

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Jurica Maltar

ROBUST VISUAL PLACE RECOGNITION USING DEEP REPRESENTATIONS AND SEQUENCE-BASED IMAGE MATCHING

DOCTORAL THESIS

Zagreb, 2023



University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Jurica Maltar

ROBUST VISUAL PLACE RECOGNITION USING DEEP REPRESENTATIONS AND SEQUENCE-BASED IMAGE MATCHING

DOCTORAL THESIS

Supervisors:

Professor Ivan Marković, PhD

Associate Professor Domagoj Matijević, PhD

Zagreb, 2023



Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Jurica Maltar

ROBUSNO VIZUALNO PREPOZNAVANJE MJESTA UPORABOM DUBOKIH REPREZENTACIJA I UPARIVANJA SLJEDOVA SLIKA

DOKTORSKI RAD

Mentori:

prof. dr. sc. Ivan Marković

izv. prof. dr. sc. Domagoj Matijević

Zagreb, 2023.

Doctoral thesis was written at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering.

Supervisors: Professor Ivan Marković, PhD and Associate Professor Domagoj Matijević, PhD

Thesis contains 111 pages

Thesis no.: _____

ABOUT THE SUPERVISORS

Ivan Marković was born in Osijek, Croatia in 1985. He finished a math-oriented high school in Osijek in 2003 and received the master and PhD degrees in Electrical Engineering under the mentorship of Prof. Ivan Petrović from the University of Zagreb Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia in 2008 and 2014, respectively. In 2008 he was employed by S.C.A.N. Ltd. where he worked on developing control algorithms and supervision interfaces for terminal automation systems. In 2009 he joined FER and the Department of Control and Computer Engineering as a research assistant where he enrolled the PhD studies in the same year. In 2009 and 2012 he was a member of the National Organizing Committee and in 2022 the Programe Committee Chair of international scientific conferences. In 2009 he became the coordinator of the technical editing team of the journal "Automatika - Journal for Control, Measurement, Electronics, Computing and Communications" and since 2018 he serves as the deputy Editor-In-Chief. In collaboration with Siemens Croatia Inc. he has been a course trainer for programmes developed for industry participants and in 2018. he recieved the recognition for his contribution in advancing education for SIMATIC automation systems. He actively participated as a researcher or work package leader in 6 national and 7 international research projects. During 2013 and 2014 he was a visiting researcher in the laboratory led by Prof. Francois Chaumette at INRIA Rennes-Bretagne Atlantique in Rennes, France. He published 23 journal papers and more than 40 conference papers in the fields of estimation theory, sensor fusion, and robot vision with applications to autonomous mobile robot and vehicle control. He is a member of the Institute of Electrical and Electronics Engineers (IEEE), Technical Committee for Robotics of the International Federation of Automatic Control (IFAC), and the Croatian Society for Communication, Computing, Electronics, Measurement and Control (KoREMA). In 2018 he received the young scientist award "Vera Johanides" of the Croatian Academy of Engineering, in 2014 silver plaque "Josip Lončar" of FER for an outstanding doctoral dissertation, in 2008 the "INETEC award" of the Institute of Nuclear Technology for academic excellence during the undergraduate and graduate studies, and in 2006 the "Josip Lončar" award of FER for student excellence in the respective academic year.

Domagoj Matijević graduated at the Department of Mathematics, University of Osijek, Croatia in 2001. He received his M.Sc. degree in 2003 and PhD in 2007 from University of Saarland, Germany. From 2007-2008 he worked as a Postdoc, from 2008-2012 as an Assistant professor and since 2012 he works as an Associate professor at Department of Mathematics in Osijek. From 2014 he is heading Computer Science Research Group at the Department of Mathematics, and since 2017 he is Deputy Head of Department for Research. His research interests are in the area of Linear and Combinatorial Optimization, Approximation Algorithms and Bioinformatics. He supervised three doctoral thesis in 2013, 2015 and 2021. He is currently mentoring work of four PhD students.

O MENTORIMA

Ivan Marković je rođen 1985. godine u Osijeku. Treću gimnaziju u Osijeku završio je 2003. godine te je diplomirao i doktorirao u polju elektrotehnike pod mentorstvom prof. dr. sc. Ivana Petrovića na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 2008. odnosno 2014. godine. Po završetku studija zapošljava se u tvrtki S.C.A.N. d.o.o. gdje radi na razvijanju algoritama upravljanja i nadzora u sustavima automatizacije naftnih terminala. U 2009. godini zapošljava se u Zavodu za automatiku i računalno inženjerstvo FER-a u zvanju znanstvenog novaka u suradničkom zvanju asistenta te iste godine upisuje i doktorske studije. U 2009. i 2012. godini bio je član organizacijskog odbora, a u 2022. predsjednik programskog odbora međunarodnih znanstvenih konferencija. Od 2009. godine koordinator je tehničkog uredništva časopisa "Automatika - časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije", a od 2018. je zamjenik glavnog i odgovornog urednika. U suradnji s tvrtkom Siemens Hrvatska d.d. bio predavač je na tečajevima za polaznike iz industrije te je 2018. godine primio priznanje za poseban doprinos unaprijeđenu nastave i edukaciji korištenjem SIMATIC automatizacijskih sustava. Aktivno je sudjelovao kao suradnik ili voditelj radnog paketa na 6 nacionalnih i 7 međunarodnih znanstvenih projekata. Tijekom 2013. i 2014. godine bio je gostujući istraživač u grupi prof. dr. sc. Francoisa Chaumettea na ustanovi INRIA Rennes-Bretagne Atlantique u Rennesu, Francuska. Objavio je 23 znanstvena rada u časopisima i više od 40 znanstvenih radova u zbornicima skupova u području estimacije, fuzije senzora i robotskog vida s primjenom u upravljanju autonomnim mobilnim robotima i vozilima. Član je društva Institute of Electrical and Electronics Engineers (IEEE), tehničkog odbora za robotiku društva International Federation of Automatic Control (IFAC) i Hrvatskog društva za komunikacije, računarstvo, elektroniku, mjerenja i automatiku (KoREMA). Godine 2018. primio je nagradu mladom znanstveniku "Vera Johanides" Akademije tehničkih znanosti Hrvatske, 2014. srebrnu plaketu "Josip Lončar" FER-a za posebno istaknutu doktorsku disertaciju, 2008. "INETEC nagradu" Instituta za nuklearnu tehnologiju za izvrsnost postignutu tijekom studija, a 2006. godine nagradu "Josip Lončar" FER-a za izvanredan uspjeh u toj akademskoj godini.

Domagoj Matijević diplomirao je 2001. godine na Odjelu za matematiku Sveučilišta u Osijeku. Nakon toga odlazi na diplomski studij računalnih znanosti na Sveučilištu Saarland, Njemačka. Nakon uspješno završenog studija, pristupa doktorskom programu Max-Planck Instituta za računalnu znanost. 2007. godine doktorira na Sveučilištu Saarland i zapošljava se kao poslijedoktorand na Odjelu za matematiku u Osijeku. Od 2008. do 2012. radio je kao docent, a od 2012. do danas kao izvanredni profesor. Od 2014. godine vodi Katedru za računarstvo na Odjelu za matematiku, a od 2017. godine obnaša dužnost zamjenika pročelnika za znanstveno-istraživačku djelatnost. Znanstveno istraživački interes nalazi mu se u području linearne i kombinatorne optimizacije, aproksimacijskih algoritama i bioinformatike. Mentorirao je tri doktorske disertacije u 2013., 2015. i 2021. godini. Trenutno mentorira ili sumentorira rad četiri asistenta na Odjelu za matematiku.

ZAHVALA

Zahvaljujem se prof. dr. sc. Ivanu Markoviću koji je pristao biti moj mentor, uputio me u ovu zanimljivu tematiku te uvijek bio dostupan i spreman pomoći. Zahvaljujući njemu, moje je znanje dovedeno na jednu višu razinu. Nadam se kako će se to znanje i dalje nadograđivati te kako je kraj ove epohe mog obrazovanja tek početak naše suradnje.

Zahvaljujem se izv. prof. dr. sc. Domagoju Matijeviću koji me je uputio u računarstvo, naučio što je to algoritam, što je to uvjetno grananje, što je to petlja… Nevjerojatno, ali istinito, on me je podučio osnovama programiranja, objektno orijentiranom programiranju, strukturama podataka, web programiranju, a ktome i osnovama robotike. Uz njegove izvanredne sposobnosti podučavanja, spomenuo bih i njegovu vedrinu koja i najtmurniji dan na poslu učini veselijim.

Zahvalio bih se i akademiku prof. dr. sc. Ivanu Petroviću koji je svojim iskustvom i znanjem uvelike pridonio pisanju naših znanstvenih radova bez kojih ne bi bilo ove disertacije. Hvala i dr. sc. Josipu Ćesiću koji me je predstavio profesorima Petroviću i Markoviću. Tome je prethodio naš sastanak na kojem me je on ohrabrio kako je moguće baviti se suvremenom robotikom bez boravka u robotičkom laboratoriju. Zahvaljujem se i svim svojim profesorima i kolegama s Odjela za matematiku (budućeg Fakulteta primijenjene matematike i informatike) Sveučilišta J. J. Strossmayera u Osijeku te svim svojim profesorima i kolegama s Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Hvala mojim roditeljima Vjekoslavu i Meliti na neizmjernoj roditeljskoj ljubavi i bespogovornoj podršci čime god se u životu bavio. Hvala mome djedu Đuri koji i dandanas sam izrađuje električne strojeve čime je uvelike zaslužan za moju ljubav prema robotici. Hvala mojoj braći Bruni i Pjeri te sestri Lani, svim ostalim članovima moje obitelji te mojim prijateljima uz koje je lijepo ponekad odahnuti od obaveza.

Naposljetku, hvala mojoj djevojci Marti koja je već deset godina sa mnom u svim životnim situacijama i koja je bila moj glavni oslonac u pisanju ove disertacije.

U Valpovu, 5. veljače 2023. godine.

ABSTRACT

Autonomous vehicles have become commonplace in transportation. Due to material imperfections, noisy sensors, and disobedient actuators, autonomous vehicles are prone to accumulate drift. Luckily, drift can be eradicated if, during *simultaneous localization and mapping*, the vehicle reappears in an already visited place and recognizes it. This act has been dubbed as *loop closing detection* because reappearing in an already visited place means traversing a loop-like trajectory that starts and ends in that place. In *visual place recognition*, we aim to recognize such already visited places given their images. Ordinary phenomena of everyday traffic, e.g., different viewpoints of the vehicle, various moving objects, changes of seasons, different times of the day, precipitations, etc., are, surprisingly, negative factors in visual place recognition. To assure robustness against these negative factors, we should design our visual place recognition as viewpoint-invariant and condition-invariant.

Therefore, when designing a visual place recognition system, we should ask: how to match a place and how to represent it? Exploiting the fact that images of places are captured in a sequential manner, we can match a place not just according to its image but also according to neighboring images. This is the idea *sequence-based place matching* methods are based on, and in this thesis, we present one such method – NOSeqSLAM. Having a capable place matching method, we should focus on how to robustly represent images that are going to be matched with this method. Computer vision helps us in this objective – we create robust image representations by using either handcrafted or learned image models. Naturally, as they adapt to data distributions, learned image models have better performance in quantitative results. Accordingly, we propose how to improve image representations obtained with such image models in the context of visual place recognition by using two techniques: *softmax regression* and *mutual information-based feature selection*. Our proposed sequence-based method, in conjunction with the proposed image representations, in terms of place recognition quantitative evaluation, outperforms competitive sequence-based methods and competitive image models. Also, in order for our system not to be an end in itself, we have adapted and used it with two different SLAM implementations, a simulated range sensor-based SLAM and a visual SLAM evaluated on a real-world dataset, that were created specifically for its justification.

KEYWORDS: autonomous vehicle, traffic, drift, visual place recognition, machine learning, image models, convolutional neural networks, factor graphs, simultaneous localization and mapping, loop closing

SAŽETAK

ROBUSNO VIZUALNO PREPOZNAVANJE MJESTA UPORABOM DUBOKIH REPREZENTACIJA I UPARIVANJA SLJEDOVA SLIKA

Donedavno znanstvena fantastika, a danas stvarnost – svjedoci smo sve većeg broja autonomnih vozila. Bilo bi sjajno odvesti se na posao čitajući knjigu i pritom ne razmišljati o prometu. I još bitnije, bilo bi sjajno da autonomna vozila jednoga dana smanje broj prometnih nesreća te gužve. Vozila, kojima je nekoć upravljao isključivo čovjek, danas sadrže mnoštvo senzora kako bi iskusili okolinu na način svojstven čovjeku. Golem broj poluvodiča i linija koda pokušavaju interpretirati tako prikupljene podatke te poduzeti radnje, također, na način svojstven čovjeku. Usprkos tome što čovjek lako savladava vožnju, mnogo je teže taj cilj postići kod autonomnih vozila. Sam po sebi, čovjek razumije prostorno-vremenski kontekst u prometu. Mogući je pristup pripojenja tog konteksta autonomnom vozilu – pridružiti vozilu njegovu poziciju i orijentaciju kako bi ono znalo gdje se nalazi i kamo treba krenuti. Također, ne bi niti imalo smisla govoriti o poziciji i orijentaciji kada ne bismo imali referentni prostor u odnosu na koji bismo te veličine mogli izraziti, stoga je nužno voditi računa i o tom prostoru, odnosno, okolini vozila.

Kada bi to bilo moguće, izračunali bismo poziciju i orijentaciju vozila na osnovu odometrije. Nažalost, zbog nesavršenosti materijala, nepreciznih senzora i neposlušnih aktuatora, zbog činjenice da su računala diskretni sustavi dok je stvarni svijet kontinuiran, vozilo, gibajući se kroz okolinu, akumulira pogrešku izazvanu zanošenjem (eng. drift). Koliko god kvalitetni senzori, aktuatori, materijali, elektroničke komponente i matematički modeli, pogreška izazvana zanošenjem neće iščeznuti. Djelotvoran je pristup umanjivanju te pogreške izgradnja karte prostora u kojoj se vozilo kreće, odnosno kartiranje prostora, i istovremeno, procjena pozicije i orijentacije vozila u odnosu na prostorna obilježja, odnosno, lokalizacija vozila. Radi se o poznatom i opširno istraživanom robotičkom problemu zvanom *istovremena lokalizacija i kartiranje* (eng. simultaneous localization and mapping, abbr. SLAM), a on se ugrubo može razdvojiti na dva manja, ali i sama po sebi istraživačka problema: kartiranje te lokalizacija. U problemu kartiranja, nastojimo predstaviti stvarni svijet s izgrađenom kartom, dok u problemu lokalizacije, s obzirom na prostorna obilježja izgrađene karte, nastojimo procijeniti gdje se vozilo nalazi. Prema [1, str. 282], SLAM se smatra tzv. "kokoš i jaje" problemom – kako bismo izgradili preciznu kartu okoline, vozilo mora biti precizno lokalizirano, i obrnuto, lokalizacija je precizna samo ako postoji precizna karta prostora. Budući da je robotika međudisciplinarna grana više znanosti, istovremena se lokalizacija i kartiranje, kao opširan robotički problem, može poboljšati na razne načine iz različitih stajališta.

Uspostavlja se da je moguće umanjiti pogrešku izazvanu zanošenjem ukoliko se vozilo pojavi na otprije posjećenom mjestu. Takva nam situacija odgovara jer težimo ka smanjenoj pogrešci, što povlači i točnije kartiranje i lokalizaciju. Pojaviti se na otprije posjećenom mjestu znači obići kružnu trajektoriju s početkom i krajem u istome mjestu. Stoga se, u kontekstu SLAM-a, taj problem naziva *detekcija zatvaranja petlje* (eng. loop closing detection, abbr. LCD) ili, još jednostavnije, *prepoznavanje mjesta* (eng. place recognition). Prepoznavanje je mjesta, također, istraživački problem sam po sebi sa svojstvenim kvantitativnim mjerama. Ukoliko je senzorski modalitet SLAM sustava vizualan, npr., mono ili stereo kamera, govorimo o *vizualnom prepoznavanju mjesta* (eng. visual place recognition, abbr. VPR) u kojem obrađujemo slike mjesta koje je vozilo snimilo. U "offline" vizualnom prepoznavanju mjesta, gdje su višestruki obilasci iste rute snimljeni unaprijed, razlikujemo dva tipa slika mjesta – skup slika za upite $\mathcal{Q}$ i referentni skup slika $\mathcal{D}$. Za najnovije snimljenu sliku iz baze upita, pretražujemo referentni skup kako bismo pronašli uparivanje. Metodologiju je "offline" vizualnog prepoznavanja mjesta moguće prenijeti na "online" vizualno prepoznavanje mjesta gdje pokušavamo prepoznati mjesto samo iz jednog skupa slika. Također, metodologiju je vizualnog prepoznavanja mjesta moguće prenijeti na druge oblike prepoznavanja mjesta, npr., prepoznavanje mjesta s obzirom na snimke LiDAR senzora.

Ako više slika, ne previše nalik jedna drugoj, prikazuju isto mjesto, čovjeku nije problem prepoznati to mjesto. U stvarnome je svijetu gotovo nemoguće imati više vizualno identičnih slika koje prikazuju isto mjesto, čime bi to mjesto bilo trivijalno prepoznatljivo sustavom za vizualno prepoznavanje mjesta. Dvije su kategorije čimbenika koji prouzrokuju vizualnu različitost slike koja sadrži mjesto, a to su:

- različiti okolišni uvjeti i
- različita gledišta kamere.

Vizualno, mjesto je uvelike određeno statičnim objektima koji se u njemu nalaze – zgrade, drveće, rasvjeta, itd. Ipak, moguće je, poglavito u svakodnevnom prometu, da se u sceni nađu gibajući objekti koji zaklanjaju scenu, odnosno, ono što ju karakterizira. Jasno je kako takvi objekti ne pridonose prepoznatljivosti mjesta. Različita doba dana također drastično mijenjaju vizualni izgled mjesta, ne samo zbog svijetlog ili tamnog neba, već i zbog različitih osvjetljenja i statičnih i gibajućih objekata. U jednakoj su mjeri kriva i različita godišnja doba i vremenski uvjeti. Ekstremni bi primjer za to, koji se čak i pojavljuje u jednome skupu podataka kojeg koristimo, bio snijeg. Čak i kada okolišni uvjeti ne odstupaju u velikoj mjeri, npr., pri online vizualnom prepoznavanju mjesta u svrhu zatvaranja petlji u istovremenoj lokalizaciji i kartiranju gdje je očekivano da se unutar nekoliko minuta uvjeti neće značajno promijeniti, očekivano je kako će doći do odstupanja u gledištima kamere. Htjeli bismo umanjiti utjecaj neprijatnih čimbenika ove dvije kategorije, stoga težimo da naš sustav vizualnog prepoznavanja mjesta bude *invarijantan na gledište* i *invarijantan na okolišne uvjete*. To polučuje točnije prepoznavanje mjesta, odnosno, točnije zatvaranje petlji, što pak polučuje točniju procjenu pozicije i orijentacije te precizniju kartu prostora. Komplementarno je ovim dvojim čimbenicima *percepcijsko preklapanje* (eng. perceptual aliasing). Ono se odnosi na pojavu više geografski različitih mjesta koja su vizualno nalik

jedno drugome. Percepcijsko nas preklapanje dovodi u opasnost netočnog zatvaranja petlje što negativno utječe na kartiranje i lokalizaciju.

Kako bismo postigli invarijantnost okolišnih uvjeta i gledišta te neranjivost na percepcijsko preklapanje, pri razvoju se sustava za vizualno prepoznavanje mjesta trebamo osvrnuti na dva aspekta dizajna takvog sustava:

- slikovnu reprezentaciju i
- uparivanje mjesta.

Iako crno-bijele ili slike u boji mogu biti korištene kao slikovne reprezentacije u vizualanom prepoznavanju mjesta, kao što je to slučaj u [2, 3], takve "jednostavne" reprezentacije jednostavno ne polučuju dobre rezultate. Naime, brojevne se vrijednosti u takvim reprezentacijama mijenjaju drastično, čak i pri najmanjim pomacima gledišta kamere i pri malim promjenama okolišnih uvjeta, čime je nemoguće međusobno uspoređivati slike. Bolje bi bilo pouzdati se u postojeće tehnike računalnog vida. U ne tako davnoj prošlosti, tehnike lokalnih značajki *ručne izrade* (eng. handcrafted) koristile su se kako bismo pronašli znamenitija područja u slici. Uz lokalne značajke dolaze i lokalni opisnici koji nekom znamenitom području pridužuju prepoznatljivu vektorsku vrijednost. Lokalne opisnike zatim *agregiramo* u globalne opisnike slike. U novije se vrijeme ipak, kao slikovna reprezentacija, koriste mape značajki izlučene iz dubokih modela slika – konvolucijskih neuronskih mreža. Kako bismo uparili mjesta preko njihovih slika, jednom kada je odgovarajući model slika odabran, možemo se poslužiti s činjenicom kako su slike, jer ih je snimilo vozilo, poredane u slijed. Stoga se, pri uparivanju, nekoliko susjednih slika može uzeti u obzir. Na toj ideji počivaju takozvane metode vizualnog prepoznavanja mjesta zasnovane na sljedovima slika koje proučavamo ovdje.

Ova disertacija predstavlja tri izvorna znanstvena doprinosa za problem vizualnog prepoznavanja mjesta. Prema karakterizaciji problema, u mogućnosti smo pridonijeti njegovom rješavanju time što ćemo poboljšati uparivanje mjesta i njegovu slikovnu reprezentaciju. Stoga se prvi i drugi izvorni znanstveni doprinosi odnose na takva poboljšanja. Kako vizualno prepoznavanje mjesta ne bi bio problem sam za sebe, u trećem ćemo ga izvornom znanstvenom doprinosu koristiti pri istovremenoj lokalizaciji i kartiranju. Izvorni su znastveni doprinosi:

#1 Metoda vizualnog uparivanja mjesta zasnovana na sljedovima slika koja koristi usmjerene acikličke grafove i najkraći put iz jednog izvora.

Predložena je metoda vizualnog uparivanja mjesta, NOSeqSLAM [4, 5], poopćenje SeqSLAM-a [2], postojeće metode vizualnog uparivanja mjesta zasnovane na sljedovima slika. Time što je zasnovana na sljedovima slika, u mogućnosti je upariti mjesto, ne samo na osnovu slike tog mjesta, već i na osnovu susjednih slika. Time što je poopćenje, NOSeqSLAM nadmašuje SeqSLAM u kvantitativnoj evaluaciji. Poopćenje je postignuto uporabom teorije grafova gdje prepoznavanje mjesta modeliramo usmjerenim acikličkim grafovima. Oni imaju posebnu topološku strukturu, pa uz predloženu metodu, predlažemo i po mjeri skrojen algoritam za pronalaženje najkraćeg puta iz jednog izvora za topologiju takvih grafova.

#2 Metoda za robusno vizualno prepoznavanje mjesta dubokim reprezentacijama koja koristi softmax regresiju i odabir značajki zasnovan na uzajamnom sadržaju informacije.

U ovom smo izvornom znanstvenom doprinosu pokazali kako prilagoditi softmax regresiju u kontekstu vizualanog prepoznavanja mjesta time što smo vizualno prepoznavanje mjesta sveli na problem višeklasne klasifikacije slika. Zatim smo provjerene duboke modele slika optimizirali s predloženim pristupom te su oni, u sinergiji s predloženom metodom prepoznavanja mjesta, postigli značajna poboljšanja u vidu kvantitativnih rezultata. Na sličan smo način prilagodili i provjerenu metodu za odabir značajki zasnovanu na uzajamnom sadržaju informacije te su tako poboljšani modeli, u sinergiji s predloženom metodom prepoznavanja mjesta, još jednom postigli značajna poboljšanja.

#3 Postupak prilagodbe metode vizualnog prepoznavanja mjesta zasnovane na sljedovima slika za zatvaranje petlje u algoritmima istovremene lokalizacije i kartiranja.

Korištenjem programske paradigme vektorizacije, metoda je vizualnog uparivanja mjesta postala još brža te ju je moguće izvršavati na grafičkoj kartici. Vektorizirana je instanca predložene metode u mogućnosti detektirati zatvaranja petlje u stvarnome vremenu. Ona se pokazala djelotvornom u čak dva različita sustava istovremene lokalizacije i mapiranja. Jedan od njih koristi senzor udaljenosti, što upućuje da predložena metoda nadilazi obim vizualnog prepoznavanja mjesta i primjeniva je za prepoznavanje mjesta općenito. Drugi je sustav vizualan i evaluiran je na skupu podataka koji su nastali u stvarnom trodimenzionalnom okruženju.

Svako od nadolazećih poglavlja ove disertacije odnosi se na jedan izvorni znanstveni doprinos. Dodatno je dodano i zaključno poglavlje. U prvome poglavlju čitatelja uvodimo u problem vizualnog prepoznavanja mjesta, dok drugo poglavlje opisuje taj problem u širem obimu gdje započinjemo s općenitom formulacijom. Uparivanje sljedova slika započinjemo s Bayesovom formulacijom, a zatim se osvrćemo na SeqSLAM, jer ta je metoda prepoznavanja mjesta uvelike utjecala na stvaranje predložene metode prepoznavanja mjesta. Zatim pokazujemo na koji je to način NOSeqSLAM stvoren kao poopćenje SeqSLAM-a. Zatim se osvrćemo na algoritme na grafovima koji su potrebni u našoj metodi i predstavljamo algoritam koji efikasno pronalazi najkraći put iz jednog izvora, za topologiju usmjerenih acikličkih grafova u NOSeqSLAM-u. Zaključujemo poglavlje s asimptotskom i empirijskom analizom vremena izvršavanja SeqSLAM-a i NOSeqSLAM-a.

U početku trećeg poglavlja iznosimo opširan pregled uobičajenih tehnika za modeliranje slika unutar računalnog vida. Govorimo o tome što su to znamenita područja slike i na koji ih način detektirati. U suštini, pregledavamo lokalne značajke i opisnike ručne izrade zasnovane na gradijentima – SIFT [6] and SURF [7]. Korištenjem tehnike nenadziranog strojnog učenja, k-means grupiranja, pronalazimo *vizualne riječi*, odnosno, centre grupa dobivene grupiranjem iz skupa lokalnih opisnika. Zatim agregiramo lokalne opisnike slike, ili s *bag-of-words* tehnikom [8] ili s vektorima lokalno agregiranih opisnika [9], u globalne opisnike slike. Dodatno, pregledavamo još jednu tehniku zasnovanu na gradijentima koja stvara globalni opisnik slike – histogram orijentiranih gradijenata. Razlog zbog kojeg proučavamo

pristupe računalnog vida ručne izrade jest što su oni računski nezahtjevni. Zatim iznosimo pregled učenih modela slika, konvolucijskih neuronskih mreža, te njihova svojstva. Iako su učeni modeli računski nešto zahtjevniji, zbog učenja se na podacima oni prilagođavaju distribucijama iz kojih podaci potječu. Imajući sustavan pregled učenih modela, predstavljamo kako vizualno prepoznavanje mjesta svesti na višeklasnu klasifikaciju kako bismo duboki model dodatno prilagodili korištenjem softmax regresije. Također, predstavljamo kako na model primijeniti odabir značajki zasnovan na uzajamnom sadržaju informacije i tako poboljšan model puštamo u rad unutar našeg sustava. Kvantitativna evaluacija pokazuje kako modeli slika poboljšani softmax regresijom i odabirom značajki zasnovanom na uzajamnom sadržaju informacije u sinergiji s predloženom metodom NOSeqSLAM nadmašuju ostale metode vizualnog uparivanja mjesta i ostale modele slika.

Četvrto se poglavlje odnosi na postupak prilagodbe predložene metode vizualnog prepoznavanja mjesta u algoritmima istovremene lokalizacije i kartiranja. Prvo se osvrćemo na *faktor grafove* kojima modeliramo stohastičke postupke. Budući da se radi o stohastičkom postupku, iznosimo kako modelirati istovremenu lokalizaciju i kartiranje korištenjem faktor grafova. Nakon toga, odjeljak posvećujemo vektorizaciji predložene metode prepoznavanja mjesta kako bi se ona mogla izvršavati u stvarnome vremenu s dodatnom mogućnosti izvršavanja na grafičkoj kartici. Zatim razvijamo SLAM № 1, simulirano okruženje za istovremenu lokalizaciju i kartiranje gdje je korišten senzor udaljenosti i odometrija zasnovana na *iterativnom algoritmu najbliže točke* (eng. iterative closest point, abbr. ICP) kako bismo olakšali postupak prilagodbe i iskušali NOSeqSLAM u kontekstu SLAM-a. Napokon, razvijamo i drugo okruženje, vizualni SLAM № 2. Ponovno iskušavamo vektorizirani NOSeqSLAM, ali ovog puta na skupu podataka koji je nastao u stvarnom okruženju. Kvalitativna evaluacija u oba okruženja ukazuje na to kako NOSeqSLAM uspješno detektira zatvaranja petlje. Drugim riječima, uz isključenu detekciju zatvaranja petlje, SLAM № 1 i SLAM № 2 akumuliraju veću pogrešku izazvanu zanošenjem.

U zaključnom se poglavlju osvrćemo na sve doprinose koji su postignuti. Na kraju, iznosimo nekoliko riječi o budućem istraživanju.

KLJUČNE RIJEČI: autonomno vozilo, promet, greška uzrokovana zanošenjem, vizualno prepoznavanje mjesta, strojno učenje, modeli slika, konvolucijske neuronske mreže, faktor grafovi, istovremena lokalizacija i kartiranje, zatvaranje petlje

CONTENTS

1 INTRODUCTION 1
1.1 Motivation and problem statement 1
1.2 Original scientific contributions 3
1.3 Outline of the thesis 4

2 SEQUENCE-BASED VISUAL PLACE RECOGNITION 6
2.1 Sequence-based visual place recognition 8
2.1.1 FAB-MAP 8
2.1.2 SeqSLAM 9
2.1.3 Cone-based SeqSLAM 10
2.1.4 Other variants of SeqSLAM 11
2.2 NOSeqSLAM 11
2.3 Tailored single source shortest path algorithm 13
2.4 Summary 17

3 ON IMAGE MODELS IN THE CONTEXT OF VISUAL PLACE RECOGNITION 19
3.1 Handcrafted image models 19
3.1.1 Local features and descriptors 20
3.1.2 Global descriptors 26
3.2 Learned image models 30
3.2.1 Neural networks 30
3.2.2 Optimization of neural networks 33
3.2.3 Convolutional neural networks 35
3.2.4 Adversities in neural networks 40
3.3 Enhancing visual place recognition on sequential data with softmax regression 42
3.3.1 Softmax regression 43
3.3.2 Feature maps extraction 45
3.3.3 Experimental results 45
3.3.4 Adversarial training and adversarial examples 50
3.4 Mutual information-based feature selection for visual place recognition 52
3.4.1 Feature selection for visual place recognition 58
3.4.2 Experimental results 59
3.5 Summary 60

4 CLOSING THE LOOP IN SIMULTANEOUS LOCALIZATION AND MAPPING 64
4.1 Factor graphs 64
4.2 Simultaneous localization and mapping 66
4.2.1 Odometry 67
4.2.2 Loop closing 69
4.2.3 Mapping 70
4.3 Adapting NOSeqSLAM for loop closing in SLAM 71
4.3.1 Problem-specific adaptation 71
4.3.2 Optimization of the execution time 73
4.4 LiDAR-based SLAM in a 2D simulated environment 77
4.4.1 SLAM № 1 79
4.4.2 Experimental results 85
4.5 Visual SLAM in a 3D real-life environment 87
4.5.1 SLAM № 2 90
4.5.2 Experimental results 92
4.6 Summary 93

5 CONCLUSION AND OUTLOOK 98

BIBLIOGRAPHY 101

CURRICULUM VITAE 112

FULL LIST OF PUBLICATIONS 113

ŽIVOTOPIS 114

1

Introduction

1.1 Motivation and problem statement

What just yesterday looked like a science fiction story, today is a reality – we are witnessing a tremendously increasing number of autonomous vehicles. Would not it be great to read a book on our way to work without worrying about traffic? And most importantly, autonomous vehicles, hopefully, one day may reduce the number of car accidents and traffic jams. Vehicles, once exclusively human-controlled, today possess a multitude of sensors in order to experience the world the way a human does. An unimaginably vast number of transistors and lines of code try to interpret such obtained information and take actions, too, the way a human does. Nevertheless, it is easy for a human to learn to drive. It is much harder for an autonomous vehicle to "understand" the way in which traffic moves on. A human inherently understands the spatio-temporal traffic context. A way to incorporate this context into a vehicle's onboard computer is to assign to the vehicle its pose in order to know where it is and where it should go. It would be pointless to mention the pose unless there is a referent space with respect to which the pose is expressed; therefore, we also need to keep track of that space – the vehicle's environment.

If it were possible, odometry itself would give us the precise pose of the vehicle. Unfortunately, due to imperfections in materials, noisy sensors, and "disobedient" actuators, due to the fact that computers are discrete systems while the real world is continuous, the vehicle, moving through an environment, accumulates the drift. Notwithstanding how good sensors, actuators, materials, electronics, and mathematical models are, the drift will remain. An effective approach to get rid of this drift is to simultaneously build a map of an environment in which the vehicle is moving through and to estimate the vehicle's pose relative to the landmarks in that map, i.e., to localize the vehicle. This is an extensively researched robotics problem called *the simultaneous localization and mapping* (abbr. SLAM) that can be coarsely categorized into two research problems on their own – *mapping* and *localization*. In mapping, we aim to represent the world accurately with a map, while in localization, according to the landmarks of a map, we aim to infer where the vehicle is. SLAM is also considered a *chicken-and-egg* problem [1, p. 282] – in order to capture a precise map of an environment, the vehicle should be localized well, and vice versa, localization is done right only if a precise map exists. Because robotics itself is an interdisciplinary branch of multiple studies, simultaneous localization and mapping, being a comprehensive robotics problem, can be tackled from multiple standpoints.

Turns out, the drift in the simultaneous localization and mapping can be reduced if the vehicle reappears in a place visited before. This situation suits us as we are aiming for the smallest drift possible, which implicates better mapping and better localization. To reappear at a place means that the vehicle traversed a circular trajectory that starts and ends in that place. Therefore, in the context of SLAM, this problem is called *the loop closure/closing detection* (abbr. LCD). Also, it is simply called *place recognition*. Place recognition is also a research problem on its own with specific quantitative evaluation measures. If a sensor modality of a SLAM system is visual, e.g., a mono or stereo camera, we are talking about *visual place recognition* (abbr. VPR). In visual place recognition, we deal with images of places visited by the vehicle. In the offline visual place recognition, a setup where we have multiple traversals of the same route captured in advance, we distinguish between two types of image streams – a query database $\mathcal{Q}$ and a reference database $\mathcal{D}$. For a newly obtained query image, we look into the reference database in order to find a match. The knowledge and insights obtained in offline visual place recognition are transferable to online visual place recognition, where we aim to recognize a place having a single stream of images from a single traversal. Many of the concepts are also transferable from visual place recognition to other types of place recognition, e.g., to place recognition with LiDAR scans.

If multiple, not too different, images depict the same place it would not be hard for a human to recognize it. In the real world, it is almost impossible to have multiple visually identical images of the same place, what would implicate that such a place can be easily recognized, so, visual place recognition would be trivial. Two categories of factors that cause visual dissimilarity in an image that depicts a place are identified as

- different environmental conditions, and,
- different viewpoints of the camera.

A place is characterized mostly by its static objects – buildings, trees, streetlights, etc. Then again, it is possible, especially in everyday traffic, that there is a lot of moving objects, also called *occlusions*, that just happened to be in a scene. It is clear that these objects do not contribute to the recognizability of a place. Also, different times of the day drastically affect visual appearance, not only because of a blue or black sky, but also because of different illumination of static and moving objects. Then, equal credit goes to different seasons and weather conditions. An extreme example, that even appears in a dataset we will use, would be snow. Even if environmental conditions are not that deviate, e.g., when performing online visual place recognition for loop closing detection in simultaneous localization and mapping where environmental conditions within a few minutes would not change the visual appearance that much, it is expected that significant deviations in viewpoints of the camera will occur. We want to undermine the impact of these two categories of unsuitable factors; therefore, our visual place recognition system should be *viewpoint-invariant* and *condition-invariant*. This, in turn, results in more accurate place recognition, i.e., in more accurate loop closing detection, which implicates better pose estimation. Complementary to viewpoint and condition variations is the problem of perceptual aliasing. It is possible that there are multiple visually similar places that are located at wholly different locations in a map. This put us in danger of loop closures being detected erroneously what, in turn, negatively affects mapping and implicitly, localization in SLAM.

Speaking of a visual place recognition system, in order to achieve condition and viewpoint invariance, and invulnerability to perceptual aliasing, the design aspects of such a system that should be considered are:

- *image representation*, and,
- *place matching*.

Although grayscale or RGB images can be used in visual place recognition [2, 3], such "plain" representations simply do not achieve good results. Numerical values in plain image representations change drastically even for the slightest viewpoint or environment change, so it almost impossible to compare images with each other. A better option would be to rely on existing *computer vision* techniques. In the not-too-distant past, handcrafted local feature techniques based on image gradients have been used in order to find *salient regions* within an image. Alongside local features, accompanied are local descriptors that assign a distinguishable vector to a salient region. Then we aggregate these local descriptors into a global image representation. In recent years, however, feature maps extracted from deep image models – convolutional neural networks – are mainly used as an image representation. To match places that are captured as images, once a good image representation is picked, we can take advantage of the fact that images, captured during the ride of a vehicle, are sequential and temporally ordered. Therefore, when deciding how to match a place, i.e., when deciding if that place is already captured and being depicted by another image, a few neighboring images can also be taken into account. This is the idea of *sequence-based* place matching methods that are investigated here.

1.2 Original scientific contributions

The thesis presents three scientific contributions to the problem of visual place recognition. According to the problem statement, we can improve visual place recognition by improving place matching and image representation. Therefore, the first and the second contributions deal with the above-mentioned design aspects. In order for visual place recognition not to be an end in itself, in the third contribution, we use it within simultaneous localization and mapping. The contributions are as follows:

#1 Sequence-based visual place matching method that uses directed acyclic graphs and single source shortest path.

The proposed place matching method, NOSeqSLAM [4, 5], generalizes an existing sequence-based place matching method SeqSLAM [2]. Being a sequence-based method, our method is able to match a place not only according to a single image of that place, but also according to neighboring images. Being a generalization of SeqSLAM, our method outperforms it quantitatively. This generalization is achieved with graph theory, where we model place recognition with directed acyclic graphs. These directed acyclic graphs have a specific topological structure; therefore, alongside a novel sequence-based place matching method itself, an accompanying tailor-made algorithm that finds single source shortest paths on such topologically sorted directed acyclic graphs is proposed too.

#2 Method for robust visual place recognition with deep representations that uses softmax regression and mutual information-based feature selection.

In this contribution, we showed how to adapt softmax regression in the context of visual place recognition by casting visual place recognition as a multinomial classification problem. Then, approved deep image models are optimized with the proposed approach, and accordingly, such optimized image models coupled with the proposed place matching method achieved significant improvements in quantitative results. In similar fashion, we demonstrated how to adapt a proven mutual information-based feature selection method for visual place recognition and justified this approach quantitatively.

#3 Procedure for adapting sequence-based visual place recognition method for loop closing in simultaneous localization and mapping algorithms.

Using the paradigm of vectorization, our proposed place matching method became even faster and executable on a graphic processing unit. The vectorized instance of our method is able to detect loop closures for simultaneous localization and mapping in real-time. This procedure showed to be effective in two different simultaneous localization and mapping implementations, one of them being a simulated LiDAR-based SLAM (what also shows how our method goes beyond the scope of visual place recognition and applies to place recognition in general), while another being a visual SLAM evaluated on a real-world dataset.

1.3 Outline of the thesis

Each of the upcoming chapters deals with a single of the scientific contributions just mentioned. Additionally, we add the concluding chapter.

Ch 2 This chapter starts with the visual place recognition formulation in general. First, the Bayesian formulation of the problem is provided. Then, the sequence-based place matching method, SeqSLAM, is presented because it has strongly influenced our proposed place matching method – NOSeqSLAM. We then show how NOSeqSLAM has been designed as a generalization of SeqSLAM. Furthermore, we talk about graph algorithms used in our method, and particularly, we present an algorithm that easily finds single source shortest paths for the topology of directed acyclic graphs in NOSeqSLAM. We conclude this chapter with the asymptotic and empirical running time analysis of SeqSLAM and NOSeqSLAM.

Ch 3 Firstly, this chapter provides an extensive overview of standard computer vision image models. It is described what good features in images are and how we detect them. Essentially, gradient-based handcrafted local features and descriptors SIFT[6] and SURF[7] are presented. An unsupervised machine learning technique, k-means clustering, helps us to find *visual words*, i.e., cluster centers from a large set of local descriptors. Then, we use aggregate local descriptors from an image, either with the bag-of-words [8] or the vector of locally aggregated descriptors [9] techniques, into

a global image descriptor. Additionally, we present a gradient-based global descriptor – histogram of oriented gradients. We do consider handcrafted image models because they are computationally undemanding. Then, we provide an overview of learned image models, convolutional neural networks, alongside their various design aspects. Unlike handcrafted image models, learned image models are more computationally demanding, however, they adapt to data distributions. Having a systematic overview of learned image models, we overview softmax regression and how visual place recognition can be formulated as a multinomial classification problem. We also present a mutual information-based feature selection technique and deploy it into our pipeline. Experimental sections reveal that image models enhanced with softmax regression and feature selection coupled with our place matching method NOSeqSLAM outperform other sequence-based place matching methods and rival image models.

Ch 4 This chapter is about the problem of loop closing in the simultaneous localization and mapping. First, we address factor graphs as an approach to model stochastic processes. Being a stochastic process, we explain how simultaneous localization and mapping can be conducted by means of factor graphs. Afterward, a section is dedicated to the vectorization of NOSeqSLAM in order to execute it in real-time with the possibility of running on a graphic processing unit. Then we create SLAM № 1, a simulated SLAM environment with a range-based sensor and ICP-based odometry, in order to adapt and try out the vectorized NOSeqSLAM in the context of SLAM. Finally, we create SLAM № 2, a visual odometry-based SLAM. Once again, we try out the vectorized NOSeqSLAM, now on the real-world KITTI dataset [10]. For both SLAM systems that have been implemented, the results obtained show how NOSeqSLAM successfully detects loop closures. More precisely, without loop closing "turned on", SLAM № 1 and SLAM № 2 achieve higher absolute trajectory error measures.

Ch 5 In the concluding chapter, we are looking back at the scientific contributions that have been achieved. In the end, we say a few words about future work.

2

Sequence-based visual place recognition

Visual place recognition, as said in [11], "is a well-defined but extremely challenging problem to solve in the general sense [...]". In visual place recognition, it has to be decided whether a place, given an image of it, has already been seen, and if so, what place it is. A place can be considered "[...] as the abstraction of a region" where a region is a subset of the environment [12]. That view of a place brings us to *a topological map* – a collection of nodes that represent places as abstract regions that are connected via arcs that represent travel paths [13]. Also mentioned in [13], a place has its *signature*, a set of features that are maximized at that place. Complementary to topological maps are metric maps, while there also exist blends of these two. A map, as a collection of places, is important in the context of visual place recognition because we would like to compare a newly obtained image with those places already contained in that map. Also, visual place recognition can be seen as a specialization of *the visual instance retrieval problem* where "[...] given a new unlabeled query image, the task is to find the same object or scene as in the query." [14]. By looking at visual instance retrieval data, e.g., the Oxford5k [15] and Paris6k [16] datasets that depict notable places of eponymous cities obtained from Flickr, it is characteristic how there are no topological relations at all. Obviously, multiple images of, e.g., Arc de Triomphe in Paris, uploaded from multiple sources, truly depict this place, but no relations, i.e., arcs, exist between these images and images of, e.g., the Eiffel tower. In contrast, visual place recognition data, to the best of our knowledge, are captured while the vehicle is driving. This imposes additional relations between data, i.e., there is a preceding or succeeding place for some place. Everything we have just said about maps in visual place recognition is neatly summarized in [11, p. 6] where *degrees of map abstraction* are:

- *pure image retrieval* maps without positional information between places,
- *topological* maps that include positional information between places, and,
- *topological-metric* maps that include both positional and metric information between places.

In our opinion, once the metric information is included, we go beyond the scope of place recognition into the scope of simultaneous localization and mapping, which lines up with the mentioned fact of how place recognition can be used for loop closing detection, i.e., it can be considered a SLAM subsystem.

Accordingly, maps in visual place recognition are topological maps with sequentially ordered images of places, albeit metric maps in simultaneous localization and mapping we are accustomed to, are far richer in its content. Therefore, it is, to an extent, disorienting talk about maps in visual place recognition at all. Preferably, we resort to the nomenclature borrowed from [17] and the mathematical notation borrowed from [18] that applies to *offline visual place recognition*[1]. In offline visual place recognition, we have two or more sets of images captured while driving around the same route. One set, referred to as *a reference database*, denoted with

$$\mathcal{D} = \{I_{d_1}, \ldots, I_{d_j}, \ldots, I_{d_{|\mathcal{D}|}}\}, \tag{2.1}$$

is a set of images against a single query image is compared. Another set, referred to as *a query database*, denoted with

$$\mathcal{Q} = \{I_{q_1}, \ldots, I_{q_i}, \ldots, I_{q_{|\mathcal{Q}|}}\} \tag{2.2}$$

is a set of images for which, provided that they depict an already seen place, a candidate from a reference database should be found. Having the notation of $\mathcal{D}$ and $\mathcal{Q}$, we define offline visual place recognition as a task in which, given a query image $I_{q_i} \in \mathcal{Q}$, a respective match $I_{q_i^*} \in \mathcal{D}$ should be found. An integral part of a visual place matching method, once we have images at our disposal, should be the way they are represented. A comprehensive consideration about this topic in provided in Chapter3. By using a specific image model, it will map an RGB/grayscale image $I \in \mathcal{Q} \cup \mathcal{D}$ into an alternative vector/matrix/tensor image representation $z \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. Hopefully, this image representation should be resistant to viewpoint variations, condition variations, and perceptual aliasing so that places are matched correctly.

Evaluation criteria that qualitatively assess visual place recognition are borrowed from information retrieval. Ultimately, we would like to correctly match all query images while also being confident about these matches. We denote *ground truth* matches for a given query image $I_{q_i} \in \mathcal{Q}$ with $\mathcal{GT}(I_{q_i}) \subseteq \mathcal{D}$. An example of ground truth matches shown as an image can be seen in Figure3.19awhere the intersection of an i-th column and a j-th row tells if $I_{d_j} \in \mathcal{D}$ is a match for $I_{q_i} \in \mathcal{Q}$ (yellow cell) or is not (purple cell). Reasonably, images $\{I_{q_i}\} \cup \mathcal{GT}(I_{q_i})$ should depict a same place. *A true positive match* for I_{q_i} is every $I_{d_j} \in \mathcal{GT}(I_{q_i})$ that has been proposed by a visual place recognition method. *A false negative match* for I_{q_i} is $I_{d_j} \in \mathcal{GT}(I_{q_i})$ being discarded in a mistaken way due to a low confidence measure. If a place recognition method suggests, with a sufficient confidence measure, that $I_{d_j} \notin \mathcal{GT}(I_{q_i})$ is a match for I_{q_i}, we are talking about *a false positive match*. Let TP denote the total number of true positives, FP the total number of false positives, and FN the total number of false negatives. Then, we define precision with

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{2.3}$$

and interpret it as "[…] a measure of how many of the identified places are actually correct […]" [19] and recall as

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2.4}$$

[1] In Section4.3, we will also introduce the online place recognition notation.

and interpret it as "[…] a measure of how many of the correct places were found […]" [19]. An ultimate place recognition method would achieve 100% precision and 100% recall. Usually, slight deviations from ground truth are tolerated and are collectively referred to as *the localization radius*. The localization radius can be expressed either by the number of frames as in [20] or by a measure of distance (e.g., in meters) as in [17]. Simultaneously having a high recall and a high precision instructs us that there are not many places misclassified and that we are confident about correct matches. By varying the threshold for a confidence measure, i.e., the threshold between true positives and false negatives, *a precision-recall curve* with precision on y axis against recall on x axis is created with an example in Figure2.1a. An additional quantitative measure of performance for visual place recognition is obtained by integrating the area under a precision-recall curve, and this measure is called the area under a curve (abbr. AUC). Another frequently used qualitative measure is the maximum recall at 100% precision (abbr. R@100%P) which indicates a good loop closing detection performance as stated in [21]: "[…] an excellent LCD method should detect as many loop closing pairs as possible, which means the recall rate should be high. Therefore, when the precision rate is 100%, the maximum recall rate is a significant index for evaluating the LCD performance."

2.1 Sequence-based visual place recognition

Given an image $I_{q_i} \in \mathcal{Q}$, a naive approach to visual place recognition would be to find its most similar counterpart $I_{q_i^*} \in \mathcal{D}$ just according to a similarity measure s (e.g., cosine similarity), i.e.,

$$I_{q_i^*} = \arg\max_{I_{d_j} \in \mathcal{D}} s(z_{q_i}, z_{d_j}), \tag{2.5}$$

or a dissimilarity measure d (e.g., the Euclidean distance), i.e.,

$$I_{q_i^*} = \arg\min_{I_{d_j} \in \mathcal{D}} d(z_{q_i}, z_{d_j}). \tag{2.6}$$

However, as noticed in [22], "matching images just according to the best similarity score produces considerable (number of) false positives […]". Just as the robot pose can be estimated recursively by including the most recent measurement and control input [1, Chapter 2], which essentially means that all measurements and control inputs by far have been included in reaching the current state estimation, we can act in a similar way in visual place recognition. *Sequence-based* place matching methods, as called by [23], exploit the local neighborhood of a place monitored by the vehicle. The way neighboring places are observed is defined by a sequence-based method.

2.1.1 *FAB-MAP*

A probabilistic approach to visual place recognition is used in FAB-MAP [24] where location estimation is modeled via a recursive Bayes' formula as

$$p(L_i \mid \mathcal{Z}^k) = \frac{p(\mathcal{Z}^k \mid L_i, \mathcal{Z}^{k-1}) p(L_i \mid \mathcal{Z}^{k-1})}{p(\mathcal{Z}^k \mid \mathcal{Z}^{k-1})}. \tag{2.7}$$

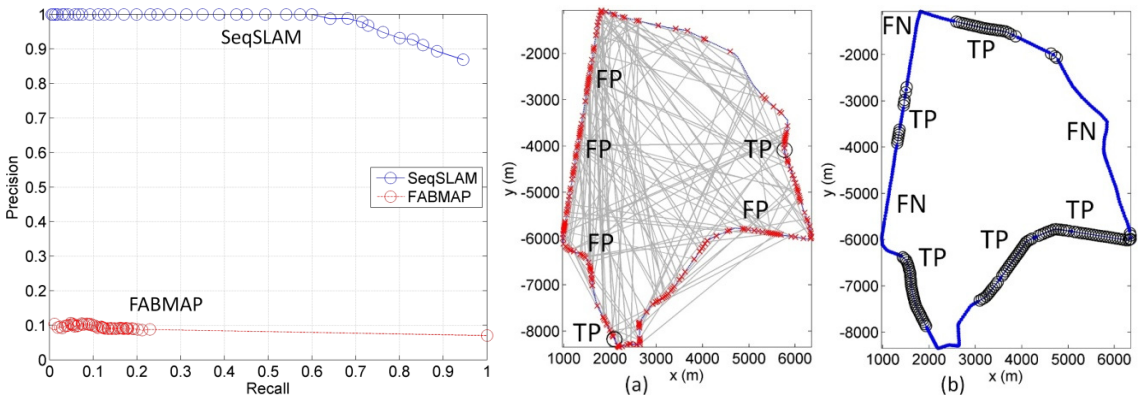


Figure 2.1: SeqSLAM [2] outperforms FAB-MAP [24], which can be displayed by showing (a) a precision-recall curve and (b) true positive, false negative, and false positive matches in a ground plan. Images taken from [2].

L_i is an element of a topological map that contains disjoint and discrete locations, and $\mathcal{Z}^k$ are all measurements up to time k. For an image model, FAB-MAP uses local SIFT [6] features in combination with bag-of-words [8]. Although supported by the theoretical background of Bayes' formula, this approach, unfortunately, is significantly outperformed by the following method – SeqSLAM (Figure2.1).

2.1.2 *SeqSLAM*

SeqSLAM [2] is a sequence-based place matching method named after the characteristic property of constructing sequences in order to find place matches. The aforementioned term "local neighborhood" in the context of offline SeqSLAM refers to $\lfloor \frac{d_s}{2} \rfloor$ previously seen places given a query image $I_{q_i} \in \mathcal{Q}$ and $\lfloor \frac{d_s}{2} \rfloor$ upcoming places to be seen. The same is true for $I_{d_j} \in \mathcal{D}$, although the number of neighboring images is defined by the method's hyperparameters. Then, a value that measures how well I_{d_j} fits to I_{q_i} is obtained by observing $2\lfloor \frac{d_s}{2} \rfloor + 1$ places from $\mathcal{Q}$ and $2\lfloor \frac{d_s}{2} \rfloor + 1$ places from $\mathcal{D}$.

The main data structure SeqSLAM deals with is called *the difference matrix* $D \in \mathbb{R}^{|\mathcal{D}|\times|\mathcal{Q}|}$. Let $z_{q_i} \in \mathbb{R}^n$ and $z_{d_j} \in \mathbb{R}^n$ denote some image representation[2] of images $I_{q_i} \in \mathcal{Q}$ and $I_{d_j} \in \mathcal{D}$, respectively. Then D is defined as

$$D[j,i] = \sum_{k=1}^{n} |z_{q_i}[k] - z_{d_j}[k]|. \tag{2.8}$$

Additionally, *a contrast-enhanced* difference matrix $\hat{D} \in \mathbb{R}^{|\mathcal{D}|\times|\mathcal{Q}|}$ is obtained by performing *normalization* on each column as described in [2]. Then the correspondence measure s_{q_i,d_j} between I_{q_i} and I_{d_j} is defined as

$$s_{q_i,d_j} = \min_{v} \sum_{t=i-\lfloor \frac{d_s}{2} \rfloor}^{i+\lfloor \frac{d_s}{2} \rfloor} \hat{D}[j + v(t-i), t]. \tag{2.9}$$

[2] Downsampled grayscale images used in the original work.

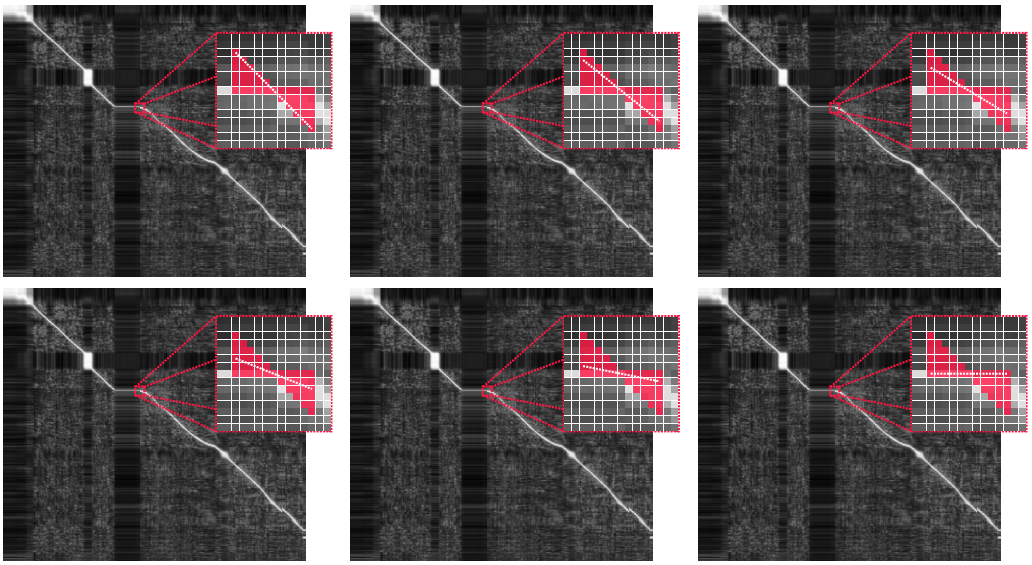


Figure 2.2: By changing the velocity parameter $v_{\min} \le v \le v_{\max}$ in SeqSLAM, we form different linear sequences over the difference matrix. Then, we accumulate differences according to (2.9) and pick an optimal one as in Figure2.5a.

What characterizes SeqSLAM is a sequence of indices $((j + v(t - i), t))_{t=i-\lfloor d_s/2 \rfloor}^{i+\lfloor d_s/2 \rfloor}$ that form linear segments over the difference matrix as illustrated in Figures2.2,2.3b. Having d_s fixed and evaluating sequences for each velocity $v_{\min} \le v \le v_{\max}$ we pick such a velocity that minimizes (2.9) what is illustrated in Figure2.5a. Note that the lesser s_{q_i,d_j} means I_{d_j} is the better match for I_{q_i}. Conversely, we can use any representation for images, and moreover not to measure a difference, e.g., we could measure a similarity between images. Let denote a matrix that measures similarities, say with the cosine distance, the association matrix A. In such a setup we maximize (2.9) by v, while the greater s_{q_i,d_j}, the better. Therefore, we can say that this algorithm, as well as the upcoming cone-based SeqSLAM and NOSeqSLAM, are *representation agnostic*.

2.1.3 *Cone-based SeqSLAM*

In the space of admissible sequences defined by d_s and $v_{\min} \le v \le v_{\max}$, also called *a cone* (Figure2.3a), cone-based SeqSLAM [25] counts a portion of the most similar matches between $\mathcal{Q}$ and $\mathcal{D}$. If we fix some query image $I_{q_t} \in \mathcal{Q}, t \in \{i - \lfloor \frac{d_s}{2} \rfloor, \dots, i + \lfloor \frac{d_s}{2} \rfloor\}$, we check whether the index d_t^* of the most similar/the least different $I_{d_t^*} \in \mathcal{D}$ for I_{q_t} is in $\{j + v(t - i) : v_{\min} \le v \le v_{\max}\}$, which means that the best match $(I_{q_t}, I_{d_t^*}) \in \mathcal{Q} \times \mathcal{D}$ resides in a cone. Then, as illustrated in Figure2.3c, we count all such pairings and the correspondence measure s_{q_i,d_j} is defined as

$$s_{q_i,d_j} = \frac{1}{d_s} \sum_{t=i-\lfloor \frac{d_s}{2} \rfloor}^{i+\lfloor \frac{d_s}{2} \rfloor} \mathbb{1}_{d_t^* \in \{j+v(t-i):v_{\min} \le v \le v_{\max}\}}. \tag{2.10}$$

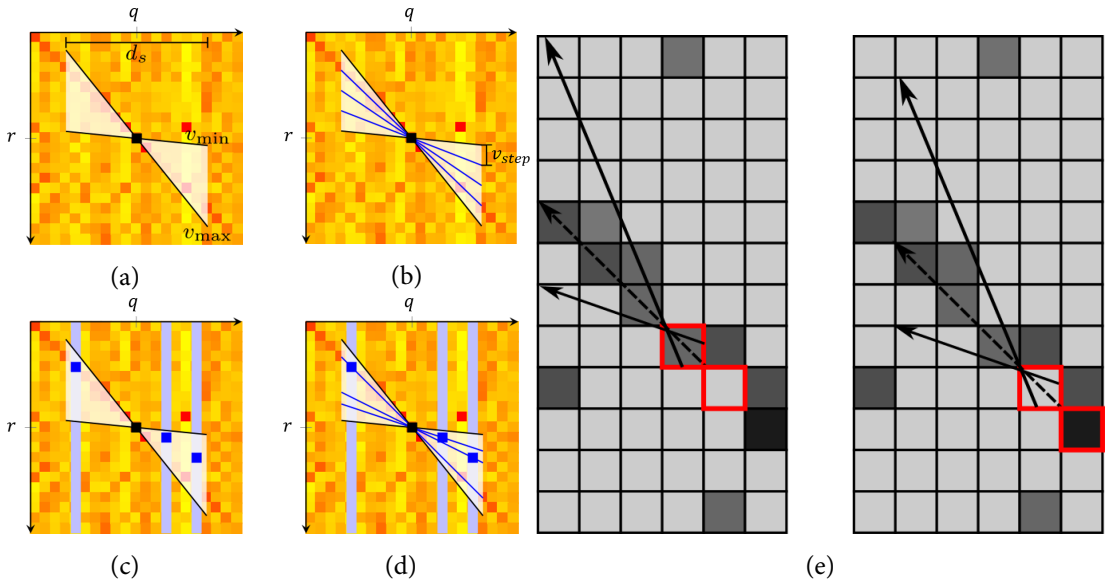


Figure 2.3: (a) The admissible search space in SeqSLAM and its derivatives is defined with hyperparameters d_s, $v_{\min}$, $v_{\max}$ and v_{step}. (b) In the original SeqSLAM method, we construct linear sequences $\left((j + v(t-i), t)\right)_{t=i-\lfloor d_s/2 \rfloor}^{i+\lfloor d_s/2 \rfloor}$ for $v_{\min} \leq v \leq v_{\max}$ that form linear segments over the difference matrix. (c) The cone-based SeqSLAM counts a portion of the most similar matches between $\mathcal{Q}$ and $\mathcal{D}$ in a cone. (d) The hybrid method is a blend between the original and cone-based SeqSLAM where those linear sequences that pass through global best matches are evaluated. (e) Fast-SeqSLAM evaluates (2.9) for hypotheses obtained with k nearest neighbor search with an additional hypothesis obtained according to an optimal velocity one step before. Images (a), (b), (c), and (d) are taken from [25]. Image (e) is taken from [26].

2.1.4 *Other variants of SeqSLAM*

A hybrid between the original and cone-based SeqSLAM, *the hybrid method* [25], for the correspondence measure evaluates those linear sequences that pass through "global best matches" and then picks an optimal (Figure2.3d). Fast-SeqSLAM [26] enhances SeqSLAM by evaluating sequences for only k nearest neighbor reference images as match hypotheses (darker cells in Figure2.3e). Additional match hypotheses for which sequences are evaluated (red cells in Figure2.3e) are calculated by presuming that the vehicle has since then moved with a previous-time-optimal velocity. Combining odometry, SMART[23] "eases" SeqSLAM evaluation in a way that images are captured "…at constant distance intervals, rather than constant time intervals." The absence of odometry, where images cannot be captured at constant distance intervals, imposes an issue for SeqSLAM and its derivatives mentioned by far. A similar issue that yields the same effect is a significant difference in velocities between multiple traversals of the same route. Luckily, our proposed method is able to account for such a negative phenomenon as follows.

2.2NOSEQSLAM

The space of admissible sequences in NOSeqSLAM [4, 5] is defined by the sequence length d_s and the expansion rate η. Given $(I_{q_i}, I_{d_j}) \in \mathcal{Q} \times \mathcal{D}$, we build *a directed acyclic graph*

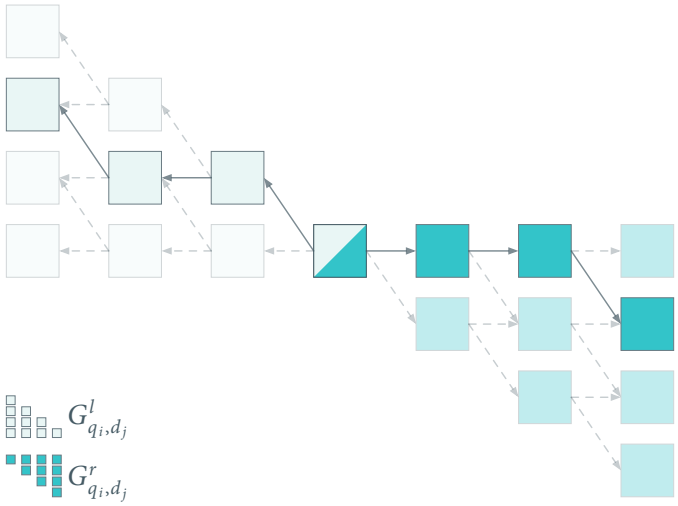


Figure 2.4: In order to account for nonlinear correlation in velocities between multiple traversals of a route, instead of constructing linear sequences as in SeqSLAM, we construct the left directed acyclic graph $G^l_{q_i,d_j}$ and the right directed acyclic graph $G^r_{q_i,d_j}$. Next, for both graphs, we evaluate a single source shortest path algorithm from the root to leaves of graphs and this way obtain a measure of correspondence between $I_{q_i} \in \mathcal{Q}$ and $I_{d_j} \in \mathcal{D}$.

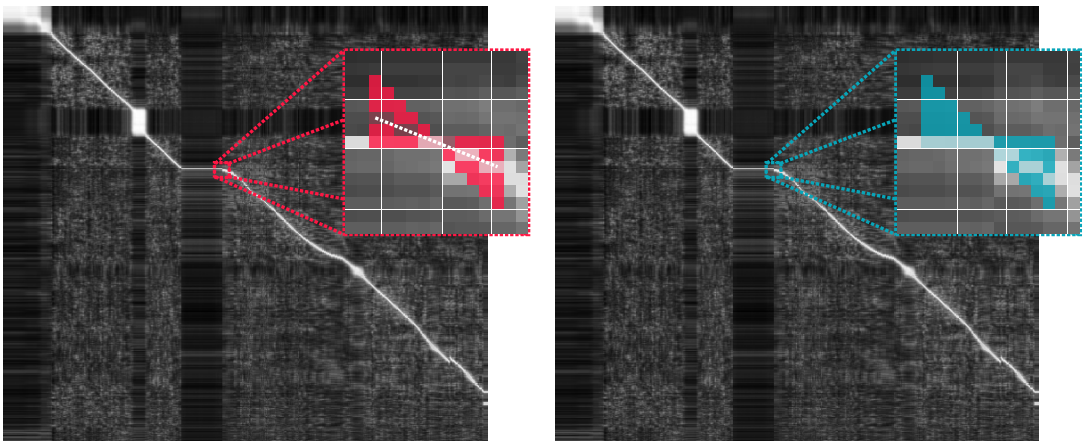
$G^l_{q_i,d_j}$ such that (I_{q_i}, I_{d_j}) is the root node that has η children $\{(I_{q_{i-1}}, I_{d_j}), \ldots, (I_{q_{i-1}}, I_{d_{j-\eta+1}})\}$. Recursively, each $(I_{q_{i-1}, I_{d_{j'}}})$ node has η children $\{(I_{q_{i-2}}, I_{d_{j'}}), \ldots, (I_{q_{i-2}}, I_{d_{j'-\eta+1}})\}$ and so on until the depth of $\lfloor \frac{d_s}{2} \rfloor$ is reached. Edges are directed from parent to children. This graph is called *the left subgraph* (shown as the white graph in Figure2.4). Symmetrically, we build *the right subgraph* $G^r_{q_i,d_j}$ (shown as the cyan graph in Figure2.4) by expanding $\{(I_{q_{i+1}}, I_{d_j}), \ldots, (I_{q_{i+1}}, I_{d_{j+\eta}})\}\ldots$ Then $G_{i,j} = (\mathcal{V}_{i,j}, \mathcal{E}_{i,j}, w)$ is defined as the union of these two subgraphs while the weight function $w : \mathcal{E}_{i,j} \to [0,1]$ is defined as

$$w((I_{q_k}, I_{d_l}), (I_{q_m}, I_{d_n})) = 1 - A[n, m]. \tag{2.11}$$

Let l^* denote the shortest path (in terms of accumulated weights) from $u_{i,j}$ to a leaf in the left directed acyclic graph and r^* denote the shortest path from $u_{i,j}$ to a leaf in the right directed acyclic graph. Moreover, let V_{l^*} and V_{r^*} denote nodes of the corresponding shortest paths respectively, and let $V^*_{\text{s.p.}} = V_{l^*} \cup V_{r^*}$.Then, NOSeqSLAM gives a measure of the similarity between $I_{q_i} \in \mathcal{Q}$ and $I_{d_j} \in \mathcal{D}$ as

$$s_{q_i,d_j} = \sum_{u_{k,l} \in V^*_{\text{s.p.}}} \frac{z_{q_k}^T z_{d_l}}{\|z_{q_k}\| \|z_{d_l}\|}. \tag{2.12}$$

NOSeqSLAM is the generalization of SeqSLAM because, by using shortest paths, sequences are not exclusively linear as in (2.9) where indices $((j + v(t-i), t))_{t=i-\lfloor d_s/2 \rfloor}^{i+\lfloor d_s/2 \rfloor}$ are used. This has a positive impact on results as NOSeqSLAM enables to capture *nonlinear correlation* of acceleration between multiple traversals, i.e., when $\mathcal{D}$ and afterward $\mathcal{Q}$ are captured. On the other hand, SeqSLAM is capable of capturing only linear correlation, e.g., when a vehicle traverses both $\mathcal{D}$ and $\mathcal{Q}$ with equal velocities. However, real-world traffic is significantly changeable; therefore, linear correlation in velocities is very unlikely. This is visible in Figure2.5where multiple not that similar matches from a cone (dark squares



(a) (b)

Figure 2.5: (a) SeqSLAM is unable to account for nonlinear correlation in velocities between multiple traversals – multiple dissimilar matches from a cone participate in the correspondence measure. (b) Using a single source shortest path, NOSeqSLAM is able to capture nonlinear correlation – only similar matches participate in the correspondence measure.

below linear sequence) participate in the correspondence measure for SeqSLAM. On the other hand, NOSeqSLAM included only those matches that are really similar (no dark squares show up below the union of shortest paths).

2.3 Tailored single source shortest path algorithm

There are numerous ways shortest paths in a graph can be found. In general, the wider the category of graphs a shortest path algorithm can be applied to, the more asymptotic time it takes. The Bellman-Ford algorithm [27] is a single source shortest path algorithm applicable to the widest category of graphs – graphs that can have negative weights on its edges and has the asymptotic running time of $\Theta(|\mathcal{V}||\mathcal{E}|)$ where $|\mathcal{V}|$ is the number of vertices and $|\mathcal{E}|$ is the number of edges. An asymptotically better option for graphs with non-negative weights where cycles are possible, although irrelevant because shortest paths cannot contain cycles, is the famous Dijkstra algorithm [28] with the asymptotic running time of $\Theta(|\mathcal{V}| \lg |\mathcal{V}| + |\mathcal{E}|)$. The most asymptotic-running-time acceptable algorithm, applicable only to directed acyclic graphs, and therefore applicable to NOSeqSLAM, is to sort nodes *topologically* and then, in topological order, do relaxation on edges. This procedure has the asymptotic running time of $\Theta(|\mathcal{V}| + |\mathcal{E}|)$.

Relaxation is a mechanism used to maintain the best shortest path hypothesis for each node starting from a source node s. Colloquially – if there is a better shortest path hypothesis than a current one being detected, we use it instead. According to the notation from [29], let w denote the weight function, let $u.adj$ denote nodes adjacent to u, let $u.\pi$ denote the predecessor in the shortest of a node $u \in V$, and let $u.d$ denote an estimate for the shortest path weight which will ultimately, when a shortest path algorithm terminate, become an exact shortest path weight. This procedure is given in Algorithm1and takes $\Theta(1)$ asymptotic running time.

A topological sort of a directed acyclic graph [30] is an ordering where, if the graph

Algorithm 1 Relax

Input: $(u, v) \in E$
if $v.d > u.d + w(u, v)$ **then**
    $v.d \leftarrow u.d + w(u, v)$
    $v.\pi \leftarrow u$
end if

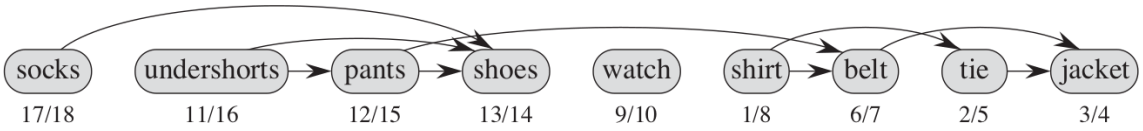


Figure 2.6: An example of topological sort where nodes are listed in a way that, if $(u, v) \in E$, then u appears before v. Times of nodes discovering/completion from the depth-first search algorithm are displayed below nodes. Nodes are listed from left to right in descending order with respect to their completion time in the depth-first search. Image taken from [29].

contains an edge (u, v), then u appears before v [29]. An illustrative example of a topologically sorted graph can be found in Figure2.6. Topological sort is established by evaluating another popular graph algorithm – *depth-first search*. Depth-first search dates from the 19th century [31], while, in the modern-day graph theory, it has been recognized since the publication of [32]. In depth-first search, we recursively examine undiscovered successors of a certain node $u \in V$, then their successors, and so on. Whenever a new node is discovered, we increase the counter value by one and assign this value as the time of its discovery. When there are no more successors to discover, thus all undiscovered successors have been found, we also increase the counter value by one and assign this value as the time when the depth-first search for this node just ended. This description gives us an intuition for the topological sort of a graph – the later the depth-first search for a node terminates, the sooner this node should appear in order because its directed edges point to neighboring nodes for which the depth-first search completed earlier. A recursive pseudocode for depth-first search can be found in [29, p. 604], while a non-recursive pseudocode can be found in [31, p. 50]. The pseudocode for topological sort is given in Algorithm2. Finally, it is easy to define the single source shortest path algorithm for a topologically sortable directed acyclic graph (Algorithm3).

Initially, Algorithm3has been used as a single source shortest path algorithm for NOSeqSLAM. Unfortunately, for the Bonn dataset (Subsection3.3.3) with $|\mathcal{Q}| = 544$ and

Algorithm 2 Topological sort

Input: Graph $G = (V, E)$
Output: Linked list L
Init linked list L
Run depth-first search for G
As soon depth-first search for $u \in V$ is completed, prepend u to L

Algorithm 3 Single source shortest path for directed acyclic graphs

Input: Graph $G = (V, E)$, source node $s \in V$
$u.\pi \leftarrow null, \forall u \in V$
$u.d \leftarrow \infty, \forall u \in V \setminus \{s\}$
$s.d \leftarrow 0$
Sort G topologically with Algorithm2
for each $u \in V$ taken in topologically sorted order **do**
  for each $v \in u.adj$ **do**
    RELAX$((u, v))$
  end for
end for

$|\mathcal{D}| = 488$ images, it took a little over an hour for NOSeqSLAM to perform an offline visual place recognition experiment given fixed hyperparameters. The cause is, in order to evaluate Algorithm3, a new graph in memory has to be constructed. This is a time-consuming operation because graphs, assuming *the adjacency-list representation* [29, p. 590], are nonlinear yet dynamically allocated structures. Being nonlinear, it is also time-consuming to fetch properties of a graph from memory, i.e., its nodes, edges, node attributes, etc. Then, there is a burden of topological sort that requires depth-first search to be evaluated, a linked list, also a nonlinear[3] structure, to be allocated, etc. Notice how it is redundant to, if a graph does not exist, to construct it, then topologically sort it in order to relax its edges in order to find shortest paths. That premise, as it turned out, spared us from redundant dynamic memory allocation and reduced running times drastically. We figured out a way in which two nested loops themselves can traverse respective edges and nodes of graphs $G^l_{q_i,d_j}$ and $G^r_{q_i,d_j}$ in a topologically-sorted order, accounting boundary conditions, and later on, it was easy to apply Algorithm1in order to find shortest paths. Edges and nodes of $G^l_{q_i,d_j}$ and $G^r_{q_i,d_j}$ are already contained in the association matrix A and can be directly accessed with nested loops' indices. This algorithm, called *the on-the-fly relaxation*, is presented in our paper [5] and is given in Algorithm4. It can, therefore, be considered a more efficient substitute of Algorithm3where an evaluation of Algorithm2is no longer needed. Interestingly, these graphs are topologically sortable in two different directions – one being defined with the outer loop while another being defined with the inner loop from Algorithm4.

In Algorithm5we provide the NOSeqSLAM pseudocode in order to analyze its asymptotic running time. For the sake of clarity, let $\delta = \lfloor \frac{d_s}{2} \rfloor$. The number of nodes both for $G^l_{q_i,d_j}$ and $G^r_{q_i,d_j}$ given d_s and η is

$$n_{\text{nodes}} = (\delta + 1) \cdot \Big(\frac{(\eta - 1) \cdot \delta}{2} + 1 \Big), \tag{2.13}$$

[3] A linked list is a nonlinear structure in the way its elements are stored in memory, albeit it is a linear structure in the way these elements are visited.

Algorithm 4 On-the-fly relaxation for NOSeqSLAM

Input: $G^l_{q_i,d_j}, G^r_{q_i,d_j}, d_s, \eta$
Output: all shortest paths from the root $u_{i,j}$ to leaves of $G^l_{q_i,d_j}$ and $G^r_{q_i,d_j}$
for $i_{\text{offset}} = 1$ to $\lfloor \frac{d_s}{2} \rfloor$ **do**
  for $j_{\text{offset}} = 0$ to $i_{\text{offset}} \cdot (\eta - 1)$ **do**
    $i'_l \leftarrow i - i_{\text{offset}}$
    $j'_l \leftarrow j - j_{\text{offset}}$
    if $0 \le i'_l \le |\mathcal{Q}| - 1$ and $0 \le j'_l \le |\mathcal{D}| - 1$ **then**
      for each $u_{\tilde{i},\tilde{j}} \in \textsc{Predecessors}(u_{i'_l,j'_l})$ **do**
        $\textsc{Relax}((u_{\tilde{i},\tilde{j}}, u_{i'_l,j'_l}))$
      end for
    end if
    $i'_r \leftarrow i + i_{\text{offset}}$
    $j'_r \leftarrow j + j_{\text{offset}}$
    if $0 \le i'_r \le |\mathcal{Q}| - 1$ and $0 \le j'_r \le |\mathcal{D}| - 1$ **then**
      for each $u_{\tilde{i},\tilde{j}} \in \textsc{Predecessors}(u_{i'_r,j'_r})$ **do**
        $\textsc{Relax}((u_{\tilde{i},\tilde{j}}, u_{i'_r,j'_r}))$
      end for
    end if
  end for
end for

Algorithm 5 NOSeqSLAM

Input: A, d_s, η
Output: $s_{q_i,d_j}, \forall I_{q_i} \in \{I_{q_{\lfloor \frac{d_s}{2} \rfloor + 1}}, \ldots, I_{q_{|\mathcal{Q}| - \lfloor \frac{d_s}{2} \rfloor}}\}, \forall I_{d_j} \in \mathcal{D}$
for each $I_{q_i} \in \mathcal{Q}$ **do**
  for each $I_{d_j} \in \mathcal{D}$ **do**
    if $0 \le q_i \le \lfloor \frac{d_s}{2} \rfloor$ or $|\mathcal{Q}| - \lfloor \frac{d_s}{2} \rfloor + 1 \le q_i \le |\mathcal{Q}|$ **then**
      continue
    end if
    Calculate $V^*_{\text{s.p.}}$
    $s_{q_i,d_j} = \sum_{u_{k,l} \in V^*_{\text{s.p}}} \frac{z^T_{q_k} z_{d_l}}{\|z_{q_k}\| \|z_{d_l}\|}$
  end for
end for

while the number of edges for both graphs is

$$n_{\text{edges}} = \eta \cdot \delta \cdot \Big(\frac{(\eta - 1) \cdot (\delta - 1)}{2} + 1 \Big). \tag{2.14}$$

If the Bellman-Ford algorithm is used in NOSeqSLAM, then its asymptotic running time would be $\Theta(|\mathcal{Q}| \cdot |\mathcal{D}| \cdot \delta^4 \cdot \eta^3) = \Theta(|\mathcal{Q}| \cdot |\mathcal{D}| \cdot d_s^4 \cdot \eta^3)$. If Dijkstra is used, NOSeqSLAM takes

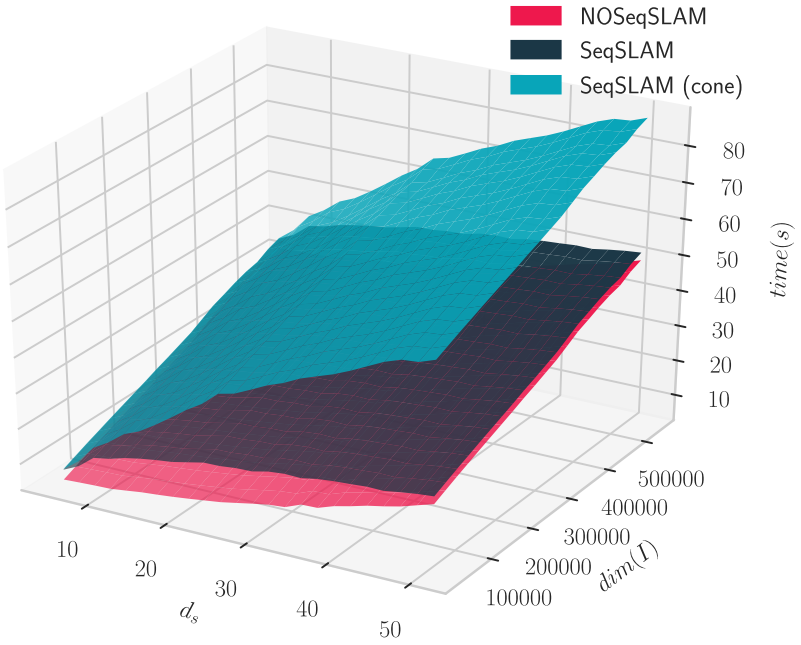


Figure 2.7: Running times with respect to sequence lengths and a number of image features for SeqSLAM, cone-based SeqSLAM, and NOSeqSLAM on the Bonn dataset. Empirically, NOSeqSLAM has better running times, while asymptotically, SeqSLAMs win. Image taken from our paper [5].

$\Theta(|\mathcal{Q}| \cdot |\mathcal{D}| \cdot d_s^2 \cdot \eta \cdot (\lg(d_s^2 \cdot \eta) + \eta))$. Asymptotic running times with both the standard topological sort and on-the-fly relaxation are $\Theta(|\mathcal{Q}| \cdot |\mathcal{D}| \cdot d_s^2 \cdot \eta^2)$. The original and cone-based SeqSLAM both take $\Theta(|\mathcal{Q}| \cdot |\mathcal{D}| \cdot d_s \cdot |V_{\text{steps}}|)$ asymptotically where $V_{\text{steps}} = \{v : v_{\min} \leq v \leq v_{\max}\}$ is a set of evenly spaced velocities.

Additionally, we have been observing empirical running times for the aforementioned sequence-based methods on the Bonn dataset using a laptop with `i7@2.8GHz` processor. We evaluated NOSeqSLAM (with Algorithm4), SeqSLAM and cone-based SeqSLAM changing $d_s \in \{5, 7, \ldots, 51\}$ and dimensionality of image representation $dim(I)$ ranging from 27034 to 540672 features. This image representation with such a number of features is obtained with a technique that will be presented in Section3.4. Running times needed to construct an association matrix and then apply a sequence-based method with respect to sequence length and a number of features are shown in Figure2.7. It is visible that NOSeqSLAM has the lowest running times compared to SeqSLAM and cone-based SeqSLAM. Also, it is visible that SeqSLAM methods have a better asymptotic running time, i.e., for a large enough d_s, these methods will become faster than NOSeqSLAM. However, these sequence lengths are just sufficiently large to incorporate a local neighborhood. Another batch of running time experiments will be presented in Section4.3.

2.4 Summary

In this chapter, one of the design aspects that constitute a visual place recognition system, place matching, has been covered. First, we have reasoned how to model sequential visual place recognition data using sequence-based methods. Then, we have presented our

approach to sequence modeling in place recognition that generalizes an existing sequence-based method – NOSeqSLAM. Additionally, we have presented an efficient single source shortest path algorithm applicable to a specific topological structure in our method. This greatly, but not entirely, justifies the first scientific contribution: Sequence-based visual place matching method that uses directed acyclic graphs and single source shortest path. We say "not entirely" because we would also like to justify this contribution quantitatively. Because both the aspects, place matching and image representation, are interdependent to a great extent, in the upcoming chapter, we will systematically cover different image models. Having both aspects covered, we can present the experiments that have been conducted and, consequently, quantitative results.

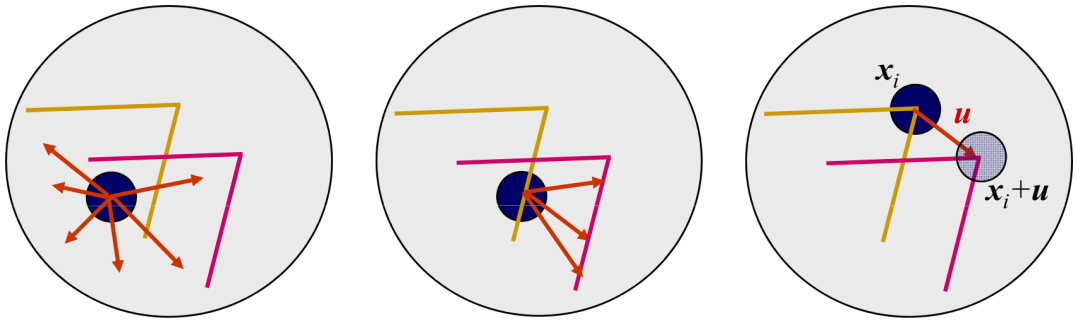
3

On image models in the context of visual place recognition

Image models are used whenever we have to make proper inferences about image data. Image models specify how to create *an image representation* – a way to represent an image in a computer program. In the previous chapter, we investigated how to match places. Therefore, in this chapter, we investigate how to represent a place by using image models. Given a raw image, either a grayscale one with $W \times H$ intensity values from 0 to 255 (i.e., $W \times H$ grayscale pixels) or an RGB image with $3 \times W \times H$ intensity values (i.e., $W \times H$ RGB pixels), we would like to map such raw image so that this mapped value fits a specific computer vision task. Visual place recognition is, in particular, a computer vision task too where we aim to find appropriate image models, so that good quantitative results are obtained. Certainly, some image models are more suitable than others; therefore this chapter provides a comprehensive explanation and evaluation of different image models and how they fit in the context of visual place recognition. We cannot know which image representations are better than others unless we evaluate different visual place recognition methods quantitatively. In this chapter we will broadly divide image models into two categories: *handcrafted image models* that will be examined in Section3.1and *learned image models* examined in Section3.2. In Section3.3, we will present the proposed approach to softmax regression in visual place recognition, while the proposed approach to feature selection will be presented in Section3.4. In the experimental evaluation of these two sections, different visual place recognition methods will be deployed in order to quantize which image models are a good choice in visual place recognition and which are not.

3.1HANDCRAFTED IMAGE MODELS

When we say that an image model is *handcrafted*, it means that such an image model is not constructed by means of optimization. Rather, at least for models mentioned here, we rely on the change of pixel intensity in a local region of an image. The fact that we consider a local region of an image introduces another categorization in image models. Some models, given a raw image, yield a variable-sized set of fixed-sized vectors that describe, i.e., *represent*, such an image. These fixed-sized vectors are called *local descriptors* as they describe significant local regions in an image – *local features*. On the other side, we have a single vector/matrix/tensor that describes an image as a whole – *a global descriptor*. There



(a) (b) (c)

Figure 3.1: The aperture problem. (a) For a location in a textureless area, it is not clear how to find the corresponding location in another image. (b) It is possible to align two patches of an edge by moving in the direction of edge normals. This indicates that an edge is a better option for localization. (c) The localization of a patch is uniquely determined when the change occurs in at least two substantially different directions, and such a patch is called a corner. Images taken from [37].

exist scenarios where it is appropriate to create a global descriptor from a set of local descriptors. Conversely, a global descriptor can be partitioned into a set of local descriptors too. Ultimately, for sequence-based visual place recognition methods, global descriptors of images are used.

3.1.1 *Local features and descriptors*

When compared to its spatial neighborhood, i.e., its neighboring pixels, a local feature is a sufficiently distinctive region in an image. It depends from image to image, but in most scenarios, there are "few" such regions, i.e., fewer local features than the number of pixels in an image, so local features are frequently mentioned alongside the adjective "sparse". There are multiple use cases where local features are used. In this chapter, local features, i.e., their corresponding descriptors, are *aggregated* together so that global image representations for images are obtained. In the upcoming chapter, they contribute to *visual odometry* [33, 34, 35]. There is a wide array of other usages, e.g., for *image stitching and alignment* [36].

These traits lead to the formula that suggests if a spatial position in an image holds an appropriate feature. Let $I(x, y) = I(\mathbf{x})$ denote a pixel intensity at position $(x, y) = \mathbf{x}$. A specific position $\mathbf{x}_i$ in an image I_0, along with its corresponding local neighborhood, i.e., a patch centered in $\mathbf{x}_i$, would not be recognizable at all in an image I_1 if no change in intensity occurs no matter what direction we move from $\mathbf{x}_i$. It is also said that such a feature cannot be localized. This problem is called *the aperture problem* and is depicted in Figure3.1a. A more discriminative patch would be an edge region because the change in pixel intensities occurs alongside the edge in directions perpendicular to the edge – *edge normals*. Therefore, it is possible to align two patches of an edge by moving in the direction of edge normals (Figure3.1b). The localization of a patch is, however, not uniquely determined as long as the change does not occur in at least two substantially different directions, as it the case in Figure3.1c. For such feature, also called *a corner* for obvious reasons, there exists a deterministic vector $\mathbf{u} = (u, v)$ that translates the feature between two images that contain

it.

Given images I_0 and I_1 that presumptively hold a mutual region, we can quantize whether that region is a corner, i.e., whether there is a unique displacement $\mathbf{u}$, by evaluating the weighted sum of squared differences

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w(\mathbf{x}_i)\big(I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)\big)^2. \tag{3.1}$$

Because it is unknown how features are displaced between two images, it is computationally costly to evaluate (3.1) for each region in I_0 and for each region in I_1. Luckily, there is another way to tell if a location contains a corner by modifying (3.1). Instead of an inter-images displacement by $\mathbf{u}$, we should observe a slight displacement by $\Delta\mathbf{u} = (\Delta u, \Delta v)$ in a single image $I = I_0$. Additionally, as images are prone to noise, and noise, in the context of images, can be characterized as pixels with significantly different intensities with respect to their neighbors, i.e., noisy pixels themselves could be erroneously characterized as corners, we resort to removing such noise by smoothing an image. Image smoothing is done by *convolving* an image with *the symmetric Gaussian kernel*

$$G_\sigma(\mathbf{x}) = \frac{1}{2\pi\sigma^2}\exp\Big(-\frac{x^2 + y^2}{2\sigma^2}\Big), \tag{3.2}$$

where σ is *the standard deviation* parameter. This way, we obtain a noise-free (up to a parameter σ) and smoothed image

$$I_\sigma(\mathbf{x}) = G_\sigma * I(\mathbf{x}), \tag{3.3}$$

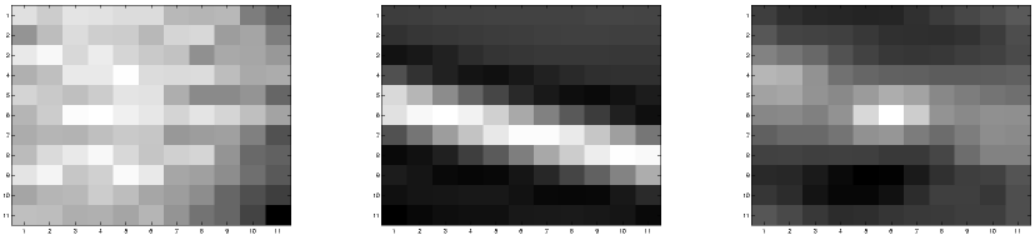
where $*$ is the convolution operation. For more details confer [38, Chapter 4, 5]. Finally, *the auto-correlation* function is defined as

$$E_{\text{AC}}(\Delta\mathbf{u}) = \sum_i \Big(I_\sigma(\mathbf{x}_i + \Delta\mathbf{u}) - I_\sigma(\mathbf{x}_i)\Big)^2. \tag{3.4}$$

When (3.4) is evaluated on a textureless region (e.g., the sky or a wall) no clear minimum is noticeable (Figure3.2a). For an edge, evaluation of the auto-correlation formula yields multiple minimal values alongside that edge (Figure3.2b). A strong single-point minimum exists for regions that contain a corner (Figure3.2c). This goes hand in hand with the aperture problem – no matter in which direction $\Delta\mathbf{u}$ we move in a textureless area, strong difference of pixel intensities does not exist. When we move for $\Delta\mathbf{u}$ perpendicular to an edge, the change is emphasized, but not alongside the edge. And finally, for a corner, i.e., a good feature, the change is significant wherever we move.

In practice, the evaluation of (3.4) is further simplified by approximating it with the second-order Taylor expansion around $\mathbf{0}$ because $\Delta\mathbf{u} \to \mathbf{0}$. The derivative of a smoothed image with respect to $\Delta\mathbf{u}$ is the same as the derivative with respect to $\mathbf{x}$ because the components of $\Delta\mathbf{u}$ (Δu and Δv, respectively) have the same directions as the components of $\mathbf{x}$ (x and y, respectively) and therefore

$$\frac{dI_\sigma(\mathbf{x})}{d\Delta\mathbf{u}} = \frac{dI_\sigma(\mathbf{x})}{d\mathbf{x}} = \nabla I_\sigma(\mathbf{x}). \tag{3.5}$$



(a) (b) (c)

Figure 3.2: Evaluation of the auto-correlation function (3.4) on (a) a textureless area, (b) an edge, and (c) a corner. In a corner region, the auto-correlation possesses a strong local minimum. Images taken from [37].

For the sake of clarity, let $\mathbf{s} = \Delta\mathbf{u}$. Then, we approximate (3.4) as

$$E_{\mathrm{AC}}(\mathbf{s}) \approx E_{\mathrm{AC}}(\mathbf{0}) + \frac{dE_{\mathrm{AC}}(\mathbf{s})}{d\mathbf{s}}\Big|_{\mathbf{s}=\mathbf{0}}\mathbf{s} + \frac{1}{2}\mathbf{s}^T\frac{d^2E_{\mathrm{AC}}(\mathbf{s})}{d\mathbf{s}^2}\Big|_{\mathbf{s}=\mathbf{0}}\mathbf{s}. \tag{3.6}$$

It is easy to see that

$$E_{\mathrm{AC}}(\mathbf{0}) = \sum_i \big(I_\sigma(\mathbf{x}_i) - I_\sigma(\mathbf{x}_i)\big)^2 = 0, \tag{3.7}$$

$$\frac{dE_{\mathrm{AC}}(\mathbf{s})}{d\mathbf{s}}\Big|_{\mathbf{s}=\mathbf{0}} = \sum_i 2\big(I_\sigma(\mathbf{x}_i) - I_\sigma(\mathbf{x}_i)\big)\nabla I_\sigma(\mathbf{x}_i) = 0, \tag{3.8}$$

and therefore

$$E_{\mathrm{AC}}(\mathbf{s}) \approx \frac{1}{2}\mathbf{s}^T\frac{d^2E_{\mathrm{AC}}(\mathbf{s})}{d\mathbf{s}^2}\Big|_{\mathbf{s}=\mathbf{0}}\mathbf{s}. \tag{3.9}$$

By expanding and evaluating the second derivative term in (3.9), we obtain

$$\frac{d^2E_{\mathrm{AC}}(\mathbf{s})}{d\mathbf{s}^2}\Big|_{\mathbf{s}=\mathbf{0}} = \sum_i 2\nabla I_\sigma(\mathbf{x}_i)\nabla^T I_\sigma(\mathbf{x}_i) + \sum_i 2\big(I_\sigma(\mathbf{x}_i) - I_\sigma(\mathbf{x}_i)\big)\nabla^2 I_\sigma(\mathbf{x}_i) \tag{3.10}$$

$$= \sum_i 2\nabla I_\sigma(\mathbf{x}_i)\nabla^T I_\sigma(\mathbf{x}_i). \tag{3.11}$$

Finally, the approximation of (3.4) can be rewritten as

$$E_{\mathrm{AC}}(\Delta\mathbf{u}) \approx \Delta\mathbf{u}^T\Big(\sum_i \nabla I_\sigma(\mathbf{x}_i)\nabla^T I_\sigma(\mathbf{x}_i)\Big)\Delta\mathbf{u} \tag{3.12}$$

$$= \Delta\mathbf{u}^T\begin{bmatrix} \sum_i\big(\frac{\partial I_\sigma(\mathbf{x})}{\partial x}\big|_{\mathbf{x}=\mathbf{x}_i}\big)^2 & \sum_i\big(\frac{\partial I_\sigma(\mathbf{x})}{\partial x}\cdot\frac{\partial I_\sigma(\mathbf{x})}{\partial y}\big)\big|_{\mathbf{x}=\mathbf{x}_i} \\ \sum_i\big(\frac{\partial I_\sigma(\mathbf{x})}{\partial x}\cdot\frac{\partial I_\sigma(\mathbf{x})}{\partial y}\big)\big|_{\mathbf{x}=\mathbf{x}_i} & \sum_i\big(\frac{\partial I_\sigma(\mathbf{x})}{\partial y}\big|_{\mathbf{x}=\mathbf{x}_i}\big)^2 \end{bmatrix}\Delta\mathbf{u} \tag{3.13}$$

$$= \Delta\mathbf{u}^T\mathbf{A}\Delta\mathbf{u} \tag{3.14}$$

$$= \Delta\mathbf{u}^T\mathbf{R}^{-1}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\mathbf{R}\Delta\mathbf{u}, \tag{3.15}$$

while it is possible to diagonalize $\mathbf{A}$ because it is a real symmetric matrix. Its eigenvalues, λ_1 and λ_2, indicate whether the change in pixel intensities occurs in perpendicular directions. Therefore, the problem of feature detection comes down to how we interpret the eigenvalues. A broader analysis can be found in [37] and [38, Chapter 5], but the idea is that the larger

an eigenvalue is, the greater the change of pixel intensity in the direction of its respective eigenvector. We aim for the change of intensity in both directions, so if both λ_1 and λ_2 are "large enough" (there is no stricter criteria), that area is considered a feature. If λ_1 and λ_2 are both "small", no change occurs, and if only a single eigenvalue is large, then an edge is present because the smaller eigenvalue, being "small", indicates there is no change in the direction of its eigenvector.

This discussion brings us to the popular quantitative measure that tells whether a region is salient, the Harris corner detector [39], defined by

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(\mathbf{A}) - k\,\mathrm{tr}(\mathbf{A})^2, k \in \mathbb{R}^+. \tag{3.16}$$

It is easy to see that (3.16) will go over a lower-bound threshold once the eigenvalues are substantially large. There exist additional approaches to corner detection by examining eigenvalues of $\mathbf{A}$, e.g., the Shi-Tomasi corner detector [40] that is defined as

$$R = \min(\lambda_1, \lambda_2). \tag{3.17}$$

Two desirable characteristics of corner detectors are invariance to orientation and scale. Having strong gradients in different directions, thus having a corner that is being detected by just discussed corner detectors, we can rotate such corner and it will be detected once again. This, alongside being justified intuitively (no matter how a corner is rotated, it remains a corner), was justified empirically too [41]. This suggests how corner detectors that rely on (3.4) are rotation invariant. However, the same cannot be said for scale. Therefore, we resort to testing a detector at different image scales in order to achieve scale invariance too.

Before we continue, we should reflect on the property for differentiation of a smoothed image. Because differentiation is linear and *shift invariant* [38], it can be shown that

$$\frac{dI_\sigma(\mathbf{x})}{d\mathbf{x}} = \frac{d(G_\sigma * I)(\mathbf{x})}{d\mathbf{x}} = \frac{dG_\sigma(\mathbf{x})}{d\mathbf{x}} * I(\mathbf{x}). \tag{3.18}$$

This property suggests that rather than smooth and then differentiate an image, we can directly convolve it with the derivative of the Gaussian. Another option for corner detection is to convolve an image with the Laplacian of the Gaussian

$$\nabla^2 G_\sigma(\mathbf{x}) = \frac{\partial^2 G_\sigma(\mathbf{x})}{\partial x^2} + \frac{\partial^2 G_\sigma(\mathbf{x})}{\partial y^2}, \tag{3.19}$$

e.g., to obtain the Laplacian of a smoothed image as

$$\nabla^2 I_\sigma(\mathbf{x}) = \nabla^2(G_\sigma * I)(\mathbf{x}) = \nabla^2 G_\sigma * I(\mathbf{x}). \tag{3.20}$$

Several different derivative-based functions for corner detection, alongside the first derivative and the Laplacian, have been compared in [42] where the combination of the Harris corner detector for a specific scale of an image alongside the Laplacian over multiple scales for that image shown the best results. However, different authors [38] claim that the Harris corner detector, although very accurate for estimation of the center of a corner, is not that accurate for scale estimation. By contrast, the Laplacian of the Gaussian is less accurate for centers but better for scale estimation. The Laplacian of the Gaussian-based scale estimation is the foundation of the forthcoming scale-invariant feature detector.

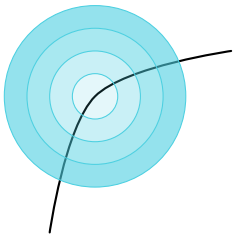


Figure 3.3: What seems to be an edge at a finer scale might be a corner at a coarser scale. In order to achieve scale invariance, we have to observe features at multiple scales.

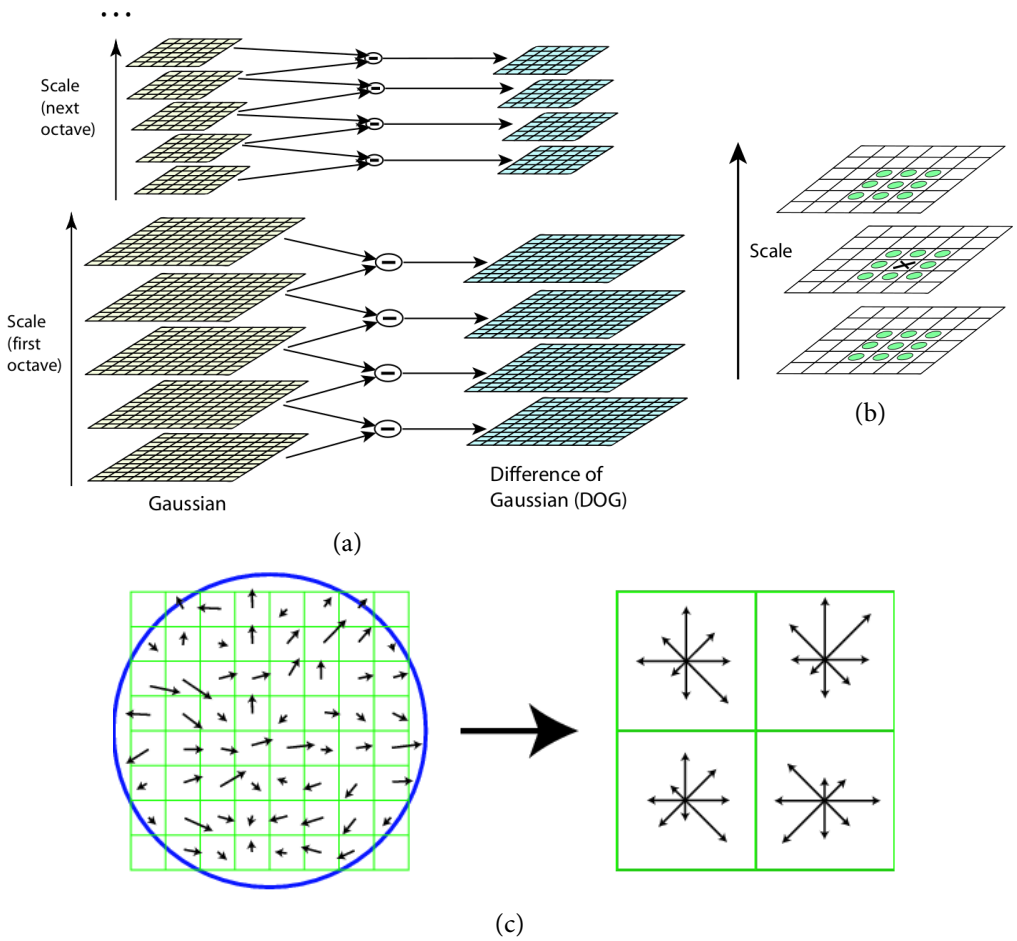


Figure 3.4: In SIFT, (a) the Laplacian of the Gaussians is approximated with the difference of Gaussians. Then, (b) a local extremum among neighboring points in scale and space is considered a feature candidate. (c) Descriptors are created by dividing a local neighborhood into smaller subblocks, then for each subblock, an 8-bin histogram of orientations is calculated, and finally, histograms are stacked together forming a descriptor. Images taken from [36].

The basic idea to achieve scale invariance is to find an optimal scale for a specific feature. In other words, to find an optimal radius of a circular area around the feature at $\mathbf{x}$ such that the response is maximized, i.e.,

$$r(\mathbf{x}) = \arg\max_{\sigma} \nabla^2 I_\sigma(\mathbf{x}) = \arg\max_{\sigma} \nabla^2 G_\sigma * I(\mathbf{x}). \tag{3.21}$$

This is illustrated in Figure3.3. What seems to be an edge at a finer scale might be a corner at

a coarser scale. Therefore, we tweak different scales in order to maximize the response. From a practical point of view, (3.21) leads us to create, given an image, an image pyramid[1] and then apply the Laplacian of the Gaussian. *The scale-invariant features* (abbr. SIFT) [6] use image pyramids for that purpose exactly. Although, for reasons of efficiency, the Laplacian of the Gaussians is approximated with *the difference of Gaussians*. In SIFT, at a specific scale, also called *an octave*, an image is blurred with Gaussian kernels of different standard deviation parameters σ, $k\sigma$, $k^2\sigma, \ldots$, and such blurred images are further subtracted in order to obtain differences of Gaussians (Figure3.4a). Among differences of Gaussians for a specific octave, *a local extremum* over scale (9 points in the previous scale and 9 points in the next scale) and space (8 neighboring points at exact scale) is considered a potential keypoint (Figure3.4b).

Once keypoint locations are found, distinctive "signatures" that describe these keypoints, descriptors, are constructed as illustrated in Figure3.4c. Say a keypoint region is a block that consists of $n \times n$ cells (16×16 in the original paper, 8×8 in Figure3.4c). This block is further divided into subblocks of size 4×4. Then, for a subblock, an 8-bins histogram that accumulates weighted gradient orientations is constructed. Histograms are then stacked together into a single vector ($4 \times 4 \times 8$-dimensional in the original paper, $2 \times 2 \times 8$ in Figure3.4c), first normalized, then thresholded, and once again normalized. This vector is a SIFT descriptor. In Figure3.6, a subsequence of visual place recognition data is visualized with detected SIFT features, where corresponding scale and orientations obtained after histogramming are marked as oriented and variable-radius circular patches.

SURF: Speeded-up robust features [7] is another approach to achieve scale invariance where the Laplacian of Gaussian is approximated with box filters. For a given image I, the authors check whether a region located at $\mathbf{x}$ holds a feature at different scales by constructing the Hessian matrix

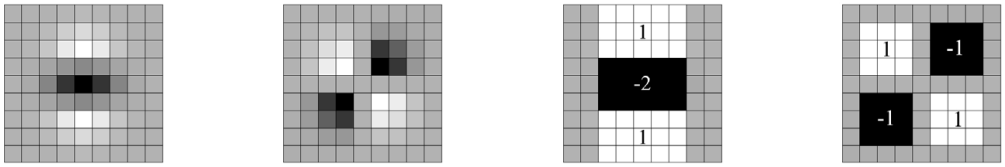
$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{3.22}$$

where

$$L_{xx}(\mathbf{x}, \sigma) = \frac{\partial^2 (G_\sigma * I)(\mathbf{x})}{\partial x^2} = \frac{\partial^2 G_\sigma}{\partial x^2} * I(\mathbf{x}) \tag{3.23}$$

is the second-order derivative of an image convolved with the Gaussian kernel with the deviation σ with respect to x. Once again, we do not have to smooth an image and then find the second-order derivatives, but we can convolve an image with the second-order derivative of the Gaussian with respect to specified directions. Analogously are defined and treated the second-order derivative with respect to y, L_{yy}, and the mixed derivative with respect to x and y, L_{xy}. The second-order derivatives of Gaussians, first being cropped and discretized (Figures3.5a,3.5b), are approximated with simpler and faster alternatives – box filters (Figures3.5c,3.5d). In a similar manner to SIFT descriptors, SURF descriptors are created by accumulating orientations that are obtained with Haar wavelets [43].

[1] A sequence that consists of a single image being blurred and downsampled multiple times, usually by a factor of 2.



(a) (b) (c) (d)

Figure 3.5: Discretized and cropped second-order derivatives of the Gaussian with $\sigma = 1.2$ with respect to (a) y^2 and (b) x and y. In SURF, (a) and (b) are approximated with box filters shown in (c) and (d). Images taken from [7].

3.1.2 *Global descriptors*

Besides it is possible to extract local features and calculate corresponding descriptors, i.e., to represent an image I with a set of local descriptors, we can also opt for a global vector/matrix/tensor representation. Among handcrafted image models that yield global descriptors, in our experiments, we have tried those approaches that map a set of local features into a global descriptor by using *visual words*. Also, we tried out an approach that splits an image into subregions that are characterized by gradients – *histogram of oriented gradients*.

AGGREGATING LOCAL FEATURES INTO GLOBAL DESCRIPTORS.It would be inefficient to compare each local descriptor from one image to each local descriptor in another image. Rather, there are mechanisms that map an indefinite-sized set of local descriptors into a definite-sized global descriptor, be it either a vector, a matrix, or a tensor. Then, by means of matrix operations, global descriptors of a definite size can be easily compared. One of the key aspects of local features aggregation is to build *a vocabulary* of *words*. This is the main idea in *the bag-of-words* model – a general discretization model used in different studies including *natural language processing*, *information retrieval*, and also in computer vision [8]. In computer vision, words are prefixed with the adjective *visual*, so local features are quantized by a set of visual words. Technically, visual words are centers of a clustered partition of a dataset that consists of a large number of local descriptors, so let us first describe the problem of cluster analysis – *k-means clustering* – and how it corresponds to the context of two global descriptor models: bag-of-words (abbr. BoW) and vector of locally aggregated descriptors (abbr. VLAD).

Let $\mathcal{I} \subseteq \mathbb{R}^n$ be a set of n-dimensional features. In k-means clustering, we should split $\mathcal{I}$ into k disjoint subsets, *clusters*, so that elements inside a specific cluster are as similar as possible, and also, elements of two different clusters are as different as possible. As defined by [44], let $\Pi \in \mathcal{P}(\mathcal{I}, k)$ be a k-partition[2] of the set $\mathcal{I}$ where $\mathcal{P}(\mathcal{I}, k)$ is the set of all k-

[2] Let $\mathcal{A}$ be a set of $m \geq 2$ elements. A partition of the set $\mathcal{A}$ into $1 \leq k \leq m$ disjoint nonempty subsets $\pi_1, \ldots, \pi_k$ such that

(i) $\bigcup_{j=1}^{k} \pi_j = \mathcal{A}$,

(ii) $\pi_r \cap \pi_s = \varnothing, r \neq s$,

(iii) $|\pi_j| \geq 1, \forall j = 1, \ldots, k$

Algorithm 6 Lloyd's algorithm

Input: Dataset $\mathcal{I}$, number of clusters k
Output: Optimal k-partition Π^*
Initialize $c_j, \forall\, j = 1, \ldots, k$ by picking a random element from $\mathcal{I}$
repeat
  for each c_j **do**
    Reassign each element from $\mathcal{I}$ to its closest center
    Recalculate c_j as the arithmetic mean of elements assigned
  end for
until convergence

partitions for $\mathcal{I}$. Given a distance function $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ to each cluster $\pi_j \in \Pi$, one can assign its center c_j in the following way:

$$c_j \in \arg\min_{x \in \mathbb{R}^n} \sum_{l \in \pi_j} d(x, l), \;\; \forall\, j = 1, \ldots, k. \tag{3.24}$$

Then we search for the optimal k-partition Π^* by solving the following global optimization problem:

$$\Pi^* \in \arg\min_{\Pi \in \mathcal{P}(\mathcal{I}, k)} \sum_{j=1}^{k} \sum_{l \in \pi_j} d(c_j, l). \tag{3.25}$$

Classic algorithms for solving (3.25) are the Lloyd's algorithm [45] (Algorithm6) and the quite similar MacQueen's algorithm [46], while Yinyang K-means [47] is a state-of-the-art approach with the GPU-ready library [48] that has also been used in our implementation.

In the context of computer vision, $\mathcal{I}$ will hold a large number of local descriptors (be it either SIFT descriptors, SURF descriptors, or any other local descriptors) from a large set of images. Once (3.25) is solved, obtained values $c_j, \forall\, j = 1, \ldots, k$ as defined in (3.24), will represent k different visual words. Let $\mathcal{C} = \{c_j : j = 1, \ldots, k\}$ denote a set of all visual words, also called a vocabulary. We build a vocabulary in advance, and once it is built, for a given image I, we create its global BoW or VLAD descriptor.

↬ BAG OF WORDS.Given a vocabulary $\mathcal{C}$, an image I is mapped to a bag-of-words global descriptor $z_I^{\text{BoW}} \in \mathbb{R}^{|\mathcal{C}|}$ first by extracting local descriptors $\mathcal{L}_I = \{l_{I,i} : i = 1, \ldots, n_I\} \subset \mathbb{R}^d$ of I with a local feature detector and descriptors of choice. Then, for each cluster π_j we determine which local descriptors belong to this cluster, i.e., for each local descriptor $l_{I,i}$ we determine its closest visual word in terms of a distance function d (the Euclidean distance in our case). The number of corresponding local descriptors for a visual word c_j will be the j-th component of z_I^{BoW}. Additionally, we normalize z_I^{BoW}. All these steps are given in Algorithm7. It is also possible to vectorize the given pseudocode, as we did in our code, by copying each local descriptor as many times as there are visual words, also by copying each visual words as many times as there are local descriptors, and then in a single matrix operation to calculate residuals, then norms, etc.

is called a k-partition of the set $\mathcal{A}$ [44].

Algorithm 7 Bag-of-words image model

Input: Vocabulary $\mathcal{C}$, an image I
Output: Global descriptor z_I^{BoW}
Initialize $z_I^{\text{BoW}} \in \mathbb{R}^k$ with zeros
Get $\mathcal{L}_I$ by using a local descriptors extractor
for each $l_{I,i} \in \mathcal{L}_I$ **do**
    Find the nearest visual word c_j
    $z_{I,j}^{\text{BoW}} \leftarrow z_{I,j}^{\text{BoW}} + 1$
end for
$z_I^{\text{BoW}} \leftarrow z_I^{\text{BoW}} / \| z_I^{\text{BoW}} \|_2$

The advantage of using bag-of-words as an image model is that resulting descriptors are only k-dimensional; therefore, it is blazingly fast to compare them with each other. However, the bag-of-words model, due to its dimensionality, has a limited capacity to represent complex image data. Notice how local descriptors themselves are not encoded in a global descriptor. Rather, z_I^{BoW} only bears the information about the occurrences of visual words; therefore, a considerable amount of information is lost. As follows, there is an approach designed to resolve this issue.

↬ VECTOR OF LOCALLY AGGREGATED DESCRIPTORS.Vector of locally aggregated descriptors [9] is a local descriptors aggregation approach where, instead of frequencies, residuals for each visual word and its corresponding local descriptors are encoded. As in the bag-of-words model, a vocabulary of visual words $\mathcal{C}$ should be provided in advance, and local descriptors $\mathcal{L}_I$ for a given image I should be extracted first. The result is a global descriptor $z_I^{\text{VLAD}} \in \mathbb{R}^{k \times d}$ where the j-th row is defined as

$$z_I^{\text{VLAD}}[j, :] = \sum_{l_{I,i} \in \mathcal{L}_i} \mathbb{1}\{l_{I,i} \in \pi_j\}(l_{I,i} - c_j). \tag{3.26}$$

In the end, we additionally flatten z_I^{VLAD} into a single-axis $k \cdot d$-dimensional vector and normalize it. The procedure that maps an image I into z_I^{VLAD} is given in Algorithm8. This algorithm can also be vectorized in the same fashion BoW is vectorized.

Notice how both BoW and VLAD are pretty similar image models, up to the way local descriptors are encoded. For a given image, VLAD descriptor bears more information than its BoW counterpart – being $k \cdot d$-dimensional, instead of k-dimensional, VLAD is a more capacitated image model what has a positive impact on quantitative results. Nevertheless, there is a severe shortcoming – once local features are detected and descriptors extracted, locations of features are no further used; thus, spatial information is lost. This is also the case for BoW. To overcome this shortcoming, we do not have to use local descriptors at all and then aggregate them, but instead, we can encode every bit of information in an image in its respective local histogram with a known spatial position as it is done in *histogram of oriented gradients*.

Algorithm 8 Vector of locally aggregated descriptors

Input: Vocabulary $\mathcal{C}$, an image I
Output: Global descriptor z_I^{VLAD}
Initialize $z_I^{\text{VLAD}} \in \mathbb{R}^{k \times d}$ with zeros
Get $\mathcal{L}_I$ by using a local descriptors extractor
for each $l_{I,i} \in \mathcal{L}_I$ **do**
    Find the nearest visual word c_j
    $z_{I,(j,:)}^{\text{VLAD}} \leftarrow z_{I,(j,:)}^{\text{VLAD}} + l_{I,i} - c_j$
end for
$z_I^{\text{VLAD}} = \text{flatten}(z_I^{\text{VLAD}})$
$z_I^{\text{VLAD}} \leftarrow z_I^{\text{VLAD}} / \| z_I^{\text{VLAD}} \|_2$

↬ HISTOGRAM OF ORIENTED GRADIENTS.Histogram of oriented gradients [49] (abbr. HOG) is an image model that relies on image gradients in the way other local features discussed above do. However, in contrast to the mentioned local features, this model produces a dense map of features distributed uniformly on a grid that divides an image into rectangular or circular regions. HOG has been used for human detection providing a compact representation of an image further used in machine learning tasks [50]. By quantizing image gradients, not only compactness is achieved, but also robustness to noise.

To obtain a global HOG descriptor z_I^{HOG}, a given image I is first preprocessed as described in [51]. Then, it is divided into $n \times n$-pixel cells (e.g., 8×8 as in [50]). Horizontal and vertical gradients $g_{\mathbf{x},x}$ and $g_{\mathbf{x},y}$ for a pixel at position $\mathbf{x}$ are calculated by using simple gradient operators $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$. Then, for each pixel, its gradient magnitude $m_{\mathbf{x}}$ and gradient orientation $\theta_{\mathbf{x}}$ are calculated as:

$$m_{\mathbf{x}} = \sqrt{g_{\mathbf{x},x}^2 + g_{\mathbf{x},y}^2}, \tag{3.27}$$

$$\theta_{\mathbf{x}} = \arctan \frac{g_{\mathbf{x},y}}{g_{\mathbf{x},x}}. \tag{3.28}$$

For each cell, a histogram with m bins that counts weighted orientations is initialized. Orientations are either unsigned (take values from $[0°, 180°)$) or signed (take values from $[0°, 360°)$). It was shown empirically that unsigned orientations with 9-bin histograms work best. Each orientation $\theta_{\mathbf{x}}$ in a cell is assigned to an appropriate bin and is scaled by its accompanying magnitude $m_{\mathbf{x}}$. Histograms are additionally normalized, not individually, but according to another hyperparameter – *the block size*. The block size of 2×2 cells, i.e., 4 histograms stacked together and then normalized, achieved the best results. Finally, all histograms stacked together constitute a global descriptor. How histogram of oriented gradients responds to a subsequence of images visual place recognition data is visualized in the second column of Figure3.6. Notice how, when containing an edge, histogram orientations are mostly accumulated into bins perpendicular to that edge. Likewise, those cells positioned in textureless areas contain histograms without dominant orientations and strong magnitudes.

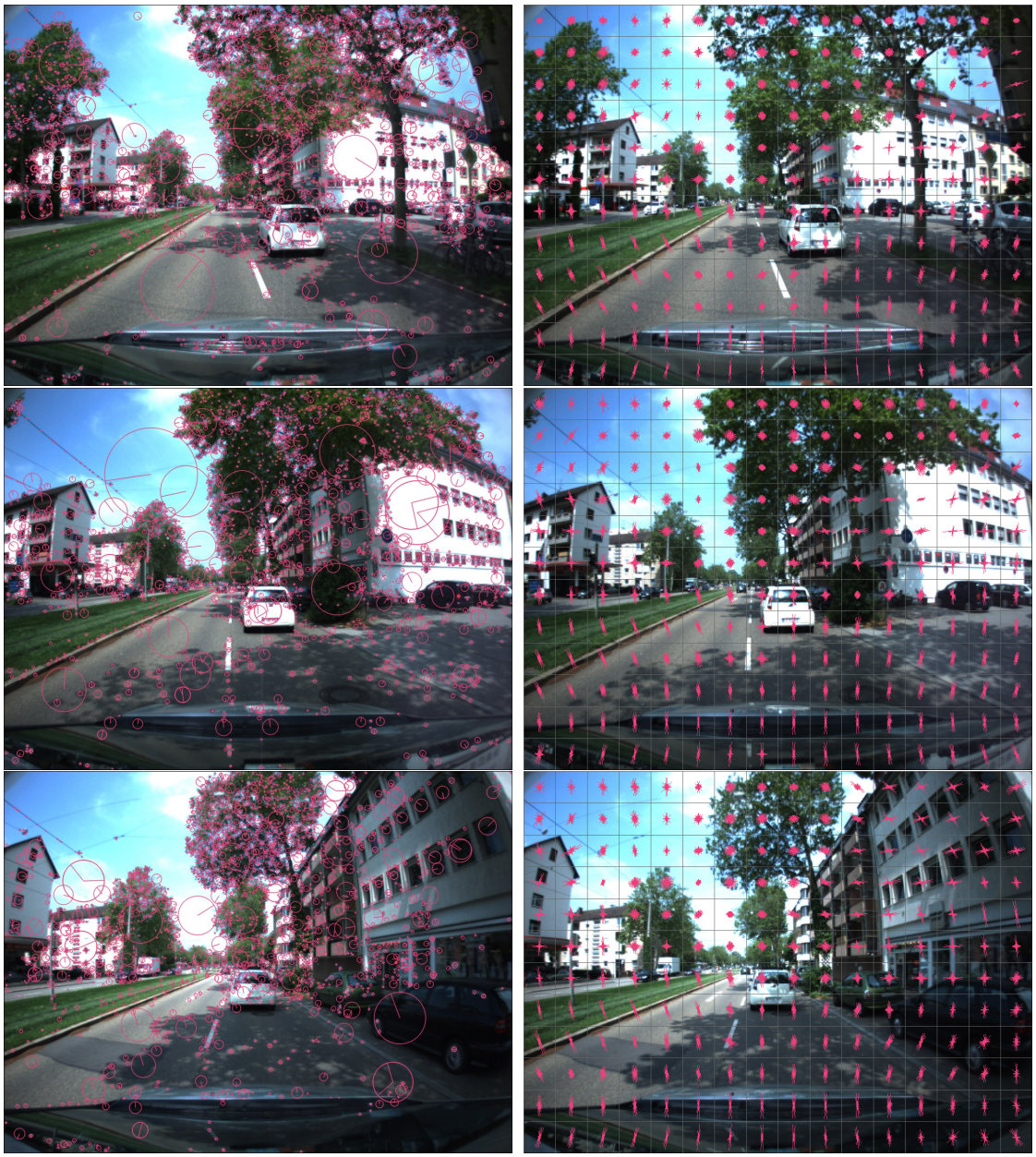


Figure 3.6: Visualization of (left column) SIFT local descriptors and (right column) HoG global descriptors on a subsequence of visual place recognition data (the Freiburg dataset [52]).

3.2 Learned image models

Learned image models are obtained by solving an optimization criterium on a large amount of data. On a regular basis, learned image models outperform handcrafted models. There exist learned image models that detect corners, such as *features from accelerated segment test* (abbr. FAST) [53, 54], but in recent years, *neural networks* are pretty much the unique method of choice, not only for images but also for text, audio and video data.

3.2.1 *Neural networks*

Neural networks (abbr. NN) are biologically inspired systems that, according to a specified task, map high-dimensional inputs, e.g., images, audio, or video files, into either

low-dimensional or high-dimensional outputs. Neural networks initially intended to model neurons, but since then "[…] become a matter of engineering […]" that achieve good results in machine learning [55]. A neuron is the basic computational unit of the brain connected with other neurons. In the process of *neurotransmission*, impulses are exchanged between neurons via *synapses*. A neuron receives multiple impulses from other neurons, its *inputs*, via *dendrites*. Then, these multiple impulses are "processed" together in the neuron's body and further transmitted to the *axon*. The axon splits up into multiple branches of axon terminals so that a single neuron output can be used as an input for other neurons. This is depicted in Figure3.7a.

This biological process is simulated with the simplest neural network, also the basic building blocks for more complex neural networks – the composition of a nonlinear activation function $f : \mathbb{R} \to \mathbb{R}$ applied component-wise to an output of an *affine map* $g : \mathbb{R}^n \to \mathbb{R}^m$ that, for a given input $x \in \mathbb{R}^n$, is defined as

$$g(x) = Wx + b, \tag{3.29}$$

where $W \in R^{m \times n}$ and $b \in \mathbb{R}^m$. In Figure3.7bit is shown how

$$y = (f \circ g)(x) \tag{3.30}$$

relates to neurotransmission. Figuratively, multiple input *cells*/*nodes* that bear information, i.e., the components $x[i]$ of a vector x, are glued together into a new output cell $y[j]$ that is a component of y defined with (3.30).

In the context of neural networks, the combination of a nonlinear activation function and (3.29) is also called *a single-layer perceptron/a single-layer feedforward network*. Notice how we can easily generalize single-layer perceptron, simultaneously increasing the representational capacity of a model, by composing multiple affine transformations and nonlinear functions alternately and this way obtain *a multi-layer perceptron* (abbr. MLP) defined as

$$\mathrm{MLP}(x; W_1, b_1, W_2, b_2, \ldots, W_k, b_k) = f_k(W_k(\ldots f_2(W_2 f_1(W_1 x + b_1) + b_2)\ldots) + b_k). \tag{3.31}$$

The arguments after ; denote *parameters*/*weights* of a model rather than input data, and we should optimize (3.31) with respect to these learnable parameters. Composing multiple layers of affine and nonlinear functions together is what makes a neural network *deep*, and therefore the name *deep neural networks*. Those layers stacked between the input and the output layer are called *hidden layers*. We can design a neural network, i.e., its *topology*, whatever suits us – the input dimensionality, the output dimensionality, the number of hidden layers and their dimensionality, a specific nonlinear activation function, etc. Two examples of neural network topologies are shown in Figure3.9. Stacking all weights together into a tensor $\mathbf{W}$, let us denote a neural network as a function $f_{\mathrm{NN}}(\,\cdot\,; \mathbf{W}) : \mathbb{R}^n \to \mathbb{R}^m$.

Deep neural networks, i.e., multi-layer perceptrons, are especially powerful because there exists a theoretical guarantee, *the universal approximation theorem*, presented in the paper "Multilayer feedforward networks are universal approximators" [56] whose name speaks for itself. According to this theorem, there exists a neural network with a linear output layer and at least one hidden layer that can approximate any continuous function on

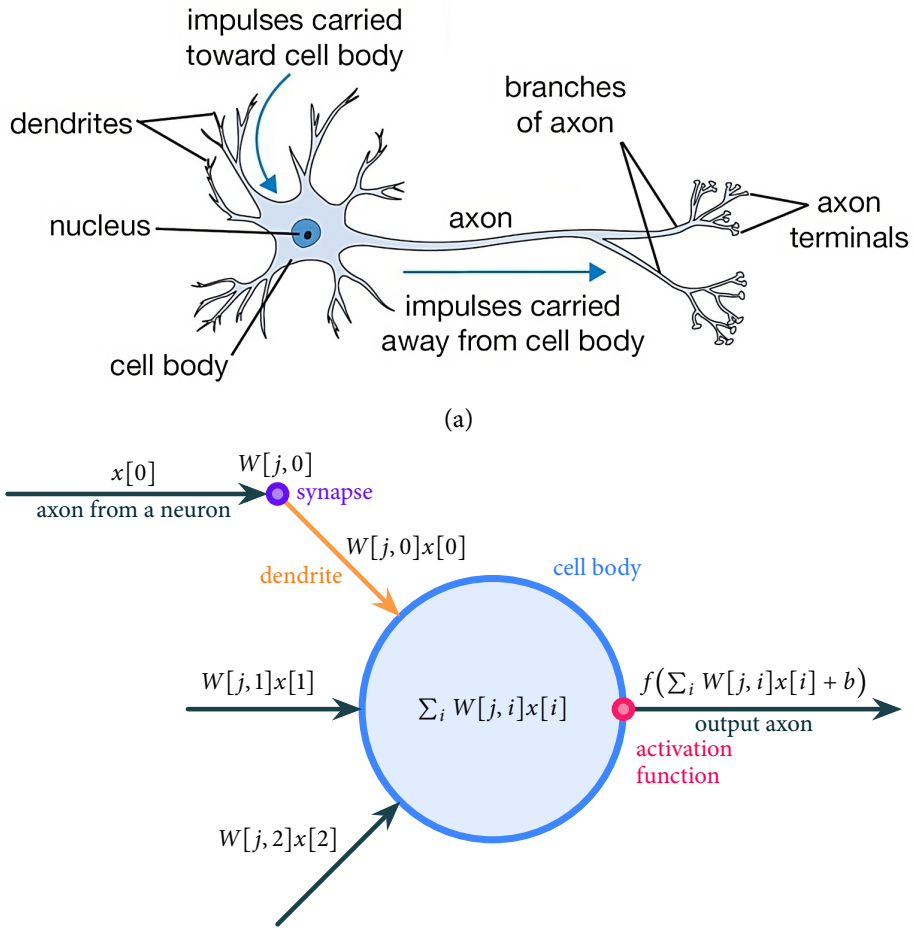


Figure 3.7: (a) A neuron receives multiple inputs via its dendrites and produces a single output being used as an input in multiple other neurons. (b) This is mimicked with the composition of a nonlinear activation function and an affine map. Image (a) taken from [55] and image (b) based on an image from [55].

a closed and bounded subset of $\mathbb{R}^n$. Also, it is not known how many parameters a model should have in order to approximate a desired function, but if we stay on a single hidden layer, such a model could be infeasibly large and hard to train. Even if we manage to train it, it may be prone to *overfitting* [57, p. 195]. Rather than "wide" and single-layer, we prefer deep, multi-layer models that are easier to train, have significantly less parameters, and *generalize* well.

If we deliberately omit nonlinear activation functions $f_1, f_2, \ldots, f_k$ from (3.31), the composition of affine functions would be an affine function, and such model would not be able to account for *nonlinearities* of complex data. This justifies why a nonlinear activation is used. Among popular choices for nonlinear activation is *the rectified linear unit* (abbr. ReLU) defined as

$$\mathrm{ReLU}(x) = \max\{0, x\}. \tag{3.32}$$

It is the default recommendation for a nonlinear activation function as claimed in [57, p. 169]: "Because rectified linear units are nearly linear, they preserve many properties that make linear models easy to optimize with gradient-based methods." According to [55],

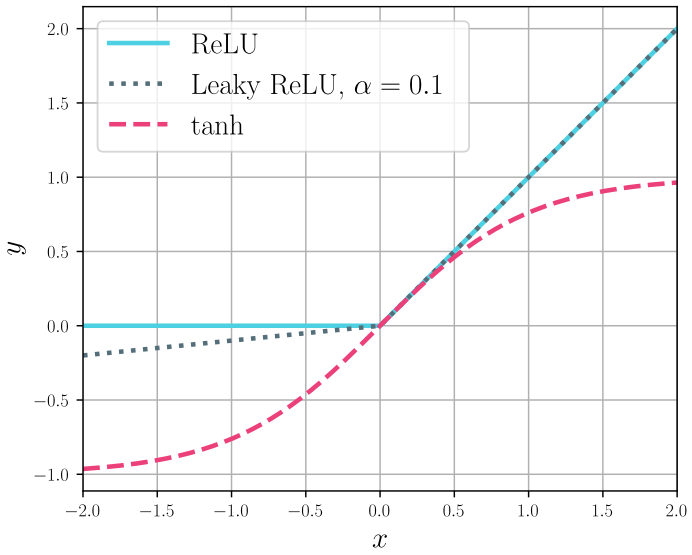
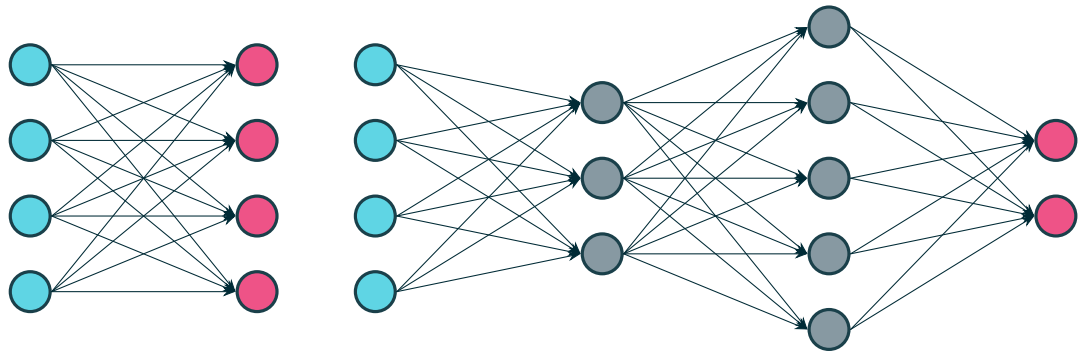


Figure 3.8: Various nonlinear activation functions used in learned image models in order to account for nonlinearities in data.

advisable alternatives to ReLU include Leaky ReLU [58] and tanh. How these functions map an input is shown in Figure3.8.



(a) (b)

Figure 3.9: (a) The example of a single-layer perceptron that maps an input $x \in \mathbb{R}^4$ (cyan nodes) into an output $y \in \mathbb{R}^4$ (red nodes). (b) The example of a multi-layer perceptron that maps $x \in \mathbb{R}^4$ into $y \in \mathbb{R}^2$ via two hidden layers (gray nodes).

3.2.2 *Optimization of neural networks*

In classic optimization, optimization is a goal in itself. In machine learning, which includes neural networks too, the ultimate goal would be to learn a model so that it works satisfactorily on unseen test data. Therefore, we optimize, i.e., we *train* a neural network by using data available in advance – training data. Although the optimization of neural networks is not a convex optimization problem, when we train a network, training error should diminish, and, in most scenarios, it is not that hard to achieve this objective. Much harder is to attain a low test error – due to the representational capacity of a model, depending mostly on the number of parameters and their configuration, it takes a lot of data to achieve satisfactory test performances.

Let $L(x_i, y_i, \mathbf{W})$ denote *the per-example loss*, a function that dictates how a network f_{NN} should be optimized given an input x_i, its corresponding label y_i and the network parameters $\mathbf{W}$. It is usually defined as the negative log-likelihood

$$L(x_i, y_i, \mathbf{W}) = -\log p(y_i \,|\, x_i, \mathbf{W}) \tag{3.33}$$

where $p(y \,|\, x, \mathbf{W})$ is the conditional probability that models an output y given an input x and the weights $\mathbf{W}$. The optimization of $f_{\text{NN}}(\cdot, \mathbf{W})$ on a large number of training examples $\{x_i\}$ with corresponding labels $\{y_i\}$ according to a specified criteria/loss L can be formulated as

$$\min_{\mathbf{W}} \mathbb{E}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})] = \min_{\mathbf{W}} \frac{1}{m} \sum_{i=1}^{m} L(x_i, y_i, \mathbf{W}). \tag{3.34}$$

We can conduct the optimization defined with (3.34) by using *the gradient descent*, an optimization method that will update $\mathbf{W}$ in the following way:

$$\mathbf{W} \leftarrow \mathbf{W} - \epsilon \nabla \mathbf{E}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})],\ \epsilon \in \mathbb{R}^+, \tag{3.35}$$

where

$$\nabla \mathbf{E}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})] = \frac{1}{m} \sum_{i=1}^{m} \nabla L(x_i, y_i, \mathbf{W}). \tag{3.36}$$

Although it is easy to formulate training of neural networks in a mathematical notation, evaluating (3.36) would take up too much time and memory. Instead, with *the stochastic gradient descent method* (abbr. SGD), we update the weights "piece by piece" on a subset $\{x_{i'}\}$, uniformly sampled from $\{x_i\}$, where $|\{x_{i'}\}| = m' << m = |\{x_i\}|$. In other words, we approximate (3.36) with

$$\nabla \tilde{\mathbf{E}}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})] = \frac{1}{m'} \sum_{i'} \nabla L(x_{i'}, y_{i'}, \mathbf{W}) \tag{3.37}$$

and perform the update step as

$$\mathbf{W} \leftarrow \mathbf{W} - \epsilon \nabla \tilde{\mathbf{E}}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})]. \tag{3.38}$$

as given in Algorithm9. Additional improvements of stochastic gradient descent exist. Among others, the momentum algorithm [59] is used in this chapter, where the update of weights is defined as a moving average, i.e.,

$$v \leftarrow \alpha v - \frac{\epsilon}{m'} \sum_{i'} \nabla L(x_{i'}, y_{i'}, \mathbf{W}),\ \alpha \in \mathbb{R}^+ \tag{3.39}$$

$$\mathbf{W} \leftarrow \mathbf{W} + v. \tag{3.40}$$

There are also methods that adapt learning rates. For a more comprehensive overview of approaches that optimize neural networks, see [57, Chapter 8].

As already mentioned at the beginning of this subsection, it is not that hard to optimize a network given training data. Actually, it is hard to achieve *generalization*, i.e., a low test error given test data. To an extent counterintuitive, this objective is achieved by a process called *regularization*, where we deliberately increase the training error in favor of decreasing the test error. A classic approach to regularization is *parameter norm penalty* such as L^1 and

Algorithm 9 Stochastic gradient descent

Input: Training data $\{(x_i, y_i)\}$, learning rate ϵ, initialized weights $\mathbf{W}$
Output: Optimized weights $\mathbf{W}^*$
repeat
  Uniformly sample $\{(x_{i'}, y_{i'})\} \subset \{(x_i, y_i)\}$
  Compute $\nabla\tilde{\mathbf{E}}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})]$ with (3.37)
  $\mathbf{W} \leftarrow \mathbf{W} - \epsilon\nabla\tilde{\mathbf{E}}_{(x,y)\sim p_{\text{data}}}[L(x, y, \mathbf{W})]$
until convergence

L^2 regularization. The idea is to add a regularization term $\|\mathbf{W}\|_1$ or $\|\mathbf{W}\|_2$ to the objective function (3.34) in order to keep the weights close to zero (more about this topic in Section3.4). A recently introduced regularization technique, dropout [60], is a simple yet powerful way to regularize a net such that connections between nodes are randomly removed with the probability p. This way, the weights are adapted to work on an ensamble [57, p. 252] of truncated networks. In other words, during training, we sample a sub-network from the full one [55]. Additionally, regularization can be achieved with adversarial examples [61] (more about this topic in Subsubsection3.2.4). For a more comprehensive overview of approaches that regularize neural networks, see [57, Chapter 7].

3.2.3 *Convolutional neural networks*

Convolutional neural networks (abbr. CNNs) are neural networks where affine maps are replaced with convolutions. This proved to be a winning ticket in the computer vision field, and over the last few years, convolutional neural networks have achieved state-of-the-art results, starting with a notable architecture AlexNet [62]. Convolutional neural networks are used for *classification* [62, 63, 64, 65], *semantic segmentation* [66, 67, 68], *object detection* [69, 70, 71], etc. The central concept of convolutional neural networks is to convolve a discrete image with a discrete convolution kernel, i.e.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.41}$$

$$= (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{3.42}$$

although it is also possible to apply *the cross-correlation*

$$S = (I \star K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.43}$$

as some deep learning libraries do[3], "which is the same as convolution but without flipping the kernel" [57, p. 324]. It does not matter which operation we are going to pick as long as we stick to that one. The visualization of convolving a 3×3 kernel K over 5×5 image I that results in an output S can be seen in Figure3.10. Alongside affine maps that are appended atop of convolutional layers mostly to perform classification tasks, in convolutional neural networks,

[3] In PyTorch [72], classes `torch.nn.Conv1d` and `torch.nn.Conv2d` apply the cross-correlation operator over an input signal.

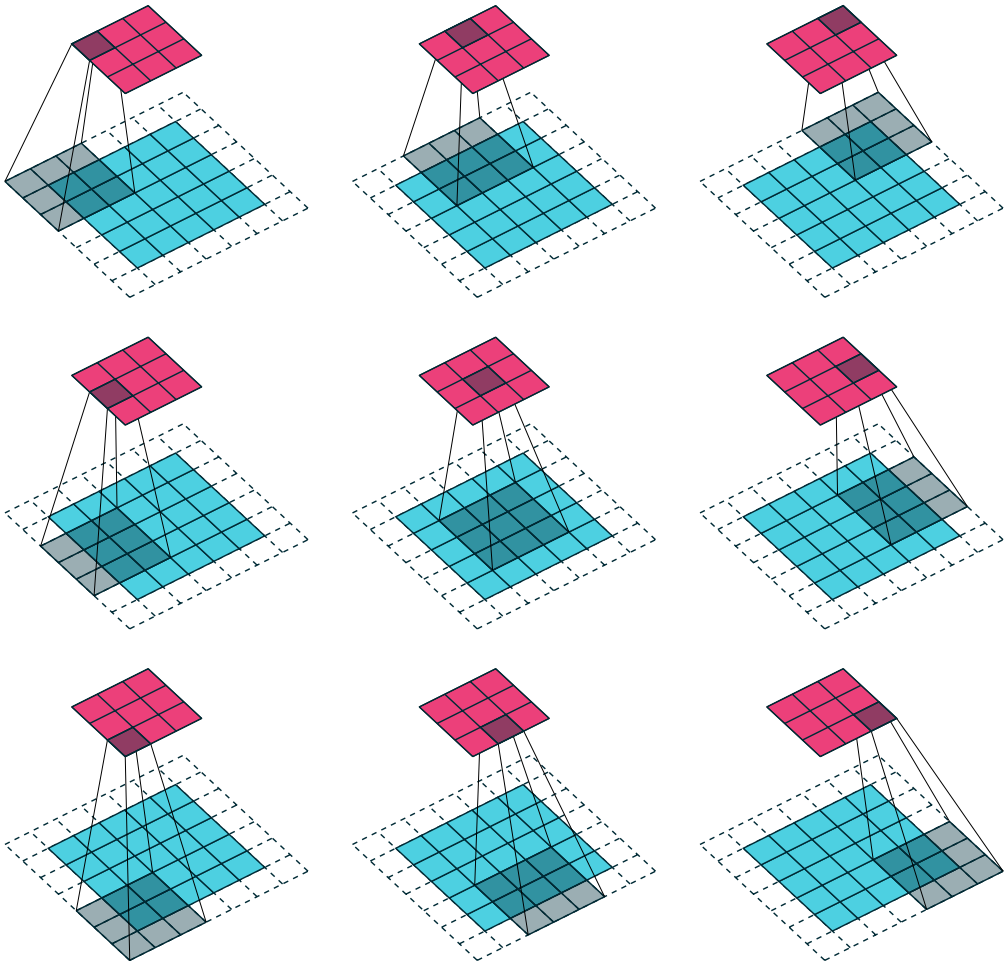


Figure 3.10: Convolution of a 5×5 image I (cyan grid) with a 3×3 kernel K(transparent gray grid) with padding of 1 pixel and stride of 2 pixels that results in a new output S (red grid). Images created with [73].

weights are mostly convolutional kernels. If a network has affine layers parametrized with $\{(W_i, b_i)\}$ and kernels $\{K_j\}$ then, once again, we can denote all such weights with $\mathbf{W}$ and denote the network as a function $f_{\text{CNN}}(\cdot, \mathbf{W})$. According to [57], concepts that make convolutional neural networks stand out among neural networks are

- *sparse interactions*,
- *parameter sharing* and
- *equivariant representations*.

All these qualities are implications of using a convolutional operation rather than matrix multiplication.

Sparse interactions refer to the way network nodes are connected. First, let us focus on a node from its input perspective (be it an input node/a hidden layer node). In a non-convolutional deep neural network, a node would participate as an information transferor (i.e., in combination with the corresponding weights, it would be an addend in the scalar product) for each node in a successive layer. The same reasoning applies to the output

perspective of a hidden layer node/an output node – such node summarises information of a few, rather than all, nodes from a previous layer. Dense connections of such neural networks result in an unnecessarily large number of paths from input to output nodes (Figure3.11a). In contrast to ordinary affine-map-based neural networks, interactions in convolutional neural networks are sparse, meaning just a few nodes transfer information to a successive layer's node. We should carefully design a convolutional neural network so that there are just enough connections to propagate information from each input cell to each output cell (Figure3.11b), i.e., to maintain an optimal *receptive field*.

A specific matrix cell has been used only once when calculating the output for the next layer in a non-convolutional neural network. By using convolutional kernels, a single kernel is spatially applied to designated input cells repeatedly. This concept is known as parameter sharing and results in a reduced number of network weights. Closely related to this is the concept of equivariant representation. Say we are convolving with a kernel that detects edges. As the kernel is applied throughout the entire spatial region, an edge can be detected anywhere. More formally, let f denote a convolution applied to an image I and let g denote a function that spatially translates I. It is said that f is equivariant to g if $(f \circ g)(I) = (g \circ f)(I)$. Therefore, if we spatially move a specific visual entity, "[...] its representation will move the same amount in the output." [57, p. 330]

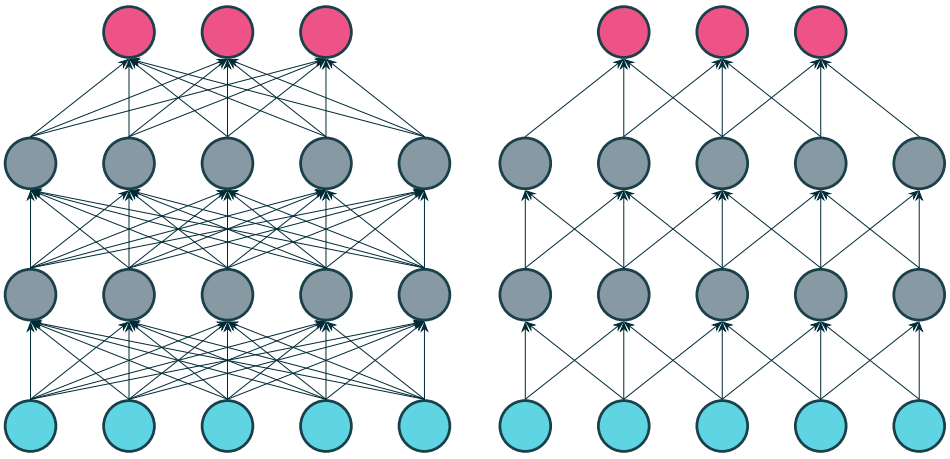


Figure 3.11: (a) In ordinary non-convolutional neural networks, connections are dense. This results in an unnecessarily large number of paths from input to output nodes. (b) In convolutional neural networks, connections are sparse. Notice how there is just enough connections to propagate an information from each input cell to each output cell.

Alongside the convolution and the activation function, an additional operation used in convolutional neural networks is called *pooling*. Pooling is a fancy name for taking a descriptive statistic of a spatial region, mostly substantially smaller in its size than a spatial region being affected by a kernel. A popular pooling choice is *max pooling*, i.e., selecting the maximum value in a region (usually a few cells, e.g., 3×3) made out of successive spatial convolutions and activations. Another popular choice is *average pooling*, where we are averaging a region. It is clear that, according to these use cases, pooling is used to downsample an output. Another, not that straightforward but useful, implication of applying

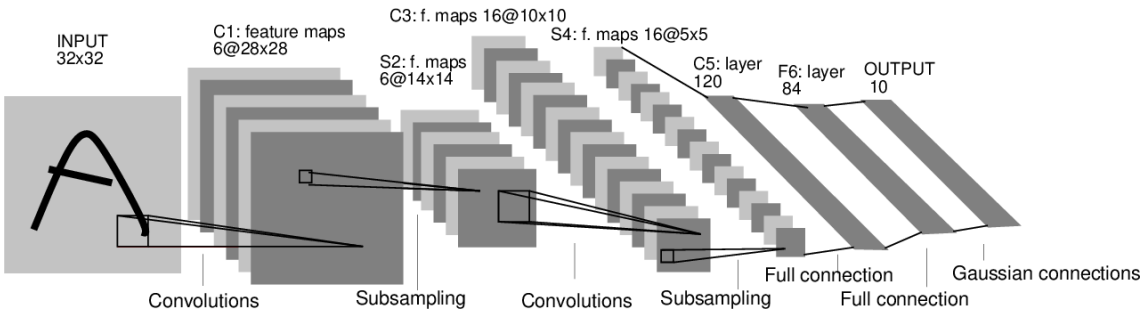


Figure 3.12: The LeNet-5 architecture. Image taken from [63].

a pooling operation is *invariance to local translation* [57, p. 331] – as we are taking descriptive statistic of a region, say we average it into a single pixel, small shifts will introduce small number of new values and the result should not deviate that much. These three components – a convolution layer, a nonlinear activation layer, and a pooling layer – constitute the basic building block of a convolutional neural network. Then we stack multiple such blocks in order to map a "grid-like" input into a "stick-like" output.

Among additional "bells and whistles" that enhance convolutional neural networks are *padding* and *stride*. Padding is a concept inherited from classic computer vision where, convolving an image $n \times n$ with a convolution of size $m \times m$ where $m = 2k + 1, k \in \mathbb{N}$, the result will have the dimension of $(n - m + 1) \times (n - m + 1)$. So we deliberately "surround" an image with zeros around it (there are other possible values too [74]) in order to keep an input and an output equally sized, e.g., in Figure3.10we add a padding of $p = 1$ zero pixels (dashed transparent rectangles around cyan rectangles). Then again, we can deliberately downsample an output size by introducing the number of steps between centers of two successive convolutions s, e.g., the stride $s = 2$ used in Figure3.10.

NOTABLE CNN ARCHITECTURES.One of the pioneering convolutional neural networks is LeNet-5 [63] used for *optical character recognition* (abbr. OCR) where handwritten digits were classified. Its architecture is shown in Figure3.12. Since then, things did not change that much – more recent convolutional neural networks used for classification also have similar topologies, i.e., they have several convolutional layers followed by a few fully-connected layers.

A major breakthrough came in 2012 when AlexNet [62] won *The ImageNet Large Scale Visual Recognition Challenge* (abbr. ILSVRC) [75] becoming the first deep convolutional neural network to do so. AlexNet achieved *a top-5 error rate* of 16.4%, more than 10% less then the second-best image model. Among the factors responsible for success of AlexNet are, first its topology that consists of 5 convolutional and 3 fully connected layers with approximately 60 million parameters what results in a sufficient representational capacity. Also, ReLU has been used as an activation function because, as reported by the AlexNet creators, "networks with ReLUs consistently learn several times faster than equivalents with saturating neurons"[62]. Additionally, dropout [60] has been used as a regularization technique.

The second-best contestant of ILSVRC 2014, the VGG network architecture [76] took a huge leap in terms of top-5 error by achieving an error rate of 7.3%. The idea behind the

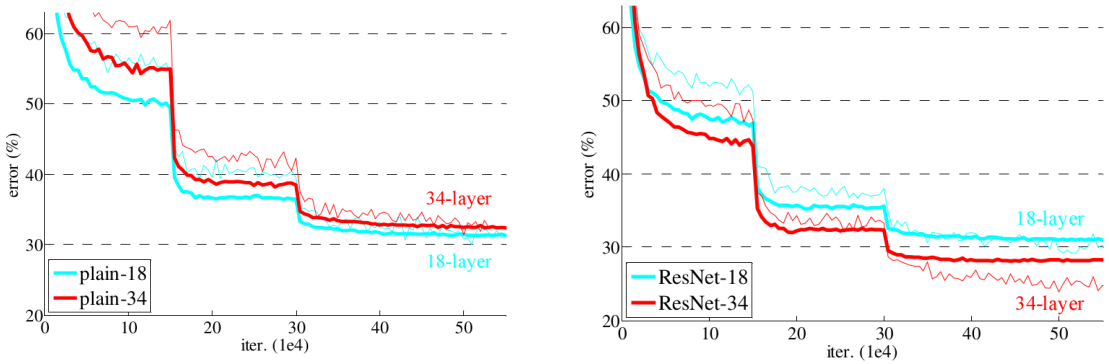


Figure 3.13: (a) The deeper a network is, the more it takes to learn it, i.e., the same number of epochs yields smaller training errors for shallow models and, implicitly, smaller test errors. (b) By exploiting residual learning, as ResNet models do, the degradation problem is no longer a concern. Images taken from [65].

VGG network was to increase the number of layers, either 16 (VGG16) or 19 (VGG19). In order to restrain the number of parameters, only 3×3 convolutions are applied with the padding $p = 1$ and the stride $s = 1$. It was substantiated that a stack of three 3×3 convolutions has the same effective receptive field as a single 7×7 convolution, which results in a smaller number of parameters – for a three 3×3 convolutions, it is $3 \cdot 3 \cdot 3 = 27$ parameters in contrast to $7 \cdot 7 = 49$ parameters for a 7×7 convolution. Still, the total number of parameters for the smaller variant of VGG, VGG16, is approximately 138 million parameters which is pretty large. The 22 layers deep GoogLeNet [64] has convincingly won the first place in ILSVRC 2014 with a top-5 error rate of 6.7% by having 5 million parameters only. The credit belongs to an effective block this network is composed of – *the inception module*.

Naturally, deeper models have better representational capacity due to a larger number of parameters, but then again, the deeper a model is, the harder it is to optimize it (Figure3.13a). Shallow models, as visible in this figure, are easier to optimize and implicitly, for the same number of epochs, mostly have smaller test errors too. In order to resolve this negative phenomenon, called *the degradation problem*, the authors of the ResNet architecture hypothesized that "if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart." [65] If $H(x)$ is a "complicated" mapping a network should learn, then this network is also capable to learn the residual of that mapping and an input, i.e., to learn $F(x) = H(x) - x$. ResNet is composed of multiple such blocks that learn residuals. Identity mappings in layers were achieved with the utilization of *skip connections* (Figure3.14). Then, by increasing the number of layers, there is no concern that a deeper network will be harder to train, and as expected, better test performances will be achieved. (Figure3.13b). The 152 layers deep ResNet is the winner of ILSVRC 2015 with a top-5 error of an astounding 3.57%.

CNNs in visual place recognition. In recent years, *feature maps* extracted from deep convolutional neural networks have been used in favor of handcrafted image models in visual place recognition. In [77] authors concluded that feature maps extracted from the 3rd and 4th convolutional layers of the AlexNet architecture [62] perform well,

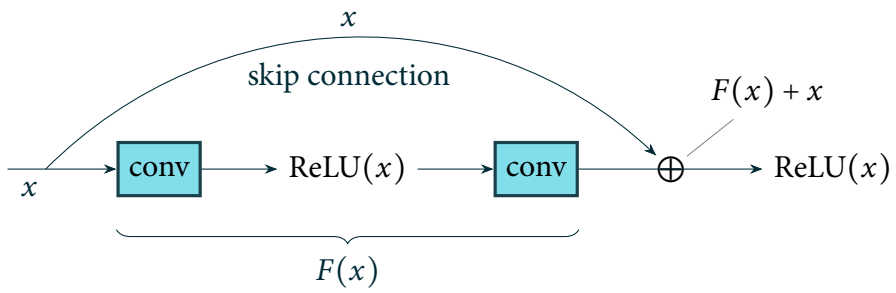


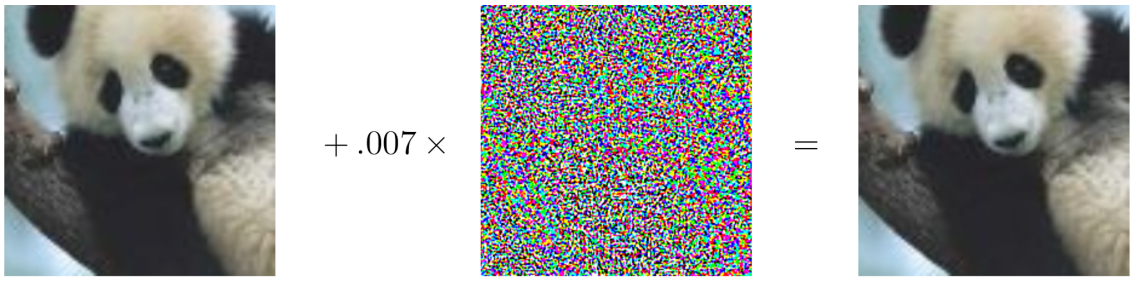
Figure 3.14: The diagram of the residual block used in the ResNet architecture [65]. Skip connection acts as an identity map of an input x, so than a residual $F(x) = H(x) - x$ can be learned.

while additionally fine-tuned architecture on Places205 dataset [78] achieves even better performance. Other architectures, such as OverFeat [79] and GoogLeNet [64], were used as feature maps extractors too [52, 80, 22]. In NetVLAD [81], a differentiable variant of VLAD [9], authors replaced the indicator function with the softmax function in order to perform *end-to-end* training for place recognition by minimizing the *triplet loss*. Similarly, in [82] authors used the *contrastive loss* function in order to optimize ResNet [65] and VGG [76] architectures. In a similar fashion to NetVLAD, LoST [17] aggregates semantic classes in an image by using semantic segmentation. *Object proposal* system was used in [83] in order to extract more significant regions that are later forward-propagated through a neural network. A compact neural network from [84] was proposed especially for the purpose of visual place recognition where atop of it a *recurrent neural network* layer was attached since it is capable of capturing *temporal information* (i.e., local neighborhood of a place). In [85], a large dataset for place recognition is proposed, and the HybridNet architecture is obtained by performing softmax regression.

3.2.4 *Adversities in neural networks*

Adversarial examples are deceitfully constructed in order to be misclassified. This is achieved by adding a carefully designed noise to an uncorrupted image. To a human, that noise added to the original image seems just like every other image noise. The sum of the original image to be corrupted and noise – *an adversarial example* – seems almost identical to the original, such that a human can not tell the difference between them (Figure3.15). Another disturbing fact is that, under an adversarial attack, a network can be really confident about its predictions on adversarial examples as shown in Figure3.15.

The technical detail of pixels being represented with values from 0 to 255, and at the same time, neural networks expect inputs to be represented usually in the 32-bit floating point representation, leaves the door open for an adversary. By adding a tiny shift to each component of a 32-bit representation of the input, no larger than some $\epsilon > 0$, the outcome, when converted back to 0–255 pixel intensities, possess invisible or barely visible changes [55, Lecture 16]. The idea of constructing an adversarial example $\tilde{x}$ is, given an original input x with the corresponding label y, to find a perturbation $\Delta x = \tilde{x} - x$ so that the per-example loss $L(\tilde{x}, y, \mathbf{W})$ is maximized. By approximating $L(\tilde{x}, y, \mathbf{W})$ with the first-order Taylor



$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 3.15: Adversarial examples are deceitfully constructed in order to be misclassified. To a human, the adversarially-perturbed image may seem indistinguishable from the original. Image taken from [61].

expansion of $L(\tilde{x}, y, \mathbf{W})$ around x, i.e.,

$$L(\tilde{x}, y, \mathbf{W}) \approx L(x, y, \mathbf{W}) + (\tilde{x} - x)^T \nabla_x L(x, y, \mathbf{W}), \tag{3.44}$$

we obtain the objective function of the constrained optimization problem

$$\max_{\tilde{x}} L(x, y, \mathbf{W}) + (\tilde{x} - x)^T \nabla_x L(x, y, \mathbf{W}), \tag{3.45}$$

$$\text{s.t. } \|\tilde{x} - x\|_\infty \leq \epsilon. \tag{3.46}$$

We add the constraint (3.46) because this is exactly what we want with each component of the input – to perturb it to a maximal extent of ϵ – so the perturbation is not visible to a human eye. The solution to (3.45), called *the fast gradient sign method* (abbr. FGSM)[61], is

$$\tilde{x} = x + \epsilon \,\text{sign}\, \nabla_x L(x, y, \mathbf{W}). \tag{3.47}$$

It was shown in [61] that training with the combination of the conventional and "adversarially-perturbed" per-example loss

$$\alpha L(x, y, \mathbf{W}) + (1 - \alpha) L(x + \epsilon \,\text{sign}\, \nabla_x L(x, y, \mathbf{W}), y, \mathbf{W}), \alpha \in [0, 1], \tag{3.48}$$

was an effective regularizer - we can regard training with adversarial examples, *adversarial training*, as a data augmentation technique that will make networks resistant to adversities.

Formally, adversarial training of a neural network $f_{\text{NN}}(\,\cdot\,; \mathbf{W})$ with the per-example loss $L(x, y, \mathbf{W})$ can be formulated as *the saddle point problem*

$$\min_{\mathbf{W}} \mathbb{E}_{(x,y)\sim p_{\text{data}}} \Big[\max_{\Delta x \in \mathcal{S}} L(x + \Delta x, y, \mathbf{W}) \Big] \tag{3.49}$$

where $\mathcal{S} \subseteq \mathbb{R}^n$ is a region of allowed perturbations. In [86], a multi-step and more powerful variant of (3.47), called *projected gradient descent* (abbr. PGD), is defined as

$$x^{t+1} = \Pi_{x+\mathcal{S}}\big(x^t + \alpha \,\text{sign}\, \nabla_x L(x, y, \mathbf{W})\big), \tag{3.50}$$

Algorithm 10 Interpolated adversarial training

Input: Training data $\{(x_i, y_i)\}$, learning rate ϵ, initialized weights $\mathbf{W}$
Output: Optimal weights $\mathbf{W}^*$
repeat
  Sample $\{((x_i, y_i), (x_j, y_j))\}$ from $\{(x_i, y_i)\} \times \{(x_i, y_i)\}$
  Compute $\tilde{x}_i$ and $\tilde{x}_j$ with (3.47) or (3.50)
  Compute $x_{i,j}$ and $\tilde{x}_{i,j}$ with (3.51)
  Compute $L_{\text{int.}}(x_{i,j}, y_i, y_j, \mathbf{W})$ and $L_{\text{int.}}(\tilde{x}_{i,j}, y_i, y_j, \mathbf{W})$ with (3.52)
  $\mathbf{W} \leftarrow \mathbf{W} - \epsilon\nabla\frac{L(x_{i,j},y_i,y_j,\mathbf{W})+L(\tilde{x}_{i,j},y_i,y_j,\mathbf{W})}{2}$
until convergence

where $\Pi_{x+\mathcal{S}}$ is a projection operator that keeps the output within the region $\mathcal{S}$ around the original input $x^0 := x$. The same authors report how both (3.47) and (3.50) "[...] can be viewed as specific attempts to solve the inner maximization problem in (3.49)" [86]. Also, it was reported that the generalization performance of a network trained with (3.48) is reduced for non-adversarial data, i.e., after additionally performing adversarial training, a standard test error would increase.

To tackle this issue, in [87], the authors propose *the interpolated adversarial training*, a mix of interpolated and adversarial training. The idea is to blend the loss on non-adversarial interpolated data with the loss on adversarially-perturbed interpolated data and optimize weights according to that combination of losses. Interpolation is achieved with *the mixup technique*[88] where we randomly draw samples $(x_i, y_i) \sim p_{\text{data}}$ and $(x_j, y_j) \sim p_{\text{data}}$. Then, the interpolated input $x_{i,j}$, i.e., the linear combination in input space, is defined as

$$x_{i,j} = \lambda x_i + (1 - \lambda)x_j, \lambda \sim B(\alpha, \beta), \tag{3.51}$$

where B denotes the Beta probability distribution parametrized with $\alpha, \beta \in \mathbb{R}^+$. Additionally, the per-example linear interpolation loss for a specified per-example loss L is defined as

$$L_{\text{int.}}(x_{i,j}, y_i, y_j, \mathbf{W}) = \lambda L(x_{i,j}, y_i, \mathbf{W}) + (1 - \lambda)L(x_{i,j}, y_j, \mathbf{W}), \lambda \sim B(\alpha, \beta). \tag{3.52}$$

The adversarially-perturbed data $(\tilde{x}_i, y_i)$ and $(\tilde{x}_j, y_j)$ (with $\tilde{x}_i$ and $\tilde{x}_j$ obtained either with (3.47) or (3.50)) is interpolated analogously into the interpolated input $\tilde{x}_{i,j}$ with the corresponding interpolated loss $L_{\text{int.}}(\tilde{x}_{i,j}, y_i, y_j, \mathbf{W})$. The interpolated adversarial training is given in Algorithm10. In the experimental evaluation of the forthcoming section, we perform numerous adversarial trainings, either with PGD-perturbed examples or with the interpolated adversarial training to see how visual place recognition performances are affected, i.e., whether performances for such trained networks improved on non-adversarial inputs and how adversarially-perturbed data affect the results.

3.3 Enhancing visual place recognition on sequential data with softmax regression

In this section, we propose to use softmax regression for visual place recognition on sequentially captured images. This enables us to adapt the image representation even further and

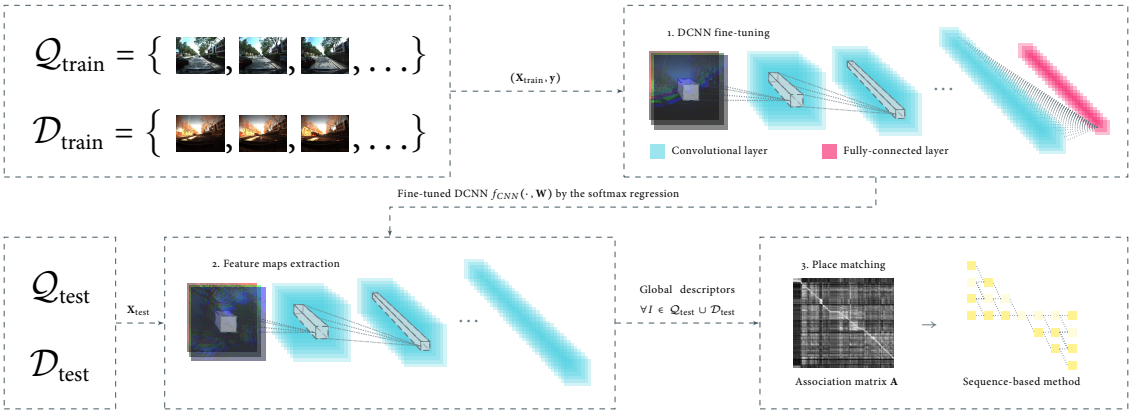


Figure 3.16: The pipeline of the proposed visual place recognition enhanced with the softmax regression. Images from $\mathcal{Q}_{\text{train}}$ and $\mathcal{D}_{\text{train}}$ are preprocessed and $(\mathbf{X}_{\text{train}}, \mathbf{y})$ is obtained. Then, softmax regression training is performed in order to fine-tune $f_{\text{CNN}}(\cdot, \mathbf{W})$. Such fine-tuned network is used as a feature maps extractor for test data $\mathcal{Q}_{\text{test}}$ and $\mathcal{D}_{\text{test}}$. Extracted feature maps, used as global descriptors of images, are further used in a place matching method.

with it to increase the similarity between images representing the same place, and vice-versa, increase the dissimilarity with other images. The pipeline of our approach to visual place recognition enhanced with the softmax regression is depicted in Fig.3.16. For training, preprocessed images from $\mathcal{Q}_{\text{train}}$ and $\mathcal{D}_{\text{train}}$ are coupled together into an appropriate data format for the softmax regression, which is then performed in order to fine-tune a deep convolutional neural network. We experimented with the most commonly used DCNN architectures – AlexNet [62], VGG16 [76], and ResNet50 [65] – all previously trained for visual tasks. In terms of quantitative performance, AlexNet and VGG16 did not perform as well as ResNet50 – we believe that previously trained AlexNet and VGG16 were not a good starting point for additional fine-tuning and optimization got stuck in a local minimum. Nevertheless, the formulation in following paragraphs is generic and can be applied on an arbitrary architecture.

3.3.1 *Softmax regression*

Softmax regression is the generalization of binomial logistic regression. In binomial logistic regression, we aim to obtain the probability distribution

$$p(y_i \,|\, x_i; \boldsymbol{\theta}), \ \boldsymbol{\theta} \in \mathbb{R}^n \tag{3.53}$$

that models whether $x_i \in \mathbb{R}^n$ belongs to a specific category ($y_i = 1$) or does not ($y_i = 0$). It is achieved by using the dot product $\boldsymbol{\theta}^T x_i$ combined with the sigmoid function $\sigma : \mathbb{R} \to [0, 1]$ defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.54}$$

Thus, we can define (3.53) as

$$p(y_i \,|\, x_i; \boldsymbol{\theta}) = (\sigma(\boldsymbol{\theta}^T x_i))^{y_i} (1 - \sigma(\boldsymbol{\theta}^T x_i))^{1 - y_i}. \tag{3.55}$$

Given a set of *independent and identically distributed* (i.i.d.) data [57] whose elements are stacked as rows in the matrix $\mathbf{X} = [x_1{}^T \dots x_m{}^T]^T \in \mathbb{R}^{m \times n}$ and accompanying labels $\mathbf{y} = [y_1 \dots y_m]^T \in \mathbb{R}^{m \times 1}$, we find the optimal $\boldsymbol{\theta}^*$ by using *the maximum log-likelihood* estimation

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p(y_i \mid x_i; \boldsymbol{\theta}) \tag{3.56}$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p(y_i \mid x_i; \boldsymbol{\theta}). \tag{3.57}$$

In contrast to the binomial case where either $y = 0$ or $y = 1$, in multinomial case we have $|\mathcal{C}|$ different classes $\mathcal{C} = \{1, \dots, |\mathcal{C}|\}$. Samples $x_i, \forall i$ remain n-dimensional, while labels $y_i, \forall i$ are $|\mathcal{C}|$-dimensional one-hot vectors. $\boldsymbol{\theta}$ is replaced with $\boldsymbol{\Theta} \in \mathbb{R}^{|\mathcal{C}| \times n}$, while the multinomial variant of (3.54) uses the softmax function $\boldsymbol{\sigma} : \mathbb{R}^{|\mathcal{C}|} \to \mathbb{R}^{|\mathcal{C}|}$ defined as

$$\boldsymbol{\sigma}(x)[c] = \frac{e^{x[c]}}{\sum\limits_{j \in \mathcal{C}} e^{x[j]}}, \ \forall c \in \mathcal{C} \tag{3.58}$$

that has probability properties, i.e., $\boldsymbol{\sigma}(x)[c] \geq 0, \forall c \in \mathcal{C}$ and $\sum\limits_{c \in \mathcal{C}} \boldsymbol{\sigma}(x)[c] = 1, \forall x$. The parametric probability distribution is then defined as

$$p(y_i \mid x_i; \boldsymbol{\Theta}) = \prod_{c \in \mathcal{C}} (\boldsymbol{\sigma}(\boldsymbol{\Theta} \cdot x_i)[c])^{y_i[c]}, \tag{3.59}$$

while the maximum log-likelihood estimation is defined as

$$\boldsymbol{\Theta}^* = \arg\max_{\boldsymbol{\Theta}} \prod_{i=1}^{m} p(y_i \mid x_i; \boldsymbol{\Theta}) \tag{3.60}$$

$$= \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} \log p(y_i \mid x_i; \boldsymbol{\Theta}) \tag{3.61}$$

$$= \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} \log \prod_{c \in \mathcal{C}} (\boldsymbol{\sigma}(\boldsymbol{\Theta} \cdot x_i)[c])^{y_i[c]} \tag{3.62}$$

$$= \arg\max_{\boldsymbol{\Theta}} \sum_{i=1}^{m} \sum_{c \in \mathcal{C}} y_i[c] \log \boldsymbol{\sigma}(\boldsymbol{\Theta} \cdot x_i)[c]. \tag{3.63}$$

Both (3.56) and (3.60) are *convex optimization problems* [89], meaning there is a theoretical guarantee that optimization will converge. Although appealing due to convexity perseverance when combined with other convex functions [90], linear mappings are not capable of capturing *nonlinear manifolds* where complex data reside [91]. Hence, we can replace $\boldsymbol{\Theta} \cdot x$ in (3.59) with a nonlinear map $f(\cdot, \mathbf{W}) : \mathbb{R}^n \to \mathbb{R}^{|\mathcal{C}|}$, e.g., with a convolutional neural network f_{CNN}.

It remains to consider how to cast visual place recognition data for softmax regression. Because the relationship between data, at least in used datasets, is *many-to-many*, i.e., each $I_{q_i} \in \mathcal{Q}$ has at least one corresponding ground truth match in $\mathcal{D}$, and each I_{d_j} can be a ground truth match for multiple elements of $\mathcal{Q}$, we formulate samples $\mathbf{X} \in \mathbb{R}^{(|\mathcal{Q}|+|\mathcal{D}|) \times n}$ and labels $\mathbf{y} \in \mathbb{R}^{(|\mathcal{Q}|+|\mathcal{D}|) \times |\mathcal{D}|}$ as

$$\mathbf{X} = \left[x_{q_1} \dots x_{q_{|\mathcal{Q}|}} x_{d_1} \dots x_{d_{|\mathcal{D}|}}\right]^T, \tag{3.64}$$

$$= \begin{bmatrix} \mathbb{1}_{I_{q_1} \epsilon \mathcal{GT}(I_{d_1})} & \cdots & \mathbb{1}_{I_{q_1} \epsilon \mathcal{GT}(I_{d_{|\mathcal{D}|}})} \\ \vdots & \ddots & \vdots \\ \mathbb{1}_{I_{q_{|\mathcal{Q}|}} \epsilon \mathcal{GT}(I_{d_1})} & \cdots & \mathbb{1}_{I_{q_{|\mathcal{Q}|}} \epsilon \mathcal{GT}(I_{d_{|\mathcal{D}|}})} \\ 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}, \tag{3.65}$$

where $x_k = \mathbf{X}[k,:]^T \in \mathbb{R}^n$ is *a flattened* n-dimensional vector obtained from $W \times H \times 3$-dimensional preprocessed RGB representation of an image I_k, thus $n = W \cdot H \cdot 3$. Referring to (3.65), $\mathbf{y}[k,:]$ tells which reference images correspond to the k-th image I_k. $\mathcal{GT}(I_{d_j})$ is a set of all query images that correspond to I_{d_j}. Each image from the reference dataset corresponds solely to itself, as there is no information about images interconnection in the ground truth file, therefore $y[|\mathcal{Q}|+1 : |\mathcal{Q}|+|\mathcal{D}|,:] = I \in \mathbb{R}^{|\mathcal{D}|\times|\mathcal{D}|}$.

3.3.2 *Feature maps extraction*

After a fine-tuned convolutional neural network $f_{\text{CNN}}(\cdot, \mathbf{W})$ is obtained, fine-tuned on training data $\mathcal{Q}_{\text{train}}$ and $\mathcal{D}_{\text{train}}$, feature maps extraction takes place for test data $\mathcal{Q}_{\text{test}}$ and $\mathcal{D}_{\text{test}}$. If we think of $f_{\text{CNN}}(\cdot, \mathbf{W}) : \mathbb{R}^n \to \mathbb{R}^{|C|}$ as the composition of its convolutional layers with its fully-connected layers[4], i.e.

$$f_{\text{CNN}}(\cdot, \mathbf{W}) = f_{\text{f.c.}}(f_{\text{conv.}}(\cdot, \mathbf{W}_{\text{conv.}}), \mathbf{W}_{\text{f.c.}}), \tag{3.66}$$

then, feature maps extraction is performed as

$$z_i = f_{\text{conv.}}(x_i, \mathbf{W}_{\text{conv.}}), \forall x_i \in \mathcal{Q}_{\text{test}} \cup \mathcal{D}_{\text{test}}. \tag{3.67}$$

Such obtained feature maps are further used for place matching, and we refer to them as image representations.

3.3.3 *Experimental results*

In this subsection, we first review the datasets that we used for this evaluation. Then, we show how the training of networks has been performed. Finally, we present quantitative results measured with area under a curve (AUC) and recall at 100% precision (R@100%P) by comparing our representation, representation from [85] and handcrafted image representation used with the mentioned sequence-based place matching methods.

↬ DATASETS. The Bonn and Freiburg datasets are accompanied by publicly available implementation from [52]. Both datasets contain vehicle traversals through urban areas of eponymous cities under different environmental conditions (Fig.3.17). In the Bonn dataset, $\mathcal{Q}$ is captured on an overcast day while $\mathcal{D}$ is captured during the night. In the Freiburg dataset, $\mathcal{Q}$ is captured on a sunny day while $\mathcal{D}$ is captured on a sunny winter afternoon. Alongside illumination variation, the accumulated snow is persistent such that

[4] For the sake of clarity, *pooling* and *nonlinear* layers are not explicitly mentioned.



(a) $\mathcal{Q}$ (b) $\mathcal{D}$ (c) $\mathcal{Q}$ (d) $\mathcal{D}$

Figure 3.17: Condition variance in the Bonn dataset as (a) $\mathcal{Q}$ is captured on an overcast day and (b) $\mathcal{D}$ is captured during the night. In the Freiburg dataset (c) $\mathcal{Q}$ is captured during a sunny day while (d) $\mathcal{D}$ is captured on a sunny winter afternoon.



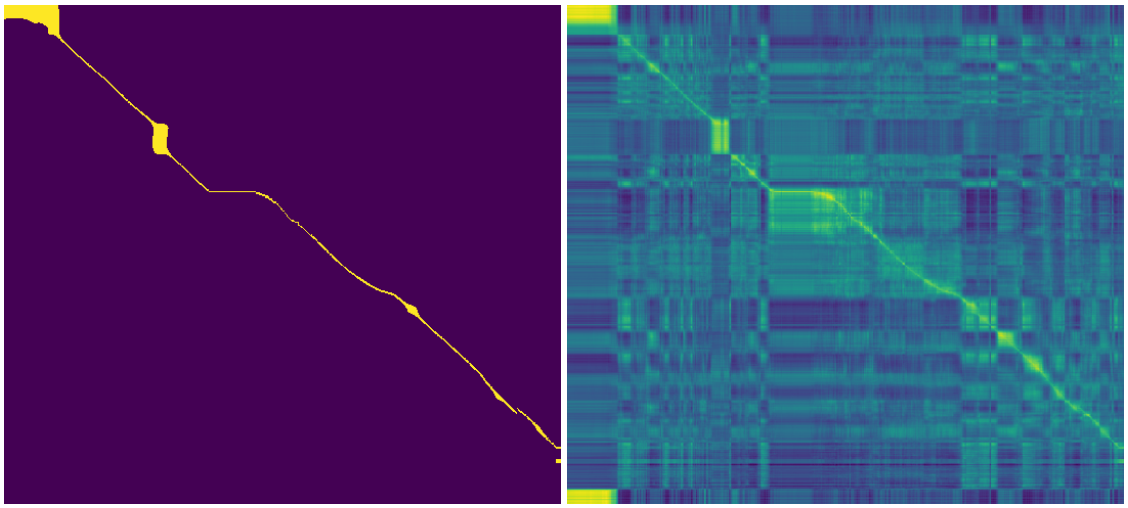
(a) (b) (c) (d)

Figure 3.18: The Nordland dataset used in visual place recognition experiments [11, 2]. The same route has been traversed during different seasons of the year.

places that are deemed to be the same are even more dissimilar in the images. Alongside serious condition variations, we observe that a slight viewpoint variance is persistent too, e.g., in Fig.3.17ca vehicle is in the right lane of a road, while in Fig.3.17dit is in the left lane. The common denominator of both datasets is that they are captured during "urban-area" traversals. Therefore, in further experiments, either the Bonn dataset is used for softmax regression training and such learned network $f_{\text{CNN}}(\cdot, \mathbf{W})$ is used as a feature-maps extractor for quantitative evaluation on the Freiburg dataset or vice versa.

Nordland is the third dataset that we have used both for training and evaluation. As it is visible in Fig.3.18, it captures the same route traversed by a train in different seasons of the year. Although viewpoint invariance does not exist as the train follows the same railway, various environmental conditions are persistent exactly because of different seasons. We picked the first 1000 places from `test/*/section1` folders of *Partitioned Nordland dataset* by [92]. For training, `spring/section1` is used as $\mathcal{Q}$ while `fall/section1` is used as $\mathcal{D}$. For evaluation, `winter/section1` is used as $\mathcal{Q}$ while `summer/section1` is used as $\mathcal{D}$. Although we can choose as many places as needed (up to the representational capacity of a convolutional neural network), we picked 1000 places deliberately as the ResNet50 [65], an architecture we fine-tune, is trained for a 1000-category classification.

↬ QUALITATIVE EVALUATION.In order to check if softmax regression is appropriate at all in the context of visual place recognition, and whether optimization will give meaningful outcomes, we conducted a simple experiment where we optimized a linear map $\mathbf{\Theta}$, instead of exhaustive-to-optimize neural networks. The result of this experiment is published in our paper [93]. Given $\mathcal{Q}$ and $\mathcal{D}$ datasets of the Bonn dataset [52], we formulated the multinomial



(a) (b)

Figure 3.19: (a) Intersection of an i-th column and a j-th row indicates whether an i-th query image matches to a j-th reference image. (b) When $\mathbf{\Theta}^* \cdot (\mathbf{X}[1:|\mathcal{Q}|, 1:n])^T$ is visualized, it truly reassembles the ground truth. Images taken from our paper [93].

classification data $\mathbf{X}$ and $\mathbf{y}$ with (3.64) and (3.65). Initially, we opted for CVXPY[5] [94] library in order to solve this problem which is a convex optimization problem [89, p.74]. In order to run a CVXPY solver at all, it is necessary for formulation to pass *the disciplined convex programming test* [90], which our formulation did. Unfortunately, even after 5 days of running and using entire 64 GB of RAM, the procedure did not converge. Then we implemented this problem in PyTorch [72] where optimal parameters were obtained with stochastic gradient descent. We conclude how *softmax regression* is an appropriate optimization problem for visual place recognition because the optimal $\mathbf{\Theta}^*$ maps the query data $(\mathbf{X}[1:|\mathcal{Q}|, 1:n])^T$ to $\mathbf{\Theta}^* \cdot (\mathbf{X}[1:|\mathcal{Q}|, 1:n])^T$ that reassembles the ground truth (Figure 3.19b).

CNN TRAINING AND FEATURES EXTRACTION. We optimized the ResNet50 architecture [65] using stochastic gradient descent on a dedicated Titan V GPU. Multiple trainings on the Bonn and Freiburg datasets, each 200 epochs long, have been performed by tweaking optimization parameters: learning rate $lr \in \{0.0005, 0.00075, 0.001, 0.002\}$, momentum $m \in \{0.9, 0.95\}$, weight decay $wd \in \{0, 0.05, 0.1, 0.15, 0.2\}$, scheduler step $s \in \{10, 20, 40\}$ and scheduler gamma $s_\gamma = 0.1$. For 500 epoch long Nordland training, we examined: learning rate $lr \in \{0.0005, 0.0001\}$, momentum $m \in \{0.9, 0.95\}$, weight decay $wd \in \{0.05, 0.1\}$, scheduler step $s \in \{20, 30\}$ and scheduler gamma $s_\gamma = 0.1$. Bonn and Freiburg images are preprocessed, both for training and evaluation, by resizing smaller size to 256 px resulting in 340×256 px images. Then, 224×224 px *center crop* from each image is taken. Images from the Partitioned Nordland dataset are preprocessed the same way beforehand. For a learned ResNet50 network, we extracted feature maps by forward propagating original images up to, and including, the 49^{th} convolutional layer of the network (`conv49`). Such feature maps represent places and are used for quantitative evaluation. Also,

[5] CVXPY is an optimization library specialized for convex optimization.

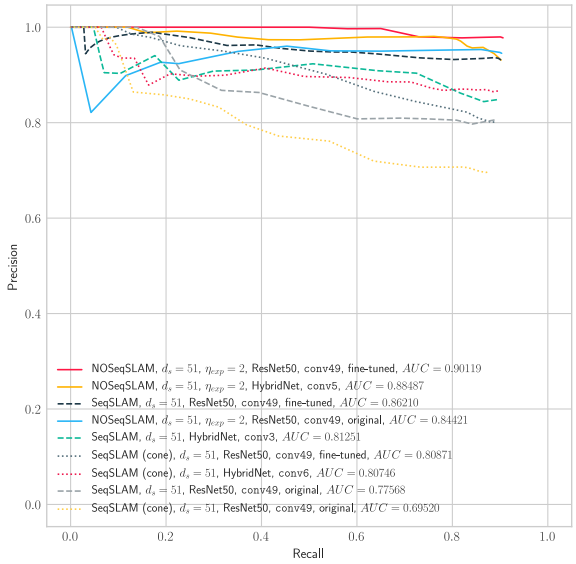
for comparative purpose, by using publicly available `Caffe` implementation, we extracted feature maps from `conv3`, `conv4`, `conv5` and `conv6` layers of HybridNet [85].

↬ LOCAL DESCRIPTORS AGGREGATION.Additionally, we extracted handcrafted SIFT and SURF descriptors (Subsection3.1.1) and aggregated them into global descriptors VLAD and BOW (Subsection3.1.2). First, for each image, all local features are extracted, and local descriptors are put into a set that is going to be clustered. We performed k-means clustering by tweaking parameter k. With this step, we created vocabularies $\text{vocabulary}_{k,l}$, $\forall k \in \{50, 100, 150, 200, 250, 300\}$, $\forall l \in \{\text{SIFT}, \text{SURF}\}$. Finally, image representations $\text{BoW}_{\text{SIFT},\,k}$, $\text{BoW}_{\text{SURF},\,k}$, $\text{VLAD}_{\text{SIFT},\,k}$ and $\text{VLAD}_{\text{SURF},\,k}$, $\forall k$ were created and compared with above-mentioned feature maps from convolutional neural networks.

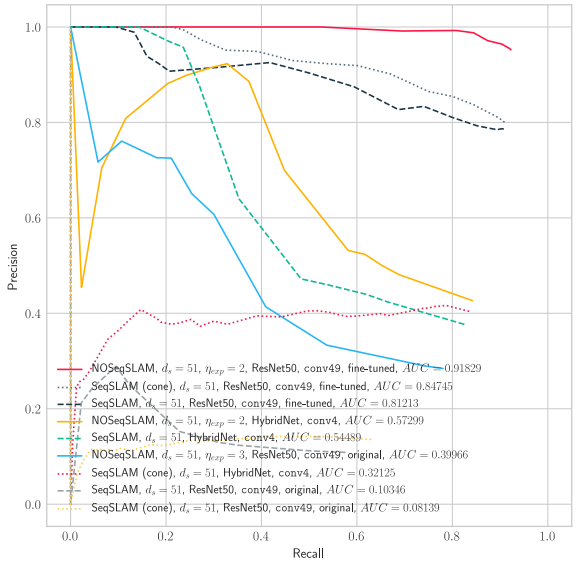
↬ QUANTITATIVE EVALUATION.Finally, we present quantitative results measured with the area under a curve (AUC) and recall at 100% precision (R@100%P) score by comparing our representation, representation from [85] and handcrafted image representation used with the sequence-based place matching methods. To assess the performance of our representations, we used the ImageNet-trained ResNet50 out of the box as a feature maps extractor, and such extracted feature maps we call *the original representation*. Thereafter, optimization was performed using softmax, and ultimately – *fine-tuned* representations adapted especially for visual place recognition were obtained. For each sequence-based algorithm in Tables3.1,3.2and3.3, we show the results of the best-performing fine-tuned ResNet50 architectures where optimization parameters are given. The original ImageNet pre-trained ResNet50 results are shown too. Additionally, the best-performing HybridNet results and best-performing aggregated global descriptors results are also presented. Place matching algorithms were evaluated for different sequence lengths $d_s \in \{31, 51\}$. Localization radius of ±3 indices is used for match proposal as in [80], while *score thresholding* [25] is used for poor matches. Accompanying *precision-recall* plots for $d_s = 51$ can be seen in Fig3.20.

Considering matching algorithms, NOSeqSLAM has the best AUC performance on both Bonn and Freiburg datasets for each sequence length. SeqSLAM follows it on the Bonn dataset, while cone-based SeqSLAM showed better performance than the trajectory-based SeqSLAM on the Freiburg dataset. Fine-tuning improved the result with respect to the original representation in all cases. Moreover, for Bonn and Freiburg, fine-tuned representations outperformed HybridNet-extracted feature maps. This is probably due to the fact that HybridNet, although fine-tuned on *a scene-centric* dataset, was not fine-tuned on road traffic scenes that were captured by a vehicle. Handcrafted features performed subpar except for the Bonn dataset; however, a fine-tuned architecture still achieved the best results. On the Nordland dataset, we can see how both HybridNet and our fine-tuned representations achieve comparable performances.

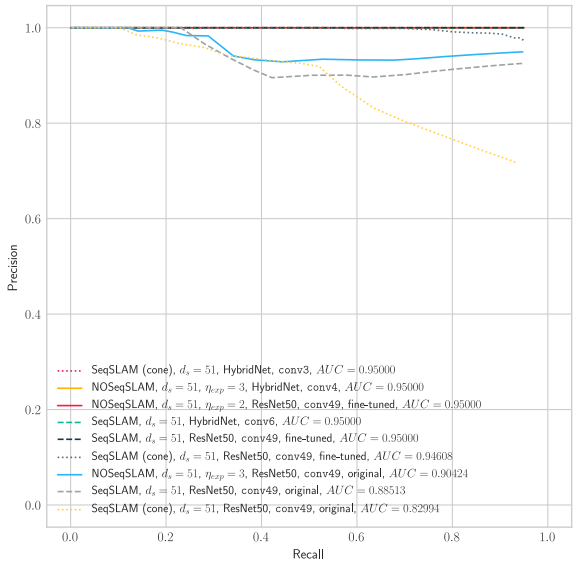
As mentioned in [23], sequence-based algorithms are unable to achieve 100% recall because the first $\lfloor \frac{d_s}{2} \rfloor$ and last $\lfloor \frac{d_s}{2} \rfloor$ query images cannot be paired. This effect can be diminished when $|\mathcal{Q}| \to \infty$. Then, a portion of unpaired queries is negligible, i.e., $2\lfloor \frac{d_s}{2} \rfloor / |\mathcal{Q}| \to 0$. Invariant to this issue is recall at 100% precision measure. Therefore, in Tables3.4,3.5and 3.6we present R@ 100%P results for the Bonn, Freiburg and Nordland datasets. For almost



(a) Bonn



(b) Freiburg



(c) Nordland

Figure 3.20: Precision recall curves for (a) the Bonn, (b) Freiburg, and (c) Nordland datasets. For each sequence-based algorithm, $d_s = 51$, the best fine-tuned representation, original representation, and HybridNet [85] representation are shown.

Table 3.1: AUC of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Bonn dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_γ	Score
31	SeqSLAM (cone)	ResNet50 (conv49)	–	0.71955
	SeqSLAM	ResNet50 (conv49)	–	0.80589
	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.10, 10, 0.1	0.82306
	SeqSLAM (cone)	HybridNet (conv4)	–	0.83792
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.84212
	SeqSLAM	HybridNet (conv6)	–	0.86103
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 150}}$	–	0.88893
	SeqSLAM	ResNet50 (conv49)	0.0005, 0.95, 0.10, 20, 0.1	0.91075
	NOSeqSLAM, $\eta = 2$	HybridNet (conv6)	–	0.91255
	NOSeqSLAM, $\eta = 2$	$\text{VLAD}_{\text{SIFT, 200}}$	–	0.91658
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.00075, 0.90, 0.10, 10, 0.1	**0.93242**
51	SeqSLAM (cone)	ResNet50 (conv49)	–	0.69520
	SeqSLAM	ResNet50 (conv49)	–	0.77568
	SeqSLAM (cone)	HybridNet (conv6)	–	0.80746
	SeqSLAM (cone)	ResNet50 (conv49)	0.001, 0.90, 0.10, 10, 0.1	0.80871
	SeqSLAM	HybridNet (conv3)	–	0.81251
	SeqSLAM	$\text{VLAD}_{\text{SURF, 100}}$	–	0.83366
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.84421
	SeqSLAM	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	0.86210
	NOSeqSLAM, $\eta = 2$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.88194
	NOSeqSLAM, $\eta = 2$	HybridNet (conv5)	–	0.88487
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.001, 0.90, 0.00, 40, 0.1	**0.90119**

every sequence on Bonn and Freiburg (except Freiburg, $d_s = 31$) NOSeqSLAM achieved the best result. Moreover, the difference between the best and the second-best results is quite noticeable as is the difference between the fine-tuned and original representations. Once again, our fine-tuned representations outperformed HybridNet, probably due to the aforementioned reason. Although comparable, HybridNet performed slightly better on the Nordland dataset in terms of the R@100%P measure. Handcrafted features, as in the case of AUC performance, achieved poor results except for the Bonn dataset.

3.3.4 *Adversarial training and adversarial examples*

We have conducted an additional batch of experiments that examine adversaries in this context. We were wondering if an adversarial training has a positive impact on quantitative results, i.e., is an adversarial training a good regularizer. Also, we tried to attack different ResNet50 architectures fine-tuned either with an adversarial or standard training to see if they are resistant to adversarial examples. For the sake of economy, we conducted 100 epoch long trainings where we tweaked standard neural network learning hyperparameters: $lr \in \{0.0005, 0.001\}$, $m = 0.9$, $wd \in \{0.0, 0.1\}$, $s \in \{10, 40\}$, $s_\gamma = 0.1$ and hyperparameters related to adversarial training: $\alpha \in \{0.01, 0.1\}$, $\epsilon \in \{0.1, 1\}$. Same αs and ϵs have been used to create adversarial examples. In terms of training, i.e., fine-tuning, we performed either standard

Table 3.2: AUC of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Freiburg dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_γ	Score
31	SeqSLAM	$\text{VLAD}_{\text{SIFT, 300}}$	–	0.11587
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.12142
	SeqSLAM	ResNet50 (conv49)	–	0.12992
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 50}}$	–	0.16526
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.32193
	SeqSLAM (cone)	HybridNet (conv4)	–	0.35708
	NOSeqSLAM, $\eta = 3$	HybridNet (conv4)	–	0.59822
	SeqSLAM	HybridNet (conv4)	–	0.60882
	SeqSLAM (cone)	ResNet50 (conv49)	0.00075, 0.90, 0.00, 40, 0.1	0.83686
	SeqSLAM	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	0.87894
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	**0.91824**
51	SeqSLAM (cone)	ResNet50 (conv49)	–	0.08139
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 50}}$	–	0.09686
	SeqSLAM	ResNet50 (conv49)	–	0.10346
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 200}}$	–	0.16896
	SeqSLAM (cone)	HybridNet (conv4)	–	0.32125
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.39966
	SeqSLAM	HybridNet (conv4)	–	0.54489
	NOSeqSLAM, $\eta = 2$	HybridNet (conv4)	–	0.57299
	SeqSLAM	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	0.81213
	SeqSLAM (cone)	ResNet50 (conv49)	0.00075, 0.90, 0.00, 40, 0.1	0.84745
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	**0.91829**

training, or an adversarial training with PGD-enhanced inputs or interpolated adversarial training. For every kind of training, we extracted feature maps, either uncorrupted or corrupted with PGD. First of all, our pipeline is not a standard deep learning pipeline where the same criterium is used both for training and testing. However, several conclusions can be drawn from Tables3.7,3.8,3.9.

Unsurprisingly, for every dataset, standard training with uncorrupted feature maps achieved the best AUC performances. Once again, NOSeqSLAM outperformed SeqSLAM in every situation. Also, interpolated adversarial training achieved better performances with regard to PGD-enhanced training no matter what sequence-based method and adversary has been used in the evaluation. Interestingly enough, IAT-fine-tuned models outperform standard-fine-tuning models on the Freiburg and Nordland datasets. PGD-enhanced training, however, did not show any promising results no matter what dataset and hyperparameters were used. In terms of adversarial attacks, uncorrupted feature maps, naturally, obtained better AUC results, and inversely, corrupted adversarial feature maps deteriorate performances (except for the Nordland dataset). When adversarially-corrupted feature maps are used, IAT-fine-tuned networks outperform both standard and PGD-fine-tuned models, which suggests that interpolated adversarial training can be effective against adversarial attacks in our visual place recognition pipeline.

Table 3.3: AUC of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Nordland dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_γ	Score
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.71579
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.77105
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.79346
	SeqSLAM	ResNet50 (conv49)	–	0.81860
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.89138
31	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.10, 20, 0.1	0.92657
	SeqSLAM (cone)	HybridNet (conv3)	–	0.96638
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	0.0005, 0.95, 0.05, 30, 0.1	0.96639
	SeqSLAM	ResNet50 (conv49)	0.0001, 0.95, 0.10, 20, 0.1	0.96806
	SeqSLAM	HybridNet (conv3)	–	**0.97000**
	NOSeqSLAM, $\eta = 2$	HybridNet (conv3)	–	**0.97000**
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.82994
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.85677
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.87603
	SeqSLAM	ResNet50 (conv49)	–	0.88513
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.90424
51	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.05, 20, 0.1	0.94608
	SeqSLAM (cone)	HybridNet (conv3)	–	**0.95000**
	SeqSLAM	HybridNet (conv6)	–	**0.95000**
	SeqSLAM	ResNet50 (conv49)	0.0005, 0.95, 0.05, 30, 0.1	**0.95000**
	NOSeqSLAM, $\eta = 3$	HybridNet (conv4)	–	**0.95000**
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.0001, 0.95, 0.05, 30, 0.1	**0.95000**

3.4 Mutual information-based feature selection for visual place recognition

The act of appending the norm of optimizable parameters to an objective function, referred in Subsection3.2.2as "parameter norm penalties", traces its roots from constrained linear regression where it is also called *feature selection*. Feature selection, the self-explanatory term, refers to a process of selecting "better" among features of given inputs in order to enhance performances. As reported in our paper [93], in linear regression we optimize a linear map $\boldsymbol{\theta} \in \mathbb{R}^n$ in order to predict a continuous variable $\hat{y} \in \mathbb{R}$ given an input vector $x \in \mathbb{R}^n$, i.e.,

$$\hat{y} = \boldsymbol{\theta}^T x. \tag{3.68}$$

Table 3.4: R@100%P of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Bonn dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_γ	Score
31	SeqSLAM	ResNet50 (conv49)	–	0.00000
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.00184
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.00368
	SeqSLAM	HybridNet (conv6)	–	0.05147
	NOSeqSLAM, $\eta = 2$	HybridNet (conv6)	–	0.05515
	SeqSLAM (cone)	HybridNet (conv5)	–	0.08640
	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.10, 40, 0.1	0.14154
	NOSeqSLAM, $\eta = 2$	$\text{VLAD}_{\text{SURF, 300}}$	–	0.37500
	SeqSLAM	ResNet50 (conv49)	0.002, 0.90, 0.00, 20, 0.1	0.43566
	SeqSLAM	$\text{VLAD}_{\text{SURF, 300}}$	–	0.476103
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.00075, 0.90, 0.05, 20, 0.1	**0.50552**
51	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.00184
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.05147
	SeqSLAM	HybridNet (conv6)	–	0.06802
	SeqSLAM (cone)	HybridNet (conv5)	–	0.09375
	SeqSLAM	ResNet50 (conv49)	–	0.13787
	SeqSLAM (cone)	ResNet50 (conv49)	0.00075, 0.90, 0.00, 40, 0.1	0.14154
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 300}}$	–	0.16360
	NOSeqSLAM, $\eta = 3$	HybridNet (conv3)	–	0.19485
	SeqSLAM	ResNet50 (conv49)	0.0005, 0.90, 0.05, 40, 0.1	0.32353
	NOSeqSLAM, $\eta = 2$	$\text{VLAD}_{\text{SURF, 150}}$	–	0.46691
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.0005, 0.95, 0.20, 10, 0.1	**0.61029**

Let $\mathbf{y} = [y_1 \dots y_m]^T \in \mathbb{R}^m$ and $\mathbf{X} = [x_1 \dots x_m]^T \in \mathbb{R}^{m \times n}$ denote data stacked as a vector and matrix respectively. The optimal map $\boldsymbol{\theta}^*$ is given with

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{3.69}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} (y_i - \boldsymbol{\theta}^T x_i)^2 \tag{3.70}$$

$$= \arg\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2. \tag{3.71}$$

Then, feature selection for linear regression is defined as a constrained variant of (3.71), i.e.,

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \ \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 \\ &\text{s.t. } \|\boldsymbol{\theta}\|_0 = k. \end{aligned} \tag{3.72}$$

However, due to the NP-hardness of optimizing with the zero norm constraint, more relaxed variants of this *constrained optimization problem* (abbr. COP) exist – *lasso regression* defined as

$$\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1, \ \lambda \in \mathbb{R}^+, \tag{3.73}$$

Table 3.5: R@100%P of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Freiburg dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_γ	Score
31	SeqSLAM (cone)	ResNet50 (conv49)	–	0.00000
	SeqSLAM	ResNet50 (conv49)	–	0.00148
	SeqSLAM (cone)	HybridNet (conv3)	–	0.00175
	SeqSLAM	HybridNet (conv6)	–	0.00740
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 300}}$	–	0.01036
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.01479
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.01923
	NOSeqSLAM, $\eta = 3$	HybridNet (conv6)	–	0.03846
	SeqSLAM	ResNet50 (conv49)	0.001, 0.90, 0.05, 20, 0.1	0.19231
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.001, 0.90, 0.05, 20, 0.1	0.30917
	SeqSLAM (cone)	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	**0.35503**
51	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.00000
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.00000
	SeqSLAM (cone)	HybridNet (conv3)	–	0.00000
	SeqSLAM	ResNet50 (conv49)	–	0.00000
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 300}}$	–	0.00740
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 300}}$	–	0.03107
	NOSeqSLAM, $\eta = 3$	HybridNet (conv6)	–	0.04586
	SeqSLAM	HybridNet (conv4)	–	0.14349
	SeqSLAM	ResNet50 (conv49)	0.001, 0.90, 0.05, 20, 0.1	0.19823
	SeqSLAM (cone)	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	0.35947
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.001, 0.95, 0.05, 10, 0.1	**0.52515**

and *ridge regression* defined as

$$\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2,\ \lambda \in \mathbb{R}^+. \tag{3.74}$$

The role of regularization terms $\lambda\|\boldsymbol{\theta}\|_1$ and $\lambda\|\boldsymbol{\theta}\|_2^2$ is to assure sparsity [94] what can be seen in Figure3.21. The more we increase λ, the more and more features diminish, while those that are significant remain.

The combination of lasso and ridge regression is called *elastic net regression*. By incorporating *information theory*, the authors of [95] enhance elastic net regression. Their *mutual information-based* feature selection is defined as

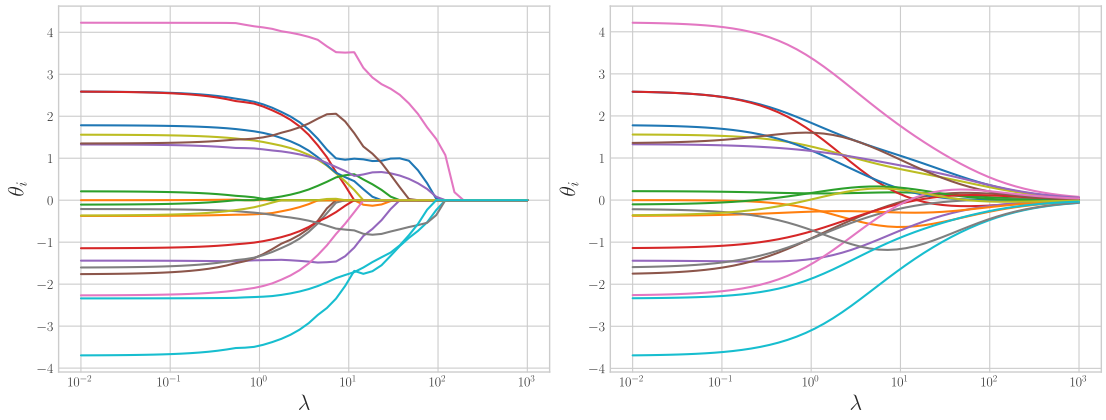
$$\min_{\boldsymbol{\theta}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|_2^2 + \lambda_1\|\boldsymbol{\theta}\|_1 + \lambda_2\|\boldsymbol{\theta}\|_2^2 - \lambda_3\boldsymbol{\theta}^T\boldsymbol{W}\boldsymbol{\theta},\ \lambda_i \in \mathbb{R}^+, \tag{3.75}$$

where **W** is called *the feature informativeness matrix* that encodes how one feature relate to another by using feature similarities based on the Jensen-Shannon divergence. In one of the authors' previous paper [96], it is defined how to evaluate this relation between features with the following formulation. Given a set of features

$$\mathcal{F} = \{f^{(1)}, \ldots, f^{(m)}\} \in \mathbb{R}^n, \tag{3.76}$$

Table 3.6: R@100%P of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Nordland dataset using feature maps from the original ResNet50 [65], fine-tuned ResNet50 (ours), and HybridNet [85].

d_s	Method	Image model	lr, m, wd, s, s_y	Score
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.00700
	SeqSLAM	ResNet50 (conv49)	–	0.05700
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.07700
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.12600
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	–	0.14700
31	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.10, 20, 0.1	0.19500
	NOSeqSLAM, $\eta = 3$	ResNet50 (conv49)	0.0005, 0.90, 0.10, 20, 0.1	0.68300
	SeqSLAM	ResNet50 (conv49)	0.0005, 0.95, 0.05, 30, 0.1	0.85300
	SeqSLAM (cone)	HybridNet (conv5)	–	0.96800
	SeqSLAM	HybridNet (conv3)	–	**0.97000**
	NOSeqSLAM, $\eta = 2$	HybridNet (conv3)	–	**0.97000**
	SeqSLAM (cone)	ResNet50 (conv49)	–	0.10300
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	–	0.13600
	NOSeqSLAM, $\eta = 3$	$\text{VLAD}_{\text{SIFT, 250}}$	–	0.15700
	SeqSLAM	ResNet50 (conv49)	–	0.23100
	SeqSLAM	$\text{VLAD}_{\text{SIFT, 200}}$	–	0.26700
51	SeqSLAM (cone)	ResNet50 (conv49)	0.0005, 0.95, 0.05, 20, 0.1	0.54100
	SeqSLAM	HybridNet (conv3)	–	**0.95000**
	SeqSLAM	ResNet50 (conv49)	0.0001, 0.95, 0.05, 30, 0.1	**0.95000**
	SeqSLAM (cone)	HybridNet (conv3)	–	**0.95000**
	NOSeqSLAM, $\eta = 2$	HybridNet (conv3)	–	**0.95000**
	NOSeqSLAM, $\eta = 2$	ResNet50 (conv49)	0.0001, 0.95, 0.05, 30, 0.1	**0.95000**



(a) (b)

Figure 3.21: The effect of (a) lasso and (b) ridge regression, relaxed variants of linear regression constrained with $\|\cdot\|_0$. As λ increases, more and more features diminish. Images taken from our paper [93].

Table 3.7: AUC of NOSeqSLAM [5] (ours) and SeqSLAM [2] for the Bonn dataset using uncorrupted and adversarially-corrupted features maps from ResNet50 fine-tuned with either a standard or an adversarial training.

d_s	Method	Training	α, ϵ	Adversary	Score
31	SeqSLAM	PGD	0.1, 0.1	PGD	0.33080
	NOSeqSLAM	PGD	0.1, 0.1	PGD	0.39810
	SeqSLAM	Standard	–	PGD	0.45561
	NOSeqSLAM	Standard	–	PGD	0.60700
	SeqSLAM	IAT	0.1, 0.1	PGD	0.73746
	SeqSLAM	PGD	0.1, 0.01	–	0.80703
	NOSeqSLAM	IAT	0.1, 0.01	PGD	0.84721
	SeqSLAM	IAT	1, 0.01	–	0.86151
	NOSeqSLAM	PGD	0.1, 0.01	–	0.87113
	SeqSLAM	Standard	–	–	0.87468
	NOSeqSLAM	IAT	1, 0.01	–	0.90393
	NOSeqSLAM	Standard	–	–	**0.91833**
51	SeqSLAM	PGD	0.1, 0.1	PGD	0.28232
	SeqSLAM	Standard	–	PGD	0.35572
	NOSeqSLAM	PGD	0.1, 0.1	PGD	0.37032
	SeqSLAM	IAT	0.1, 0.01	PGD	0.59642
	NOSeqSLAM	Standard	–	PGD	0.66347
	SeqSLAM	PGD	0.1, 0.01	–	0.71219
	SeqSLAM	IAT	1, 0.01	–	0.79825
	NOSeqSLAM	IAT	0.1, 0.01	PGD	0.80554
	SeqSLAM	Standard	–	–	0.83271
	NOSeqSLAM	PGD	0.1, 0.01	–	0.85369
	NOSeqSLAM	IAT	1, 0.01	–	0.87438
	NOSeqSLAM	Standard	–	–	**0.90244**

the respective set of probability distributions for each feature is obtained in order to evaluate features with the Shannon entropy. First, for $f^{(k)}$, a related undirected graph

$$G_{f^{(k)}} = (V_{f^{(k)}}, E_{f^{(k)}}) \tag{3.77}$$

is defined with nodes $V_{f^{(k)}}$ assigned to each component of the feature. Then, the weight function $w : E_{f^{(k)}} \to \mathbb{R}_0^+$ from $v_l^{(k)} \in V_{f^{(k)}}$ to $v_m^{(k)} \in V_{f^{(k)}}$ is defined as

$$w(v_l^{(k)}, v_m^{(k)}) = \sqrt{(f^{(k)}[l] - f^{(k)}[m])^2}. \tag{3.78}$$

Finally, a probability distribution p is assigned to $f^{(k)}$ by using *the steady state random walk* procedure[97]. For each node $v_l^{(k)} \in V_{f^{(k)}}$, its probability is defined as

$$p(v_l^{(k)}) = \frac{\deg(v_l^{(k)})}{\sum\limits_{v_m^{(k)} \in V_{f^{(k)}}} \deg(v_m^{(k)})}, \tag{3.79}$$

Table 3.8: AUC of NOSeqSLAM [5] (ours) and SeqSLAM [2] for the Freiburg dataset using uncorrupted and adversarially-corrupted features maps from ResNet50 fine-tuned with either a standard or an adversarial training.

d_s	Method	Training	α, ϵ	Adversary	Score
31	SeqSLAM	Standard	–	PGD	0.11498
	NOSeqSLAM	Standard	–	PGD	0.15109
	NOSeqSLAM	PGD	1, 0.01	PGD	0.32262
	SeqSLAM	PGD	1, 0.01	PGD	0.32439
	NOSeqSLAM	PGD	0.1, 0.01	–	0.47250
	SeqSLAM	PGD	0.1, 0.01	–	0.50085
	SeqSLAM	Standard	–	–	0.63290
	SeqSLAM	IAT	1, 0.01	PGD	0.66515
	NOSeqSLAM	IAT	1, 0.01	PGD	0.69967
	SeqSLAM	IAT	0.1, 0.1	–	0.75217
	NOSeqSLAM	Standard	–	–	0.76551
	NOSeqSLAM	IAT	0.1, 0.1	–	**0.78083**
51	SeqSLAM	Standard	–	PGD	0.09232
	NOSeqSLAM	Standard	–	PGD	0.12559
	SeqSLAM	PGD	1, 0.01	PGD	0.37506
	NOSeqSLAM	PGD	1, 0.01	PGD	0.39314
	SeqSLAM	PGD	0.1, 0.01	–	0.46996
	NOSeqSLAM	PGD	0.1, 0.01	–	0.51407
	SeqSLAM	IAT	1, 0.01	PGD	0.64731
	SeqSLAM	Standard	–	–	0.65978
	NOSeqSLAM	IAT	1, 0.01	PGD	0.70099
	SeqSLAM	IAT	0.1, 0.1	–	0.76150
	NOSeqSLAM	Standard	–	–	0.77304
	NOSeqSLAM	IAT	0.1, 0.1	–	**0.77913**

where

$$\deg(v_l^{(k)}) = \sum_{v_m^{(k)} \in V_{f^{(k)}}} w(v_l^{(k)}, v_m^{(k)}). \tag{3.80}$$

The Jensen-Shannon divergence, a method that measures the similarity between probability distributions, is defined for respective probabilities of $f^{(k)} \in \mathcal{F}$ and $f^{(l)} \in \mathcal{F}$ as

$$JSD(p(f^{(k)}), p(f^{(l)})) = H_S\left(\frac{p(f^{(k)}) + p(f^{(l)})}{2}\right) - \frac{H_S(p(f^{(k)})) + H_S(p(f^{(l)}))}{2}, \tag{3.81}$$

where

$$H_S(p(f^{(k)})) = -\sum_{v_m^{(k)} \in V_{f^{(k)}}} p(v_m^{(k)}) \log p(v_m^{(k)}) \tag{3.82}$$

is the Shannon entropy. Finally, the similarity between $f^{(k)} \in \mathcal{F}$ and $f^{(l)} \in \mathcal{F}$ is defined as

$$I_S(p(f^{(k)}), p(f^{(l)})) = \exp\big(-JSD(p(f^{(k)}), p(f^{(l)}))\big). \tag{3.83}$$

Table 3.9: AUC of NOSeqSLAM [5] (ours) and SeqSLAM [2] for the Nordland dataset using uncorrupted and adversarially-corrupted features maps from ResNet50 fine-tuned with either a standard or an adversarial training.

d_s	Method	Training	α, ϵ	Adversary	Score
31	NOSeqSLAM	Standard	–	PGD	0.90464
	SeqSLAM	Standard	–	PGD	0.90690
	NOSeqSLAM	PGD	0.1, 0.01	–	0.94245
	SeqSLAM	PGD	0.1, 0.1	PGD	0.94581
	SeqSLAM	PGD	0.1, 0.01	–	0.94807
	NOSeqSLAM	PGD	0.1, 0.1	PGD	0.94928
	NOSeqSLAM	Standard	–	–	0.96785
	SeqSLAM	IAT	0.1, 0.01	PGD	0.96837
	NOSeqSLAM	IAT	0.1, 0.01	PGD	0.96942
	SeqSLAM	Standard	–	–	0.97000
	SeqSLAM	IAT	0.1, 0.01	–	0.97000
	NOSeqSLAM	IAT	0.1, 0.01	–	**0.97000**
51	NOSeqSLAM	PGD	0.1, 0.01	–	0.93302
	NOSeqSLAM	PGD	0.1, 0.1	PGD	0.93392
	SeqSLAM	PGD	0.1, 0.01	–	0.93730
	NOSeqSLAM	Standard	–	PGD	0.93878
	SeqSLAM	PGD	0.1, 0.1	PGD	0.93883
	SeqSLAM	Standard	–	PGD	0.94187
	SeqSLAM	IAT	0.1, 0.1	PGD	0.94900
	NOSeqSLAM	Standard	–	–	0.94900
	SeqSLAM	IAT	0.1, 0.1	–	0.95000
	SeqSLAM	Standard	–	–	0.95000
	NOSeqSLAM	IAT	1, 0.1	–	0.95000
	NOSeqSLAM	IAT	0.1, 0.01	PGD	**0.95000**

3.4.1 *Feature selection for visual place recognition*

According to the formulation stated above, we will briefly mention how to formulate feature selection for visual place recognition data – $\mathcal{Q}$ and $\mathcal{D}$ datasets. More information can be found in our paper [5]. First, images $I_i \in \mathcal{Q} \cup \mathcal{D}$ are mapped to appropriate image representations $z_i \in \mathbb{R}^n$. Then, we stack query images as the rows of the matrix $\mathcal{Q}_M \in \mathbb{R}^{|\mathcal{Q}| \times n}$ that is defined as

$$\mathcal{Q}_M = \begin{bmatrix} z_{q_1} \\ z_{q_2} \\ \vdots \\ z_{q_{|\mathcal{Q}|}} \end{bmatrix}. \tag{3.84}$$

We exploit the fact, at least in datasets that we use, that for each query image $I_{q_i} \in \mathcal{Q}$, there exists at least one reference image $I_{d_j} \in \mathcal{D}$. In order to have the same dimension, we build

the matrix $\mathcal{D}_M \in \mathbb{R}^{|\mathcal{Q}| \times n}$ as

$$\mathcal{D}_M = \begin{bmatrix} z_{d_{1^*}} \\ z_{d_{2^*}} \\ \vdots \\ z_{d_{|\mathcal{Q}|^*}} \end{bmatrix}, \tag{3.85}$$

with $z_{d_{i^*}}$ chosen such that

$$z_{d_{i^*}} = \underset{I_{d_j} \in \mathcal{GT}(I_{q_i})}{\arg\max}\ s_{z_{q_i}, z_{d_j}}, \tag{3.86}$$

i.e., such that the i-th row of $\mathcal{D}_M$ is the most similar representation of images from $\mathcal{D}$ for the i-th image from $\mathcal{Q}$.

Then, we can define *features* according to the formulation from [96] as columns of $\mathcal{Q}_M$ and $\mathcal{D}_M$, respectively, i.e., we define the k-th feature of $\mathcal{Q}_M$ and the k-th feature of $\mathcal{D}_M$ as

$$f_{\mathcal{Q}}^{(k)} = \mathcal{Q}_M[:, k],\ f_{\mathcal{D}}^{(k)} = \mathcal{D}_M[:, k]. \tag{3.87}$$

A good feature should be similar when occurs in both $\mathcal{Q}$ and $\mathcal{D}$ so we measure *the quality* of the k-th feature as

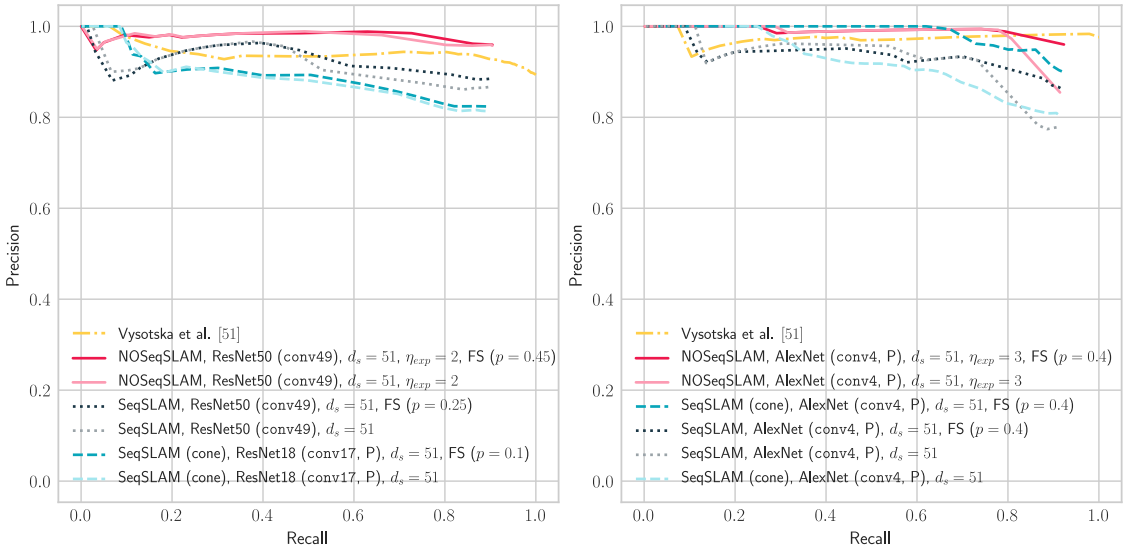
$$q(k) = I_S(p(f_{\mathcal{Q}}^{(k)}), p(f_{\mathcal{D}}^{(k)})), \tag{3.88}$$

and dismiss all those features below a specified threshold.

3.4.2 *Experimental results*

As in the previous batch of experiments with softmax regression, here we also used Bonn and Freiburg datasets (Subsubsection3.3.3.1). Regarding handcrafted image models, we experimented with histogram of oriented gradients (Subsubsection3.1.2.4). Regarding learned image models, we extracted feature maps from AlexNet including the 3rd convolutional layer (`conv3`), feature maps from AlexNet including the 4th convolutional layer (`conv4`), feature maps from ResNet18 and ResNet50 including their last convolutional layers (`conv17` and `conv49`, respectively), and, already included with the datasets, feature maps from the OverFeat architecture [98]. AlexNet and ResNet instances were either trained on the ImageNet or the Places365 dataset [78]. Also, we tried to select features for the best-performing fine-tuned architectures from Section3.3, but no improvements have been observed. This suggests that softmax regression already fine-tuned stock ImageNet-trained architectures well. Therefore, in this section, we were focused solely on non-fine-tuned convolutional neural networks. We compared sequence-based methods discussed earlier, and additionaly, a non-sequence-based method by [52]. This method does not depend on d_s so we omit its results from tables.

We constructed features and evaluated (3.88) on the Freiburg dataset in order to obtain feature qualities $q(k), \forall k$. Then we performed feature selection on the Bonn dataset for different percentiles $q_i, \forall i$ of feature qualities where those features that fall below a threshold equal to a specified percentile q_i were removed. According to specified percentiles, we conducted multiple visual place recognition experiments using different sequence-based methods. In the same manner, we evaluated (3.88) on the Bonn dataset and obtained feature qualities further used for visual place recognition evaluation on the Freiburg dataset.



(a) (b)

Figure 3.22: Precision recall curves for (a) the Bonn and (b) the Freiburg datasets. For each sequence-based algorithm, d_s = 51, the best-performing image model with feature selection and its no-feature-selection counterpart are shown. Images taken from our paper [5].

In Tables3.10,3.12AUC results are reported. For sequence lengths $d_s \in \{31, 43, 51\}$, for each sequence-based method, we picked its best-performing image model where no feature selection is conducted and its best-performing image model where feature selection is conducted. It is visible how the proposed feature selection technique is effective as image models with their features selected achieved better results for each sequence length. ResNet architectures have proven to be more effective for the Bonn dataset, while AlexNet was a better option for the Freiburg dataset. Also, our NOSeqSLAM method outperformed other sequence-based methods. In Figure3.22we see the corresponding precision-recall curves. The method by [52] achieved the AUC of 0.94309 for the Bonn dataset and 0.97471 for the Freiburg dataset, respectively. This is better than sequence-based methods because these methods cannot match the first $\lfloor \frac{d_s}{2} \rfloor$ and last $\lfloor \frac{d_s}{2} \rfloor$ images as already discussed (Subsubsection3.3.3.5). This is also visible in Figure3.22where the corresponding curve of [52], for this reason, reaches 1 on x-axis. Accordingly, we also report R@100%P results for the same datasets and place recognition methods. Once again, the proposed feature selection with the proposed sequence-based method, NOSeqSLAM, achieved the best results. The method by [52] performed poorly – 0.06250 for the Bonn dataset and 0.07396 for the Freiburg dataset, respectively.

3.5SUMMARY

In this comprehensive chapter on image models in visual place recognition, multiple image models have been covered. First, we reviewed handcrafted gradient-based image models and then ubiquitous learned models – convolutional neural networks – that are state-of-the-art image models. We have shown how to adapt softmax regression and mutual information-based feature selection for image models in the context of visual place recognition. The

Table 3.10: AUC of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Bonn dataset using image models with proposed feature selection (q_i column with value) and without feature selection (q_i column with –).

d_s	Method	Image model	Dim.	η	q_i	Score
31	SeqSLAM (cone)	OverFeat (conv10)	153600	–	–	0.83521
	SeqSLAM (cone)	OverFeat (conv10)	46080	–	0.7	0.83683
	SeqSLAM	ResNet18 (conv17)	200192	–	–	0.89153
	SeqSLAM	ResNet18 (conv17)	140134	–	0.3	0.90005
	NOSeqSLAM	ResNet50 (conv49)	800768	2	–	0.91501
	NOSeqSLAM	ResNet50 (conv49)	680653	2	0.15	**0.91890**
43	SeqSLAM (cone)	ResNet18 (conv17, Places365)	200192	–	–	0.81094
	SeqSLAM (cone)	ResNet18 (conv17, Places365)	190182	–	0.05	0.81692
	SeqSLAM	ResNet50 (conv49)	800768	–	–	0.87506
	SeqSLAM	ResNet50 (conv49)	40038	–	0.95	0.88787
	NOSeqSLAM	ResNet50 (conv49)	800768	2	–	0.89647
	NOSeqSLAM	ResNet50 (conv49)	120116	2	0.85	**0.89798**
51	SeqSLAM (cone)	ResNet18 (conv17, Places365)	200192	–	–	0.78836
	SeqSLAM (cone)	ResNet18 (conv17, Places365)	180172	–	0.1	0.79570
	SeqSLAM	ResNet50 (conv49)	800768	–	–	0.82185
	SeqSLAM	ResNet50 (conv49)	600576	–	0.25	0.83078
	NOSeqSLAM	ResNet50 (conv49)	800768	2	–	0.88189
	NOSeqSLAM	ResNet50 (conv49)	440424	2	0.45	**0.88592**

Table 3.11: R@100%P of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Bonn dataset using image models with proposed feature selection (q_i column with value) and without feature selection (q_i column with –).

d_s	Method	Image model	Dim.	η	q_i	Score
31	NOSeqSLAM	AlexNet (conv4)	360448	2	–	0.01103
	SeqSLAM	ResNet18 (conv17)	200192	–	–	0.10846
	SeqSLAM	ResNet18 (conv17)	140134	–	0.3	0.13419
	SeqSLAM (cone)	ResNet50 (conv49, Places365)	800768	–	–	0.13603
	SeqSLAM (cone)	ResNet50 (conv49, Places365)	360347	–	0.55	0.14522
	NOSeqSLAM	AlexNet (conv4)	36045	2	0.9	**0.14706**
43	SeqSLAM	AlexNet (conv4)	360448	–	–	0.01103
	NOSeqSLAM	AlexNet (conv4)	360448	2	–	0.04044
	SeqSLAM (cone)	ResNet18 (conv17)	200192	–	–	0.11029
	SeqSLAM	AlexNet (conv4)	54068	–	0.85	0.12684
	SeqSLAM (cone)	ResNet18 (conv17)	120116	–	0.4	0.12868
	NOSeqSLAM	AlexNet (conv4)	126157	2	0.65	**0.17647**
51	NOSeqSLAM	AlexNet (conv4)	360448	3	–	0.00000
	SeqSLAM	AlexNet (conv4)	360448	–	–	0.01654
	SeqSLAM (cone)	ResNet18 (conv17)	200192	–	–	0.11397
	SeqSLAM (cone)	ResNet18 (conv17)	150144	–	0.25	0.13603
	SeqSLAM	AlexNet (conv4)	54068	–	0.85	0.15257
	NOSeqSLAM	AlexNet (conv4)	36045	3	0.9	**0.17647**

Table 3.12: AUC of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Freiburg dataset using image models with proposed feature selection (q_i column with value) and without feature selection (q_i column with –).

d_s	Method	Image model	Dim.	η	q_i	Score
31	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.63907
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.85604
	SeqSLAM	AlexNet (conv4, Places365)	360448	–	–	0.87314
	SeqSLAM	AlexNet (conv4, Places365)	216268	–	0.4	0.88080
	NOSeqSLAM	AlexNet (conv4, Places365)	360448	3	–	0.92429
	NOSeqSLAM	AlexNet (conv4, Places365)	216268	3	0.4	**0.93119**
43	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.66569
	SeqSLAM	AlexNet (conv4, Places365)	360448	–	–	0.86373
	SeqSLAM	AlexNet (conv4, Places365)	216268	–	0.4	0.86775
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.87361
	NOSeqSLAM	AlexNet (conv4, Places365)	360448	3	–	0.91016
	NOSeqSLAM	AlexNet (conv4, Places365)	216268	3	0.4	**0.92177**
51	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.67670
	SeqSLAM	AlexNet (conv4, Places365)	360448	–	–	0.84598
	SeqSLAM	AlexNet (conv4, Places365)	216268	–	0.4	0.85850
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.87534
	NOSeqSLAM	AlexNet (conv4, Places365)	360448	3	–	0.89907
	NOSeqSLAM	AlexNet (conv4, Places365)	216268	3	0.4	**0.91346**

Table 3.13: R@100%P of NOSeqSLAM [5] (ours), SeqSLAM [2] and cone-based SeqSLAM [25] for the Freiburg dataset using image models with proposed feature selection (q_i column with value) and without feature selection (q_i column with –).

d_s	Method	Image model	Dim.	η	q_i	Score
31	SeqSLAM	ResNet50 (conv49)	800768	–	–	0.10207
	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.23669
	SeqSLAM	ResNet50 (conv49)	520499	–	0.35	0.41864
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.57101
	NOSeqSLAM	ResNet50 (conv49, Places365)	800768	3	–	0.62574
	NOSeqSLAM	ResNet50 (conv49, Places365)	480459	3	0.4	**0.63166**
43	SeqSLAM	AlexNet (conv4, Places365)	360448	–	–	0.05917
	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.26036
	SeqSLAM	AlexNet (conv4, Places365)	18023	–	0.95	0.28107
	NOSeqSLAM	AlexNet (conv4, Places365)	360448	3	–	0.30325
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.58432
	NOSeqSLAM	AlexNet (conv4, Places365)	180224	3	0.5	**0.69822**
51	SeqSLAM	AlexNet (conv4, Places365)	360448	–	–	0.11243
	SeqSLAM (cone)	AlexNet (conv4, Places365)	360448	–	–	0.24852
	NOSeqSLAM	AlexNet (conv4, Places365)	360448	3	–	0.28698
	SeqSLAM	AlexNet (conv4, Places365)	36045	–	0.9	0.32101
	SeqSLAM (cone)	AlexNet (conv4, Places365)	216268	–	0.4	0.61982
	NOSeqSLAM	AlexNet (conv4, Places365)	162202	3	0.55	**0.74112**

extensive evaluation shows that such obtained image models outperform original non-fine-tuned models and models where no features are selected, which justifies the second contribution of the thesis: Method for robust visual place recognition with deep representations that uses softmax regression and mutual information-based feature selection. Additionally, powered by our image models, our sequence-based method NOSeqSLAM outperformed SeqSLAM and cone-based SeqSLAM, so the first contribution is, as foreshadowed in Section2.4, justified quantitatively too.

4

Closing the loop in simultaneous localization and mapping

In the final chapter of the thesis, we will introduce factor graphs (Section4.1), a powerful tool for estimation. So far, visual place recognition has been a standalone research topic, but from now on, we will use it within simultaneous localization and mapping (Section4.2), another research topic on its own. Further, in Section4.3, we will present how our place matching method, NOSeqSLAM, is adapted in order to fit into the SLAM pipeline, i.e., how we made it an online place matching method. Finally, we will present two fundamentally different SLAM implementations, one being two-dimensional and range sensor-based (Section4.4) and the other being three-dimensional and based on visual sensors (Section4.5). In both of these implementations, we will check how place matching performs in order to detect loops, which in turn improves estimation results. Through this chapter, we will use the terms "place recognition", "place matching" and "loop closing detection" interchangeably, as these terms are synonyms – to recognize an already seen place is to match it with its previous measurement/image/scan which is a way to tell a SLAM system that an additional constraint, a loop closure, can be incorporated into a set of constraints that make the estimation more accurate.

4.1 Factor graphs

Factor graphs are topological structures used to estimate stochastic processes that consist of multiple *states, measurements* and, optionally, *control inputs*. They provide a convenient interface for the optimization of such processes, which means that by using factor graphs we factorize an objective function and optimize it with respect to variables we would like to estimate. This way, an error is diminished and an optimal guess of the state of a process is obtained.

As described in [99], a factor graph $F = (\mathcal{U} \cup \mathcal{V}, \mathcal{E})$ [1] is a *bipartite graph* [2] comprising nodes $\mathcal{U} \cup \mathcal{V}$ and edges $\mathcal{E}$. Nodes in $\mathcal{U}$ are called *factors* while nodes in $\mathcal{V}$ are called *variables*. Each factor $\phi_i \in \mathcal{U}$ is parametrized by $X_i \subseteq \mathcal{V}$, thus $\phi_i := \phi_i(X_i)$. An edge $e_{i,l} \in \mathcal{E}$ holds an information whether $x_l \in \mathcal{V}$ parametrizes ϕ_i, i.e., whether x_l is contained in X_i. In other

[1] In [99], F is defined as $(\mathcal{U}, \mathcal{V}, \mathcal{E})$. To define a graph, we use an ordered pair rather than an ordered triple.

[2] A bipartite graph is an undirected graph $G = (\mathcal{V}, \mathcal{E})$ in which $\mathcal{V}$ can be partitioned into disjoint $\mathcal{V}_1$ and $\mathcal{V}_2$ such that $\{u, v\} \in \mathcal{E}$ implies either $u \in \mathcal{V}_1$ and $v \in \mathcal{V}_2$ or $u \in \mathcal{V}_2$ and $v \in \mathcal{V}_1$ [29, p. 1172].

words, $e_{i,l} = \{\phi_i, x_l\} \in \mathcal{E} \Leftrightarrow x_l \in X_i \subseteq \mathcal{V}$. Altogether, the factorization by F is defined as

$$\phi(\mathcal{V}) = \prod_i \phi_i(X_i), \tag{4.1}$$

while the optimal estimate, called *the maximum a posteriori estimate* is

$$\mathcal{V}^* = \arg\max_{\mathcal{V}} \phi(\mathcal{V}). \tag{4.2}$$

What an estimate of variables X_i is and what, after the optimization, this estimate should be, is incorporated within factor ϕ_i defined as

$$\phi_i(X_i) \propto \exp\{-\frac{1}{2} \|h_i(X_i) - z_i\|^2_{\Sigma_i}\}, \tag{4.3}$$

where h_i is a nonlinear model for variables X_i that should correspond to a real measurement z_i that has a zero-centered Gaussian noise with covariance Σ_i. Notice how (4.2) is *a nonlinear least-squares problem* that can be rewritten as

$$\begin{aligned} \mathcal{V}^* &= \arg\max_{\mathcal{V}} \prod_i \phi_i(X_i) && (4.4)\\ &= \arg\max_{\mathcal{V}} \sum_i \log(\phi_i(X_i)) && (4.5)\\ &= \arg\min_{\mathcal{V}} \sum_i \|h_i(X_i) - z_i\|^2_{\Sigma_i} && (4.6) \end{aligned}$$

and can be solved with methods like the Gauss-Newton algorithm [100] and the Levenberg-Marquardt algorithm [101].

The generalized notation of factor graphs (4.1) covers numerous usage scenarios, the simplest one being *tracking*. In tracking, we estimate the trajectory X given the measurements Z. More precisely, we model a conditional probability with

$$p(X \mid Z) \propto p(x_1)p(z_1|x_1) \prod_{i>1} p(x_i \mid x_{i-1})p(z_i \mid x_i), \tag{4.7}$$

where $p(x_1)$ is *the prior*, $p(x_i \mid x_{i-1})$ is *the motion model* and $p(z_i \mid x_i)$ is *the measurement model*. The factor graph for tracking has a topology depicted in Figure4.1a.

Then, a slight upgrade to the tracking problem is *a switching system* as depicted in Figure4.1b. In it, we distinguish between multiple variable types. This way a more complex probabilistic modeling is achieved – e.g., we can define a conditional probability $p(x_i \mid x_{i-1}, y_{i-1})$ where x_i is, alongside x_{i-1}, conditioned on y_{i-1} too. *Pose graph optimization* (abbr. PGO) (Figure4.1c) is an instance of the tracking problem where both variables and measurements that impose factor constraints (i.e., z_i in (4.3)) are members of *the special Euclidean group* $SE(n)$, $n \in \{2, 3\}$. Additionally, PGO in combination with other type of variables – landmarks – form *the simultaneous localization and mapping problem* (abbr. SLAM). We will address the theoretical aspect of SLAM more thoroughly in Section4.2, while additional, rather practical aspects of SLAM are considered in further sections of this chapter.

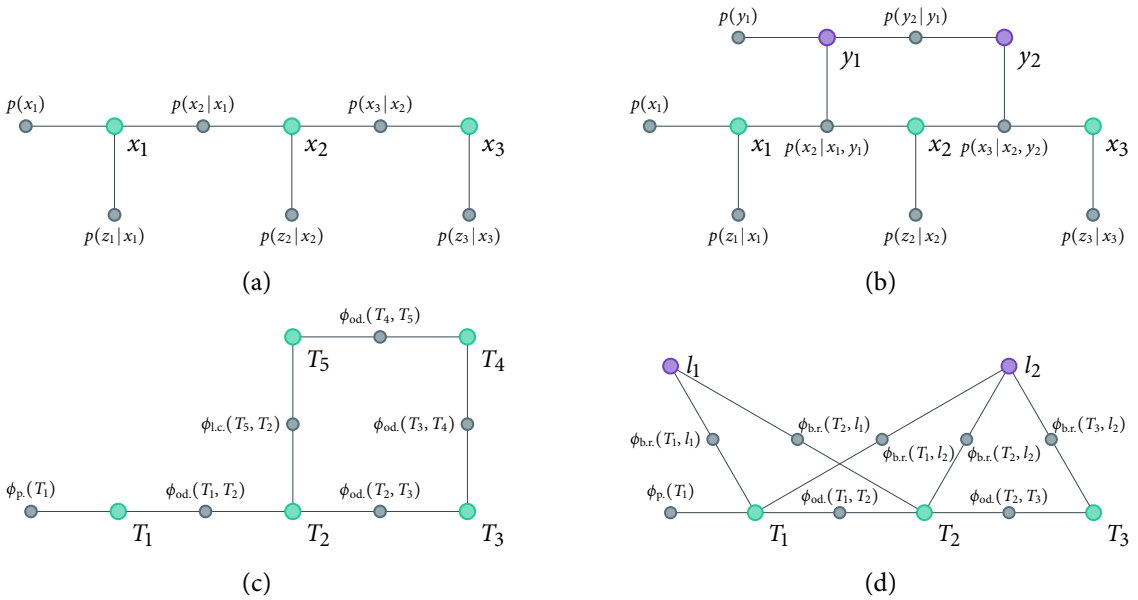


Figure 4.1: Factor graph examples. In a simple tracking graph (a) we have one type of variables connected via factors. The switching system shown in (b) is a generalization where multiple types of variables exist. Pose graph optimization (PGO) shown in (c) is a variant of tracking where factors are from $SE(n)$. A generalization to PGO is simultaneous localization and mapping (SLAM) shown in (d) where, alongside poses, landmarks are modeled too.

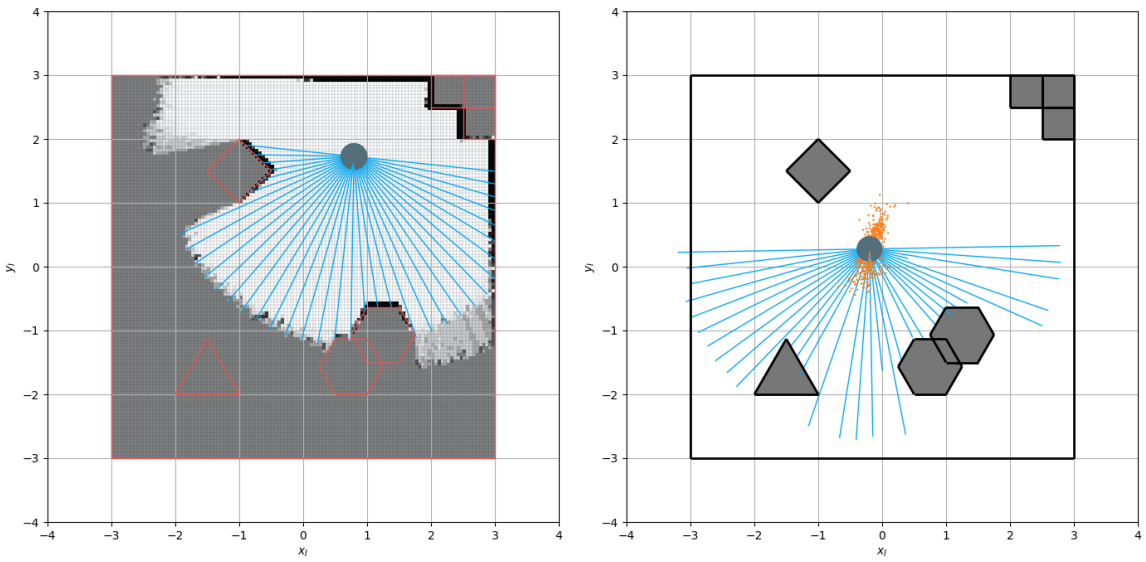
4.2 Simultaneous localization and mapping

Simultaneous localization and mapping is a robotic problem where, as the robot progresses through its environment, we would like to infer the robot's pose according to the most recent knowledge about the environment that is materialized in a map, and also, to upgrade that map if there are better pose estimates or new measurements. SLAM is one of the most challenging robotics problems. Truly, if there is no noise in measurements and motion, by means of forward kinematics, i.e., by mapping applied wheel velocities into the robot's local pose derivation and then accumulating it into the robot's global pose, localization would be completely trivial. Implicitly, the map would be trivially built by spatially transforming measured features of the environment (e.g., LiDAR scans) according to the perfectly restored poses. This realistically unattainable scenario is methodologically disassembled into two slightly harder problems: *mapping* (under the assumption of a perfect localization) and *localization* (under the assumption of a perfectly built map). Both of these are research problems on their own. Occupancy grid mapping [1, Chapter 6] is an approach to mapping where *a volumetric representation*[3] of a place is built. *Gaussian* and *Monte Carlo* (i.e., particle filter) localization [1, Chapters 7 & 8] are two popular localization approaches. We can combine various mapping and localization approaches in order to define a SLAM system.

Historical overview and classification of different SLAM paradigms can be found in [102, Chapter 46], where a SLAM system can be classified into three classes listed below:

- an extended Kalman filter-based SLAM,

[3] Which means that each point in the space has its label.



(a) (b)

Figure 4.2: Mapping and localization in a simulated environment where the ground truth map has red borders. (a) Occupancy grid mapping under the assumption of perfect localization. (b) Monte Carlo, i.e., *a particle filter*-based localization under the assumption of a perfectly built map. Sensor beams may seem incorrect because errors defined in *the beam model for range finder* [1, p. 153] were sampled and deliberately added to actual measurements.

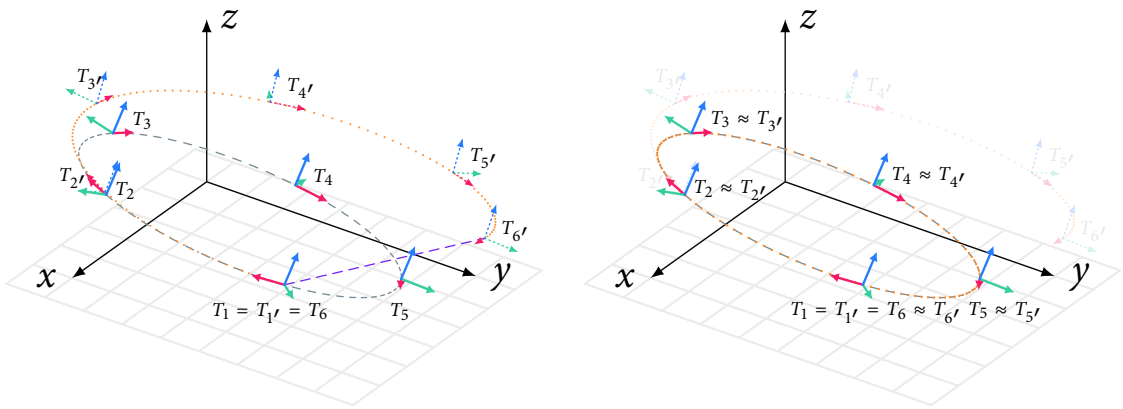
- a particle filter-based SLAM,
- a factor graph-based SLAM.

The first two types are a generalization of respective localization-only approaches (Gaussian and Monte Carlo localization) – e.g., in the particle filter-based FastSLAM [103] both poses and landmarks are modeled via particles. Due to versatility and convenience that factor graphs offer for stochastic optimization, much of which can be attributed to excellent open-source factor graphs software libraries such as GTSAM[104], factor graph-based approaches are prevailing today.

4.2.1 *Odometry*

From a factor-graphs point of view, pose graph optimization is a SLAM subset. Within SLAM, pose graph optimization is the way we should maintain localization. As can be said that SLAM = Localization + Mapping, it can be said that PGO = Odometry + Loop Closing. Before we explain how to incorporate loop closing with odometry, which is essentially the main contribution of this chapter, we will briefly address odometry. Odometry is composed of the Greek words ὁδός (pronounced as "odos", eng. path) and μέτρον (pronounced as "metron", eng. measure). Thus, odometry is the process that, in the form of a specific quantity (e.g., the angular velocity of a wheel or translational and angular velocity in 3D), tells how much of a route has been crossed and at which rate. Technically speaking, an odometry sensor gives us the rate of change (i.e., the derivative) of a path crossed rather than an accumulated quantity. This imposes a serious flaw as we have to integrate all *to-some-extent-precise* rates of change, meaning that errors will be integrated as time elapses.

No matter how good a particular odometry sensor is, the deviation from an actual pose – *the odometry drift* – will show up eventually. As expected, better odometry sensors yield a lower drift, and vice versa, with lower quality odometry sensors, e.g., a pair of cheap wheel encoders, drift is more pronounced. Drift, in general, is illustrated in Figure4.3a. Say an aerial vehicle moves along a regular circular path from the pose $T_1 \in SE(3)$. Moreover, let us assume it is known that the initial pose estimation $T_{1'} \in SE(3)$ is aligned with T_1 and that the vehicle will actually end up in the pose $T_6 \in SE(3)$ aligned with T_1. The first odometry measurement, the transformation from T_1 to T_2, will bear a small odometry drift so that the estimated pose $T_{2'}$ has just slightly drifted from the actual position T_2. As the vehicle is moving further, more and more drift is going to be accumulated, hence, in the end, resulting in a notable error between the actual pose T_6 and its estimate $T_{6'} \in SE(3)$.



(a) (b)

Figure 4.3: Factor graph for *pose graph optimization* (a) before and (b) after loop closing. In (a) we see how the odometry drift increases with each new odometry measurement. Then, we can add a loop closing factor (the purple dashed line) and figuratively think of it as an elastic bond that will, after the optimization takes place, reconfigure variables more closely to the actual state as illustrated in (b).

Without odometry, no pose graph optimization is possible, i.e., a loop-closing-only system would be useless as its outcome would not be further used. This cannot be said for an odometry-only system, as its outcome, no matter how erroneous, is useful on its own and gives us spatial transformations. So let's examine how to incorporate odometry measurements into factor graphs before loop closing is incorporated too.

Usually, for the sake of simplicity, in wheeled mobile robotics, we model the pose (i.e., the position and the orientation) of a robot, as well as spatial transformations between poses, with a two-dimensional point $(x, y) \in \mathbb{R}^2$ for the position and a scalar value $\theta \in \mathbb{R}$ for the orientation. This can be compactly represented as *a homogeneous transformation matrix* $T \in SE(2) \subset \mathbb{R}^{3\times3}$. In three-dimensional space, we can represent poses and transformation between poses with a six-dimensional vector, although, *at-least-seven-dimensional* parametrization is used in order to avoid *singularities*. Singularity-free three-dimensional spatial poses and transformations between poses are represented with $T \in SE(3) \subset \mathbb{R}^{4\times4}$. Alternatively, we can use other *over-parametrized* representations equivalent to $SE(3)$, e.g., a three-dimensional spatial position vector in combination with either a four-dimensional

unit quaternion or a four-dimensional *angle-axis* orientation representation. For a detailed examination confer [102] or [105, Chapter 3]. Therefore, in an odometry-at-least factor graph, a factor node $\phi_{\text{od.}}(T_i, T_{i+1})$ with an actual measurement $z_{i,i+1} \in SE(n)$ models the odometry between successive poses $T_i \in SE(n)$ and $T_{i+1} \in SE(n)$.

Odometry sensor modalities are chosen according to the application. In planar[4] mobile robotics, either wheel encoders and/or *an inertial measurement unit* (abbr. IMU) and/or two-dimensional range sensor can be used. In a three-dimensional space, IMUs and range sensors are used too (e.g., *the iterative closest point algorithm* (abbr. ICP) [106], as used by [107, 108], can find relative pose transformations between range scans). On the other side of the sensor modalities spectrum, we have *visual odometry* (abbr. VO) accomplished via a stream of mono/stereo/RGB-D images (e.g., *Direct Sparse Odometry* (abbr. DSO) [33], and the neural network-based DeepVO [109] are mono approaches to VO, while among prominent stereo approaches are SOFT2 [35] and VISO2 [110]). It is also possible to use a combination of odometry types (e.g., V-LOAM[111] is a visual and LiDAR-based odometry system while a comprehensive overview of *visual-inertial* odometry can be found in [112]).

4.2.2 *Loop closing*

As already ascertained, more and more error will be accumulated as the vehicle moves forward. Luckily, there exists a way this error could be diminished – by loop closing. To "close a loop" means to recognize a previously seen place, find the relative transformation between a currently seen and a previously seen place, add a loop closing factor in a factor graph and then the optimization (4.2) will do the rest. This is illustrated in Figure4.3. First, at the currently estimated pose $T_{6'}$, the loop closing system recognizes an already seen place associated with $T_{1'}$. Then, either with visual odometry or ICP, we find the relative transformation between $T_{6'}$ and $T_{1'}$ and add it as a loop closing factor $\phi_{\text{l.c.}}(T_{6'}, T_{1'})$. We can figuratively think of $\phi_{\text{l.c.}}(T_{6'}, T_{1'})$ as an elastic bond[5] (the dashed purple line in Figure4.3a) that will, after the optimization takes place, force the estimated poses closer to the true poses so that $T_i \approx T_{i'}$, $\forall i$ (Figure4.3b). Technically, a loop closing factor $\phi_{\text{l.c.}}$ is indistinguishable from an odometry factor $\phi_{\text{od.}}$ - i.e., both measurements for respective factors are from $SE(n)$ and both share the same model h_i from (4.3). In terms of programming language implementation, both factors are instances of the same *object-oriented programming* class [6]. In fact, the only thing that distinguishes these two is the temporal order of variables within a factor – for an odometry factor, it is the i-th and the $(i+1)$-th variable, while for loop closing, it is the j-th and the i-th variable.

In contrast to the systematic categorization of odometry approaches, it is not that easy to categorize loop closing approaches (except for, to the best of our knowledge, sequence-based visual place recognition approaches). An individual approach is, presumably, constructed so that it fits a SLAM system's constraints, e.g.,ORB-SLAM2 [114] is a state-of-the-art approach to mono/stereo/RGB-D SLAM that uses *oriented FAST and rotated BRIEF* (ORB) features

[4] i.e., modeled in a two-dimensional space.

[5] The idea for this phrase came from [113] where a loop closing constraint is called "a rubber bar".

[6] At least in GTSAM [104] where `gtsam::BetweenFactor<gtsam::Pose2>` is a class for factors with $SE(2)$ measurements and `gtsam::BetweenFactor<gtsam::Pose3>` is a class for factors with $SE(3)$ measurements.

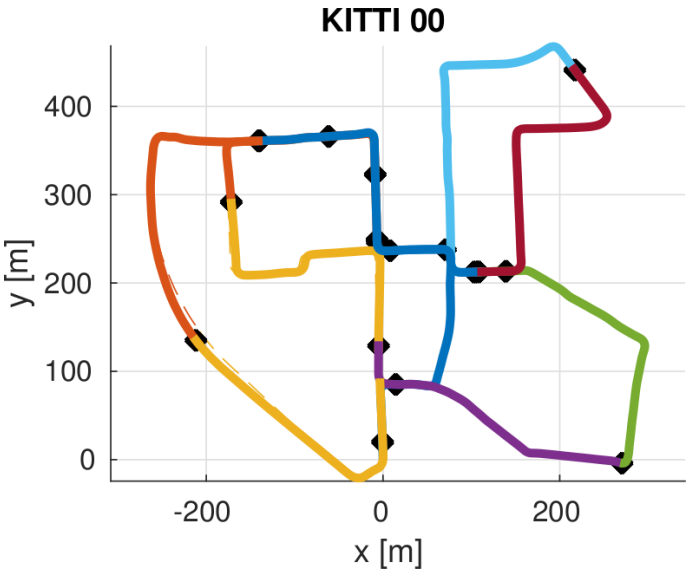


Figure 4.4: In Data-Efficient Decentralized Visual SLAM multiple robots perform visual odometry on their own. Then, these locally built pose graphs are merged via a NetVLAD-based place recognition into a single global graph. Image is taken from [117].

[115] as its local image features. Its loop closing detection system is built atop DBoW2 [116], also an ORB-based place recognition system from the same research group. On the other hand, *Data-Efficient Decentralized Visual SLAM* [117] is an interesting and eclectic use-case of loop closing in a decentralized visual SLAM where multiple robots perform visual odometry each. A NetVLAD-based [81] place recognition system then finds matches, i.e., closes loops, between locally obtained (i.e., obtained by a single robot) graphs into a single global graph as depicted in Figure4.4.

4.2.3 *Mapping*

A map that consists of landmarks in the environment is built dynamically. Not only it means that new landmarks, once the robot senses them, are here to stay, but also, landmarks can dynamically be updated according to their relations with poses. The relation between landmarks and poses is many-to-many. A landmark can be visible from multiple poses, and inversely, in a particular pose, multiple landmarks can be observed. These relations are modeled with another type of factor – *a bearing-range factor* $\varphi_{\text{b.r.}}$ (Figure4.1d). It stochastically models the orientation (bearing) and the Euclidean distance (range) between a pose $T_i \in SE(n)$ and a landmark $l_j \in \mathbb{R}^n$. How these factors and landmarks are added into a SLAM system is application-specific.

The better the localization, the better the mapping, and vice versa. However, notice that landmarks, regarded as a factor graph's variables, are not connected to loop closing factors. Rather, loop closing factors interconnect pose variables only. In terms of both definition and implementation, this means that a loop closing subsystem should be invariant of a mapping subsystem.

4.3 Adapting NOSeqSLAM for loop closing in SLAM

By far, the NOSeqSLAM method presented in Chapter2worked in an offline fashion exclusively as an *l'art pour l'art* visual place recognition system. It has been evaluated on typical "visual place recognition" datasets, and its performance we have measured quantitatively with the standard VPR measures – the area under a curve and the recall at 100% precision. In such a setup we had two different sequences of images – $\mathcal{Q}$ and $\mathcal{D}$. The questions are: how to adapt this pipeline so it fits in a SLAM system and how to quantitatively measure its performance?

The latter question is rather easy to answer. The better the quantitative performance of a SLAM system that is being evaluated, the better its loop closing subsystem is. This specifically means that a particular SLAM system evaluated on a dataset that has loops, under the assumption that loops are detected correctly, should perform better when odometry alongside loop closing is used than its odometry-only subsystem. On the other hand, even a single wrongly detected loop totally corrupts estimated poses – a SLAM system with wrongly detected loops has a subpar estimation performance when compared to an odometry-only estimation. Simply put, it is expected that an odometry + loop closing system should achieve better estimation than an odometry-only system.

There exist two standard SLAM performance measures: *the absolute trajectory error* (abbr. ATE) and *the relative error* (abbr. RE) [118]. Through our experiments in this chapter, we will use the absolute trajectory error. Given ground truth poses $T_i \in SE(n),\ \forall i = 1 \dots N$, i.e., their corresponding rotation matrices[7] and position vectors $(R_i, p_i) \in SO(n) \times \mathbb{R}^n,\ \forall i$, and aligned[8] estimated poses $T_{i'} \in SE(n),\ \forall i = 1 \dots N$, i.e., $(R_{i'}, p_{i'}) \in SO(n) \times \mathbb{R}^n,\ \forall i$, we measure the position part $\mathrm{ATE}_{\mathrm{pos.}}$ of the absolute trajectory error as

$$\mathrm{ATE}_{\mathrm{pos.}} = \left(\frac{1}{N} \sum_{i=1}^{N} \left\| \Delta p_i \right\|^2 \right)^{\frac{1}{2}} = \left(\frac{1}{N} \sum_{i=1}^{N} \left\| p_i - p_{i'} \right\|^2 \right)^{\frac{1}{2}} \tag{4.8}$$

and the orientation part $\mathrm{ATE}_{\mathrm{rot.}}$ of the absolute trajectory error as

$$\mathrm{ATE}_{\mathrm{rot.}} = \left(\frac{1}{N} \sum_{i=1}^{N} \left\| \measuredangle \left(\Delta R_i \right) \right\|^2 \right)^{\frac{1}{2}} = \left(\frac{1}{N} \sum_{i=1}^{N} \left\| \measuredangle \left(R_i R_{i'}^T \right) \right\|^2 \right)^{\frac{1}{2}}, \tag{4.9}$$

where $\measuredangle\,(\cdot)$ maps the rotation matrix to an angle of the angle-axis representation.

4.3.1 *Problem-specific adaptation*

When performing SLAM, there is a single stream of images and/or a single stream of range scans. Let us denote this generic stream of measurements with $\mathcal{M}$. The most recent

[7] Just as transformation matrices are members of $SE(n)$, rotation matrices are members of *the special orthogonal group* $SO(n) \subset \mathbb{R}^{n \times n},\ n \in \{2, 3\}$. This guarantees that the product of multiplying two rotation matrices, hence members of $SO(n)$, will be a member of $SO(n)$ too. Also, because of orthogonality, a rotation matrix's inverse is its transpose. For more information, confer [105, Chapter 3].

[8] As claimed by [118], either all estimated poses or only one/few initial poses can be aligned with ground truth poses. A popular method for trajectory alignment is defined by Umeyama [119]. In the first experiment in Section4.4, the initial estimated pose is aligned with the ground truth, while for the experiment in Section 4.5we use Umeyama's method.

measurement should correspond to what "a query image" means in the context of visual place recognition. Additionally, the same reasoning can be applied to range-sensor measurements without the presence of visual sensor modalities, meaning it also makes sense to consider not only visual place recognition, but also place recognition in general. The reason for this is simple – just as an image I can be represented with a specific n-dimensional global descriptor $z \in \mathbb{R}^n$, so can a range scan s be represented with $z \in \mathbb{R}^n$, and so can any measurement m be represented with $z \in \mathbb{R}^n$. Therefore, the expression "a query measurement" denotes a generic measurement we would like to match.

As the newest measurement $m_i \in \mathcal{M}$ is obtained, we would like to detect whether there is a loop. Notice how two consecutive measurements are largely similar. Even a few consecutive measurements should be similar by a large amount. Also, it is difficult to expect a loop will occur after only a few consecutive measurements, e.g., a typical situation for loop closing in SLAM would be to once again arrive at the crossroad of a circular residential area and a substantial amount of measurements will be obtained on such a detour until we arrive at the same crossroad. These facts motivate us to introduce an additional place matching hyperparameter: α. It is the number that tells how many of the most recent measurements, excluding m_i, are not going to be compared with q_i for place matching [9] . We dubbed α *the dismissal rate*. So, for a fixed query measurement m_i, we will not consider α previously recorded measurements, meaning a particular m_i will have

$$\mathcal{D}_i = \{m_0, m_1, \ldots, m_{i-\alpha-2}, m_{i-\alpha-1}\} \subset \mathcal{M} \tag{4.10}$$

as its reference dataset. It would be redundant to introduce the query dataset notation because in the current scenario, there are no two different-in-time-and-appearance streams of images. Hence we only reason about the newest measurement of a place m_i and how to compare it with $\mathcal{D}_i$ in order to recognize an already experienced place and, in turn, to close the loop. However, this does not mean that $m_{i-1}, m_{i-2}, m_{i-3} \ldots$ will be useless. Recall how, in order to incorporate sequentiality of data, we used a query image $I_{q_i} \in \mathcal{Q}$ and its temporal neighbors and compare them with $I_{d_j} \in \mathcal{D}$ and its temporal neighbors.

If we perform SLAM online, we will not have future measurements at disposal[10]. So if m_i is the most recent measurement, i.e., $\mathcal{M} = \{m_0, m_1, \ldots, m_{i-1}, m_i\}$, temporal neighborhood of m_i are previous measurements $m_{i-1}, m_{i-2}, m_{i-3} \ldots$ In this fashion we will construct *the online NOSeqSLAM* method. In the offline NOSeqSLAM, for a fixed pair of images $(I_{q_i}, I_{d_j}) \in \mathcal{Q} \times \mathcal{D}$, we built the left DAG $G^l_{q_i,d_j}$ and the right DAG $G^r_{q_i,d_j}$. $G^l_{q_i,d_j}$ models a similarity between pair (I_{q_i}, I_{d_j}) and similarities between I_{q_i}'s temporal predecessors and I_{d_j}'s temporal predecessors. Similarly, $G^r_{q_i,d_j}$ models a similarity of a fixed pair and temporal successors. This means that in the online NOSeqSLAM, we only build the left DAG. From now on, we will refer to this left DAG as "DAG" and omit the l superscript. Moreover, as we do not distinguish between $\mathcal{Q}$ and $\mathcal{D}$, subscripts i, j are sufficient (instead of q_i, d_j).

In accordance with the new notation, let $m_i \in \mathcal{M}$ denote the most recent measurement and let $\mathcal{D}_i$ denote its reference dataset. To measure the correspondence between m_i and $m_j \in \mathcal{D}_i$, we first build a DAG $G_{i,j} = (\mathcal{V}_{i,j}, \mathcal{E}_{i,j})$. Its root node $u_{i,j}$ is the parent to η nodes $\{u_{i-1,j}, \ldots, u_{i-1,j-\eta+1}\}$. Each node $u_{i-1,j'}$ is the parent to η nodes $\{u_{i-2,j'}, \ldots, u_{i-2,j'-\eta+1}\}$. We

[9] α is the first letter of the Greek word ἀπόπεμπτος (pronounced as "apopemptos", eng. dismissed).

[10] Unless we perform *time series forecasting*, which can be considered as a subject for future research.

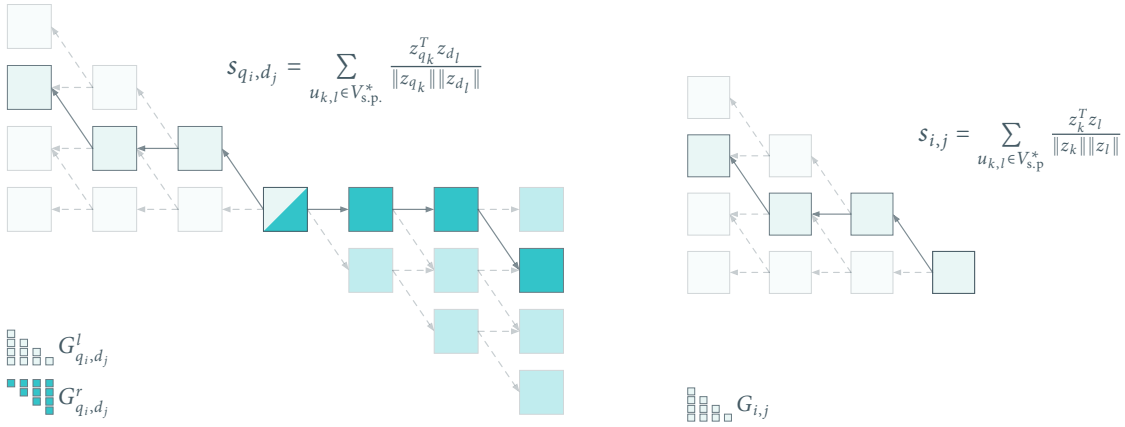
repeat this building procedure recursively until the depth of $G_{i,j}$ is $d_s - 1$. Note how such depth accompanied with the root node will yield a sequence of length $d_s - 1 + 1 = d_s$, just as, in the offline NOSeqSLAM, the union on the left DAG and the right DAG, both of depth $\lfloor \frac{d_s}{2} \rfloor$, will yield a sequence of length $\lfloor \frac{d_s}{2} \rfloor + \lfloor \frac{d_s}{2} \rfloor + 1 = d_s$[11]. The weight function $w_{i,j} : \mathcal{E}_{i,j} \to [0,1]$ for $G_{i,j}$ is also defined as in the case of the offline NOSeqSLAM, i.e.

$$w_{i,j}((u_{k,l}, u_{m,n})) = 1 - \frac{z_m^T z_n}{\|z_m\| \|z_n\|}. \tag{4.11}$$

Then the correspondence is measured as

$$s_{i,j} = \sum_{u_{k,l} \in V^*_{\text{s.p.}}} \frac{z_k^T z_l}{\|z_k\| \|z_l\|} \tag{4.12}$$

where $V^*_{\text{s.p.}}$ are vertices of the shortest among all shortest paths from the root to leaves of $G_{i,j}$. The way the online NOSeqSLAM differs from the offline NOSeqSLAM is illustrated in Fig.4.5. All shortest paths from the root can be found by using the Algorithm11. The difference between this algorithm and its offline variant is that the boundaries now also include the dismissal rate α while here, we only move in the direction of previous temporal measurements ($i' = i - i_{\text{offset}}$ and $j' = j - j_{\text{offset}}$).



(a) (b)

Figure 4.5: (a) In the offline NOSeqSLAM method, we build the left directed acyclic subgraph $G^l_{q_i,d_j}$ and the right directed acyclic subgraph $G^r_{q_i,d_j}$. $G^l_{q_i,d_j}$ encodes similarities between previously seen places, while $G^r_{q_i,d_j}$ encodes similarities of places yet to be seen w.r.t. $(I_{q_i}, I_{d_j}) \in \mathcal{Q} \times \mathcal{D}$. (b) It is impossible for an online system to foresee future places' measurements (either images or range scans), so we model the online place recognition via NOSeqSLAM by using only a fixed pair of measurements (m_i, m_j) and their respective previous temporal neighbors which means the single graph $G_{i,j}$ will be built.

4.3.2 *Optimization of the execution time*

Recall how the topological sort-based implementation, long ago discarded, took more than an hour of execution time to perform an offline VPR experiment on a tiny dataset

[11]Presumably, in the offline NOSeqSLAM, d_s is an odd number, so such equation holds.

Algorithm 11 On-the-fly relaxation for the online NOSeqSLAM

Input: $G_{i,j} = (\mathcal{V}_{i,j}, \mathcal{E}_{i,j})$, d_s, η, α
Output: all shortest paths from the root $u_{i,j}$ to leaves
for $i_{\text{offset}} = 1$ to d_s **do**
  for $j_{\text{offset}} = 0$ to $i_{\text{offset}} \cdot (\eta - 1)$ **do**
    $i' = i - i_{\text{offset}}$
    $j' = j - j_{\text{offset}}$
    if $i' > j' + \alpha$ and $i' \geq 0$ and $j' \geq 0$ **then**
      for each $u_{\tilde{i},\tilde{j}} \in \text{PREDECESSORS}(u_{i',j'})$ **do**
        $\text{RELAX}((u_{\tilde{i},\tilde{j}}, u_{i',j'}))$
      end for
    end if
  end for
end for

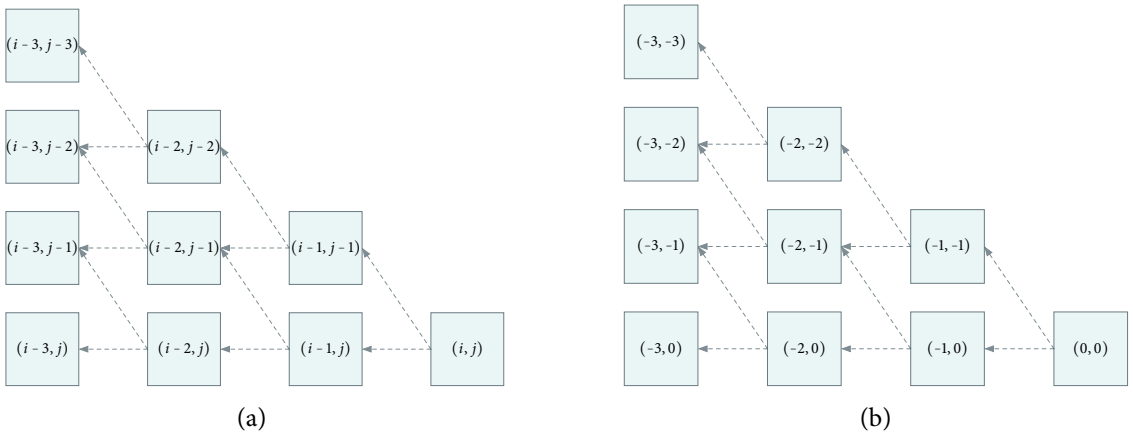
Algorithm 12 The online NOSeqSLAM

Input: the most recent $m_i \in \mathcal{M}$, d_s, η, α
Output: $s_{i,j}$, $\forall\, m_j \in \mathcal{D}_i$
for each $m_j \in \mathcal{D}_i$ **do**
  Construct $G_{i,j}$
  Calculate $V^*_{\text{s.p.}}$ with Algorithm11
  $s_{i,j} = \sum_{u_{k,l} \in V^*_{\text{s.p}}} \frac{z_k^T z_l}{\|z_k\| \|z_l\|}$
end for

comprising $|\mathcal{Q}| = 544$ and $|\mathcal{D}| = 488$ images. Since then, execution times have significantly improved with Algorithm4, enabling us to perform a vast amount of offline experiments. It turns out, neither this implementation succeeds at providing a real-time performance when a SLAM evaluation dataset $\mathcal{M}$ has a far larger number of images, e.g., the KITTI dataset [10] sequence 00 with 4541 images for a single camera.

The key for the execution time optimization lies in the paradigm of *array programming*, also called *vectorization*. Vectorization refers to the way software executes. Given a data array, we would like to perform a specific operation on the entire array at once (instead of performing the operation on a single datum). Colloquially said, by vectorizing a code, we replace loops and single-datum operations in loops with vectors, matrices, tensors, and appropriate linear algebra operations. To an end user, programming languages/software libraries that support array programming should provide as efficient as possible execution of implemented functionalities.

The goal of the Algorithm11is to calculate all shortest paths from the root of $G_{i,j}$ to its leaves. In the achievement of vectorization, and hence, reduced execution time, the first step was to get rid of the following fashion of iterations: "for each i_{offset}, for each j_{offset}, do …". Moreover, this was done in a loop for each pair comprising a new query measurement



(a) (b)

Figure 4.6: (a) DAGs used in NOSeqSLAM are topologically sorted – from left to right i indices increase and, from top to bottom, j indices increase. (b) For fixed parameters d_s and η, each DAG will have the same topological structure. This means that a generic DAG with relative indices can be built in advance.

m_i and $m_j \in \mathcal{D}_i$, which is, fundamentally, the online NOSeqSLAM (Algorithm12). By exploiting the fact that all online-NOSeqSLAM DAGs, for fixed d_s and η, have the same topological structure – examine in Fig.4.6ahow DAG is sorted topologically from left to right (i indices increase) and from top to bottom (j indices increase) – we can, in advance, create a generic DAG (Fig.4.6b) with relative indices, add a number i to the first component and add a number j to the second component of relative indices, and this way obtain a DAG $G_{i,j}$. The procedure similar to the Algorithm11is used in order to build such a topologically sorted graph with relative indices.

The next step towards vectorization is to advancedly construct all paths from the root to leaves, which is done in the Algorithm13, and put them as rows in the matrix $P_1 \in \mathbb{R}^{n\times(d_s-1)}$. In contrast to P_1 which holds dissimilarities as edge weights, the matrix $P_2 = J_{n\times(d_s-1)} - P_1 \in \mathbb{R}^{n\times(d_s-1)}$ holds similarities as weights, where $J_{n\times(d_s-1)}$ is the $n \times (d_s - 1)$-dimensional all-ones matrix. Then, the rows of P_2 are summed, so the summation yields the vector $p \in \mathbb{R}^n$ that holds accumulated similarities in its components. The maximal among such components is $s_{i,j}$ - a value used as the matching score between m_i and $m_j \in \mathcal{D}_i$. In our implementation, we also vectorized this even further so that $s_{i,j}$, $\forall m_j \in \mathcal{D}_i$ is calculated at once. The trick is to build the tensor $\mathbf{P}_1 \in \mathbb{R}^{|D_i|\times n\times(d_s-1)}$ that has $n \times (d_s - 1)$-dimensional matrices of paths on its first axis, then to build $\mathbf{P}_2 \in \mathbb{R}^{|D_i|\times n\times(d_s-1)}$, etc.

And lastly, before we move to the experiments in Sections4.4and4.5, empirical execution times of an online place matching will be analyzed. We evaluated the conventionally implemented NOSeqSLAM, the vectorized NOSeqSLAM implementations deployed either on CPU or GPU, and the vectorized naive matching implementations deployed either on CPU or GPU. We tweaked the NOSeqSLAM's hyperparameter $d_s \in \{4, 6, 8, 11\}$ while $\eta = 2$ and $\alpha = 85$ remained fixed. For the evaluation dataset, we used the KITTI sequence 00 with 4541 images per camera (only a single stream of images is needed, so we picked the left RGB camera). Images, originally of size 1241×376, are mapped via the original ResNet50 architecture [65] into feature maps and flattened into 100352-dimensional vectors. This procedure takes (16.59 ± 0.80) ms per single image when the model is deployed on GPU,

Algorithm 13 Construct Paths

Input: $G_{i,j} = (\mathcal{V}_{i,j}, \mathcal{E}_{i,j})$, paths, path, e_{current}, e_{previous}, d_s
Output: all paths from the root $u_{i,j}$ to leaves
procedure ConstructPaths($G_{i,j}$, paths, path, e_{current}, e_{previous}, d_s)
  if GoesFromTheRoot(e_{current}) **then**
    path = $[e_{\text{current}}]$ ▷ path is a recently created list with e_{current} as its only element
  else
    if StartingNode(e_{current}) = EndingNode(e_{previous}) **then**
      Append(path, e_{current})
    else
      return
    end if
  end if
  if Length(path) = $d_s - 1$ **then** ▷ If there is $d_s - 1$ edges, i.e., d_s nodes
    Append(paths, path)
    return
  end if
  for each $e_{\text{next}} \in \{e \in \mathcal{E}_{i,j} : e$ starts from the ending node of $e_{\text{c.}}\}$ **do**
    path_copy ← DeepCopy(path)
    ConstructPaths($G_{i,j}$, paths, path_copy, e_{next}, e_{current}, d_s)
  end for
end procedure

and this time is also accounted in the results. Note how each image should be mapped via DCNN only once, as we can store its feature maps if there is enough memory. As we iterate through the dataset, an image of the current iteration, considered the most recent measurement m_i, is being mapped into feature maps z_i, and then we try to match it with images in $\mathcal{D}_i$. With each iteration, the reference dataset increases, which means it takes more and more time to find a loop.

In Table4.1and Fig.4.7we see how the conventional implementation, once considered very efficient, performs poorly when compared against its vectorized variants. At the beginning of an online place matching experiment – when there are only few reference images – the conventional NOSeqSLAM, the CPU-deployed vectorized NOSeqSLAM and the CPU-deployed naive matching behave similarly. It is not surprising that, due to NOSeqSLAM computational complexity, the execution time of the conventional implementation increases as the number of to-be-compared reference images increases. Still, surprisingly enough, the CPU(GPU)-deployed NOSeqSLAM can stand the pace of the CPU(GPU)-deployed naive matching at smaller sequence lengths. This means that the vectorization was completed successfully. By looking in Table4.1, the maximum time it takes for NOSeqSLAM to find a loop is 35.04 ms, 40.77 ms, 92.76 ms and 760.66 ms, respectively. The first three times are feasible and will not be a runtime bottleneck in a multithreaded online SLAM system with an adequate data acquisition frequency. For example, the data acquisition frequency for the

Table 4.1: Execution time statistics between the naive matching and the online NOSeqSLAM for $d_s \in \{4, 6, 8, 11\}$ on the KITTI 00 sequence with 4541 images. The conventional, i.e., non-vectorized implementation compared with vectorized implementations that run on CPU and GPU, respectively.

d_s	Setup	Min. [ms]	Max. [ms]	Mean [ms]	Total [s]
4	NOSeqSLAM, conventional	66.54	1399.18	587.98	2670.03
	NOSeqSLAM, vectorized, CPU	66.04	100.81	91.03	413.35
	NOSeqSLAM, vectorized, GPU	**20.59**	**35.04**	**22.86**	**103.81**
6	NOSeqSLAM, conventional	65.36	2275.19	1153.11	5236.27
	NOSeqSLAM, vectorized, CPU	65.68	109.35	93.52	424.69
	NOSeqSLAM, vectorized, GPU	**20.39**	**40.77**	**26.76**	**121.54**
8	NOSeqSLAM, conventional	66.04	3921.22	1917.58	8707.72
	NOSeqSLAM, vectorized, CPU	68.48	157.47	119.16	541.12
	NOSeqSLAM, vectorized, GPU	**20.26**	**92.76**	**52.41**	**238.01**
11	NOSeqSLAM, conventional	67.24	7015.14	3438.76	15615.41
	NOSeqSLAM, vectorized, CPU	66.42	833.47	433.44	1968.25
	NOSeqSLAM, vectorized, GPU	**20.45**	**760.66**	**362.59**	**1646.51**
-	Naive, vectorized, CPU	65.12	99.73	90.24	409.80
	Naive, vectorized, GPU	**19.88**	**27.34**	**21.68**	**98.43**

KITTI dataset is 10 Hz, which means, until the next measurement, there will be enough time for NOSeqSLAM with not especially long sequence lengths to find loop closure candidates.

4.4 LiDAR-based SLAM in a 2D simulated environment

This is the first of two experiments where NOSeqSLAM is used for loop closing in simultaneous localization and mapping. As noticeable from the title of this section, this experiment is not a visual one with a visual sensor modality. Instead, we have a range sensor-based measurements in the two-dimensional Euclidean space. And even though the evaluation of this experiment was successful, as will be presented, we cannot justify the third scientific contribution of the thesis: "Procedure for adapting sequence-based visual place recognition method for loop closing in simultaneous localization and mapping algorithms". In this section, NOSeqSLAM is a range-based place recognition method rather than a visual place recognition method.

Extrinsic motivation for this experiment was to have a prototype for the next one. Then again, by using NOSeqSLAM in non-visual setups adds an additional value to its capability. Recall from Subsection4.2.1how there are various sensor modalities for odometry. If we are in the two-dimensional Euclidean space, the odometry output will be in $SE(2)$, and in the three-dimensional Euclidean space, it will be in $SE(3)$. The way we work with elements in $SE(2)$ is comparable in certain aspects to the way we work with elements in $SE(3)$. At the end of the day, any odometry – be it an ICP-based odometry in this experiment or visual odometry in the next experiment – will provide us with relative transformations between poses that will ultimately be used for factors in factor graphs. The same applies to loop closing. Somehow we have to detect an already experienced place – either by having range

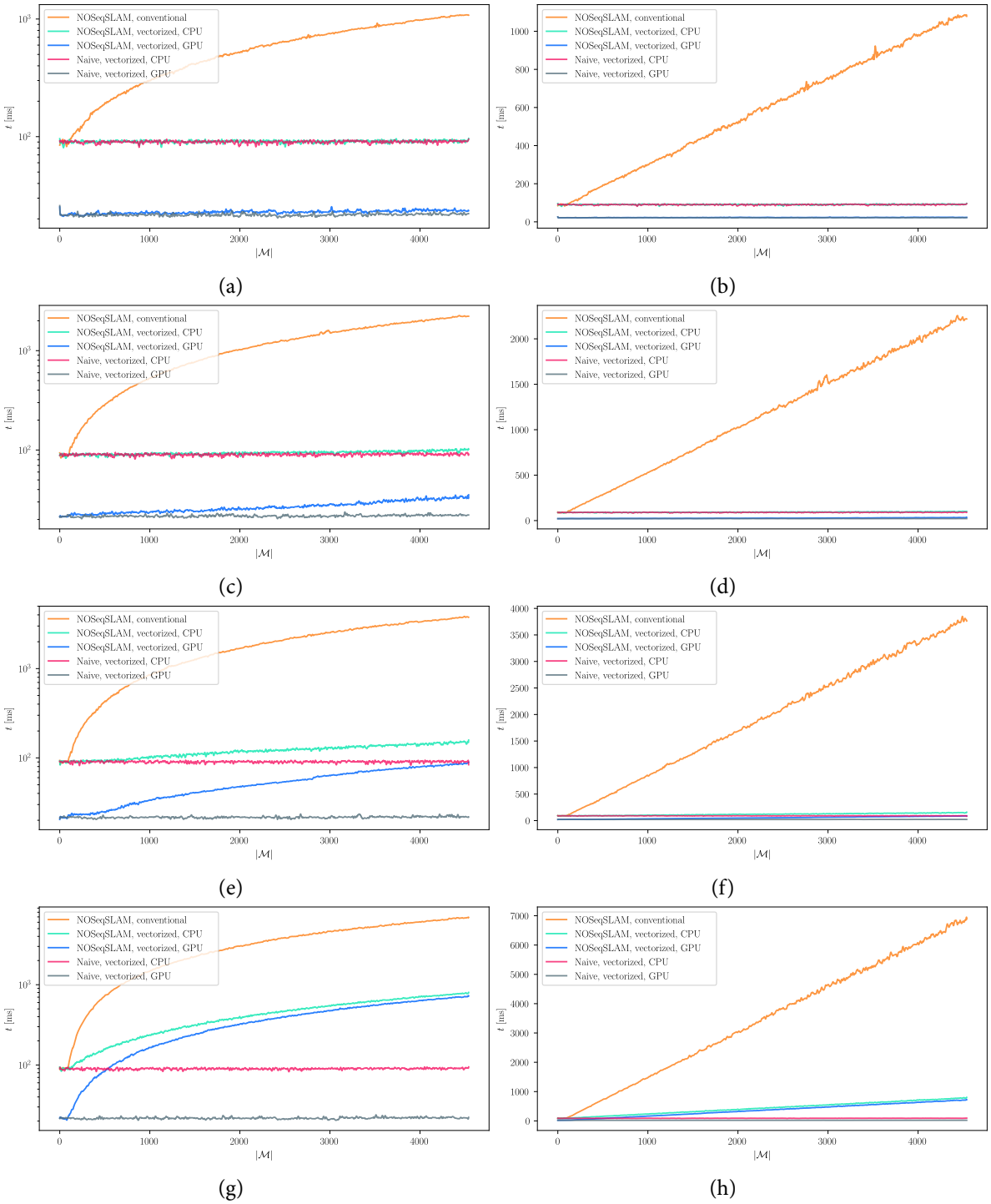


Figure 4.7: Comparison of execution time in the logarithmic y axis scale (left column) and the linear y axis scale (right column) between the naive matching and the online NOSeqSLAM for $d_s = 4$ (a, b), $d_s = 6$ (c, d), $d_s = 8$ (e, f) and $d_s = 11$ (g, h). The conventional, i.e., non-vectorized implementation, is compared with vectorized implementations that run on CPU and GPU, respectively.

scans of an environment or by having images. Luckily, NOSeqSLAM is invariant to a sensor modality and can be deployed in both scenarios. Once a loop closure is detected, no matter how, the relative transformation of matched measurements is obtained with "no matter what" odometry and is included as a loop closing factor.

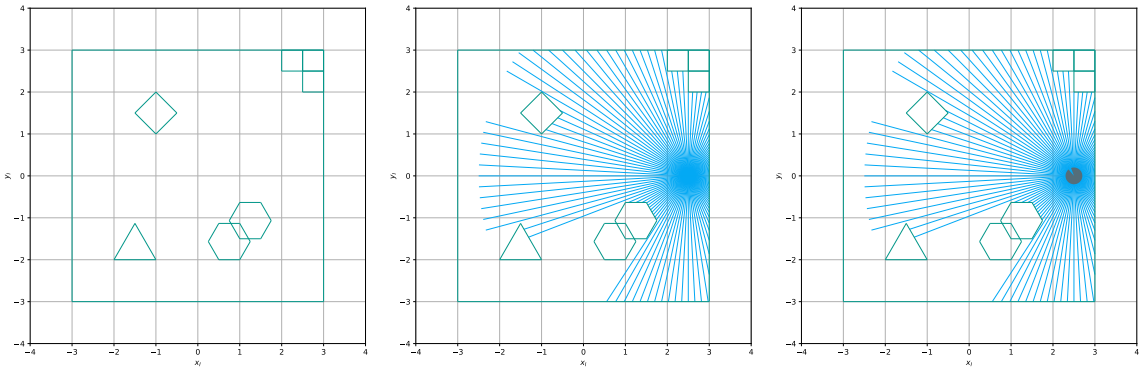
By being in a simulated environment, it is easy to specify the simulated robot – its wheel radius, wheelbase lengths, maximum velocities for motors, *cycles per revolution* (abbr. CPR) for wheel encoders, etc. Moreover, it is easy to obtain a ground truth trajectory by computing the kinematics. It is easy to deliberately corrupt various simulation parts in order to make an estimation more difficult so that a simulation is more real. As the environment has to be specified somewhere, we know the ground truth state of it. In conclusion, every single detail in a simulation of a real-time and real-life physical process can be tweaked and we will have such a real-time process within our grasp – a real-time ground truth trajectory and real-time measurements of a ground truth environment.

For the purpose of this experiment, we have developed the SLAM system in Python dubbed SLAM № 1. Python is perfect for rapid prototyping of robotics applications – it is easy to code given a pseudocode, there is a lot of libraries, just modified code can start quickly, etc. However, it is not that great at runtime unless compute-intensive tasks are outsourced to a "C under-the-hood" implementation. The experience and knowledge we had been acquiring by developing SLAM № 1 had greatly accelerated the development of the visual SLAM – dubbed SLAM № 2 – that was developed in C++ for the purpose of the upcoming experiment in Section4.5.

4.4.1 *SLAM № 1*

SLAM № 1 is a simulated robotic environment with the addition of pose graph optimization. The majority of code is written in NumPy [120] as it provides a blazingly fast C performance of linear algebra functionalities in Python. In order to find relative transformations between scans, iterative closest point library libpointmatcher [121] has been used. To build a factor graph for pose graph optimization, GTSAM [104] has been used. Both of these libraries support NumPy's `array` structure; therefore, it was not that complicated to communicate between NumPy, libpointmatcher and GTSAM. NumPy API is exploited in a great extent resulting in a code that lacks `for` and `while` loops in Python, except for a few loops during the initialization of the vectorized NOSeqSLAM, e.g., for the Algorithm13. It would not be fair not to mention matplotlib [122], the de facto standard for scientific plotting, whose animation API, alongside the standard set of functionalities is used for visualization. The absolute trajectory error results (Table4.2) and position errors with respect to the distance traveled plots (Figures4.10,4.11) were obtained with the rpg_trajectory_evaluation package [123].

↬ WORLD.The world in this experiment is a two-dimensional $3\,\mathrm{m} \times 3\,\mathrm{m}$ room comprising a few convex shapes: three squares, a rectangle, a triangle, and two hexagons (Fig.4.8a). These shapes, as well as the boundaries of the inner space, are specified as linear segments that are placed in a single tensor. The robot can sense the inner space with its simulated range sensor.



(a) (b) (c)

Figure 4.8: In SLAM № 1, (a) we simulate the world as a 2-dimensional planar space, (b) then add sensors measurements where the simulated range sensor is placed in the center of the simulated robot (c).

RANGE SENSOR & MEASUREMENTS. From a specified 2D point, i.e., from the sensor center, we radially distribute beams every $\beta_{\text{step}}°$ starting from $\beta_{\text{start}}°$ to $\beta_{\text{end}}°$ (e.g., in Figures4.8b,4.8c: $\beta_{\text{step}} = 3°$, $\beta_{\text{start}} = 0°$ and $\beta_{\text{start}} = 360°$). Each beam has its maximum and minimum range. Start and end points of a beam, expressed in the local sensor's frame, make a segment and all segments are placed in a single tensor. All beams together make a scan.

As beams of the simulated range sensor are of the same type as planar shape sides – segments – it is easy to calculate their intersections via the *line-line intersection* formula. Given two line segments L_1 and L_2, represented as linear Bézier curves

$$L_1 = (x_1, y_1) + t(x_2 - x_1, y_2 - y_1),\ t \in [0,1], \tag{4.13}$$

$$L_2 = (x_3, y_3) + u(x_4 - x_3, y_4 - y_3),\ u \in [0,1], \tag{4.14}$$

we find their intersection by testing whether

$$0 \leq t^* = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \leq 1 \tag{4.15}$$

and

$$0 \leq u^* = \frac{(x_1 - x_3)(y_1 - y_2) - (y_1 - y_3)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \leq 1. \tag{4.16}$$

If these conditions hold, then the intersection between L_1 and L_2 is

$$(x^*, y^*) = (x_1 + t^*(x_2 - x_1), y_1 + t^*(y_2 - y_1)) = (x_3 + u^*(x_4 - x_3), y_3 + u^*(y_4 - y_3)). \tag{4.17}$$

We check for intersections between sensor beams and lines in the environment each to each, which can be done efficiently in a vectorized way. The minimal distance from the starting point of a beam to a point it intersects with a line segment of the environment is the "measured" distance for that beam.

In each iteration, the currently measured scan is stored as an n-dimensional vector, i.e., as a NumPy `array`. Each such scan is associated with the corresponding estimated pose of the robot at the time it is captured. It is important to store scans for ICP odometry and loop closing. Also, scans that are shown in the map indicate whether the estimation process

had succeeded. For this purpose we developed an efficient vectorized procedure that will instantly move scans when pose graph variables are perturbed. Although it is possible to include *a culling policy* that will discard redundant scanned points as time passes, we do not cull any point in order to keep things simple.

ROBOT. The robot in SLAM N º 1 is a two-wheel differential-drive robot with the wheel radius $r_{\text{wheel}} = 5\,\text{cm}$ and the wheelbase length $l = 10\,\text{cm}$. Local coordinate systems of the robot and the sensor are aligned, so no additional transformation should be performed in order to express the sensor in the robot's local frame. The absolute angular velocity of wheels is bounded from above with $\dot{\varphi}_{\max} = \frac{2\pi \cdot 100}{60}\,\frac{\text{rad}}{\text{s}}$. The robot has a circular frame of the specified radius $r_{\text{frame}} = 20\,\text{cm}$ so if it is detected that the distance between the robot and an obstacle is less than or equal to this radius, a collision occurs[12] . We run the robot for 3000 iterations with the simulation interval time of $\Delta t_1 = \frac{1}{30}\,\text{s}$.

Wheel encoders are simulated in the following way. Let $c_{\text{p.r.}}$ denote the specified number of cycles per revolution for an encoder. If $\dot{\varphi}$ is an angular velocity we apply to an imaginary motor, we can calculate the difference of the current and previous number of the encoder's cycles, and this way simulate the encoder as

$$\Delta c = c_k - c_{k-1} = \left\lfloor \frac{\dot{\varphi} \cdot \Delta t \cdot c_{\text{p.r.}}}{2\pi} \right\rfloor \tag{4.18}$$

where Δt is the time between two iterations. Then, a measured angular velocity will be

$$\tilde{\dot{\varphi}} = \frac{\Delta c \cdot 2\pi}{\Delta t \cdot c_{\text{p.r.}}}\,\frac{\text{rad}}{\text{s}}. \tag{4.19}$$

The simulated "measured" angular velocities of the left and the right wheel, $\tilde{\dot{\varphi}}_{\text{left}}$ and $\tilde{\dot{\varphi}}_{\text{right}}$, are mapped via forward kinematics to the robot's local frame approximated velocity $\tilde{\dot{\xi}}_R$ then mapped to the robot's approximated velocity in the global frame $\tilde{\dot{\xi}}_I$, which is used in order to keep the currently estimated pose $\tilde{\xi}_{I,k}$ up to date, i.e.,

$$\tilde{\xi}_{I,k} = \tilde{\xi}_{I,k-1} + \tilde{\dot{\xi}}_I \cdot \Delta t. \tag{4.20}$$

The pose obtained with wheel odometry can further be improved with the ICP-based odometry and loop closing. Additionally, in our experiments, we have specified two different wheel encoder types:

- *good*: $c_{\text{p.r.}} = 4096$, drift $\epsilon \sim \mathcal{N}(0, 0.05)$ added to $\tilde{\dot{\varphi}}$,
- *bad*: $c_{\text{p.r.}} = 300$, drift $\epsilon \sim \mathcal{N}(0.01, 0.05)$ added to $\tilde{\dot{\varphi}}$,

where, as expected, good encoders ensure significantly lower error. We also checked multiple drift hyperparameters by having $c_{\text{p.r.}} = 300$ fixed, but it was $c_{\text{p.r.}}$ that actually affects the performance.

[12] A more sophisticated and CPU intensive way to detect collisions is to use a tree structure *quadtree* [124]. However, this sophistication plays no role in our evaluation.

PATH PLANNING. The path planning system is modeled after the no longer available Coursera course "Control of Mobile Robots" by Magnus Egerstedt based on his research [125]. The idea is to specify the robot's behavior as *a finite-state automaton* that is going to act according to its current state. In our setup, we distinguish between three simple states of the automaton, the robot:

1. "moves towards a goal",
2. "avoids obstacles",
3. "stands still".

Its inputs, measured values, dictate the next state. So if the robot "moves towards a goal" and an obstacle is spotted near, e.g., the minimal measured distance is sufficiently small, it switches to the state of obstacle avoidance. While it "avoids obstacles", a collision could occur, so the robot "stands still". Also, it can be sensed that an obstacle is no longer a threat, and the robot just "moves towards a goal". The robot will "stand still" once when it reaches a single goal, or in the case of multiple goals, "moves towards a (next) goal".

Obstacles are all inner shapes and boundaries of the room, therefore, all imaginary walls in the simulated space. The reason we have obstacles at all is, not to try out whether the robot can avoid them, but to make the space as distinguishable as possible. The reason we have multiple successive goals is to reach an already measured and distinguishable-due-to-obstacles scene once again so that a place recognition system can be tried out. Goals are two-dimensional points, one after the other, $(-2.5, -2.5)$, $(2.5, -2.5)$, $(2, 2)$, $(-2.5, 2.5)$, once again $(-2.5, -2.5)$, once again $(2.5, -2.5)$ and once again $(2.0, 2.0)$. The robot should successively reach every goal until there are no goals left. As it moves forward, obstacles should be avoided. To achieve these two tasks, we have implemented two algorithms: *the go-to-goal algorithm* and *the avoid-obstacles algorithm*. These are by no means state-of-the-art algorithms, and neither they work perfectly – there are situations when a goal cannot be reached and when a robot crashes into an obstacle. However, these algorithms just serve their purpose in SLAM № 1 resulting in runs with no collisions and with every goal being reached.

The algorithm that drives the robot toward a goal, the go-to-goal algorithm, is based on a proportional–integral–derivative (abbr. PID) controller. Let $(x_k, y_k, \theta_k) = \tilde{\xi}_{I,k}$ denote the robot's current pose and (x_g, y_g) denote the position of a goal. The robot will head toward a goal if its desired orientation is

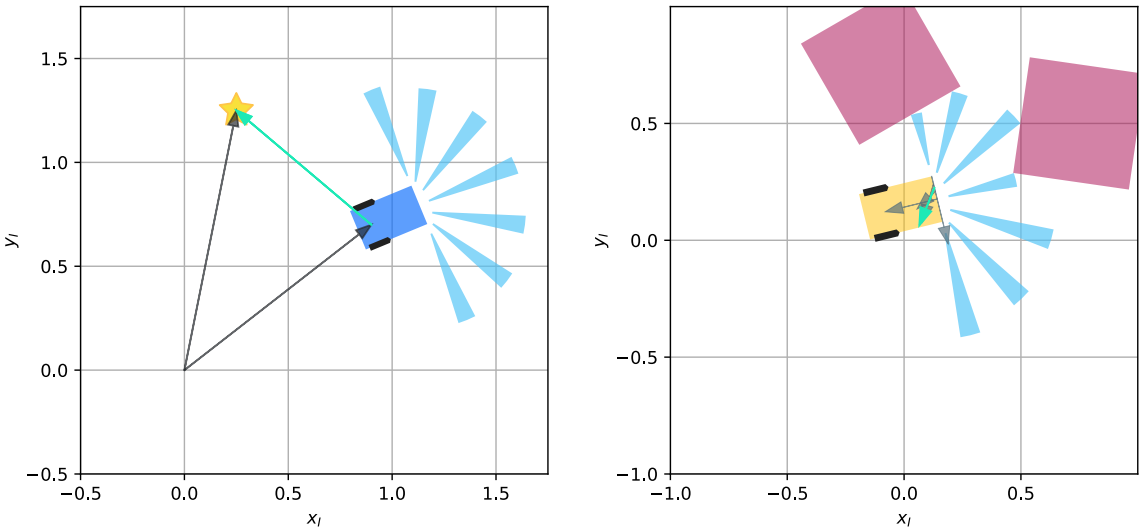
$$\theta^* = \operatorname{atan2}(y_g - y_k, x_g - x_k) \tag{4.21}$$

as depicted in Figure4.9a. In reality, its current orientation is θ_k so we model the error as

$$e = \theta^* - \theta_k. \tag{4.22}$$

The PID controller will produce the required orientation velocity $\dot{\theta}$ as

$$\dot{\theta}(t) = k_P e(t) + k_I \int_0^t e(\tau) d\tau + k_D \frac{de(t)}{dt}. \tag{4.23}$$



(a) (b)

Figure 4.9: (a) The PID controller based go-to-goal algorithm in SLAM № 1. The position of the goal and the robot itself is used to calculate a vector (green) the robot should follow in order to reach the goal. (b) The PID controller-based obstacle-avoidance algorithm in SLAM № 1. All beams that actually sense something are used to find a vector with the direction the robot should head to.

Then, according to the robot's kinematics and $\dot{\varphi}_{\max}$, simple geometric manipulations are performed in order to obtain the maximum translational velocity $\dot{x}$. Put together, these values define a velocity in the robot's local frame $\dot{\xi}_R = (\dot{x}, 0, \dot{\theta})$ which is further mapped via *inverse kinematics* to the wheel velocities $\dot{\varphi}_{\text{left}}$ and $\dot{\varphi}_{\text{right}}$ we would like to apply.

The pipeline for the avoid-obstacles algorithm is identical up to a way we model the error. The desired orientation we would like the robot to head to is the average of vectors depicted in Fig.4.9b. If there are beams that actually sense something, i.e., their measured values are smaller than the maximum $d_{\max}$ minus some constant ϵ_d, we consider such beams *active*. For an active beam b_i, we create a vector from its origin to its end point that has just been measured, then we find a vector (gray vectors in Fig.4.9b) with the opposite direction and with the magnitude of the maximum range $d_{\max}$ subtracted by a measured value d_i. All these vectors are averaged into a vector (a green vector in Fig.4.9b) with the direction θ^* we would like to obtain with the robot and this way to avoid an obstacle.

ITERATIVE CLOSEST POINT. The libpointmatcher library [121] has a twofold role in SLAM № 1. First, it is used for ICP odometry between an i-th and an $(i+1)$-th scan. Second, it is used to obtain the relative transformation between the most recent i-th scan and a previous j-th scan from a reference dataset $\mathcal{D}_i$ if a loop is detected. We have two instances of the class `PointMatcher.ICPSequence` in Python – one for odometry and one for relative transformations on loop closing. Each instance's hyperparameters are tuned in their own way so they fit the specified task. The outcome of both processes are homogeneous transformations ${}^{i}T_{i+1} \in SE(2)$ and ${}^{i}T_{j} \in SE(2)$, respectively.

Depending on the quality of wheel odometry, i.e., whether selected wheel encoders are *good* or *bad*, ICP odometry can either improve ATE results (when bad encoders are used)

or deteriorate ATE results (when good encoders are used). In fact, the preliminary results were on par for both wheel encoder types if ICP is used; therefore, we decided not to use ICP odometry, and instead rely on wheel odometry only, when good wheel encoders are used. If ICP odometry is used, a transformation obtained via wheel odometry serves as an initial guess for ICP odometry. The quality of odometry does not affect loop closing in any way. A loop is detected no matter if the robot's pose estimation is precise or not – we just compare scans and not their associated poses. However, when a loop is detected, relative transformation is needed too, so in this case, ICP is critical, and without it, we could not add a loop closing factor. An initial guess for the transformation of a loop closure is obtained from the factor graph's current state of variables.

↬ PLACE MATCHING.Place matching is used in order to detect loops. We match places either naively or by using the online NOSeqSLAM. A place is represented with its range scan $s_i \in \mathbb{R}^{n_1}$. Although it is easy to further map s_i with a neural network into a tensor of more discriminative feature maps $z_i \in \mathbb{R}^{n_2}$ [126], it was not essential in our experiments because the bare scan representation also succeeded to represent a place discriminatively enough. As the most recent scan s_i is retrieved, it is normalized, stored with all scans retrieved by far, and in a single matrix operation it is multiplied with scans in $\mathcal{D}_i$ that are stacked as rows. Then we run a place matching procedure which outcome are Boolean variables that say whether s_i matches with $s_j \in \mathcal{D}_i$, $\forall j$.

↬ POSE GRAPH OPTIMIZATION.By having all the aforesaid pieces put together, pose graph optimization can take place. iSAM2 [127] is used as an algorithm for sparse nonlinear incremental optimization with the relinearization threshold set to $\beta = 0.1$. The prior factor deviation $\Sigma_{\text{p.}}$, odometry factor deviations $\Sigma_{\text{od.}}$ and loop closing factor deviations $\Sigma_{\text{l.c.}}$ are experimentally set to

$$\Sigma_{\text{p.}} = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & \pi/36 \end{bmatrix}, \quad \Sigma_{\text{od.}} = \Sigma_{\text{l.c.}} = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & \pi/36 \end{bmatrix}. \tag{4.24}$$

We assume that the initial pose $T_{1'}$ is known and equals to the ground truth initial pose T_1 in order to avoid trajectory alignment. Moreover, this way we can clearly see how the drift is accumulated over the time and how loop closing exterminate it. The first variable $T_{1'}$ (counting from 1)[13] and the prior factor $\phi_{\text{p.}}(T_{1'})$ are set to the pose graph before the robot starts to move. Measurements are taken on every 10-th iteration, which also entails that variables, odometry factors and potential loop closing factors are added to the factor graph then. In the end of a run, both estimated and ground truth trajectories will consist of 300 poses (excluding the initial pose). During each "measurement" iteration i (counting from 0), a variable $T_{(i+2)'}$ and a factor $\phi_{\text{od.}}(T_{(i+1)'}, T_{(i+2)'})$ are added to the pose graph.

If the place matching subsystem says there is a match between an i-th a j-th place, i.e., a loop is detected, and ICP finds the transformation ${}^{i'}T_{j'}$, then a loop closing factor $\phi_{\text{l.c.}}(T_{i'}, T_{j'})$ is added to the graph. Right now, we perform pose graph optimization, i.e.,

[13] In GTSAM examples found inthe official repositiory, variables are indexed from 1.

Table 4.2: ATE for different SLAM № 1 setups.

Wheel encoders	Setup	$ATE_{pos.}$ [m]	$ATE_{rot.}$ [°]
bad	ICP odometry + NOSeqSLAM	**0.02836**	**0.59485**
	ICP odometry + Naive	0.07611	1.40690
	ICP odometry (w/o place matching)	0.25755	5.01363
	Wheel odometry (w/o place matching)	0.58594	12.19906
good	Wheel odometry + NOSeqSLAM	**0.01079**	**0.21707**
	Wheel odometry + Naive	0.02016	0.39933
	Wheel odometry (w/o place matching)	0.15541	3.15750

we call the `isam2.update` method, only when $\phi_{\text{l.c.}}$ is added, otherwise optimization does not have any effect. In other words, unless a loop closure is detected, we cannot expect results that are any better than the odometry-only results. For the sake of efficiency, it is also possible to reduce the frequency of update calls, say on every n-th occurrence of a loop, however in our scenario with 301 variables, 300 odometry factors and substantially less loop closing factors this was unnecessary.

4.4.2 *Experimental results*

The experiments are evaluated quantitatively with the absolute trajectory error for position and orientation. The correspondence measure (4.12) obtained with NOSeqSLAM is additionally divided by d_s so it is between 0 and 1. Correspondences obtained either naively or with NOSeqSLAM are then thresholded with τ. The maximal correspondence greater than or equal to τ is a loop closing candidate. To find the optimal hyperparameters, we run a grid search over the dismissal rate $\alpha \in \{50, 70, 90\}$, the threshold value $\tau \in \{0.9, 0.92, 0.94, 0.96, 0.98\}$, the sequence length $d_s \in \{6, 8, 11, 16\}$, while the NOSeqSLAM's $\eta = 2$ remained fixed. As already mentioned, we also have two different wheel encoder types– good and bad. We present the best-performing runs for each wheel encoder type, further categorized with respect to an odometry type and a loop closing type – wheel odometry (w/o place matching), ICP odometry (w/o place matching), ICP odometry + naive and ICP odometry + NOSeqSLAM. For each of these categories,we picked the best result and present it in Table4.2.

Given the bad wheel encoders, ICP halves an error. Then, with loop closing, error rates are decreased by a factor of 5 for the naive matching and by a factor of 10 for NOSeqSLAM. Similarly, given the good wheel encoders, the best odometry-only run performs subpar, while with loop closing, the performance is significantly better. We omit to use ICP odometry with good encoders because such obtained results are almost identical to the results obtained with bad encoders + ICP odometry. Additional insight about how different encoders and loop closings influence error can be found in Figure4.10for bad encoders and in Figure4.11 for good encoders, respectively. From top to bottom, setups are listed in the same order as in Table4.2. For each setup, on the left side, the position error (the difference between estimated and ground truth positions) is shown with respect to the distance traveled. On the right, it is the orientation error with respect to the distance traveled. The better an estimation is, the more straight its error curve is, and the error converges to 0.

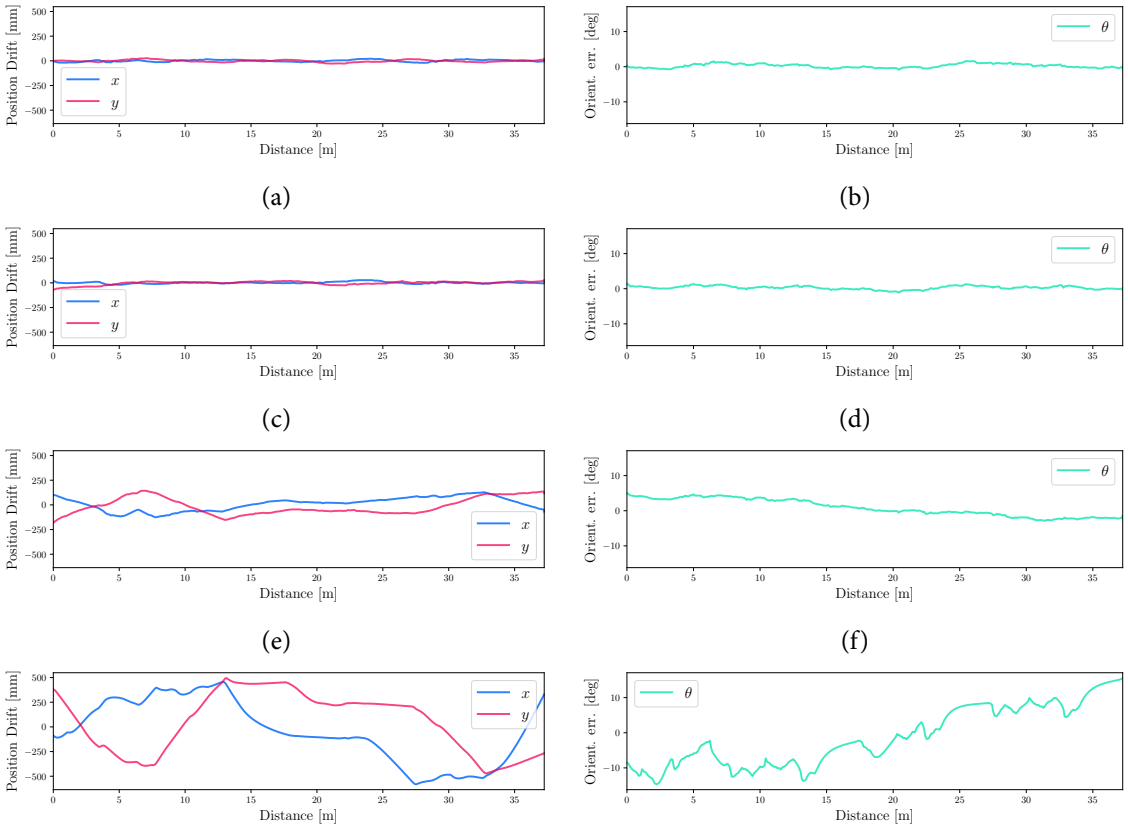


Figure 4.10: The position error (left) and the orientation error (right) with respect to the distance traveled for setups with bad encoders. The more straight an error curve is, the better. Plots created with [123].

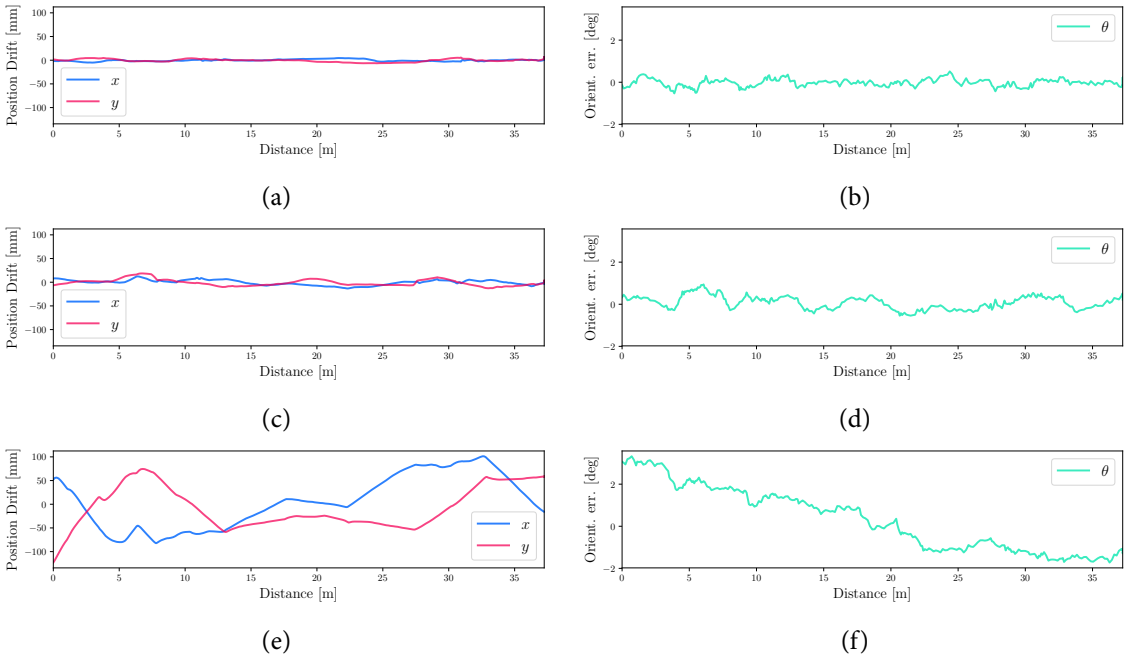


Figure 4.11: The position error (left) and the orientation error (right) with respect to the distance traveled for setups with good encoders. The more straight an error curve is, the better. Plots created with [123].

Qualitatively, performances can be observed in Figures4.12,4.13and4.14. If estimation is fine, an estimated state itself (the yellow transparent robot in plots) should be aligned with the ground truth robot (the gray robot in plots). Due to estimation error, blue range scan beams are what the ground truth robot senses, while gray range scan beams are what we think it senses. So, a good estimation implicates that a map that was being built during a run (red points) is more aligned with the ground truth map. Bad wheel odometry yields the worst estimated map (Figure4.12d). Things are much better if ICP is added then (Figure 4.12c). As mentioned previously in Section4.2, illustrated in Figure4.3, and right now, shown for the naive matching (Figure4.12b) and the online NOSeqSLAM (Figure4.12a), even if a substantial amount of drift exists, loop closing eradicates it.

In the bottom rows of Figures4.12,4.13and4.14, it is shown how loops are proposed in NOSeqSLAM (Figures4.12e,4.13dand4.14c) and how they are proposed naively (Figures4.12f,4.13eand4.14d) for their associated runs shown in top rows. The online NOSeqSLAM/the cosine similarity is evaluated between the pairs of all measurements by far. The intersection of an i-th column and a j-th row bears an association measure of the scans s_i and s_j. Those scans s_i with indices i less than α could not be matched anyway. Once there are enough scans, a reference dataset will become non-empty, and will increase by one in each iteration. Therefore, as we move between association matrix columns from left to right, each column has more unblacked cells. If a cell value is less than τ, its associated scans s_i and s_j are not a match, so s_j is not a loop closing candidate. We visualize it by additionally making these cells more dark. Then, we have cells that survived thresholding, and such cells are undarkened. In a column, among all such cells, we pick the one with the highest value. Such a cell is a loop closing candidate and is colored green.

We see how NOSeqSLAM is more "shrewd" with loop closing candidates. Also, when comparing Figure4.12ewith Figure4.12f, NOSeqSLAM proposals follow a straight linear segment as should be, because this is the way the robot moves – once again through an already seen environment with same velocities. We also included screenshots of bad runs with the naive matching (Figures4.14b,4.14d) and with NOSeqSLAM (Figures4.14a,4.14c). This is due to an inadequate, too low threshold τ. A lower threshold means that a lot of loop closure candidates can be proposed with a certain number of false matches. Conversely, a higher threshold means a smaller number of candidates, potentially none. Although neither NOSeqSLAM nor the naive matching in Figure4.14proposed correct loop closing candidates, the map obtained in the NOSeqSLAM run is still somewhat "structured".

4.5VISUAL SLAM IN A 3D REAL-LIFE ENVIRONMENT

In this experiment, we run a visual SLAM system in order to justify the third scientific contribution of the thesis. Instead, in a simulated environment, the SLAM system is evaluated on a real-world data – the KITTI Visual Odometry dataset. As being reported in [128], KITTI sequences 00, 02, 05, 06, 07, and 09 contain loops. Also, these sequences, along with other training sequences, contain ground truth trajectories. Therefore, we can quantify the performance of the system we have purposefully developed – SLAM № 2.

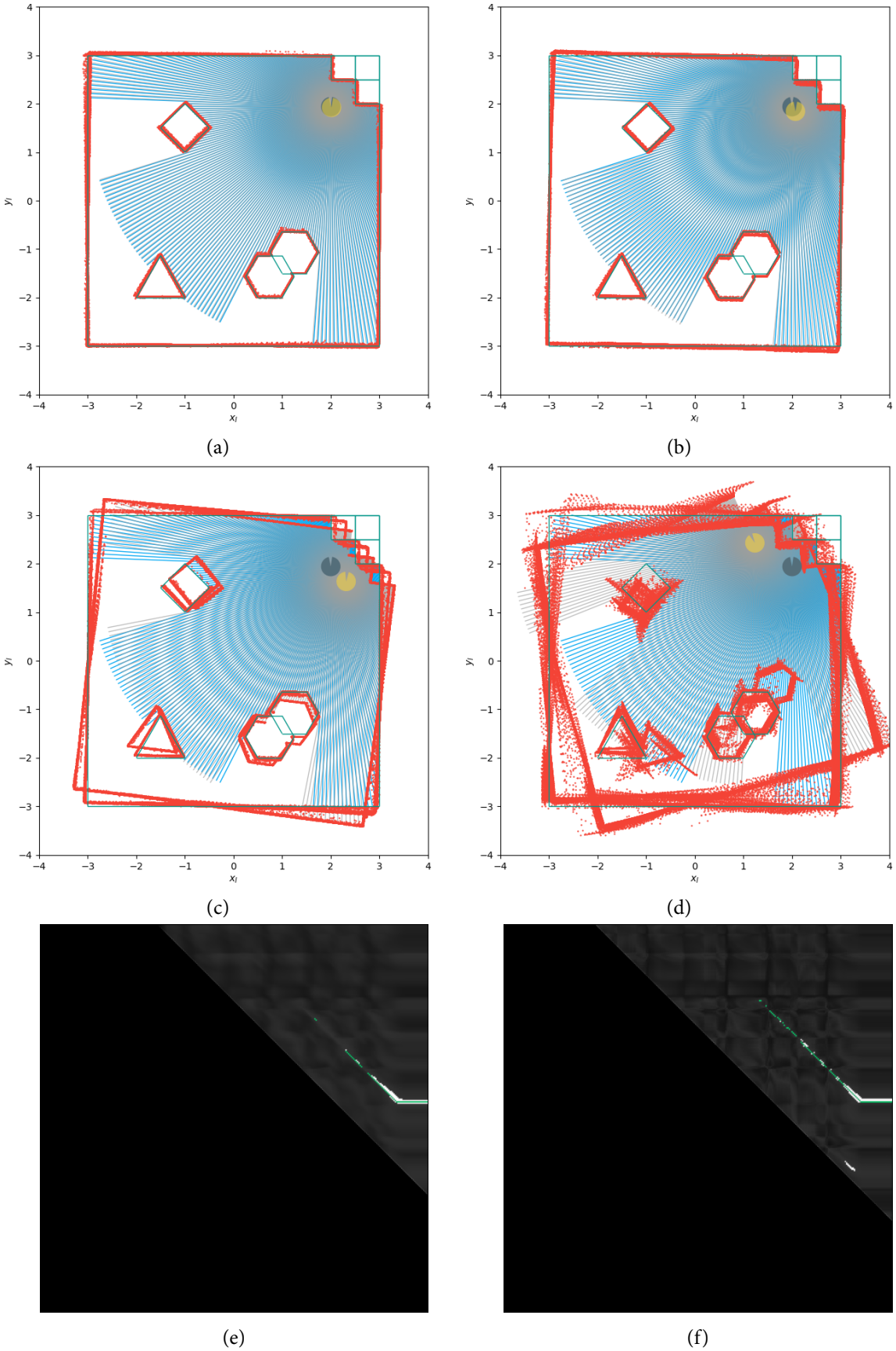


Figure 4.12: The best performing runs of different SLAM № 1 setups: (a), (e) ICP odometry + NOSeqSLAM, (b), (f) ICP odometry + naive matching, (c) ICP odometry (w/o place matching) and (d) wheel odometry (w/o place matching). The robot has bad encoders.

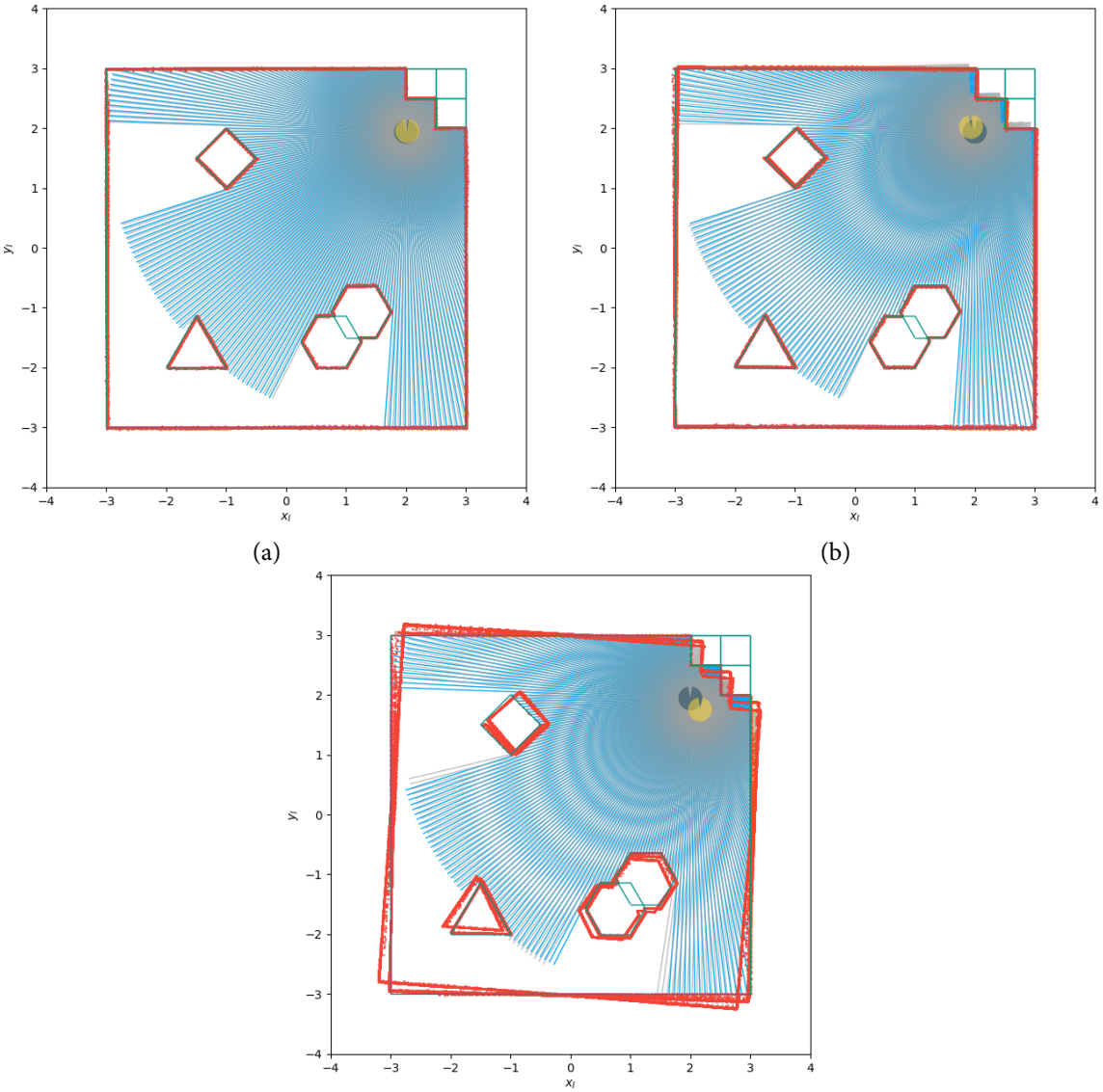


Figure 4.13: The best performing runs of different SLAM № 1 setups: (a), (d) wheel odometry + NOSeqSLAM, (b), (e) wheel odometry + naive matching and (c) wheel odometry (w/o place matching). The robot has good encoders.

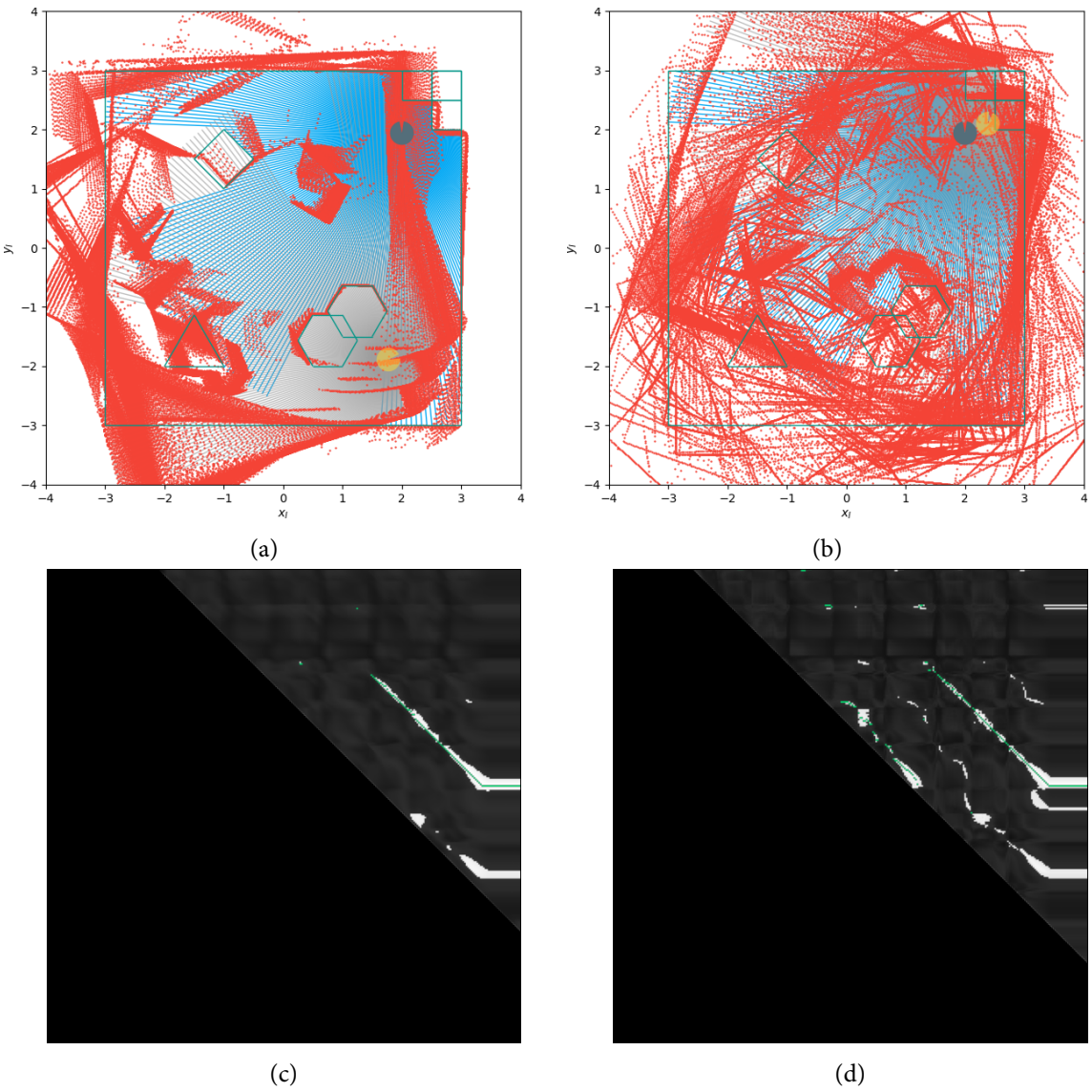


Figure 4.14: Unsuccessful runs with false loop closing proposals of (a), (c) NOSeqSLAM and (b), (d) the naive matching due to an inadequate threshold τ. Although both runs are inaccurate, the map obtained with the NOSeqSLAM setup is still more structured.

4.5.1 *SLAM № 2*

The C++-based SLAM № 2 is a visual SLAM that integrates VISO2 [34] as its visual odometry subsystem, GTSAM for pose graph optimization, and either NOSeqSLAM or the naive matching as the loop closing subsystem. Our own code, the loop closing library named liblc and the code that glues all the subsystems together, was written in libtorch, the C++ implementation of PyTorch [72]. Bearing in mind our further SLAM research, we made SLAM № 2 subsystems invariant enough between themselves, which means it is already possible to replace VISO2 with any other visual odometry, to replace GTSAM with another nonlinear optimization library, etc. The ATE results (Table4.3) and the plots of trajectories (Figures4.15) were obtained with the evo package [129] while position errors with respect to distance traveled plots (Figures4.16,4.17) were obtained with the rpg_trajectory_evaluation

package [123].

↬ VISUAL ODOMETRY.By exploring a few visual odometry implementations, we picked VISO2 as the primary visual odometry choice for SLAM № 2 due to its simplicity and due to the well written and easily integrable C++ library. VISO2 is a stereo approach to visual odometry with its specific local image features and their corresponding Sobel-based [130] descriptors. Given two pairs of stereo images, i.e., the current left and right and the previous left and right images, features are matched in a "circle" – first, a feature is matched between the current left and the previous left image, then between the previous left and the previous right, then between the previous right and the current right, and finally, between the current right and current left image. Let f_i denote a feature that is being matched between two consecutive frames. f_i's spatial position in the previous image can be mapped via *triangulation* into a 3D point X_i in the 3D Euclidean space. Given all N matched features between two stereo pairs, *egomotion estimation* is conducted by minimizing the sum of reprojection errors

$$\min_{T \in SE(3)} \sum_{i=1}^{N} \|x_i^{(l)} - \pi^{(l)}(X_i; T)\|^2 + \|x_i^{(r)} - \pi^{(r)}(X_i; T)\|^2, \tag{4.25}$$

where $\pi^{(l)}$ / $\pi^{(r)}$ maps a 3D point X_i into the current left/right image at position $x_i^{(l)}$ / $x_i^{(r)}$ given $T \in SE(3)$. Additionally, the obtained relative transformation is, as said by the authors, "refined" with the Kalman filter. As it was the case with iterative closest point, we use visual odometry both to obtain a relative transformation between two consecutive stereo image frames ${}^{i}T_{i+1} \in SE(3)$ and to obtain a transformation ${}^{i}T_j \in SE(3)$ between those frames that close a loop. Once again, transformations of both categories, as in SLAM № 1, are used to create factors in the factor graph.

↬ PLACE MATCHING.Although a stereo odometry pipeline requests a pair that consists of a left and a right image, left RGB images only (i.e., those images in the subfolder `image_2` of a given KITTI sequence) are used for visual place recognition. Place recognition takes place in the liblc library where we look for a match between the most recent image $I_i \in \mathcal{M}$ and the corresponding previously seen images $\mathcal{D}_i$ by comparing their global descriptors. As a new image is taken, liblc calculates its global descriptor, runs a place matching method, and returns a list of candidates according to specified place recognition hyperparameters, sorted in a downward direction by an association measure.

I_i is preprocessed by resizing a smaller size to 224 px, then by center-cropping it to a 224 px × 224 px image, and finally by normalizing its RGB pixel intensities as required for PyTorch Imagenet-trained image models. A preprocessed image $I_i \in \mathcal{M}$ is forward propagated through a DCNN model into feature maps that are additionally flattened into a single-axis n-dimensional vector $z_i \in \mathbb{R}^n$. The model is the trained ResNet50 without its fully-connected layers. Thanks to the TorchScript, a mechanism that allows for communication between PyTorch in Python and libtorch in C++, in further research it will be relatively easy to use any other deep model. Once a global descriptor z_i is obtained, it will be stored in the memory, so no additional forward propagations for an image I_i are required. It was already presented in Section4.3that forward-propagation is done either on a CPU or a GPU

and how fast it is done. Regarding the place matching implementation itself, it was easy to implement a C++ variant of the online NOSeqSLAM and the naive matching because these were already implemented in numpy for SLAM № 1. Required tensor manipulations are almost the same both in numpy and in libtorch, therefore, the SLAM № 2 C++ code for place matching has identical functions with identical signatures and the identical number of lines as the SLAM № 1 code.

POSE GRAPH OPTIMIZATION. By having the visual odometry and place matching subsystems established, the pose graph optimization subsystem, implemented with the C++ GTSAM, has all that is required for the 3D pose estimation from images taken by a vehicle. Once again, the iSAM2 algorithm is used for optimization. The prior factor deviation $\Sigma_{\text{p.}}$, odometry factor deviations $\Sigma_{\text{od.}}$ and loop closing factor deviations $\Sigma_{\text{l.c.}}$ are experimentally set to

$$\Sigma_{\text{p.}} = \Sigma_{\text{od.}} = \Sigma_{\text{l.c.}} = \begin{bmatrix} 1e-6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e-6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e-6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e-4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e-4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e-4 \end{bmatrix}. \tag{4.26}$$

Factors and variables in the factor graph are added the same way as in SLAM № 1.

4.5.2 *Experimental results*

Similarly to SLAM № 1, we compare the no-loop-closing VISO2, VISO2 in combination with the naive matching and VISO2 with NOSeqSLAM. In addition to a standard set of hyperparameters being tuned, we also tweaked "the occurrence frequency" of optimization, i.e., how often the `isam2.update` procedure is called, which means, in SLAM № 2, the pose graph optimization may occur:

1. at each loop closing,
2. at each 10^{th} loop closing + once at the end of a run, or,
3. once at the end of a run.

Regarding the standard hyperparameters, a grid search is performed over the threshold value $\tau \in \{0.15, 0.20, 0.22, 0.25, 0.28, 0.30, 0.32, 0.35, 0.38, 0.40\}$ and the dismissal rate $\alpha \in \{40, 80\}$, while the NOSeqSLAM hyperparameters remained fixed and set to $d_s = 6$ and $\eta = 2$.

For each KITTI sequence in Table4.3, VISO2 without loop closing performs worst. Then again, VISO2 is the backbone for SLAM № 2 and non-VISO2-only setups are surely largely dependent on its visual odometry. Both NOSeqSLAM and the naive matching perform on par, with a slight advantage for NOSeqSLAM. Also, plots of errors with respect to distance traveled (Figures4.16,4.17) suggest that loop closings reduce the drift. Qualitatively, the proof that loop closing contributed to accuracy, is shown in Figure4.15. VISO2 + NOSeqSLAM trajectories are totally aligned with the ground truth trajectories, while VISO2-only trajectories are affected by the drift. VISO2 + Naive trajectories were visually identical

to their VISO2 + NOSeqSLAM counterparts, so we have omitted them from the plots. A reason both of these place matching methods perform approximately equal may be the additional verification of a transformation in VISO2. Certainly, such "loops" are not added to the factor graph. The occurrence frequency of optimization does not discriminate results that much. Also, it is noticeable that the visual representation of choice, the ResNet50 feature maps, makes a fine distinction between different places. Otherwise, it would not be possible to close loops successfully, and this way, obtain good results.

On the other side of the spectrum, we expected to see bad runs due to suboptimal hyperparameters, just as in SLAM № 1. Although we were able to improve the performance by detecting true positive stereo image matches as just reported, it is also possible to have false positives that will completely ruin an estimated trajectory. As an example, in Figure4.18a, the correspondence between the images I_{375} and I_{529} from the KITTI sequence 06 obtained by the naive matching was above the specified threshold τ. It is visible how these two images are remarkably similar – the same supports for saplings on the left side of the lane, two almost identical cars on the right, both dark and both estates, same street lighting elements, similarly shaped roofs, etc. Not only is it not possible to detect whether this is a true positive in terms of place matching, but also, due to a substantial amount of common visual elements, VISO2 gives a relative longitudinal translation of a few meters when in reality, places are far away from one another. In Figure4.18b, we see how this disastrously affects estimation when incorrect loop closing factors are added.

4.6 Summary

The final chapter, in our opinion, the most challenging and, for the same reason, the most interesting, deals with the adaptation of our sequence-based place matching method for simultaneous localization and mapping. For this chapter, we purposefully built two different SLAM systems. Before SLAM № 1 and SLAM № 2 were developed, and their respective experiments were conducted, we were really concerned if NOSeqSLAM would be effective in this context at all. In any case, as visible from quantitative and qualitative evaluation as well as the running times study in Subsection4.3.2, the adaptation of place recognition methods for loop closing, both NOSeqSLAM and the naive matching, was successful, so we justified the third scientific contribution of the thesis: Procedure for adapting sequence-based visual place recognition method for loop closing in simultaneous localization and mapping algorithms. For further research, it would be interesting to see how different visual odometries affect results. Also, in addition to other image models we would examine, it would also be interesting to see how traditional handcrafted image models perform and whether results between different place matching methods would become more pronounced.

Table 4.3: ATE for different SLAM № 2 setups.

KITTI seq.	Setup	$ATE_{pos.}$ [m]	$ATE_{rot.}$ [°]
00	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	**2.35143**	**1.99284**
	VISO2 + NOSeqSLAM, optimized at each l.c.	2.36887	2.00722
	VISO2 + Naive, optimized at each 10^{th} l.c.	2.43332	1.99450
	VISO2 + Naive, optimized at each l.c.	2.43867	1.99924
	VISO2 + NOSeqSLAM, optimized once in the end	2.57337	2.00320
	VISO2 + Naive, optimized once in the end	2.68447	2.06434
	VISO2	32.11955	7.78634
02	VISO2 + Naive, optimized once in the end	**10.38643**	**3.77741**
	VISO2 + NOSeqSLAM, optimized once in the end	10.80160	3.79477
	VISO2 + Naive, optimized at each 10^{th} l.c.	10.83066	3.89191
	VISO2 + Naive, optimized at each l.c.	10.84811	3.89597
	VISO2 + NOSeqSLAM, optimized at each l.c.	11.46714	3.95923
	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	11.53330	3.96953
	VISO2	34.75933	7.92865
05	VISO2 + NOSeqSLAM, optimized once in the end	**2.08566**	1.93868
	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	2.13437	1.79494
	VISO2 + Naive, optimized once in the end	2.16836	1.99971
	VISO2 + NOSeqSLAM, optimized at each l.c.	2.18852	**1.74227**
	VISO2 + Naive, optimized at each 10^{th} l.c.	2.24392	1.94242
	VISO2 + Naive, optimized at each l.c.	2.24445	1.97885
	VISO2	12.53722	4.76568
06	VISO2 + NOSeqSLAM, optimized at each l.c.	**1.04006**	1.65912
	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	1.04270	1.65694
	VISO2 + Naive, optimized at each l.c.	1.05863	1.76240
	VISO2 + Naive, optimized at each 10^{th} l.c.	1.06338	1.76470
	VISO2 + NOSeqSLAM, optimized once in the end	1.06446	**1.63609**
	VISO2 + Naive, optimized once in the end	1.09571	1.78365
	VISO2	3.84693	3.48457
07	VISO2 + Naive, optimized at each l.c.	**0.64489**	1.37286
	VISO2 + NOSeqSLAM, optimized at each l.c.	0.68947	1.24952
	VISO2 + Naive, optimized at each 10^{th} l.c.	0.73165	1.16022
	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	0.73819	**1.15006**
	VISO2 + NOSeqSLAM, optimized once in the end	0.83023	1.33335
	VISO2 + Naive, optimized once in the end	0.83401	1.34250
	VISO2	5.50098	4.75943
09	VISO2 + NOSeqSLAM, optimized once in the end	**3.39437**	1.69139
	VISO2 + NOSeqSLAM, optimized at each 10^{th} l.c.	3.39437	1.69139
	VISO2 + NOSeqSLAM, optimized at each l.c.	3.51840	**1.67810**
	VISO2 + Naive, optimized once in the end	3.55512	1.76402
	VISO2 + Naive, optimized at each 10^{th} l.c.	3.55512	1.76402
	VISO2 + Naive, optimized at each l.c.	3.66601	1.74672
	VISO2	19.49097	8.32224

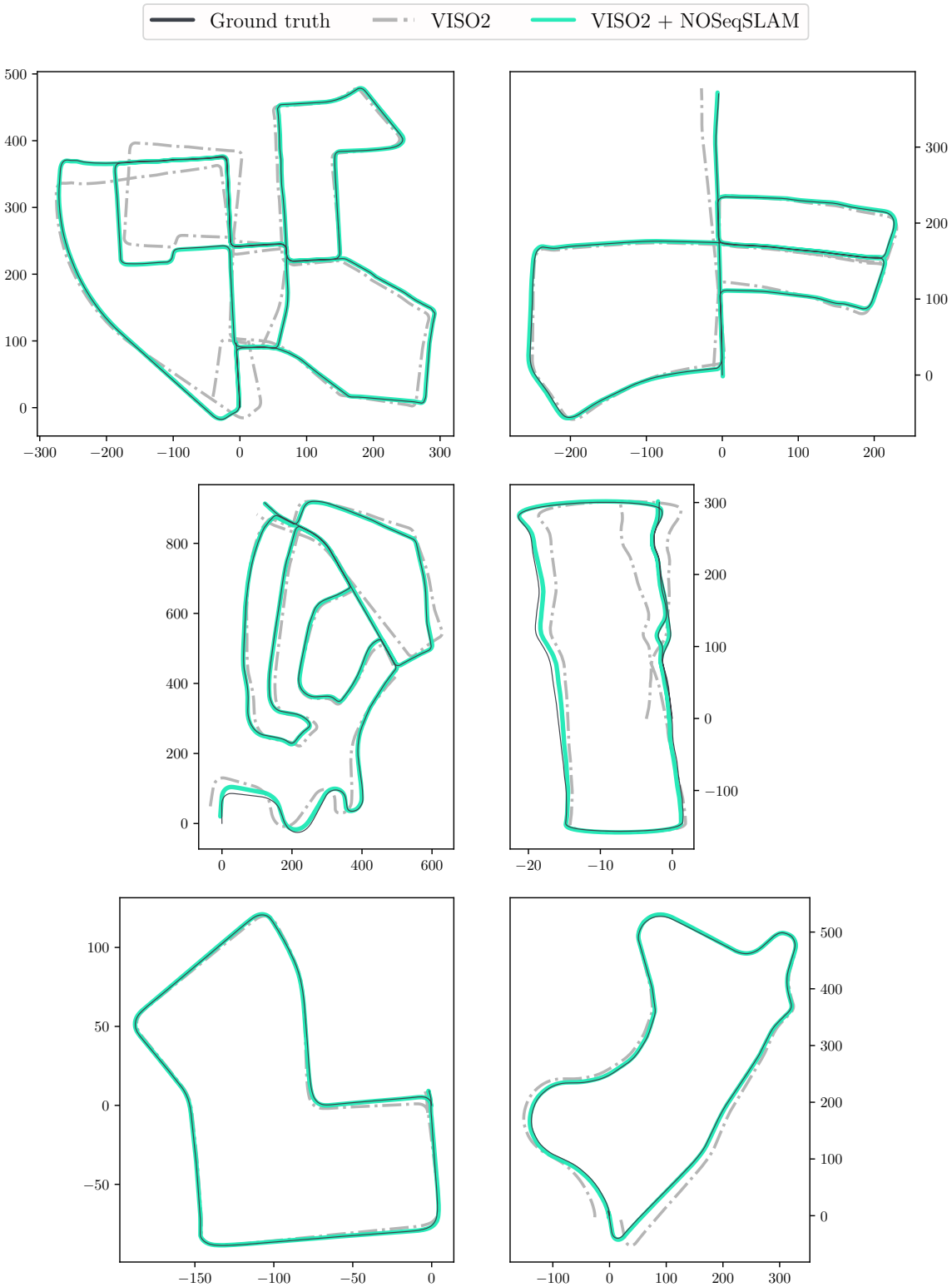


Figure 4.15: The ground truth, VISO2, and VISO2 + NOSeqSLAM trajectories for KITTI sequences 00 (top left), 05 (top right), 02 (middle left), 06 (middle right), 07 (bottom left) and 09 (bottom right).

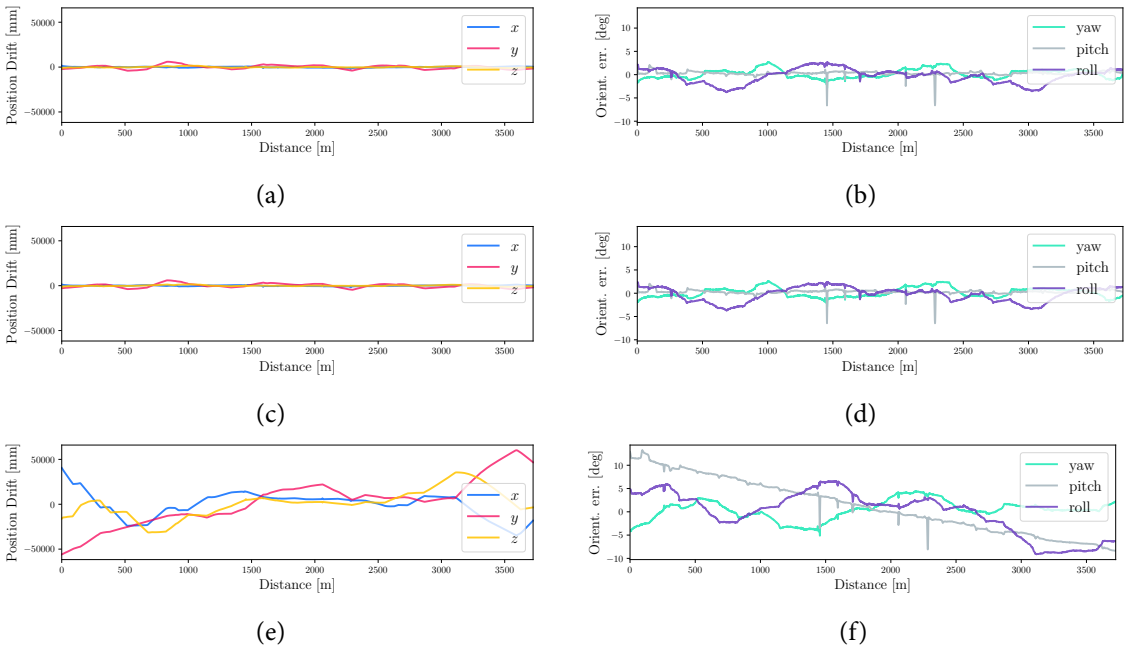


Figure 4.16: The position error (left) and the orientation error (right) with respect to distance traveled for the best performing (top) VISO2 + NOSeqSLAM, (middle) VISO2 + Naive matching, and (bottom) VISO2 runs. Plotted with the y-axis limits of the worst performing run (VISO2). Plots created with [123].

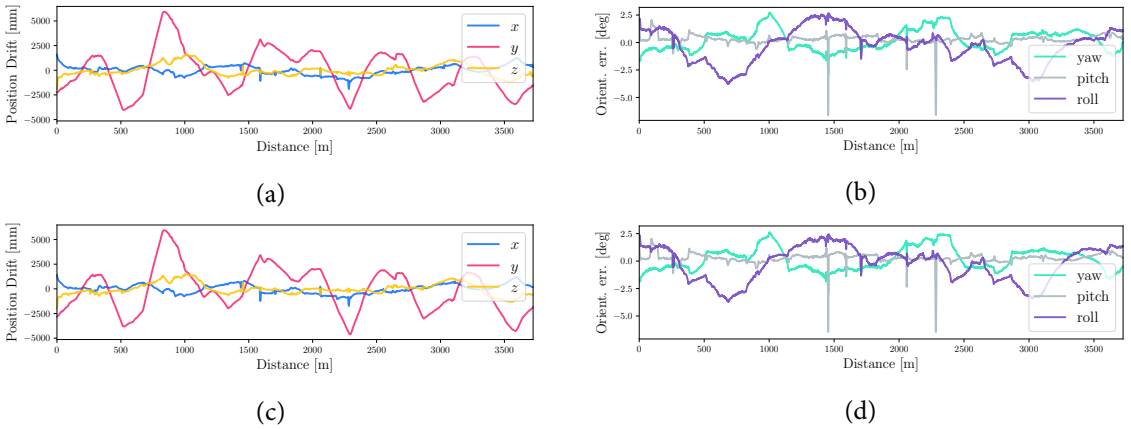
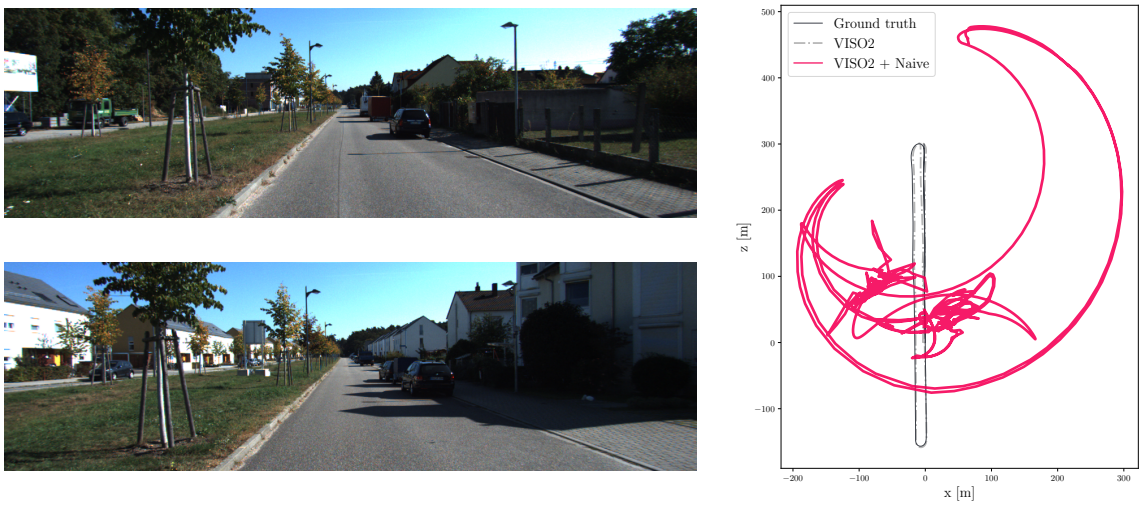


Figure 4.17: The position error (left) and the orientation error (right) with respect to distance traveled for the best performing (top) VISO2 + NOSeqSLAM, and (bottom) VISO2 + Naive matching runs where y-axis limits are magnified as much so that curves fit. Plots created with [123].



(a) (b)

Figure 4.18: The unsuccessful run for a SLAM № 2 setup with the naive matching due to a suboptimal threshold τ. Both place recognition and visual odometry failed to discard this falsely-detected loop. In turn, not only that a setup was unable to keep an estimation as good as a VISO2-only baseline estimation, but the estimated trajectory was even dramatically worse.

5

Conclusion and Outlook

For a human, it is an easy task to recognize a previously seen place. It is much more challenging to accomplish this task for an autonomous vehicle, i.e., its onboard computer. This task is an extensively investigated robotics problem – visual place recognition – that can be, as we have seen, incorporated within another robotics problem – simultaneous localization and mapping – for loop closing detection. Surprisingly, intrinsic properties of ordinary traffic situations, e.g., a lot of moving objects, different times of the day, different seasons, etc., impose difficulties for a place to be recognized by a computer program. By investigating this research problem, it became clear that, firstly, a place should be matched by taking into account a sequence of neighboring images rather than a single image of that place, and secondly, a robust image representation should be used. Hopefully, this would lead us to a viewpoint-invariant and condition-invariant place recognition system invulnerable to perceptual aliasing. To accomplish all that, three scientific contributions in the thesis were proposed.

The first scientific contribution of the thesis was to design a method that would successfully recognize such already visited places. In order to achieve this objective, by following patterns of other methods, we knew that a sequence of images, rather than a single image, should be considered. Particularly, we have been looking to SeqSLAM and then trying to generalize it, as we did with NOSeqSLAM. By generalizing it, we were able to achieve better quantitative results, although not without difficulties. Constructs that achieve that generalization are directed acyclic graphs accompanied with a single source shortest path algorithm. Once we managed to implement the first version of NOSeqSLAM, it took too long to run it even on a tiny dataset due to nonlinear data structures that demand dynamic allocation. By analyzing single source shortest path algorithms, both asymptotically and empirically where we measured running times, we have come to the optimal algorithm that is presented here. Our tailor-made algorithm, on-the-fly relaxation, perfectly fits the topology of directed acyclic graphs used in NOSeqSLAM. This, as it turned out, had a positive impact on the running time. Although SeqSLAM and its cone-based variant have a somewhat better asymptotic running time, obviously, because NOSeqSLAM generalizes SeqSLAM, empirically, NOSeqSLAM runs well for sequences of substantial lengths. Area under a curve and recall at 100% precision, borrowed from information theory, are two evaluation measures we use in order to quantize a visual place recognition system performance, but before we ventured to do that, an appropriate image representation, obtained by applying an image model, should have been provided to a place matching method.

Therefore, we put our effort into analyzing different image models. Classic computer vision models, i.e., gradient-based local features and descriptors in conjunction with aggregation techniques, as well as histogram of oriented gradients, were used in the robotics community up until recently, albeit, they are newly outperformed by deep image models – convolutional neural networks. Firstly, we have experimented with classic computer vision image models and convolutional-neural-network-extracted feature maps that came with certain datasets out of the box. Then we realized, that if we adapt an image model according to our needs, better results could be achieved. In the beginning, this was not an easy task. We thought we would be able to adapt deep image models by fine-tuning them either with *the contrastive loss* [131] or *triplet loss*. Neural networks coupled with these two losses are referred to as *Siamese neural networks*. Siamese neural networks can be considered a neural-network approach to *metric learning* where a network should map the same category inputs in a manifold where these inputs are close to each other, and inversely, inputs of different categories should be mapped away from each other. Although some authors used them in visual instance retrieval and visual place recognition [81, 82], we were only able to minimize the contrastive loss on an exemplar $|\mathcal{Q}| = 70, |\mathcal{D}| = 48$ dataset. We even extracted feature maps with an ResNet101 architecture from [82] and compared them with feature maps extracted from the original ImageNet-trained ResNet101. These results were reported in a seminar report of a Ph.D. course and were on par, which meant we should opt for another approach. For this reason, Siamese neural networks were not a topic being discussed here, and rather, we switched to softmax regression, an approach to multinomial classification with neural networks. We tried our best to adapt it in visual place recognition context, and after that, a lot of optimization experiments were conducted, which proved to be, according to the results reported, a good decision. Similarly, we had been lucky with a mutual information-based feature selection technique. Altogether, the proposed place matching method, in synergy with adapted image representation, achieved notable quantitative results, and consequently, the first and the second scientific approach were justified.

Visual place recognition should not be a goal in itself. Instead, we should use it for a higher purpose – loop closing detection in simultaneous localization and mapping. That ambition required us, alongside the adaptation of NOSeqSLAM, in which it became an online place recognition method, to develop two different SLAM systems. The adaptation started with the online place recognition formulation. Then, by virtue of array programming, we managed to speed up the online NOSeqSLAM and even make it executable on a graphic processing unit. This has proven to be a good thing because GPUs are the natural environment for deep image models. The not-an-easy task of developing two different SLAM systems was achieved thanks to the brilliant software libraries that were used. Therefore, the development of SLAM № 1 and SLAM № 2 came down to the integration of these libraries. Although it may seem that SLAM № 1 is more a toy-like system, it was demanding to develop it because of a real-life process simulation. Also, we had no previous experience, neither with factor graphs nor with the iterative closest point algorithm. Once things had been integrated and we had the functional SLAM № 1, it was much clearer how SLAM № 2 should be developed. Luckily, the absolute trajectory results reported for both SLAM systems, with loop closing detection turned on, suggest that we have succeeded with the

third scientific contribution.

In the end, here are a few words about further research. On the foundations of the proposed place matching method NOSeqSLAM and the adapted deep image models, we plan to conduct more SLAM experiments on multiple datasets to see how we can further improve this pipeline. As said, we are interested in trying out out different visual odometries, but also, different image representations that are, thanks to the popularity of deep learning-based computer vision, improved on a daily basis. Because the evaluation of SLAM № 1 showed us how NOSeqSLAM goes beyond the scope of visual place recognition, we would try to use it with real-life LiDAR-based SLAM systems such as [132]. Also, we have an idea of how to generalize NOSeqSLAM even further, so it is more glued together with deep image models, but more about it once when, hopefully, good results are attained.

BIBLIOGRAPHY

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[2] Michael J. Milford and Gordon F. Wyeth. SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1643–1649. IEEE, may 2012.

[3] Stephanie Lowry and Michael Milford. Supervised and Unsupervised Linear Learning Techniques for Visual Place Recognition in Changing Environments. *IEEE Transactions on robotics*, 32(3):600–613, 2016.

[4] Jurica Maltar, Ivan Marković, and Ivan Petrović. NOSeqSLAM: Not only Sequential SLAM. In Manuel F Silva, José Luís Lima, Luís Paulo Reis, Alberto Sanfeliu, and Danilo Tardioli, editors, *Robot 2019: Fourth Iberian Robotics Conference*, pages 179–190, Cham, 2020. Springer International Publishing.

[5] Jurica Maltar, Ivan Marković, and Ivan Petrović. Visual place recognition using directed acyclic graph association measures and mutual information-based feature selection. *Robotics and Autonomous Systems*, 132:103598, 2020.

[6] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1150–1157 vol.2. IEEE, sep 1999.

[7] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3951 LNCS(4):404–417, 2006.

[8] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):591–606, 2009.

[9] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating Local Image Descriptors into Compact Codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, sep 2012.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[11] Stephanie Lowry, Niko Sunderhauf, Paul Newman, John J. Leonard, David Cox, Peter Corke, and Michael J. Milford. Visual Place Recognition: A Survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[12] Benjamin Kuipers and Benjamin Kuipers. The Spatial Semantic Hierarchy (v.2). *Artificial Intelligence*, 119(1-2):191–233, 1999.

[13] Benjamin Kuipers and Yung-tai Byun. A robot exploration and mapping strategy based on a semantic. 56:47–63, 1991.

[14] Ali Sharif Razavian, Josephine Sullivan, Stefan Carlsson, and Atsuto Maki. Visual Instance Retrieval with Deep Convolutional Networks. (June 2017), 2014.

[15] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9–11, 2007.

[16] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9–11, 2007.

[17] Sourav Garg, Niko Sünderhauf, and Michael Milford. LoST? Appearance-Invariant Place Recognition for Opposite Viewpoints using Visual Semantics. *CoRR*, abs/1804.0, 2018.

[18] Tayyab Naseer, Luciano Spinello, Wolfram Burgard, and Cyrill Stachniss. Robust visual robot localization across seasons using network flows. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2564–2570, 2014.

[19] Fabio Ramos, Ben Upcroft, Suresh Kumar, and Hugh Durrant-Whyte. A Bayesian approach for place recognition. *Robotics and Autonomous Systems*, 60(4):487–497, apr 2012.

[20] Marvin Chancán and Michael Milford. Sequential Place Learning: Heuristic-Free High-Performance Long-Term Place Recognition. 2021.

[21] Jiayi Ma, Xinyu Ye, Huabing Zhou, Xiaoguang Mei, and Fan Fan. Loop-Closure Detection Using Local Relative Orientation Matching. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–14, 2021.

[22] Tayyab Naseer, Michael Ruhnke, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard. Robust visual SLAM across seasons. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:2529–2535, 2015.

[23] E. Pepperell, P. I. Corke, and M. J. Milford. All-environment visual place recognition with smart. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1612–1618, May 2014.

[24] Mark Cummins and Paul Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *International Journal of Robotics Research*, 27(6):647–665, 2008.

[25] Ben Talbot, Sourav Garg, and Michael Milford. OpenSeqSLAM2.0: An Open Source Toolbox for Visual Place Recognition Under Changing Conditions. *IEEE Robotics and Automation Letters*, 1(1):213–220, apr 2018.

[26] S. M. Siam and H. Zhang. Fast-seqslam: A fast appearance based place recognition algorithm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5702–5708, May 2017.

[27] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 1958.

[28] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.

[29] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[30] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 1962.

[31] Shimon Even. *Graph Algorithms*. Cambridge University Press, 2 edition, 2011.

[32] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[33] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry, 2016.

[34] Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3d reconstruction in real-time. *IEEE Intelligent Vehicles Symposium, Proceedings*, (Iv):963–968, 2011.

[35] Igor Cvisic, Ivan Markovic, and Ivan Petrovic. SOFT2: Stereo Visual Odometry for Road Vehicles Based on a Point-to-Epipolar-Line Metric. *IEEE Transactions on Robotics*, 2022.

[36] Richard Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–10, 2006.

[37] Richard Szeliski. *Feature Detection and Matching*, pages 333–399. Springer International Publishing, Cham, 2022.

[38] David A. Forsyth and Jean Ponce. *Computer Vision - A Modern Approach, Second Edition.* Pitman, 2012.

[39] C. Harris and M. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, 1988.

[40] Jianbo Shi and Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, number June, pages 593–600. IEEE Comput. Soc. Press, 1994.

[41] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[42] Krystian Mikolajczyk, Cordelia Schmid, Krystian Mikolajczyk, Cordelia Schmid, Ieee Computer, and Cordelia Schmid. Indexing based on scale invariant interest points To cite this version :. pages 525–531, 2010.

[43] A. Haar. Zur theorie der orthogonalen funktionensysteme. (zweite mitteilung). *Mathematische Annalen*, 71:38–53, 1912.

[44] Rudolf Scitovski, Kristian Sabo, Francisco Martínez-Álvarez, and Šime Ungar. *Data Clustering*, pages 31–64. Springer International Publishing, Cham, 2021.

[45] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[46]J. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.

[47] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. *32nd International Conference on Machine Learning, ICML 2015*, 1:579–587, 2015.

[48] Vadim Markovtsev. "yinyang" k-means and k-nn using nvidia cuda. `https://github.com/src-d/kmcuda`, 2017.

[49]Robert K. McConnell. Method of and apparatus for pattern recognition, 1982.

[50] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3952 LNCS:428–441, 2006.

[51] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[52] Olga Vysotska and Cyrill Stachniss. Lazy Data Association For Image Sequences Matching Under Substantial Appearance Changes. *IEEE Robotics and Automation Letters*, 1(1):213–220, 2016.

[53] Edward Rosten and Tom Drummond. Machine Learning for High-Speed Corner Detection. pages 430–443. 2006.

[54] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, 2010.

[55] Andrej Karpathy. Cs231n: Deep learning for computer vision. `https://cs231n.github.io/`, 2022.

[56] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[57] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[58] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 28, 2013.

[59] B T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[60] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. pages 1–18, 2012.

[61] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–11, 2015.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[63] Y LeCun, L Bottou, Y Bengio, and P Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[67] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015.

[68] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation, 2019.

[69] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[70] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2980–2988, 2017.

[71]Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. 2018.

[72] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[73] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[74] Georgia Tech. Introduction to computer vision. `https://www.udacity.com/course/introduction-to-computer-vision--ud810`.

[75] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, sep 2014.

[76] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[77] Niko Sünderhauf, Sareh Shirazi, Feras Dayoub, Ben Upcroft, and Michael Milford. On the performance of ConvNet features for place recognition. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:4297–4304, 2015.

[78] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 487–495. Curran Associates, Inc., 2014.

[79] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Robert Fergus, and Yann Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[80] O. Vysotska, T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Efficient and effective matching of image sequences under substantial appearance changes exploiting gps priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2774–2779, May 2015.

[81] R Arandjelović, P Gronat, A Torii, T Pajdla, and J Sivic. NetVLAD: CNN Architecture for Weakly Supervised Place Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1437–1451, 2018.

[82] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Fine-Tuning CNN Image Retrieval with No Human Annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1655–1668, 2019.

[83] Niko Sünderhauf, Sareh Shirazi, Adam Jacobson, Feras Dayoub, Edward Pepperell, Ben Upcroft, and Michael Milford. Place Recognition with ConvNet Landmarks: Viewpoint-Robust, Condition-Robust, Training-Free. In *Robotics: Science and Systems XI*. Robotics: Science and Systems Foundation, jul 2015.

[84] Marvin Chancán, Luis Hernandez-Nunez, Ajay Narendra, Andrew B. Barron, and Michael Milford. A Compact Neural Architecture for Visual Place Recognition. 2019.

[85] Zetao Chen, Adam Jacobson, Niko Sünderhauf, Ben Upcroft, Lingqiao Liu, Chunhua Shen, Ian Reid, and Michael Milford. Deep learning features at scale for visual place recognition. *Proceedings - IEEE International Conference on Robotics and Automation*, 1:3223–3230, 2017.

[86] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 1–28, 2018.

[87] Alex Lamb, Vikas Verma, Kenji Kawaguchi, Alexander Matyasko, Savya Khosla, Juho Kannala, and Yoshua Bengio. Interpolated Adversarial Training: Achieving robust neural networks without sacrificing too much accuracy. *Neural Networks*, 154:218–233, 2022.

[88] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. MixUp: Beyond empirical risk minimization. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 1–13, 2018.

[89] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004.

[90] Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined convex programming. In *Global Optimization: From Theory to Implementation, Nonconvex Optimization and Its Application Series*, pages 155–210. Springer, 2006.

[91] Junlin Hu, Jiwen Lu, and Yap Peng Tan. Discriminative deep metric learning for face verification in the wild. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1875–1882, 2014.

[92] Daniel Olid, José M. Fácil, and Javier Civera. Single-view place recognition under seasonal changes. In *PPNIV Workshop at IROS 2018*, 2018.

[93] Jurica Maltar and Domagoj Matijevic. Optimization techniques for image representation in visual place recognition. *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology, MIPRO 2022 - Proceedings*, pages 877–882, 2022.

[94] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[95] Lixin Cui, Lu Bai, Zhihong Zhang, Yue Wang, and Edwin R. Hancock. Identifying the most informative features using a structurally interacting elastic net. *Neurocomputing*, 336:13–26, 2019.

[96] Lixin Cui, Lu Bai, Yue Wang, Xiao Bai, Zhihong Zhang, and Edwin R. Hancock. P2p lending analysis using the most relevant graph-based features. In Antonio Robles-Kelly, Marco Loog, Battista Biggio, Francisco Escolano, and Richard Wilson, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 3–14, Cham, 2016. Springer International Publishing.

[97] Lu Bai, Luca Rossi, Horst Bunke, and Edwin R. Hancock. Attributed graph kernels using the Jensen-Tsallis q-differences. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[98] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. dec 2013.

[99] Frank Dellaert. Factor Graphs: Exploiting Structure in Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:141–166, 2021.

[100] William R Esposito and Christodoulos A Floudas. *Gauss–Newton method: Least squares, relation to Newton's method*, pages 1129–1134. Springer US, Boston, MA, 2009.

[101] Jorge J Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G A Watson, editor, *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[102] Kenneth J. Waldron and James Schmiedeler. *Kinematics*, pages 11–36. Springer International Publishing, Cham, 2016.

[103] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, page 593–598, USA, 2002. American Association for Artificial Intelligence.

[104] Frank Dellaert, Richard Roberts, Varun Agrawal, Alex Cunningham, Chris Beall, Duy-Nguyen Ta, Fan Jiang, lucacarlone, nikai, Jose Luis Blanco-Claraco, Stephen Williams, ydjian, John Lambert, Andy Melim, Zhaoyang Lv, Akshay Krishnan, Jing Dong, Gerry Chen, Krunal Chande, balderdash devil, DiffDecisionTrees, Sungtae An, mpaluri, Ellon Paiva Mendes, Mike Bosse, Akash Patel, Ayush Baid, Paul Furgale, matthewbroadwaynavenio, and roderick koehle. borglab/gtsam, May 2022.

[105] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA, 1st edition, 2017.

[106] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. Range Image Understanding.

[107] Ellon Mendes, Pierrick Koch, and Simon Lacroix. Icp-based pose-graph slam. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–200, 2016.

[108] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. Ct-icp: Real-time elastic lidar odometry with loop closure. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5580–5586, 2022.

[109] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2043–2050, 2017.

[110] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[111] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):2174–2181, 2015.

[112] Chang Chen, Hua Zhu, Menggang Li, and Shaoze You. A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives. *Robotics*, 7(3):1–20, 2018.

[113] The MathWorks Inc. Understanding slam using pose graph optimization | | autonomous navigation, part 3. `https://youtu.be/saVZtgPyyJQ`.

[114] Raul Mur-Artal and Juan D. Tardos. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[115] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[116] D. Galvez-López and J. D. Tardos. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, oct 2012.

[117] Titus Cieslewski, Siddharth Choudhary, and Davide Scaramuzza. Data-efficient decentralized visual SLAM. *CoRR*, abs/1710.05772, 2017.

[118] Zichao Zhang and Davide Scaramuzza. A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. *IEEE International Conference on Intelligent Robots and Systems*, pages 7244–7251, 2018.

[119] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, apr 1991.

[120] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[121] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148, February 2013.

[122] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[123] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[124] Raphael Finkel and Jon Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Inf.*, 4:1–9, 1974.

[125] Magnus Egerstedt. Control of Autonomous Mobile Robots. *Handbook of Networked and Embedded Control Systems*, pages 767–778, 2005.

[126] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A. Chapman, Dongpu Cao, and Jonathan Li. Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3412–3432, 2021.

[127] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J. Leonard, and Frank Dellaert. ISAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, 31(2):216–235, 2012.

[128] Raul Mur-Artal, J. M.M. Montiel, and Juan D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[129] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[130] Irwin Sobel. An Isotropic 3x3 Image Gradient Operator. *Presentation at Stanford A.I. Project 1968*, 2014.

[131] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE.

[132] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.

CURRICULUM VITAE

Jurica Maltar was born Osijek, Croatia in 1993. He graduated from the Jesuit Classical Gymnasium with right of public, Osijek in 2012. During the undergraduate university study programme in mathematics and the graduate university study programme in mathematics at the Josip Juraj Strossmayer University of Osijek, Department of Mathematics, he participated in a programming competition IEEEXtreme, did summer internships at Adacta Ltd. and Farmeron Ltd., a student job at Roxoft Ltd and a workshop at Summer school of informatics in the organization of Osijek Software City. Under the supervision of Associate Professor Domagoj Matijević, PhD, he received the master degree in mathematics (cum laude) in 2017.

That same year, after graduating from college, he was employed as a research and teaching associate at the Josip Juraj Strossmayer University of Osijek, Department of Mathematics, where he is actively involved in computer science lectures. At the same time, he was enrolled in the doctoral study programme at University of Zagreb, Faculty of Electrical Engineering and Computing under the supervision of Professor Ivan Marković, PhD and Associate Professor Domagoj Matijević, PhD.

The main areas of his scientific interest are mobile robotics, computer vision, machine and deep learning. As part of his research, he published one journal paper and two conference papers.

FULL LIST OF PUBLICATIONS

JOURNAL PUBLICATIONS:

1. J. Maltar, I. Marković and I. Petrović. Visual place recognition using directed acyclic graph association measures and mutual information-based feature selection. *Robotics and Autonomous Systems*, 132, 2020, IF: 2.825 (Q2).

CONFERENCE PUBLICATIONS:

1. J. Maltar, I. Marković and I. Petrović. NOSeqSLAM: Not only Sequential SLAM. *Robot 2019: Fourth Iberian Robotics Conference*. Porto, Portugal, 179–190, 2019.
2. J. Maltar, D. Matijević. Optimization techniques for image representation in visual place recognition. *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija, Croatia, 877–882, 2022.

ŽIVOTOPIS

Jurica Maltar rođen je 1. travnja 1993. godine u Osijeku. Srednjoškolsko je obrazovanje završio u Isusovačkoj klasičnoj gimnaziji s pravom javnosti u Osijeku 2012. godine. Tijekom preddiplomskog i diplomskog studija matematike na Odjelu za matematiku Sveučilišta Josipa Jurja Strossmayera u Osijeku sudjeluje na natjecanju iz programiranja IEEEXtreme, pohađa studentske prakse u poduzećima Adacta d.o.o. i Farmeron d.o.o., obavlja studentski posao u poduzeću Roxoft d.o.o. te sudjeluje kao predavač Ljetne škole informatike u organizaciji Osijek Software City-a. Pod mentorstvom izv. prof. dr. sc. Domagoja Matijevića, 2017. godine završava diplomski studij matematike, smjer matematika i računarstvo te stječe zvanje magistra matematike (cum laude).

Po završetku diplomskog studija, zaposlen je na mjestu asistenta Katedre za računarstvo Odjela za matematiku Sveučilišta Josipa Jurja Strossmayera u Osijeku gdje aktivno sudjeluje u izvođenju nastave iz nekoliko kolegija. Također, upisuje doktorski studij Fakulteta elektrotehnike i računarstva Sveučilišta u Zagreb, smjer računarstvo, pod mentorstvom prof. dr. sc. Ivana Markovića i izv. prof. dr. sc. Domagoja Matijevića.

Njegova su glavna područja znanstvenih interesa mobilna robotika, računalni vid, strojno i duboko učenje. U sklopu svojih istraživanja objavio je jedan rad u znanstvenom časopisu te dva rada u zbornicima međunarodnih konferencija.

COLOPHON

This document was typeset and inspired by the typographical look-and-feel `classicthesis` developed by André Miede, which was based on Robert Bringhurst's book on typography *The Elements of Typographic Style*, and by the `FERElemental` developed by Ivan Marković whose design was based on `FERBook` developed by Jadranko Matuško.