrukcji $3D$. Konwencjonalne metody uzupełniania danych o głębi często opierały się na zasadach geometrycznych, takich jak struktura z ruchu (SfM) i stereo z wielu widoków, a także klasyczne techniki przetwarzania obrazów. Filtracja bilateralna [203] minimalizację, minimalizacja energii [111] oraz transformata Fouriera [94] były jednymi z pierwszych podejść do tego problemu, mając na celu wygładzenie map głębi przy jednoczesnym zachowaniu krawędzi. Mimo że były one skuteczne w kontrolowanych warunkach, ich efek-

tywność była ograniczona w przypadku dużych regionów oraz znacznej zmienności scen wewnętrznych, szczególnie w dynamicznych lub zatłoczonych środowiskach [93]. Wraz z rozwojem metod głębokiego uczenia pojawiły się bardziej zaawansowane techniki, oferujące rozwiązania radzące sobie z tymi trudnościami. Rozwiązania te wykorzystują do uczenia duże zbiory danych i konwolucyjne sieci neuronowe.

W tym nurcie badań Eigen i współautorzy [120] utorowali drogę do wykorzystania konwolucyjnych sieci neuronowych (CNN) do estymacji głębi na podstawie pojedynczego zdjęcia, podczas gdy Zhang i Funkhouser [246] zaproponowali dwuetapowe podejście, w którym najpierw przewidywane są lokalne właściwości powierzchni, takie jak granice przesłonięć czy normalne do powierzchni, a następnie odtwarzana jest mapa głębi na podstawie globalnej optymalizacji. Z kolei Yu i współautorzy [245] pokazali, że zastosowanie w sieci neuronowej mechanizmu uwagi kontekstowej poprawia zachowanie krawędzi w mapach głębi.

Obszerny przegląd współczesnych metod estymacji głębi sceny został zawarty w pracach [158, 175], gdzie podkreślono znaczenie dużych, dobrze opisanych zestawów danych w procesie uczenia sieci neuronowych. W pracy [132] (w której wykorzystano sieć Monodepth) zaproponowano podejście wykorzystujące obrazy stereo do uczenia bez nadzoru, przy czym w fazie inferencji wymagane są jedynie dane RGB. W późniejszych pracach dodano adaptacyjny operator splotu do progresywnego uzupełniania map głębi. Xian i współautorzy [242] zaproponowali metodę łączącą surowe mapy głębi z danymi RGB, podczas gdy Senushkin i inni [213] wprowadzili przestrzennie adaptacyjne bloki denormalizacyjne w celu poradzenia sobie z różnicami statystycznymi między pozyskanymi danymi, a obszarami braku danych („dziurami" w obrazach głębi). W nowszej pracy [232] zaproponowano wielokanałową, progresywnie atencyjną sieć do stopniowego odtwarzania map głębi o wysokiej rozdzielczości.

Dalsze ulepszenia metod neuronowej estymacji głębi sceny okazały się możliwe poprzez integrację informacji semantycznej oraz wykorzystanie modeli już wytrenowanych do innych zadań (*transfer learning*). W pracy [241] podjęto się rozwiązania problemu generalizacji estymacji głębi w złożonych scenach, proponując podejście, które generuje mapy głębi niezależne od metryki (wymiarów sceny i obiektów). Połączono to podejście z gałęzią sieci neuronowej uczącą się metryki na podstawie spójności pomiędzy reprojekcją uzyskaną z lewego i prawego obrazu. Wymaga to znajomości pozycji kamery, co osiągnięto dzięki wykorzystaniu sieci neuronowej

PoseNet do symulacji widzenia stereowizyjnego w sekwencji obrazów z pojedynczej kamery, co znacznie komplikuje proces uczenia.

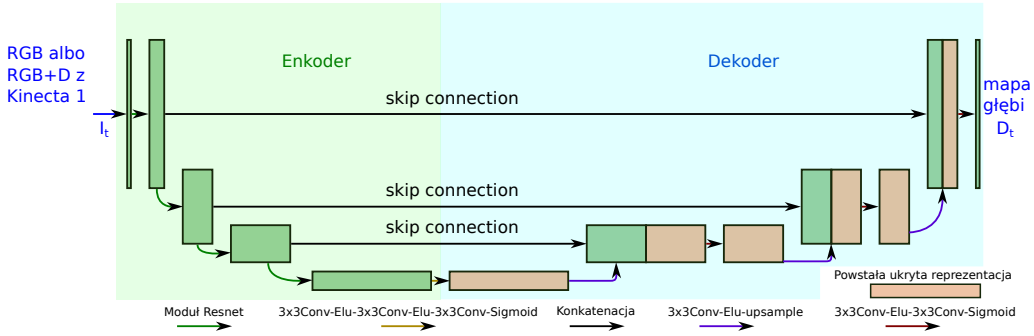
Po wstępnym przeglądzie dostępnych rozwiązań, które obejmowały różne architektury sieci neuronowych, do zastosowania w niniejszej pracy wybrano architekturę Monodepth, ponieważ najlepiej odpowiadała zadaniu odzyskiwania map głębi z danych RGB-D. Kluczowym czynnikiem przemawiającym za wyborem Monodepth była jego zdolność do estymacji głębi w sposób samonadzorowany, co eliminuje potrzebę korzystania z rozległych, oznaczonych zbiorów danych o głębi, które często są trudne do uzyskania. Architektura ta oferuje również zaawansowane mechanizmy, takie jak minimalizacja błędu reprojekcji, co pozwala na lepsze radzenie sobie z problemem przesłonięć oraz zapewnia wyższą dokładność estymacji głębi. Ponadto, Monodepth jest efektywna pod względem przetwarzania i zużycia pamięci, co czyni ją odpowiednią do praktycznego zastosowania w niedrogich, mobilnych robotach operujących w dynamicznych i nieuporządkowanych środowiskach. Dzięki tym cechom Monodepth uznano za odpowiedni wybór do integracji estymowanych informacji o głębi z systemem odometrii wizyjnej.

5.3.1 Monodepth

Monodepth to otwartoźródłowy model sieci neuronowej [180], który nadaje się do ulepszenia systemu odometrii wizyjnej poprzez uzupełnienie danych głębi. Ze względu na brak bezpośrednich danych głębi D w oryginalnym podejściu Monodepth, konieczne było wykorzystanie dodatkowej sieci obliczającej transformacje pomiędzy kolejnymi ujęciami. Dzięki temu możliwa była triangulacja i obliczenie głębi dla odpowiadających sobie pikseli po uwzględnieniu, czy na jednym z obrazów nie występują obiekty przysłaniające. W związku z bezpośrednim dostępem do danych głębi D z sensorów Kinect v1 i Kinect v2, np. w zbiorze danych PUTKK [166], oryginalny proces uczenia, który wykorzystywał dane RGB, można uprościć, stosując mapy głębi pochodzące z sensora Kinect v2 jako *ground truth*.

Sieć Monodepth [132] składa się z enkodera w postaci sieci ResNet [139] (w wariantach zawierających 18 i 50 warstw), wstępnie wytrenowanej na zbiorze danych ImageNet [206]. Enkoder stopniowo zmniejsza rozdzielczość wejściowego obrazu, aby uchwycić cechy wysokiego poziomu. Następnie dekoder głębi odtwarza mapy głębi o różnych skalach, z dodat-

kowymi połączeniami (*skip connections*) między enkoderem a dekoderem w miejscach zmiany skali, co pozwala na zachowanie detali z wyższych rozdzielczości, podobnie jak w sieciach U-net (rys. 5.3).



Rysunek 5.3: Architektura U-Net konwolucyjnej sieci neuronowej Monodepth

Dekoder zastosowany w Monodepth, z nieweilkimi modyfikacjami, został zainspirowany pracą [176]. Składa się on z pięciu bloków, w których występują:

1. Konwolucja z jądrem o wymiarze 3 z aktywacją ELU, z wcześniejszym dodaniem elementu tensora na krawędziach (*reflection padding*), co prowadzi do zmniejszenia liczby kanałów.

2. Dwukrotne zwiększenie rozdzielczości poprzez interpolację najbliższego sąsiada.

3. Konkatenacja z odpowiednią warstwą ResNet w tej samej skali, tj. tensorem o tych samych wymiarach.

4. Konwolucja z jądrem o wymiarze 3 z aktywacją sigmoidalną, bez zmiany liczby kanałów.

W czterech ostatnich blokach, poza najgłębszym o największej liczbie kanałów, wyjścia są generowane za pomocą warstwy konwolucyjnej z jądrem o wymiarze 3, aktywowanej funkcją sigmoidalną, co pozwala na uzyskanie wyników o niższej rozdzielczości.

Wstępne badania przeprowadzone na potrzeby niniejszej rozprawy wykorzystywały oryginalną architekturę przyjmującą na wejście obraz RGB,

co umożliwiło zastosowanie transfer learningu w celu przeniesienia już wyuczonych wag.

Uczenie sieci Monodepth było projektowane z myślą o działaniu również na zewnątrz budynków, przy czym dane były zbierane za pomocą dwóch kamer, co oznaczało brak bezpośredniego dostępu do map głębi. W tych badaniach sieć była uczona zarówno z wykorzystaniem obrazu stereowizyjnego, jak i z pojedynczej kamery w wariancie samo nadzorowanym, eliminując konieczność posiadania dużego zbioru danych z etykietami głębi. Wyznaczenie mapy głębi potraktowano jako zadanie pośrednie w generowaniu nowego ujęcia, które widziałaby kamera nieco obrócona i przesunięta. Transformacje pomiędzy dwoma ujęciami były obliczane za pomocą innej, równolegle uczonej sieci PoseNet. Taki sposób postawienia problemu ma jednak wady, ponieważ tracimy informację o skali: istnieje wiele nieprawidłowych wartości głębi przypisanych pikselowi, które mogą poprawnie wygenerować nowe ujęcie. Dodatkowo, wyznaczenie położenia widzianej na obrazie $2D$ linii w przestrzeni stanowi problem [216, 85], ponieważ wiele różnych ułożeń linii w przestrzeni $3D$ może wyglądać identycznie na obrazie.

Na ostateczny błąd składa się fotometryczny błąd reprojekcji oraz błąd gładkości, który uwzględnia krawędzie. W przypadku korzystania z kilku ujęć w celu estymacji głębi, konieczne było poradzenie sobie z pikselami, które nie mieszczą się w kadrze docelowego obrazu, jak i z tymi, które są widoczne na jednych zdjęciach, a na innych nie. W takich przypadkach przyjmowano, że są one przysłonięte, a do obliczeń wybierano najbliższy punkt zamiast wartości średniej. Dodatkowo, maskowane były piksele, które nie zmieniały się na dwóch kolejnych ujęciach–uznawano, że jest to obiekt poruszający się z taką samą prędkością co kamera, kamera nie porusza się, albo jest to region ubogi w tekstury. Ostateczny błąd stanowiła średnia z błędów policzonych dla wszystkich pikseli, rozdzielczości i partii zdjęć, przy czym rozdzielczość obrazów o rozdzielczości niższej niż oryginalna była najpierw skalowana do oryginalnej i do liczenia błędu wykorzystywano obraz RGB podany na wejście sieci.

Rozdział 6

Badania eksperymentalne

6.1 Metodologia uczenia sieci

W trakcie prac bardzo pomocne okazała się platforma programistyczna FastAi. W zamian za konieczność kosmetycznego dostosowania niektórych architektur (również badanej Monodepth) udostępnia ona szereg narzędzi ułatwiających proces uczenia, poszerzania zbioru uczącego i monitorowania postępów.

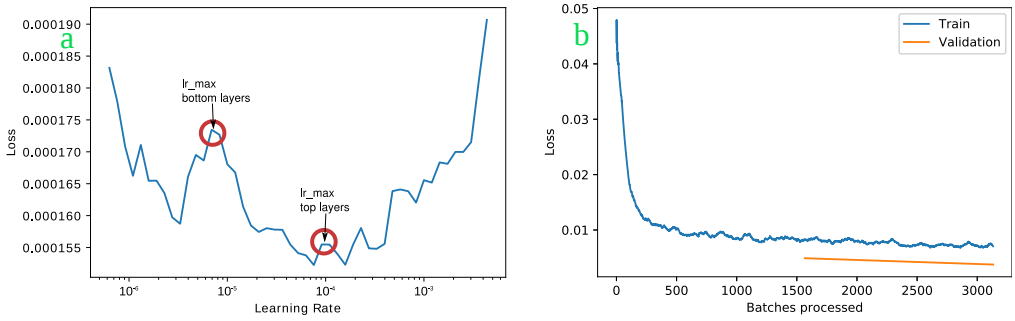
Pierwsze warstwy sieci w podobnych zadaniach bardzo niewiele się zmieniają. W związku z tym, by zapobiec ich dużym zmianom wynikającym z dużych gradientów powstałych przy zmianie zadania, na wstępnym etapie douczania sieci zamraża się wszystkie grupy warstwy sieci poza ostatnią. To w głównej mierze warstwy w niej zawarte są odpowiedzialne za klasyfikację i to je należy najpierw douczyć. Odblokowujemy możliwość zmiany ich wag, dopiero gdy błąd w zestawie walidacyjnym przestaje się znacząco zmniejszać. Nawet wtedy nie wszystkie grupy warstwy są uczone równo. Dla pierwszych grup warstw (na spodzie modelu) współczynnik uczenia jest od rzędu do dwóch niższy od końcowych grup warstw, zaś waga uczenia dla warstw pomiędzy jest modyfikowana zgodnie z postępem geometrycznym–warstwy sieci uczymy dyskrymitywnie.

W badaniach dokonano ręcznego podziału warstw sieci Monodepth na grupy, tak że każda grupa zawierała jedną warstwę. Na wstępie douczano ostatnie warstwy dekodera (6 to 13) z tą samą wagą uczącą, a następnie wszystkie według opisanej wyżej metodologii. Z uwagi na architekturę sieci, jak i możliwości obliczeniowe wykorzystanego komputera, w bada-

niach przyjęto, że karta graficzna będzie dokonywać obliczeń dla 6 pakietów. W związku z zaleceniami [220] starano się adekwatnie dobrać wagę uczenia (powinna być mniejsza).

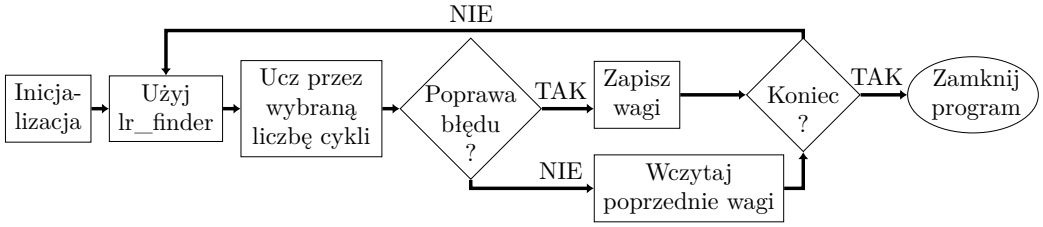
6.1.1 Uczenie w cyklu

Biblioteka FastAi [145, 146] implementuje politykę uczenia w cyklu (zaproponowaną w [220, 221]). Oryginalnie dla każdej grupy warstw zwiększana jest waga ucząca od $lr_{\text{max}}/\text{div}_{\text{factor}}$ (domyślnie $\text{div}_{\text{factor}}$ jest równy 25) do lr_{max}, po czym wagi są zmniejszane w ten sam sposób. Pod koniec cyklu uczenia wagi są dodatkowo zmniejszane o kilka rzędów wielkości, tj. od $lr_{\text{max}}/\text{div}_{\text{factor}}$ do np. $lr_{\text{max}}/(\text{div}_{\text{factor}} \cdot 100)$. Powinno się wybrać wartość wagi uczącej przed minimum dla górnych warstw ($lr_{max\_top\_layers}$) i co najmniej o jeden rząd mniejszą dla spodnich ($lr_{max\_bottom\_layers}$). Rys. 6.1 podsumowuje proces poszukiwania wagi uczącej (lewa część obrazu) i wyniki douczania zestawem PUTKK (prawy część obrazu).



Rysunek 6.1: Poszukiwanie najlepszego współczynnika uczenia a) z biblioteką FastAi, i b) wyniki uczenia

W kilku pierwszych sesjach uczenia korzystano z domyślnego w bibliotece podziału na fazy zwiększania (domyślnie 30% wszystkich iteracji) i zmniejszania wagi uczenia. W późniejszych zaś zmieniono na równy podział 50%. Wagi sieci są zapisywane co kilka cykli i wykorzystywana jest procedura poszukiwania wagi uczącej LR Finder (FastAi), by określić proces dalszego douczania. Rys. 6.2 przedstawia ten schemat douczania.



Rysunek 6.2: Schemat douczania sieci Monodepth

6.1.2 Funkcja błędu

Obecnie dosyć popularne jest stosowanie indeksu strukturalnego błędu SSIM, który jest już zaimplementowany w większości nowoczesnych frameworków. Do map głębi nie muszą się odnosić wszystkie zasady tyczące się standardowych zdjęć RGB, a głównym naszym celem nie jest otrzymanie ładnych dla człowieka obrazów, ale takich które pozwolą na osiągnięcie lepszej trajektorii. Jednym z założeń konstrukcyjnych miary SSIM jest by była ona niezależna od średniej luminacji czy kontrastu, czyli immanentnych cech składowych obrazu (mapy głębi). Jasność pikseli oznacza odległość obiektu od kamery, a kontrast pozwala dokładnie określić położenie obiektu i od jego jakości zależy jakość odtwarzanej trajektorii.

Z tych względów, jak i z uwagi na prostotę funkcji MSE, dostępność danych odniesienia i fakt, że badania opisane w krytyce SSIM, wskazują wady tego indeksu, w badaniach zdecydowano się na wykorzystanie błędu średniokwadratowego MSE dla pikseli pomiędzy obrazem wejściowym a wyjściowym. Nie wybrano na funkcję błędu miary średniej kwadratowej błędów RMSE, ponieważ cechuje się mniejszymi, od miary MSE, wartościami dla małych, bliskich zeru błędów. Wydłużyło by proces douczania. Miarę D_{MSE} obliczamy z niezerowych różnic pomiędzy inferencją sieci a syntetyzowanymi danymi odniesienia (6.1):

$$D_{\text{MSE}} = \frac{1}{n} \sum_{uv} (I_{uv} - G_{uv})^2, \tag{6.1}$$

gdzie n to liczba niezerowych pikseli w skonwertowanym obrazie Kinecta v2, I_{uv} reprezentuje piksel w obrazie z inferencji o współrzędnych uv, a G_{uv} jest odpowiadającą temu pikselowi częścią z obrazu odniesienia.

Metrykę MSE zmodyfikowano w taki sposób, by pod uwagę brała tylko te piksele, dla których są dane w zestawie danych wzorcowych. Oznacza

to pomijanie pikseli o wartości równej 0. Ponieważ platforma programistyczna FastAi standardowo normalizuje wczytane dane obrazu poprzez dzielenie wszystkich pikseli przez 255, a dane głębi są zapisane w formacie 16-bitowym (a nie standardowym 8-bitowym), to należy wykonać dla nich dodatkowe dzielenie przez 255.

6.2 Przygotowanie zestawów danych

Niniejszy podrozdział opisuje jakie dane podawane są na wejście sieci neuronowej oraz jakie są traktowane jako dane odniesienia. Obydwa rodzaje danych są normalizowane przy wczytywaniu ich przez sieć.

6.2.1 Dane wejściowe

Wejściem sieci jest tensor składający się z 4 kanałów. Pierwsze 3 odpowiadają barwom RGB, a ostatni mapie głębi widzianej przez Kinect v1. Ze względu na brak obsługi przez język Python obrazów png zapisanych w formacie 16-bitowym, obrazy głębi skonwertowano do obsługiwanego formatu tiff. Zmieniono jednocześnie liczbę jednostek przypadających na $1[\mathrm{m}]$, tak by była taka sama jak w danych odniesienia. Powody tej zmiany opisane są w części dotyczącej zapisu obrazów, w podrozdziale dotyczącym danych odniesienia 6.2.2. Otrzymany w ten sposób zbiór danych wejściowych i odniesienia jest losowo, ale ze stałym ziarnem by uzyskane wyniki były powtarzalne, dzielony na stanowiący 90% zbiór uczący i 10% walidacyjny.

6.2.2 Dane odniesienia

W trakcie badań pojawił się nowszy model (Kinect v2) wykorzystywanego sensora. W związku z tym postanowiono go wykorzystać do poprawy wyników, jakie da się osiągnąć wykorzystując gorszy sensor. By ten cel osiągnąć przetworzono widziane przez kamerę Kinect v2 mapy głębi, tak by przypominały te, który mógłby zobaczyć Kinect v1. Następnie z ich pomocą uzupełniono braki w danych w mapach głębi z Kinecta v1.

Zmiana rozdzielczości i punktu widzenia kamery

W tym celu punkty widziane przez Kinecta v2 należy zamienić na chmurę punktów przy wykorzystaniu jej wewnętrznych parametrów kalibracyjnych **K**: ogniskowych (f_x, f_y) i współrzędnych położenia środkowego piksela (c_x, c_y), współczynnika głębi $factor1\_depth$–czyli ile jednostek oznacza 1m, a także współczynników zniekształceń radialnych k_i oraz tangensowych (decentrycznych) p_i. W tym celu wykorzystano funkcję, przybliżającą rozwiązanie w ciągu, domyślnie 5 iteracji `undistortPoints` z biblioteki OpenCV, co podsumowuje algorytm 4: gdzie RR to macierz obrotu po

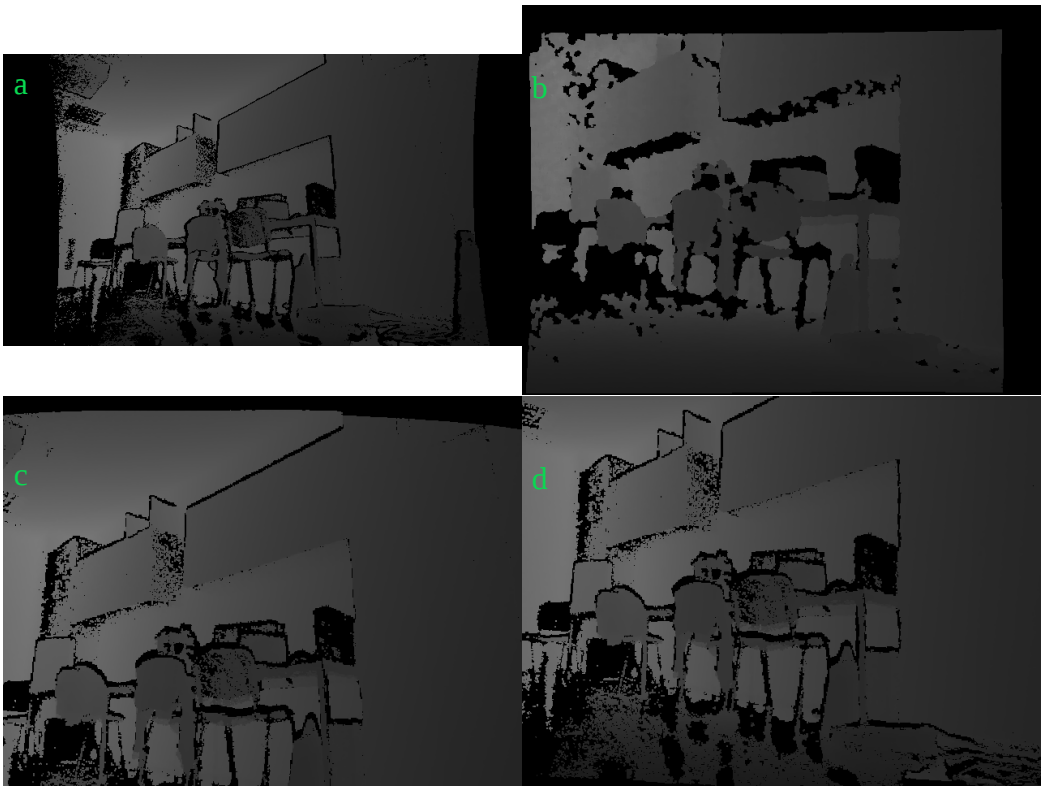
Algorytm 4: Korygowanie współrzędnych x,y

$x0 = x = (x - cx)/fx$
$y0 = y = (y - cy)/fy$
for $i = 0; i < 5; ++i$ **do**
  $r^2 = x'^2 + y'^2$
  $iR = (1 + k_4r^2 + k_5r^4 + k_6r^6)/(1 + k_1r^2 + k_2r^4 + k_3r^6)$
  $deltaX = 2p_1 * xy + p_2(r^2 + 2x^2)$
  $deltay = 2p_2 * xy + p_1(r^2 + 2y^2)$
  $x = (x0 - deltaX) * iR$
  $y = (y0 - deltaY) * iR$
  $i = i + 1$
$xx = RR[0][0] * x + RR[0][1] * y + RR[0][2]$
$yy = RR[1][0] * x + RR[1][1] * y + RR[1][2]$
$ww = 1/(RR[2][0] * x + RR[2][1] * y + RR[2][2])$
$x = xx * ww\ y = yy * ww$

rektyfikacji w przypadku systemu stereowizyjnego–w naszym przypadku jednostkowa, $x0$ i $y0$ to położenie pikseli przed korekcją, x i y to ich położenie po korekcji, c_x, c_y, f_x, f_y to parametry wewnętrzne kamery a k_i i p_i to parametry dystorsji.

Następnie te punkty są obracane i przesuwane zgodnie z macierzami roto-transacji Robot-Kinect2 $E_R^{K_2}$ i odwróconą Robot-Kinect1 $E_R^{K_1{}^{-1}}$, danymi w [166]. Niestety podane parametry zewnętrzne kamer odbiegały nieco od rzeczywistych przekształceń–z wizyjnego porównania obrazów wynikało, że jedna z kamer po kalibracji zmieniła kąty nachylenia. Fakt istnienia różnic pomiarowych, różnych modeli kamer oraz sposobów ich

kalibracji i radzenia sobie z różnego rodzajami zniekształceniami, błędami w zapisie danych widzianej przez kamerę rzeczywistości, oznacza, że nie ma najlepszego sposobu określenia błędu dopasowania dwóch obrazów map głębi. Dlatego zdecydowano się na heurystyczne poprawienie macierzy roto-translacji. Mogło to być spowodowane np. zdjęciem z niej znacznika wizyjnego, wykorzystywanego do estymacji jej pozycji przez zewnętrzny system kamer motion–capture podczas kalibracji, co mogło spowodować odciążenie stelaża trzymającego kamerę a w efekcie jego podniesienie i przesunięcie. Problem ten został przedstawiony na rysunku 6.3.



Rysunek 6.3: Obrazy map głębi widzianych przez Kamery w sekwencji *putkk _Dataset_1_Kin_1*. Dla lepszej prezentacji wartości a) i b) zwiększono 5-krotnie, by obrazy nie były zbyt ciemne i posiadały równą intensywność. a) Kinect v2, b) Kinect v1, c) Po transformacjach bez poprawek, d) Po transformacjach z uwzględnieniem poprawek

Dalej uwzględniane są zniekształcenia radialne (6.2), które powodują soczewki Kinecta v1, by móc rzutować punkty na obraz, który widziałby

Kinect v1. Robione jest to w sposób odwrotny do przekształcania ich na chmurę punktów. Może się zdarzyć, że część punktów znajdzie się poza widzialnym dla starszego sensora obszarem, ponieważ Kinect v2 posiada większe kąty widzenia–te punkty nie są dalej rozpatrywane. W takim przypadku punkty te przepadają. Z uwagi na to, że nowsza kamera ma większą wertykalną i horyzontalną rozdzielczość, to na 1 piksel widziany przez Kinecta v1 może przypaść kilka zaobserwowanych przez Kinecta v2. Punkt, który znalazłby się na krawędzi czy narożniku piskela, zostaje zaliczony do obu pikseli, bo nie da się ustalić, do którego należy. Każdemu takiemu, wchodzącemu w nowego skład piksela punktowi, przypisuje się wagę równą jego bezwzględnej odległości od środka piksela–na tej zasadzie działa interpolacja sterownika iai_kinect2 [240], proces ten nazywany jest rejestracją. Z kolei wagę w wykorzystywanej w Pytorchu filtracji bilinearnej stanowi powierzchnia rozpinana przez prostokąt o bokach o długościach składających się na odległość od środka piksela. Ostateczna odległość jaką piksel reprezentuje obliczana jest jako średnia ważona punktów wchodzących w skład tego piksela.

$$
\begin{aligned}
x' &= \frac{x}{z} \\
y' &= \frac{y}{z} \\
r^2 &= x'^2 + y'^2 \\
R &= \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \\
x'' &= x' R + 2p_1 * x' y' + p_2(r^2 + 2x'^2) \\
y'' &= y' R + 2p_2 * x' y' + p_1(r^2 + 2y'^2) \\
u &= f_x x'' + c_x \\
v &= f_y x'' + c_y
\end{aligned}
\tag{6.2}
$$

Interpolacja brakujących danych i zapis obrazów

Niestety, ale nawet lepszy sensor, jakim jest Kinect v2, nie pozwala na uzyskanie kompletnych map głębi. Nadal zawiera pozbawione informacji piksele i obszary.

Posiadając w ten sposób uzupełnioną i odpowiednio transformowaną mapę głębi z nowszej wersji sensora, można ją wykorzystać do uzupełnienia, poprzez bezpośrednie zastąpienie, braków w mapach widzianych przez Kinecta v1.

Przed zapisem całość skalowana jest do 5000 jednostek na 1m, a nie jak oryginalnie w zestawie danych 1000. Wynika to z tego, że dane są zapisywane jako 16-bitowy `unsigned int`, co przy tym skalowaniu pozwala na zapisanie odległości do 65m. Microsoft podaje jednak, że maksymalna odległość dla której kamera Kinect v2 jest w stanie podać dobry pomiar to $4,5$m (choć inni użytkowcy twierdzą, że może widzieć nawet do 8m). Przy wykorzystaniu 5000 jednostek na $1[\mathrm{m}]$ możemy zapisać odległość do 13m. Taka wartość jest również wybrana w zestawie danych *TUM RGB-D* [225]. Oznacza to, że jeśli za końcową wartość zostanie obrana średnia, a nie odległość widziana do najbliższego obiektu, to można uzyskać dokładniejszą mapę głębi w wiarygodnym zasięgu pomiaru sensora.

Tak skonstruowane mapy są wykorzystywane jako etykiety odniesienia (ang. *ground truth*) w procesie uczenia.

Dobrą praktyką jest normalizowanie danych wejściowych, bo sieci działają najlepiej w obszarze największej aktywności swoich funkcji aktywacji [86].

6.3 Augmentacja danych

Podczas wstępnych badań na implementacji sieci Monodepth douczano sieć korzystając z zestawu PUTKK. W tym procesie skorzystano z metod poszerzania danych–data augmentation, by zwiększyć zbiór uczący. Okazało się, że następujące transformacje pogarszały jakość estymowanej mapy głębi, więc na dalszych etapach z nich zrezygnowano:

- Tylko w obrazie wejściowym RGB zmieniano kontrasty zdjęć, dodawano wariancję do kolorów, wycinano ich fragmenty.
- W obrazie wejściowym RGB, jak i uczącym wycinano w losowych miejscach kilka fragmentów.

Pozostawiono jedynie horyzontalne i wertykalne odbicie oraz obrót o $90°$ danych wejściowych i uczących.

6.4 Wykorzystane publicznie dostępne zestawy danych

W celu zapewnienia możliwości sprawdzenia badań wykorzystano publicznie dostępne zestawy danych, takie jak TUM RGB-D [225], PUTKK [166], Messor II [98, 87].

6.4.1 TUM RGB-D

Dane w tym zestawie były pobierane z trzymanej w ręce kamery Kinect v1 (ruch był gładki i łagodny) oraz z wykorzystaniem robota mobilnego. Dane odniesienia zbierano za pomocą systemu wielokamerowego ale czasy pobierania danych nie były wymuszone, ani zsynchronizowane co spowodowało potrzebę interpolacji danych. Znaczniki czasowe danych odniesienia oraz tych zebranych przez kamerę nie pasują do siebie idealnie. Zapisane są zarówno w tradycyjnym formacie, jak i w formacie „rosbag".

W eksperymentach korzystano z sekwencji, w których kamera była trzymana w ręce:

- *fr1_desk* składającej się z 595 klatek
- *fr1_room* 1360 klatek
- *fr3_long_office_household* 2585 klatek

6.4.2 PUTKK

Ten zestaw danych został zebrany za pomocą sensorów Kinect v1 i v2 przytwierdzonych do poruszającego się po laboratorium robota kołowego. Poza różną rozdzielczością oraz innymi polami widzenia, te kamery różnią się sposobem zbierania informacji o głębi. Starsza wersja wykorzystywała specjalnie do tego celu stworzony wzór, co było przyczyną znaczących obszarach bez danych o odległość, w szczególności blisko krawędzi obiektów. Nowsza działa na zasadzie czasu przelotu ToF, dostarczając spójniejszą mapę głębi. Zarejestrowany zbiór ośmiu trajektorii obejmuje znaczną część pomieszczenia i zawiera zamknięte pętle. Robot wraca w pobliże miejsca startu.

Zestaw danych odniesienia otrzymano z wysokiej rozdzielczości, pięciokamerowego (wykorzystano kamery Basler acA1600 z soczewkami cechującymi się niewielkimi zniekształceniami), systemu wizyjnego PUT Ground Truth (PUT GT) [211]. Wymagało to zamontowania na śledzonym obiekcie (robocie) pasywnego znacznika w postaci dużej, kilkupolowej szachownicy i dokonania odpowiedniej kalibracji [209] (tak by co najmniej dwie kamery go widziały), by móc wyznaczyć przekształcenia układów współrzędnych pomiędzy układami kamer a znacznikiem robota i przymocowanych do niego kamer. Czasz pobierania danych przez robota jak i kamer PUT GT zsynchronizowano, tak by nie było potrzeby wykonywania interpolacji.

W eksperymentach korzystano z sekwencji:

- *putkk_Dataset_1_Kin_1* składającej się z 1540 klatek
- *putkk_Dataset_2_Kin_1* składającej się z 2564 klatek
- *putkk_Dataset_3_Kin_1* składającej się z 2755 klatek
- *putkk_Dataset_4_Kin_1* składającej się z 2855 klatek
- *putkk_Dataset_5_Kin_1* składającej się z 411 klatek

6.4.3 Messor II

Ten zestaw danych [98] został zebrany za pomocą sensora Asus Xtion PRO Live przytwierdzonego do poruszającego się po laboratorium sześcionożnego robota kroczącego. Taki typ chodu jest dyskretny i powoduje problemy wynikające z oscylacji i wibracji, wprowadzając duże rozmycia obrazów zbieranych podczas ruchu. Messor II z sensorem Asus Xtion PRO Live poruszał się po makiecie terenu (o wymiarach 2×2m) o niewielkich nierównościach i pokrytej piaskiem, zakreślając trajektorie przypominające kwadrat o długości ok. 2m. Robot wykorzystywał różne chody i prędkości (zarówno translacyjne jak i rotacyjne). Do określenia transformacji układu współrzędnych między robotem a kamerą wykorzystano kalibrację [101]. Dane odniesienia zostały zebrane za pomocą systemu wizyjnego PUT Ground Truth (PUT GT) [211], który wykorzystywał tylko centralną kamerę– robot Messor II poruszał się po małym obszarze. Czasy pobierania klatek

poprzez ten system jak i robota zostały zsynchronizowane i były pobierane z częstotliwością 15 Hz–limit klatek na sekundę, jaką mogą dostarczyć kamery systemu PUT GT (Basler acA1600).

W pierwszych dwóch sekwencjach Messor II wykorzystywał domyślny chód typu tripod–najszybszy statycznie stabilny dlatego wielonożnego robota. Był on zdalnie sterowany przez człowieka dżojstikiem, więc prędkość jego ruchu się różniła w trakcie pokonywania drogi. W pierwszej sekwencji prędkość wynosiła 95% jego maksymalnej prędkości a w drugiej 45%, więc wibracje tułowia robota były mniejsze. W trzeciej sekwencji znacznie spowolniono ruchy robota i zmieniając jego typ na pełzający (ang. *crawl*). Robot poruszał jednocześnie tylko jedną kończyną. Wszystkie obrazy były zbierane, gdy robot był podparty co najmniej pięcioma odnóżami. Zredukowało to znacząco wibracje (powodowane przez chwilowe przeciążenie serwomotorów) i rozmycia obrazu. W eksperymentach skorzystano z następujących sekwencji (o niżej podanej liczbie klatek i średnich prędkościach):

- *messor2_1* składającej się z 589 klatek. Prędkość średnia robota wynosiła $0,15$ [m]/[s].
- *messor2_2* składającej się z 1505 klatek. Prędkość średnia robota wynosiła $0,09$ [m]/[s].
- *messor2_3* składającej się z 1219 klatek. Prędkość średnia robota wynosiła $0,05$ [m]/[s].

6.5 Wykorzystane miary błędów

Ponieważ kwintesencją rozprawy jest oszacowywanie przebytej przez kamerę trajektorii, to niezbędne jest korzystanie z miar błędów wiążącymi się z jej jakością. Dlatego jako miary dopasowania wykorzystano bezwzględny błąd trajektorii ATE (ang. *Absolute Trajectory Error*) oraz względny błąd pozycji RPE (ang. *Relative Pose Error*).

Są to miary jakości zaproponowane w [225] (skrypty można pobrać np. z [88]) i powszechnie stosowane w robotyce do ewaluacji systemów SLAM oraz VO. Do oceny wykorzystywano głównie pierwiastek błędu średniokwadratowego RMSE (ang. *Root Mean Squar Error*), do której obliczenia wykorzystuje się wszystkie odpowiednie składowe ATE oraz RPE.

Zwyczajowo przyjęło się stosowanie miary ATE do oceny systemów SLAM, a względnych błędów pozycji RPE do oceny VO. Jednakże w niniejszej rozprawie do oceny systemu odometrii wizyjnej również stosujemy metrykę ATE, ponieważ mamy na uwadze to, że VO może być wykorzystana w systemie SLAM (który jest w tej metryce oceniany). W dodatku nasze podejście nie jest w stanie zareagować na zmiany postrzegania, odbioru sceny jakie mogą nastąpić wzdłuż trajektorii (poza próbą online). W związku z tym miara RPE jest dla nas nieco mniej użyteczna, bo określa względny translacyjny albo rotacyjny błąd pomiędzy kolejnymi klatkami RGB-D.

Dysponując trajektorią odniesienia (ang. *ground truth*), zbieraną przez zewnętrzny system, $\mathbf{T}^{\text{gt}} = \{\mathbf{T}_1^{\text{gt}}, \mathbf{T}_2^{\text{gt}}, \ldots, \mathbf{T}_k^{\text{gt}}\} \in SE(3)$, oraz trajektorią estymowaną $\mathbf{T}(\theta) = \{\mathbf{T}_1(\theta), \mathbf{T}_2(\theta), \ldots, \mathbf{T}_k(\theta) \in SE(3)\}$, gdzie $\mathbf{T}_i(\theta)$ i $\mathbf{T}_i^{\text{gt}}$ są pozycjami sensora w przestrzeni trójwymiarowej i-tego położenia danymi w postaci macierzy jednorodnych o wymiarach 4×4, a k jest liczbą punktów trajektorii można zdefiniować miary ATE oraz RPE. Zakłada się, że czasy zbierania obydwóch trajektorii zostały zsynchronizowane, albo odpowiednio dopasowane (poprzez znaczniki czasu).

W stworzonym oprogramowaniu wykorzystano narzędzia udostępnione przez twórców benchmarku TUM RGB-D [225] i są one wywoływane do obliczeń. Do obliczeń błędu RPE stworzono reimplementację wykorzystywanej miary w języku `C/C++`, by nie było potrzeby wywoływania zewnętrznego skryptu.

6.5.1 Bezwzględny błąd trajektorii ATE

Miara bezwzględnego błędu trajektorii $\mathbf{E}_i^{\text{ATE}}$ ATE (ang. *Absolute Trajectory Error*) określa euklidesową odległość pomiędzy i-tą pozycją sensora wzdłuż trajektorii $\mathbf{T}_i$, a dokładną trajektorią odniesienia $\mathbf{T}_i^{\text{gt}}$ (6.3):

$$\mathbf{E}_i^{\text{ATE}} = \left(\mathbf{T}_i^{\text{gt}}\right)^{-1} \mathbf{T}_i(\theta). \tag{6.3}$$

Jednostką miary tego odchylenia jest zawsze $1[\text{m}]$. Wyznaczanie miary ATE [225, 164] odbywa się po uprzednim obrocie i przesunięciu szacowanej trajektorii tak, aby najlepiej pokrywała się z trajektorią odniesienia

(6.4):

$$\underset{\theta}{\operatorname{argmin}} F_{\mathrm{ATE}} = \sum_{i=1}^{k} \left(\mathbf{T}_i^{\mathrm{gt}}\right)^{-1} \mathbf{T}_i(\theta). \tag{6.4}$$

6.5.2 Względny błąd pozycji RPE

Miara względnego błędu pozycji RPE (ang. *Relative Pose Error*) odnosi się do lokalnego dryftu trajektorii. Oblicza ona względną różnicę w transformacji (dlatego występuje cześć rotacyjna i translacyjna), jakiej podlegałby sensor w i-tej pozycji, podążając oszacowaną trajektorią przez założony czas względem trajektorii odniesienia (6.5):

$$\mathbf{E}_i^{\mathrm{RPE}} = \left((\mathbf{T}_i^{\mathrm{gt}})^{-1}\mathbf{T}_{i+1}^{\mathrm{gt}}\right)^{-1} \left(\mathbf{T}_i^{-1}(\theta)\mathbf{T}_{i+1}(\theta)\right). \tag{6.5}$$

W związku z tym część rotacyjną mierzymy w stopniach (1[°]) a translacyjną w metrach (1[m]).

$$\underset{\theta}{\operatorname{argmin}} F_{\mathrm{RPE}_t} = \sum_{i=1}^{k} \mathrm{Trans}\left(\left((\mathbf{T}_i^{\mathrm{gt}})^{-1}\mathbf{T}_{i+1}^{\mathrm{gt}}\right)^{-1} \left(\mathbf{T}_i^{-1}(\theta)\mathbf{T}_{i+1}(\theta)\right)\right), \tag{6.6}$$

gdzie $\mathrm{Trans}(.)$ jest operatorem wyłuskującym translacyjną część transformacji i wyznaczającym jej normę euklidesową. W przedstawionych badaniach założono czas ruchu równy $1s$.

Wykresy ATE RMSE i RPE RMSE

Standardowy skrypt z Monachium zmieniono w taki sposób by czerwone segmenty, przedstawiające błąd euklidesowy pomiędzy pozycją odniesienia a oszacowaną, były wyświetlane co 10-ty raz oraz dla pierwszej i ostatniej pary.

We wszystkich wykresach błędów ATE RMSE, w niniejszej rozprawie, przyjęto konwencję, by zaznaczać kolorem czarnym trajektorię odniesienia, niebieskim oszacowaną, a czerwonym błędy (różnice pomiedzy nimi).

6.6 Badania

6.6.1 Porównanie par detektor-deskryptor

Badania rozpoczęto od porównania dostępnych i popularnych metod detekcji i deskrypcji punktów kluczowych (pewnych charakterystycznych cech występujących na obrazie) w estymacji trajektorii klatka po klatce (odometrii wizyjnej) bazującej na danych RGB-D. Ponieważ w testowym zastosowaniu nie możemy kompensować nieuniknionego dryftu, to dzięki temu można zobaczyć, jak różne podejścia ustalania lokalizacji punktów kluczowych wpływają na jakość odzyskanej trajektorii. Nacisk położyliśmy na wtedy nowo zaproponowanych metodach. Wykonanie takich niezależnych od autorów systemów nawigacyjnych badań jest niezwykle ważne, bo porównują oni swoją pracę z innymi na podstawie innych zestawów danych (zwykle należącymi do nich samych) w taki sposób, że ich podejście wygląda lepiej od innych.

Od połowy zeszłego stulecia powstało wiele prac poświęconych detekcji i deskrypcji punktów kluczowych. Nic więc dziwnego, że wraz z nowymi algorytmami pojawiają się nowe porównania. Jednymi z nowszych, jakie były dostępne w trakcie badań były np. prace Mikołajczyka i Schmidta [177], Mikołajczyka i innych [178] czy Schaeffera i Camerona [208]. Jednakże nie porównywały one detektorów i deskryptorów cech pod kątem estymacji trajektorii. Prace Filipa i Alexandre'a [126, 127] porównują punkty kluczowe $3D$ poprzez miarę powtarzalności, a nie w kontekście nawigacji robota. Schmidt i inni. [210] porównują kilka nowszych par detektor deskryptor w kontekście nawigacji wizyjnej, ale biorąc pod uwagę tylko dane fotometryczne (RGB) i oceniając wydajnosc swoją własną metryką. W pracy Manoja i innych [202] wykorzystano, tak jak tu, absolutny błąd trajektorii i względny błąd pozycji do porównania detektorów i deskryptorów cech $3D$ (z danych głębi). Jednakże w [202] trajektoria jest liczona poprzez śledzenie ujęcia z kamery do modelu, co w porównaniu z podejściem dopasowywania klatka po klatce, skutkuje tłumionym dryftem trajektorii.

Także niektóre prace dotyczące RGB-D SLAMu próbowały porównać wyniki osiągane przy stosowaniu różnych detektorów/deskryptorów w implementacji algorytmu SLAM. Np. Belter i inni [99] oceniali wydajność, bazującego na pozycjach systemu RGB-D SLAM, działającego z algorytmami FAST, SURF czy ORB. To porównanie daje cenny wgląd w to, ja-

ką rolę odgrywa wydajność algorytmu detekcji punktów kluczowych w systemie RGB-D SLAM, jednak nie bierze pod uwagę nowszych podejść, takich jak BRISK albo KAZE/AKAZE.

Jak przedstawiono w [103], niepewność położenia punktu kluczowego RGB-D ma dominujący wpływ na wydajność w bazującym na cechach systemie RGB-D SLAM.

Opis eksperymentu

W celu detekcji i deskrypcji cech skorzystano z algorytmów: SURF, KAZE, AKAZE, BRISK oraz ORB (są one dostępne w bibliotece OpenCV). Wyniki uzyskane za pomocą odometrii wizyjnej VO (zawierające dryft) porównujemy z tymi osiąganymi przez RGB-D SLAM v2, który może korzystać z trzech różnych detektorów i deskryptorów: SURF, SIFT oraz ORB, by pokazać możliwie najlepsze oszacowania trajektorii, których można by się spodziewać po zastosowaniu optymalizacji i redukcji dryftu. Z uwagi na ograniczoną pamięć komputera, na którym przeprowadzano wówczas badania, system RGB-D SLAM v2 mógł działać jedynie z co szóstą klatką. Zmienia on również parametry detektora w czasie działania.

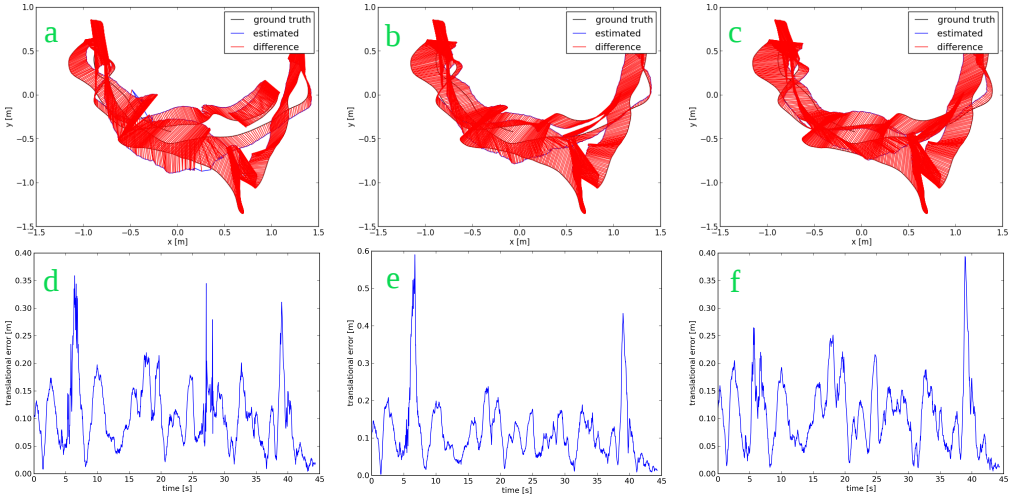
Wyniki

Dla sekwencji *fr1_room* pokazano na rys. 6.4, 6.5, oraz 6.6 wykresy błędów ATE, oraz RPE. Natomiast dla wszystkich sekwencji zdecydowano się ograniczyć do zebrania sumarycznych wyników ilościowych w tabelach, które dotycząc błędów ATE i RPE, czasów detekcji, deskrypcji, dopasowywania cech, działania RANSACa, algorytmu Kabscha, oraz średniej liczby inlierów i outlierów.

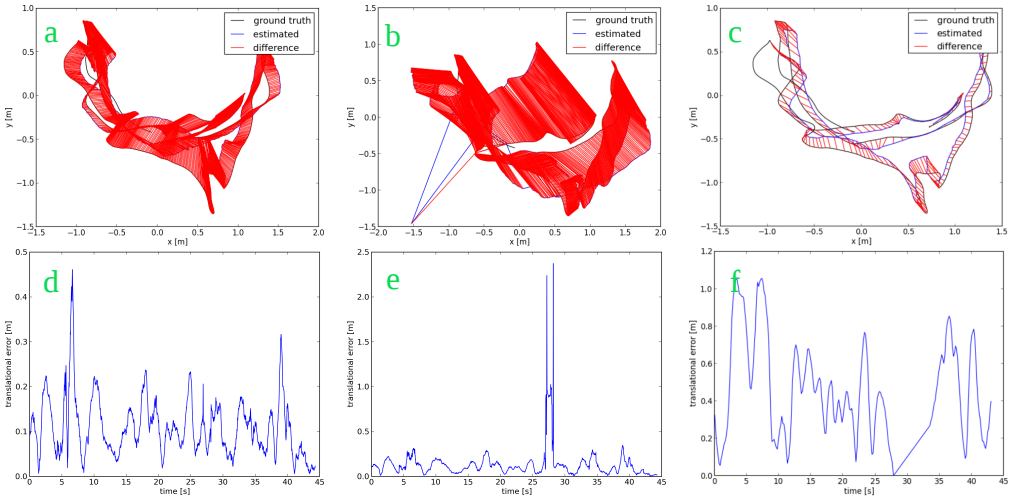
W sekwencji *fr1_room* można zobaczyć, że algorytm BRISK nie radzi sobie zbyt dobrze z ustawiony na stałe parametrami. Pozostałe podejścia zdają się dość dobrze ze sobą rywalizować. Wyniki dla tej sekwencji zebrano w tabelach 6.1 i 6.2.

Wyniki uzyskane przez wersję RGB-D SLAM v2 działającą z algorytmem ORB są niesatysfakcjonujące, bo nie są lepsze niż działająca klatka po klatce odometria, która nie wykonuje optymalizacji grafowej.

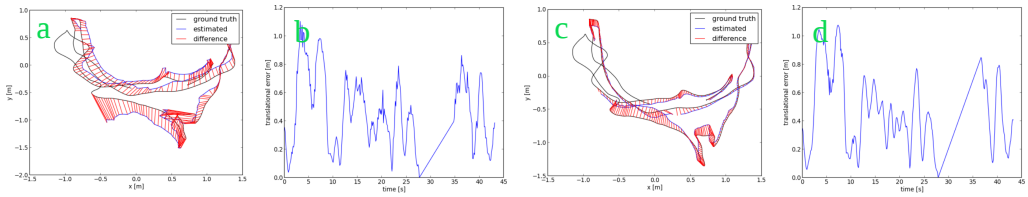
Podejście wykorzystujące ORB okazało się najszybsze w przypadku algorytmów detekcji i deskrypcji wykorzystywanych na potrzeby systemu odometrii wizyjnej. Tuż za nim, pod tym względem, okazał się AKAZE.



Rysunek 6.4: Wykresy błędów ATE (a–c) oraz RPE (d–f) dla oszacowanych trajektorii dla sekwencji *fr1_room* dla: (a, d) AKAZE, (b, e) KAZE, (c, f) SURF dla prostego VO



Rysunek 6.5: Wykresy błędów ATE (a–c) oraz RPE (d–f) oszacowanych trajektorii dla sekwencji *fr1_room* dla: (a, d) ORB, (b, e) BRISK, (c, f) RGB-D SLAM v2 (SURF) dla prostego VO



Rysunek 6.6: (a, c) Wykresy błędów ATE oraz (b, d) RPE oszacowanych trajektorii dla sekwencji *fr1_room* dla: (a, c) RGB-D SLAM v2 (ORB), (b, d) RGB-D SLAM v2 (SIFT) dla prostego VO

Tabela 6.1: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr1_room* dla prostego VO

	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
AKAZE	0,334	0,119	4,100
KAZE	0,373	0,137	3,986
SURF	0,377	0,123	3,939
ORB	0,379	0,127	3,873
BRISK	0,725	0,206	9,841
RGB-D SLAM v2 (ORB)	0,285	0,533	53,06
RGB-D SLAM v2 (SIFT)	0,095	0,537	54,61
RGB-D SLAM v2 (SURF)	0,124	0,540	53,91

Tabela 6.2: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr1_room* dla prostego VO

	średni czas [s]					średnia liczba	
	detekcji	deskrypcji	dopasowywania	RANSAC	Kabsch	inlierów	outlierów
AKAZE	0,107	0,090	0,052	0,607	$4,528 \cdot 10^{-5}$	232,77	86,08
KAZE	0,476	0,338	0,009	0,342	$2,692 \cdot 10^{-5}$	171,36	67,62
SURF	0,109	0,185	0,020	0,630	$4,370 \cdot 10^{-5}$	203,12	75,66
ORB	0,009	0,011	0,083	0,653	$3,799 \cdot 10^{-5}$	255,50	110,99
BRISK	0,006	0,004	0,066	0,750	$3,419 \cdot 10^{-5}$	157,65	77,31

SURF i BRISK były niewiele gorsze, natomiast KAZE był najwolniejszym z badanych algorytmów.

Jeśli chodzi o stosunek inlierów do outlierów, to prawie wszędzie był on równy i wynosił 3, poza BRISKiem, gdzie wyniósł on 2.

W sekwencji *fr1_desk* algorytm BRISK, choć pod względem błędów RPE nie odbiegał dalece od reszty, to pod względem błędów ATE po raz kolejny okazał się najgorszym. Najlepiej działały tu podejścia SURF oraz ORB, choć AKAZE niemal im dorównywał. Wyniki uzyskiwane przez RGB-D SLAM v2 są w przybliżeniu o 3 razy lepsze niż najlepszy wynik ATE uzyskany przez odometrię wizyjną, działającą klatka po klatce.

Wszystkie badane algorytmy detekcji i deskrypcji miały stosunek inlierów do outlierów równy $2,5$, poza BRISKiem, gdzie wyniósł on 2.

Najszybszym podejściem okazał się ORB. Niewiele wolniejszymi były SURF, KAZE i AKAZE (najdłuższy czas spędzony w testach RANSAC oznacza, że dopuszczalny próg błędu był wielokrotnie zwiększany). Najwolniejszym podejściem był algorytm BRISK. Wyniki dla tej sekwencji zebrano w tabelach 6.3 i 6.4.

Tabela 6.3: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr1_desk* dla prostego VO

	ATE	Trans. RPE	Rot. RPE
	RMSE [m]	RMSE [m]	RMSE [°]
AKAZE	0,096	0,074	2,487
KAZE	0,101	0,069	2,379
SURF	0,085	0,070	2,201
ORB	0,085	0,080	3,246
BRISK	0,202	0,099	3,089
RGB-D SLAM v2 (ORB)	0,063	0,719	39,551
RGB-D SLAM v2 (SIFT)	0,042	0,704	41,756
RGB-D SLAM v2 (SURF)	0,034	0,705	41,773

W sekwencji *fr3_long_office_household* najlepiej, pod względem błędów ATE, działały SURF oraz AKAZE. Pozostałe były niewiele gorsze. Natomiast błędy RPE są tu większe niż w poprzednich sekwencjach. Być może wynika to z jej długości i dryftu występującego w naturalny sposób w podejściu typu klatka po klatce.

Tu ORB również okazał się najszybszym algorytmem. Następnie dość porównywalnie szybko działały SURF, BRISK i AKAZE. Najwolniejszym był KAZE–był o ok. 50% wolniejszy od ORB.

Tabela 6.4: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr1_desk* dla prostego VO

	średni czas [s]					średnia liczba	
	detekcji	deskrypcji	dopasowywania	RANSAC	Kabsch	inlierów	outlierów
AKAZE	0,111	0,103	0,094	1,103	$5,950 \cdot 10^{-5}$	314,02	121,37
KAZE	0,477	0,353	0,016	0,562	$3,363 \cdot 10^{-5}$	230,50	92,72
SURF	0,114	0,218	0,033	0,886	$5,092 \cdot 10^{-5}$	261,80	100,77
ORB	0,010	0,011	0,085	0,844	$3,744 \cdot 10^{-5}$	260,38	108,90
BRISK	0,008	0,005	0,115	1,428	$4,152 \cdot 10^{-5}$	217,73	105,71

W tej sekwencji widać również zalety stosowania technik typu SLAM, gdzie błędy ATE dla wersji działającej z SURF i SIFT są o rząd wielkości mniejsze. Stosunek inlierów dla BRISKa pozostał ten sam i wyniósł 2. Dla pozostałych różni się wynosząc od $2,25$ dla ORB, do ok. $2,5$ dla AKAZE. Wyniki zebrano w tabelach 6.5 i 6.6

Tabela 6.5: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr3_long_office_household* dla prostego VO

	ATE	Trans. RPE	Rot. RPE
	RMSE [m]	RMSE [m]	RMSE [°]
AKAZE	0,861	0,192	10,832
KAZE	1,067	0,181	10,787
SURF	0,892	0,183	10,618
ORB	1,040	0,207	11,016
BRISK	1,190	0,188	10,714
RGB-D SLAM v2 (ORB)	0,404	0,342	20,846
RGB-D SLAM v2 (SIFT)	0,056	0,341	20,517
RGB-D SLAM v2 (SURF)	0,089	0,338	20,417

Wnioski

Biorąc pod uwagę wyniki wszystkich przeprowadzonych w tym badaniu eksperymentów, możemy powiedzieć, że najlepszymi algorytmami do estymacji trajektorii są SURF, AKAZE oraz niewiele gorszy ORB. Każdy z nich ma swoje zalety i wady. SURF i AKAZE pozwalają równie dobrze oszacować przebytą trajektorię w podobnym czasie. Jednak biorąc pod

Tabela 6.6: Wyniki ATE oraz RPE uzyskane dla sekwencji *fr3_long_office _household* dla prostego VO

	średni czas [s]					średnia liczba	
	detekcji	deskrypcji	dopasowywania	RANSAC	Kabsch	inlierów	outlierów
AKAZE	0,110	0,096	0,066	1,389	$5,340 \cdot 10^{-5}$	296,60	122,46
KAZE	0,458	0,334	0,011	0,738	$3,131 \cdot 10^{-5}$	222,61	97,44
SURF	0,116	0,205	0,024	1,290	$4,703 \cdot 10^{-5}$	240,06	104,94
ORB	0,010	0,012	0,110	1,246	$4,704 \cdot 10^{-5}$	341,66	150,93
BRISK	0,007	0,004	0,098	1,535	$4,401 \cdot 10^{-5}$	231,33	113,04

uwagę to, że w trakcie początkowych badań SURF był opatentowany, polecono podejście AKAZE jako dobre i wolne rozwiązanie.

ORB okazał się być najszybszym algorytmem służącym do detekcji i deskrypcji, pozwalając wykonać to zadanie o jeden rząd wielkości szybciej niż pozostałe podejścia. Choć zdaje się, że i tak był on spowolniony przez skupioną na uzyskaniu najlepszych wyników procedurę RANSAC. Skorzystanie z innej metody prawdopodobnie przyspieszyłoby ten proces, to stało by się to kosztem zwiększenia, mierzonego miarą ATE, dryftu.

Celem przeprowadzonych eksperymentów było uzyskanie najlepszej trajektorii metodą klatka po klatce, poprzez wykorzystanie określonych detektorów i deskryptorów bez dalszej optymalizacji grafowej. Znalezienie dobrego kompromisu pomiędzy prędkością działania a dokładnością estymacji trajektorii, np. poprzez stosowanie algorytmu wyszukiwania bazującego na populacji do optymalizacji parametrów, mógłby być interesującym krokiem dalszych badań. Biorąc pod uwagę optymalizację grafową, którą wykonuje system RGB-D SLAM v2, zdaje się, że ORB nie znajduje najlepszych punktów do tej optymalizacji.

6.6.2 Porównanie metod n-punktowych

W przeprowadzonych badaniach do zbierania danych wykorzystano wielokamerowy system o wysokiej rozdzielczości [211], bądź system przechwytywania ruchu (ang. *motion capture*) Vicon [225].

W tym eksperymencie skupiono się na sprawdzeniu metod n-punktowych, bazujących na cechach charakterystycznych i korzystających z obrazów RGB, w systemie odometrii wizyjnej. Pozwalają one na oszacowanie ob-

rotu kamery (a więc i robota), a kilka z nich także na estymację przebytej trajektorii z dokładnością do nieznanej skali. Do wstępnej obróbki obrazów wykorzystano pary detektor–deskryptor algorytmów AKAZE, ORB i SURF. Uzyskane wyniki odniesiemy do tych osiąganych przez algorytm Kabscha, który dodatkowo wykorzystuje pełną informacje o głębi, jakie mogą dostarczać powszechnie dostępne na rynku konsumenckim kamery RGB-D.

W początkowej fazie prowadzenia badań nie istniały metody pozwalające na odtworzenie z obrazu RGB informacji o odległości. To uniemożliwiało dokładne ustalenie stopnia, skali translacji pomiędzy dwoma ujęciami. Jednakże możliwe było ustalenie rotacji i wykorzystanie jej do ograniczenia orientacji robota. To ma kluczowe znaczenie, bo drobne błędy w tym zadaniu mogą spowodować duże, nakładające się błędy translacji (bo zakładamy ruch w innym kierunku niż rzeczywisty), tym większe jeśli pochodzi ona dodatkowo z innego źródła. W [210] pokazano, że dodanie oszacowania zmiany orientacji na podstawie danych wizyjnych jest korzystne dla systemów SLAM.

Opis eksperymentu

Do badań wykorzystano odometrię wizyjną podobną do wykorzystanej w badaniach zaprezentowanych w [163], jak i opisanej w rozdziale 3.6. Do eksperymentów wykorzystano 2 różne trajektorie: *putkk_Dataset_1_Kin_1* z zestawu PUTKK (do niego będziemy się dalej odnosić jako zestaw1) oraz *fr3_long_office_household* z benchmarku TUM RGB-D (do którego będziemy się odnosić jako zestaw2). Do oceny jakości odtwarzanej trajektorii skorzystano z metryk ATE RMSE i RPE RMSE, opisanych w podrozdziale 6.5.

W przypadku wykorzystywania algorytmu Eigensolver zezwolono w trakcie procedury RANSAC na wewnętrzną zmianę stosunku punktów uznawanych za przystające do odstających (oprócz progu służącego do ustalenia do której kategorii należy dany punkt). Na tym etapie korzystano również z dwóch różnych miar błędów: błędu reprojekcji oraz prostej odległości euklidesowej pomiędzy dwoma wektorami wodzącymi. Druga metoda pomija translację zakładając, że jest pomijalnie mała. Ostatecznie trajektoria liczona jest przy wykorzystaniu wszystkich dopasowań, które zostały.

Wyjątkiem jest tu algorytm Kneipa, który akceptuje od 5 do 8 par wektorów wodzących. Jest to także jedyny algorytm który nie oblicza transla-

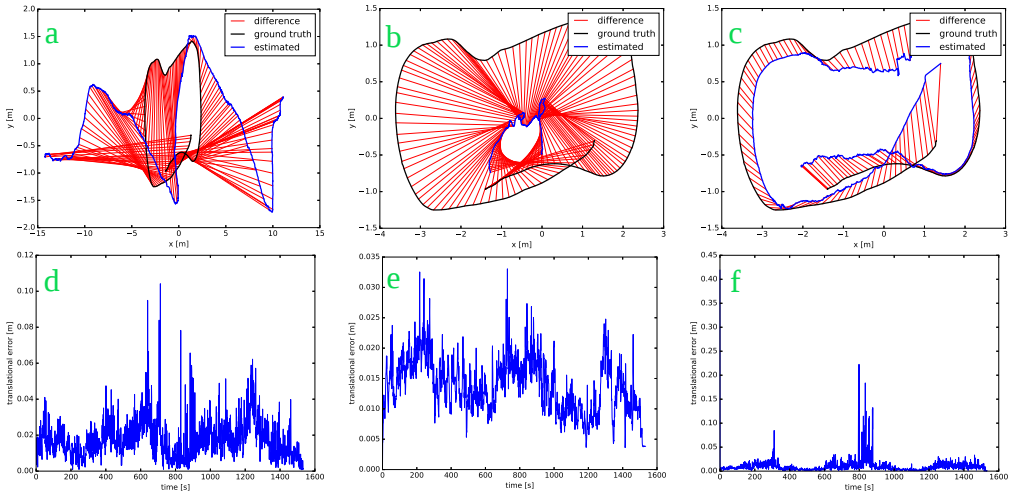
cji. Zatem by obliczyć błąd reprojekcji zastosowano algorytm dwu-punktowy, którego implementacja także jest dostępna w bibliotece OpenGV. Ponieważ kilka agorytmów, tj. Kneipa, Nistera oraz Ośmiopunktowy zwracają kilka wyników, to najlepszy wybierany jest na podstawie odpowiedniej funkcji błędu. W przypadku otrzymania pustego zbioru rozwiązań, będącego wynikiem odrzucania rozwiązań wewnątrz badanego algorytmu, tak długo, jak nie przekracza to maksymalnej liczby iteracji metody RANSAC, losuje się 8 punktów z zestawu *inlierów* do obliczenia obrotu i przemieszczenia. Jeśli liczba pozostałych punktów uznawanych za poprawnie dopasowane jest niewystarczająca do oszacowania trajektorii, to cały przebieg obliczeń zostaje przerwany jako nie potrafiący poprawnie odtworzyć trajektori.

W celu lepszej, bardziej zwartej i czytelniejszej prezentacji wyników skrócono część nazw. Trajektorie, które odtworzono przy wykorzystaniu Centroidów skrótowo oznaczono poprzez C. Te do których wykorzystano informację o Skali, gdzie wykorzystano mapy głębi z dwóch pierwszych ujęć do ustalenia skali, poprzez S. Dalej, skorzystanie z błędu Reprojekcji w procedurze RANSAC oznaczono poprzez R., a z błędu Euklidesowego poprzez E. Skrótowo zapisano również algorytm Eigensolver jako EigSol., oraz 8-punktowy jako 8-punkt.

Wyniki

W pierwszym eksperymencie okazało się w 3 z 4 przypadków, że sceneria była albo zbyt wymagająca albo znaleziono zbyt mało cech charakterystycznych, by Eigensolver czy algorytm Nistera odtworzyły całą trajektorię. Możliwe, że te podejścia byłyby w stanie podołać temu zadaniu gdyby bazowały na innym zestawie punktów kluczowych. Tym niemniej pokazuje to, że te metody nie są tak odporne jak pozostałe. Dalej, z perspektywy błędów rotacyjnych RPE RMSE, dość dyskusyjne jest to, czy algorytm Kneipa odtworzył przebytą przez kamerę trajektorię. Najlepszym, pod kątem względnej pozycji algorytmem n-punktowym okazał się algorytm 8-punktowy osiągając najlepszy wynik ze stałą skalą, głównie gdy ignorował translacyjny człon w schemacie RANSAC. Mimo, że zależy to od kroku inicjalizacji jak i wykrytych punktów, to różnice nie są tu wielkie. Jednak algorytm Kabscha, wykorzystujący pełną informację o głębi, osiągną od niego lepsze wyniki, co pokazano na rys. 6.7. Ta metoda, w połączeniu z detektorem/deskryptorem SURF osiąga niemal dwukrotnie

mniejsze błędy niż z inną parą detektora/deskryptora. Pełniejszy wgląd w wyniki daje wgląd w tab. 6.7.



Rysunek 6.7: Wykresy błędów ATE (a–c) oraz RPE (d–f) oszacowanych trajektorii dla sekwencji *putkk_Dataset_1_Kin_1* dla: (a, d) ORB C.E. Eigensolver, (b, e) SURF S.E. Nister, (c, f) SURF Kabsch

W drugim eksperymencie wszystkie algorytmy, poza podejściem Nistera sparowanym z detektorem/deskryptorem AKAZE, były w stanie zrekonstruować trajektorię. Najlepsze wyniki osiągnęła metoda Kabscha korzystająca z danych o głębi. Najlepszym, względem błędu ATE, algorytmem do oszacowania obrotu pomiędzy dwoma ujęciami, na podstawie tylko danych RGB, okazał się algorytm ośmiopunktowy. Natomiast jeśli metryka RPE stanowiłaby kryterium, to wówczas podejście Eigensolver zdaje się być lepsze. Wygląda również, że algorytm 8-punktowy działa nieco lepiej, jeśli procedura RANSAC korzysta z zaproponowanego błędu odległości Euklidesowej zamiast błędu reprojekcji. Jednakże w przypadku podjeść korzystających ze stałej skali, nie jest to regułą, gdyż w przypadku Eigensolvera dla cech SURF przynosi to korzyść, a dla AKAZE nie. Do problemów podejścia Eigensolver na pewno można zaliczyć fakt, że nie zawsze dostarcza rozwiązanie zadanego problemu. Nie jest więc tak niezawodny jak algorytm 8-punktowy.

Największe błędy RPE, z badanych metod, osiągnął algorytm Kneipa. Najprawdopodobniej wynika to z braku możliwości wykonania obliczeń dla większej liczby punków niż 8. Patrząc na rys. 6.8b i 6.8e można zało-

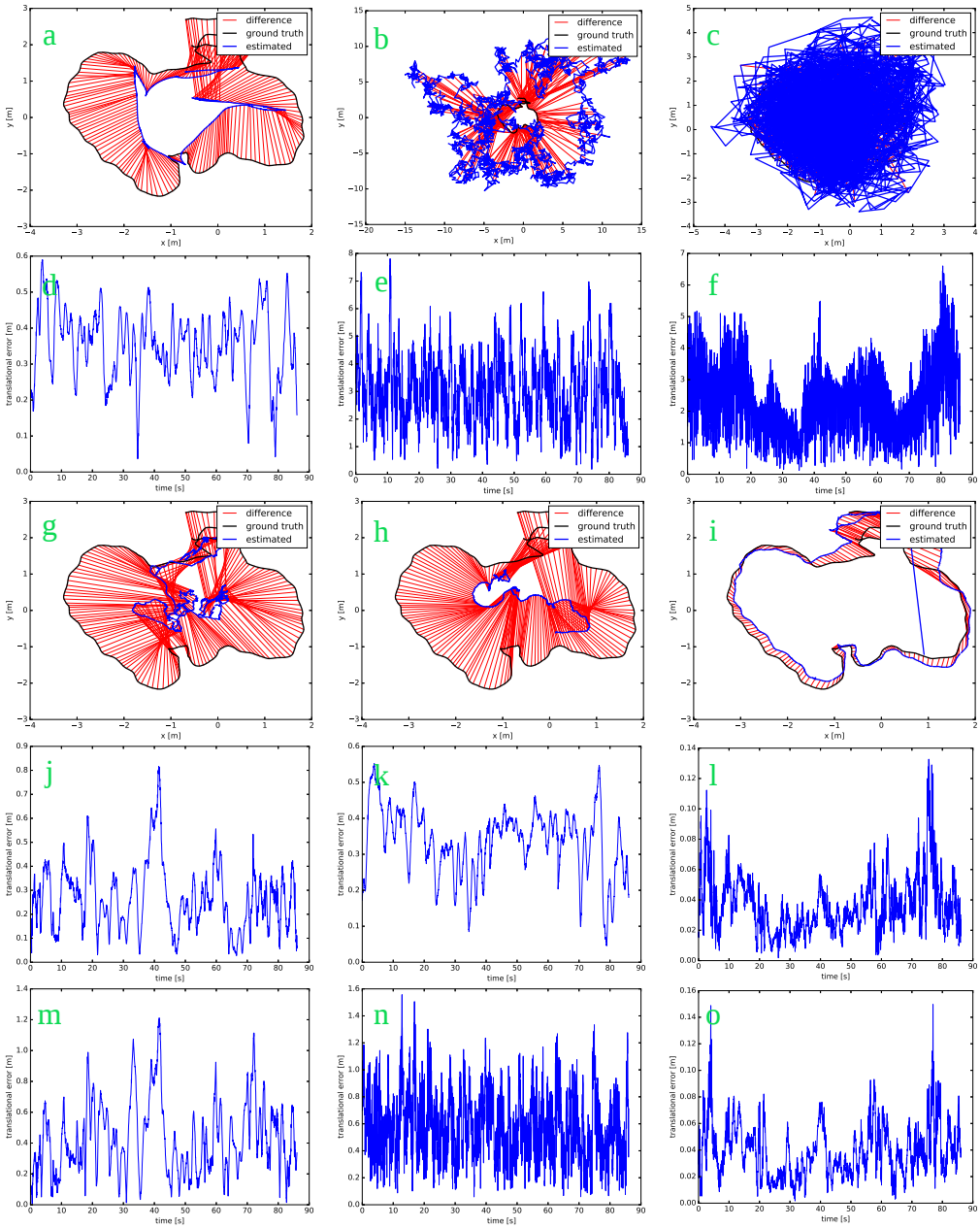
Tabela 6.7: Porównanie bezwzględnych błędów trajektorii (ATE RMSE) oraz względnych błędów pozycji (RPE RMSE) dla zestawu *putkk_Dataset_1_Kin_1*

algorytm	AKAZE			ORB			SURF		
	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]
C.E.EigSol.	7,267	0,021	0,372	7,292	0,021	0,372	7,261	0,021	0,372
C.E.Kneip	3,866	3,863	114,451	—	—	—	3,583	3,885	114,153
C.E.Nister	—	—	—	—	—	—	—	—	—
C.E.8-punkt.	4,044	0,047	4,683	4,067	0,042	0,928	4,064	0,042	0,926
C.R.EigSol.	—	—	—	—	—	—	—	—	—
C.R.Kneip	4,316	3,835	115,062	3,833	3,772	114,803	3,725	3,869	116,144
C.R.Nister	4,090	0,283	9,031	3,759	0,230	6,189	4,123	0,355	10,094
C.R.8-punkt.	3,843	0,063	4,691	4,033	0,042	0,973	3,968	0,042	0,966
S.E.EigSol.	2,043	0,014	0,372	2,044	0,014	0,372	2,032	0,014	0,372
S.E.Kneip	11,806	0,932	115,418	1,858	0,212	115,817	3,699	0,434	115,101
S.E.Nister	—	—	—	2,044	0,014	3,900	1,974	0,014	4,375
S.E.8-punkt.	1,745	0,018	0,922	1,735	0,018	0,924	1,715	0,018	4,685
S.R.EigSol.	—	—	—	—	—	—	—	—	—
S.R.Kneip	2,651	0,424	113,133	4,553	0,574	113,945	2,869	0,277	114,681
S.R.Nister	2,143	0,021	11,260	4,463	0,063	9,325	1,932	0,017	8,919
S.R.8-punkt.	1,861	0,025	0,956	1,768	0,023	0,963	1,699	0,021	0,955
Kabsch	0,724	0,019	0,413	0,714	0,020	0,430	0,470	0,020	0,417

żyć, że „zrekonstruował" przebytą przez kamerę trajektorię, gdy działał bez map głębi tylko na podstawie informacji o skali i korzystał z błędu reprojekcji. Natomiast ta metoda zawiodła, o czym świadczą rys. 6.8c i 6.8f, gdy wykorzystywała dane głębi w formie centroidów, a procedura RANSAC bazowała na błędzie euklidesowym. Ta metoda cechuje się dużymi błędami RPE. Powodem tego są pozostające outliery, zyskujące na znaczeniu przy liczeniu ostateczne estymaty. Inne algorytmy wykonują pewną formę optymalizacji w sensie najmniejszych kwadratów i dlatego wpływ pozostających punktów odstających na obliczenia jest silnie tłumiony. Wyniki wszystkich przebadanych podejść zestawu2 (sekwencja *fr3_long_office_household*) zebrano w tabeli 6.8 i uzupełniono wykresami zaprezentowanymi na rys. 6.8.

Tabela 6.8: Porównanie bezwzględnych błędów trajektorii (ATE RMSE) oraz względnych błędów pozycji (RPE RMSE) dla zestawu *fr3_long_office_household*

algorytm	AKAZE			ORB			SURF		
	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]
C.E.EigSol.	1,683	0,376	19,313	1,822	0,320	11,743	1,790	0,302	11,743
C.E.Kneip	2,571	2,628	131,423	3,211	2,925	131,514	3,334	2,918	130,940
C.E.Nister	—	—	—	2,066	1,224	43,236	1,636	1,128	35,555
C.E.8-punkt.	2,745	0,808	22,853	1,957	0,503	29,023	1,615	0,489	22,757
C.R.EigSol.	1,467	0,371	18,975	2,290	1,275	48,219	3,211	0,924	46,888
C.R.Kneip	2,933	2,653	132,805	3,996	3,016	131,065	3,579	2,949	131,599
C.R.Nister	2,028	1,131	45,637	2,459	1,842	73,429	2,316	1,138	42,402
C.R.8-punkt.	2,865	0,815	22,956	2,177	0,515	37,691	2,152	0,490	29,017
S.E.EigSol.	1,972	0,260	105,122	1,984	0,269	11,743	1,913	0,286	11,743
S.E.Kneip	1,957	0,367	131,377	1,737	0,465	132,218	1,920	0,508	131,092
S.E.Nister	—	—	—	1,857	0,349	42,416	1,600	0,400	33,549
S.E.8-punkt.	1,181	0,368	22,819	1,467	0,352	22,895	1,202	0,358	22,777
S.R.EigSol.	1,803	0,244	21,157	1,998	0,253	49,985	1,935	0,269	49,098
S.R.Kneip	7,865	3,266	131,034	2,076	0,623	131,848	2,105	1,173	132,467
S.R.Nister	1,916	0,345	46,895	2,140	0,543	71,752	1,869	0,513	39,916
S.R.8-punkt	1,246	0,354	22,887	1,786	0,394	29,294	1,925	0,391	29,788
Kabsch	0,560	0,042	1,267	0,576	0,046	1,372	0,619	0,044	1,434



Rysunek 6.8: Wykresy błędów ATE (a–c, g–i) oraz RPE (d–f, j–o) oszacowanych trajektorii dla sekwencji *fr3_long_office_household* dla: (a, d) AKAZE S.E. 8-point, (b, e) AKAZE S.R. Kneip, (c, f) AKAZE C.R. Kneip, (g, j) SURF S.E. Eigensolver, (h, k) ORB S.E. Nister, (i, l) AKAZE Kabsch, m) SURF C.E. 8-point, n) ORB S.R. Kneip, o) SURF Kabsch

Wnioski

Biorąc pod uwagę przedstawione wyniki badań można dojść do wniosku, że najlepszym algorytmem do rozwiązania problemu szacowania ruchu klatka po klatce jest algorytm Kabscha, który jednak wymaga danych głębi, np. z sensora RGB-D. W pasywnych systemach korzystających z jednej kamery, najlepszy okazał się algorytm 8-punktowy, jako że Eigensolver, mimo iż dokładny, nie zawsze był w stanie dostarczyć rozwiązanie problemu. Gdy tylko nie ma dostępu do danych głębi to tylko algorytm 8-punktowy może odtworzyć przebytą trajektorię. Pozostałe algorytmy zwykle działały znacznie gorzej od dwóch wcześniej wymienionych podejść. Wyższość algorytmu ośmiopunktowego może wynikać z lepszej optymalizacji metodą najmniejszych kwadratów. Ponadto, pracując z rzeczywistymi danymi, niedoskonałymi danymi, założenia leżące u podstaw algorytmu 8-punktowego są o wiele bardziej realistyczne niż rygorystyczne założenia algorytmu 5-punktowego.

Zdaje się, że wybór detektora/deskryptora ma niewielki wpływ w porównaniu z miarą błędu wykorzystywaną w procedurze RANSAC, a także ma niewielki wpływ na osiągane wynik. Natomiast algorytm Kabscha, w zależności od środowiska, raz działa lepiej z detektorem/deskryptorem SURF a raz z AKZAE i ORB. Wynika z tego, że dla różnych środowisk potrzeba innych podejść do wykrywania cech charakterystycznych i nie ma jeszcze idealnego detektora takich cech. Wydaje się, że algorytm 8-punktowy działa lepiej z cechami AKAZE lub SURF niż z ORB. Algorytm cech ORB znany jest ze swej szybkości działania i to właśnie ze względu na tą wydajność obliczeniową był brany pod uwagę, lecz niestety jego wyniki w naszych eksperymentach były raczej rozczarowujące. Biorąc pod uwagę, że algorytm AKAZE jest otwartoźródłowym oprogramowaniem, zalecamy rozważenie wykorzystania tej pary detektora/deskryptora zamiast SURF.

6.6.3 Analiza metod RGB-D SLAM w samolokalizacji robotów mobilnych

Niniejsza praca stanowi próbę odpowiedzi na pytanie, jak różne podejścia do zagadnienia przetwarzania danych RGB-D w systemie SLAM wpływają na jakość estymowanej trajektorii ruchu sensora umieszczonego na ro-

bocie mobilnym. W tym celu przeprowadzono także badania korzystając z danych zebranych przy użyciu rzeczywistych, poruszających się robotów, kołowego i kroczącego [87], pracujących w różnych typach środowisk oraz dostępnych otwartoźródłowych metod SLAM.

Opis eksperymentu

Postęp w rozumieniu struktury problemu jednoczesnej samolokalizacji i budowy mapy otoczenia (ang. *Simultaneous Localization and Mapping–SLAM*) oraz dostępność otwartych implementacji wydajnych algorytmów optymalizacji nieliniowej umożliwiły w ostatnich latach odejście od tradycyjnych metod SLAM opartych na filtracji na rzecz optymalizacji pewnej (często grafowej) reprezentacji tego problemu [230, 224]. Kamery RGB-D okazały się być praktyczne w lokalizacji robotów mobilnych, nawet tych o ograniczonym rozmiarze, nośności czy mocy obliczeniowej [190]. Ich pojawienie się spowodowało również przesunięcie nacisku badań nad systemami SLAM z wizji pasywnej na te korzystające z bardziej wiarygodnych, bezpośrednich pomiarów głębi [184]. W efekcie architektury ewoluujące w kierunkach wizyjnych czy też RGB-D [218] stały się dominujące, ale nie wykształciła się jednak dominująca architektura systemu SLAM, zarówno wizyjnego, jak i RGB-D.

Publicznie dostępne zestawy danych umożliwiają porównanie nowych architektur do rozwiązań znanych z literatury [183]. Jednakże te porównania bazują na danych zebranych za pomocą ręcznie trzymanej kamery we względnie ograniczonej przestrzeni [225], bądź na sztucznych danych pochodzących z symulacji [137]. Tego rodzaju ewaluacja nie obrazuje jednak potencjału danego rozwiązania w aplikacjach z obszaru robotyki mobilnej, gdzie różne rodzaje robotów, o różnej dynamice ruchu (nie raz nagłej) wymagają rozwiązań SLAM odpornych na takie czynniki jak rozmycie obrazu (ang. *motion blur*), brak tekstury w polu widzenia, przesłonięcia, cienie i zmiany oświetlenia. W rzeczywistości ich nasilenie może być różne.

W tym badaniu podejmujemy próbę porównania reprezentatywnych architektur systemów RGB-D SLAM w kontekście użycia ich do samolokalizacji robotów mobilnych w pomieszczeniach. Wybrano jedynie systemy, których kod dostępny jest na zasadach otwartoźródłowego oprogramowania (ang. *opensource*).

Badane systemy różną się podejściem do problemu SLAM. Stosują różne rozwiązania w celu likwidacji dryfu trajektorii pojawiającego się w odo-

metrii wizyjnej VO, często będącej prostszym podejściem do samolokalizacji na podstawie takich samych danych. Wykorzystano zarówno podejścia bazujące na cechach charakterystycznych: (ORB-SLAM2 [184], CCNY_RGBD [118], RGB-D SLAM v2 [121]) jak i bez (KinFu Large Scale [199]). Skoncentrowano się na rozwiązaniach wykorzystujących cechy punktowe, kierując się przekonaniem o ich większej przydatności w zadaniach nawigacji robotów mobilnych. Wykorzystanie wszystkich pikseli wymaga sporej mocy obliczeniowej i wsparcia GPGPU, które nie jest dostępne dla większości robotów.

Wstępne badania wykazały, że RGB-D SLAM v2 działa lepiej z detektorem/deskryptorem SURF (potwierdza to również praca [113]). Cechy ORB okazały się nieskuteczne przy zamykaniu pętli. Te wyniki potwierdzono niezależnie w [113]. Systemy CCNY_RGBD oraz RGB-D SLAM v2 działały w środowisku systemu programowania robotów ROS (ang. *Robot Operating System*) [89]. W celu zbadania i zobrazowania jakie znaczenie mają te rozwiązania dla osiąganej precyzji estymacji trajektorii i pokazania w jakim stopniu różne techniki redukcji dryftu poprawiają dokładność oszacowanej trajektorii, wykorzystano implementację prostego system odometrii wizyjnej RGB-D (VO RGB-D)–a więc możliwie najprostszego podejścia do rozwiązania problemu lokalizacji.

Osiągana precyzja nie tylko wpływa na jakość budowanej mapy otoczenia ale jest bardzo ważna np. przy planowaniu kolejnych punktów podparcia dla robota kroczącego–należy dokonać lokalizacji z dokładnością mniej więcej równą wielkości [101].

Do badań wykorzystano zestawy danych: *fr3_long_office_household*, *putkk_Dataset_1_Kin_1* oraz *messor2_2*. Dalej skorzystano z sekwencji *messor2_1* i *messor2_3*. Pierwsza sekwencja charakteryzuje się znaczną długością. Jednak kamera poruszana jest powoli ręcznie i obserwuje niewielki wycinek przestrzeni. Druga sekwencja zarejestrowała była rejestrowana przez robota kołowego. Ostatnie trzy sekwencje rejestrowane były przez robota kroczącego.

Wyniki

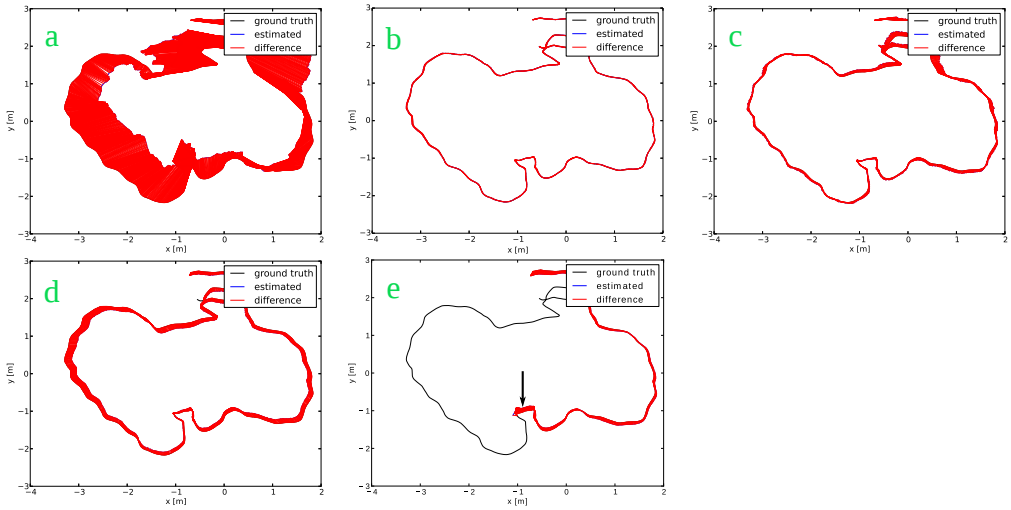
Wyniki pierwszego eksperymentu stanowią punkt odniesienia dla testów na danych uzyskanych z robotów. Wszystkie systemy SLAM wykorzystujące cechy punktowe (ORB-SLAM2, CCNY_RGBD, RGB-D SLAM v2) prawidłowo odtworzyły trajektorię sensora z błędem ATE RMSE w granicach

10 cm (wyniki liczbowe przedstawiono w zbiorczej tabeli 6.9, natomiast odpowiednie wykresy ATE i RPE na rys. 6.9 oraz 6.10). Architektura SLAM wykorzystująca optymalizację mapy cech–ORB-SLAM2 (rys. 6.9b) uzyskała bardzo dużą dokładność estymacji trajektorii z błędem ATE na poziomie typowego błędu pomiaru odległości sensora Kinect.

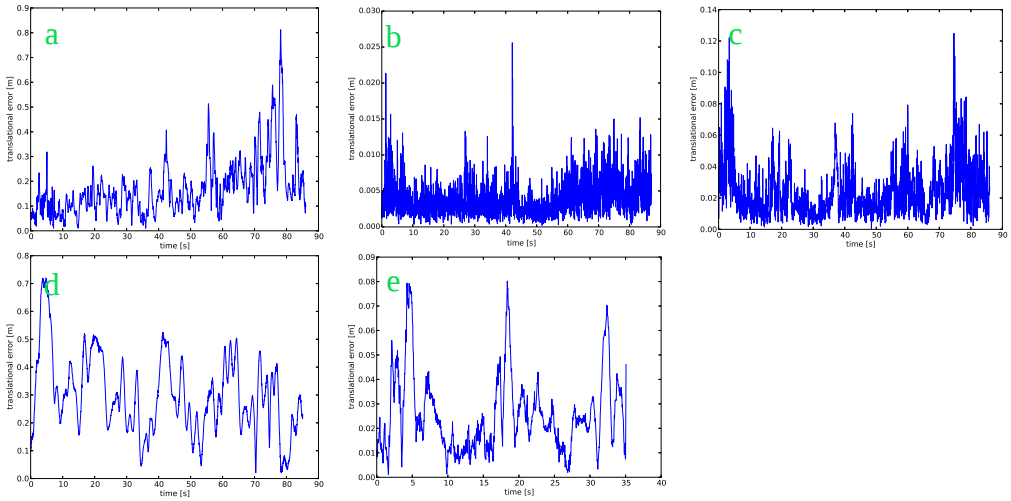
Tabela 6.9: Porównanie bezwzględnych błędów trajektorii (ATE RMSE) oraz względnych błędów pozycji (RPE RMSE) badanych systemów

System lokalizacji	*fr3_long _office_household*			*putkk_Dataset _1_Kin_1*			*messor2_2*		
	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]
VO RGB-D (ORB)	1,098	0,206	10,95	0,612	0,012	0,22	1,553	0,337	16,16
ORB-SLAM2 (ORB)	0,009	0,004	0,25	0,018	0,004	0,12	—	—	—
CCNY_RGBD (ORB)	0,107	0,029	1,07	0,529	0,033	1,95	0,308	0,118	6,40
RGB-D SLAM v2 (SURF)	0,095	0,331	19,81	—	—	—	0,211	0,140	10,55

Niewielka przewaga ORB-SLAM2 wynika prawdopodobnie głównie z precyzyjniejszej lokalizacji cech za pomocą zmodyfikowanego, wieloskalowego detektora ORB. Zarówno system, który nie wykorzystywał optymalizacji mapy cech a bazował na filtrze Kalmana (CCNY_RGBD, rys. 6.9c), jak i system oparty na optymalizacji grafu pozycji (RGB-D SLAM v2, rys. 6.9d) odnotowały większe błędy, co wskazuje na korzyści płynące z bezpośredniego dopasowywania cech do mapy podlegającej optymalizacji wraz z pozycjami sensora metodą w stylu regulacji wiązki BA (ang. *Bundle Adjustment*). System KinFu Large Scale (do którego będziemy się dalej odnosić skrótowo jako KinFu LS), pomimo niewielkich błędów ATE na początku trajektorii nie odtworzył jej w całości, tracąc możliwość śledzenia sensora przy jego bardziej gwałtownych ruchach (miejsce zaznaczone strzałką na rys. 6.9e). Trajektorię odtworzył natomiast w całości prosty system odometrii wizyjnej RGB-D (rys. 6.9a), dla którego jednak kumulujący się dryf doprowadził do bardzo dużego błędu ATE przy porównywalnych do innych systemów błędach względnych RPE.



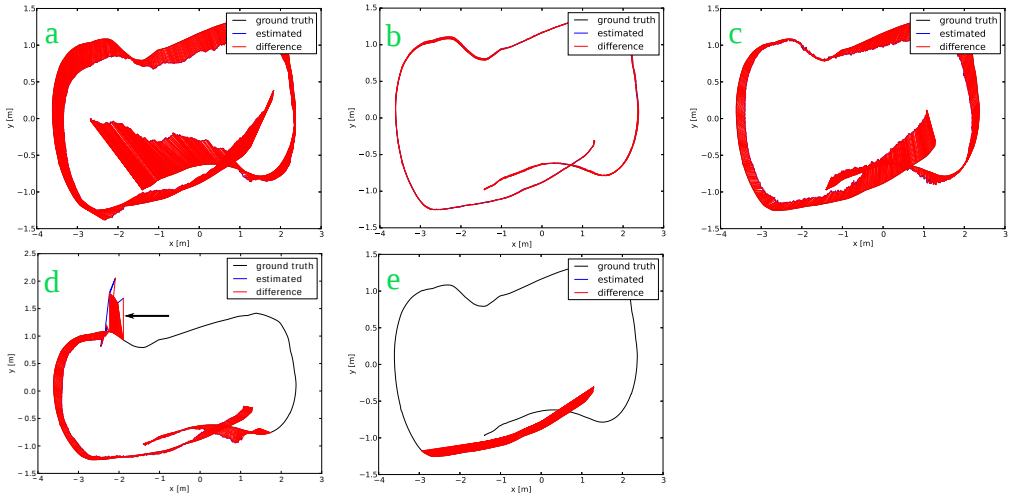
Rysunek 6.9: Wykresy błędów ATE oszacowanych trajektorii dla sekwencji *fr3 _long_office_household* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS



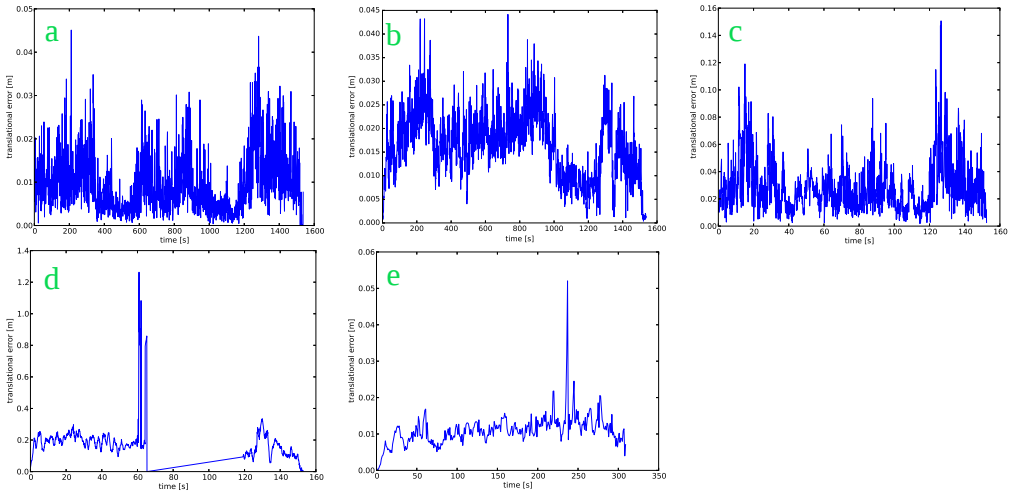
Rysunek 6.10: Wykresy błędów RPE oszacowanych trajektorii dla sekwencji *fr3 _long_office_household* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

Eksperyment wykorzystujący dane z robota kołowego ujawnił rolę zamykania pętli w architekturze SLAM. Odometria wizyjna pomimo skutecznego dopasowywania kolejnych klatek (mała wartość RPE w tab. 6.9) odnotowała duży błąd ATE spowodowany dryfem (rys. 6.11a). Podobną wartość ATE zaobserwowano dla CCNY_RGBD (rys. 6.11c), co sugeruje, że korygowanie pozycji pojedynczych cech bez optymalizacji powiązań cech i pozycji jest mało skuteczne dla map obejmujących większy obszar, gdzie ponowne obserwacje cech następują po nagromadzeniu znacznego dryfu trajektorii. RGB-D SLAM v2 nie zdołał odtworzyć całej trajektorii w tym eksperymencie. System ten zaprzestał śledzenia sensora w miejscu, gdzie kolejne ramki RGB charakteryzowały się małą liczbą cech (miejsce zaznaczone strzałką na rys. 6.11d). Jednak przetwarzając kolejne ramki RGB-D SLAM v2 wykrył zamknięcie pętli przy powrocie robota w okolice miejsca startu i wznowił obliczanie pozycji sensora (rys. 6.11d). Wartość ATE RMSE dla całej trajektorii jest w tym przypadku niemiarodajna i została pominięta w tab. 6.9. W tym przypadku ORB-SLAM2 (rys. 6.11b) uzyskał zdecydowanie lepszą dokładność estymacji trajektorii. Wynik ten jest spowodowany dwoma czynnikami–wykorzystaniem jawnego wykrywania zamknięcia pętli na podstawie analizy widoków, co pozwoliło skorygować trajektorię w jej końcowym fragmencie, oraz korzystaniem także z cech nieposiadających danych o głębi, co pozwoliło systemowi ORB-SLAM2 na wykorzystanie bardzo odległych cech (sekwencja była zbierana w dużym pomieszczeniu). System KinFu LS (rys. 6.11e) odtworzył tylko fragment trajektorii, urywając śledzenie przy pierwszym skręcie.

Trzeci eksperyment pozwolił określić odporność badanych systemów na negatywne zjawiska związane z gwałtownymi ruchami sensora, w tym przypadku spowodowanymi dyskretną naturą ruchu robota kroczącego i nieuniknionymi poślizgnięciami. Rozmyte obrazy RGB powodowały często niedokładną estymację ruchu sensora między kolejnymi ramkami. W przypadku prostej odmetrii wizyjnej spowodowało to odtworzenie całkowicie błędnej trajektorii (rys. 6.13a). Pełnej trajektorii nie odtworzył także ORB-SLAM2, kończąc śledzenie na sekwencji następujących po sobie rozmytych obrazów (rys. 6.13a), będących wynikiem pierwszego skrętu. Zawiodły również próby ponownego odnalezienia własnej pozycji (co widać w postaci linii ciągłej na rys. 6.14b). System ORB-SLAM2 wykorzystuje podczas tworzenia ograniczeń w mapie tylko błąd reprojekcji na obrazie (jak SLAM monokularowy), ignorując dane o głębi. Powoduje to zwiększoną wrażliwość na zakłócenia obrazu takie jak rozmycie i rolling shut-

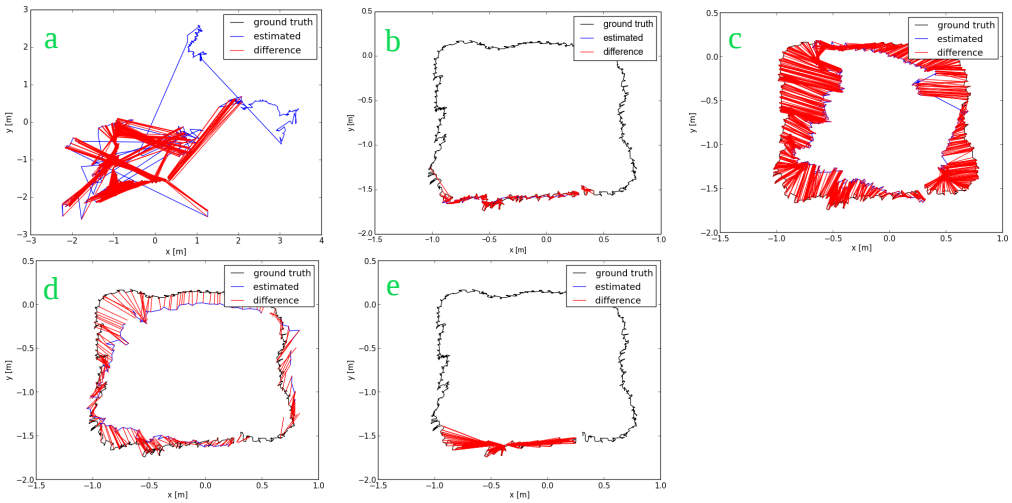


Rysunek 6.11: Wykresy błędów ATE oszacowanych trajektorii dla sekwencji *putkk _Dataset_1_Kin_1* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS



Rysunek 6.12: Wykresy błędów RPE oszacowanych trajektorii dla sekwencji *putkk _Dataset_1_Kin_1* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

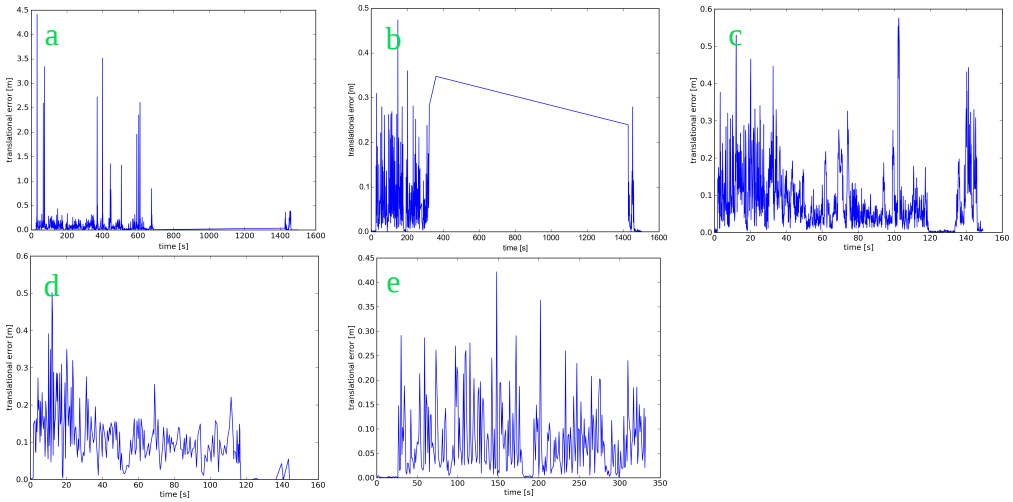
ter. Systemom CCNY_RGBD (rys. 6.13c) oraz RGB-D SLAM v2 (rys. 6.13d) udało się odtworzyć trajektorię robota, lecz w ich przypadku błędy ATE RMSE były zauważalnie duże. System KinFu LS nie używa cech punktowych, więc rozmycie obrazów RGB nie ma dla niego znaczenia. Jednak i tym razem KinFu LS nie odtworzył całej trajektorii, tracąc zdolność do śledzenia sensora przy pierwszym ostrym skręcie (rys. 6.13e).



Rysunek 6.13: Wykresy błędów ATE oszacowanych trajektorii dla sekwencji *messor2_2* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

Następnie do dalszych badań wykorzystano sekwencje *messor2_1* oraz *messor2_3*, koncentrując się tylko na rozwiązaniach SLAM. Tabela 6.10 zawiera wyniki liczbowe dla wspomnianych sekwencji.

W czwartym eksperymencie, robot poruszał się z większą prędkością, co powodowało większe wibracje tułowia robota. Jak widać na wykresach ATE oraz RPE (rys. 6.15 i6.16) jakość odtworzonych trajektorii jest kiepska dla wszystkich badanych systemów SLAM. Prosta odmetria wizyjna odtworzyła całkowicie błędną trajektorię (rys. 6.15a). System RGB-D SLAM v2 zdołał odtworzyć sensownie (w globalnym sensie) całą trajektorię, jednak jego błędy ATE RMSE (rys. 6.15d) wynoszą ok. 15 cm i jest zbyt wielki np. do planowania kolejnych pozycji nóg robora kroczącego–błędy te powinny być niższe niż 5 cm. Błędy RPE (rys. 6.16d) również były duże. System CCNY_RGBD nie był w stanie poprawić niektórych dużych błędów w ustalaniu pozycji oraz dryftu, więc uzyskana trajektoria jest bez-



Rysunek 6.14: Wykresy błędów RPE oszacowanych trajektorii dla sekwencji *messor2_2* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

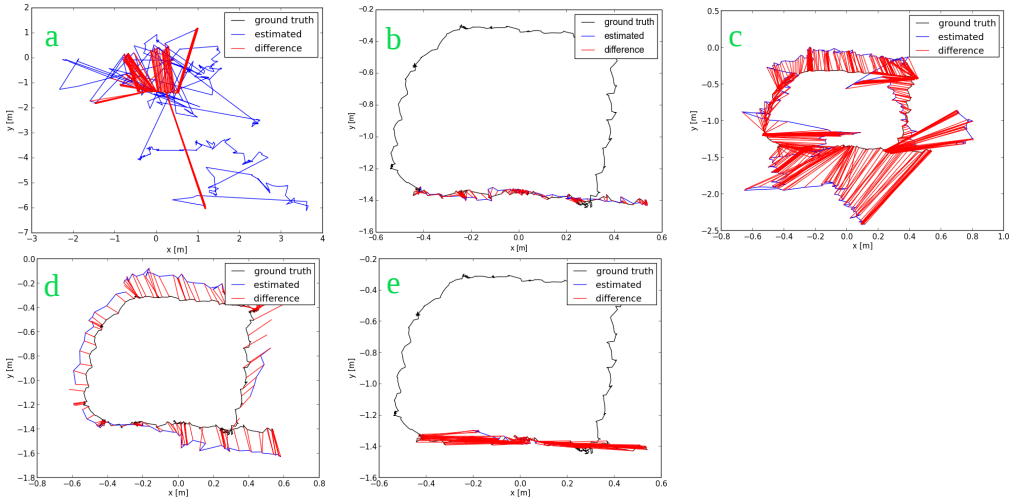
Tabela 6.10: Porównanie bezwzględnych błędów trajektorii (ATE RMSE) oraz względnych błędów pozycji (RPE RMSE) badanych systemów

System lokalizacji	*messor2_1*			*messor2_3*		
	ATE [m]	RPE [m]	RPE [°]	ATE [m]	RPE [m]	RPE [°]
VO RGB-D (ORB)	3,000	0,434	15,92	0,114	0,014	1,04
ORB-SLAM2 (ORB)	—	—	—	0,085	0,012	5,43
CCNY_RGBD (ORB)	0,468	0,141	10,22	0,103	0,032	6,37
RGB-D SLAM v2 (SURF)	0,155	0,138	15,94	0,198	0,095	11,04

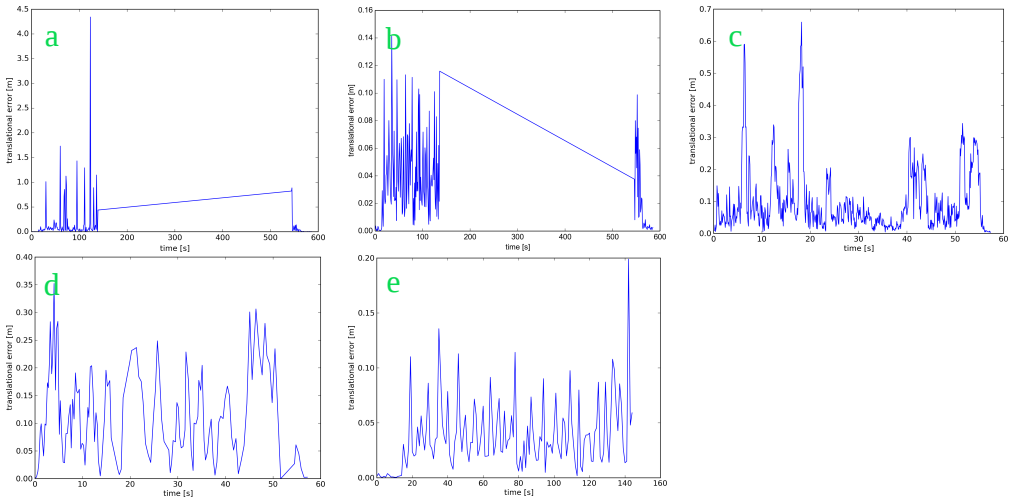
użyteczna (rys. 6.15c i 6.16c). Obydwa systemy osiągnęły podobne wyniki, jak poprzednio, choć odtworzona poprzednio (w *messor2_2*) trajektoria przez CCNY_RGBD była jakościowo lepsza. Zaskakująco ORB-SLAM2 znany z dostarczania bardzo dokładnych estymat trajektorii ruchu, w kilku benchmarkach [184] nie podołał postawionemu w tej sekwencji zadaniu (rys. 6.15b). Ten system także i w tej sekwencji stracił możliwość śledzenia ruchu sensora podczas pierwszego skrętu, gdy robot szybko skręcał. Zawiodły (tak jak poprzednio) również próby ponownego odnalezienia własnej pozycji (co widać w postaci linii ciągłej na rys. 6.16b). Najgorsze wyniki zaobserwowano dla systemu KinFu LS. Metryki ATE oraz RPE były obliczone tylko dla części trajektorii przed pierwszym skrętem, gdyż bazująca na algorytmie ICP architektura SLAM nie była w stanie dodać kolejnych klatek głębi po pierwszym ostrym skręcie robota w pierwszym narożniku występującym na jego drodze (rys. 6.15e i 6.16e).

Główną przyczyną błędnego ustalania pozycji sensora, a nawet niepowodzeniem odtworzenia całej trajektorii przez system SLAM, może być zbyt duże rozmycie obrazu, jakie da się zaobserwować na zarejestrowanych klatkach RGB. Sprawiło to, że wykryte na obrazach cechy punktowe są niewiarygodne. Być może wpłynęło to także na obliczenia błędu reprojekcji w systemie ORB-SLAM2. Architektura KinFu, która na nich nie bazuje (bo w ogóle nie korzysta z obrazów RGB), zawiodła z powodu zbyt dużej odległości fizycznej pomiędzy kolejnymi klatkami głębi, spowodowanymi niską częstotliwością (15 Hz) pobierania danych oraz wysoką prędkością kątową robota (gdy obracał się niemalże natychmiastowo w miejscu). Ukazało to procedurę ICP jako nieskuteczną.

W ostatnim eksperymencie, przeprowadzonym na zestawie *messor2_3* zaobserwowano znaczną poprawę działania prostego systemu odometrii wizyjnej. Nie tylko jest to pierwsza sekwencja z zestawu Messor II, dla której odtworzył on całą trajektorię, ale też uzyskał lepsze wyniki od systemów SLAM. Może to być spowodowane wysoką stabilnością robota, która pozwoliła zrobienie nierozmytych zdjęć i dokładnych map głębi. Dzięki temu zatarte zostały różnice pomiędzy systemami SLAM, które w trakcie optymalizacji były w stanie zniwelować błędne lokalizacje poszczególnych pozycji robota, wynikające z chaotycznie zmienianych położeń punktów kluczowych (powodowanych przez wibracje). Nie bez znaczenia było też stopniowe zwiększanie akceptowalnego progu błędu, po przekroczeniu którego system VO odrzucał badaną hipotezę w procedurze RANSAC. W tym badaniu nie zaobserwowano poprawy systemu RGB-D SLAM v2

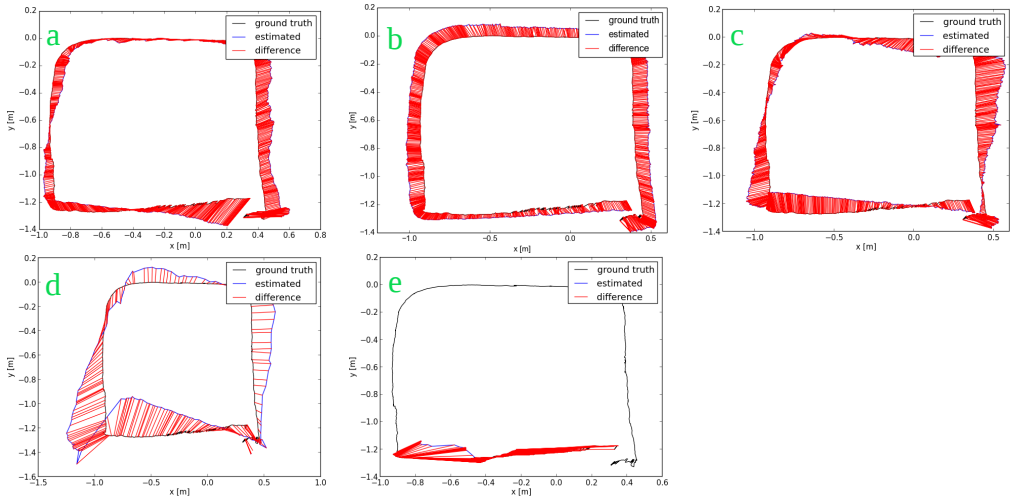


Rysunek 6.15: Wykresy błędów ATE oszacowanych trajektorii dla sekwencji *mes-sor2_1* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

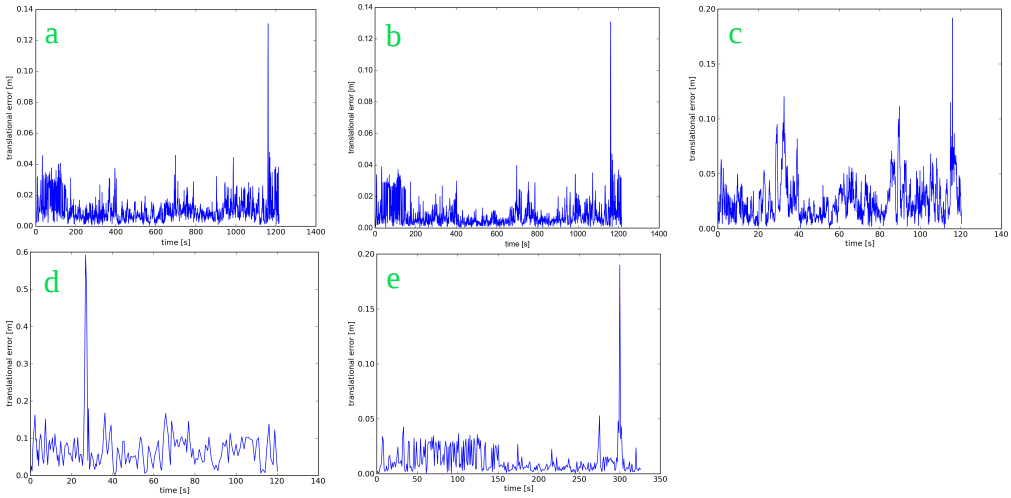


Rysunek 6.16: Wykresy błędów RPE oszacowanych trajektorii dla sekwencji *mes-sor2_1* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

(rys. 6.17d) pomimo mniejszych błędów RPE (rys. 6.18d). Uzyskane wyniki (gorsze od zaproponowanej prostej odometrii wizyjnej) mogą rzucają podejrzenia co do wdrożonej odometrii wizyjnej, jak i uniwersalności nastaw parametrów (które należy pamiętać, że były dopasowane do konkretnych zestawów danych). W przeciwieństwie do tego wyniki uzyskane przez CCNY_RGBD się poprawiły (rys. 6.17c i 6.18c). Była to również pierwsza sekwencja (z zestawu Messor II) dla której system ORB-SLAM2 odtworzył całą trajektorię. Ponieważ wartości RMSE błędów ATE i RPE były niewielkie, to można wysnuć wniosek, że rozmycie (którego prawie tu nie ma) obrazów RGB spowodowane ruchem w połączeniu z szybkimi obrotami robota były głównymi przyczynami niepowodzeń w trzecim (*messor2_2*)i czwartym (*messor2_3*) eksperymencie. KinFu LS nadal nie był w stanie poradzić sobie z szybkimi obrotami sensora przy pierwszym zakręcie (rys. 6.17e) i w efekcie ponownie nie był w stanie odtworzyć całej trajektorii. Nie uwzględniono dlatego systemu wyników w tabeli 6.10, ponieważ uzyskał on tylko częściowe wyniki.



Rysunek 6.17: Wykresy błędów ATE oszacowanych trajektorii dla sekwencji *messor2_3* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS



Rysunek 6.18: Wykresy błędów RPE oszacowanych trajektorii dla sekwencji *messor2_3* dla: a) RGB-D VO, b) ORB-SLAM2, c) CCNY_RGBD, d) RGB-D SLAM v2, e) KinFu LS

Wnioski

Jak wykazały przeprowadzone eksperymenty, architektury systemów RGB-D SLAM uważane za reprezentatywne dla obecnego stanu wiedzy nie w każdym przypadku radzą sobie dobrze z danymi zebranymi podczas eksperymentów odzwierciedlających warunki pracy rzeczywistego robota mobilnego. Największe problemy sprawiły gwałtowne, szarpane i nieprzewidywalne ruchy sensora, w szczególności tego przenoszonego przez robota kroczącego, które pogarszały pobierane obrazy (to uwidacznia się głównie w systemach SLAM bazujących na cechach). Burzą one zarówno założenie o dostępności precyzyjnie ulokowanych na obrazie cech punktowych (ze względu na rozmycie), jak i przyjęte proste modele ruchu sensora (jak w ORB-SLAM2). Trochę mniejsze błędy uzyskano, gdy robot wspierany był poprzez pięć odnóż. Sugeruje to, że być może należałoby połączyć proces pobierania danych o otoczeniu wraz z algorytmem kontrolującym chód robota. Problemem okazały się też, obserwowane w eksperymencie drugim, ramki pozbawione cech punktowych ze względu na charakter środowiska (jednolite powierzchnie w polu widzenia) lub z powodu niewystarczającego zasięgu pomiaru odległości sensorem RGB-D (duże pomieszczenie).

System ORB-SLAM2 nie był w stanie odtworzyć dwóch trajektorii z zestawu Messor II, które cechowały się dużymi rozmyciami obrazów. Jed-

nakże to podejście, wykonujące optymalizację grafową, wykazało się skutecznością działania w eksperymentach bez nadmiernych wibracji i nagłych ruchów. Osiągnęło w nich najlepsze wyniki dzięki wykorzystaniu stałej mapy cech wizyjnych, pozwalającej na nałożenie dużej liczby ograniczeń na oszacowaną trajektorię. Okazuje się także, że lepiej sobie radzi z cechami ulokowanymi w różnych odległościach od sensora, nawet tymi leżącymi poza zasięgiem sensora RGB-D, bez informacji o głębi, które wobec ograniczonego zasięgu, mogą być bardzo istotne w praktyce. System ten mógłby osiągnąć jeszcze lepsze wyniki, gdyby zamiast prostego modelu ruchu sensora zintegrował z sobą szybki system odometrii wizyjnej. Można w nim również zmienić sposób radzenia sobie z niepewnościami pomiarów głębi.

System CCNY_RGBD pokazał, że stała mapa cech pozwala na osiągnięcie małych względnych błędów w szacowaniu ruchu sensora. Dokładność odwzorowywanej trajektorii ulegała poprawie wraz z poprawą jakości obrazów. Jednakże wykorzystany filtr Kalmana nie był w stanie poradzić sobie z dużą niepewnością lokalizacji cech wykrytych na rozmytych obrazach. System RGB-D SLAM v2, korzystający z, bazującej na pozycjach, optymalizacji grafowej g^2o, odnosi niewielkie korzyści ze zwiększonej liczby cech kluczowych zaobserwowanych przy zmniejszonej prędkości ruchu robota, uzyskując mniej dokładne estymaty trajektorii. W trakcie badań zaobserwowano, że system RGB-D SLAM v2 zwalniał wraz ze wzrostem liczby klatek kluczowych, bo optymalizacja zajmowała więcej czasu. W eksperymentach przeprowadzonych na sekwencjach Messor II system RGB-D SLAM v2 nie był w stanie działać w czasie rzeczywistym.

Architektura SLAM wykorzystująca „gęste“ dane o głębi–KinFu LS okazała się bardzo zawodna. Rozwiązanie to powstało z myślą o aplikacjach dotyczących rozszerzonej rzeczywistości i nie radzi sobie z szybkimi ruchami sensora/robota (szczególnie obrotami), które nie pozwalają na prawidłowe dopasowywanie ramek do mapy metodą ICP (najwyraźniej wynika to ze zbyt dużych odległość pomiędzy nadchodzącymi obrazami głębi). Wpływ na to miał także brak bardziej złożonych (lepiej określonych, zdefiniowanych) struktur geometrycznych w polu widzenia.

Przeprowadzone badania pokazują, że im gładszy jest ruch robota, tym lepsze wyniki badane systemy osiągają, ale to systemy korzystające z optymalizacji grafowej robią najlepszy pożytek z jakościowych cech punktowych.

W celu dalszej poprawy uzyskanych wyników można by również do-

konać integracji, bazującego na obrazach RGB-D, systemu SLAM z pomiarami udostępnianymi przez inne sensory, np. jednostkę do nawigacji inercyjnej IMU (ang. *Inertial Measurements Unit*). W [128] pokazano, że w bazującym na danych stereo system S-PTAM nawet luźne podejście do integracji danych z IMU poprawia dokładność lokalizacji pomagając przezwyciężyć problemy spowodowane gwałtownymi ruchami. Dokonano tego w nowszej (3) wersji ORB-SLAM [109].

6.6.4 Wykorzystanie metod populacyjnych do poszukiwania najlepszych parametrów systemu odometrii wizyjnej RGB-D w zagadnieniu offline

Różnorodność rozwiązań systemów VO i SLAM jak i ich szczegółowość powoduje, że trudno jest stwierdzić jak poszczególne wybory projektowe i wartości parametrów (zarządzających zachowaniem oddzielnych bloków, algorytmów system) wpływają na wydajność. W [99] pokazano, że zaawansowany backend nie jest w stanie skompensować kiepsko działającego frontendu VO RGB-D. Nawet dla rozwiązania opartego na dyskretnych cechach–punktach kluczowych [164] optymalizacja parametrów wymaga przeszukania wielowymiarowej przestrzeni rozwiązań, która w zależności od wyboru optymalizowanych parametrów nie musi być ciągła (bo np. nie da się w prosty sposób zlinearyzować wartości parametrów). Dlatego, w niniejszym badaniu, staramy się sprawdzić, czy możliwa jest automatyzacja doboru parametrów dla wybranych parametrów prostego przypadku systemu lokalizacji wizyjnej RGB-D–odometrii wizyjnej opartej na cechach punktowych. Chcemy znaleźć najlepsze parametry dla prostego systemu odometrii wizyjnej za pomocą algorytmów optymalizacji opartych na populacji rozwiązań–interesuje nas wybranie odpowiedniej metodologii.

W tym celu założono stałą strukturę systemu i do optymalizacji parametrów wykorzystano sprawdzone już (np. w budowie modelu dynamiki do symulacji sześcionożnego robota [102]) metodę optymalizacji roju cząstek (ang. *Particle Swarm Optimization–PSO*) [119] i algorytm ewolucyjny (ang. *Evolutionary Algorithm–EA*) [91], które wzajemnie się uzupełniają i radzą sobie z bardzo trudnymi problemami optymalizacyjnymi [192], takimi jak znalezienie najlepszych parametrów dla adaptacyjnego filtru Kalmana [133]. PSO znane jest ze swej prostoty oraz szybkiej zbieżności, podczas

gdy adaptacyjny EA oferuje odporne działanie z minimalnym nastawianiem parametrów, utrzymując bardziej zróżnicowaną populację.

Kierując się wynikami poprzednich badań [163], do detekcji i deskrypcji cech punktowych wykorzystujemy algorytm AKAZE [195]. Alternatywne podejście typu „gęstego" [130, 157] jest zbyt wymagające obliczeniowo w kontekście rozważanych tu metod optymalizacji. Pomimo tego, że optymalizacji dokonujemy dla danego zestawu danych, sprawdzamy również jak bardzo otrzymane parametry są uniwersalne–tj. jakie wyniki można dzięki nim uzyskać w innych sekwencjach. W tej wersji odometrii wizyjnej nałożyliśmy szczególny nacisk, by zminimalizować jakiekolwiek losowe elementy (innymi niż RANSAC–zwiększono maksymalną liczbę iteracji do 10000), która może negatywnie wpłynąć na zbieżność optymalizacji, osłabiając zależność pomiędzy obliczoną wartością funkcji dopasowania a poszukiwanymi parametrami [151].

W zależności od algorytmu (PSO bądź EA) zestaw parametrów nazywa się cząstkami bądź osobnikami, poprzez środowisko rozumie się system odometrii wizyjnej VO a za funkcję celu (miarę dopasowania) wykorzystywaną miarę błędu–ATE lub RPE.

Parametry systemu

Dla jakości estymowanej trajektorii sensora kluczowe znaczenie ma efektywność dopasowywania cech między kolejnymi klatkami oraz jak najmniejsze odległości między cechami uznanymi za poprawnie skojarzone. Z drugiej jednak strony, dążenie do małych błędów resztkowych dopasowania może prowadzić do odrzucenia wielu dopasowań w procedurze RANSAC. To z kolei prowadzi do wyznaczania transformacji na podstawie małej liczby punktów i obniża jakość rozwiązania.

Kluczowe znaczenie dla uzyskania dobrych jakościowo trajektorii ma dobór parametrów procedur RANSAC (odpowiednio dla pierwszej i drugiej procedury RANSAC). Pierwsze dwa to progi odległości euklidesowych $d_{E,1}$ i $d_{E,2}$ po przekroczeniu którego daną parę punktów uznaje się za odstającą, bo błąd pomiędzy pozycją punktu z poprzedniego ujęcia po transformowaniu go do nowego ujęcia, a rzeczywistym położeniem na nowym ujęciu jest zbyt duży. Kolejne dwa ważne to stosunki między liczbą akceptowalnych (*inliers*) i nieakceptowalnych (*outliers*) dopasowań $\Gamma_{o,1}$ i $\Gamma_{o,2}$ (także dla pierwszej i drugiej procedury RANSAC).

System VO potrzebuje również właściwie dobranej do charakteru scen wartości progu detekcji τ_A dla detektora AKAZE. Jest to najważniejszy parametr systemu, gdyż głównie od niego zależy liczba wykrywanych cech punktowych. Od ich jakości i liczby bezpośrednio zależy jakość otrzymanej trajektorii. Opisane powyżej pięć zmiennych tworzy wektor parametrów $\theta = [d_{E,1}, d_{E,2}, \Gamma_{o,1}, \Gamma_{o,2}, \tau_A]$. Parametry te są poddawane optymalizacji zarówno wspólnie, jak i w wersji hierarchicznej, gdzie po ustaleniu najlepszych progów RANSAC optymalizacji podlega jedynie parametr detektora (wówczas $\theta = \tau_A$). W trakcie wykonywania obliczeń dla danej sekwencji nie zmieniamy tych parametrów $\Gamma_{o,1}, \Gamma_{o,2}, \tau_A$, a $d_{E,1}, d_{E,2}$ zmienia jedynie procedura RANSAC w sytuacji gdy zostanie przekroczona maksymalna liczba dozwolonych losowań.

Parametry algorytmu zostały dobrane tak, by uzyskać wyniki w rozsądnym czasie (kilkanaście godzin). Przyjęto c_1=c_2=2, a dopuszczalna prędkość cząstki wynosiła od $-0,005$ do $0,005$. Liczba cząstek wynosiła 40, a maksymalna liczba iteracji 9 i 20 (w zależności czy optymalizowane było 5 parametrów, czy tylko parametr detektora τ_A). Dobrana liczba cząstek miała związek z maksymalną liczbą wątków równolegle obsługiwanych przez program. Maksymalna liczba iteracji nie została osiągnięta podczas eksperymentów, optymalizacja kończyła się po mniejszej liczbie iteracji ze względu na warunek niewystarczającej poprawy wyniku w kolejnych iteracjach.

Opis eksperymentu

W celu poprawy uzyskiwanych wyników postanowiliśmy dokonać optymalizacji parametrów systemu odometrii wizyjnej. By takie postępowanie miało sens musieliśmy sprawdzić, czy uzyskany za pomocą badanych metod populacyjnych zestaw parametrów jest uniwersalny, to znaczy czy da się go stosować z powodzeniem w innych środowiskach, niż to wykorzystane w optymalizacji. Dlatego posłużyliśmy się trzema różnymi zestawami danych RGB-D. Dwa pochodziły z *TUM RGB-D*: *fr1_desk* i *fr1_room*–to właśnie je wykorzystywaliśmy do optymalizacji parametrów. Do uzyskanych w ten sposób zestawów parametrów będziemy się dalej odnosić odpowiednio jako OP1 i OP2. Do weryfikacji otrzymanych wyników wykorzystaliśmy sekwencję *putkk_Dataset_1_Kin_1* ze znacząco innego, również publicznie dostępnego zestawu danych PUTKK. Nie była ona wykorzystywana w optymalizacji.

Te trzy wykorzystane sekwencje różnią się wielkością sceny, dynamiką poruszania się kamery, jak i widzianymi przez nią obiektami. Skutkuje to różną liczbą wykrytych istotnych cech na pojedynczej klatce RGB-D (Tab. 6.11).

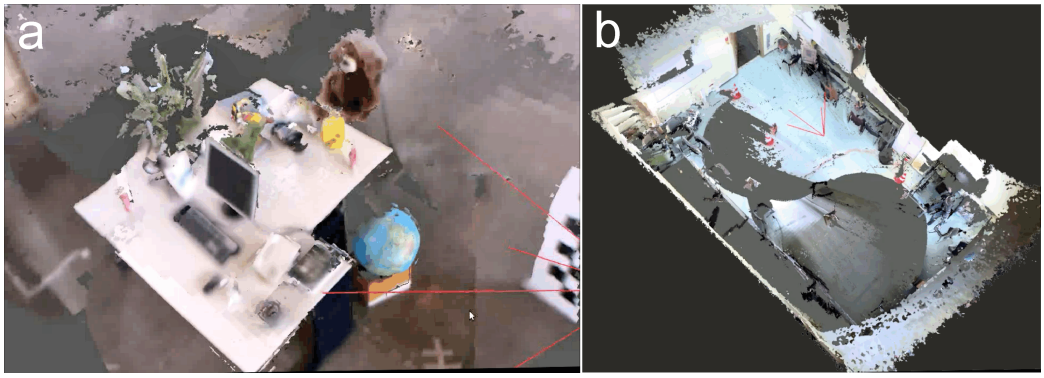
Tabela 6.11: Liczba istotnych cech na klatkę wykryta przez VO z progiem detekcji τ_A AKAZED zoptymalizowanym przez algorytm ewolucyjne za pomocą funkcji dopasowania bazującej na metryce ATE

sekwencja danych RGB-D	min. liczba punktów	śr. liczba punktów	max. liczba punktów
fr1_desk	105	659	1435
fr1_room	12	509	1301
putkk_Dataset_1_Kin_1	42	328	795

W sekwencjach benchmarku TUM RGB-D Kinectem poruszano ręką. W *fr1_desk* powoli, zaś o wiele szybciej w *fr1_room*, gdzie dodatkowo trajektoria pokrywała znacznie większą objętość przestrzeni. W przeciwieństwie do tego w zestawie PUTKK, sensor Asus Xtion został przymocowany do kołowego robota, czego efektem był szybszy, ale i gładszy ruch. W związku z tym minimalna liczba cech wykrytych na klatce była znacznie mniejsza w *fr1_room*, głównie z powodu występujących tam szybkich obrotów kamery. W *putkk_Dataset_1_Kin_1* nigdy nie ma tak małej liczby wykrytych cech, choć ich średnia i maksymalna liczba jest mniejsza niż w obydwu sekwencjach benchmarku TUM RGB-D. Wynika to z większego pomieszczenia i ograniczonego zasięgu odbioru głębi sensora RGB-D. Na rys. 6.19 pokazano przykładowe wizualizacje środowisk wykorzystanych w badaniu.

Badania rozpoczęto od optymalizacji wszystkich pięciu parametrów systemu: progu detekcji AKAZE τ_A i parametrów RANSACA. Użyliśmy algorytmu PSO z metryką ATE jako funkcji celu. Sesja optymalizacja nie przekroczyła dziewięciu iteracji.

Żeby przedstawić zachowanie się cząstek w wielowymiarowej przestrzeni poszukiwań w czasie, posłużyliśmy się rys. 6.20 pokazującym ewolucję cząstek podczas eksperymentów OP1 oraz OP2. Wizualizacji ewolucji podczas jednego eksperymentu składa się z dwóch wykresów $3D$, których osie z oznaczają progi detekcji AKAZE. Czerwone kropki reprezentują początkowe (losowane) pozycje cząstek, niebieskie pośrednie (w środkowej



Rysunek 6.19: Wizualizacja kolorowych chmur punktów zarejestrowanych za pomocą trajektorii odniesienia z sekwencji a) *fr1_desk*, i b) *putkk_Dataset_1_Kin_1*

iteracji sesji optymalizacyjnej), a zielone ostateczne, po osiągnięciu kryterium stopu. Większa, czarna cząstka oznacza zbiór parametrów optymalnych, tj. najlepszą cząstkę. W obydwu eksperymentach parametry dążą do optymalnych wartości.

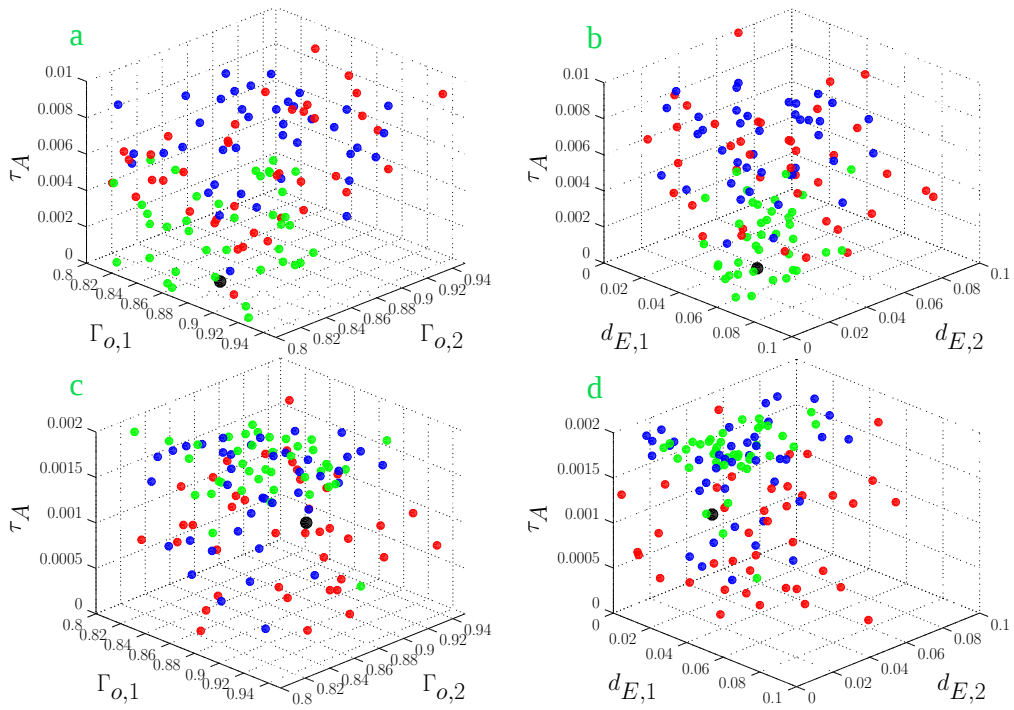
Tab. 6.12 zawiera ręcznie dobrane wartości parametrów oraz te będące wynikiem optymalizacji, znalezione podczas tego eksperymentu.

Tabela 6.12: Zoptymalizowane parametry VO z ich najlepszymi wartościami

Metoda	$d_{E,1}$	$\Gamma_{o,1}$	$d_{E,2}$	$\Gamma_{o,2}$	τ_A
OP1	0.059	0,812	0,042	0,836	0,0021
OP2	0.042	0,916	0,012	0,855	0,0013
Ręcznie	0.030	0,800	0,003	0,800	0,0020

W obydwu przypadkach znalezione optymalne wartości parametru $\Gamma_{o,2}$ są bardzo podobne. Oznacza to, że po wstępnej, zgrubnej filtracji cech osiągamy podobną liczbę punktów odstających (outlierów). Jednakże inne progi odległości euklidesowej $d_{E,2}$ sugerują, że dokładność tych cech zależy od sceny. Wartość $d_{E,2}$ zależy od $\Gamma_{o,1}$–im więcej punktów przejdzie do drugiej pętli RANSAC, tym bardziej wymagająca może być ta filtracja.

O wiele ciekawsza jest sytuacja początkowych progów odległości euklidesowej $d_{E,1}$. Są one podobne i znajdują się w środku ustawionego zakresu, podczas gdy można by się spodziewać, że będą blisko minimalnej wartości, pozwalając na stopniowe zwiększanie akceptowalnego progu błędu, gdy nie da się znaleźć odpowiednio satysfakcjonującej transformacji.



Rysunek 6.20: Ewolucja cząstek reprezentujących parametry podczas optymalizacji na sekwencji: (a, b) *fr1_desk*, (c, d) *fr1_room*. Na (a, c) oś x przedstawia parametr $\Gamma_{o,1}$ a oś y $\Gamma_{o,2}$, natomiast na (b, d) oś x $d_{E,1}$ a y $d_{E,2}$. Na wszystkich osiach z zaznaczony jest parametr τ_A. Znaczenie kolorów opisano w głównym tekście

Pokazuje to, że mimo że dałoby się zaakceptować wstępną transformację sprawdzoną w procedurze RANSAC, to mogą istnieć inne podobne rozwiązania. Sugeruje to, że lepiej jest wykorzystać kilka punktów więcej do estymacji transformacji niż mniejszą liczbę najlepszych cech. Nawet jeśli uważa się, że część tych punktów jest outlierami dla tej konkretnej transformacji, to mogą być lepsze dla innego, bliskiego, podobnego rozwiązania będącego wynikiem wyboru innych trzech punktów kluczowych na potrzeby filtracji RANSAC.

Parametr τ_A jest niemal identyczny dla parametrów OP1 i manualnie dobranych. Natomiast $d_{E,1}$, $\Gamma_{o,1}$ oraz $\Gamma_{o,2}$ minimalnie się różnią. Tylko ręcznie dobrana wartość $d_{E,2}$ mocniej się różni od odpowiadającego mu parametru OP1. Zestaw parametrów OP2 ma największy stosunek $\Gamma_{o,2}$ z wszystkich zestawów parametrów. Oznacza to, że w trudniejszych scenach (jaką jest np. *fr1_desk*) powinno się możliwie jak najszybciej pozbyć słabych cech, szczególnie w trudnych ujęciach, nie pozwalając na późniejszy negatywny wpływ słabo powiązanych, dopasowanych par punktów na estymację trajektorii.

Wartości ATE RMSE oraz RPE RMSE (translacyjne i rotacyjne) dla tych zestawów parametrów oraz tych znalezionych ręcznie są zebrane w tab. 6.13.

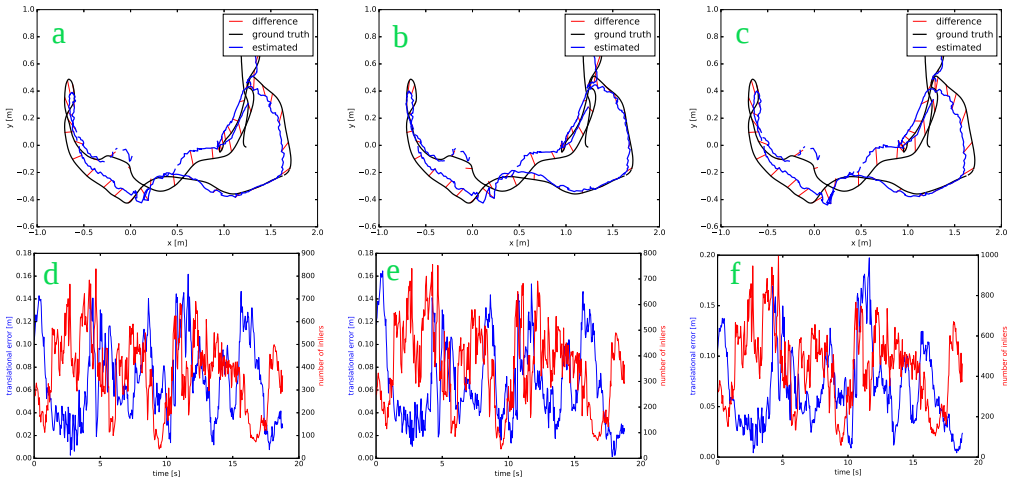
Tabela 6.13: Porównanie ATE RMSE oraz RPE RMSE dla trzech sekwencji

	fr1_desk			*fr1_room*		
Metryka błędu	Ręcznie	OP1	OP2	Ręcznie	OP1	OP2
ATE RMSE [m]	0,095	0,082	0,113	1,573	1,728	0,288
Trans. RPE RMSE[m]	0,078	0,076	0,080	0,348	0,358	0,115
Rot. RPE RMSE [°]	2.876	2,746	2,621	22,632	26,201	2,529

	putkk_Dataset_1_Kin_1		
Metryka błędu	Ręcznie	OP1	OP2
ATE RMSE [m]	0,829	0,912	0,697
Trans. RPE RMSE[m]	0,019	0,020	0,013
Rot. RPE RMSE [°]	0,374	0,407	0,248

Różnice w jakości oszacowania trajektorii metodą klatka po klatce dla różnych zestawów parametrów dobrze widać na rys. 6.21(d–f), 6.22(d–f), 6.23(d–f). Pokazują one „lokalną" jakość estymacji trajektorii, np. pomijając wpływ, jaki ma niedokładne oszacowanie poprzedniej części trajekto-

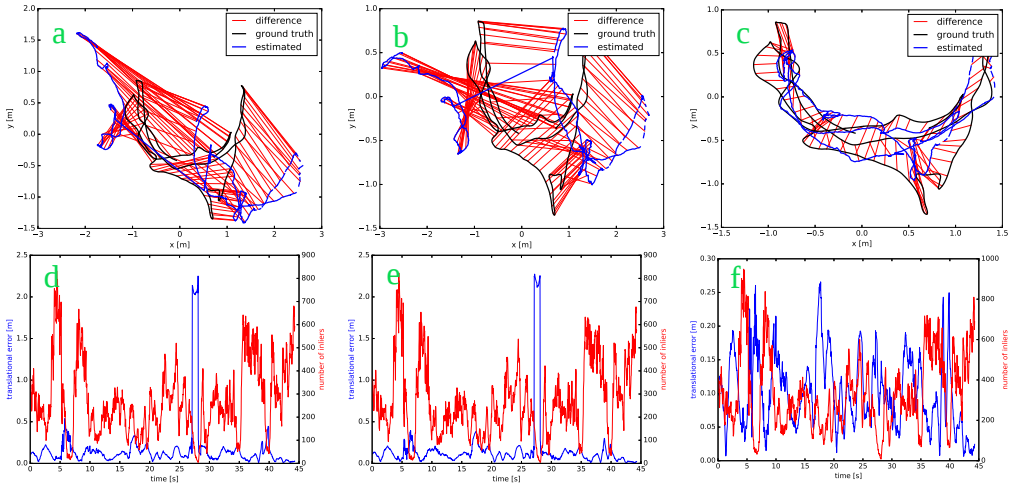
rii na obecne położenie kamery. Z tego porównania widać, że parametry optymalizowane na wystarczająco kłopotliwej sekwencji (jaką np. był OP2, która zawierała o wiele szybsze ruchy kamerą i zróżnicowane obiekty w jej polu widzenia) pozwalają na osiągnięcie względnych translacji bez dużych błędów. Widać również związek pomiędzy liczbą *inlierów* wykorzystanymi do oszacowania przebytej trajektorii, a jej jakością. Gdy tych punktów jest zbyt mało, to zaczynają pojawiać się problemy a jakość znacznie ona spada. W trakcie oglądania wykresów warto zwrócić na skalę osi y (błędu translacji) rys. 6.22f.



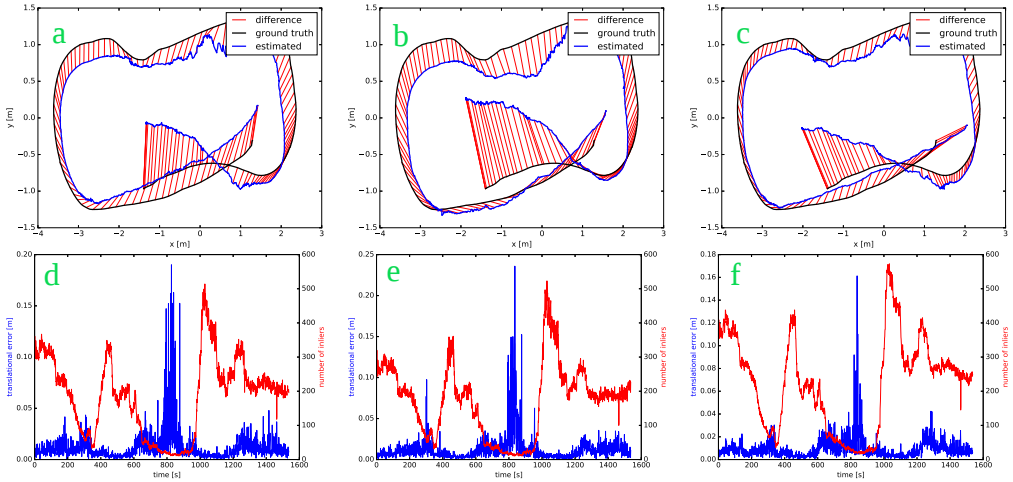
Rysunek 6.21: Wykresy błędów ATE (a–c) oraz (kolor niebieski) RPE (d–f) z naniesioną liczbą *ilnierów* biorących udział w ostatecznym oszacowaniu trajektorii (kolor czerwony) dla sekwencji *fr1_desk*. Parametry były albo (a, d) ustawiane ręcznie, albo (b, e) uzyskane w eksperymencie optymalizacyjnym OP1, albo (c, f) OP2

By uzupełnić ilościowe wyniki eksperymentu PSO, pokazano również jakościowe wyniki w formie trajektorii z naniesionymi błędami ATE. Rys. 6.21(a–c), 6.22(a–c), 6.23(a–c) pokazują wykresy ATE dla *fr1_desk*, *fr1_room* w drugim i dla sekwencji kontrolnej *putkk_Dataset_1_Kin_1* w trzecim.

Dość krótka sekwencja *fr1_desk* przedstawia dane zebrane za pomocą trzymanej w ręku kamery Kinect. Ponieważ nie była ona tak wymagająca jak pozostałe dwie, to znalezione optymalnych dla niej parametry nie uogólniają się dobrze na inne zestawy danych. Jednakże zestawy parametrów OP1 oraz OP2 pozwalają na uzyskanie podobnych metryk RPE. Sugeruje to, że błąd RPE nie jest wystarczająco dobrze rozróżniającą metryką w tym przypadku.



Rysunek 6.22: Wykresy błędów ATE (a–c) oraz (kolor niebieski) RPE (d–f) z naniesioną liczbą *ilnierów* biorących udział w ostatecznym oszacowaniu trajektorii (kolor czerwony) dla sekwencji *fr1_room*. Parametry były albo (a, d) ustawiane ręcznie, albo (b, e) uzyskane w eksperymencie optymalizacyjnym OP1, albo (c, f) OP2



Rysunek 6.23: Wykresy błędów ATE (a–c) oraz (kolor niebieski) RPE (d–f) z naniesioną liczbą *ilnierów* biorących udział w ostatecznym oszacowaniu trajektorii (kolor czerwony) dla sekwencji *putkk_Dataset_1_Kin_1*. Parametry były albo (a, d) ustawiane ręcznie, albo (b, e) uzyskane w eksperymencie optymalizacyjnym OP1, albo (c, f) OP2

Sekwencja (*fr1_room*) okazała się być o wiele bardziej wymagająca niż ta poprzednia. Co więcej, okazało się również, że manualnie dobrane parametry pozwalały na osiągnięcie lepszych wyników, niż parametry znalezione podczas optymalizacji PSO pracującej na sekwencji *fr1_desk*. Poza Tym, że obydwa te zestawy parametrów nie były w stanie sprostać wymaganiom tego zestawu, to osiągały podobne wyniki.

Jak widać w zestawie odniesienia *putkk_Dataset_1_Kin_1*, trajektorię da się dość dobrze odzyskać poza środkowym fragmentem–widać to zarówno na wykresach ATE jak i RPE. Mniej więcej w środkowej części eksperymentu, pojawiły się problemy podczas estymacji trajektorii metodą klatka po klatce przy korzystaniu z wszystkich zestawów parametrów. Powodem tego była niewielka liczba cech, którą można było uchwycić za pomocą kamer. Analiza klatek RGB wykazała, że wówczas kamera skierowana była na znajdujące się w pomieszczeniu szyby, przez które padało światło. Spowodowało to przyciemnienie rejestrowanego obrazu a to mogło spowodować błędne dopasowania punktów kluczowych. Tym niemniej zoptymalizowany zestaw parametrów OP2 uzyskał najlepsze wyniki. Wbrew temu, że parametry OP1 generowały największe błędy RPE, to wykresy ATE sugerują (mimo, że nie są to dobre parametry do VO), że uniknęły one powodowania dużych błędów orientacji skutkujących zejściem z trajektorii.

Wnioski

Niniejsze badanie pokazało, że możliwa jest automatyczna optymalizacja parametrów w systemie odometrii wizyjnej (VO) oraz, że (pod warunkiem wykorzystania do optymalizacji dostatecznie wymagającej sekwencji RGB-D) pozwala to na uzyskanie lepszych wyników niż ręcznie dobrane parametry. Parametry RANSAC, znalezione w różnych sekwencjach, są podobne i można je wykorzystać w innych zestawach danych. Inaczej jest w przypadku parametru dotyczącego detektora cech kluczowych–różni się dla różnych środowisk.

Jednakże procedura optymalizacyjna nie była w stanie znaleźć zestawu parametrów zapewniającego znacząco większą dokładność oszacowania trajektorii w sensie metryki ATE. Przyczyną tego zdają się być parametry RANSAC, które są pewnego rodzaju kompromisem pomiędzy dokładnością (mniejsze d_E) a liczbą cech punktowych wystarczającą do obliczenia poprawnej transformacji metodą klatka po klatce (większe d_E). Stąd wyda-

je się, że jesli to możliwe, to powinno unikać się podejścia typu RANSAC i osiągać pewniejsze oszacowanie transformacji poprzez miejscowe, lokalne zastosowanie metody regulacji wiązki BA (ang. *Bundle Adjustment*), tak jak zrobiono to w [184]. Z drugiej strony, pomocna byłaby procedura optymalizująca parametry odnoszące się do percepcyjnej części (np. detektora cech) na bieżąco, online, pozwalając na dostosowanie, adoptowanie się do zmian występujących na widzianej scenie. To był możliwy kierunek do przeprowadzenia badań, które też przeprowadzono i opisano w 6.6.6.

6.6.5 Wykorzystanie metod populacyjnych do poszukiwania parametru τ_A offline.

Opis eksperymentu

W następnych testach zdecydowaliśmy się ponownie wykorzystać parametry filtracji RANSAC, ponieważ dla różnych środowisk nieznacznie się zmieniały, a te uzyskane w procesie optymalizacji OP2 na sekwencji *fr1 _room* pozwoliły na uzyskanie najlepszych wyników.

W związku z tym, w następnych eksperymentach optymalizowaliśmy tylko parametr τ_A detektora AKAZE, ale przy wykorzystaniu dwóch różnych podejść inspirowanych naturą. W obydwu przypadkach proces kończył się po 20 iteracjach.

Wyniki

Tab. 6.14 pokazuje najlepsze wartości parametru τ_A uzyskane w wyniku optymalizacji za pomocą algorytmów PSO i EA z funkcjami celu bazującymi alternatywnie na mierze ATE bądź RPE oraz metodą przeszukania logarytmicznego opartej o miarę RPE. By badanie było kompletne, zdecydowano się również na optymalizację progu detekcji algorytmu SURF ale tylko z wykorzystaniem metody ewolucyjnej z funkcją celu opartej o miarę RPE, oraz metody przeszukania logarytmicznego, korzystającego z tej samej miary. Tab. 6.15 zawiera uzyskane wartości parametru. W tab. 6.16 zebrano wartości błędu systemu VO, jakie uzyskał korzystając z tychże parametrów.

Uzyskany próg detekcji τ_A jest znacząco niższy w przypadku korzystania w optymalizacji z ATE RMSE zamiast RPE RMSE. Pozwala zatem wziąć

Tabela 6.14: Parametr τ_A detektora AKAZE i jego optymalne wartości podług różnych wariantów optymalizacji

Metoda	PSO ATE	PSO RPE	EA ATE	EA RPE	AKAZE EX
próg detekcji	0,000623	0,001367	0,000703	0,001412	0,000830

Tabela 6.15: Parametr detektora SURF i jego optymalne wartości podług różnych wariantów optymalizacji

Metoda	SURF EA RPE	SURF EX
próg detekcji	883,547	322,866

pod uwagę więcej punktów podczas szacowania trajektorii, ale kosztem nieco większych tymczasowych błędów RPE. Znaleziony próg τ_A za pomocą metody przeszukania logarytmicznego jest podobny do tych odnalezionych za pomocą metod populacyjnych kierowanych błędem ATE.

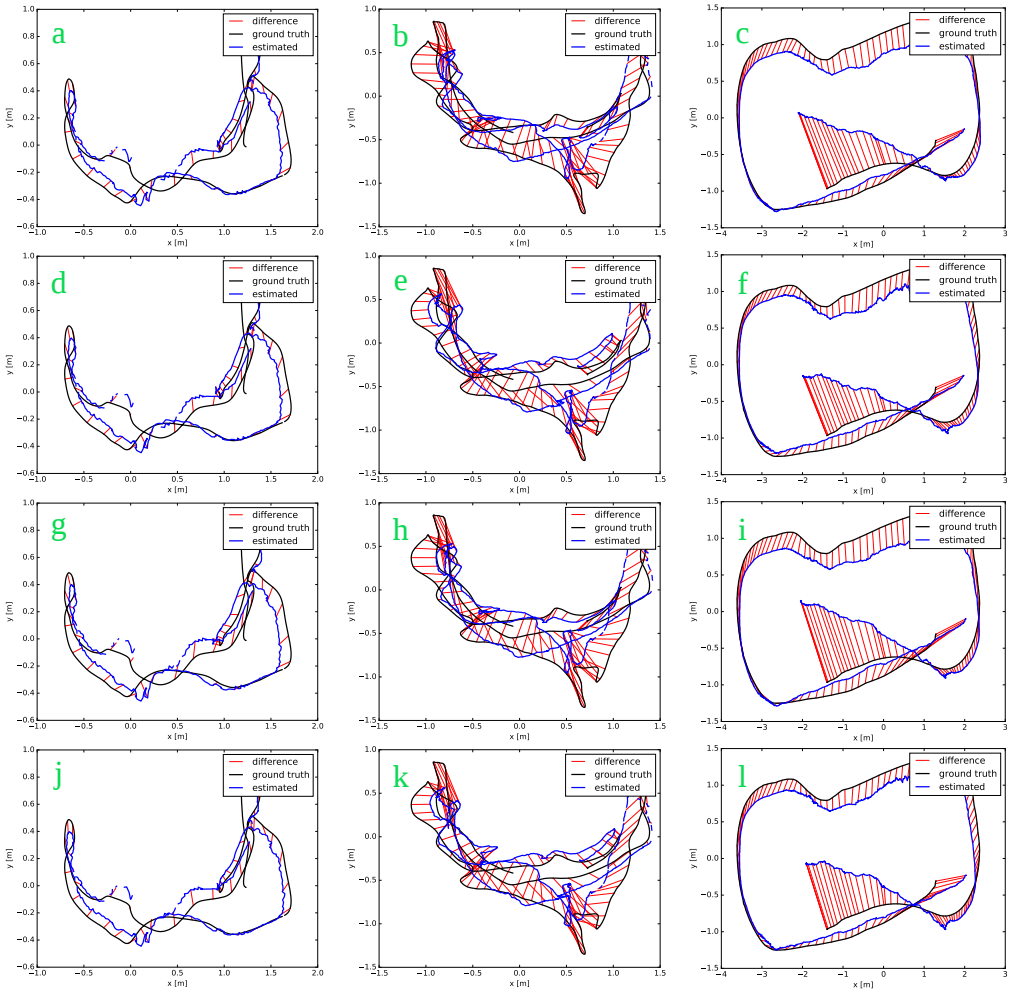
Jednakże nie pozwala to uzyskać znacząco lepszych wyników ATE RMSE podczas weryfikacji. Dlatego lepiej jest używać progu detekcji uzyskanego w optymalizacji bazującej na mierze RPE jako funkcji celu, ponieważ w ten sposób obliczenia są szybsze. W tab. 6.16 zawarto również wyniki uzyskiwane przez system VO z detektorem-deskryptorem SURF, którego próg detekcji optymalizowano z zastosowaniem tej samej procedury.

Całości porównania dopełniają wyniki uzyskane dla progów detekcji AKAZE i SURF znalezionych za pomocą metody przeszukania logarytmicznego. W przypadku algorytmu SURF są one lepsze niż te uzyskane za pomocą metod populacyjnych ale ustępują tym uzyskanym przy pomocy AKAZE. Jeśli chodzi o algorytm AKAZE, to uzyskane wyniki metodą przeszukania logarytmicznego są już porównywalne z tymi uzyskanymi metodami populacyjnymi.

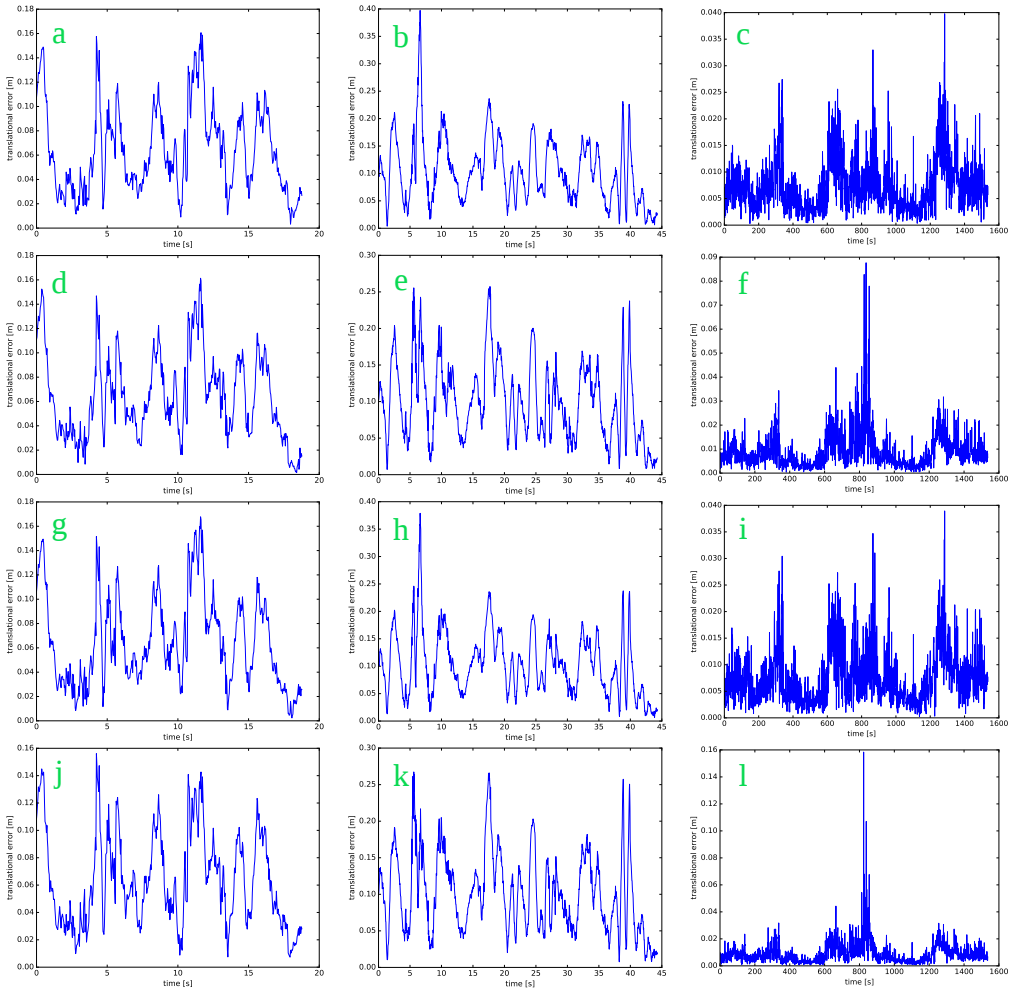
Pokazuje to, że nasze podejście można zastosować w systemach odometrii wizyjnej korzystających z innych parametrów. Rys. 6.24 i 6.26 pokazują porównanie odzyskanych trajektorii wizualizując błędy ATE, a rys. 6.25 i 6.27 stanowią ich ilościowe dopełnienie o „lokalną“ jakość estymacji trajektorii. Ogólnie rzecz ujmując, wszystkie zestawy parametrów wykorzystane w eksperymentach pozwoliły na poprawne oszacowanie trajektorii. Z tego wynika, że obydwie badane metody optymalizacji nadają się do automatycznego poszukiwania najlepszych wartości parametrów systemu odometrii wizyjnej RGB-D.

Tabela 6.16: Wyniki estymacji trajektorii dla τ_A będącego wynikiem optymalizacji

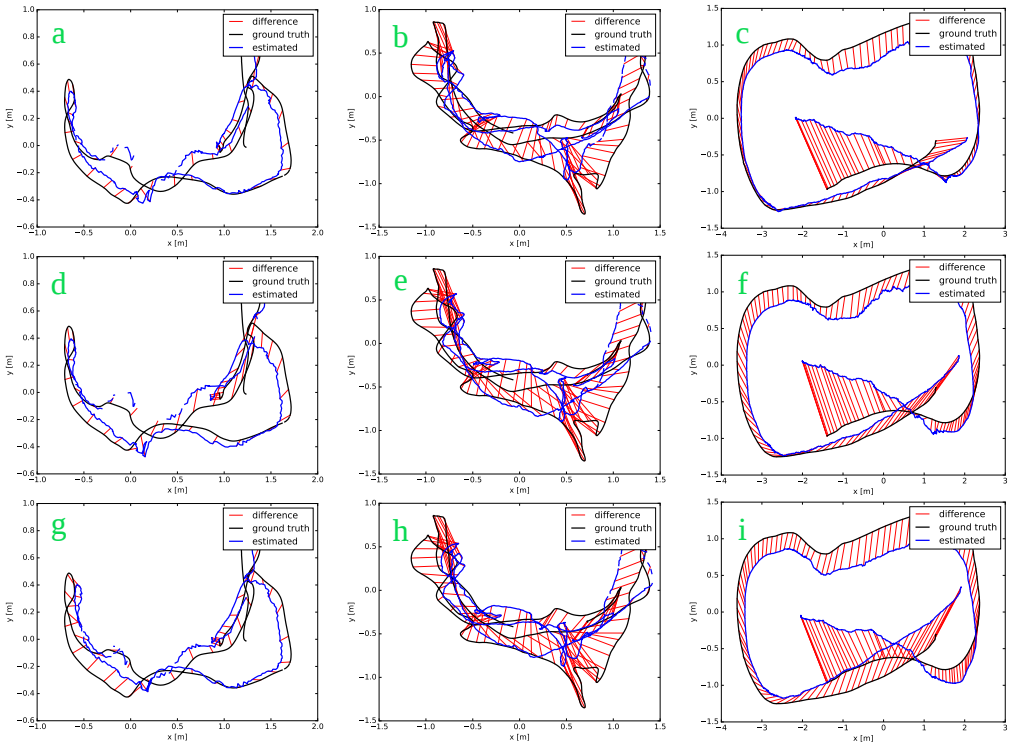
	fr1_desk						
Metryka błędu	PSO ATE	PSO RPE	EA ATE	EA RPE	AKAZE EX	SURF EA RPE	SURF EX
ATE RMSE [m]	0,109	0,105	0,126	0,095	0,098	0,127	0,108
Trans. RPE RMSE [m]	0,074	0,074	0,076	0,073	0,074	0,079	0,085
Rot. RPE RMSE [°]	2,470	2,455	2,477	2,483	2,442	2,581	2,573
	fr1_room						
Metryka błędu	PSO ATE	PSO RPE	EA ATE	EA RPE	AKAZE EX	SURF EA RPE	SURF EX
ATE RMSE [m]	0,290	0,292	0,295	0,312	0,305	0,382	0,366
Trans. RPE RMSE [m]	0,120	0,114	0,119	0,115	0,117	0,123	0,125
Rot. RPE RMSE [°]	2,403	2,585	2,382	2,633	2,368	2,691	2,607
	putkk_Dataset_1_Kin_1						
Metryka błędu	PSO ATE	PSO RPE	EA ATE	EA RPE	AKAZE EX	SURF EA RPE	SURF EX
ATE RMSE [m]	0,577	0,661	0,634	0,726	0,596	0,693	0,629
Trans. RPE RMSE [m]	0,009	0,012	0,009	0,012	0,009	0,010	0,008
Rot. RPE RMSE [°]	0,170	0,227	0,175	0,235	0,178	0,210	0,167



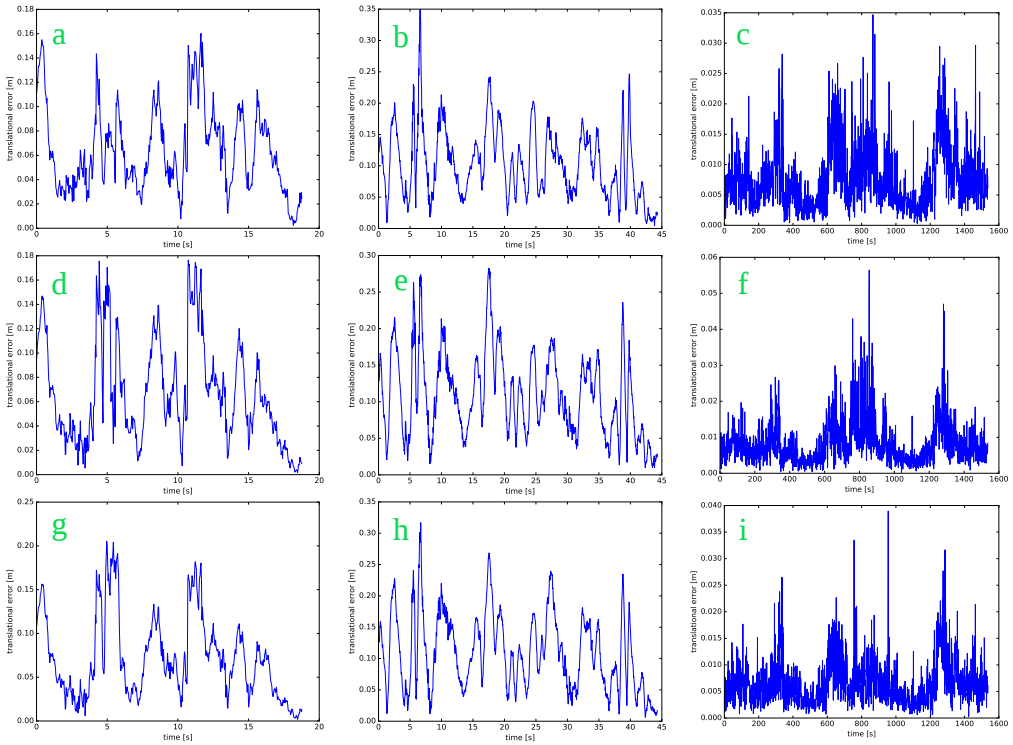
Rysunek 6.24: Wykresy błędów ATE (a, d, g, j) uzyskane dla sekwencji *fr1_desk*, (b, e, h, k) dla *fr1_room*, a (c, f, i, l) dla *putkk_Dataset_1_Kin_1*. Parametry były albo (a, b, c) wynikiem optymalizacji PSO ATE, (d, e, f) PSO RPE, (g, h, i) EA ATE, albo (j, k, l) EA RPE



Rysunek 6.25: Wykresy błędów translacji RPE (a, d, g, j) uzyskane dla sekwencji *fr1_desk*, (b, e, h, k) dla *fr1_room*, a (c, f, i, l) dla *putkk_Dataset_1_Kin_1*. Parametry były albo (a, b, c) wynikiem optymalizacji PSO ATE, (d, e, f) PSO RPE, (g, h, i) EA ATE, albo (j, k, l) EA RPE



Rysunek 6.26: Wykresy błędów ATE (a, d, g) uzyskane dla sekwencji *fr1_desk*, (b, e, h) dla *fr1_room*, a (c, f, i) dla *putkk_Dataset_1_Kin_1*. Parametry były albo (a, b, c) wynikiem przeszukania logarytmicznego AKAZE, (d, e, f) optymalizacji SURF EA RPE, albo (g, h, i) przeszukania logarytmicznego SURF



Rysunek 6.27: Wykresy błędów translacji RPE (a, d, g) uzyskane dla sekwencji *fr1_desk*, (b, e, h) dla *fr1_room*, a (c, f, i) dla *putkk_Dataset_1_Kin_1*. Parametry były albo (a, b, c) wynikiem przeszukania logarytmicznego AKAZE, (d, e, f) optymalizacji SURF EA RPE, albo (g, h, i) przeszukania logarytmicznego SURF

Wnioski

Opracowane oprogramowanie pozwala automatycznie dobrać parametry przy minimalnym zaangażowaniu użytkownika. Wyniki badań potwierdzają, że wyselekcjonowane parametry systemu VO związane z detekcją cech oraz

dopasowywaniem do siebie kolejnych klatek mają kluczowe znaczenie dla jakości estymat trajektorii.

Zaproponowana procedura optymalizacji jest wystarczająco uniwersalna, aby znajdować parametry odpowiednie

dla danego typu środowiska, a nie tylko danej trajektorii, co potwierdzają wyniki dla danych weryfikacyjnych. Pomyślne użycie procedury do optymalizacji parametru detektora SURF także wskazuje na jej uniwersalność.

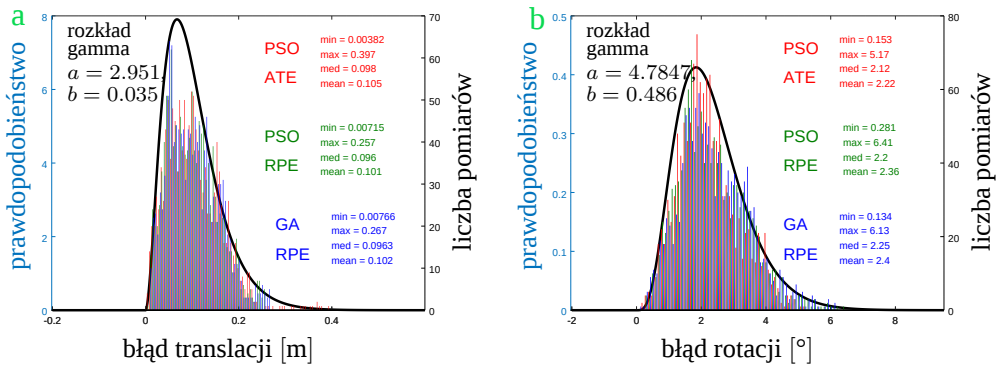
Przedstawione wyniki pokazują także, że optymalizacji parametrów systemu VO można dokonać na kilka różnych sposobów. Jednakże są pewne ważne różnice. Algorytm PSO przeszukał znacznie większy obszar przestrzeni parametrów. We wszystkich eksperymentach optymalizacja za pomocą PSO była ok. pięciokrotnie dłuższa niż z wykorzystaniem EA (58 godzin do 11 dla funkcji celu bazującej na ATE). Poza tym konfiguracje z funkcją celu bazującą na metryce RPE wymagały ok. 10% mniej czasu, niż te bazujące na metryce ATE.

We wszystkich tych przypadkach, program działał na Linuxie korzystając ze wszystkich dostępnych 12 wątków na procesorze $i5$ stacjonarnego komputera PC z blisko 100% obciążeniem procesora. Biorąc pod uwagę czasy obliczeń i podobne wyniki zalecamy algorytm ewolucyjny EA z funkcją celu bazującą na metryce RPE jako poręczne narzędzie programistyczne do szybkiego szukania sensownych parametrów systemu VO. Jednakże jeśli kogoś interesuje dokładniejsza optymalizacja (np. w przypadku konkretnego środowiska, które nie zmienia się znacznie w czasie), to zalecamy algorytm PSO z uwagi na jego lepsze właściwości poszukiwawcze. Zdaje się, że funkcja celu bazująca na metryce ATE nie ma większych pozytywnych efektów na ostateczny wynik, dlatego polecamy tą bazującą na RPE dla obydwu metod optymalizacyjnych, bo pozwala na szybsze obliczanie wartości dopasowania.

Analiza statystyczna

Wykreślono także histogramy błędów translacji i rotacji dla miary RPE– rys. 6.28a przedstawia błędy przesunięcia, natomiast rys. 6.28b błędy rotacji dla sekwencji *fr1_room*. Histogramy te pozwalają oszacować rozkłady gęstości prawdopodobieństwa (ang. *probability density function*–PDF) zmiennych określających te błędy. Kolor czerwony i zielony, na tym i kolejnych rysunkach z histogramami, dotyczy błędów częściowych najlepszej trajektorii otrzymanej w trakcie optymalizacji algorytmem rojowym z wykorzystaniem metryki ATE RMSE, bądź RPE RMSE. Niebieski został zarezerwowany dla najlepszych rezultatów osiąganych przez algorytmu ewolucyjnego korzystającego z metryki RPE RMSE.

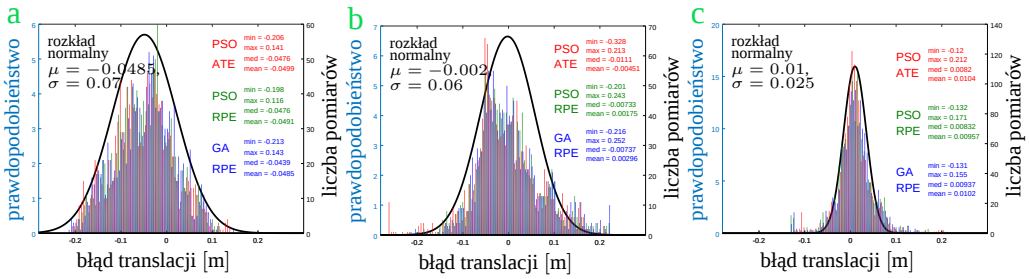
Obydwa histogramy dotyczące błędów translacji i rotacji przypominają rozkład normalny, jednak z wyraźną asymetrią. Wynika to z statystycznej złożoności miary RPE, zależnej nie tylko od przesunięć, ale i obrotów, ponieważ do obliczania przesunięcia metodą klatka po klatce (algorytm Kabscha) wymagana jest znajomość rotacji pomiędzy tymi klatkami. Sama zaś część translacyjna tworzona jest za pomocą sumy kwadratów zmiennych losowych o niestandardowych rozkładach normalnych, tj. posiadają $\mu \neq 0, \sigma^2 \neq 1$, które w dodatku nie muszą być jednakowe dla każdej z osi x, y, czy z. W związku z powyższymi argumentami, do opisu parametrycznego skorzystano z ciągłego rozkładu gamma, który dostatecznie dobrze przybliża ten trudny do wyznaczenia analitycznie rzeczywisty rozkład.



Rysunek 6.28: Histogramy: a) błędów translacji i b) błędów rotacji dla trajektorii oszacowanych przez RGB-D VO z automatycznie dobranymi parametrami z *fr1_room*. Wyniki dla różnych metod doboru parametrów wyróżniono kolorami (odcieniami szarości)

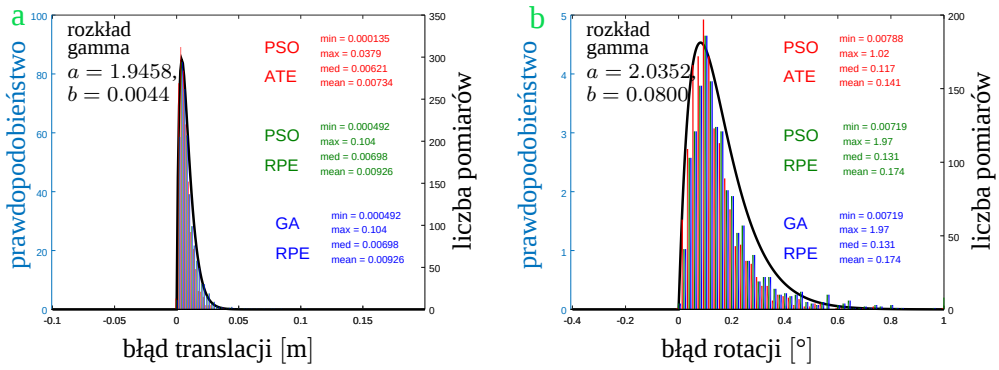
Dodatkowo na rys. 6.29 wykreślono histogramy składowych błędów RPE, czyli błędów w osi x, y oraz z, a więc w tych, w których robot się poruszał. Widać, że rzeczywiste rozkłady nie układają się perfekcyjnie pod krzywą Gaussa. Najprawdopodobniej wynika to z występowania w środku długości optymalizowanej sekwencji szeregu ubogich w cechy klatek obrazu RGB, co spowodowało oszacowanie translacji i rotacji na podstawie znacznie mniejszej liczby cech. Kolejną przyczyną może być drobna niedoskonałość zestawu danych *TUM RGB-D*–nie wymuszano tam robienia zdjęć RGB oraz D w tym samym czasie ani nie zsynchronizowano tego z systemem motion-capture. Kamera była trzymana w ręce i dominował jeden kierunek obrotu. Nie wyklucza to, że występują również inne przyczyny.

Co interesujące, wszystkie mediany błędów dla poszczególnych osi x, y i z, składających się na błąd RPE RMSE translacji, są albo ujemne albo bliskie zeru. Może to oznaczać, że podane w zestawie danych parametry kalibracji sensora, w szczególności stała użyta do przeliczania liczb całkowitych podanych w mapach głębi na metry, nie są idealne i kalibrację można jeszcze poprawić, np. włączając jej parametry do zbioru wartości optymalizowanych.



Rysunek 6.29: Histogramy składowych błędu translacji w osi a) x b) y i c) z dla trajektorii oszacowanych przez RGB-D VO z automatycznie dobranymi parametrami z *fr1_room*. Wyniki dla różnych metod doboru parametrów wyróżniono kolorami (odcieniami szarości)

W celu porównawczym, ponowną analizę wykonano dla innego zestawu danych: *putkk_Dataset_1_Kin_1*. W tym przypadku obydwa histogramy 6.30 udało się przybliżyć za pomocą rozkładu gamma o odpowiednio dobranych parametrach. Obydwa błędy RPE są o rząd wielkości mniejsze.



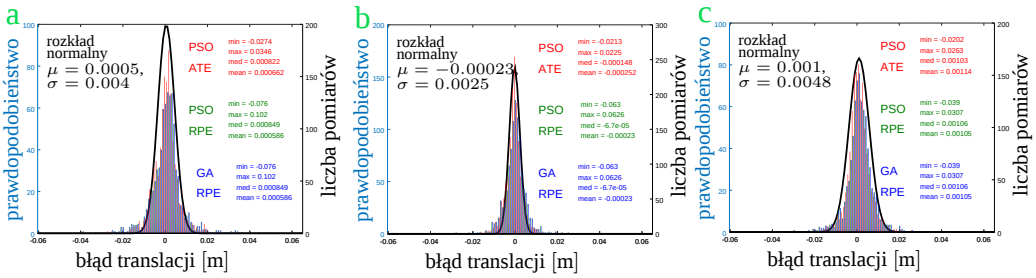
Rysunek 6.30: Histogramy: a) błędów translacji i b) błędów rotacji dla trajektorii oszacowanych przez RGB-D VO z automatycznie dobranymi parametrami z *putkk_Dataset_1_Kin_1*. Wyniki dla różnych metod doboru parametrów wyróżniono kolorami (odcieniami szarości)

Dlatego zestawu również wykreślono histogramy (rys. 6.31) błędów w osi x, y oraz z, będących składowymi błędów RPE. Wszystkie bardzo dobrze układają się pod krzywą Gaussa i są także o rząd wielkości mniejsze, niż te z sekwencji *fr1_room* (a robot również tu obracał się w jedną stronę).

Wziąwszy pod uwagę, że mamy do czynienia z robotem kołowym, poruszającym się po podłodze można by założyć stałą wysokość w tej osi. Jednakże kamera zamocowana jest na szczycie robota, za pomocą plastikowego uchwytu, który może drgać również przy obrotach. Kolejnym źródłem drgań mogą być kable (zaledwie kilka) na podłodze czy inne bardzo drobne nierówności. Tak czy inaczej, można by założyć, że błędy w tej osi są głównie losowe, co zresztą widać na histogramie 6.31c–krzywa Gaussa dość dobrze modeluje rzeczywisty rozkład prawdopodobieństwa.

Wnioski

Powyższe histogramy ukazują, że przy znacznej ilości danych, rozkłady prawdopodobieństwa względnych błędów RPE oszacowania przemieszczenia sensora nie są idealnie Gaussowskie. Sugeruje to istnienie dominującego źródła błędów lub błędów systematycznych, które można by próbować usunąć, np. włączając parametry kalibracji sensora do wektora wartości optymalizowanych. Można to wykorzystać przy ocenie ufności co do proponowanych przez algorytm RANSAC wstępnych rozwiązań lub



Rysunek 6.31: Histogramy składowych błędu translacji w osi a) x b) y i c) z dla trajektorii oszacowanych przez RGB-D VO z automatycznie dobranymi parametrami z *putkk_Dataset1_Kin_1*. Wyniki dla różnych metod doboru parametrów wyróżniono kolorami (odcieniami szarości)

w algorytmach typu SLAM przy optymalizacji trajektorii. W tym samym celu można by również wykorzystać informacje na temat średnich błędów przesunięcia według konkretnej osi, jednak w tym celu należałoby zbadać pod tym kątem znacznie większą liczbę sekwencji.

6.6.6 Wykorzystanie metod populacyjnych do poszukiwania najlepszych parametrów systemu odometrii wizyjnej RGB-D w zagadnieniu online

Opis eksperymentu

Uzyskane w eksperymentach offline wyniki zachęciły nas by pójść o krok dalej i spróbować nastroić parametry detektora AKAZE online, tak by umożliwić systemowi odometrii przystosowywanie się do zmieniających się warunków takich jak zmienne naświetlenie czy ilość występujących w środowisku tekstur, która wprost wpływa na liczbę wykrywanych punktów kluczowych.

Poza progiem detekcji τ_A, optymalizowaliśmy również parametr odpowiedzialny za percentyl τ_P. We wstępnych badaniach zauważyliśmy, że oszacowana trajektoria była znacznie gorsza, gdy biorących udział w estymacji punktów kluczowych, po filtracji algorytmem RANSAC, było mniej niż 250. W związku z tym zdecydowaliśmy się na wykorzystanie algorytmu PSO do poszukiwania parametrów pozwalających na wykrycie odpowiedniej liczby cech w tych trudniejszych ujęciach. Ze względu na

szybkość działania optymalizację rozpoczynaliśmy, gdy na obrazie zostało wykrytych mniej niż 50 punktów kluczowych.

W eksperymencie jako miarę błędu wykorzystano metrykę RPE, bo odzwierciedla ona lokalne zmiany oszacowanej trajektorii. Tutaj także wykorzystaliśmy 40 cząstek ale pierwsza cząstka została zarezerwowana dla obecnie najlepszego wektora parametrów, znalezionego podczas optymalizacji offline. Kolejne 11 (wykorzystywany komputer mógł uruchomić 12 wątków równolegle) cząstek stanowiły zestaw uprzednio najlepszych parametrów. Są one liczone tylko raz, gdy jest zbyt mało punktów kluczowych. Wtedy do tej puli dodaje się tylko 1 najlepszy wektor, a po wyczerpaniu wolnych miejsc, zastępowany jest ten który wykrył najmniej punktów.

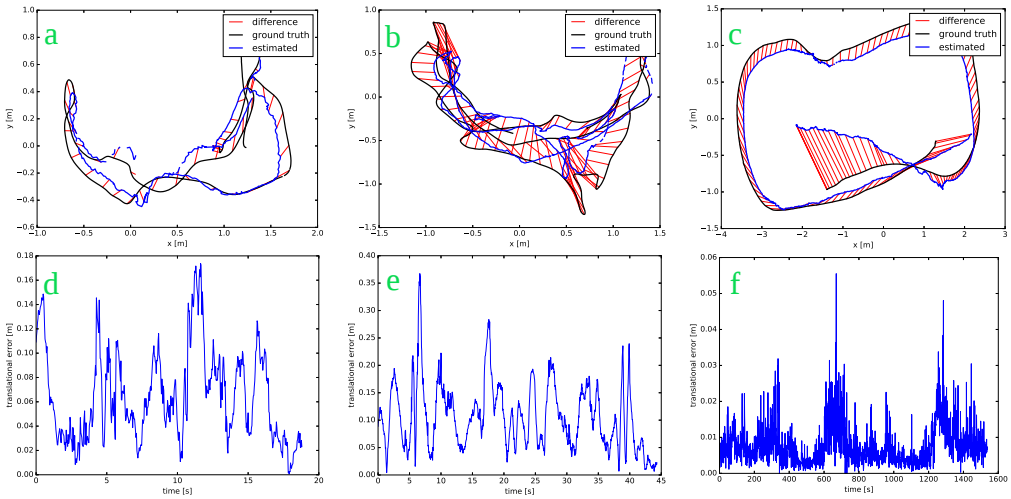
Wyniki

W tabeli 6.17 przedstawiono wyniki dla trzech sekwencji, uprzednio wykorzystywanych w optymalizacji offline. Rys. 6.32 ukazuje odpowiednie wykresy błędów ATE i RPE.

Tabela 6.17: Wyniki estymacji trajektorii uzyskanej przez optymalizację online parametrów detektora AKAZE

sekwencja	ATE [m]	Trans. RPE [m]	Rot. RPE [°]
fr_desk	0,1225	0,0758	2,5094
fr_room	0,3233	0,1212	2,5794
putkk_Dataset_1_Kin_1	0,5000	0,0097	0,1739

Mimo zmiany progu detekcji algorytmu AKAZE online za pomocą procedury PSO, ogólnie nie udało się osiągnąć poprawy wyników ATE RMSE i RPE RMSE w stosunku do najlepszych wyników optymalizacji offline. Nieznaczną poprawę uzyskano tylko dla sekwencji *putkk_Dataset_1_Kin_1*. Jest to stosunkowo płynna sekwencja cechująca się stałymi warunkami oświetlenia, co sugeruje, że populacyjna metoda optymalizacji jest zbyt wolna, by dostosować parametry online do nagłych zmian w przychodzących danych.



Rysunek 6.32: Wyniki eksperymentów optymalizacji online trzech sekwencji: (a, d) *fr1_desk*, (b, e) *fr1_room*, i (c, f) *putkk_Dataset1_Kin_1*

Wnioski

Powyższe badania pokazują jak proste w implementacji populacyjne metody optymalizacji można wykorzystać w zadaniu poszukiwania najlepszych parametrów dla systemu odometrii wizyjnej RGB-D. Zaproponowane procedura może zastąpić wyczerpujące poszukiwania, które są czasami wykorzystywane do znalezienia parametrów gwarantujących najlepsze wyniki na popularnych sekwencjach benchmarkowych. Pomimo tego, że jej największą zaletą jest zwiększona szybkość (godziny w porównaniu do kilku dni), pokazaliśmy również, że tak zoptymalizowane parametry można z powodzeniem stosować również w innych sekwencjach, tak długo jak przedstawiają one podobne środowiska i dynamiki kamery.

Pokazano również, że parametry algorytmu RANSAC (które są tak szeroko stosowane w VO i SLAM) są bardzo ważne dla zapewnienia odpowiedniego kompromisu pomiędzy liczbą cech punktowych przetwarzanych na potrzeby obliczania transformacji klatka po klatce i dokładności ich dopasowywania.

Przedstawiono nowatorskie podejście optymalizacji online wybranych parametrów VO, których zmian można by się spodziewać wraz ze zmianami w obserwowanym środowisku. Podczas gdy wstępne wyniki optymalizacji online pokazują nieznaczną poprawę, nadal istnieje sporo miejsca

na dalsze prace badawcze tym podejściem z wykorzystaniem szybszych metod wyszukiwania parametrów. Wydajność online można również poprawić poprzez masowo równoległą implementację standardowego algorytmu PSO na GPU, ale nie uważamy tego za wykonalne podejście dla VO na pokładzie robota mobilnego ze względu na ograniczenia zużycia energii.

6.6.7 Użycie klasycznych metod do uzupełniania mapy głębi

Jednym z większych problemów związanych z wykorzystaniem sensorów pomiaru głębi, podobnych do Kionecta v1, są niekompletne i zaszumione mapy głębi z (często z obecnymi artefaktami) jakie one rejestrują. Powodem braku pomiarów dla, części pikseli, są odbijające, przeźroczyste bądź nieregularne powierzchnie [135]. To wszystko ma wpływ na odometrię wizyjną VO.

Metody uzupełniania głębi wychodzą tym problemom na przeciw, choć większość prac nad uzupełnianiem głębi związana jest z zastosowaniami takimi jak rekonstrukcja $3D$ [93].

Opis eksperymentu

Dotychczas prace skupiały się na znalezieniu wyznaczeniu przebytej przez kamerę robota drogi i optymalizacji parametrów systemu. W tym eksperymencie sprawdzano jaki wpływ może mieć uzupełnianie brakujących danych mapy głębi na jakość odzyskiwanej trajektorii. Ponieważ zmienia to postawione zadanie, również i tu dokonujemy optymalizacji PSO i EA.

Z uwagi na to, że w dalszych planach oprócz wykorzystania metod klasycznych; tj. algorytmów Telei [229] czy Naviera-Stokesa [105], planowano skupić się na wykorzystaniu sztucznej sieci neuronowej, to w celu przyspieszenia badań postanowiono zmniejszyć liczbę iteracji algorytmu RANSAC (po których następowało zwiększenie akceptowalnej odległości poniżej której dany punkt był uznawany za typu *inlier*) z 10000, ponieważ wstępne eksperymenty wykazały znaczne wydłużenie czasu obliczeń w przypadku wykorzystania uzupełnionych map głębi.

Wykorzystując 2.27 oraz najlepsze znalezione poprzednio (w eksperymencie OP2) stosunki *inlierów* do *outlierów* $\Gamma_{o,1} = 0,916089$ oraz $\Gamma_{o,2} = 0,854762$, to zakładając prawdopodobieństwo sukcesu $p = 0,9999$ (choć

standardowo niektórzy [248] proponują $0,99$), możemy zmniejszyć maksymalną liczbę iteracji do 7 i 10 (odpowiednio dla pierwszej i drugiej filtracji). Z uwagi na to, że po odrzuceniu *outlierów* zebrane dane również nie są jeszcze perfekcyjne (bo występują np. zakłócenia pomiarowe i inne niedokładności), by zwiększyć szansę na wybór tych lepszych zwiększono liczbę iteracji do $k_1 = 21$ oraz $k_2 = 30$.

Choć tak dobrana liczba iteracji pozwalała na poprawne odzyskanie przebytej przez kamerę trajektorii, to niestety uzyskiwane wyniki ATE/RPE cechowała zauważalna wariancja. W związku z tym postanowiono jeszcze zwiększyć liczbę losowań algorytmu RANSAC. Po przeprowadzeniu wstępnych badań zdecydowano się na $k_1 = 315$ oraz $k_2 = 450$.

Uzupełnianie danych było wykonywane *offline*. Oznacza to, że przed rozpoczęciem wyznaczania trajektori zamieniono obrazy głębi na te pozbawione „dziur", przy okazji zmieniając liczbę jednostek przypadających na 1[m] z 1000 na 5000. Ta zamiana miała pozwolić na zwiększenie rozdzielczości w odległościach występujących w pomieszczeniach jak i tych, dla których Kinect jest w stanie dostarczyć dokładne pomiary.

W badaniach wykorzystano dostępne w bibliotece OpenCV implementacje algorytmów Telei i Naviera-Stokesa. Za promień inpaintingu odpowiada zmienna `InpaintRadius`. Choć ta zmienna jest typu `double`, to na wstępie jest ona zaokrąglana do liczby całkowitej (`int`) i ograniczana do zakresu $[1, 100]$. Implementacja w języku C++ nie posiada przypisanej do niej wartości domyślnej, jednak wersja napisana w języku Python już posiada i przypisuje jej wartość 3. Badania przeprowadzono dla promienia o długościach 1, 2, 3, 4 i 5 piksela.

Wyniki

Do wstępnych badań posłużono się sekwencją *putkk_Dataset_5_Kin_1*. Jest ona wystarczająco długa, by zawrzeć w sobie interesujące, stwarzające problemy cechy. Tab. 6.18 zawiera wyniki ATE oraz RPE uzyskiwane przez system odometrii wizyjnej, pracujący z parametrami uzyskanymi w optymalizacji PSO ATE, który pracował na danych z sekwencji *putkk_Dataset_5_Kin_1*, gdzie dane głębi były albo nieuzupełnione, albo uzupełniane przy pomocy algorytmu Naviera-Stokesa (oznaczone jako NS) bądź Telei z promieniami inpaintingu od 1 do 5. Taki zakres wydawał się najrozsądniejszy gdyż, jak wykazała inspekcja wizyjna, na mapach głębi brakowało danych głównie w obszarach gdzie znajdowały się cienkie rurki (np. nóżki krze-

seł), inne wąskie przedmioty, złożone strukturalnie obiekty, ciemniejsze fragmenty obrazu, przysłonięcia czy dalsze odległości.

Tabela 6.18: Wyniki estymacji trajektorii bez wypełniania map głębi oraz z uzupełnianiem za pomocą algorytmów Telei i Naviera-Stokesa przy różnych promieniach inpaintingu

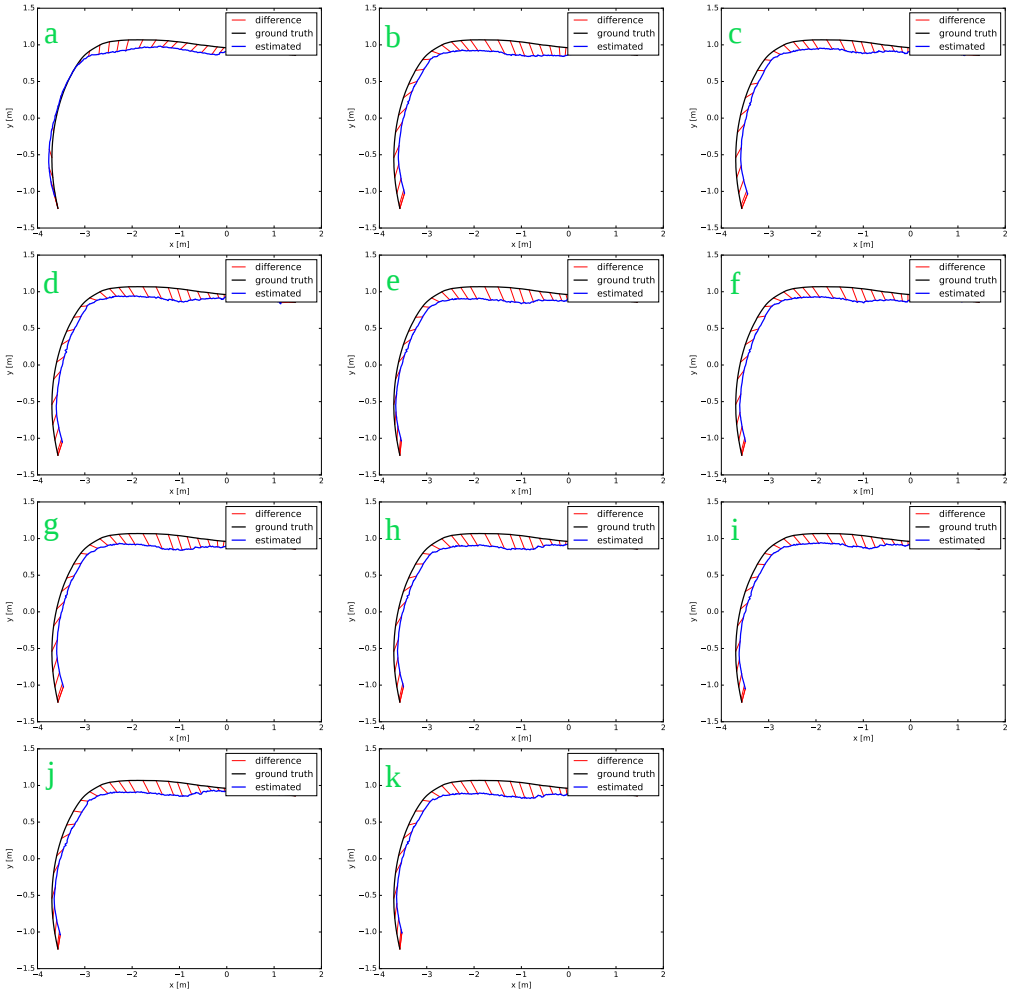
	putkk_Dataset_5_Kin_1					
Metryka błędu	bez wypełniania	NS 1	NS 2	NS 3	NS 4	NS 5
ATE RMSE [m]	0,198	0,234	0,203	0,201	0,207	0,220
Trans. RPE RMSE [m]	0,012	0,011	0,011	0,011	0,011	0,011
Rot. RPE RMSE [°]	0,174	0,175	0,178	0,187	0,182	0,178
Metryka błędu		Telea 1	Telea 2	Telea 3	Telea 4	Telea 5
ATE RMSE [m]		0,221	0,222	0,204	0,212	0,231
Trans. RPE RMSE [m]		0,011	0,011	0,012	0,011	0,011
Rot. RPE RMSE [°]		0,177	0,179	0,183	0,190	0,186

Jak widać z danych zebranych w tab. 6.18 najlepsze wyniki uzyskuje się nie stosując żadnego uzupełniania danych, choć te uzyskane przez NS $2-4$ oraz Telea 3 i 4 znacznie nie odbiegają (należy pamiętać, że wyniki same w sobie cechują się drobną losowością z uwagi na RANSAC). Wyniki numeryczne uzupełniono również o adekwatne wykresy ATE oraz RPE, przedstawione na rys. 6.33 i rys. 6.34.

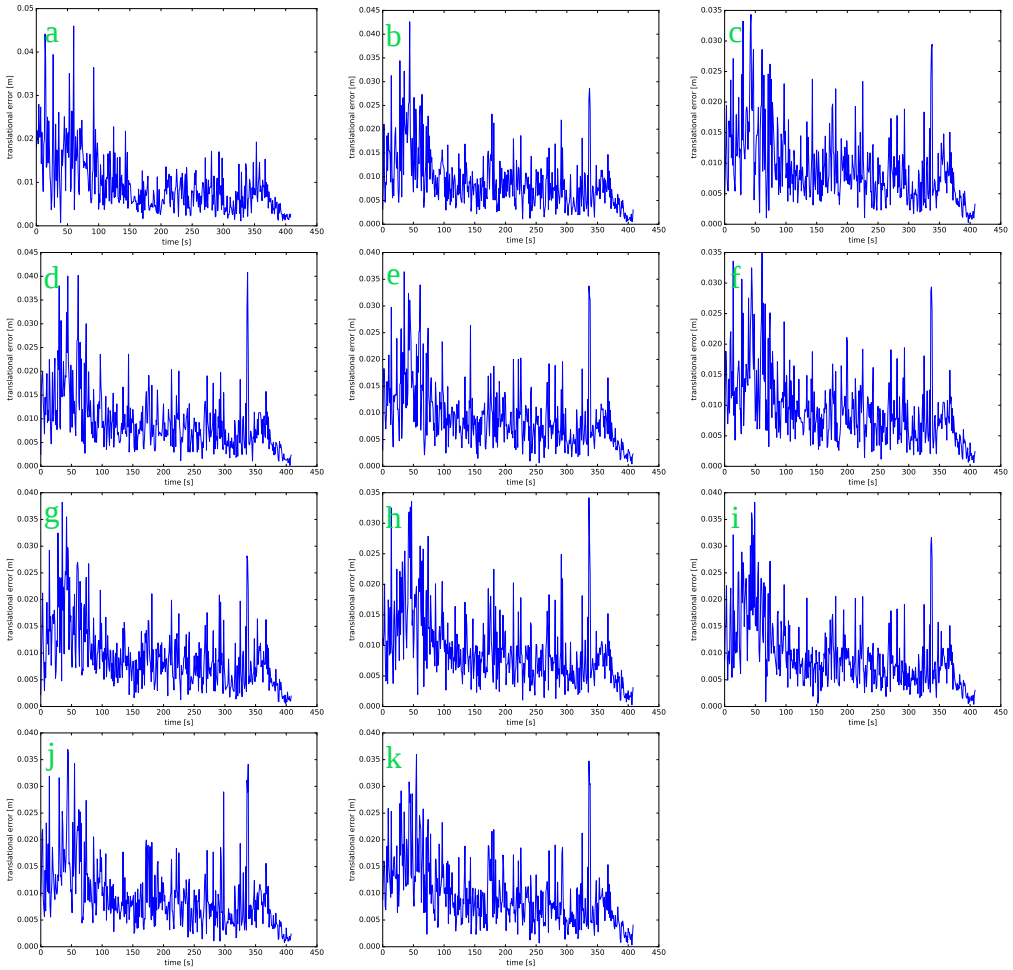
Jak widać na rys. 6.33 oraz rys. 6.34 wszystkie badane metody uzupełniania dziur w mapie głębi pozwoliły na poprawne odtworzenie przebytej przez kamerę trajektori oraz dostarczyły danych, w których miejscach powstawały największe względne błędy. Obrazy RGB w początkowej fazie ujawniają, że kamera rejestrowała obiekty znajdujące się od niej dalej, niż to było w końcowej fazie ruchu.

Z uwagi na to, że najlepsze wyniki uzupełniania osiągnięto dla promienia 3 (który również jest tym domyślnym w OpenCV), to w kolejnych badaniach wykorzystano właśnie go. Dalej, w celach porównawczych skorzystano z trajektorii *fr1_desk*, *fr1_room* oraz *putkk_Dataset_1_Kin_1*. Dane numeryczne uzupełniono o wyniki uprzednio uzyskane bez uzupełniania dla PSO ATE i zebrano w tabeli 6.19

Na podstawie tab. 6.19 wyraźnie widać, że bez ingerencji w dane głębi uzyskuje się lepsze wyniki, choć wyjątkiem jest sekwencja *putkk_Dataset_1*



Rysunek 6.33: Wykresy błędów ATE. (a) Uzyskano bez inpaintingu, (b-f) uzupełniając braki algoryrtmem Naviera-Stokesa o promieniach $1-5$, (g-k) uzupełniając braki algoryrtmem Telei o promieniach $1-5$



Rysunek 6.34: Wykresy błędów translacji RPE. (a) Uzyskano bez uzupełniania, (b-f) uzupełniając braki algorytmem Naviera-Stokesa o promieniach $1-5$, (g-k) uzupełniając braki algorytmem Telei o promieniach $1-5$

Tabela 6.19: Wyniki estymacji trajektorii bez wypełniania map głębi oraz z uzupełnianiem za pomocą algorytmów Telei i Naviera-Stokesa przy promieniu inpaintingu 3

	fr1_desk			*fr1_room*		
Metryka błędu	bez wypełniania	NS 3	Telea 3	bez wypełniania	NS 3	Telea 3
ATE RMSE [m]	0,109	0,131	0,123	0,290	0,369	0,343
Trans. RPE RMSE [m]	0,074	0,080	0,077	0,120	0,130	0,128
Rot. RPE RMSE [°]	2,470	2,721	2,613	2,403	2,656	2,589

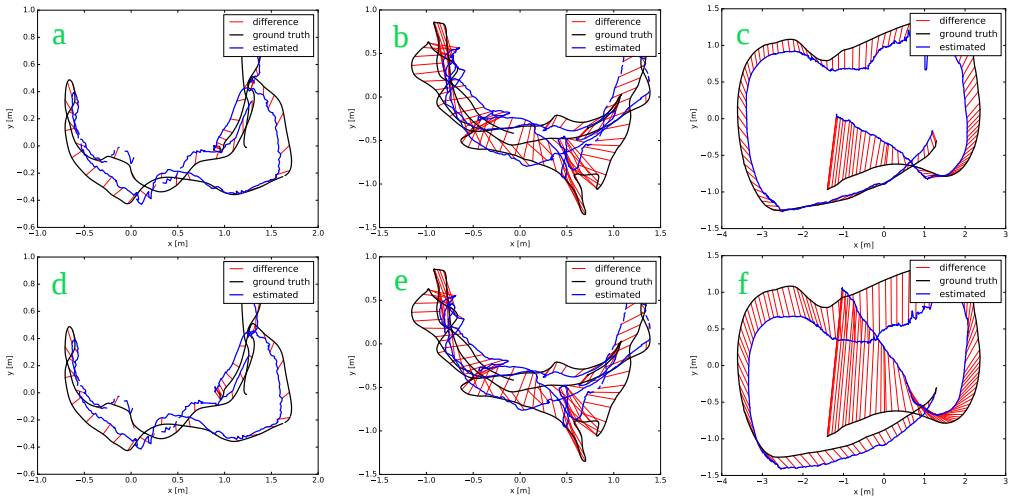
	putkk_Dataset_1_Kin_1		
Metryka błędu	bez wypełniania	NS 3	Telea 3
ATE RMSE [m]	0,577	0,476	0,820
Trans. RPE RMSE [m]	0,009	0,022	0,018
Rot. RPE RMSE [°]	0,170	0,542	0,428

_Kin_1, w której algorytm Naviera-Stokesa uzyskał trochę lepsze wyniki. Wyniki numeryczne również uzupełniono o adekwatne wykresy ATE oraz RPE (rys. 6.35 i 6.36) pomijając jednak te bez inpaintingu.

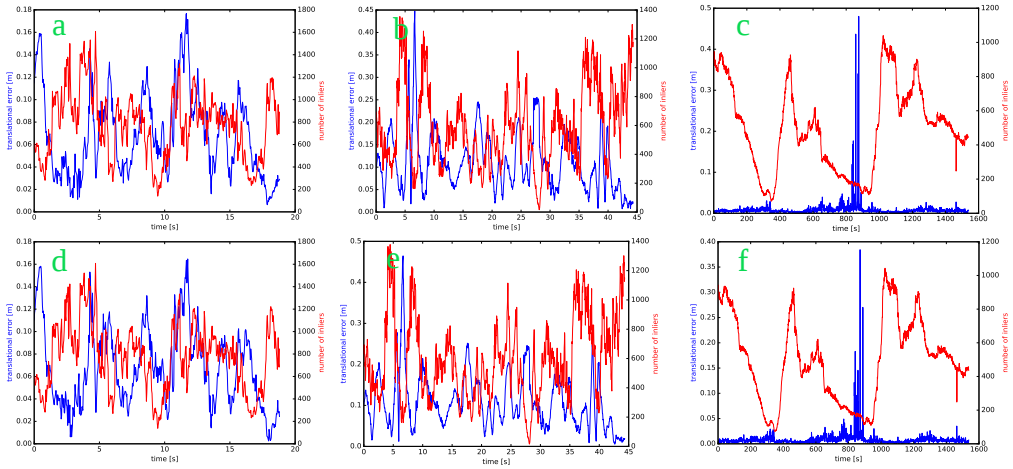
Jak widać z wykresów błędów ATE rys. 6.35 tu również udało się odtworzyć przebyte trajektorie. Widać jedynie znaczny wzrost błędu dla *putkk _Dataset_1_Kin_1* w przypadku algorytmu Telei. Dość reprezentatywne są wykresy RPE z naniesioną liczbą punktów kluczowych 6.36. Jeśli porówna się je z poprzednimi (np.:6.21(d–f), 6.22(d–f), 6.23(d–f), 6.25, 6.27), to widać poprawę w odzyskiwaniu trajektrori dla *putkk_Dataset_1_Kin_1* poza środkowym fragmentem.

Wnioski

To badanie wykazało, że w niektórych zastosowaniach uzupełnianie danych może przynieść korzyści, jednakże takie proste uzupełnianie nie zawsze zdaje egzamin, a nieraz może być źródłem pogorszenia wyników.



Rysunek 6.35: Wykresy błędów ATE. (a, d) Uzyskano dla sekwencji *fr1_desk*, (b, e) dla *fr1_room*, a (c, f) dla *putkk_Dataset_1_Kin_1*. Mapę głębi uzupełniano z wykorzystaniem promienia inpaintingu o długości 3 (a, b, c) algorytmem Naviera-Stokesa, albo (d, e, f) Telei



Rysunek 6.36: Wykresy błędów translacji RPE (kolor niebieski) z naniesioną liczbą *ilnierów* biorących udział w ostatecznym oszacowaniu trajektorii (kolor czerwony). (a, d) uzyskano dla sekwencji *fr1_desk*, (b, e) dla *fr1_room*, a (c, f) dla *putkk_Dataset_1_Kin_1*. Mapę głębi uzupełnianio z wykorzystaniem promienia inpaintingu o długości 3 (a, b, c) algorytmem Naviera-Stokesa, albo (d, e, f) Telei

6.6.8 Wykorzystanie sztucznej sieci neuronowej do uzupełniania mapy głębi

Z biegiem lat powstały nowsze, bardziej zaawansowane kamery, jak choćby kolejne wersje samego Kinecta: Kinect v2 czy Kinect Azure, pozwalające na uzyskanie lepszych wyników. Z uwagi na rozwój oraz wyniki jakie pozwalają uzyskać nowe metody wykorzystujące sztuczne sieci neuronowe (szczególnie te głębokie) postanowiono sprawdzić, czy wykorzystanie ich pozwoli na uzyskanie lepszych estymat trajektorii, poprzez ulepszenie map głębi, poprzez zawarcie danych widzianych przez nowszą kamerę Kinect v2. W tym celu skorzystano z architektury Monodepth [132]. Ta architektura jest odpowiednia do dynamicznych środowisk, ale jednocześnie wydajna w przetwarzaniu i zajmująca mało miejsca w pamięci [125], więc jest rozsądnym wyborem do praktycznych zastosowań w budżetowych robotach poruszających się wewnątrz budynków.

W pracy Castra i innych [110] zastosowano podobne podejście do tego wykorzystanego w przedstawionych badaniach. Także wykorzystano strukturę U-net, po którym uzupełniano mapy głębi z budżetowego sensora RGB-D za pomocą modułu udoskonalającego. Jednak do uczenia wykorzystują inną funkcję błędu: składającą się z ważonego, średniego absolutnego błędu euklidesowego, błędu gradientu oraz błędu SSIM. W procesie nauczania nie są wykorzystywane mapy głębi pobrane bezpośrednio z lepszego sensora, a oryginalne odpowiednio uzupełnione. Z uwagi na niekompletność map głębi dostarczanych przez Kinecta v1, ponieważ kamera głębi widzi mniej niż RGB, wycięto krańcowe piksele. Kamera Kinecta v2 posiada większe pole widzenia od Kinecta v1, więc mogliśmy z tego nadmiaru danych zrezygnować, zachowując oryginalny rozmiar obrazów.

Obecnie prowadzone są badania nad wykorzystaniem estymowanych map głębi w systemach VO czy SLAM. W [228] robione jest to na podstawie danych monokularowych. Do oszacowania głębi oraz niepewności wykorzystywana jest sieć konwolucyjna. System DeepVO [235] wykorzystuje głębokie uczenie do łącznej estymacji mapy głębi oraz pozycji robota. Otrzymane w ten sposób mapy pozwalają na ulepszenie rozpoznawania zamknięcia pętli, optymalizacji mapy w systemach SLAM, dopasowywanie cech oraz redukowanie dryftu. W [227] pokazano, że integracja estymowanych map głębi z systemem ORB-SLAM2 poprawia jego wyniki w zamkniętych środowiskach. W związku z tym, wykorzystanie metod głębokiego uczenia zdaje się być bardzo interesującym kierunkiem do prowa-

dzenia badań.

Opis eksperymentu

W tym eksperymencie najpierw skupiono się na stworzeniu zbioru uczącego. W tym celu wykorzystano dane widziane przez Kinecta v2, które były zbierane przez robota wyposażonego również w kamerę Kinect v1. Te dane udostępniane są w zbiorze PUTKK [166].

W eksperymentach wykorzystano metodę Telei oraz Naviera-Stokesa do uzupełnienia dziur w mapie głębi dla promienia o długości 3, ponieważ taki okazał się najlepszy w poprzednim eksperymencie. W ramach wstępnych eksperymentów zbadano jaki wpływ mają mapy głębi wygenerowane przez sieć Monodepth z oryginalnymi wagami oraz powstałe przez wykorzystanie ich do uzupełnienia dziur. Zbadano również jaki wpływ ma uzupełnianie za pomocą sieci Monodepth douczonej na podstawie sekwencji z zestawu PUTKK, w której danymi odniesienia były odpowiednio przetworzone obrazy z Kinecta v2 (proces tego przetwarzania opisano w 6.2.2). Zbadano trzy warianty:

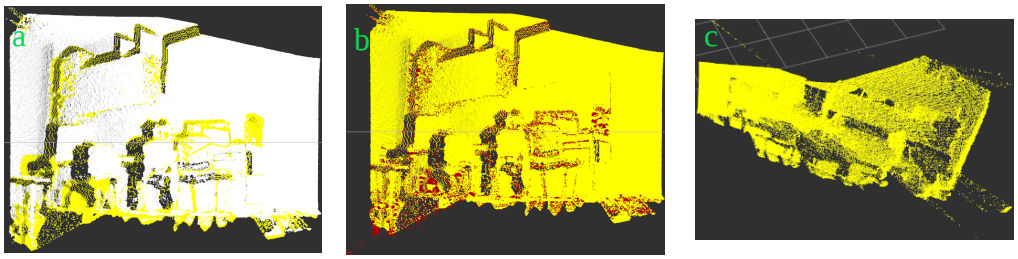
- Zmodyfikowano wejście sieci, tak by przyjmowała dane RGB-D
- Nie zmodyfikowano wejścia sieci (przyjmowała dane RGB), ale tak powstałe mapy głębi zostały wykorzystane do uzupełnienia braków w mapach głębi rejestrowanych przez kamerę Kinect v1
- Zmodyfikowano wejście sieci, tak by przyjmowała dane RGB-D oraz tak powstałe mapy głębi zostały wykorzystane do uzupełnienia braków w mapach głębi rejestrowanych przez kamerę Kinect v1

Otrzymane wyniki porównano z wynikami uzyskiwanymi bez ingerencji w mapy głębi, z wykorzystaniem oryginalnych danych.

Wyniki–wstępne eksperymenty

W ramach wstępnych eksperymentów wykorzystano metodę Telei do uzupełnienia dziur w mapie głębi powstałej po transformacji danych z Kinecta v2 do obrazu, jaki widziałby Kinect v1. Uzupełnianie dziur przeprowadzono również przed transformacją do Kinecta v1, na danych dostarczonych przez Kinecta v2. Następnie tak powstałe mapy głębi zwizualizowano w $3D$, dzięki wykorzystaniu systemu ROS. Przedstawia to rys. 6.37,

uwzględniający punkty powstałe z nieuzupełnionej mapy głębi (tylko i wyłącznie przekształconej z Kinecta v2 na Kinect v1). Rys. 6.37 obrazuje



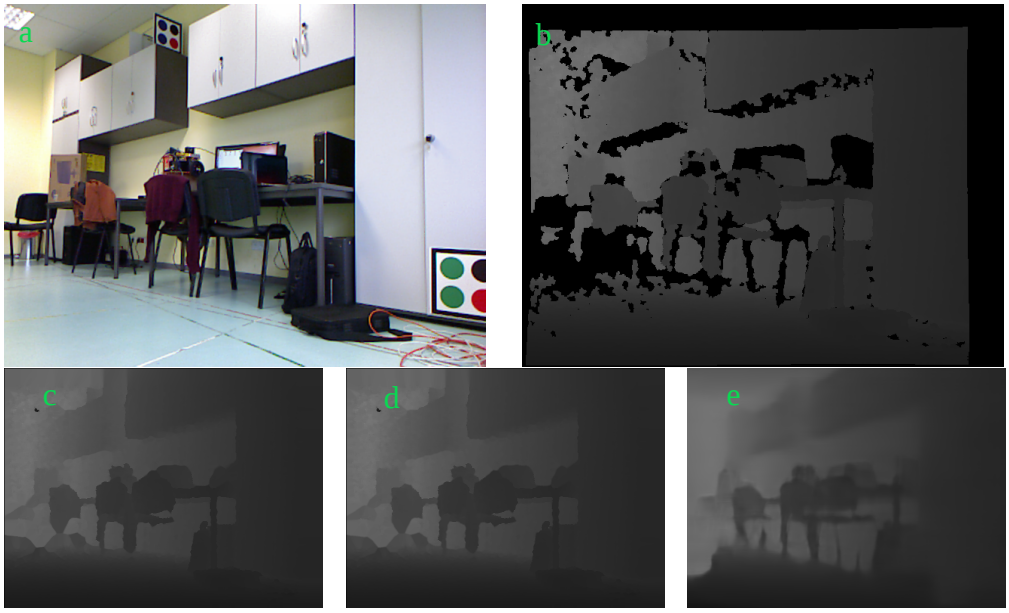
Rysunek 6.37: Chmury punktów wygenerowane na postawie przekształconego obrazu z Kinecta v2 na Kinecta v1: a) na biało bez uzupełniania a na żółto z uzupełnianiem po transformacji do Kinecta v1, b) na żółto z uzupełnianiem po transformacji do Kinecta v1 a na czerwono najpierw inpainting a tak uzupełniona mapa transformowana do kadru Kinecta v1, c) rzut z góry chmury punktów powstałej po transformacji do Kinecta v1 i uzupełnianiu

trudność uzupełniania dziur w mapie głębi widzianej przez Kinecta v1.

Na rys. 6.38 przedstawiono porównanie oryginalnego obrazu głębi z uzupełnionymi za pomocą algorytmu Naviera-Stokesa, Telei, oraz po inferencji, douczonej na podstawie danych RGB-D, sieci neuronowej Monodepth.

Obrazy głębi, uzupełniane klasycznymi metodami, nie różnią się znacznie. Nie odtwarzają większości nóg od krzeseł i w niektórych miejscach są rozmazane. Obiekty odtworzone za pomocą sieci neuronowej mają gładkie granice, wiarygodniej przedstawiają nogi od krzeseł i ogólnie wyglądają lepiej, pomimo braku rekonstrukcji wszystkich detali widocznych na obrazie RGB.

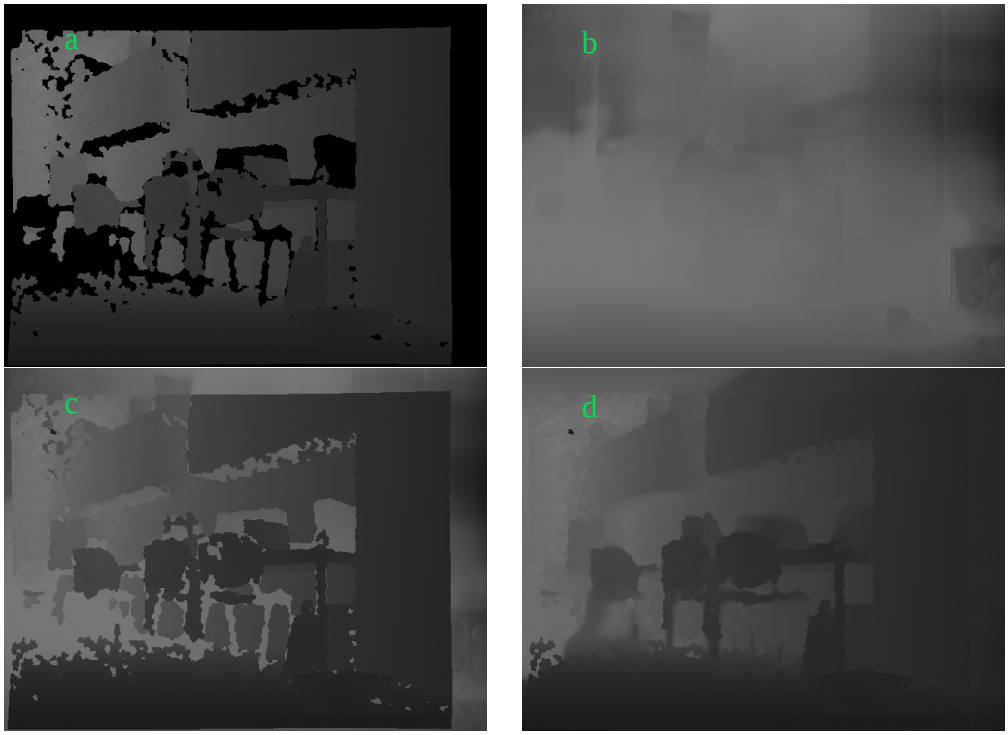
W ramach wstępnych eksperymentów zbadano jaki wpływ mają mapy głębi wygenerowane przez sieć z oryginalnymi wagami (po przeskalowaniu, tak by minimalne wartości i maksymalne były takie same jak w obrazie z Kinect v1) oraz te powstałe przez wykorzystanie ich do uzupełnienia dziur. Powstałe mapy głębi przedstawiono na rys. 6.39. To porównanie miało na celu ustalenie, czy oryginalny, otwartoźródłowy model z wcześniej uczonymi (przez autorów sieci Monodepth) wagami, jest w stanie dostarczyć użyteczne mapy głębi dla zestawu PUTKK. Rys. 6.39b ukazuje przykładowy wynik dla pierwszego obrazu 0 z pierwszej sekwencji (*putkk_Dataset_1_Kin_1*). Jest to dowód, że sieć Monodepth nie jest w stanie spełnić naszych wymogów bez dalszego douczania na docelowym zestawie



Rysunek 6.38: a) Oryginalny obraz RGB, Mapy głębi: b) oryginalna, c) Navier-Stokes(NS) d) Telea, e) inferencja sieci Monodepth douczonej na podstawie danych RGB-D

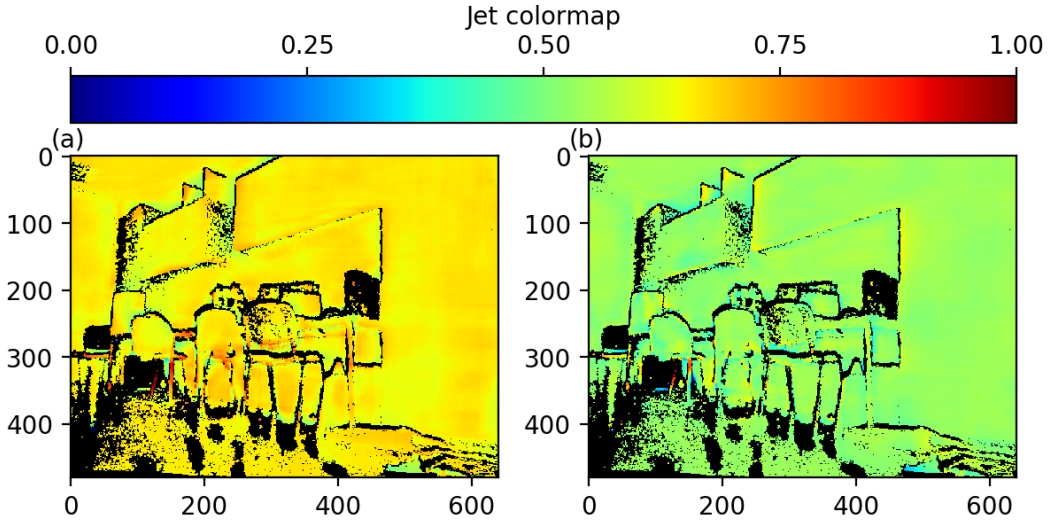
danych. Na obrazie będącym połączeniem oryginalnej mapy głębi oraz tej z inferencji 6.39c widać poważne zmiany intensywności (a więc odległości) między uzupełnianymi obszarami, a tymi z dostępnymi pomiarami. W przeciwieństwie do tego, później wykorzystywane uzupełnione oryginalne dane inferencją douczonej sieci pracujące na danych wejściowych RGB są wizualnie zadowalające. W związku z tym w badaniach wykorzystaliśmy tylko douczoną sieć Monodepth.

Mimo tego, że wizualne porównanie klatek obrazu jest zachęcające do przeprowadzenia dalszych eksperymentów, to te wyniki jakościowe powinny być uzupełnione poprzez wyniki ilościowe. W związku z tym dokładnie przyjrzeliśmy się oszacowanym wartościom głębi bezpośrednio porównując prosty douczony model, bazujący na danych RGB do inferencji, oraz bardziej złożony, biorący na wejście dane RGB oraz D z Kinecta v1. Na rys. 6.40 porównano błędy estymacji głębi w odniesieniu do wzorcowych map uzyskanych (i odpowiednio przekształconych) przez Kinecta v2 dla tych dwóch wariantów. Obydwie mapy różnic oddzielnie znormalizowano do zakresu $< 0,1 >$ i wykorzystano mapy kolorów do zwizu-



Rysunek 6.39: Mapy głębi z ujęcia 0 zestawu *putkk_Dataset_1_Kin_1*: a) Oryginalna, b) estymowana przez oryginalne wagi sieci Monodepth, oraz c) oryginalne mapy uzupełnione inferencją RGB sieci Monodepth z oryginalnymi wagami d) oryginalne mapy uzupełnione inferencją RGB douczonej sieci Monodepth

alizowania zbyt bliskich oszacowań jako tych od 0 do $0,5$ (oznaczonych niebieskim kolorem), będących blisko danych odniesienia jako bliskich $0,5$ (kolorem zielonym), a tych oszacowań lokujących punkt zbyt daleko jako zbliżających się do 1 (w związku z tym przybierających barwę czerwoną). W obydwu obrazach, obszary bez informacji w danych odniesienia Kinecta v2, zaznaczono kolorem czarnym.



Rysunek 6.40: Mapy kolorów wizualizujące różnice pomiędzy estymowaną głębią sceny a uzyskaną mapą odniesienia z Kinecta v2 dla a) douczonej sieci Monodepth z inferencją na podstawie danych RGB tylko, i b) z zarówno danymi RGB jak i D z Kinecta v1

Jak widać na rys. 6.40 wariant korzystający do inferencji tylko z danych RGB ma skłonności do przypisywania zbyt dużych odległości (rys. 6.40a). Natomiast wersja wykorzystująca łącznie dane RGB oraz D z Kinecta v1 do inferencji cechuje się poprawniejszymi szacunkami wartości głębi. Największe różnice są widoczne w trudnych obszarach wokół nóg krzeseł i krawędzi.

Te wyniki są dalej sprawdzane kolejnych badaniach ukazujących jak różne podejścia do uzupełniania brakujących danych w mapach głębi wpływają wyniki osiągane przez system odometrii wizyjnej.

Wyniki–obszerniejsze badania

W celu poprawy uzyskiwanych wyników postanowiliśmy dokonać optymalizacji parametrów systemu odometrii wizyjnej. Przeprowadziliśmy ją tak jak w 6.6.4 i 6.6.5.

Również i tu sprawdzono, czy uzyskany za pomocą badanych metod populacyjnych zestaw parametrów jest uniwersalny. W tym celu do optymalizacji posłużono się *putkk_Dataset_5_Kin_1* a do weryfikacji wybrano sekwencję *putkk_Dataset_1_Kin_1*. Do uzyskanych parametrów za pomocą algorytmu PSO z metryką ATE będzie się dalej odnoszono się jako PSO ATE a do zestawu parametrów PSO z metryką RPE jako PSO RPE.

Dla porównania sprawdzono jakie wyniki osiąga odometria wizyjna bez uzupełniania map głębi, z poprzednio znalezionymi, w sekwencjach *TUM RGB-D*, optymalnymi parametrami: OP1 na podstawie *fr1_desk* i OP2 *fr1_room*.

Najpierw wykorzystano optymalizację PSO do znalezienia 4 parametrów algorytmu RANSAC i progu detekcji AKAZE τ_A w maksymalnie 9 iteracjach. Tab. 6.20 zawiera optymalne wartości parametrów znalezione podczas tego eksperymentu.

Wartości ATE RMSE oraz RPE RMSE dla tych zestawów parametrów są zebrane w tab. 6.21.

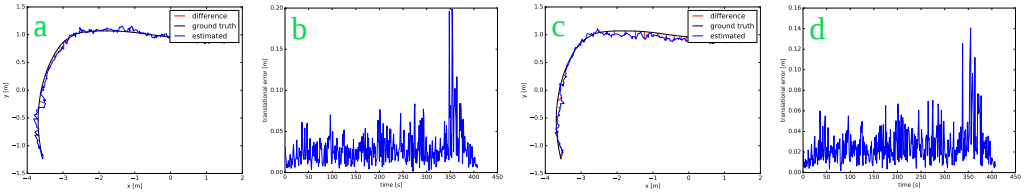
Tabela 6.20: Zoptymalizowane parametry VO z ich najlepszymi wartościami

Metoda	$d_{E,1}$	$\Gamma_{o,1}$	$d_{E,2}$	$\Gamma_{o,2}$	τ_A
PSO ATE	0,043	0,914	0,051	0,892	0,0001
PSO RPE	0,040	0,859	0,041	0,800	0,0001
OP1	0,059	0,812	0,042	0,836	0,0021
OP2	0,042	0,916	0,012	0,855	0,0013
Ręcznie	0,03	0,80	0,003	0,800	0,002

Tabela 6.21: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

Metoda	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
PSO ATE	0,064	0,037	0,833
PSO RPE	0,083	0,033	0,723

Dane tabelaryczne dopełniają odpowiednie wykresy ATE i RPE 6.41.



Rysunek 6.41: Wykresy błędów ATE (a, c) oraz błędów translacji RPE (b, d) dla sekwencji *putkk_Dataset_5_Kin_1*. (a, b) Uzyskano w wyniku optymalizacji PSO ATE, a (c, d) PSO RPE

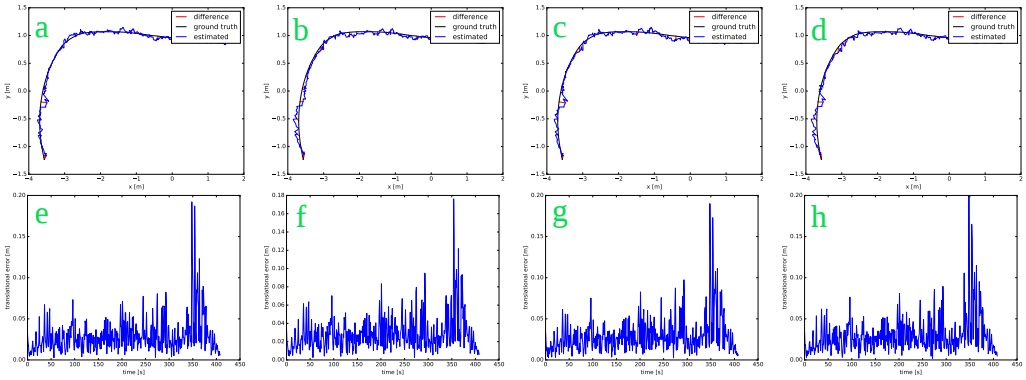
Jako że najlepsze wyniki osiągnięto w optymalizacji PSO ATE, to wybrano zestaw parametrów dotyczący algorytmu RANSAC i, tak jak poprzednio, optymalizowano dalej tylko detektor τ_A za pomocą algorytmów PSO i EA (bo poprzednie badania nie wyłoniły skuteczniejszego podejścia) wraz z alternatywnymi miarami dopasowania ATE oraz RPE.

Tym razem optymalizację PSO skończono po 9 iteracjach–tyle wystarczyło by wszystkie cząsteczki (zarówno w optymalizacji PSO ATE jak i RPE) osiągnęły minimalną wartość dla progu detekcji ($\tau_{Amin} = 0,00010$). W przypadku algorytmu ewolucyjnego optymalizacja trwała 20 pokoleń, τ_A nie osiągną wartości minimalnej i dla optymalizacji EA ATE wyniósł $0,000114$ a dla EA RPE $0,000124$. Przedstawiamy wyniki w tabeli 6.22 dopełnianej wykresami błędów ATE i RPE 6.42.

Tabela 6.22: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

	putkk_Dataset_5_Kin_1			
Metryka błedu	PSO ATE	PSO RPE	EA ATE	EA RPE
ATE RMSE [m]	0,059	0,066	0,066	0,072
Trans. RPE RMSE [m]	0,037	0,035	0,037	0,037
Rot. RPE RMSE [°]	0,835	0,785	0,831	0,837

Ponieważ osiągnięto minimalną wartość progu detekcji, to postanowiono o jej obniżeniu do $\tau_{Amin} = 0,00001$ (jest to domyślna minimalna wartość tego parametru w bibliotece AKAZE) i o ponowieniu eksperymentów. Wyniki uzyskane za pomocą parametrów będących wynikami optymalizacji z poprzednią, wyższą minimalną wartością τ_{Amin} będą oznaczone dopiskiem E1, a te będące efektem kolejnej optymalizacji, z niższą minimalną wartością τ_{Amin}, dopiskiem E2.



Rysunek 6.42: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_5_Kin_1*. (a, e) Uzyskano w wyniku optymalizacji PSO ATE, (b, f) PSO RPE (c, g) EA ATE, albo (d, h) EA RPE

Tab. 6.23 zawiera optymalne wartości parametrów znalezione podczas tego eksperymentu oraz te znalezione uprzednio.

Tabela 6.23: Zoptymalizowane parametry VO z ich najlepszymi wartościami

Metoda	$d_{E,1}$	$\Gamma_{o,1}$	$d_{E,2}$	$\Gamma_{o,2}$	τ_A
PSO ATE E2	0,060	0,890	0,068	0,892	0,00001
PSO RPE E2	0,048	0,805	0,043	0,800	0,00001
PSO ATE E1	0,043	0,914	0,051	0,892	0,0001
PSO RPE E1	0,040	0,859	0,041	0,800	0,0001
OP1	0,059	0,812	0,042	0,836	0,0021
OP2	0,042	0,916	0,012	0,855	0,0013
Ręcznie	0,03	0,80	0,003	0,800	0,002

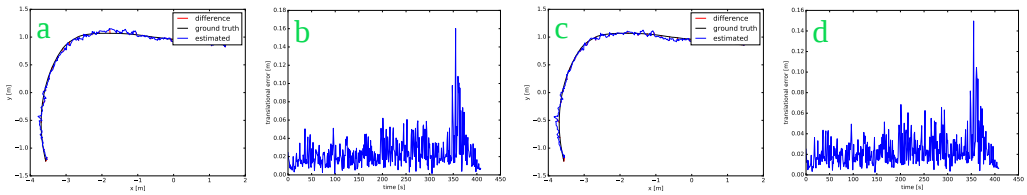
Uzyskane parametry dotyczące algorytmu RANSAC w eksperymentach E1 i E2 (dla tej samej funkcji celu) niewiele się różnią, zaś efekt optymalizacji τ_A jest niemal identyczny. Tu również osiągnięto minimalną wartość. Tak jak poprzednio dokonano dalszej optymalizacji parametru tylko detektora. Dla algorytmu RANSAC wybrano parametry uzyskane w PSO ATE E2, gdyż one pozwoliły na uzyskanie najlepszych wyników, które dla obydwu funkcji celu, zawiera tab. 6.24.

Tu również przezentujemy uzupełnienie odpowiednimi wykresami ATE i RPE 6.43

Następnie również optymalizowano dalej próg detekcji τ_A. W przypadku podejścia rojowego wystarczyło 9 iteracji, a algorytmowi ewolucyjne-

Tabela 6.24: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

Metoda	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
PSO ATE E2	0,048	0,030	0,675
PSO RPE E2	0,056	0,029	0,643



Rysunek 6.43: Wykresy błędów ATE (a, c) oraz błędów translacji RPE (b, d) dla sekwencji *putkk_Dataset_5_Kin_1*. (a, b) Uzyskano w wyniku optymalizacji PSO ATE, a (c, d) PSO RPE

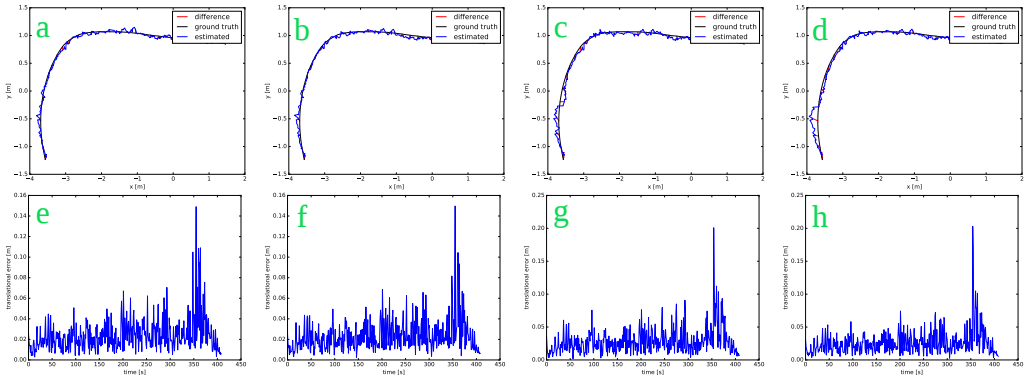
mu zajęło to 20 pokoleń, w obydwu przypadkach, dla obydwu miar dopasowania. Progi detekcji wynosiły odpowiednio PSO ATE $0,00001$, PSO RPE $0,00001$, EA ATE $0,000104$, EA RPE $0,000034$. Odpowiednie wyniki zebrano w tab. 6.25 oraz na wykresach ATE i RPE 6.44.

Tabela 6.25: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

	putkk_Dataset_5_Kin_1			
Metryka błedu	PSO ATE	PSO RPE	EA ATE	EA RPE
ATE RMSE [m]	0,049	0,056	0,070	0,072
Trans. RPE RMSE [m]	0,030	0,029	0,035	0,033
Rot. RPE RMSE [°]	0,665	0,643	0,794	0,745

Najlepszym zestawem parametrów okazały sie te uzyskane za pomocą optymalizacji PSO z miarą ATE, bo pozwalały uzyskać najwierniejsze odtworzenie trajektorii (oczywiście wariant PSO RPE uzyskał trochę lepsze wyniki w metryce RPE RMSE). Ponieważ metryka ATE lepiej odzwierciedla globalną dokładność oszacowanych trajektorii, to skupiono się na tych parametrach. Uzyskane parametry PSO ATE z obydwu eksperymentów (PSO ATE E1 i PSO ATE E2) zostały zweryfikowane na innych zestawach. Uzyskane wyniki porównaliśmy z wynikami otrzymanymi dla:

- oryginalnych map głębi i VO pracującą z parametrami OP2 z dalej



Rysunek 6.44: Wykresy błędów ATE (a, b, c, d–na czarno: trajektoria odniesienia, na niebiesko oszacowana, na czerwono błędy) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_5_Kin_1*. (a, e) Uzyskano w wyniku optymalizacji PSO ATE, (b, f) PSO RPE (c, g) EA ATE, albo (d, h) EA RPE

optymalizowanym detektorem metodą PSO ATE, do nich odnosimy się jako O OP2;

- uzupełnionych map głębi metodami klasycznymi i VO pracującą z parametrami OP2, z dalej optymalizowanym detektorem metodą PSO ATE, do nich odnosimy się jako T OP2 albo NS OP2;
- uzupełnionych map głębi metodami klasycznymi i VO pracującą z parametrami E1, z dalej optymalizowanym detektorem metodą PSO ATE, do nich odnosimy się jako T E1 albo NS E1;
- uzupełnionych map głębi metodami klasycznymi i VO pracującą z parametrami E2, do nich odnosimy się jako T E2 albo NS E2;
- uzupełnionych map głębi za pomocą sieci neuronowej i VO pracującą z parametrami E1, do nich odnosimy się jako CN E1;
- uzupełnionych map głębi za pomocą sieci neuronowej i VO pracującą z parametrami E2. do nich odnosimy się jako CN E2;

Wyniki RMSE dla zestawów *putkk_Dataset_1_Kin_1*, *putkk_Dataset_2_Kin_1*, *putkk_Dataset_3_Kin_1*, *putkk_Dataset_4_Kin_1* i *putkk_Dataset_5_Kin_1* zebrano w tab. 6.26–6.28. Generalnie wyniki RPE dla zmodyfikowanych wersji są gorsze od tych korzystających z oryginalnych danych. Oczywiście może to wynikać ze specyfiki sceny, bo różni się od tej wykorzystanej

do douczania parametrów sieci, albo z postawienia nacisku na optymalizację ATE.

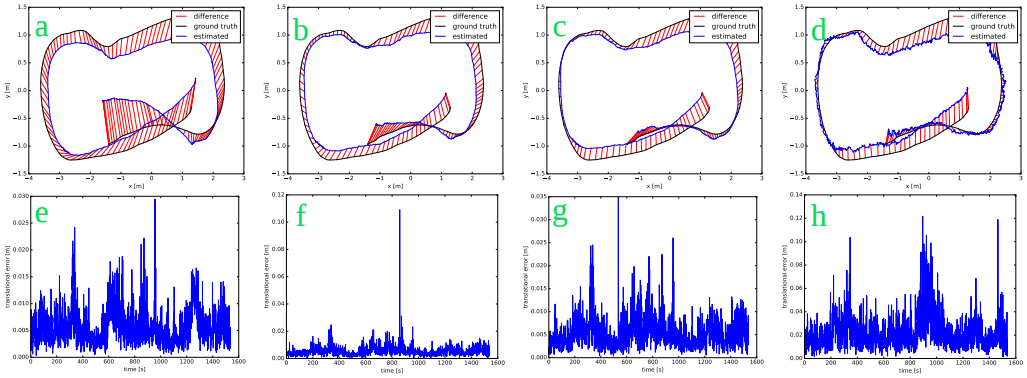
Tabela 6.26: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_1_Kin_1* oraz *putkk_Dataset_2_Kin_1*

	putkk_Dataset_1_Kin_1			*putkk_Dataset_2_Kin_1*		
Metoda	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
O OP2	0,596	0,009	0,168	0,677	0,010	0,243
O E1	0,592	0,007	0,157	0,758	0,009	0,200
O E2	0,558	0,007	0,152	0,739	0,009	0,184
NS OP2	1,112	0,021	0,507	1,148	0,030	0,713
T OP2	0,443	0,016	0,369	0,616	0,033	0,694
NS E1	0,339	0,010	0,234	0,679	0,018	0,374
T E1	0,328	0,010	0,223	0,939	0,019	0,391
NS E2	0,297	0,007	0,165	0,535	0,009	0,210
T E2	0,341	0,006	0,156	0,504	0,009	0,203
CN E1	0,275	0,029	0,681	0,817	0,044	0,997
CN E2	0,359	0,027	0,641	0,502	0,038	0,852

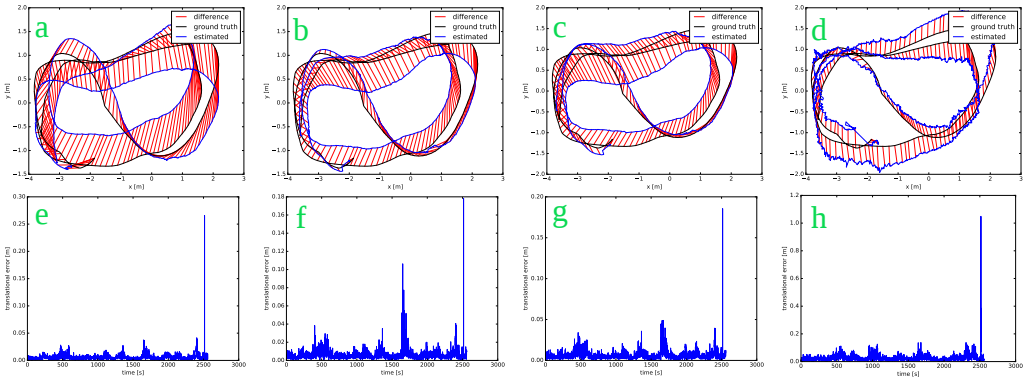
W przypadku *putkk_Dataset_1_Kin_1* widać znaczną poprawę po wykorzystaniu klasycznych metod uzupełniania punktów i odpowiedniego zestawu parametrów systemu odometrii wizyjnej VO. Wykorzystanie uzupełniania bazującego na sztucznych sieciach neuronowych dało co prawda lepsze wyniki ATE, ale za cenę gorszych RPE.

Dla *putkk_Dataset_2_Kin_1* najlepszy wyniki uzyskano korzystając z CN E2. CN E1 dało pogorszenie. Być może w obliczeniach transformacji wzięło udział zbyt mało punktów, wśród których były być może błędnie uzupełniane punkty. Parametry znalezione w eksperymencie E1 dawały ogólnie gorsze rezultaty, bez względu z którego podejścia uzupełniania głębi skorzystano i czy w ogóle z niego skorzystano. Tu wykorzystanie sieci neuronowych zwiększało błędy RPE. Dane tabelaryczne i tu uzupełniono wykresami ATE oraz RPE (rys. 6.45 i 6.46), ale dla już wybranych podejść.

Na wykresach ATE widać jak nierówną trajektorię udaje się odzyskać korzystając z map głębi po inferencji sieci. Mimo iż wykres RPE dla CN ma



Rysunek 6.45: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_1_Kin_1*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2



Rysunek 6.46: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_2_Kin_1*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2

mniejszą skalę niż dla adekwatnych trajektorii wykorzystujących oryginalne bądź uzupełniane algorytmem Telei braki, to odzyskana trajektoria jest od nich bardziej postrzępiona, nierówna. Różnice w lokalnych maksimach błędów RPE RMSE mogą wynikać z wykorzystanych map głębi- różne sposoby mogą być lepsze w innych warunkach.

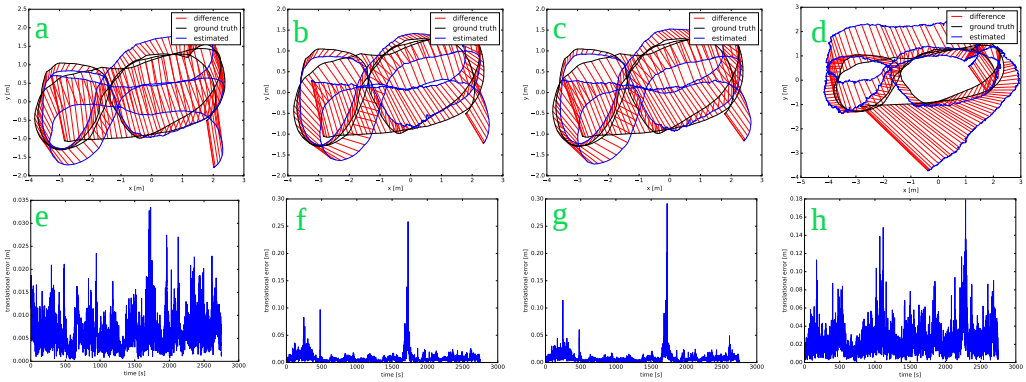
Tabela 6.27: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_3_Kin_1* oraz *putkk_Dataset_4_Kin_1*

	putkk_Dataset_3_Kin_1			*putkk_Dataset_4_Kin_1*		
Metoda	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
O OP2	1,145	0,012	0,251	—	—	—
O E1	1,056	0,009	0,220	1,047	0,009	0,349
O E2	1,024	0,007	0,190	1,052	0,008	0,294
NS OP2	0,934	0,033	0,607	—	—	—
T OP2	1,092	0,030	0,532	—	—	—
NS E1	0,909	0,023	0,435	1,266	0,012	0,469
T E1	0,899	0,022	0,415	1,558	0,012	0,407
NS E2	0,789	0,012	0,266	1,111	0,010	0,292
T E2	0,819	0,013	0,274	1,036	0,010	0,280
CN E1	0,807	0,036	0,972	4,363	0,045	3,616
CN E2	1,230	0,031	0,916	0,474	0,032	1,099

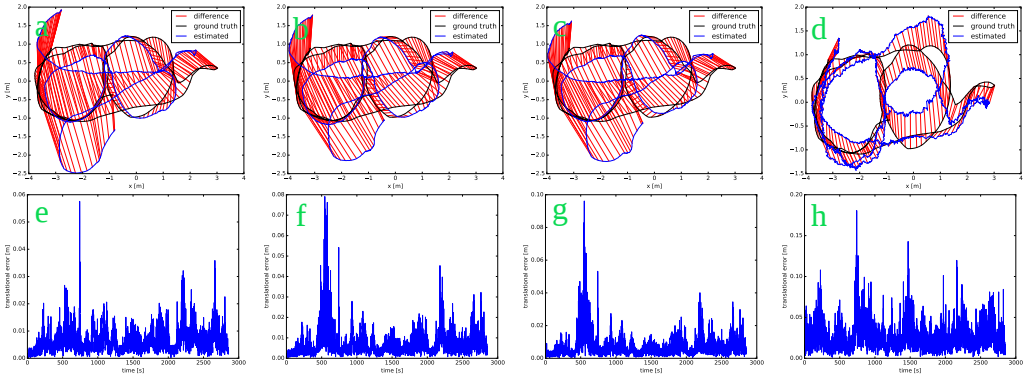
W *putkk_Dataset_3_Kin_1* najlepsze rezultaty uzyskano w przy wykorzystaniu algorytmu Naviera-Stokesa z parametrami E2. W przypadku ulepszonych map głębi za pomocą sztucznych sieci neuronowych zestaw E1 okazał się lepszy od E2, który nie poprawiał ani nie pogarszał znacznie uzyskiwanych wyników.

W zestawie *putkk_Dataset_4_Kin_1* w przypadku ich wykorzystania w eksperymencie CN E1 wystąpił moment, w którym transformację klatka po klatce wyznaczono w złym kierunku, bo obliczana była ze zbyt małą liczbą cech charakterystycznych (poniżej 20). Należy zwrócić uwagę, że była to trudna trajektoria, gdyż ubogie w cechy charakterystyczne miejsca uniemożliwiły odtworzenie trajektorii przy wykorzystaniu standardowych parametrów. W E2 wykorzystanie ulepszania map głębi pozwoliło na uzyskanie lepszych rezultatów, ale tylko w przypadku ustawienia

progu detekcji punktów kluczowych na zalecane minimum. W krytycznym momencie dostępne były dobre cechy pozwalające na odrzucenie wadliwych powodujących podążanie w złym kierunku. Stosowanie klasycznych podejść nie pozwoliło na uzyskanie lepszych wyników ATE, ale RPE, co można dostrzec na wykresach ATE i RPE (rys. 6.47 i 6.48).



Rysunek 6.47: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_3_Kin_1* (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2



Rysunek 6.48: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_4_Kin_1*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2

W przypadku *putkk_Dataset_3_Kin_1* stosowanie uzupełnianych map głębi za pomocą sieci neuronowej nie przyniosło ani korzyści, ani szkód. Za to klasyczne metody były nieco lepsze. Jeśli chodzi o *putkk_Dataset_4*

_Kin_1 to nowoczesne podejście okazało się o wiele lepsze od poprzednich. Na uwagę zasługuje fakt, że są to bardzo długie trajektorie.

Tabela 6.28: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

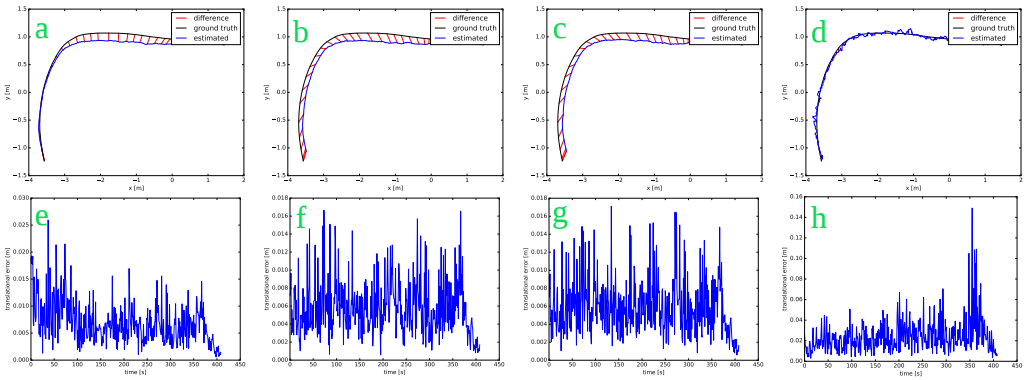
	putkk_Dataset_5_Kin_1		
Metoda	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
O OP2	0,198	0,012	0,174
O E1	0,192	0,009	0,153
O E2	0,176	0,008	0,149
NS OP2	0,201	0,011	0,187
T OP2	0,204	0,012	0,183
NS E1	0,173	0,009	0,152
T E1	0,172	0,009	0,155
NS E2	0,212	0,007	0,143
T E2	0,212	0,007	0,138
CN E1	0,059	0,037	0,835
CN E2	0,049	0,030	0,665

Zestaw *putkk_Dataset_5_Kin_1* to ten, który służył do optymalizacji parametrów systemu odometrii wizyjnej. W przypadku miary ATE widać tu znaczną przewagę wykorzystania map uzupełnianych za pomocą sieci neuronowych, ale za cenę znacznego wzrostu błędów RPE. Rys.6.29 przedstawia wykresy ATE i RPE.

W przypadku wykorzystania innego zestawu danych: *fr1_desk* i *fr1_room*, uzupełnianie brakujących danych głębi nie przyniosło żadnych korzyści. Jeśli chodzi o wykorzystanie sztucznych sieci neuronowych to widać znaczne pogorszenie, co można zaobserwować na rys. 6.50 i 6.51 przedstawiających wykresy ATE i RPE.

Jak widać skorzystanie z sztucznej sieci neuronowej nie dało żadnych korzyści a jedynie pogorszenie. Może to wynikać z przeuczenia sieci, niedostępności istniejących tu elementów w środowisku uczącym albo z innych parametrów kamery. Jak widać na rys. 6.52 otrzymane po inferencji obrazy są niewyraźne, w przeciwieństwie do tych oryginalnych.

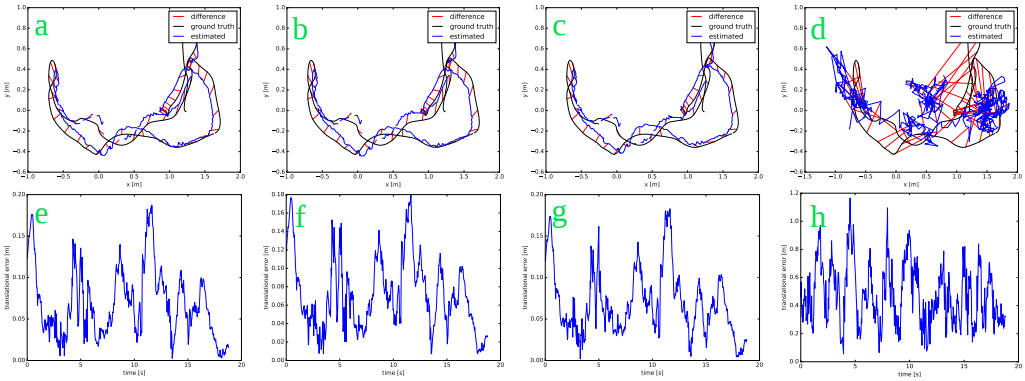
To co powoduje znaczne problemy to prawdopodobnie maksymalne błędy RPE powodujące powstanie nieskompensowanego błędu orientacji.



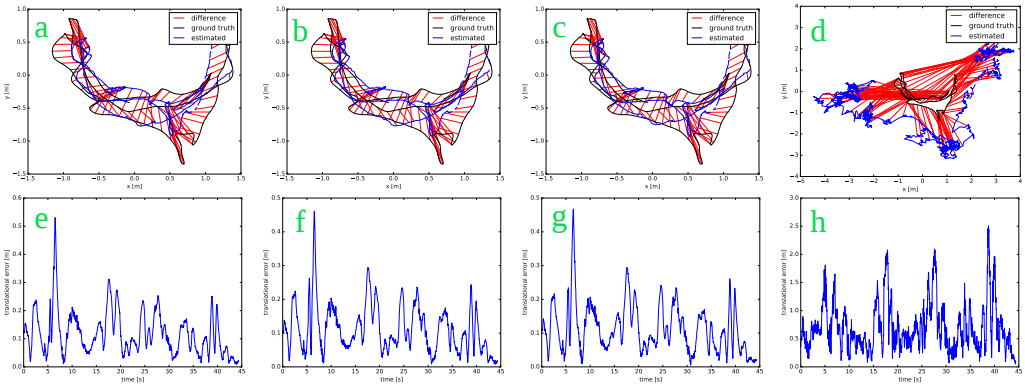
Rysunek 6.49: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *putkk_Dataset_5_Kin_1*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2

Tabela 6.29: Porównanie ATE RMSE oraz RPE RMSE dla *fr1_desk* oraz *fr1_room*

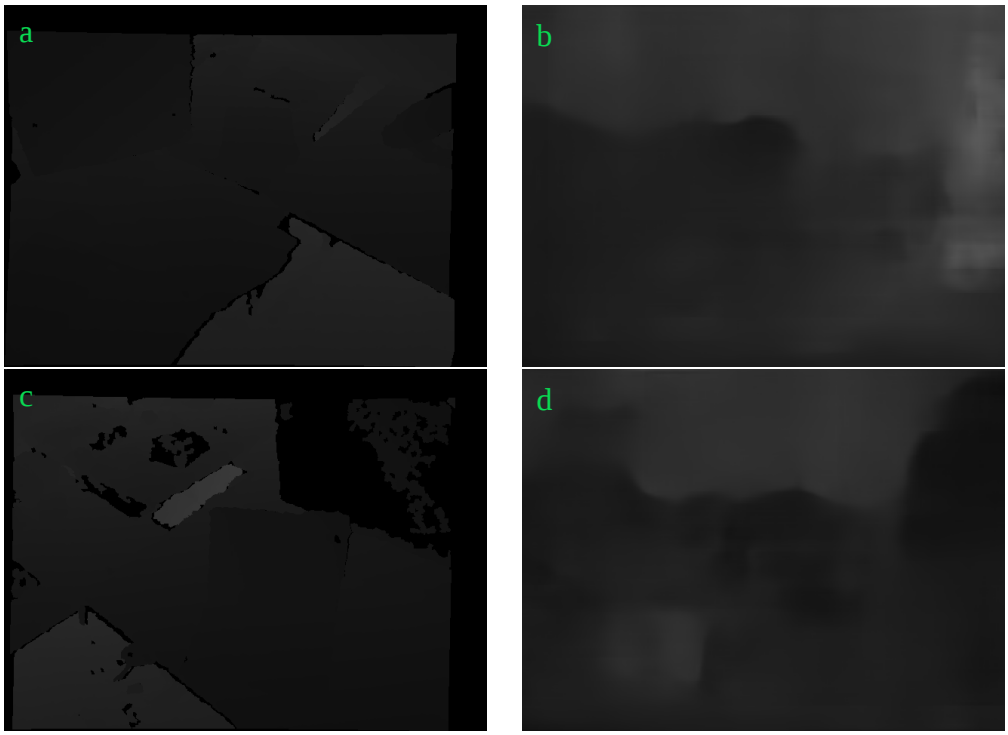
Metoda	*fr1_desk*			*fr1_room*		
	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
O OP2	0,109	0,074	2,470	0,290	0,120	2,403
O E1	0,108	0,081	2,626	0,357	0,136	2,615
O E2	0,130	0,082	2,572	0,375	0,142	2,727
NS OP2	0,131	0,080	2,721	0,369	0,130	2,656
T OP2	0,123	0,077	2,613	0,343	0,128	2,589
NS E1	0,113	0,085	2,904	0,412	0,137	2,617
T E1	0,118	0,088	2,965	0,398	0,135	2,617
NS E2	0,125	0,083	2,677	0,398	0,133	2,637
T E2	0,129	0,081	2,618	0,379	0,135	2,618
CN E1	0,630	0,603	25,34	2,013	0,893	30,908
CN E2	0,572	0,513	20,32	2,849	0,832	28,942



Rysunek 6.50: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *fr1_desk*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2

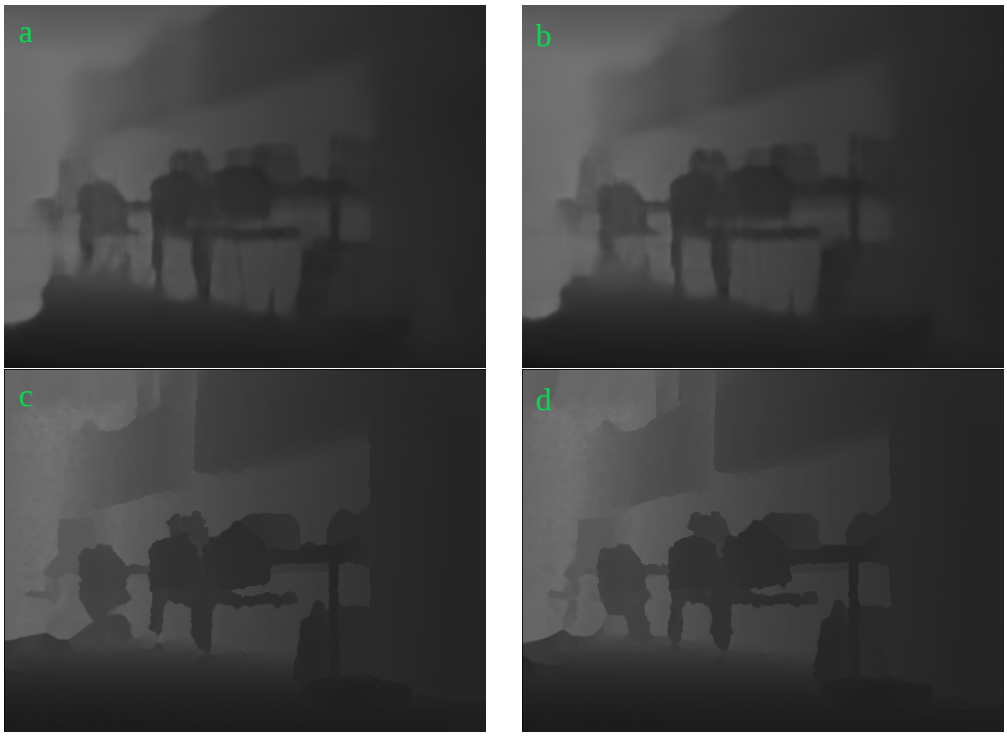


Rysunek 6.51: Wykresy błędów ATE (a, b, c, d) oraz błędów translacji RPE (e, f, g, h) dla sekwencji *fr1_room*. (a, e) VO z oryginalnymi obrazami E2, (b, f) VO z NS E2, (c, g) VO z Telea E2, albo (d, h) CN E2



Rysunek 6.52: Mapy głębi widziane przez kamerę robota a) oryginalne *fr1_desk*, b) po inferencji *fr1_desk* c) oryginalne *fr1_room*, d) po inferencji *fr1_room*

Kolejny problem obrazuje rys. 6.53.



Rysunek 6.53: Mapy głębi zestawu *putkk_Dataset_1_Kin_1* a) ujęcie 23 po inferencji, b) ujęcie 24 po inferencji, c) ujęcie 23 NS, d) ujęcie 24 NS

Widać na nim, że na mapach głębi, z ujęcia na ujęcie, potrafią pojawiać się nie widziane przedtem przedmioty: dorysowywane są czasami nogi od krzeseł. Jeśli wystąpi to klatka po klatce na obiektach, na których wykryto punkty kluczowe, to spowoduje to dodanie błędnych danych do obliczania roto-translacji. O wiele lepsze jest jeśli tych nóg od krzeseł nie widać, niż to że pojawiają się i znikają.

Wyniki–dalsze badania nad uzupełnianiem danych

W kolejnych badaniach posłużyliśmy się podobną metodologią do znalezienia odpowiednich parametrów dla VO działającej z oryginalnymi mapami głębi z Kinecta v1 uzupełnionymi inferencją douczonej sieci Monodepth na podstawie danych RGB, bądź RGB-D. Wykorzystano poprzednio znalezione parametry algorytmu RANSAC. Natomiast do optymali-

zacji progu detekcji algorytmu AKAZE τ_A wykorzystano tylko podejście rojowe PSO z miarą ATE RMSE, wykorzystując sekwencję *putkk_Dataset_5_Kin_1* (tę samą co poprzednio). W efekcie otrzymano progi detekcji $\tau_A = 0,001880$ oraz $\tau_A = 0,000597$. W tabeli 6.30 zawato wyniki dla odometrii wizyjnej działającej na podstawie oryginalnych map głębi z Kinecta v1 (z parametrami OP2) oraz dla systemu VO działającego z uzupełnionymi mapami i znalezionymi parametrami τ_A).

Tabela 6.30: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_5_Kin_1*

	putkk_Dataset_5_Kin_1		
Metryka błędu	Bez uzupełniania	Uzupełnianie RGB	Uzupełnianie RGB-D
ATE RMSE [m]	0,198	0,176	0,147
Trans. RPE RMSE [m]	0,012	0,015	0,012
Rot. RPE RMSE [°]	0,174	0,289	0,212

Łączne wykorzystanie obrazów RGB oraz głębi pozwoliło nam na uzyskanie lepszych wyników niż pozwalała na to odometria wizyjna pracująca z oryginalnymi obrazami z Kinecta v1 oraz uzupełnionymi tylko na podstawie danych RGB. By to potwierdzić wykorzystano, tak jak poprzednio, inne sekwencje z zestawu PUTKK, dla których wyniki zebrano w tab. 6.31.

Weryfikacja potwierdziła nasze przypuszczenia: wykorzystanie danych RGB-D w inferencji sieci pozwala na uzyskanie lepszych wyników niż nieulepszone, oryginalne dane z kamery Kinect v1. Dzieje się to kosztem zwiększonych będów RPE, ale może to wynikać z położenia nacisku w optymalizacji na minimalizację błędu ATE, bądź zbytniego dopasowania parametrów do wykorzystanej sekwencji. W efekcie prosty system VO nie był w stanie dokonać estymacji całej trajektorii zestawu *putkk_Dataset_4_Kin_1*. Było to spowodowane znalezieniem zbyt małej ilości cech (w okolicach 700-setnego ujęcia) przez detektor AKAZE z dobranym w optymalizacji parametrem τ_A.

Wykorzystanie do inferencji tylko danych RGB nie pozwala na uzyskanie takich samych wyników jak danych RGB-D, lecz da się zauważyć poprawę w kilku obszarach. Tym niemniej lepiej wykorzystać podejście RGB-D, które korzysta z pełnej informacji udostępnianej przez kamerę Kinect v1 wyciągając z nich współzależności.

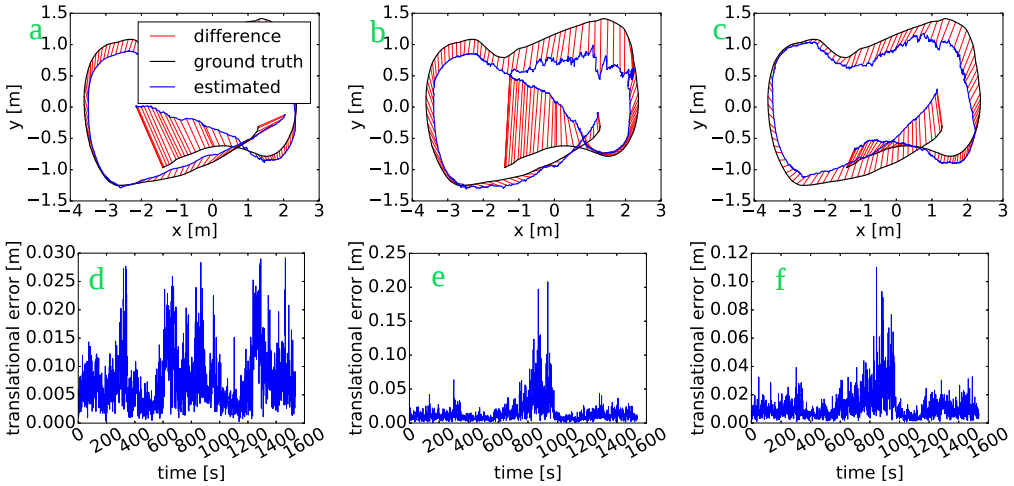
Tabela 6.31: Porównanie ATE RMSE oraz RPE RMSE dla *putkk_Dataset_3_Kin_1* oraz *putkk_Dataset_4_Kin_1*

Metoda	*putkk_Dataset_3_Kin_1*			*putkk_Dataset_4_Kin_1*		
	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]	ATE RMSE [m]	Trans. RPE RMSE [m]	Rot. RPE RMSE [°]
Bez uzupełniania	0,596	0,009	0,168	0,677	0,010	0,243
Uzupełnianie RGB	0,701	0,022	0,535	1,122	0,027	0,593
Uzupełnianie RGB-D	0,443	0,015	0,375	0,605	0,020	0,446
Inferencja RGB-D	0,359	0,027	0,641	0,502	0,038	0,852
	putkk_Dataset_3_Kin_1			*putkk_Dataset_4_Kin_1*		
Bez uzupełniania	1,145	0,012	0,251	—	—	—
Uzupełnianie RGB	0,886	0,027	0,591	—	—	—
Uzupełnianie RGB-D	0,773	0,019	0,407	—	—	—
Inferencja RGB-D	1,230	0,031	0,916	0,474	0,032	1,099

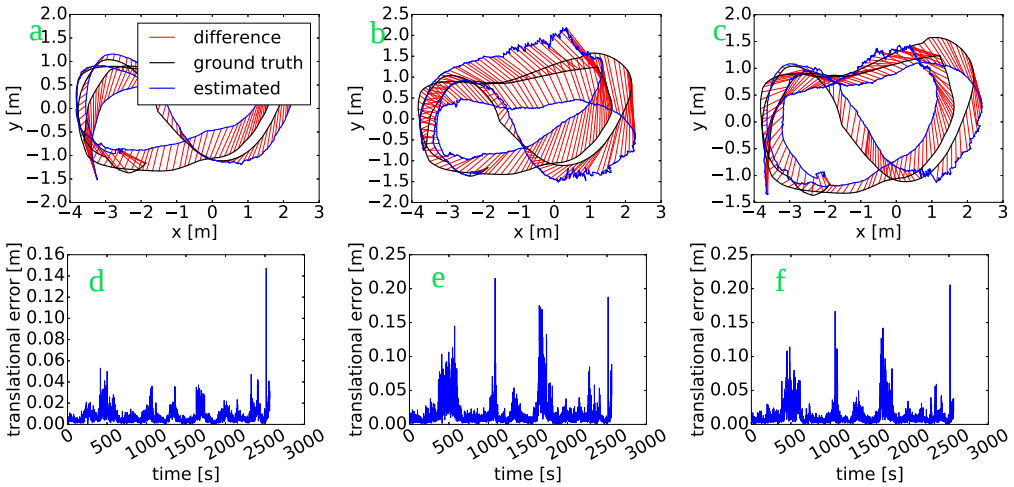
Odpowiednie wykresy ATE oraz RPE (przedstawione na rys. 6.55) pokazują jak przedstawione metody radzą sobie z sekwencjami nie wykorzystywanymi do optymalizacji parametrów: *putkk_Dataset_1_Kin_1*, *putkk_Dataset_2_Kin_1*, *putkk_Dataset_3_Kin_1*. Nie pokazano wykresów dla sekwencji *putkk_Dataset_4_Kin_1*, ponieważ pełną trajektorię dla niej uzuskała tylko wersja pracująca na danych pochodzących z inferencji RGB-D (bez uzupełniania) dla której wykresy przedstawiono wcześniej na rys. 6.51d i 6.51h. Jak widać na wykresach ATE zaproponowana metoda uzupełniania głębi na podstawie danych RGB-D pochodzących z douczonej sieci działa lepiej, niż ta wykorzystująca do tego celu tylko dane RGB.

Wnioski

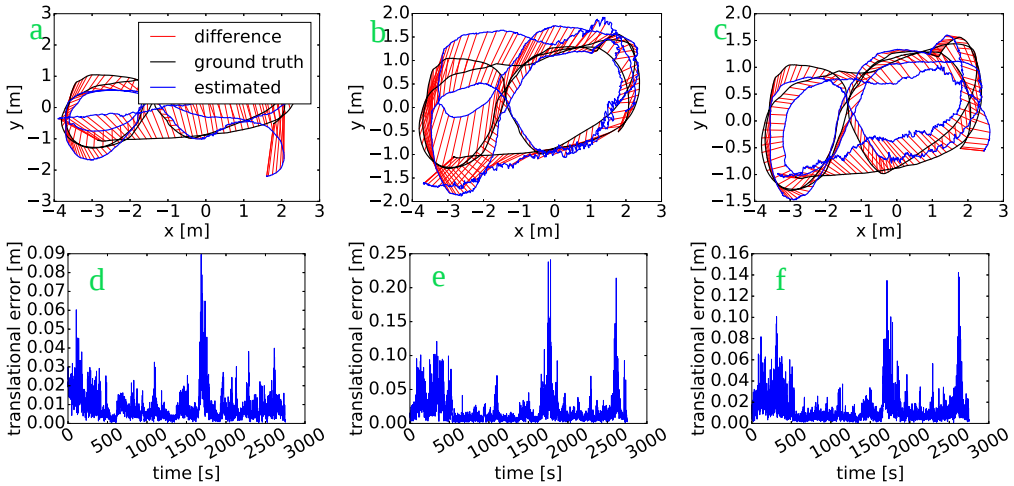
Powyższe badania pokazały, że korzystając z metod głębokiego uczenia da się wykorzystać informacje dostarczane przez lepszy sensor w budżetowej wersji systemu RGB-D VO. Ukazuje to potencjał wykorzystania tego rodzaju skrośnego uczenia do poprawy nawigacji budżetowych robotów, poruszających się wewnątrz zamkniętych przestrzeni. Zaproponowana metoda głębokiego uczenia pozwala na osiągnięcie lepszych rezultatów, niż klasyczne metody nie wykorzystujące kontekstu RGB-D. Uczenie



Rysunek 6.54: Wykresy błędów ATE (a, b, c) oraz błędów translacji RPE (d, e, f) dla sekwencji *putkk_Dataset_1_Kin_1*. (e, f, g, h) Wykresy błędów translacji RPE. (a, d) VO z oryginalnymi obrazami bez uzupełniania, (b, e) VO z uzupełnianiem RGB, albo (c, f) VO z uzupełnianiem RGB-D



Rysunek 6.55: Wykresy błędów ATE (a, b, c) oraz błędów translacji RPE (d, e, f) dla sekwencji *putkk_Dataset_2_Kin_1*. (e, f, g, h) Wykresy błędów translacji RPE. (a, d) VO z oryginalnymi obrazami bez uzupełniania, (b, e) VO z uzupełnianiem RGB, albo (c, f) VO z uzupełnianiem RGB-D



Rysunek 6.56: Wykresy błędów ATE (a, b, c) oraz błędów translacji RPE (d, e, f) dla sekwencji *putkk_Dataset_3_Kin_1*. (e, f, g, h) Wykresy błędów translacji RPE. (a, d) VO z oryginalnymi obrazami bez uzupełniania, (b, e) VO z uzupełnianiem RGB, albo (c, f) VO z uzupełnianiem RGB-D

modelu przy wspólnym wykorzystaniu map głębi i zdjęć RGB oraz wykorzystanie lepszych, bardziej kompletnych map głębi, jako dane odniesienia dla funkcji błędu, pozwalają modelowi zawrzeć w sobie zależności pomiędzy wyglądem obiektów a brakującymi przestrzeniami w mapach głębi Kinecta v1. Choć wykorzystanie tylko danych RGB nie jest aż tak korzystne, to nadal jest warte rozważenia przy ograniczonej ilości informacji o odległości.

Wykorzystanie sztucznej sieci neuronowej do uzupełniania braków w mapach głębi wszędzie zwiększyło błędy RPE. Dane głębi po inferencji cechowały się pewną zmiennością z klatki na klatkę. Nie zawsze odzyskane na poprzednim ujęciu nogi od krzeseł znajdowały się na kolejnym i vice versa. Obrysy niektórych przedmiotów np. krzeseł potrafiły zajmować raz większą, a raz mniejszą przestrzeń, raz być jaśniejsze, a raz ciemniejsze, choć kamera się zbliżała i powinny się ściemniać (bo ciemniejszy kolor odpowiada krótszemu dystansowi do obiektu). Najprawdopodobniej wynika to z danych RGB i zmienności naświetlenia. Jest to pewnego rodzaju przewaga klasycznych metod polegająca na tym, że powstałe obrazy cechują się mniejszą wariancją a propagowane izofoty mogą odtworzyć narożniki.

Z tego wynika również jak istotne jest zebranie odpowiedniego zesta-

wu uczącego. Jak widać drobne zmiany oświetlenia mają wpływ na odtwarzane odległości. Dlatego, do poprawnego uczenia należy wykorzystać taki zestaw danych, w którym zdjęcia RGB i D są pobierane w dokładnie tym samym czasie. W przypadku prezentowanych badań tak też zostało uczynione, mimo że głównym kryterium była dostępność danych z lepszego sensora. Należy zaznaczyć, że choć w PUTKK [166] jest to dobrze zrobione, to nie jest to standardem i w innych jak np. TUM RGB-D [225], tak już nie jest to zrobione.

Przeprowadzone eksperymenty świadczą o poprawie jakości map głębi oraz wzroście wiarygodności i dokładności odometrii wizyjnej, jaką osiąga się poprzez zastosowanie metod głębokiego uczenia. Problemem zaproponowanego podejścia jest jednak to, że nasz model zbytnio dopasował się do wykorzystywanego w uczeniu środowiska (tj. został przeuczony) i niekoniecznie musi być właściwy w innych środowiskach. Być może zwiększenie modelu mogłoby w tym wypadku pomóc. Parametry detektora punktów kluczowych znalezione przy pomocy PSO/EA z ulepszonymi mapami głębi różnią się od tych znalezionych dla oryginalnych danych, pozwalając na wykrycie większej ilości punktów. Takie parametry również nie generalizują się dobrze dla różnych scen, ale wykorzystana metoda jest efektywniejsza niż standardowe podejścia.

6.7 Ocena wydajności w kontekście czasu obliczeń

6.7.1 Algorytmów ewolucyjnych

Optymalizacje algorytmami populacyjnymi bazującymi na niemodyfikowanych danych dostarczanych przez Kinecta v1 dokonywano na komputerze wyposażonym w procesor Intela i7-5820K ($3,30$ GHz) oraz kartę graficzną NVIDIA GeForce GTX 750 Ti. W wczesnych eksperymentach optymalizacja za pomocą PSO (z 20 iteracjami) była ok. pięciokrotnie dłuższa niż z wykorzystaniem EA (z 20 pokoleniami)–58 godzin do 11 dla funkcji celu bazującej na ATE. Poza tym konfiguracje z funkcją celu bazującą na metryce RPE wymagały ok. 10% mniej czasu, niż te bazujące na metryce ATE.

W późniejszych eksperymentach zmieniono liczbę iteracji w poszczególnych pętlach algorytmu RANSAC: w pierwszej z 10000 do 315, a drugiej z 10000 do 450. Optymalizacja PSO dla progu detekcji τ_A trwała 9 itera-

cji, a EA 20 pokoleń. Optymalizacja PSO ATE trwała $72575,2$ms, PSO RPE $69545,7$ms, EA ATE $282587,5$ms, a EA RPE $291178,9$ms. W obydwu przypadkach optymalizacje z wykorzystaniem błędu RPE trwały odpowiednio o ok. 4% krócej i 3% dłużej niż ich odpowiednie części ATE. Nie widać tu znacznej różnicy w długości. Optymalizacja PSO była tu ok. czterokrotnie szybsza.

To które podejście jest szybsze zależy od specyfiki danego problemu optymalizacyjnego. Algorytm rojowy zdaje się mieć przewagę, w przypadku gdy optimum znajduje się blisko granicy zakresu jednego z parametrów. Może to wynikać z faktu, że na uzyskanie takiego samego efektu algorytm ewolucyjny musi poświęcić wiele pokoleń, aby zdążyła w nim wystąpić odpowiednia mutacja. W badanych zagadnieniach algorytm rojowy pozwalał na znalezienie lepszych zestawów parametrów.

6.7.2 Uzupełniania map głębi

Uzupełnianie map głębi jest dodatkową czynnością wykonywaną w trakcie przetwarzania danych z Kinecta w zaproponowanej prostej odometrii wizyjnej VO. Tab. 6.32 zawiera czasy potrzebne na uzupełnienie map głębi sekwencji *putkk_Dataset_5_Kin_1* dla trzech metod: Telei [229], Navier-Stokesa [105] oraz zaproponowanego podejścia wykorzystującego do tego sztuczną sieć neuronową działającą z podawanymi na wejście obrazami RGB-D. Te wyniki uzyskano na komputerze działającym pod systemem Linux z procesorem Intel i7-6820HQ ($2,7$ GHz) oraz kartą graficzną NVIDIA Quadro M2000M. Warto zaznaczyć, że tradycyjne metody uzupełniania głębi wykorzystujące do obliczeń procesor nie są w stanie rywalizować, pod względem czasu potrzebnego na obliczenia, z czasem inferencji sieci CNN pracującej na karcie graficznej. Dodatkowy czas na uzupełnienie jest także niewielki, w porównaniu do czasu przetwarzania pojedynczego ujęcia przez system VO, który działa z prędkością ok. 15 fps (ang. *frames per second*) klatek na sekundę, gdy działa na nieulepszonych, oryginalnych danych z Kinecta v1.

Tabela 6.32: Czasy przetwarzania różnych metod uzupełniania głębi sekwencji *putkk_Dataset_5_Kin_1*

	putkk_Dataset_5_Kin_1		
Czas uzupełniania głębi	NS	Telea	Inferencja douczonej sieci
Wartość średnia [ms]	221,0	220,9	6,5
Odchylenie standardowe [ms]	33,3	33,3	1,9

Rozdział 7

Podsumowanie i wnioski

7.1 Podsumowanie rezultatów badań

Celem niniejszej pracy było opracowanie i ocena nowych podejść do estymacji trajektorii oraz poprawy jakości danych zbieranych przez systemy odometrii wizyjnej lub SLAM. Przeprowadzone badania koncentrowały się na optymalizacji parametrów algorytmów oraz wykorzystaniu sztucznych sieci neuronowych do uzupełniania braków w mapach głębi, a także na analizie wpływu różnych metod na dokładność i efektywność systemów stosowanych w robotyce mobilnej.

Badania nad parami detektor–deskryptor wykazały, że algorytmy SURF i AKAZE są najlepsze w zadaniu estymacji trajektorii, zapewniając cechy pozwalające na dokładne oszacowanie przebytej drogi w porównywalnym czasie. Algorytm ORB, choć nieco mniej precyzyjny, okazał się najszybszy, jednak jego wydajność była ograniczana przez procedurę RANSAC, skoncentrowaną na uzyskaniu najlepszych wyników, co powodowało wielokrotne próby estymacji ruchu kamery między sąsiednimi klatkami. Algorytmy KAZE i BRISK uznano za mniej efektywne w porównaniu z wymienionymi wcześniej. Wnioski te są szczególnie istotne w kontekście zastosowań, gdzie precyzja trajektorii jest kluczowa, a czas obliczeń ma istotny wpływ na działanie systemu.

Architektury systemów RGB-D SLAM, które wykorzystują dane zebrane przez kamery głębi, nie zawsze sprawdzają się w rzeczywistych warunkach, zwłaszcza gdy robot mobilny wykonuje gwałtowne, nieprzewidywalne ruchy. Ruchy te mogą prowadzić do pogorszenia jakości obrazów,

co z kolei negatywnie wpływa na algorytmy SLAM bazujące na cechach punktowych. Problemy te są szczególnie widoczne w kontekście założeń dotyczących precyzyjnego umiejscowienia punktów cech na obrazie oraz prostych modeli ruchu sensora. W badaniach wykazano, że stabilniejszy ruch kamery prowadzi do mniejszych błędów estymacji trajektorii, co sugeruje konieczność zintegrowania algorytmu sterującego ruchem robota z procesem akwizycji danych. Najlepsze wyniki osiągają systemy SLAM, które korzystają z optymalizacji grafowej i cech punktowych wysokiej jakości, co podkreśla znaczenie stabilności ruchu robota w kontekście jakości wyników.

Wykorzystanie zależności pomiędzy kolejnymi obrazami RGB-D z danymi głębi, np. poprzez zastosowanie algorytmu Kabscha, pozwala na uzyskanie bardziej precyzyjnych estymacji trajektorii metodą klatka po klatce w systemach odometrii wizyjnej. W przypadku systemów pasywnych, które korzystają wyłącznie z jednej kamery, najlepszym okazał się algorytm 8-punktowy, jako jedyny zdolny do poprawnego odtworzenia przebytej trajektorii. Założenia tego algorytmu są bardziej realistyczne w kontekście rzeczywistych danych niż założenia algorytmu 5-punktowego. Wyniki te podkreślają znaczenie wyboru odpowiedniej pary detektor–deskryptor oraz zastosowanego algorytmu, które mają kluczowy wpływ na jakość estymowanej trajektorii w różnych środowiskach.

Optymalizacja parametrów w prostym algorytmie odometrii wizyjnej za pomocą algorytmów populacyjnych, dostosowana do specyfiki danych wejściowych, pozwala na znaczącą poprawę uzyskiwanych wyników. W szczególności, parametry związane z procedurą RANSAC okazały się stabilne i mogą być stosowane w różnych zestawach danych, co jest kluczowe dla osiągnięcia odpowiedniego kompromisu między liczbą przetwarzanych cech punktowych a dokładnością ich dopasowywania. Z kolei parametry detektora cech kluczowych mogą się różnić w zależności od środowiska, co sugeruje, że optymalizację można przeprowadzać na kilka sposobów, dostosowując ją do specyfiki zadania. Ponadto, algorytm PSO lepiej radzi sobie niż EA w sytuacjach, gdy optymalne wartości parametrów znajdują się blisko wartości granicznych (np. parametry detektora cech), co jest kluczowe w procesie optymalizacji parametrów systemów odometrii wizyjnej.

Optymalizacja parametrów VO w trybie online wykazała tylko nieznaczną poprawę wyników, jednak otwiera to możliwości dalszych badań nad tym zagadnieniem. Analiza statystyczna wyników sugeruje, że rozkłady

prawdopodobieństwa względnych błędów (RPE) oszacowania przemieszczenia sensora nie są idealnie Gaussowskie, co wskazuje na obecność dominujących źródeł błędów systematycznych. Wiedza ta może być wykorzystana do oceny wiarygodności rozwiązań proponowanych przez algorytm RANSAC oraz do dalszej optymalizacji tego procesu.

W niektórych przypadkach uzupełnianie brakujących danych głębi za pomocą sztucznych sieci neuronowych może przynieść korzyści, jednak proste podejścia do tego zagadnienia mogą pogorszyć wyniki. Badania wykazały, że dane zebrane przez nowsze kamery, takie jak Kinect v2, mogą poprawić jakość estymacji trajektorii przy użyciu starszych sensorów, takich jak Kinect v1. Zastosowanie metod głębokiego uczenia pozwala na osiągnięcie lepszych rezultatów niż klasyczne podejścia, które nie uwzględniają kontekstu RGB-D. To pokazuje, że nowoczesne metody przetwarzania danych mają potencjał do znacznej poprawy wyników w systemach odometrii wizyjnej.

Warto również zauważyć, że dane głębi uzyskane w wyniku inferencji mogą charakteryzować się zmiennością pomiędzy kolejnymi klatkami, co może wpłynąć na dokładność trajektorii. Badania nad zastosowaniem sztucznych sieci neuronowych do uzupełniania braków w mapach głębi podkreślają, jak ważne jest zbudowanie dobrej jakości zbioru uczącego. Przykładem może być zbiór PUTKK, który dzięki synchronizacji obrazów RGB z obrazami głębi zredukował wpływ zmieniającego się oświetlenia. Rozważenie większego modelu sieci neuronowej mogłoby zapobiec nadmiernemu dopasowaniu do środowiska uczącego.

7.2 Wnioski w kontekście postawionych tez szczegółowych

Z przeprowadzonych badań wynika, że automatyczna optymalizacja parametrów w prostym algorytmie odometrii wizyjnej, dostosowana do specyfiki danych wejściowych, jest możliwa i efektywna przy użyciu algorytmów populacyjnych. Proces ten znacząco redukuje nakład pracy ludzkiej i prowadzi do uzyskania stabilnych parametrów, które mogą być stosowane w różnych zestawach danych zebranych w podobnych środowiskach. Dodatkowo, optymalizacja parametrów systemu VO może być przeprowadzana na różne sposoby, co daje możliwość dostosowania algorytmów

do specyficznych warunków operacyjnych.

Zastosowanie metod głębokiego uczenia do uzupełniania brakujących danych okazało się korzystne, szczególnie gdy dane zebrane za pomocą nowszych kamer są wykorzystywane do poprawy jakości danych zebranych przez starsze sensory. Zaproponowana metoda uczenia przyniosła lepsze wyniki niż klasyczne podejścia, które nie uwzględniają kontekstu RGB-D, co pokazuje potencjał sztucznych sieci neuronowych w poprawie jakości trajektorii.

Powyższe wnioski częściowe sugerują prawdziwość tezy mówiącej, że zastosowanie populacyjnych metod optymalizacji oraz wybranych algorytmów uczenia maszynowego pozwala zoptymalizować strukturę i parametry systemu lokalizacji RGB-D, ze szczególnym uwzględnieniem lokalnego ruchu sensora.

7.3 Propozycje dalszego rozwoju przedstawionej koncepcji

W przyszłych badaniach warto rozważyć zastosowanie innych kombinacji detektorów i deskryptorów, w tym nowo powstałych. Badania wykazały, że metody n-punktowe, choć mniej efektywne od algorytmów uwzględniających mapy głębi, mogą być dalej rozwijane w kontekście poprawy estymacji trajektorii. Jednym z głównych problemów była trudność w określeniu współczynnika skali przesunięcia oraz jego stabilności w trakcie całej trajektorii. Choć estymowane mapy głębi na podstawie danych RGB, uzyskane za pomocą sztucznych sieci neuronowych, cechują się pewną zmiennością, skumulowany efekt wielu punktów pozwala na zmniejszenie losowości uzyskanych wyników. Sugeruje to możliwość wykorzystania sztucznych sieci neuronowych do odzyskiwania współczynnika skali, co mogłoby umożliwić fuzję algorytmu n-punktowego z algorytmem Kabscha oraz poprawę ogólnej precyzji systemu.

Dalsze badania mogą również skupić się na mniej zbadanych aspektach systemu, takich jak metody filtracji punktów kluczowych, optymalizacja algorytmów detekcji i deskrypcji w trybie online, czy też zastosowanie sztucznych sieci neuronowych do poprawy jakości obrazu RGB. Warto również rozważyć poszerzenie kąta widzenia starszych kamer, co mogłoby pozytywnie wpłynąć na jakość odtwarzanej trajektorii oraz umożliwić

lepszą obsługę brzegów obrazu (ang. *padding*).

Badania przeprowadzone na robocie kroczącym Messor II ukazały wyzwania związane z akwizycją danych przez tego typu urządzenia. W przyszłości warto stworzyć specjalny, dokładny zestaw danych, który umożliwiłby zastosowanie metod głębokiego uczenia do poprawy jakości obrazów zebranych w trudnych warunkach. Taki zestaw danych mógłby znacząco przyczynić się do poprawy wyników estymacji trajektorii w robotyce mobilnej.

Bibliografia

[1] https://en.wikipedia.org/wiki/Refractive_index.

[2] https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies.

[3] http://msdn.microsoft.com/en-us/library/hh438998.aspx. Microsoft.

[4] http://www.depthbiomechanics.co.uk/?cat=18.

[5] http://structure.io/. Structure.

[6] http://en.wikipedia.org/wiki/ZCam. Wikipedia.

[7] https://razorvision-blog.tumblr.com/post/15039827747/how-kinect-and-kinect-fusion-kinfu-work. Razorvision.

[8] https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[9] https://pl.wikipedia.org/wiki/Robot.

[10] https://pl.wikipedia.org/wiki/Josef_%C4%8Capek.

[11] https://pl.wikipedia.org/wiki/Robot_mobilny.

[12] https://en.wikipedia.org/wiki/Visual_odometry.

[13] https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping.

[14] https://github.com/CCNYRoboticsLab/ccny_rgbd_tools.

[15] https://github.com/felixendres/rgbdslam_v2.

[16] https://en.wikipedia.org/wiki/Random_sample_consensus.

[17] https://www.anaconda.com/.

[18] https://jupyter.org/.

[19] https://chpatrick.github.io/flann.

[20] https://aigeekprogrammer.com/pl/konwolucyjne-sieci-neuronowe-klasyfikacja-obrazow-czesc-2/.

[21] https://aigeekprogrammer.com/pl/konwolucyjne-sieci-neuronowe-klasyfikacja-obrazow-czesc-3/.

[22] https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning.

[23] https://twitter.com/soumithchintala/status/963072442510974977?lang=en.

[24] http://caffe.berkeleyvision.org/.

[25] https://github.com/BVLC/caffe/graphs/contributors.

[26] https://www.nvidia.com/en-au/data-center/gpu-accelerated-applications/caffe/.

[27] https://github.com/NVIDIA/caffe.

[28] http://daggerfs.com/.

[29] https://caffe2.ai/.

[30] https://github.com/facebookarchive/caffe2/tree/master.

[31] http://torch.ch.

[32] https://github.com/torch/torch7.

[33] https://developer.nvidia.com/deep-learning-frameworks.

[34] https://pytorch.org/.

[35] https://github.com/pytorch/pytorch.

[36] https://numpy.org/.

[37] https://cloud.google.com/blog/products/ai-machine-learning/introducing-pytorch-across-google-cloud.

[38] https://www.fast.ai/.

[39] https://github.com/fastai/fastai.

[40] https://www.tensorflow.org/.

[41] https://forums.fast.ai/t/fastai-vs-keras-tensorflow/14257/9.

[42] https://github.com/tensorflow/tensorflow.

[43] https://keras.io/.

[44] https://www.tensorflow.org/community/roadmap.

[45] https://github.com/keras-team/keras.

[46] https://github.com/PaddlePaddle/Paddle.

[47] https://www.quora.com/What-do-you-think-of-Baidus-open-source-deep-learning-platform-PaddlePaddle%EF%BC%9F.

[48] https://www.techinasia.com/baidus-deep-learning-framework-give-china-leg-ai-race.

[49] https://deeplearning4j.org/.

[50] https://github.com/deeplearning4j/deeplearning4j/tree/master/deeplearning4j.

[51] https://www.mathworks.com/campaigns/products/trials/targeted/dpl.html.

[52] https://www.mathworks.com/solutions/deep-learning.html?s_tid=hp_brand_deeplearning.

[53] https://www.h2o.ai/.

[54] https://github.com/VowpalWabbit/vowpal_wabbit.

[55] https://spark.apache.org/docs/latest/ml-guide.html.

[56] https://software.intel.com/en-us/articles/bigdl-distributed-deep-learning-on-apache-spark.

[57] https://github.com/Microsoft/CNTK.

[58] https://www.microsoft.com/en-us/cognitive-toolkit/.

[59] https://www.cnbc.com/2018/11/15/microsoft-allies-with-facebook-on-pytorch-onnx-ai-software.html.

[60] http://vertex.ai/.

[61] https://ai.intel.com/reintroducing-plaidml/.

[62] https://github.com/plaidml/plaidml.

[63] http://deeplearning.net/software/theano/.

[64] https://medium.com/@pymc_devs/theano-tensorflow-and-the-future-of-pymc-6c9987bb19d5.

[65] https://pymc-devs.github.io/pymc/README.html#purpose.

[66] https://chainer.org/.

[67] https://github.com/chainer/chainer.

[68] https://mxnet.apache.org.

[69] https://github.com/awslabs/keras-apache-mxnet.

[70] https://github.com/apache/incubator-mxnet.

[71] https://mxnet.apache.org/versions/master/gluon/index.html.

[72] https://github.com/amzn/amazon-dsstne/blob/master/docs/getting_started/userguide.md.

[73] https://developer.nvidia.com/digits.

[74] https://github.com/NVIDIA/DIGITS.

[75] https://sonnet.readthedocs.io/en/latest/.

[76] https://eng.uber.com/horovod/.

[77] https://software.intel.com/en-us/ai-academy/frameworks/neon.

[78] https://github.com/dirkneumann/deepdist.

[79] https://docs.databricks.com/applications/deep-learning/deep-learning-pipelines.html.

[80] https://github.com/amplab/SparkNet.

[81] https://github.com/autumnai/leaf.

[82] https://github.com/keras-team/autokeras.

[83] https://github.com/melodyguan/enas.

[84] https://github.com/carpedm20/ENAS-pytorch.

[85] http://persci.mit.edu/pub_pdfs/InternationalSymposiumIntelligentRobotics1993.pdf.

[86] https://aigeekprogrammer.com/pl/konwolucyjne-sieci-neuronowe-klasyfikacja-obrazow-czesc-1/.

[87] http://lrm.put.poznan.pl.

[88] https://cvg.cit.tum.de/data/datasets/rgbd-dataset/tools#evaluation.

[89] https://www.ros.org/.

[90] Pablo Fernández Alcantarilla, Adrien Bartoli i Andrew J. Davison. "KAZE Features". W: *Computer Vision – ECCV 2012*. Wyed. Andrew Fitzgibbon i in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 214–227. ISBN: 978-3-642-33783-3.

[91] Mauro Annunziato i Stefano Pizzuti. "Adaptive Parameterization of Evolutionary Algorithms Driven by Reproduction and Competition". W: *In Proceedings of ESIT-2000*. 2000.

[92] Relja Arandjelović i Andrew Zisserman. "Three things everyone should know to improve object retrieval". W: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, s. 2911–2918. DOI: 10.1109/CVPR.2012.6248018.

[93] Amir Atapour-Abarghouei i Toby P. Breckon. "Dealing with Missing Depth: Recent Advances in Depth Image Completion and Estimation". W: *RGB-D Image Analysis and Processing*. Cham: Springer, 2019, s. 15–50.

[94] Amir Atapour-Abarghouei, Gregoire Payen de La Garanderie i Toby P. Breckon. "Back to Butterworth - a Fourier basis for 3D surface relief hole filling within RGB-D imagery". W: *23rd International Conference on Pattern Recognition (ICPR)*. 2016, s. 2813–2818.

[95] Arif Ayoppan. "A Genetic Algorithm with Online Learning Approach for Improving Loop Closure Detection of a Visual SLAM". W: *International Journal of Advanced Trends in Computer Science and Engineering* 8 (12/2019), s. 159–166.

[96] T. Bailey i H. Durrant-Whyte. "Simultaneous localization and mapping (SLAM): part II". W: *IEEE Robotics & Automation Magazine* 13.3 (2006), s. 108–117. DOI: 10.1109/MRA.2006.1678144.

[97] Herbert Bay i in. "Speeded-Up Robust Features (SURF)". W: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, s. 346–359. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2007.09.014. URL: https://www.sciencedirect.com/science/article/pii/S1077314207001555.

[98] Dominik Belter, Michał Nowicki i Piotr Skrzypczyński. “Evaluating Map-Based RGB-D SLAM on an Autonomous Walking Robot”. W: *Challenges in Automation, Robotics and Measurement Techniques*. Wyed. Roman Szewczyk, Cezary Zieliński i Małgorzata Kaliczyńska. Cham: Springer International Publishing, 2016, s. 469–481. ISBN: 978-3-319-29357-8.

[99] Dominik Belter, Michał Nowicki i Piotr Skrzypczyński. “On the Performance of Pose-Based RGB-D Visual Navigation Systems”. W: *Computer Vision – ACCV 2014*. Wyed. Daniel Cremers i in. Cham: Springer International Publishing, 2015, s. 407–423. ISBN: 978-3-319-16808-1.

[100] Dominik Belter i Piotr Skrzypczynski. “A biologically inspired approach to feasible gait learning for a hexapod robot”. W: *Applied Mathematics and Computer Science* 20 (3/2010), s. 69–84. DOI: 10.2478/v10006-010-0005-7.

[101] Dominik Belter i Piotr Skrzypczynski. “Precise self-localization of a walking robot on rough terrain using parallel tracking and mapping”. W: *Industrial Robot: An International Journal* 40 (4/2013). DOI: 10.1108/01439911311309924.

[102] Dominik Belter i Piotr Skrzypczyński. “Population-based Methods for Identification and Optimization of a Walking Robot Model”. W: *Robot Motion and Control 2009*. Wyed. Krzysztof R. Kozłowski. London: Springer London, 2009, s. 185–195. ISBN: 978-1-84882-985-5.

[103] Dominik Belter i Piotr Skrzypczyński. “The importance of measurement uncertainty modelling in the feature-based RGB-D SLAM”. W: *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*. 2015, s. 308–313. DOI: 10.1109/RoMoCo.2015.7219752.

[104] Dominik Belter i in. “Lightweight RGB-D SLAM system for search and rescue robots”. W: t. 351. 1/2015, s. 11–21. ISBN: 9783319158464. DOI: 10.1007/978-3-319-15847-1_2.

[105] M. Bertalmio, A.L. Bertozzi i G. Sapiro. “Navier-Stokes, fluid dynamics, and image and video inpainting”. W: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. T. 1. 2/2001, s. 355–362. DOI: 10.1109/CVPR.2001.990497.

[106] Gabriele Berton, Carlo Masone i Barbara Caputo. *Rethinking Visual Geo-localization for Large-Scale Applications*. 2022. arXiv: 2204.02287 [cs.CV]. URL: https://arxiv.org/abs/2204.02287.

[107] G. Bradski. "The OpenCV Library". W: *Dr. Dobb's Journal of Software Tools* (2000).

[108] Gary Bradski i Adrian Kaehler. *Learning OpenCV - computer vision with the OpenCV library: software that sees*. First. O'Reilly Media, Inc., 1/2008. ISBN: 978-0-596-51613-0.

[109] Carlos Campos i in. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM". W: *IEEE Transactions on Robotics* 37.6 (12/2021), s. 1874–1890. ISSN: 1941-0468. DOI: 10.1109/tro.2021.3075644. URL: http://dx.doi.org/10.1109/TRO.2021.3075644.

[110] Augusto R. Castro., Valdir Grassi Jr.. i Moacir A. Ponti. "Deep Depth Completion of Low-cost Sensor Indoor RGB-D using Euclidean Distance-based Weighted Loss and Edge-aware Refinement". W: *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - Volume 4: VISAPP*. 2022, s. 204–212.

[111] Chongyu Chen i in. "Kinect Depth Recovery Using a Color-Guided, Region-Adaptive, and Depth-Selective Framework". W: *ACM Trans. Intell. Syst. Technol.* 6.2 (2015).

[112] Ondřej Chum, Jiri Matas i Josef Kittler. "Locally Optimized RANSAC". W: *DAGM-Symposium*. T. 2781. 9/2003, s. 236–243. ISBN: 978-3-540-40861-1. DOI: 10.1007/978-3-540-45243-0_31.

[113] Petr Cížek i Jan Faigl. "On localization and mapping with RGB-D sensor and hexapod walking robot in rough terrains". W: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2016, s. 002273–002278. DOI: 10.1109/SMC.2016.7844577.

[114] Brian Curless i Marc Levoy. *A Volumetric Method for Building Complex Models from Range Images*. https://graphics.stanford.edu/papers/volrange/volrange.pdf. Uniwersytet w Stanford, 1996.

[115] Davison. “Real-time simultaneous localisation and mapping with a single camera”. W: *Proceedings Ninth IEEE International Conference on Computer Vision*. T. 2. 2003, s. 1403–1410. DOI: 10.1109/ICCV.2003.1238654.

[116] A.J. Davison i D.W. Murray. “Simultaneous localization and map-building using active vision”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (2002), s. 865–880. DOI: 10.1109/TPAMI.2002.1017615.

[117] Daniel DeTone, Tomasz Malisiewicz i Andrew Rabinovich. *SuperPoint: Self-Supervised Interest Point Detection and Description*. 2018. arXiv: 1712.07629 [cs.CV].

[118] Ivan Dryanovski, Roberto G. Valenti i Jizhong Xiao. “Fast visual odometry and mapping from RGB-D data”. W: *2013 IEEE International Conference on Robotics and Automation*. 2013, s. 2305–2310. DOI: 10.1109/ICRA.2013.6630889.

[119] R. Eberhart i J. Kennedy. “A new optimizer using particle swarm theory”. W: *MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 1995, s. 39–43. DOI: 10.1109/MHS.1995.494215.

[120] David Eigen, Christian Puhrsch i Rob Fergus. “Depth map prediction from a single image using a multi-scale deep network”. W: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. Montreal, Canada, 2014, s. 2366–2374.

[121] Felix Endres i in. “3-D Mapping With an RGB-D Camera”. W: *IEEE Transactions on Robotics* 30.1 (2/2014), s. 177–187. DOI: 10.1109/TRO.2013.2279412.

[122] Felix Endres i in. “An evaluation of the RGB-D SLAM system”. W: *2012 IEEE International Conference on Robotics and Automation*. 2012, s. 1691–1696. DOI: 10.1109/ICRA.2012.6225199.

[123] Björn Engquist i S. Osher. “Stable and entropy satisfying approximations for transonic flow calculations”. W: *Mathematics of Computation* 34 (1980), s. 45–75.

[124] Juan M. Falquez, Michael Kasper i Gabe Sibley. “Inertial aided dense & semi-dense methods for robust direct visual odometry”. W: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, s. 3601–3607. DOI: 10.1109/IROS.2016.7759530.

[125] Cheng Feng i in. *Real-time Monocular Depth Estimation on Embedded Systems*. 2024. arXiv: 2308.10569 [cs.CV].

[126] Sílvio Filipe i Luís Alexandre. “A Comparative Evaluation of 3D Keypoint Detectors”. W: *9th Conference on Telecommunications – ConfTele 2013*. T. 1. Castelo Branco, Portugal, 5/2013, s. 145–148.

[127] Sílvio Filipe i Luís A. Alexandre. “A comparative evaluation of 3D keypoint detectors in a RGB-D Object Dataset”. W: *Proceedings of the 9th International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP, (VISIGRAPP 2014)*. T. 1. INSTICC. SciTePress, 2014, s. 476–483. ISBN: 978-989-758-003-1. DOI: 10.5220/0004679904760483.

[128] Thomas Fischer i in. “Stereo vision-based localization for hexapod walking robots operating in rough terrains”. W: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), s. 2492–2497.

[129] Martin A. Fischler i Robert C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. W: *Communications of the ACM* 24.6 (1981), s. 381–395. DOI: 10.1145/358669.358692. URL: https://app.dimensions.ai/details/publication/pub.1033921345.

[130] Friedrich Fraundorfer i Davide Scaramuzza. “Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications”. W: *IEEE Robotics & Automation Magazine* 19 (2012), s. 78–90. DOI: 10.1109/MRA.2012.2182810.

[131] Wolfram Burgard Gian Diego Tipaldi. *Robot Mapping FastSLAM–Feature-Based SLAM with Particle Filters*. http://ais.informatik.uni-freiburg.de/teaching/ws17/mapping/pdf/slam12-fastslam.pdf. Uniwersytet w Freiburgu.

[132] Clément Godard i in. *Digging Into Self-Supervised Monocular Depth Estimation*. 2019. arXiv: 1806.01260 [cs.CV].

[133] Jaroslaw Goslinski, Michał Nowicki i Piotr Skrzypczynski. “Performance Comparison of EKF-Based Algorithms for Orientation Estimation on Android Platform”. W: *Sensors Journal, IEEE* 15 (7/2015), s. 3781–3792. DOI: 10.1109/JSEN.2015.2397397.

[134] Giorgio Grisetti i in. “A Tutorial on Graph-Based SLAM”. W: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), s. 31–43. DOI: 10.1109/MITS.2010.939925.

[135] Georg Halmetschlager-Funek i in. “An Empirical Evaluation of Ten Depth Cameras: Bias, Precision, Lateral Noise, Different Lighting Conditions and Materials, and Multiple Sensor Setups in Indoor Environments”. W: *IEEE Robotics & Automation Magazine* 26.1 (2019), s. 67–77.

[136] Dongxiao Han i in. “Multi-Objective Optimization of Loop Closure Detection Parameters for Indoor 2D Simultaneous Localization and Mapping”. W: *Sensors* 20.7 (2020).

[137] Ankur Handa i in. “A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM”. W: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, s. 1524–1531. DOI: 10.1109/ICRA.2014.6907054.

[138] Richard Hartley i Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2 wyd. Cambridge University Press, 2004.

[139] Kaiming He i in. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[140] H. Hirschmuller, P.R. Innocent i J.M. Garibaldi. “Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics”. W: *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002*. T. 2. 2002, 1099–1104 vol.2. DOI: 10.1109/ICARCV.2002.1238577.

[141] Nghia Ho. *Finding optimal rotation and translation between corresponding 3D points*. http://nghiaho.com/?pageid=671. 2011.

[142] Berthold K. P. Horn. “Closed-form solution of absolute orientation using unit quaternions”. W: *Journal of the Optical Society of America A* 4.4 (1987), s. 629–642.

[143] Armin Hornung i in. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. W: *Autonomous Robots* 34.3 (4/2013), s. 189–206. DOI: 10.1007/s10514-012-9321-0.

[144] Andrew Howard. “Real-time stereo visual odometry for autonomous ground vehicles”. W: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, s. 3946–3952. DOI: 10.1109/IROS.2008.4651147.

[145] Jeremy Howard i in. *fastai*. https://github.com/fastai/fastai. GitHub, 2018.

[146] Jeremy Howard i Sylvain Gugger. “FastAi: A Layered API for Deep Learning”. W: *Information* 11.2 (2020), s. 108.

[147] Albert S. Huang i in. “Visual odometry and mapping for autonomous flight using an RGB-D camera”. W: *In Proc. of the International Symposium on Robotics Research (ISRR)*. Flagstaff, Arizona, USA, 2011.

[148] Martin Humenberger i in. *Methods for visual localization*. NAVER LABS Europe, 2021. URL: https://europe.naverlabs.com/blog/methods-for-visual-localization/.

[149] Ł. Ilnicki i B. Siemiątkowska. “Algorytm łączenia chmur punktów na podstawie informacji wizualno-metrycznej”. Polish. W: *Prace Naukowe Politechniki Warszawskiej. Elektronika* z. 182, t. 1 (2012), s. 207–216.

[150] Sergey Ioffe i Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: http://arxiv.org/abs/1502.03167.

[151] Nick Jacobi, Phil Husbands i Inman Harvey. “Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics”. W: *Proceedings of the Third European Conference on Advances in Artificial Life*. Berlin, Heidelberg: Springer-Verlag, 1995, s. 704–720. ISBN: 3540594965.

[152] Haifeng Jin, Qingquan Song i Xia Hu. “Auto-Keras: An Efficient Neural Architecture Search System”. W: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, s. 1946–1956.

[153] W. Kabsch. “A discussion of the solution for the best rotation to relate two sets of vectors”. W: *Acta Crystallographica Section A* 34.5 (9/1978), s. 827–828. DOI: 10.1107/S0567739478001680. URL: https://doi.org/10.1107/S0567739478001680.

[154] W. Kabsch. “A solution for the best rotation to relate two sets of vectors”. W: *Acta Crystallographica Section A* 32.5 (9/1976), s. 922–923. DOI: 10.1107/S0567739476001873. URL: https://doi.org/10.1107/S0567739476001873.

[155] Alex Kendall, Matthew Grimes i Roberto Cipolla. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*. arXiv, 12/2015. DOI: 10.48550/ARXIV.1505.07427. URL: https://arxiv.org/abs/1505.07427.

[156] Christian Kerl, Jürgen Sturm i Daniel Cremers. “Dense visual SLAM for RGB-D cameras”. W: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, s. 2100–2106. DOI: 10.1109/IROS.2013.6696650.

[157] Christian Kerl, Jürgen Sturm i Daniel Cremers. “Robust odometry estimation for RGB-D cameras”. W: *2013 IEEE International Conference on Robotics and Automation*. 2013, s. 3748–3754. DOI: 10.1109/ICRA.2013.6631104.

[158] Faisal Khan, Saqib Salahuddin i Hossein Javidnia. “Deep Learning-Based Monocular Depth Estimation Methods—A State-of-the-Art Review”. W: *Sensors* 20.8 (2020).

[159] Georg Klein i David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. W: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. 2007, s. 225–234. DOI: 10.1109/ISMAR.2007.4538852.

[160] Laurent Kneip i Paul Timothy Furgale. “OpenGV: A unified and generalized approach to real-time calibrated geometric vision”. W: *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), s. 1–8. DOI: 10.1109/ICRA.2014.6906582.

[161] Laurent Kneip i Simon Lynen. “Direct Optimization of Frame-to-Frame Rotation”. W: *2013 IEEE International Conference on Computer Vision*. 2013, s. 2352–2359. DOI: 10.1109/ICCV.2013.292.

[162] Laurent Kneip, Roland Siegwart i Marc Pollefeys. “Finding the Exact Rotation between Two Images Independently of the Translation”. W: *European Conference on Computer Vision*. Wyed. Andrew Fitzgibbon i in. Berlin, Heidelberg: Springer Berlin Heidelberg, 10/2012, s. 696–709. ISBN: 978-3-642-33782-6. DOI: 10.1007/978-3-642-33783-3_50.

[163] Aleksander Kostusiak. “The Comparison of Keypoint Detectors and Descriptors for Registration of RGB-D Data”. W: *Challenges in Automation, Robotics and Measurement Techniques*. Wyed. Roman Szewczyk, Cezary Zieliński i Małgorzata Kaliczyńska. T. 440. Cham: Springer International Publishing, 4.2.2016, s. 609–622. ISBN: 978-3-319-29357-8. DOI: https://doi.org/10.1007/978-3-319-29357-8_53.

[164] Aleksander Kostusiak, Michał Nowicki i Piotr Skrzypczynski. “On the Application of RGB-D SLAM Systems for Practical Localization of Mobile Robots”. W: *Journal of Automation, Mobile Robotics and Intelligent Systems* 11.2 (6/2017), s. 57–66. DOI: 10.14313/JAMRIS_2-2017/17. URL: https://www.jamris.org/index.php/JAMRIS/article/view/429.

[165] Marek Kraft i in. “Efficient RGB–D data processing for feature–based self–localization of mobile robots”. W: *Int. J. Appl. Math. Comput. Sci.* 26.1 (3/2016), s. 63–79. ISSN: 1641-876X. DOI: 10.1515/amcs-2016-0005. URL: https://doi.org/10.1515/amcs-2016-0005.

[166] Marek Kraft i in. “Towards evaluation of visual navigation algorithms on RGB-D data from the first and second generation Kinect”. W: *Machine Vision and Applications*. T. 28. Springer International Publishing, 2/2016, s. 61–74. DOI: 10.1007/s00138-016-0802-6.

[167] Jan Kukačka, Vladimir Golkov i Daniel Cremers. *Regularization for Deep Learning: A Taxonomy*. 2017. arXiv: 1710.10686 [cs.LG].

[168] Rainer Kümmerle i in. “g^2o: A general framework for graph optimization”. W: *2011 IEEE International Conference on Robotics and Automation*. 2011, s. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

[169] Stefan Leutenegger, Margarita Chli i Roland Y. Siegwart. “BRISK: Binary Robust invariant scalable keypoints”. W: *2011 International Conference on Computer Vision*. 2011, s. 2548–2555. DOI: 10.1109/ICCV.2011.6126542.

[170] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. W: *International Journal of Computer Vision* 60 (2004), s. 91–110.

[171] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints.” W: *International Journal of Computer Vision* 60.2 (5.1.2004), s. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[172] David G Lowe. “Object recognition from local scale-invariant features”. W: *Proceedings of the seventh IEEE international conference on computer vision*. T. 2. IEEE. 1999, s. 1150–1157.

[173] Robert Maier. *”Real-time 3D Reconstruction and Localization Hauptseminar Computer Vision & Visual Tracking for Robotic Applications SS2012”*. http://www6.in.tum.de/pub/Main/TeachingSs2012SeminarComputerVisionTrackingRobotics/RobertMaier-3dRecAndLoc.pdf. Uniwersytet Techniczny w Monachium, 12.6.2012.

[174] Zoltan Csaba Marton, Radu Bogdan Rusu i Michael Beetz. “On fast surface reconstruction methods for large and noisy point clouds”. W: *2009 IEEE International Conference on Robotics and Automation(ICRA)*. Kobe, Japonia, 2009, s. 3218–3223. DOI: 10.1109/ROBOT.2009.5152628.

[175] Armin Masoumian i in. “Monocular Depth Estimation Using Deep Learning: A Review”. W: *Sensors* 22.14 (2022).

[176] Nikolaus Mayer i in. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. W: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (6/2016). DOI: 10.1109/cvpr.2016.438. URL: http://dx.doi.org/10.1109/CVPR.2016.438.

[177] Krystian Mikolajczyk i Cordelia Schmid. “A Performance Evaluation of Local Descriptors”. W: *IEEE transactions on pattern analysis and machine intelligence* 27 (11/2005), s. 1615–30. DOI: 10.1109/TPAMI.2005.188.

[178] Krystian Mikolajczyk i in. “A Comparison of Affine Region Detectors”. W: *International Journal of Computer Vision* 65 (11/2005), s. 43–72. DOI: 10.1007/s11263-005-3848-x.

[179] Mark M. Millonas. *Swarms, Phase Transitions, and Collective Intelligence*. 1993. arXiv: adap-org/9306002 [adap-org]. URL: https://arxiv.org/abs/adap-org/9306002.

[180] *Monodepth Project*. https://github.com/nianticlabs/monodepth2.

[181] Pierre Moulon i in. “OpenMVG: Open multiple view geometry”. W: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, s. 60–74.

[182] Marius Muja i David G. Lowe. “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration”. W: *International Conference on Computer Vision Theory and Applications*. 2009.

[183] Raúl Mur-Artal, J. M. M. Montiel i Juan D. Tardós. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. W: *IEEE Transactions on Robotics* 31.5 (10/2015), s. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/tro.2015.2463671. URL: http://dx.doi.org/10.1109/TRO.2015.2463671.

[184] Raúl Mur-Artal i Juan D. Tardós. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. W: *IEEE Transactions on Robotics* 33.5 (2017), s. 1255–1262. DOI: 10.1109/TRO.2017.2705103.

[185] Liz Murphy i in. “Experimental Comparison of Odometry Approaches”. W: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Wyed. Jaydev P. Desai i in. Heidelberg: Springer International Publishing, 2013, s. 877–890. ISBN: 978-3-319-00065-7. DOI: 10.1007/978-3-319-00065-7_58. URL: https://doi.org/10.1007/978-3-319-00065-7_58.

[186] Ken Museth i in. “Algorithms for Interactive Editing of Level Set Models”. W: *Computer Graphics Forum* 24.4 (2005), s. 821–841. DOI: https://doi.org/10.1111/j.1467-8659.2005.00904.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2005.00904.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2005.00904.x.

[187] Richard A. Newcombe i in. “KinectFusion: Real-time dense surface mapping and tracking”. W: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 2011, s. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[188] Jim Nilsson i Tomas Akenine-Möller. *Understanding SSIM*. 2020. arXiv: 2006.13846 [eess.IV].

[189] D. Nister. “An efficient solution to the five-point relative pose problem”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.6 (2004), s. 756–770. DOI: 10.1109/TPAMI.2004.17.

[190] Michał Nowicki i Piotr Skrzypczynski. “Experimental Verification of a Walking Robot Self-Localization System with the Kinect Sensor”. W: *Journal of Automation, Mobile Robotics & Intelligent Systems* 7 (10/2013), s. 42. DOI: 10.14313/JAMRIS_4-2013/43.

[191] Michał Nowicki, Jan Wietrzykowski i Piotr Skrzypczynski. “Simplicity or flexibility? Complementary Filter vs. EKF for orientation estimation on mobile devices”. W: 6/2015, s. 166–171. DOI: 10.1109/CYBConf.2015.7175926.

[192] Sung-Kwun Oh, Han-Jong Jang i Witold Pedrycz. “A comparative experimental study of type-1/type-2 fuzzy cascade controller based on genetic algorithms and particle swarm optimization”. W: *Expert Systems with Applications* 38.9 (2011), s. 11217–11229. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2011.02.169. URL: https://www.sciencedirect.com/science/article/pii/S0957417411003939.

[193] *ORB-SLAM Project*. https://github.com/raulmur/ORB_SLAM2.

[194] Stanley Osher i James A Sethian. “Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations”. W: *Journal of Computational Physics* 79.1 (1988), s. 12–49. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(88)90002-2. URL: https://www.sciencedirect.com/science/article/pii/0021999188900022.

[195] Adrien Bartoli Pablo Fernández Alcantarilla (Georgia Institute of Technolog) Jesus Nuevo (TrueVision Solutions AU). “Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces”. W: *Proceedings of the British Machine Vision Conference*. BMVA Press, 9/2013. DOI: 10.5244/C.27.13.

[196] Adam Paszke i in. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[197] F. Pedregosa i in. “Scikit-learn: Machine Learning in Python”. W: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830. URL: https://scikit-learn.org/stable/.

[198] Michele Pirovano. *"3 D structure from visual motion 2011 / 2012 Project Assignment Kinfu – an open source implementation of Kinect Fusion + case study: implementing a 3 D scanner with PCL"*. 2013. URL: http://www.michelepirovano.com/pdf/3d-scanning-pcl.pdf.

[199] *Point Cloud Library*. https://pointclouds.org/.

[200] *Point Cloud Library*. http://www.pointclouds.org/blog/srcs/fheredia/all.php.

[201] Marc Pollefeys. "Visual 3D Modeling from Images." W: 1/2004, s. 3.

[202] Sai Manoj Prakhya i in. "Sparse Depth Odometry: 3D keypoint based pose estimation from dense depth data". W: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, s. 4216–4223. DOI: 10.1109/ICRA.2015.7139780.

[203] Christian Richardt i in. "Coherent Spatiotemporal Filtering, Upsampling and Rendering of RGBZ Videos". W: *Computer Graphics Forum* 31.2pt1 (2012), s. 247–256.

[204] Edward Rosten i Tom Drummond. "Machine Learning for High-Speed Corner Detection". W: *Computer Vision – ECCV 2006*. Wyed. Aleš Leonardis, Horst Bischof i Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 430–443. ISBN: 978-3-540-33833-8.

[205] Ethan Rublee i in. "ORB: An efficient alternative to SIFT or SURF". W: *2011 International Conference on Computer Vision*. 2011, s. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[206] Olga Russakovsky i in. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[207] Davide Scaramuzza i Friedrich Fraundorfer. "Visual Odometry: Part I: The First 30 Years and Fundamentals". eng. W: *IEEE robotics & automation magazine* 18.4 (2011), s. 80–92. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943233.

[208] Cameron Schaeffer. *A Comparison of Keypoint Descriptors in the Context of Pedestrian Detection: Freak vs. Surf vs. Brisk*. 2012.

[209] Adam Schmidt i in. "Calibration of the Multi-camera Registration System for Visual Navigation Benchmarking". W: *International Journal of Advanced Robotic Systems* 11.6 (2014), s. 83. DOI: 10.5772/58471.

[210] Adam Schmidt i in. “Comparative Assessment of Point Feature Detectors and Descriptors in the Context of Robot Navigation”. W: t. 7. Journal of Automation, Mobile Robotics & Intelligent Systems, 2013, s. 11–20.

[211] Adam Schmidt i in. “The registration system for the evaluation of indoor visual SLAM and odometry algorithms”. W: *Journal of Automation Mobile Robotics and Intelligent Systems* 7 (1/2013), s. 46–51.

[212] Adarsh Sehgal i in. “Lidar-Monocular Visual Odometry with Genetic Algorithm for Parameter Optimization”. W: *International Symposium on Visual Computing*. 2019.

[213] Dmitry Senushkin i in. *Decoder Modulation for Indoor Depth Completion*. 2021. arXiv: 2005.08607 [cs.CV].

[214] James A. Sethian. “A fast marching level set method for monotonically advancing fronts.” W: *Proceedings of the National Academy of Sciences* 93.4 (1996), s. 1591–1595. DOI: 10.1073/pnas.93.4.1591. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.93.4.1591. URL: https://www.pnas.org/doi/abs/10.1073/pnas.93.4.1591.

[215] James A. Sethian. “Theory, algorithms, and applications of level set methods for propagating interfaces”. W: *Acta Numerica* 5 (1996), s. 309–395. DOI: 10.1017/S0962492900002671.

[216] Pawan Sinha i Edward Adelson. “Recovering 3-D Shapes from 2-D Line-Drawings”. W: *Proceedings of the IEEE International Symposium on Intelligent Robotics*. Bangalore, India, 7 – 9.1.1993, s. 51–60.

[217] P. Skrzypczyński. *Metody analizy i redukcji niepewności percepcji w systemie nawigacji robota mobilnego*. Rozprawy - Politechnika Poznańska. Wydawn. Politechniki Poznańskiej, 2007. ISBN: 9788371432620. URL: http://lrm.cie.put.poznan.pl/psrozfin.pdf.

[218] Piotr Skrzypczyński. “Mobile Robot Localization: Where We Are and What Are the Challenges?” W: *Automation 2017*. Wyed. Roman Szewczyk, Cezary Zieliński i Małgorzata Kaliczyńska. Cham: Springer International Publishing, 2017, s. 249–267. ISBN: 978-3-319-54042-9.

[219] Piotr Skrzypczyński. “Simultaneous localization and mapping: A feature-based probabilistic approach”. en. W: *International Journal of Applied Mathematics and Computer Science* 19.4 (2009), s. 575–588. ISSN: 1641-876X. DOI: 10.2478/v10006-009-0045-z. URL: https://www.amcs.uz.zgora.pl/?action=paper&paper=460.

[220] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018. eprint: arXiv:1803.09820.

[221] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2017. arXiv: 1506.01186 [cs.CV].

[222] Joan Solà. "Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach." Prac. dokt. Institut National Polytechnique de Toulouse, 2007.

[223] Henrik Stewénius, Christopher Engels i David Nistér. "Recent developments on direct relative orientation". W: *ISPRS Journal of Photogrammetry and Remote Sensing* 60.4 (2006), s. 284–294. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2006.03.005.

[224] Hauke Malte Strasdat, José M. M. Montiel i Andrew J. Davison. "Visual SLAM: Why filter?" W: *Image Vis. Comput.* 30 (2012), s. 65–77. URL: https://api.semanticscholar.org/CorpusID:924291.

[225] Jrgen Sturm i in. "A Benchmark for the Evaluation of RGB-D SLAM Systems". W: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. 10/2012, s. 573–580. ISBN: 978-1-4673-1737-5. DOI: 10.1109/IROS.2012.6385773.

[226] Maciej Szaleniec i Ryszard Tadeusiewicz. *LEKSYKON SIECI NEURONOWYCH [Lexicon on Neural Networks]*. 15.2.2015. ISBN: 978-83-63270-10-0.

[227] Kaitao Tang i in. "An Improved ORB-SLAM2 With Refined Depth Estimation". W: *IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2019, s. 885–889.

[228] K. Tateno i in. "CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction". W: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, 2017, s. 6565–6574.

[229] Alexandru Telea. "An Image Inpainting Technique Based on the Fast Marching Method". W: *Journal of Graphics Tools* 9.1 (2004), s. 23–34. DOI: 10.1080/10867651.2004.10487596. eprint: https://doi.org/10.1080/10867651.2004.10487596. URL: https://doi.org/10.1080/10867651.2004.10487596.

[230] Bill Triggs i in. “Bundle Adjustment — A Modern Synthesis”. W: *Vision Algorithms: Theory and Practice*. Wyed. Bill Triggs, Andrew Zisserman i Richard Szeliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, s. 298–372. ISBN: 978-3-540-44480-0.

[231] S. Umeyama. “Least-squares estimation of transformation parameters between two point patterns”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (1991), s. 376–380. DOI: 10.1109/34.88573.

[232] Jiachen Wang i Qingjiu Huang. “Depth Map Super-Resolution Reconstruction Based on Multi-Channel Progressive Attention Fusion Network”. W: *Applied Sciences* 13.14 (2023).

[233] Jiayi Wang i Yasutaka Fujimoto. “High Accuracy Real-Time 6D SLAM with Feature Extraction Using a Neural Network”. W: *IEEJ Journal of Industry Applications* 10.5 (2021), s. 512–519.

[234] Sen Wang i in. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. W: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE Press, 2017, s. 2043–2050. DOI: 10.1109/ICRA.2017.7989236. URL: https://doi.org/10.1109/ICRA.2017.7989236.

[235] Sen Wang i in. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. W: *IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, 2017, s. 2043–2050.

[236] Zhou Wang i in. “Image quality assessment: from error visibility to structural similarity”. W: *IEEE Transactions on Image Processing* 13.4 (13.4.2004), s. 600–612. DOI: 10.1109/TIP.2003.819861.

[237] Michael Warren i in. “Unaided stereo vision based pose estimation”. W: *Proceedings of the 2010 Australasian Conference on Robotics and Automation*. Wyed. G Wyeth i B Upcroft. Australia: Australian Robotics & Automation Association, 2010, s. 1–8. URL: https://eprints.qut.edu.au/39881/.

[238] Thomas Whelan i in. “Kintinuous: Spatially Extended KinectFusion”. W: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Research presented in this paper was funded by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the Irish National Development Plan and the Embark Initiative of the Irish Research Co-

uncil for Science, Engineering and Technology. 19.7.2012, s. 1–8. URL: https://dspace.mit.edu/handle/1721.1/71756.

[239] Thomas Whelan i in. “Robust real-time visual odometry for dense RGB-D mapping”. W: *2013 IEEE International Conference on Robotics and Automation*. 2013, s. 5724–5731. DOI: 10.1109/ICRA.2013.6631400.

[240] Thiemo Wiedemeyer. *IAI Kinect2*. https://github.com/code-iai/iai_kinect2. Dostęp 12.10, 2021. University Bremen: Institute for Artificial Intelligence, 2014-2015.

[241] Cho-Ying Wu i in. “Toward Practical Monocular Indoor Depth Estimation”. W: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), s. 3804–3814.

[242] Chuhua Xian i in. “Fast Generation of High-Fidelity RGB-D Images by Deep Learning With Adaptive Convolution”. W: *IEEE Transactions on Automation Science and Engineering* 18.3 (2021), s. 1328–1340.

[243] Z. Xinchao. “A perturbed particle swarm algorithm for numerical optimization”. W: *Applied Soft Computing* 10 (1 2010), s. 119–124.

[244] Xiaohan Yang i in. “Overfitting reduction of pose estimation for deep learning visual odometry”. W: *China Communications* 17.6 (2020), s. 196–210. DOI: 10.23919/JCC.2020.06.016.

[245] J. Yu i in. “Generative Image Inpainting with Contextual Attention”. W: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, s. 5505–5514.

[246] Y. Zhang i T. Funkhouser. “Deep Depth Completion of a Single RGB-D Image”. W: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, 2018, s. 175–185.

[247] Liqian Zhou i in. “Adaptive SLAM Methodology Based on Simulated Annealing Particle Swarm Optimization for AUV Navigation”. W: *Electronics* 12.11 (2023).

[248] Marco Zuliani. “RANSAC for Dummies With examples using the RANSAC toolbox for Matlab™ & Octave and more...” W: 2014.