Norges miljø- og biovitenskapelige universitet

Masteroppgave 2017 30 stp
Fakultet for realfag og teknologi

Kamerastøttet treghetsnavigasjon for bilbåren datainnsamling

Camera aided inertial navigation for mobile mapping systems

Kim Willem Sidenius van Woensel Kooy
Geomatikk

Sammendrag

Bilbåren datainnsamling har i de siste årene blitt en mye brukt metode for å samle inn geografiske data. Metoden er svært effektiv til å for eksempel lage detaljerte 3D-modeller av byer og landskap. For å beregne posisjonen er en kombinasjon av satellitt- og treghetsnavigasjon mye brukt. Da oppnås høy nøyaktighet på posisjonen. Ved fravær av satellittsignaler vil posisjonsestimatet etterhvert avvike fra den faktiske posisjonen på grunn av drift i treghetssensorene.

Digital bildeanalyse, eller *Computer Vision*, er et fagfelt i enorm utvikling. Det lanseres stadig nye funksjoner og algoritmer som kan tolke innholdet i digitale bilder. Med dette har det åpnet seg muligheter for å bruke kamera som støttesensor til et navigasjonssystem. Denne oppgaven ser på to ulike innfallsvinkler for dette: nullhastighetsoppdatering og visuell odometri.

For nullhastighetsoppdateringen blir det gjort en enkel antagelse om at dersom andelen endrede pikselverdier mellom to bilder er nærme null, så har det ikke vært bevegelse mellom bildene. Under enkelte forutsetninger viser dette seg å stemme godt.

I delen om visuell odometri beregnes den relative orienteringen mellom hvert bilde ved å tracke *nøkkelpunkter* gjennom bildene. Ut i fra det kan kameraets relative posisjon estimeres. Dette har vist seg å være en krevende prosess, men tidligere forsøk har vist at det ligger et stort potensialet i visuell odometri.

Abstract

Mobile mapping has become a much used method to collect geographical data. This is a highly efficient method to for instance create 3D-models of cities and landscape with high grade of details. A combination of satellite- and inertial navigation system are used to derive the exact position. This results in a high accuracy. With absence of satellite-signals the system must estimate it's position with the inertial sensors. Unfortunatly, these sensors are drifting, so the estimate of the position will differ from the true position if there is a longer absence from satellite-signals.

With the big development within *Computer Vision*, it is now possible to use a camera as an aiding sensor to the navigation system. This thesis look at two different problems: zero velocity update, and visual odometry.

To detect the zero velocity there is a basic assumption that if the difference in pixel values between two images is close to zero, then the system is not moving. This turns out to be good assumption.

The visual odometry is used to calculate the relative orientation between images. The relative orientation is then used to estimate the relative position of the camera. This is done by tracking keypoints through the images.

Forord

Denne oppgaven markerer slutten på fem års studier ved Norges Miljø- og Biovitenskapelige Universitet. Selve oppgaven bygger på mye av kunnskapen jeg har tilegnet meg i løpet av disse årene, samt mye ny teori.

Jeg vil takke min veileder, Narve Schipper Kjørsvik ved TerraTec AS. Først og fremst er jeg veldig takknemmelig for alt han har lært meg om temaene som inngår i denne oppgaven. Det har vært en lærerik og krevende prosess, men Narve har vært tolmodig og hjelpsom. Døren hans har alltid stått åpen, og jeg kunne ikke bedt om en bedre veileder.

Jeg vil også rekke en stor takk til min biveileder Jon Glenn Gjevestad og de andre foreleserne på Geomatikkseksjonen ved NMBU. De har alle bidratt til et godt læringsmiljø.

I tillegg vil jeg takke mine venner og familie. En spesiell takk til de andre masterstudentene som har hjulpet til med å holde motet oppe, min fetter og hans kjæreste for hjelp med programmering, og min venninne Ida for hjelp med bilder til oppgaven.

Til slutt vil jeg også takke mine foreldre. Deres støtte og tro har gitt meg motivasjon til å stå på gjennom hele masterperioden.

Innhold

Figurer

Kapittel 1

Introduksjon og problemdefinisjon

1.1 Bakgrunn

Bilbåren datainnsamling (eng.: Mobile Mapping) er en effektiv metode for å samle inn store mengder geografiske data. Metoden går ut på å montere en rekke sensorer på et kjøretøy for å kunne drive datainnsamling langs veier og andre aktuelle områder. Laserskannere og kameraer brukes til å observere omgivelsene, og posisjonen blir bestemt ved hjelp av satellitt- og treghetsnavigasjon. På den måten kan alle objektene som er synlig fra kjøretøyet bli observert og få en eksakt posisjon i et bilde eller i en 3D-modell. Et moderne laserskanningssystem kan samle inn over en million punkter per sekund, så metoden er langt mer effektiv enn tradisjonell landmåling. Med en god navigasjonsløsning kan kvaliteten bli veldig høy.

1.1.1 Tidligere systemer

Bilbåren datafangst ble tatt i bruk allerede i 1983 i enkelte stater og provinser i USA og Canada. Det første systemet het *Mobile Highway Inventory System* (MHIS) [El-Sheimy, 1996] og ble brukt til å kartlegge motorveier. En kombinasjon av navigasjons- og bildesensorer ble brukt til å registrere alle synlige objekter langs en vei, og informasjonen ble brukt til å danne en

kontinuerlig 3D-modell av hele veikorridoren. Posisjonen ble bestemt av enkle treghetssensorer og et odometer, og nøyaktigheten var ikke spesielt god. Senere på 1980-tallet ble også satellittnavigasjon tatt i bruk, og systemene som integrerte dette fikk langt bedre nøyaktighet [El-Sheimy, 1996].

VISAT

V*ideo cameras,* ***I****nertial system and* ***SAT****ellite GPS recievers* var et datafangstsystem som ble utviklet av Universitetet i Calgary på midten av 1990-tallet [El-Sheimy, 1996]. Hovedformålet var å lage et effektivt og nøyaktig system for å samle inn geografiske data. Målet var å kunne kjøre i inntil 60km/t og oppnå en absolutt nøyaktighet på 0.3 m, og en relativ nøyaktighet på 0.1 m innenfor en 35 m radius.

For å klare dette bestod systemet av to GPS-antenner, en fastmontert IMU (de ulike delene navigasjonssystemet omtales nærmere i kapittel 2) og åtte kameraer. To av disse kameraene ble satt opp til å ta bilder av høytliggende objekter som strømledninger. De siste seks av kameraene var plassert slik at de dekket 220° i horisontal retning og 37° i vertikal retning, og ga mulighet til å sy sammen bildene. Dette var en utfordring ettersom det på den tiden fantes lite utstyr som var i stand til å synkronisere såpass mange kameraer til å ta bilder samtidig. Hele kamerasystemet var også koblet mot navigasjonssystemet slik at alle bildene ble lagret sammen med tidsstempling fra GPS.

I etterprosesseringen var målet å lage et geografisk informasjonssystem (GIS) der alle objektene skulle ha 3D-koordinater. De endelige resultatene for VISAT-systemet endte med en nøyaktighet på 0.2 m [El-Sheimy, 1996].

1.1.2 Dagens systemer

TerraTec bruker i dag en Toyota Land Cruiser 150 GX med navigasjonssystemet POS LV 610 fra Applanix[1] og laserskanningssystemet Optech Lynx Mobile Mapper SG1[2].
Navigasjonssystemet benytter en kombinasjon av satellittnavigasjon og treghetsnavigasjon. Førstnevnte består av en GNSS-mottaker med to antenner. Sistnevnte består av en IMU, som igjen består av ett akselerometer og ett gyroskop på hver de tre aksene. I tillegg er systemet koblet til et odometer. GNSS-mottakeren har mulighet til å måle på alle frekvensene til både GPS, GLONASS, Galileo og Beidou. IMUen er av høy kvalitet og fra spesifikasjonene lover den en posisjonsnøyaktiget innenfor 0.1 m etter seksti sekunder uten GNSS-dekning. Odometeret brukes som en støttesensor, og gir kun informasjon om bilens hastighet og tilbakelagt avstand.
Laserskanningssystemet består av to skråstilte skannere med en målerate på opptil 1.2 millioner punkter per sekund. Systemet har også støtte for inntil fem digitalkameraer med oppløsning på 5 megapiksler, samt et 360° ladybug kamera.

Figur 1.1: Optech Lync Mobile Mapper.

[1] http://www.applanix.com/pdf/specs/POSLV_Specifications_dec_2015.pdf
[2] http://www.teledyneoptech.com/wp-content/uploads/specification_lynx-sg1.pdf

1.2 Problemdefinisjon

Dagens navigasjonssystem fra Applanix gir gode resultater i områder med god GNSS-dekning, og noe svekkede resultater i tuneller og andre områder med liten eller ingen GNSS-dekning. Uten satellittnavigasjon vil systemet estimere posisjonen ved bruk av kun treghetsnavigasjonen. Ved lange perioder uten GNSS-dekning vil den estimerte posisjonen etterhvert avvike fra den faktiske posisjonen på grunn av drift i treghetssensorene. Med satellittnavigasjon tilgjengelig kan systemet estimere driften i treghetssensorene. Odometeret gir som nevnt informasjon om hastighet og tilbakelagt avstand, og vil dermed kunne estimere eventuell drift i kjøreretningen.

Oppgaven er delt i to; den første delen går ut på å bestemme når bilen har stått i ro (nullhastighetsoppdatering), og den andre delen går ut på å bruke kameraet til å estimere systemets relative posisjon (visuell odometri).

1.2.1 Del 1: Nullhastighetsoppdatering

Ved fravær av satellittnavigasjon vil systemet som nevnt måtte stole på treghetsnavigasjonen. Når kjøretøyet står i ro vil akselerometerene i IMUen kun observere tyngdens akselerasjon. Denne akselerasjonen er kjent og kan dermed korrigeres for, men det forutsetter at systemets orientering også er kjent. Dersom orienteringen ikke er korrekt bestemt vil tyngdeakselerasjonen påvirke akselerometerene på en annen måte enn det korrigeres for. Da vil systemet påstå at det er i bevegelse selv om det er i ro. Et odometer kan benyttes til å si om systemet er i ro eller ikke, men ved fravær av odometermålinger (for eksempel på jernbane) er det ingen sensorer til å fortelle om systemet faktisk er i ro eller ikke. Denne delen av oppgaven vil dermed se på om det er mulig å bruke et kamera til å detektere nullhastighet.

1.2.2 Del 2: Visuell odometri

Denne delen av oppgaven vil ta for seg en noe mer avansert bruk av bilder - i såkalt visuell odometri. Prinsippet går ut på å avgjøre hvor langt, samt i hvilken retning kameraet har beveget seg mellom to bildeeksponeringer. Det vil altså fungere som et odometer, men gir i tillegg informasjon om systemets orientering. I systemer som benytter enkle treghetssensorer med høy drift, kan visuell odometri potensielt gi en markant forbedring av posisjonsestimatet ved lengre perioder med dårlig eller ingen tilgang til satellittnavigasjon.

Visuell odometri er en mye brukt teknikk i robotnavigasjon. Typisk vil en robot lage sitt eget kart over omgivelsene mens den beveger seg. Dette gjør den ved hjelp av både bilde- og lasersensorer. Den samler altså inn data, for så å bruke dataene til å navigere seg frem. Denne metoden kalles SLAM[3] (*Simultaneous Localization And Mapping*).

1.3 OpenCV

For å løse begge problemstillingene har biblioteket OpenCV[4] blitt benyttet. Dette er et bibliotek med åpen kildekode som består av funksjoner for bildebehandling, men med et spesielt fokus på sanntids *Computer vision*. Computer vision er et tverrfaglig felt som handler om hvordan datamaskiner kan tolke digitale bilder og videoer[5]. Dette kan være for å automatisere oppgaver som tidligere krevde mennesker (for eksempel bomstasjoner som nå er automatiske). I autonome kjøretøy er også tolkning av bilder en sentral del. Det å bruke bilder til avanserte formål er dermed i en veldig utvikling. OpenCV er regnet som et av de største bibliotekene for bildebehandling og Computer vision. Funksjonene i biblioteket er skrevet i programmeringsspråket C++, og de er dermed veldig effektive. Funksjonene kan også brukes med både C, Python og Java. For mer informasjon se hjemmesiden til OpenCV: http://opencv.org.

[3]SLAM: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping
[4]OpenCV: https://en.wikipedia.org/wiki/OpenCV
[5]Computer vision: https://en.wikipedia.org/wiki/Computer_vision

Kapittel 2

Navigasjonssystem

2.1 Introduksjon

Det finnes i dag ulike teknikker for å estimere posisjoner. Det skilles typisk mellom absolutt og relativ posisjonsbestemmelse. For å bestemme den absolutte posisjonen benyttes i dag satellittnavigasjon (delkapittel 2.3). For å bestemme den relative posisjonen benyttes gjerne treghetsnavigasjon (delkapittel 2.4) og/eller odometri. Dette går ut på å måle endringer i posisjon, hastighet og orientering relativt en starttilstand.

Det er fordeler og ulemper med begge disse metodene, og til enkelte formål vil det være nødvendig å kombinere absolutt og relativ posisjonsbestemmelse (delkapittel 2.5).

2.2 Referanserammer

Dette delkapitlet vil ta for seg ulike referanserammer som brukes i navigasjonssystemer. Ulike sensorer gjør målinger i ulike referanserammer, og det vil derfor være nødvendig å transformere mellom de ulike rammene. For å transformere må det benyttes *rotasjonsmatriser*, og utledning av disse kan sees i [Farrell, 2008, kap. 2.4, s. 35 - 39]. I denne oppgaven brukes C som

rotasjonsmatrise. C_a^b definerer en rotasjonsmatrise som transformerer fra referanseramme a til referanseramme b. Den inverse av C_a^b transformerer fra referanseramme b til referanseramme a. Definisjon av de påfølgende referanserammene er hentet fra [Farrell, 2008, kap. 2.2, s. 23 - 27].

i-frame: Inertiell referanseramme
En inertiell referanseramme er en referanseramme der Newtons bevegelsesligninger er gjeldende. Det vil si at referanserammen ikke er akselerert. Treghetssensorer som gyroskoper og akselerometere gjør målinger relativt den inertielle referanserammen.

e-frame: jordens referanseramme (ECEF)
Dette er et referansesystem med origo i jordsentrum, Z-aksen pekende opp langs rotasjonsaksen, x-aksen mot krysningspunktet mellom ekvator og Greenwichmeridianen og Y-aksen fullfører et høyrehånds kartesisk koordinatsystem. Aksene følger med jordrotasjonen, og dermed kan ikke Newtons lover anvendes direkte i denne referanserammen. I satellittnavigasjon blir målingene referert til e-frame.

g-frame: lokal geografisk referanseramme
Aksene i denne geografiske referanserammen peker mot nord, øst og ned, og utspenner et kartesisk koordinatsystem. Referanserammen vil følge systemet etterhvert som det beveger seg relativt ellipsoiden. Typisk brukt til navigasjonsformål.

b-frame: kjøretøyets referanseramme
Dette er kjøretøyets referanseramme. Denne referanserammen er i hovedsak til for å kunne bestemme hvordan ulike sensorer på kjøretøyet er plassert i forhold til hverandre.

s-frame: sensorens referanseramme
Dette er treghetssensorens referansesystem. Aksene utspenner et høyrehånds, kartesisk koordinatsystem.

2.3 Satellittnavigasjon (GNSS)

Satellittnavigasjon baseres på enveies-radiokommunikasjon fra en satellitt til en mottaker. Satellitten sender radiobølger med en viss frekvens som mottakeren fanger opp og leser av. I radiobølgene ligger informasjon om satellittens posisjon og hastighet, samt nøyaktig tidspunkt for når signalet ble sendt [Hofmann-Wellenhof et al., 2007].

Det finnes i dag flere satellittnavigasjonssystemer: GPS (USA), GLONASS (Russland), Galileo (EU) og BeiDou (Kina). De to sistnevnte er fortsatt under utvikling. GNSS-mottakeren som er en del av navigasjonssystemet fra Applanix kan måle på alle systemenes frekvenser, og er dermed klar når f. eks. Galileo blir satt i full drift (estimert til 2020 [Tegedor, 2015]).

2.3.1 Observasjonsligning og feilkilder

GNSS-mottakere til profesjonelt bruk gjør målinger direkte på satellittenes bærebølger. Det fungerer ved at mottakeren genererer et signal tilsvarende det signalet som satellitten sender. Ved måletidspunktet blir fasen, altså differansen mellom de to signalene, målt. I tillegg beregnes antall hele bølgelengder mellom satellitt og mottaker [Tegedor, 2015]. En forenklet versjon av observasjonsligningen for fasemåling kan settes opp slik:

$$L_{i_r}^s = \rho_r^s + T_r^s - I_{i_r}^s + c(dt_r - dt^s) + N_r^s\lambda + \epsilon_L \tag{2.1}$$

der:

- ρ_r^s er den geometriske avstanden mellom satellitt og mottaker.
- T_r^s og $I_{i_r}^s$ er henholdsvis troposfærens og ionosfærens forsinkelse av signalet mellom satellitt og mottaker.
- dt_r og dt^s er klokkefeil i henholdsvis mottaker og satellitt.
- N_r^s er antall hele bølgelengder mellom satellitt og mottaker.
- λ er signalets bølgelengde.

- c er lysets hastighet i vakum.
- ϵ_L er andre feilkilder som for eksempel flerveisinterferens.

Dobbeltdifferanser

Ved å gjøre målinger med flere mottakere samtidig kan det settes opp differanser. Disse differansene vil eliminere feilene i både mottaker- og satellittklokkene, samt redusere effekten av troposfæren og ionosfæren. Dette gjør det mulig å estimere heltallsløsningen for antall bølgelengder mellom mottaker og satellitt. Med denne teknikken kan centimeters-nøyaktighet oppnås under gode forhold [Hofmann-Wellenhof et al., 2007].

Flerveisinterferens

I områder der det ikke er fri sikt til satellittene, som i bymiljøer, daler og i enkelte skogsområder med høye trær vil flerveisinterferensen gjøre det vanskelig å få en nøyaktig posisjonsbestemmelse. I disse områdene er det sannsynlig at signalene blir reflektert før de kommer frem til mottakeren. Refleksjonene fører til at signalenes reisevei blir lenger enn den faktiske avstanden mellom satellitt og mottaker. Den estimerte posisjonen blir dermed feilbestemt [Hofmann-Wellenhof et al., 2007]. I bilbåren datainnsamling kan ikke dette alltid unngås - selv med stadig flere satellitter som følge av flere systemer. Det må derfor benyttes andre metoder til å bestemme posisjonen når GNSS-dekningen er dårlig eller borte. Integrasjon med treghetsnavigasjon (INS) er mye brukt, og da oppnås i tillegg langt høyere målerate enn med satellittnavigasjon alene.

2.4 Treghetsnavigasjon (INS)

Treghetsnavigasjon er en navigasjonsteknikk som måler og beregner endring i posisjon, hastighet og orientering til en sensor. Dette gjøres relativt en kjent starttilstand (posisjon, hastighet og orientering) ved hjelp av akselerometere og gyroskoper. En IMU (Inertial Measurment Unit) er en treghetssensor som typisk består av ett akselerometer og ett gyroskop på hver på de tre aksene i sensoren (s-frame). Når IMUen kombineres med en datamaskin til å regne om observasjonene til posisjonsestimeter kalles det et INS (*Inertial Navigation System*) [Woodman, 2007].

Skrogmonterte systemer (strapdown)
I et skrogmontert system er IMUen fastmontert i kjøretøyets skrog. Det betyr at alle målingene er gitt relativt b-frame. De målte vinkelhastighetene fra gyroskopene blir integrert slik at orienteringen kan estimeres. De målte akselerasjonene transformeres til en global referanseramme ved hjelp av den estimerte orienteringen. De transformerte akselerasjonene integreres to ganger for å estimere posisjonen. Skrogmonterte systemer krever noe mer regnekraft enn gimbalmonterte systemer, men har til gjengjeld en enklere konstruksjon. Dette har ført til at skrogmonterte systemer er den mest brukte formen for INS [Woodman, 2007].

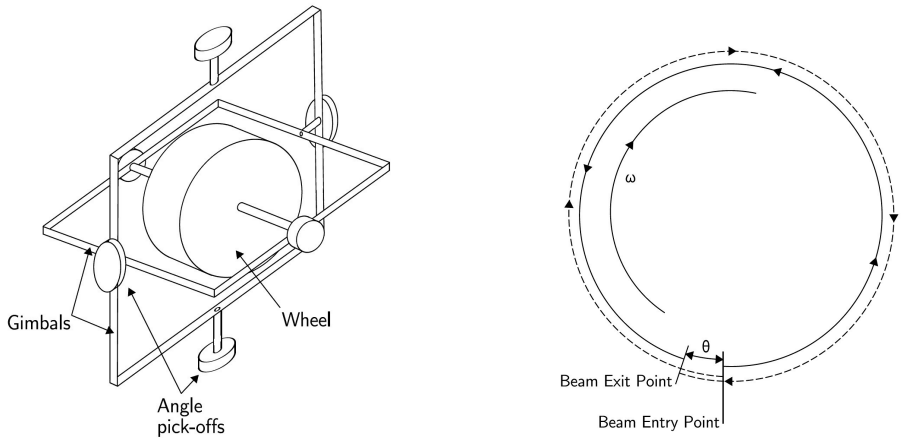
2.4.1 Gyroskop

Et gyroskop brukes til å måle vinkelhastigheter. Det er i hovedsak tre ulike typer gyroskop [Woodman, 2007]:

- Mekaniske gyroskoper består av et spinnende hjul som er festet i to gimbaler for å kunne rotere om alle tre aksene, se figur 2.1a. På grunn av *bevaringen av spinn* vil det spinnende hjulet motstå endringer i gyroskopets orientering.
- Optiske gyroskoper kan deles inn i fiberoptiske- og ringlaser-gyroer. Prinsippene for disse er begge basert på *Sagnaceffekten*. Det går ut på å sende lys/laser i ulik retning gjennom en løkke og inn i en detektor.

Ved eventuell rotasjon vil strålene bruke ulik tid, og det vil oppstå et interferensmønster i detektoren. Intensiteten til interferensmønsteret omregnes til rotasjons- eller vinkelhastighet.

- MEMS (Mikroelektromekaniske systemer) baserte gyroskoper er veldig små, enkle og billige sensorer. Disse består typisk av vibrerende elementer. Når gyroskopet blir rotert vil det oppstå en ny vibrasjon som følge av Coriolis-effekten, og denne vibrasjonen brukes til å beregne vinkelhastigheten.



(a) Mekanisk gyroskop. (b) Optisk gyroskop.

Figur 2.1: Ulike gyroskoper [Woodman, 2007].

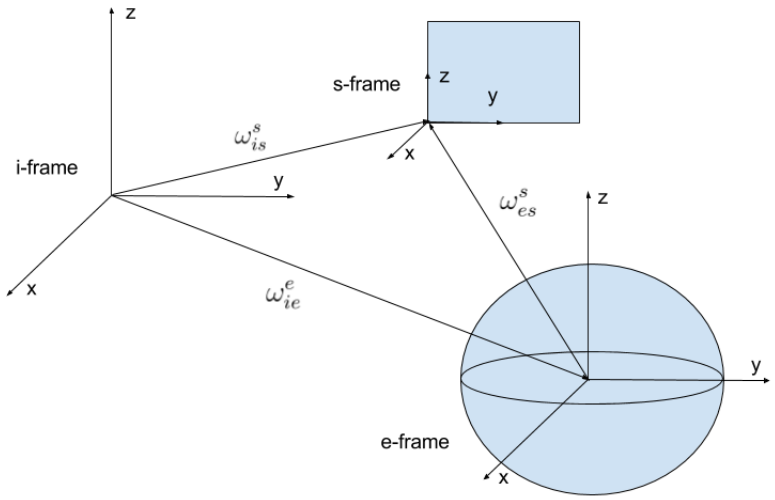
Observasjonsligningen for et gyroskop kan settes opp slik:

$$\omega_{es}^{s} = \omega_{is}^{s} - C_{e}^{s}\omega_{ie}^{e} \tag{2.2}$$

der

- ω_{es}^{s} er sensorens vinkelhastighet relativt e-frame, gitt i s-frame.
- ω_{is}^{s} er sensorens målte vinkelhastighet relativt i-frame, gitt i s-frame.
- ω_{ie}^{e} er jordens rotasjonshastighet relativt i-frame, gitt i e-frame. Rotasjonsmatrisen C_{e}^{s} transformererer jordens rotasjonshastighet slik at denne også blir gitt i s-frame.

Figur 2.2 gir en visuell oversikt over disse referanserammene og hvordan de henger sammen.



Figur 2.2: Sammenhengen mellom i-frame, e-frame og s-frame.

2.4.2 Akselerometer

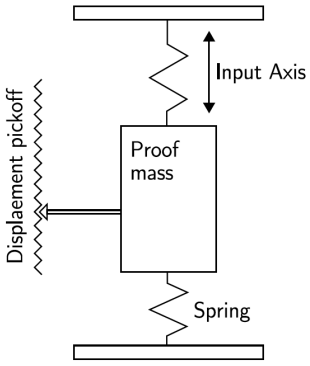
Et akselerometer er en sensor som måler akselerasjoner langs en akse. Det er i hovedsak tre ulike typer akselerometere [Woodman, 2007]:

- Mekaniske akselerometere kan enten være fjærbaserte eller servobaserte pendelakselerometere. Av disse er sistnevnte mest brukt, og teknikken går ut på å holde en testmasse i ro ved å tilføre spenning. Ved eventuelle akselerasjoner langs den aktuelle aksen må spenningen økes eller minkes for at testmassen skal holdes på samme sted. Med denne spenningsendringen kan akselerasjonen beregnes. Figur 2.3a viser et fjærbasert akselerometer. Ved eventuelle akselerasjoner vil sensoren bevege på seg, mens testmassen holder seg i ro relativt i-frame. Sensorens bevegelse relativt testmassen omregnes til akselerasjon.

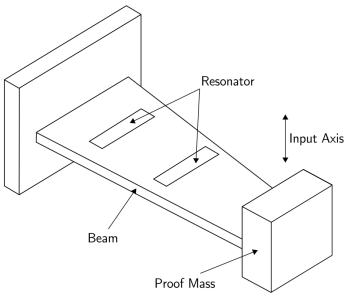
- Solid state akselerometere kan igjen deles i flere undergrupper. Felles er at de er små, pålitelige og robuste. Et eksempel er *Surface Acoustic Wave* (SAW) akselerometer. Disse består av en liten bjelke som kun

er festet i den ene enden, og i andre enden er det festet en testmasse. Bjelken resonerer ved en viss frekvens, og ved eventuelle akselerasjoner langs den aktuelle aksen vil frekvensen endres. Ved å måle frekvensendringen kan akselerasjonen beregnes. Se figur 2.3b.

- MEMS (Mikroelektromekaniske systemer) baserte akselerometere kan deles i to undergrupper. Den første er mekaniske akselerometere som måler forflytningen av en testmasse. Den andre måler frekvensendring i et vibrerende element. Det er altså de samme prinsippene som for tradisjonelle akselerometere, men i en enklere og billigere form. MEMS-akselerometerene er heller ikke like nøyaktige.



(a) Mekanisk fjærbasert akselerometer.



(b) Surface Acoustic Wave akselerometer.

Figur 2.3: Eksempel på akselerometere [Woodman, 2007].

Den grunnleggende observasjonsligningen for akselerasjoner i i-frame kan settes opp slik:

$$\ddot{x}^i = f^i + g^i_{att} \tag{2.3}$$

der

- $\ddot{x}^i$ er akselerasjonen langs den aktuelle aksen.
- f^i er den målte spesifikke kraften.
- g^g_{att} er gravitasjonsakselerasjonen.

2.4.3 Navigasjonsligninger i e-frame

Navigasjon og posisjonering med treghetssensorer er basert på integrasjon av observerte akselerasjoner. I dette avsnittet tas det utgangspunkt i en posisjonsligning. Denne deriveres til en hastighetsligning og videre til en akselerasjonsligning. Utledningene kan også sees i [Jekeli, 2001, kap. 4.3, s. 123 - 126]. En posisjon i i-frame, x^i, er gitt ved posisjonen i e-frame, x^e, venstremultiplisert med en rotasjonsmatrise, C_e^i:

$$x^i = C_e^i x^e \tag{2.4}$$

Hastigheten, $\dot{x}^i$, fås ved å derivere posisjonen i ligning 2.4 med hensyn på tiden:

$$\dot{x}^i = \dot{C}_e^i x^e + C_e^i \dot{x}^e \tag{2.5}$$

der den deriverte av transformasjonsmatrisen er avhengig av jordrotasjonen, $\Omega_{ie}^e = [\omega_{ie}^e \times]$:

$$\dot{C}_e^i = C_e^i \Omega_{ie}^e \tag{2.6}$$

slik at det endelige uttrykket for hastigheten blir:

$$\dot{x}^i = C_e^i \Omega_{ie}^e x^e + C_e^i \dot{x}^e \tag{2.7}$$

Akselerasjonen, $\ddot{x}^i$, fås ved å derivere hastigheten i ligning 2.7 med hensyn på tiden:

$$\ddot{x}^i = C_e^i (\Omega_{ie}^e \Omega_{ie}^e + \dot{\Omega}_{ie}^e) x^e + 2 C_e^i \Omega_{ie}^e \dot{x}^e + C_e^i \ddot{x}^e \tag{2.8}$$

Ved antagelse om konstant jordrotasjonshastighet ($\dot{\Omega}_{ie}^e = 0$) og substitusjon av $\ddot{x}^i$ (fra ligning 2.3), fås et uttrykk for akselerasjonen i e-frame, $\ddot{x^e}$:

$$C_e^i \ddot{x}^e = \ddot{x}^i - C_e^i(\Omega_{ie}^e \Omega_{ie}^e)x^e + 2C_e^i \Omega_{ie}^e \dot{x}^e \tag{2.9}$$

$$\ddot{x}^e = f^e + g_{att}^e - \Omega_{ie}^e \Omega_{ie}^e x^e + 2\Omega_{ie}^e \dot{x}^e \tag{2.10}$$

$$\ddot{x}^e = f^e + g^e - 2\Omega_{ie}^e \dot{x}^e \tag{2.11}$$

der $g^e = g_{att}^e - \Omega_{ie}^e \Omega_{ie}^e x^e$ representerer tyngdens akselerasjon (summen av gravitasjonsakselerasjon og sentripetalakselerasjon) uttrykt i e-frame.

Posisjon og hastighet i e-frame
De endelige ligningene for posisjon og hastighet er gitt under. Disse kan videre løses ved numerisk integrasjon.

$$\frac{d}{dt}x^e = \dot{x}^e \tag{2.12}$$

$$\frac{d}{dt}\dot{x}^e = f^e + g^e - 2\Omega_{ie}^e \dot{x}^e \tag{2.13}$$

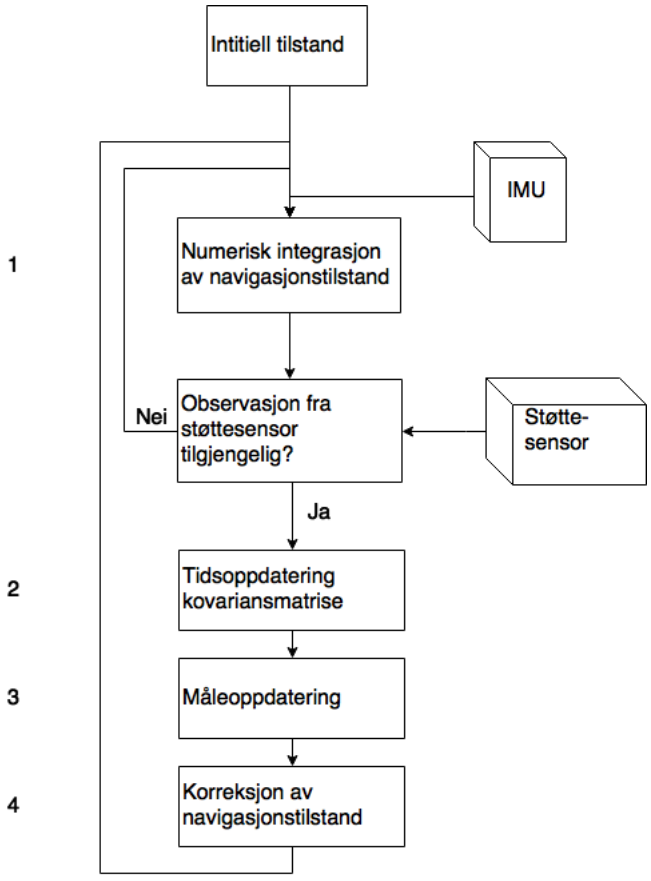
Orientering i e-frame
I et skrogmontert treghetssystem måler gyroene vinkelhastigheter i s-frame relativt i-frame. Ved å integrere vinkelhastighetene kan rotasjonsmatrisen C_s^e beregnes. Denne benyttes videre til å transformere akselerasjonsmålingene. Den endelige ligningen for orienteringen er gitt slik (ligning 2.6):

$$\dot{C}_s^e = C_s^e \Omega_{es}^s \tag{2.14}$$

2.5 Treghetsnavigasjon integrert med støttesensorer

GNSS har som beskrevet i delkapittel 2.3 høy absolutt nøyaktighet. Målefrekvensen er derimot relativt lav (typisk 1-2Hz). I tillegg vil tidvis dårlig

sikt til satellittene forringe kvaliteten på grunn av flerveisinterferens. Ved å kombinere satellittnavigasjon med treghetsnavigasjon vil disse problemene reduseres. IMUer har en målefrekvens på typisk 100 - 500 Hz, og vil dermed kunne gi posisjonsestimater mellom hver GNSS-observasjon. Når disse teknikkene integreres i et Kalmanfilter vil driften i treghetssensorene estimeres, samtidig som eventuelle feilmålinger i satellittnavigasjonen kan glattes over. Dette gjøres ved at de estimerte INS-tilstandene vektes mot de observerte støttesensor-tilstandene. Figur 2.4 viser gangen i dette, mens avsnitt 2.5.1 gir en punktvis beskrivelse av alle delene, samt utledning av de aktuelle ligningene.



Figur 2.4: Flytdiagram for integrasjon av treghetsnavigasjon med andre sensorer, som f. eks. GNSS.

2.5.1 Kalmanfilter

I systemet som beskrives her er det avvikene mellom INS-tilstandene og støttesensor-tilstandene som tas inn i Kalmanfilteret. Altså ikke målingene i seg selv. Dette er en effektiv metode for å oppnå et lineært system og for å unngå komplekse tilpasninger av filteret. Denne måten å sette opp systemet på kalles for et *komplementærfilter* [Farrell, 2008, s. 161 - 162].

1: Numerisk integrasjon av navigasjonstilstand
For å estimere tilstandene ved tidspunkt k må de inkrementelle målingene fra IMUen fremskrive tilstandene ved tidspunkt k-1. For posisjon og hastighet gjøres dette med trapesmetoden for numerisk integrasjon. Disse utledningene kan sees i [Jekeli, 2001, kap. 4.3, s. 134 - 138]. De beregnede akselerasjonene ($\ddot{x}^s$) i s-frame transformeres til e-frame.

$$\ddot{x}^e = \overline{C}^e_s \ddot{x}^s - (2\Omega^e_{ie}\dot{x}^e - C^e_g g^g)\delta t \tag{2.15}$$

der rotasjonsmatrisen ($\overline{C}^e_s$) representerer et gjennomsnitt for oppdateringsintervallet (δt) og defineres slik:

$$\overline{C} = C^e_s(t_{k+1})[I - \frac{1}{2}(\rho^s\times)]) \tag{2.16}$$

der ρ^s representerer de inkrementelle vinkelendringene fra gyroskopmålingene ($\Delta\theta^s_{is}$) korrigert for jordrotasjonen:

$$\rho^s = \Delta\theta^s_{is} - C^s_e\omega^e_{ie}\delta t \tag{2.17}$$

og C^e_s kan fremskrives ved rekkeutvikling av de korrigerte vinkelendringene [Titterton et al., 2004, s. 311 - 312]:

$$\sigma = \sqrt{\rho_1^2 + \rho_2^2 + \rho_3^2} \tag{2.18}$$

$$\Sigma = [\sigma\times] \tag{2.19}$$

$$A = I + \frac{sin(\sigma)}{\sigma}\Sigma + \frac{(1 - cos(\sigma)}{\sigma^2}\Sigma^2 \tag{2.20}$$

$$C^e_s(t_{k+1}) = C^e_s(t_k)A \tag{2.21}$$

Posisjonen og hastigheten kan dermed oppdateres slik:

$$x^e(t_{k+1}) = x^e(t_k) + \dot{x}^e(t_k)\delta t + \ddot{x}^e\frac{\delta t}{2} \tag{2.22}$$

$$\dot{x}^e(t_{k+1}) = \dot{x}^e(t_k) + \ddot{x}^e \tag{2.23}$$

På denne måten blir posisjonen og hastigheten estimert ved hjelp av forrige tilstand og målingene fra IMUen. Merk at posisjonen oppdateres med den gjennomsnittlige hastigheten for intervallet.

2: Tidsoppdatering kovariansmatrise
Den numeriske integrasjonen vil gå så lenge det kommer observasjoner fra IMUen. Når det kommer observasjoner fra en støttesensor blir gjort en tidsoppdatering av kovariansmatrisen. Utgangspunktet for dette er en førsteordens differensialligning. Følgende utledninger kan også sees i [Brown and Hwang, 1997, kap. 5.3, s. 198 - 204]:

$$\dot{x}(t) = F(t)x(t) + G(t)u(t) \tag{2.24}$$

der

- x(t) er en tilstandsvektor.
- F(t) og G(t) er kjente matriser.
- u(t) er en vektor med støy (Gaussisk hvit støy).

Utledning av leddene i disse matrisene kan sees i avsnitt 2.5.2.

Løsningen på ligning 2.24 er gitt slik:

$$x(t) = \Phi(t, t_0)x(t_0) + \int_{t_0}^{t} \Phi(t, t')G(t')u(t')dt' \tag{2.25}$$

der $\Phi(t, t_0)$ kalles for en transisjonsmatrise. Denne fremskriver tilstanden $x(t)$ fra t_0 til t og er gitt slik:

$$\Phi(t, t') = e^{F(t-t')} \tag{2.26}$$

Ligning 2.26 kan løses ved rekkeutvikling:

$$\Phi(t, t') = I + F(t - t') + \frac{1}{2!}(F(t - t'))^2 + \frac{1}{3!}(F(t - t'))^3 + ... \tag{2.27}$$

En tidsoppdatering av tilstanden er gitt slik:

$$x_k = \Phi(t_k, t_{k-1})x_{k-1} + \omega_k \tag{2.28}$$

der ω_k er:

$$\omega_k = \int_{t_{k-1}}^{t^k} \Phi(t, t')G(t')u(t')dt' \tag{2.29}$$

Kovariansmatrisen til ω_k:

$$Q_k = \int_{t_{k-1}}^{t_k} \Phi(t, \eta)G(\eta)WG^T(\eta)\Phi^T(t, \eta)d\eta \tag{2.30}$$

Den tidsoppdaterte kovariansmatrisen er gitt slik:

$$P_k = \Phi(t_k, t_{k-1})P_{k-1}\Phi^T(t_k, t_{k-1}) + Q_k \tag{2.31}$$

Dette er modellens estimat på kovariansmatrisen.

3: Måleoppdatering
Her tas målingene (fra støttesensor) inn i systemet og det beregnes en måleoppdatering. Det betyr at både tilstandene og kovariansmatrisen oppdateres. Merk at dette er ligningene for et *extended Kalmanfilter*. Som nevnt er det differansen mellom den estimerte (a priori) INS-tilstanden og tilstanden beregnet fra støttesensoren som tas inn i Kalmanfilteret. Følgende utledninger kan også sees i [Brown and Hwang, 1997, kap. 9.1, s. 343 - 348] og [Farrell, 2008, kap. 4.8, s. 144 - 146].

Sammenhenengen mellom a priori tilstanden ($\tilde{x}_k$), den sanne tilstanden (x_k) og korreksjonen (δx_k) er gitt slik:

$$x_k = \tilde{x}_k + \delta x_k \tag{2.32}$$

En vektor z_k med målinger er relatert til tilstandsvektoren (x_k) gjennom den ikke-lineære funksjonen h_k slik:

$$z_k = h_k(x_k) + v_k \tag{2.33}$$

der v_k er tilfeldig observasjonsstøy.

En lineær tilnærming er gitt ved:

$$\delta z_k = z_k - h(\tilde{x}_k) \tag{2.34}$$
$$= h(x_k) + v_k - h(\tilde{x}_k) \tag{2.35}$$
$$\approx H_k \delta x_k + v_k, \tag{2.36}$$

der

$$H_k = \frac{\partial h}{\partial x} \mid_{x=\tilde{x}_k} \tag{2.37}$$

er designmatrisen med partiell deriverte.

Ettersom feilestimatene blir brukt til å oppdatere tilstanden, blir $\delta\tilde{x}_k \equiv 0 \Rightarrow x_k = \tilde{x}_k$ (fra ligning 2.32). Dette gjør at tidsoppdateringen til tilstandsvektoren er triviell og kan hoppes over.

De endelige ligningene for extended Kalmanfilter er dermed:

$$\tilde{P}_k = \Phi_{k-1}\hat{P}_{k-1}\Phi_{k-1}^T + Qd_{k-1} \tag{2.38}$$
$$K_k = \tilde{P}H_k^T[H_k\tilde{P}_kH_k^T + R_k]^{-1} \tag{2.39}$$
$$\delta\hat{x}_k = K_k\delta z_k \tag{2.40}$$
$$\hat{P}_k = (I - K_hH_k)\tilde{P}_k \tag{2.41}$$

der

- $\tilde{P}$ er den tidsoppdaterte kovariansmatrisen.
- K_k er Kalman gain - funksjonen som vekter modellens estimat mot målingene.
- $\delta\hat{x}_k$ er korreksjonen til tilstandene.
- $\hat{P}$ er den måleoppdaterte kovariansmatrisen.

4: Korreksjon av navigasjonstilstand
De estimerte korreksjonene brukes deretter til å korrigere navigasjonstilstanden (tilsvarer ligningene 2.48 - 2.53 i avsnitt 2.5.2):

$$\hat{x}_k^e = \tilde{x}_k^e + \delta\hat{x}_k^e \tag{2.42}$$
$$\hat{\dot{x}}^e = \tilde{\dot{x}}^e + \delta\hat{\dot{x}}^e \tag{2.43}$$
$$\hat{C}_s^e = (I + \hat{\Psi}^e)\tilde{C}_s^e \tag{2.44}$$

der $\hat{\Psi}^e = [\hat{\psi}^e\times]$ er systemets orientering.

I tillegg blir også sensorbiasene ($\hat{b}_f^s$ og $\hat{b}_\omega^s$) og tyngdekorreksjonen ($\hat{g}^e$) korrigert:

$$\hat{b}_f^s = \tilde{b}_f^s + \delta\hat{b}_f^s \tag{2.45}$$
$$\hat{b}_\omega^s = \tilde{b}_\omega^s + \delta\hat{b}_\omega^s \tag{2.46}$$
$$\hat{g}^e = \tilde{g}^e + \delta\hat{g}^e \tag{2.47}$$

Siden sensorbiasene blir korrigert etter hver måleoppdatering vil de påfølgende INS-tilstandene bli stadig bedre.

2.5.2 Lineariserte feilligninger

For å kunne bruke Kalmanfilterligningene som en optimal estimeringsteknikk (for å oppnå minimum varians) må uttrykkene for matrisene i ligning 2.24 utledes. Utgangspunktet er et sett med lineære perturberte variabler [Farrell, 2008, s. 392].

$$\tilde{x}^e = x^e - \delta x^e \tag{2.48}$$
$$\tilde{\dot{x}}^e = \dot{x}^e - \delta\dot{x}^e \tag{2.49}$$
$$\tilde{f}^s = f^s + \delta f^s \tag{2.50}$$
$$\tilde{\omega}^s = \omega^s + \delta\omega^s \tag{2.51}$$
$$\tilde{C}_s^e = (I - \Psi^e)C_s^e \tag{2.52}$$
$$\tilde{g}^e = g^e - \delta g^e \tag{2.53}$$

der b_f^s og b_ω^s er sanne feil i henholdsvis den spesifikke kraften og gyromålingene. Perturbasjonene legges til ligningene for posisjon (ligning 2.12), hastighet (ligning 2.13) og orientering (ligning 2.14).

Perturbert hastighetsligning:

$$\frac{d}{dt}\delta\dot{x} = -2\Omega_{ie}^e\delta\dot{x}^e + \Psi C_s^e f^s - C_s^e\delta f^s + \delta g^e \tag{2.54}$$

der δg^e representerer feil i tyngdekorreksjonen og er gitt slik:

$$\delta g^e = \Gamma^e\delta x^e - \Omega_{ie}^e\Omega_{ie}^e\delta x^e + \delta g_\epsilon^e \tag{2.55}$$

der $\Gamma^e \equiv \frac{\partial g_{att}^e}{\partial x^e}$ og δg_ϵ^e er en liten gravitasjonsanomali.

Den siste utledningen er for orienteringsfeilen. Ved å legge perturbasjonen i ligning 2.52 til ligning 2.14 fås følgende perturberte ligning for orienteringen:

$$\delta\dot{C}_s^e = \delta C_s^e\Omega_{es}^s + C_s^e\delta\Omega_{es}^s \tag{2.56}$$

Full utledning av feilligningene kan sees i [Farrell, 2008, kap. 11.4, s. 392 - 396].

Endelig feilligning for posisjon:

$$\frac{d}{dt}\delta x^e = \delta\dot{x}^e \tag{2.57}$$

Endelig feilligning for hastighet:

$$\frac{d}{dt}\delta\dot{x}^e = -2\Omega_{ie}^e\delta\dot{x}^e + (\Gamma^e - \Omega_{ie}^e\Omega_{ie}^e)\delta x^e - (f^e\times)\psi - C_s^e\delta f^s + \delta g_\epsilon^e \tag{2.58}$$

Endelig feilligning for orientering:

$$\dot{\psi}^e = -\Omega_{ie}^e\psi^e - C_s^e\delta\omega_{is}^s \tag{2.59}$$

Feilligningene for posisjon, hastighet og orientering kan samles og settes på matriseform. Dette er den mest generelle formen av ligning 2.24.

$$\frac{d}{dt}\begin{bmatrix}\delta x^e \\ \delta \dot{x}^e \\ \psi^e\end{bmatrix} = \begin{bmatrix}0 & I & 0 \\ (\Gamma - \Omega^e_{ie}\Omega^e_{ie}) & -2\Omega^e_{ie} & -f^e\times \\ 0 & 0 & -\Omega^e_{ie}\end{bmatrix}\begin{bmatrix}\delta x^e \\ \delta \dot{x}^e \\ \psi^e\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 \\ 0 & -C^e_s & I \\ -C^e_s & 0 & 0\end{bmatrix}\begin{bmatrix}\delta\omega^s_{is} \\ \delta f^s \\ \delta g^e_\epsilon\end{bmatrix} \quad (2.60)$$

Del I

Nullhastighetsoppdatering

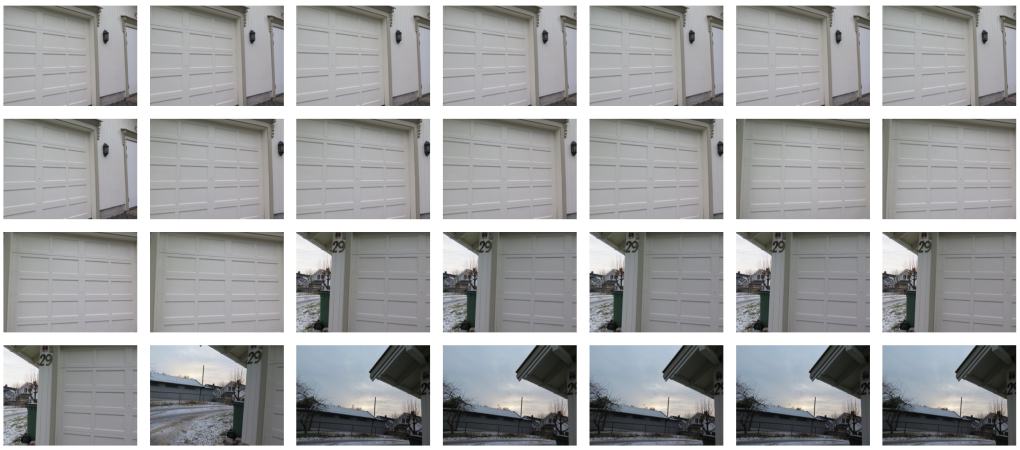
Kapittel 3

Verktøy og metode

3.1 Introduksjon

På grunn av integralets natur vil en feil i den beregnede akselerasjonen forplante seg kvadratisk til posisjonsestimatet. Feilen i den beregnede akselerasjonen kan deles i to: den ene skyldes målefeil i selve sensoren (sensorbias) og den andre skyldes feil i orienteringsbestemmelsen av sensoren. Når systemet er i bevegelse kan disse feilene skilles fra hverandre, mens når kjøretøyet er i ro kan de ikke skilles [Jekeli, 2001, s. 160-161]. Dette viser viktigheten av å kunne avgjøre med andre sensorer om systemet har vært i ro eller ikke. Vanligvis vil satellittnavigasjonen og hjulodometeret kunne benyttes til dette, men ved fravær av begge disse (for eksempel i en jernbanetunell) må andre metoder benyttes. Dette kapittelet vil se på funksjoner og metoder som benytter pikselbasert matching mellom bilder til å detektere eventuell nullhastighet.

Testdatasett
For å teste ut og vise funksjonene som er brukt til å løse denne delen av oppgaven er det benyttet et sett med bilder av en garasjeport, se figur 3.1. Bildene ble tatt med jevne mellomrom av et kamera plassert på et stativ. Med ujevne mellomrom ble kameraet rotert mot venstre. Rotasjonen ble gjort for å minne om faktisk bevegelse når kameraet er plassert på et kjøretøy.



Figur 3.1: Testdatasettet.

3.2 OpenCV - pikselbasert matching

OpenCV har blitt et populært verktøy til blant annet videoovervåkning. Det finnes mye dokumentasjon og mange forumtråder med ulike problemstillinger innenfor dette på internett. Deteksjon av bevegelse i overvåkningsammenheng er en problemstilling som på sett og vis er den samme som deteksjon av nullhastighet. For å løse denne delen av oppgaven er det dermed tatt utgangspunkt i metodene som ble brukt til å løse disse problemstillingene. En enkel antagelse er at nullhastighet fører til liten eller ingen endring i pikselverdiene mellom to bilder. Motsatt tilsier dette at bevegelse vil føre til større endring i pikselverdiene mellom to bilder. Utgangspunktet blir da å bruke funksjoner som kan beregne differansen mellom to bilder.

3.2.1 Funksjoner til pikselbasert matching

Imread og VideoCapture
Funksjoner som leser inn enkeltbilder eller bildesekvenser (video) som matriser. Dersom det leses fra en videofil blir ett og ett bilde lest inn. Figur 3.2 viser tre av bildene fra bildedatasettet (gjort om til svart hvitt). Figur 3.2a og 3.2b er *nesten* helt like. I figur 3.2c har kameraet blitt rotert noe mot venstre.

(a) Originalbilde 1.

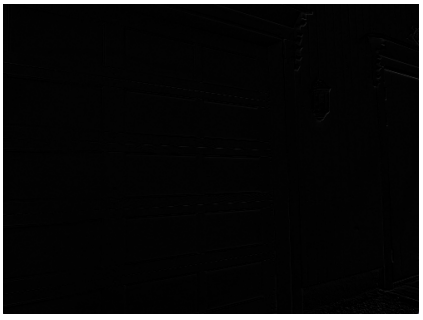
(b) Originalbilde 2.

(c) Originalbilde 3.

Figur 3.2: Originalbilder.

Subtract

Denne funksjonen beregner differansen mellom to bilder. De av pikslene som har lik verdi i begge bildene vil representeres med verdien null, altså svart. Motsatt vil pikslene med ulik verdi i de to bildene representeres med verdier over null (opp til og med én) avhengig av hvor stor differansen mellom de opprinnelige pikselverdiene var.



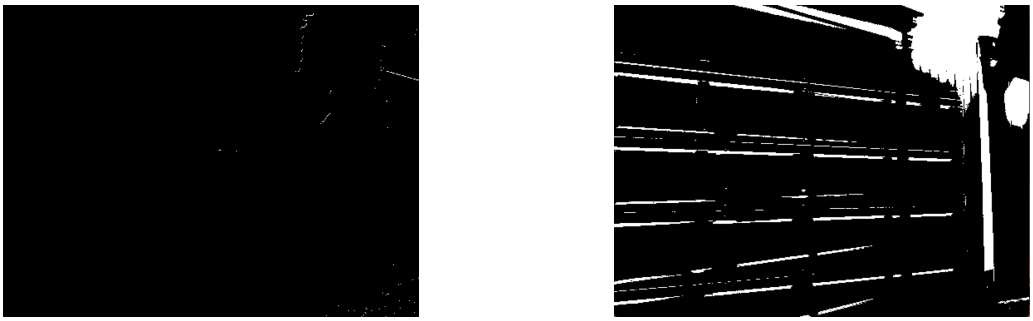
(a) Differansebilde 1.

(b) Differansebilde 2.

Figur 3.3: Bildedifferanser.

Figur 3.3a viser differansen mellom Originalbilde 1 og 2. Som forventet er dette bildet tilnærmet svart (ved nøye ettersyn kan det skimtes enkelte lyse piksler som følge av bevegelse i kameraet under bildeeksponering). Figur 3.3b viser differansen mellom Originalbilde 2 og 3, og her er forskjellen mellom de to bildene tydelig. Spesielt er lampen godt synlig.

Threshold
Denne funksjonen er en terskeltest som tar utgangspunkt i et bilde, gjerne i gråtoner. Det settes en terskelverdi som hvert piksel blir testet mot. Dersom pikselverdien er større enn terskelverdien blir pikselen tildelt verdien 1 (hvit). Dersom pikselverdien er mindre enn terskelverdien blir pikselen tildelt verdien 0 (svart). Med utgangspunkt i et differansebilde vil de hvite pikslene etter terskeltesten tilsvare de pikslene med endret verdi.



(a) Threshold bilde 1. (b) Threshold bilde 2.

Figur 3.4: Threshold bilder.

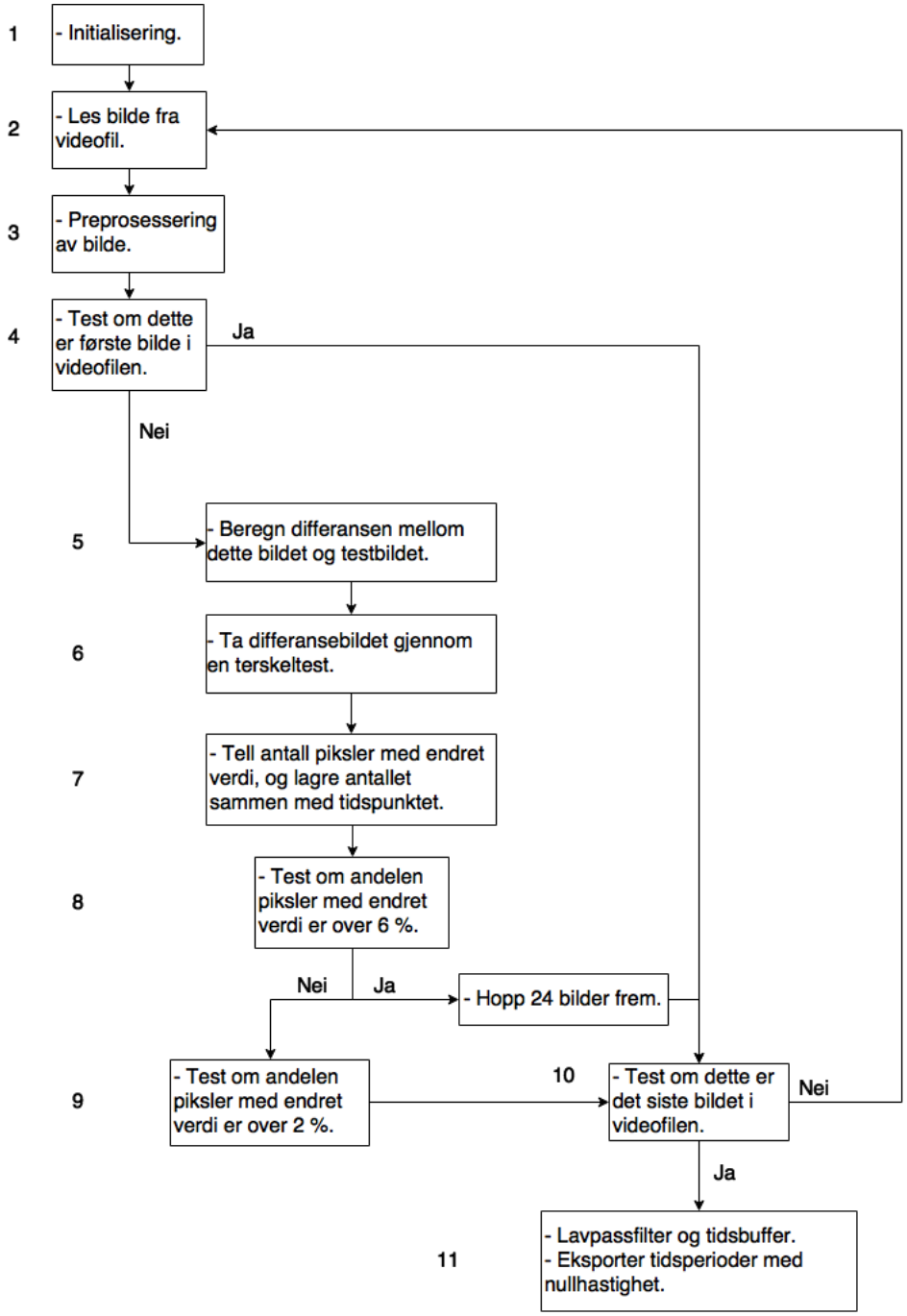
Figur 3.4a viser resultatet etter threshold-funksjonen for differansebilde 1 (figur 3.3a). Figur 3.4b viser resultatet etter threshold-funksjonen for differansebilde 2 (figur 3.3b). Begge figurene viser at de av pikslene som fikk endret pikselverdi etter subtract-funksjonen sees mye tydeligere etter denne terskeltesten.

3.2.2 Alternativ metode

Bruk av pikselbasert matching er en forholdsvis enkel tilnærming til deteksjon av nullhastighet, og resultatene i kapittel 4 viser at det er enkelte utfordringer knyttet til dette. En alternativ metode hadde vært å ta i bruk objektbasert matching slik som i Del 2 av denne oppgaven. Den metoden baserer seg på å tracke nøkkelpunkter fra bilde til bilde, og ut i fra det beregne den *relative orienteringen* mellom bildene.

3.3 Beskrivelse av metoden

Ettersom problemet er løst med programmering i Python vil det i dette delkapittelet bli gitt en beskrivelse av programmeringsgangen. Metoden som er brukt baserer seg på funksjonene som ble beskrevet i avsnitt 3.2.1, og målet er å detektere nullhastighet. Dette vil gjøres ved å se på differansen mellom to og to bilder, for så å beregne antall piksler som har ulik verdi i de to bildene. Dersom tilstrekkelig mange piksler har lik verdi kan det antas å ikke ha vært bevegelse mellom de to bildene. Figur 3.5 gir en visuell oversikt over dataflyten i programmet, og avsnitt 3.3.1 gir en punktvis gjennomgang av alle programmets deler.



Figur 3.5: Dataflyt nullhastighetsoppdatering.

3.3.1 Punktvis beskrivelse av programmet

1: Initialisering
I denne første funksjonen defineres følgende parametere:

- Videofilen som skal leses inn.
- Ny bildehøyde som senere benyttes til å kutte vekk nederste del av bildene som leses inn. På den måten eksluderes bilens panser og dashbord dersom kameraet er montert på innsiden av bilen.
- Totalt antall bilder i videofilen blir beregnet.

2: Les bilde fra videofil
Dette er den ytre løkken i programkoden, og denne leser inn bilde for bilde med funksjonen VideoCapture.

3: Preprosessering av bilde
Når bildet er lest inn blir den nye bildehøyden fra initialiseringen lagt til. Med den nye høyden blir det totale antall piksler i bildet beregnet. I tillegg blir bildenummeret og tidspunktet definert, samt bildet gjort om til svart hvitt for å ha færre verdier å jobbe med i de påfølgende operasjonene.

4: Test om dette er første bilde i videofilen
Det første bildet blir satt som testbilde. Det vil si at de neste bildene vil testes mot dette bildet frem til det registreres en bevegelse.

5: Beregn differansen mellom dette bildet og testbildet
Med funksjonen Subtract blir bildet som nå er lest inn subrahert med testbildet.

6: Ta differansebildet gjennom en terskeltest
Differansebildet fra forrige funksjon blir tatt inn i Threshold-funksjonen. Her vil alle pikslene med en viss intensitet bli klassifisert som piksler med endret verdi.

7: Tell antall piksler med endret verdi, og lagre antallet sammen med tidspunktet
Antall piksler med endret verdi blir telt opp, og andelen i forhold til det totale antall piksler blir beregnet. Viktigheten av å eksludere bilens dashbord og panser kommer til synlighet her. Disse er relativt konstante, og ville gitt et feilaktig inntrykk av andelen piksler uten endret verdi. Andelen piksler med endret verdi, samt tidspunktet, blir lagret i hver sin liste.

8: Test om andelen piksler med endret verdi er over 6 %
Dersom andelen piksler med endret verdi er over 6 % kan det med stor sikkerhet antas at kjøretøyet er i bevegelse. For å effektivisere prosesseringen hoppes det dermed over tjuefire bilder. Selve terskelen på 6 % er kommet frem til ved testing, se delkapittel 4.2.1, og kan varieres avhengig av datasett og behov.

9: Test om andelen piksler med endret verdi er over 2 %
Dersom andelen piksler med endret verdi er under 2 % antas det at kjøretøyet er i ro. Over 2 % antas det at kjøretøyet er i bevegelse. Merk at ved pikselendring under 2 % vil testbildet forbli det samme helt til pikselendringen overstiger 2 %. Dette er for at lav hastighet, der pikselendringen mellom to bilder etter hverandre kan være lav, ikke skal registreres om nullhastighet. Selve terskelen på 2 % er kommet frem til ved testing, se delkapittel 4.2.1, og kan varieres avhengig av datasett og behov.

10: Test om dette er det siste bildet i videofilen
Hvis dette ikke var det siste bildet i videofilen vil den ytre løkken lese inn et nytt bilde. Ellers avbrytes den ytre løkken, og programmet går inn den den avsluttende delen.

11: Lavpassfilter og tidsbuffer. Eksporter tidsperioder med nullhastighet
Hvis dette var det siste bildet i videofilen blir prosessen avsluttet. Listen med andel endrede piksler blir tatt gjennom et lavpassfilter (se avsnitt 3.3.2) for å fjerne eventuelle uteliggere. Etter lavpassfiltreringen er det satt et tidskrav for hva som registreres som nullhastighet (se avsnitt 3.3.3). Resultatene i delkapittel 4.2 viser at disse uteliggersøkene er nødvendige for å oppnå gode resultater.

3.3.2 Lavpassfilter

Det er lagt til et lavpassfilter for å glatte over potensiell støy (for eksempel et tre som har beveget seg mye mellom to bilder). Dette gjøres med et såkalt *glidende gjennomsnitt* der det regnes et gjennomsnitt av fire og fire verdier som ligger etter hverandre. Fire verdier ga et godt kompromiss mellom ønsket glatting og signalforsinkelse.

3.3.3 Tidskriterie for nullhastighet

For å være helt sikker på at det ikke feilaktig blir registrert nullhastighet er det satt et kriterie om at minimum hundre og syttifem bilder etter hverandre ligger under 2% terskelen. Dette tilsvarer minimum syv sekunder i videofilen. For hver periode som registreres som nullhastighet blir det også lagt til en buffer på ett sekund i hver ende før resultatet eksporteres. Dette gjøres på grunn av en usikkerhet knyttet til kameraets tidsstempling av bildene. Og det er igjen bedre at deler av de stillestående periodene ikke blir registrert som nullhastighet enn at perioder med bevegelse blir registrert som nullhastighet.

Kapittel 4

Resultater

4.1 Nullhastighetsoppdatering med pikselbasert matching

I forrige kapittel ble programmeringsgangen, samt en beskrivelse av de ulike funksjonene for pikselbasert matching vist sammen med et forholdsvis enkelt bildedatasett. I dette kapitlet skal programmet testes med en videofil fra et oppdrag i Trondheim. Videoen er tatt fra dashbordet til Terratec Lynx Mobile Mapper (kameraet rettet forover) i desember 2016, og den er omtrent ti minutter lang.

Kameraet som er brukt er av typen Sony Actioncam med integrert GPS-mottaker. Dette er et forholdsvis enkelt kamera som kan filme i Full HD. Det er uvisst om kameraet benytter GPS-mottakeren til å tidsstemple bildene i videoen. Det antas dermed at tidsstemplingen blir gjort av en intern oscillator, og at GPS-mottakeren kun brukes til georeferering. Kvalitetet til den interne oscillatoren, samt den absolutte tidsstemplingen er ukjent, og det er dermed knyttet en del usikkerhet til tidsstemplingen av hvert enkelt bilde.

Selve turen inneholder både korte og lange stopp. Siden det er i et bymiljø er det mye trafikk og annen bevegelse. Dette har ført til en klassifisering av fire typer scenarier som avhenger av homogeniteten til og bevegelsen i omgivelsene. De neste avsnittene viser eksempler på disse scenariene.

4.1.1 Kjøretøyet er i bevegelse - inhomogene omgivelser

(a) Kjøretøy i bevegelse.

(b) 22.74 % piksler med endret verdi.

Figur 4.1: Kjøretøy i bevegelse - inhomogene omgivelser.

Figur 4.1a viser det faktiske bildet, mens figur 4.1b viser pikslene som har fått endret verdi siden forrige bilde. Her er kjøretøyet midt i en sving, med mange objekter rundt (svært inhomogene omgivelser) og det er dermed hele 22.74% av pikslene som har endret verdi.

4.1.2 Kjøretøyet er i ro - lite bevegelse i omgivelsene

(a) Kjøretøy i ro.

(b) 0.54 % piksler med endret verdi.

Figur 4.2: Kjøretøy i ro - lite bevegelse i omgivelsene.

Her har bilen stoppet for en barnevogn som skal over veien. Figur 4.2a viser det faktiske bildet, mens figur 4.2b viser pikslene som har fått endret verdi siden forrige bilde. Her er det tydelig at det kun er barnevognen som er i bevegelse mellom de to bildene (noe endring kan også skimtes i treet oppe til høyre), og det er da kun 0.54% av pikslene som har fått endret verdi.

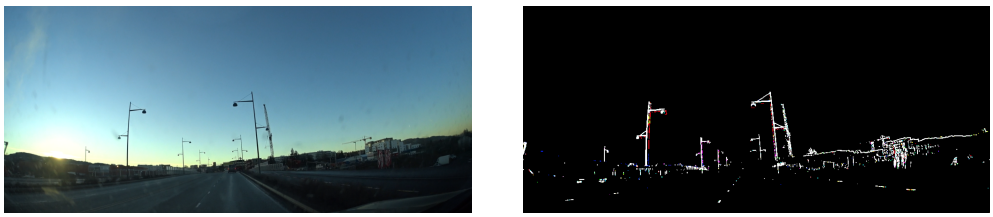
4.1.3 Kjøretøyet er i ro - mye bevegelse i omgivelsene

(a) Kjøretøy i ro. (b) 2.49 % piksler med endret verdi.

Figur 4.3: Kjøretøy i ro - mye bevegelse i omgivelsene.

Her står bilen på samme sted som i forrige scenarie, men her har 2.49% av pikslene har fått endret verdi siden forrige bilde. Grunnen til dette synes i figur 4.3b, der det er tydelig at treet har beveget seg mellom de to bildene.

4.1.4 Kjøretøyet er i bevegelse - homogene omgivelser



(a) Kjøretøy i bevegelse. (b) 1.78 % piksler med endret verdi.

Figur 4.4: Kjøretøy i bevegelse - homogene omgivelser.

I figur 4.4 har bilen kommet til et område med homogene omgivelser (få objekter). Figur 4.4a viser det faktiske bildet, som kun inneholder veien, himmelen, lyktestolper og noen fjerne bygninger. Endringen fra forrige bilde vises i figur 4.4b, og det er kun 1.78% av pikslene som har fått endret verdi. Det er en lavere andel enn da bilen sto i ro i scenario 3.

4.2 Resultater og bestemmelse av terskelverdier

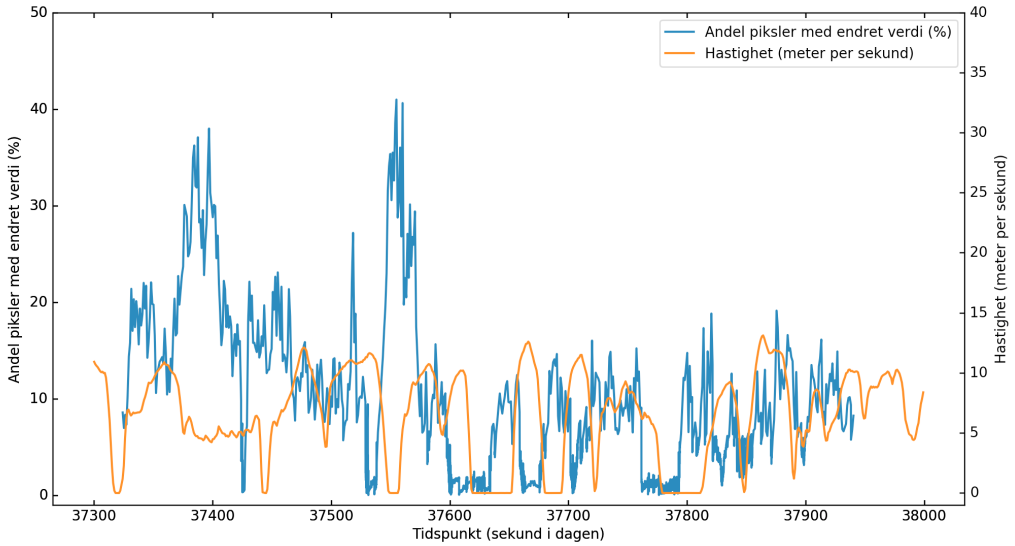
Tabell 4.1 viser resultatene fra de fire scenariene. Blant disse er det enklest å skille de to første fra hverandre. Dessverre vil et datasett fra et bymiljø som i dette eksempelet bestå av en blanding av alle de fire scenariene. For å avgjøre hva som skal registreres som nullhastighet må det settes en terskelverdi. Tallene i tabell 4.1 gir ikke nok grunnlag til å bestemme terskelverdier alene.

Scenario	**Endrede piksler**
1: Kjøretøy i bevegelse (mange objekter)	22.74 %
2: Kjøretøy i ro (få objekter i bevegelse)	0.54 %
3: Kjøretøy i ro (mange objekter i bevegelse)	2.49 %
4: Kjøretøy i bevegelse (få objekter)	1.78 %

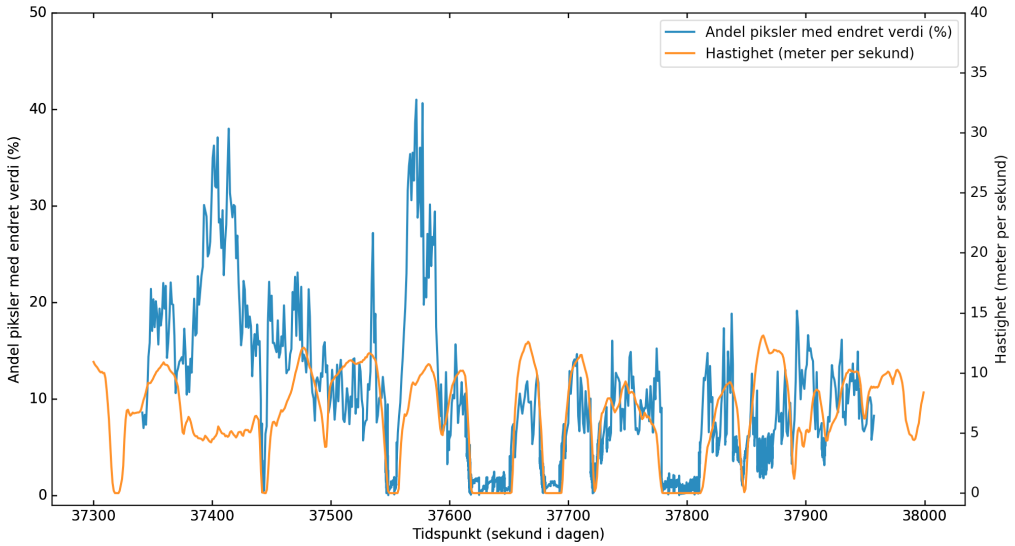
Tabell 4.1: Eksempel på de fire scenariene med andel endrede piksler.

Figur 4.5 viser andelen piksler med endret verdi sammen med bilens faktiske hastighet (prosessert fra GNSS/INS-løsningen). Dette er plottet mot tidspunkt (sekund i dagen). I videofilen er det markert at videoen starter kl 10.22.03, altså 37232 sekunder ut i dagen. Ved direkte sammenligning med GNSS/INS-løsningen er det tydelig at dette tidspunktet er feil. Tidspunktene der andelen pikselendringer er omtrent null burde sammenfalt med tidspunktene for faktisk nullhastighet. I stedet kan det se ut som en tidsforskyvelse på i underkant av et halvt minutt. Dette kan tyde på at kameraet følger UTC-tid, mens GNSS/INS-løsningen følger GPS-tid. Per desember 2016 lå UTC-tid sytten sekunder etter GPS-tid på grunn av tillagte skuddsekunder[1]. I figur 4.6 er disse sekundene korrigert for. Da ser det ut til at tidspunktene for faktisk nullhastighet og tidspunktene der andelen pikselendringer er omtrent null sammenfaller.

[1] https://en.wikipedia.org/wiki/Leap_second



Figur 4.5: Andel piksler med endret verdi plottet med faktisk hastighet - ikke korrigert for skuddsekunder.



Figur 4.6: Andel piksler med endret verdi plottet med faktisk hastighet - korrigert for skuddsekunder.

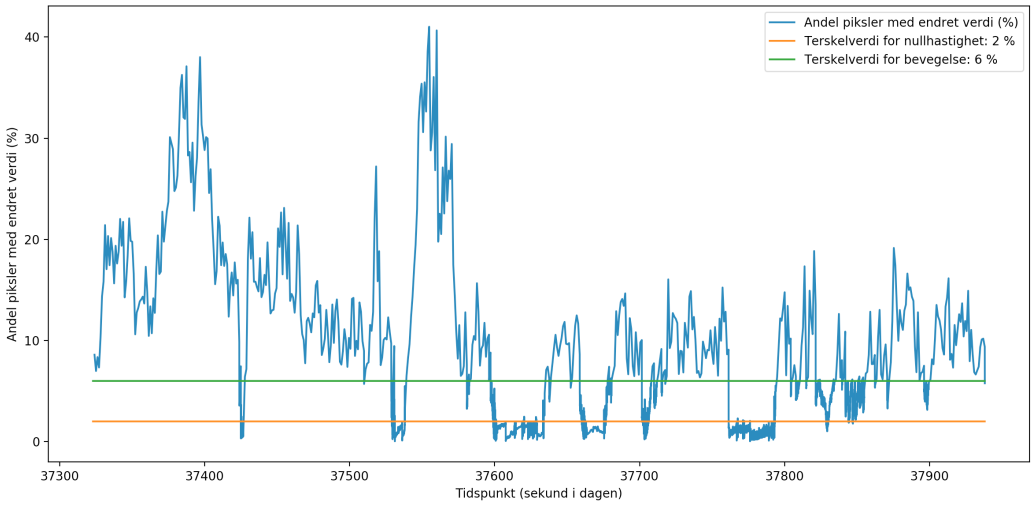
4.2.1 Terskelverdier

Av figurene 4.5 og 4.6 kan det se ut til at mesteparten av bevegelsen fører til at minimum 6-7 % av pikselene får endret verdi. Som beskrevet i punkt åtte under avsnitt 3.3.1 er det ønskelig med en terskelverdi for hva som garantert er bevegelse. Med verdier over denne terskelverdien vil programmet hoppe over tjuefire bilder før det gjøres en ny test. Ved å sette denne terskelverdien til 6 % oppnås effektiv prosessering uten å gå glipp av eventuelle perioder med nullhastighet. Dersom en bildedifferanse gir mindre enn 6 % pikselendring vil alle bildene testes.

Av figurene 4.5 og 4.6 kan det også sees at perioder med nullhastighet fører til at maks 2-3 % av pikselene får endret verdi. Ved å sette terskelverdien for nullhastighet til 2 % unngås feilaktige klassifiseringer.

Uten lavpassfilter

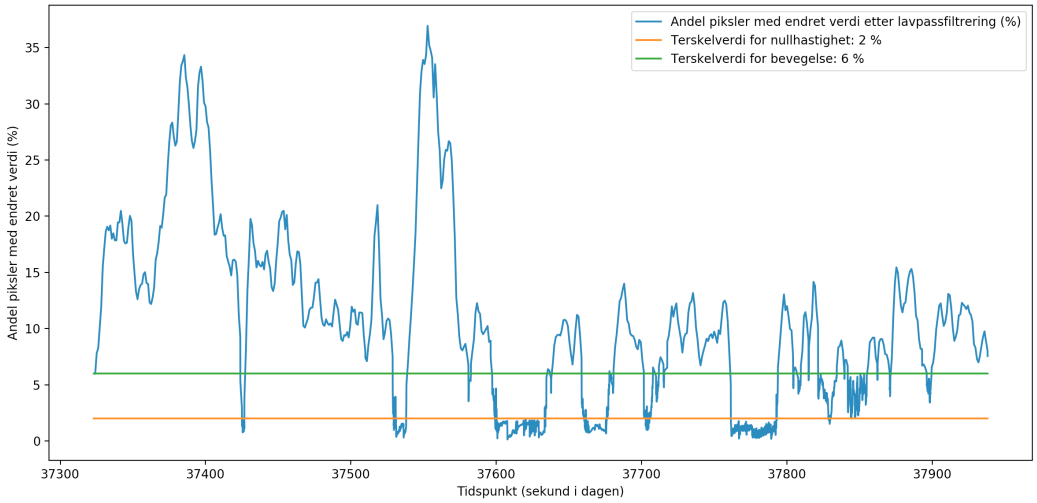
Figur 4.7 viser andelen endrede piksler sammen med de to terskelverdiene på 6 og 2 %. Dette fører til at enkelte topper på grafen havner over terskelverdien for nullhastighet. Lavpassfilteret som ble beskrevet i avsnitt 3.3.2 vil kunne glatte over disse toppene slik at hver periode blir registrert som sammenhengende nullhastighet.



Figur 4.7: Andel piksler med endret verdi plottet med terskelverdiene.

Med lavpassfilter

Resultatet med lavpassfiltreringen kan sees i figur 4.8. Her er de største toppene på grafen blitt jevnet ut, og alle periodene med nullhastighet havner under terskelverdien på 2 %. Det er fortsatt enkelte tidspunkt med bevegelse som havner under terskelverdien, men disse vil ikke bli registrert som nullhastighet på grunn av tidskriteriet som ble beskrevet i avsnitt 3.3.3.



Figur 4.8: Andel piksler med endret verdi etter lavpassfiltrering plottet med terskelverdiene.

4.3 Oppsummering Del 1

Disse resultatene viser at det er mulig å bruke kamera som støttesensor for å detektere nullhastighet. Metoden i dette avsnittet baseres på en forholdsvis enkel antagelse om at bevegelse fører til at en stor andel piksler får endret verdi mellom to bilder. Dette har vist seg å stemme til en viss grad, men som beskrevet i delkapittel 4.2 er det usikkerhet knyttet til grensetilfellene (avsnitt 4.1.2 og 4.1.4). For at denne metoden skal gi tilfredsstillende resultater må det dermed innføres en form for lavpassfilter og et tidskriterie. Tidskriteriet vil også gjøre programmet mer robust mot feil tidsstempling i kameraet.

Del II

Visuell odometri

Kapittel 5

Teori

5.1 Visuell odometri

Visuell odometri ble for alvor et kjent begrep etter artikkelen *Visual Odometry* av David Nistér, Oleg Naroditsky og James Bergen i 2004 [Nistér et al., 2004]. Selve navnet ble valgt på grunn av likheten til hjulbasert odometri, der antall hjulomdreininger blir integrert over tid. Visuell odometri er basert på det samme prinsippet. Teknikken gir ingen absolutt posisjonsbestemmelse, men estimerer inkrementelle posisjonsendringer. Dette gjøres ved å beregne den relative orienteringen mellom to og to bilder tatt av et kamera som er montert på kjøretøyet. Det skilles mellom bruk av ett og to kameraer. Med kun ett kamera vil målestokken i utgangspunktet være ukjent, mens med to kameraer kan denne estimeres [Scaramuzza and Fraundorfer, 2012].

5.1.1 Tidligere resultater

Som nevnt innledningsvis lanserte David Nistér, Oleg Naroditsky og James Bergen artikkelen *Visual Odometry* i 2004 [Nistér et al., 2004]. Der viser de at det er mulig å oppnå svært gode resultater med visuell odometri. En tur gjennom skogen på 366 meter (målt med GPS) ble estimert til 372 meter av det visuelle odometeret. Dette gir et avvik på under to prosent. Tatt i

betraktning at datasettet bestod av totalt 2944 bilder var dette resultatet meget godt [Nistér et al., 2004]. Riktignok ble det benyttet kameraer av bedre kvalitet enn det som er brukt i denne oppgaven. Likevel fører det til en forventning om at visuell odometri potensielt kan bidra som støttesensor til et treghetsnavigasjonssystem.

5.2 Målestokk- og rotasjonsuavhengige nøkkelpunkter

I dette delkapittelet vil teorien bak SIFT-algoritmen [Lowe, 2004] (*Scale-Invariant Feature Transform*) bli beskrevet. Dette er en algoritme som detekterer nøkkelpunkter som er invariante til målestokk og rotasjoner. Det betyr at punktene kan benyttes til å matche bilder som har ulik målestokk og orientering. I visuell odometri er dette gunstig siden kameraet stort sett er i bevegelse og objekter vil avbildes ulikt selv i to påfølgende bilder. Merk at SIFT kun er en av flere ulike algoritmer som gjør dette, men prinsippene er forholdsvis like og formålet er det samme. Delkapittel 6.2 gir en oversikt over ulike algoritmer.

5.2.1 Deteksjon av nøkkelpunkter

Det første steget i algoritmen er å finne områder i bildet som er uavhengig av målestokk og orientering. Dette gjøres ved å beregne en Gaussisk differanse (Engelsk: Difference of Gaussian - DoG).

Gaussisk differanse

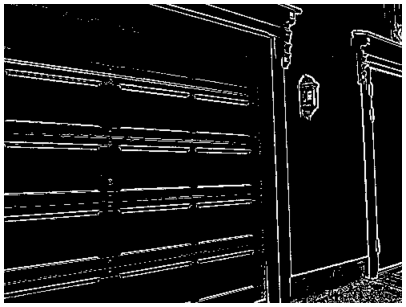
For å finne nøkkelpunkter som er uavhgengige av målestokk tas det først utgangspunkt i et antall versjoner av et bilde med ulike grad av skarphet. Ved å beregne differanser mellom de ulike skarphetsnivåene, fås resultatbilder som viser konturer av områdene der *intensiteten* i bildet endres, se figur 5.1. I disse områdene vil lokale ekstremverdier typisk være potensielle nøkkelpunkter.

Figur 5.1a viser et bilde med original skarphet. I figur 5.1b er bildet gjort uskarpt. Figur 5.1c viser differansen mellom figur 5.1a og 5.1b. Differansen er såpass liten at den såvidt kan skimtes i dette bildet. I figur 5.1d er differansebildet tatt gjennom Threshold-funksjonen fra avsnitt 3.2.1 for å synliggjøre forskjellen på de to bildene.

(a) Originalt bilde.

(b) Uskarpt bilde.

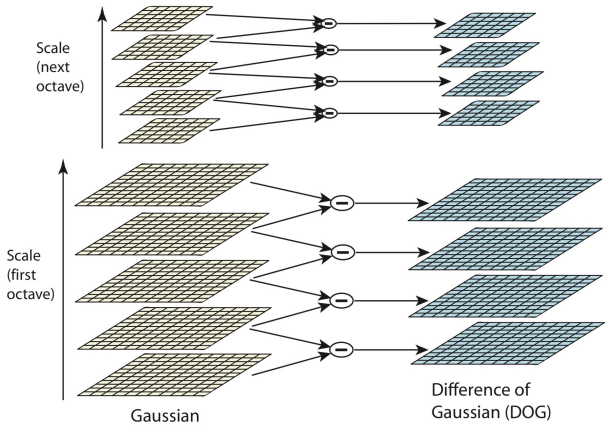
(c) Differansebilde.



(d) Threshold bilde.

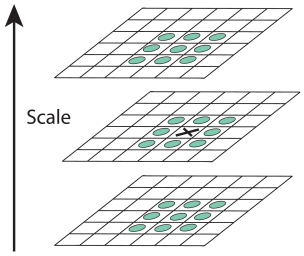
Figur 5.1: Bilder av garasjeport for å illustrere resultatet av en Gaussisk differanse.

Det neste steget er å gjøre akkurat det samme, men med en annen *skalering* av bildet. Dette gjør at hvert nøkkelpunkt blir lagret i versjoner med ulik målestokk. Nøkkelpunktene kan dermed finnes igjen i et annet bilde med en annen målestokk. Figur 5.2 viser hele denne prosessen. Et sett med bilder som har samme målestokk, men ulik skarphetsgrad kalles en oktav, og et sett med oktaver kalles for en bildepyramide [Lowe, 2004].



Figur 5.2: Gaussisk differanse [Lowe, 2004].

Deteksjon av lokale ekstremverdier For å detektere de lokale maksimum- og minimumspunktene blir hvert punkt sammenlignet med sine 26 naboer (åtte i sitt eget bilde, og ni i skaleringen over og under). Punktet blir kun valgt dersom det er større eller mindre enn alle sine naboer [Lowe, 2004]. Figur 5.3 illustrerer dette.



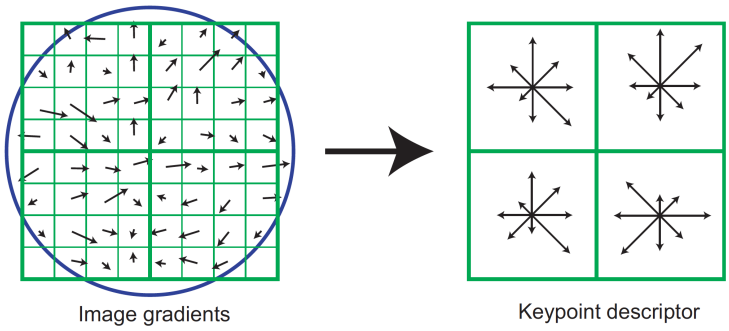
Figur 5.3: Deteksjon av lokalt ekstrempunkt [Lowe, 2004].

Den Gaussiske differansen finner typiske blob[1]- og hjørnelignende bildestrukturer, men gir også god respons langs kanter (se figur 5.1). Det som ønskes av et nøkkelpunkt er at det skal være lokalt distinkt. Det vil si at det ikke skal være mulig å finne et likt punkt i området rundt, slik at det blir enkelt å finne det igjen i et annet bilde. Et punkt langs en kant er ikke lokal distinkt og det vil være nærmest umulig å finne akkurat det samme punktet i et annet bilde. Derfor må eventuelle kanter lukes ut.

[1]Blob-strukturer: områder i bilder som skiller seg ut i for eksempel lysstyrke og farge. https://en.wikipedia.org/wiki/Blob_detection

5.2.2 Beregning av deskriptorer

For hvert nøkkelpunkt som blir detektert blir det også beregnet en *deskriptor*. Deskriptoren gir nøkkelpunktet en detaljert beskrivelse slik at det blir enklere å finne det igjen i et annet bilde. Dette gjøres ved at størrelsen og orienteringen til bildegradientene blir samplet rundt nøkkelpunktet, se venstre del av figur 5.4. Disse samplingene blir akkumulert til orienteringshistogrammer som illustreres til høyre i figuren [Lowe, 2004].



Figur 5.4: Deskriptor [Lowe, 2004].

Merk at figur 5.4 er noe forenklet. Den illustrerer at det er fire orienteringshistogrammer rundt nøkkelpunktet, som gir til sammen trettito verdier for deskriptoren. En faktisk SIFT-deskriptor består av seksten histogrammer à åtte verdier. Altså hundre og tjueåtte verdier totalt. Alle disse verdiene blir lagret i en vektor som senere kan brukes til matching [Lowe, 2004].

5.2.3 Matching av deskriptorer

For å matche to bilder med hverandre benyttes deskriptorene som ble beskrevet i forrige avsnitt. Deskriptorvektorene blir testet mot hverandre i de to bildene, og de som ligger nærmest hverandre i det 128 dimensjonale vektorrommet antas å tilhøre det samme nøkkelpunktet. I denne oppgaven vil de matchede deskriptorene brukes til å beregne den relative orienteringen mellom bildene, men de kunne også blitt brukt til for eksempel å sy sammen flere bilder til et panoramabilde. Ulike matchefunksjoner, samt valg av de beste matchene omtales i avsnitt 6.2.2 og under punkt 6 i avsnitt 6.3.1.

5.3 Relativ orientering

Relativ orientering betyr hvordan to eller flere bilder er orientert i forhold til hverandre. Den relative orienteringen mellom to bilder kan beregnes kun ved å se på punkter som finnes i begge bildene, og den gir ingen informasjon om den absolutte orienteringen. Det finnes i dag ulike algoritmer for å bestemme den relative orienteringen, og disse baserer seg typisk på hvor mange matchede punkter som må være med i beregningen. Dersom kameraets indre orienteringselementer er ukjente må åtte punkter være med i beregningen, men dersom disse er kjent trengs kun fem matchede punkter[2]. Sistnevnte er kjent som fempunkts-algoritmen, og det er denne som er brukt i denne oppgaven. For å bestemme kameraets indre orienteringselementer må det foretas en kalibrering av kameraet.

5.3.1 Kamerakalibrering

De indre orienteringselementene består av en kamerakonstant og to prinsipalpunkter (samt noen flere parametere som ikke er nødvendige til dette formålet) [Andersen, 2003, kap. 4.3]

- Kamerakonstanten (c) er avstanden fra kameralinsen til kameraets projeksjonssenter.
- Prinsipalpunktet er der normalen fra kameraets projeksjonssenter skjærer bildeplanet.

For å bestemme disse verdiene må det tas bilder av et karakteristisk mønster (for eksempel et sjakkbrett) fra flere vinkler. Med korrekt verktøy (OpenCV har funksjoner som gjør dette) blir de indre orienteringselementene bestemt, og det blir tre færre ukjente i den relative orienteringen.

[2]Forelesning av Cyrill Stachniss om relativ orientering: https://www.youtube.com/watch?v=Cl1Kne_WqPg&t=1557s

5.3.2 Fempunkts-algoritmen og RANSAC

RANSAC (RANdom SAmple Consensus) er en iterativ metode som i visuell odometri benyttes til å velge de punktene som gir den beste relative orienteringen mellom to bilder[3]. Et typisk utgangspunkt vil være å ha et forholdsvis stort antall matchede punkter i de to bildene. Ut i fra et slik datasett består RANSAC-algoritmen av følgende to hovedsteg [Zuliani, 2009]:

1. **Hypotese**
   Fem tilfeldige punkter velges ut og benyttes til å beregne den relative orienteringen mellom bildene.

2. **Testing**
   Resten av punktene i datasettet blir testet med den beregnede relative orienteringen.

Disse stegene blir gjort iterativt helt til sannsynligheten for å finne en bedre relativ orientering faller under en bestemt terskel. Terskelen, eller antall iterasjoner som må gjøres, avhenger av andelen uteliggere i datasettet. Denne kan utledes slik:

Sannsynligheten for suksess (P) er gitt slik:

$$P = (1 - e)^S \tag{5.1}$$

der e er andelen uteliggere og S er antall samplinger (fem i tilfelle for relativ orientering med fempunkts-algoritmen).

Sannsynligheten for ikke-suksess er gitt ved $1 - P$:

$$1 - P = (1 - (1 - e)^S)^T \tag{5.2}$$

der T er antall iterasjoner. Med P spesifisert etter ønske og e og S kjent kan T beregnes:

$$log(1 - P) = T \cdot log(1 - (1 - e)^S) \tag{5.3}$$

$$\Rightarrow T = \frac{log(1 - P)}{log(1 - (1 - e)^S)} \tag{5.4}$$

[3]Forelesning av Cyrill Stachniss om SIFT og RANSAC: https://www.youtube.com/watch?v=oT9c_LlFBqs&t=1717s

Ligning 5.4 viser at antall samplinger (S) i stor grad påvirker hvor mange iterasjoner som må gjøres. Dette gjør fempunkts-algoritmen til et godt valg for å beregne relativ orientering.

Sammenligning av å bruke fem eller åtte punkter til relativ orientering med RANSAC (99 % suksessansynlighet og 30 % uteliggere):

$$5\ punkter: \qquad T = \frac{log(1-0.9)}{log(1-(1-0.3)^5)}$$
$$T \approx 13$$

$$8\ punkter: \qquad T = \frac{log(1-0.9)}{log(1-(1-0.3)^8)}$$
$$T \approx 39$$

Med åtte punkter må det i dette tilfelle gjøres tre ganger så mange itererasjoner som med fem punkter. Med en høyere andel uteliggere ville dette blitt enda verre.

5.4 Oppsett og løsning av normalligningssystem

Forrige delkapittel viste hvordan den relative orienteringen mellom to bilder kan beregnes med fem matchede punkter. I dette delkapittelet skal rotasjonen og translasjonen fra den relative orienteringen forbedres gjennom en utjevning. Dette gjøres ved at bildene deles inn i partier på tjuefem[4]. Alle målingene (punktene) i de tjuefem bildene blir relatert til et foreløpig verdenssystem i det første bildet i partiet gjennom *kolinearitetsligningen*. Punktene (som alle er målt i minst to bilder) får dermed flere overbestemmelser i form av at alle observasjonene knyttes sammen gjennom et *normalligningssystem*.

[4]Antall bilder i hvert parti omtales nærmere i delkapittel 6.3.

5.4.1 Kolinearitetsprinsippet

Forholdet mellom bilde- og terrengkoordinater er gitt ved kolinearitetsligningen [Andersen, 1994, kap. 5]:

$$X_T = X_O + \lambda C^T X_B \tag{5.5}$$

der

- X_T er foreløpige terrengkoordinater.
- X_O er den beregnede translasjonsvektoren fra den relative orienteringen (projeksjonssenterets koordinater).
- λ er målestokken.
- C er den beregnede rotasjonsmatrisen fra den relative orienteringen.
- X_B er bildemålinger (matchede nøkkelpunkter).

Med ligning 5.5 blir alle bildemålingene relatert til referansesystemet i det første bildet. Med utskrevne matriser blir ligningen slik:

$$\begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} = \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix} + \lambda \begin{bmatrix} c_{11} & c_{21} & c_{31} \\ c_{12} & c_{22} & c_{32} \\ c_{13} & c_{23} & c_{33} \end{bmatrix} \begin{bmatrix} x_B \\ y_B \\ -c \end{bmatrix} \tag{5.6}$$

Det er ønskelig å uttrykke ligning 5.5 med hensyn på bildekoordinatene:

$$X_B = \frac{1}{\lambda} R(X_T - X_O) \tag{5.7}$$

$$\begin{bmatrix} x_B \\ y_B \\ -c \end{bmatrix} = \lambda \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} x_T - x_O \\ y_T - y_O \\ z_T - z_O \end{bmatrix} \tag{5.8}$$

Utskrevet gir dette:

$$x_B = \frac{1}{\lambda} \cdot (c_{11}(x_T - x_O) + c_{12}(y_T - y_O) + c_{13}(z_T - z_O)) \tag{5.9}$$

$$y_B = \frac{1}{\lambda} \cdot (c_{21}(x_T - x_O) + c_{22}(y_T - y_O) + c_{23}(z_T - z_O)) \tag{5.10}$$

$$-c = \frac{1}{\lambda} \cdot (c_{31}(x_T - x_O) + c_{32}(y_T - y_O) + c_{33}(z_T - z_O)) \tag{5.11}$$

I ligning 5.11 er kamerakonstanten (c) beregnet ved kamerakalibrering, mens målestokken λ er ukjent. Ligningen kan dermed dermed settes opp som et uttrykk for målestokken:

$$\frac{1}{\lambda} = \frac{-c}{(c_{31}(x_T - x_O) + c_{32}(y_T - y_O) + c_{33}(z_T - z_O))} \tag{5.12}$$

Uttrykket for målestokken kan settes inn i ligning 5.9 og 5.10:

$$x_B = -c \cdot \frac{(c_{11}(x_T - x_O) + c_{12}(y_T - y_O) + c_{13}(z_T - z_O))}{(c_{31}(x_T - x_O) + c_{32}(y_T - y_O) + c_{33}(z_T - z_O))} \tag{5.13}$$

$$y_B = -c \cdot \frac{(c_{21}(x_T - x_O) + c_{22}(y_T - y_O) + c_{23}(z_T - z_O))}{(c_{31}(x_T - x_O) + c_{32}(y_T - y_O) + c_{33}(z_T - z_O))} \tag{5.14}$$

I tradisjonell fotogrammetri kalles disse for *de vanlige ligninger for perspektivisk avbilding*. Disse gir sammenhengen mellom bildemålinger, terrengkoordinater og de ytre orienteringselementene (orienteringen relativt terrenget) for bildet [Andersen, 1994, kap. 5]. I dette tilfelle vil terrengkoordinatene x_T, y_T, z_T, rotasjonsleddene og projeksjonssenterets posisjon x_O, y_O, z_O være ukjente størrelser, mens kamerakonstanten (c) er beregnet ved kamerakalibrering og bildepunktene x_B, y_B er målt.

5.4.2 Feilligninger

Med kolinearitetsprinsippet er sammenhengen mellom bildekoordinater og terrengkoordinater definert. For å bestemme endelige terrengkoordinater, rotasjoner og projeksjonssenter må det settes opp et normalligningssystem. Fullstendig utledning av feilligningene kan sees i [Andersen, 1994, kap. 5].

Ligning 5.7 gjelder for feilfrie målinger. Feilfrie målinger er sjeldent tilfelle, så ligningen kan skrives slik:

$$X' + V = \frac{1}{\tilde{\lambda}(1 + d\lambda)} \delta C \cdot \tilde{C}(\tilde{X}_T + \delta X_T - \tilde{X}_O - \delta X_O) \tag{5.15}$$

der

- $X' = \begin{bmatrix} x' \\ y' \\ -c \end{bmatrix}$ der x' og y' er målte bildekoordinater, c er kamerakonstanten.
- $V = \begin{bmatrix} e_x \\ e_y \\ 0 \end{bmatrix}$ er feilvektor.
- $\tilde{\lambda}$ er foreløpig målestokksfaktor (ligning 5.12).
- $\delta\lambda$ er målestokkskorreksjon slik at $\tilde{\lambda}(1 + \delta\lambda)$ blir utjevnet målestokk.
- $\tilde{C}$ er foreløpig rotasjonsmatrise.
- δC er korreksjonsmatrise slik at $\delta C \cdot \tilde{C}$ blir utjevnet rotasjonsmatrise.
- $\tilde{X}_T$ og $\tilde{X}_O$ er foreløpige koordinater for terrengpunkt og projeksjonssenter.
- δX_T og δX_O er korreksjoner til $\tilde{X}_T$ og $\tilde{X}_O$.

De generelle feilligningene kan settes opp slik [Andersen, 1994, s. 93]:

$$e_x = \frac{1}{\tilde{\lambda}}(c_{11} + \frac{\tilde{x}_B}{c}c_{31})dx_T + \frac{1}{\tilde{\lambda}}(c_{12} + \frac{\tilde{x}_B}{c}c_{32})dy_T + \frac{1}{\tilde{\lambda}}(c_{13} + \frac{\tilde{x}_B}{c}c_{33})dz_T$$
$$- \frac{1}{\tilde{\lambda}}(c_{11} + \frac{\tilde{x}_B}{c}c_{31})dx_O - \frac{1}{\tilde{\lambda}}(c_{12} + \frac{\tilde{x}_B}{c}c_{32})dy_O - \frac{1}{\tilde{\lambda}}(c_{13} + \frac{\tilde{x}_B}{c}c_{33})dz_O$$
$$+ \frac{\tilde{x}_B\tilde{y}_B}{c}d\omega - (c + \frac{\tilde{x}_B\tilde{x}_B}{c})d\phi - \tilde{y}_B d\kappa + (\tilde{x}_B - x') \quad (5.16)$$

$$e_y = \frac{1}{\tilde{\lambda}}(c_{21} + \frac{\tilde{y}_B}{c}c_{31})dx_T + \frac{1}{\tilde{\lambda}}(c_{22} + \frac{\tilde{y}_B}{c}c_{32})dy_T + \frac{1}{\tilde{\lambda}}(c_{23} + \frac{\tilde{y}_B}{c}c_{33})dz_T$$
$$- \frac{1}{\tilde{\lambda}}(c_{21} + \frac{\tilde{y}_B}{c}c_{31})dx_O - \frac{1}{\tilde{\lambda}}(c_{22} + \frac{\tilde{y}_B}{c}c_{32})dy_O - \frac{1}{\tilde{\lambda}}(c_{23} + \frac{\tilde{y}_B}{c}c_{33})dz_O$$
$$- (c + \frac{\tilde{y}_B\tilde{y}_B}{c})d\phi + \frac{\tilde{x}_B\tilde{y}_B}{c}d\omega - \tilde{x}_B d\kappa + (\tilde{y}_B - y') \quad (5.17)$$

Begge disse ligningene må løses for hvert målte bildepunkt. Videre må det settes opp og løses normalligninger for å forbedre de foreløpige verdiene.

5.4.3 Normalligninger

For å sette opp normalligningssystemet må feilligningene 5.16 og 5.17 settes opp på matriseform slik:

$$\delta z = H\delta x - e \quad (5.18)$$

der

- δz er korreksjoner til de foreløpige bildemålingene.
- δx er korreksjoner til de foreløpige projeksjonssenterene, rotasjonene og terrengkoordinatene.
- H er en funksjon som kobler δx og δz.
- e er ukjente feil.

Med utskrevne matriser blir ligningen slik:

$$\begin{bmatrix} x' - \tilde{x}_B \\ y' - \tilde{y}_B \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} & h_{17} & h_{18} & h_{19} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} & h_{26} & h_{27} & h_{28} & h_{29} \end{bmatrix} \begin{bmatrix} dx_0 \\ dy_0 \\ dz_0 \\ d\omega \\ d\phi \\ d\kappa \\ dx_T \\ dy_T \\ dz_T \end{bmatrix} - \begin{bmatrix} e_x \\ e_y \end{bmatrix} \tag{5.19}$$

der leddene i H-matrisen kommer fra feilligningene 5.16 og 5.17, her delt inn i projeksjonssenter, rotasjoner og terrengkoordinater:

Projeksjonssenterets koordinater:

- $h_{11} = -\frac{1}{\tilde{\lambda}}(c_{11} + \frac{\tilde{x}_B}{c} c_{31})$
- $h_{12} = -\frac{1}{\tilde{\lambda}}(c_{12} + \frac{\tilde{x}_B}{c} c_{32})$
- $h_{13} = -\frac{1}{\tilde{\lambda}}(c_{13} + \frac{\tilde{x}_B}{c} c_{33})$
- $h_{21} = -\frac{1}{\tilde{\lambda}}(c_{21} + \frac{\tilde{y}_B}{c} c_{31})$
- $h_{22} = -\frac{1}{\tilde{\lambda}}(c_{22} + \frac{\tilde{y}_B}{c} c_{32})$
- $h_{23} = -\frac{1}{\tilde{\lambda}}(c_{23} + \frac{\tilde{y}_B}{c} c_{33})$

Rotasjonene:

- $h_{14} = \frac{\tilde{x}_B \tilde{y}_B}{c}$
- $h_{15} = -(c + \frac{\tilde{x}_B \tilde{x}_B}{c})$
- $h_{16} = -\tilde{y}_B$
- $h_{24} = -(c + \frac{\tilde{y}_B \tilde{y}_B}{c})$
- $h_{25} = \frac{\tilde{x}_B \tilde{y}_B}{c}$
- $h_{26} = -\tilde{x}_B$

Terrengkoordinatene:

- $h_{17} = \frac{1}{\tilde{\lambda}}(c_{11} + \frac{\tilde{x}_B}{c} c_{31})$
- $h_{18} = \frac{1}{\tilde{\lambda}}(c_{12} + \frac{\tilde{x}_B}{c} c_{32})$
- $h_{19} = \frac{1}{\tilde{\lambda}}(c_{13} + \frac{\tilde{x}_B}{c} c_{33})$
- $h_{27} = \frac{1}{\tilde{\lambda}}(c_{21} + \frac{\tilde{y}_B}{c} c_{31})$
- $h_{28} = \frac{1}{\tilde{\lambda}}(c_{22} + \frac{\tilde{y}_B}{c} c_{32})$
- $h_{29} = \frac{1}{\tilde{\lambda}}(c_{23} + \frac{\tilde{y}_B}{c} c_{33})$

Med dette kan normalligningene settes opp:

$$N = H^T P H$$
$$h = H^T P z$$

P er vektsmatrisen. Utjevningen gjøres etter minste kvadraters metode:

$$d\hat{x} = N^{-1} \cdot h \tag{5.20}$$
$$\hat{x} = \tilde{x} + d\hat{x} \tag{5.21}$$

Korreksjonene som kommer ut av utjevningen brukes til å oppdatere de foreløpige projeksjonssenterkoordinatene, rotasjonene og terrengkoordinatene. Med korrigerte verdier blir utjevningen gjentatt helt til korreksjonene er tilstrekkelig små og løsningen konvergerer. Merk at så lenge målestokken er ukjent vil dette systemet ikke være mulig å løse. Derfor må det legges til en ekstra betingelse om for eksempel avstand mellom første og siste projeksjonssenter. Mer om dette i avsnitt 6.3.2.

Kapittel 6

Verktøy og metode

6.1 Introduksjon

Med teorien i kapittel 5 er grunnlaget for visuell odometri lagt. I dette kapittelet skal teorien benyttes til å lage et program som detekterer og matcher nøkkelpunkter. De matchede nøkkelpunktene blir brukt til å beregne den relative orienteringen mellom bildene.

Testdatasett
For å teste ut og vise de aktuelle funksjonene for objektbasert matching er det tatt utgangspunkt i det samme settet med bilder som i Del 1, se figur 3.1. Kameraet stod på et stativ og det ble tatt bilder manuelt med jevne mellomrom. Med ujevne mellomrom ble kameraet rotert mot venstre. Disse bildene var i utgangspunktet ikke beregnet på å brukes til å teste visuell odometri, og dette gjenspeiles i at det ikke er noen translasjon mellom bildeeksponeringene.

6.2 OpenCV - objektbasert matching

OpenCV består i dag av mange ulike algoritmer for deteksjon og beregning av nøkkelpunkter og deskriptorer. Valget av algoritme avhenger av bruksområdet, ettersom algoritmene yter forskjellig. Teorien i delkapittel 5.2 er i hovedsak gjeldende for SIFT-algoritmen, men mange av prinsippene går igjen også for de andre algoritmene. Felles er også at alle kan brukes til enten matching eller tracking. Under gis en kort beskrivelse av utvalgte algoritmer.

- **SIFT**[1] **(Scale-Invariant Feature Transform)**
  SIFT er en algoritme utviklet av D. Lowe og presentert i artikkelen *Distinctive Image Features from Scale-Invariant Keypoints* i 2004 [Lowe, 2004]. Dette var en av de første nøkkelpunkt-detektorene som ikke baserte seg på hjørnedeteksjon (hjørner er ikke invariante til skalering). SIFT er en av få patenterte algoritmer i OpenCV, og koster dermed penger å bruke.

- **SURF**[2] **(Speeded-Up Robust Features)**
  SURF er en algoritme utviklet av Herbert Bay, Andreas Ess, Tinne Tuytelaars og Luc Van Gool og presentert i artikkelen *SURF: Speeded Up Robust Features* i 2006 [Bay et al., 2008]. Formålet med denne algoritmen var å lage en raskere versjon av SIFT. Blant annet benyttes en annen metode for å gjøre nøkkelpunktene invariante til skalering, fremfor Gaussisk differanse som SIFT baseres på (avsnitt 5.2.1). SURF er på lik linje som SIFT en patentert algoritme.

- **FAST**[3] **(Features from Accelerated Segment Test)**
  FAST er en algoritme utviklet av Edward Rosten og Tom Drummond og presentert i artikkelen *Machine learning for high-speed corner detection* i 2006 [Rosten and Drummond, 2006]. Dette er en veldig effektiv algoritme for deteksjon av hjørnepunkter, men gir ingen invarians til orientering.

[1] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[2] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[3] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html

- **BRIEF**[4] **(Binary Robust Independent Elementary Features)**
  BRIEF er en algoritme utviklet av Michael Calonder, Vincent Lepetit, Christoph Strecha, og Pascal Fua og presentert i artikkelen *BRIEF: Binary Robust Independent Elementary Features* i 2010 [Calonder et al., 2010]. BRIEF beregner kun deskriptorer, og må derfor brukes i kombinasjon med en annen algoritme for deteksjon av selve nøkkelpunktene. Algoritmen er mer effektiv enn SIFT ved at det benyttes en annen form for komprimering som sparer på bruken av minne.

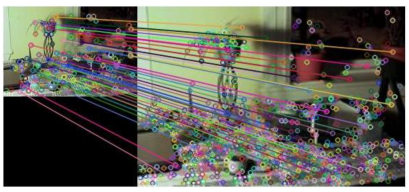
- **ORB**[5] **(Oriented FAST and Rotated BRIEF)**
  ORB er en algoritme utviklet av OpenCV-teamet Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski, og presentert i artikkelen *ORB: An efficient alternative to SIFT or SURF* i 2011 [Rublee et al., 2011]. Algoritmen er en kombinasjon av FAST og BRIEF, men med enkelte forbedringer. Først detekteres nøkkelpunkter med FAST. I ORB er denne algoritmen modifisert til å også være invariant til orientering. Når nøkkelpunktene er detektert benyttes en modifisert versjon av BRIEF, som er mer robust mot rotasjon, til å beregne deskriptorer [Rublee et al., 2011].

[4] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_brief/py_brief.html

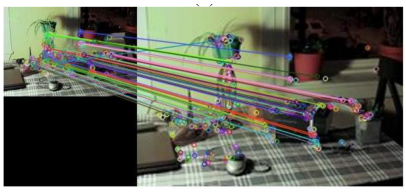
[5] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html

6.2.1 Valg av algoritme for deteksjon av nøkkelpunkter

Ettersom SIFT og SURF er patenterte algoritmer, har valget i denne oppgaven falt på ORB. I følge flere studier som er gjort har ORB vist seg å være både raskest og best, så det antas å være et trygt valg. Riktignok blir det påpekt i [Karami et al., 2015] at ORB-algoritmen har en tendens til å finne nøkkelpunkter rundt bildets senter, se figur 6.1. I tilfelle for visuell odometri kan dette bli et problem dersom kameraet peker forover eller bakover. Da vil typisk punkter midt i bildet svare til objekter som befinner seg langt fra kameraet. Punkter langt unna vil ikke synes å flytte stort på seg mellom hvert bilde. Geometrien blir dermed dårligere enn med objekter som befinner seg nærmere kameraet.



(a) SIFT nøkkelpunkter og matcher - spredt i hele bildet.



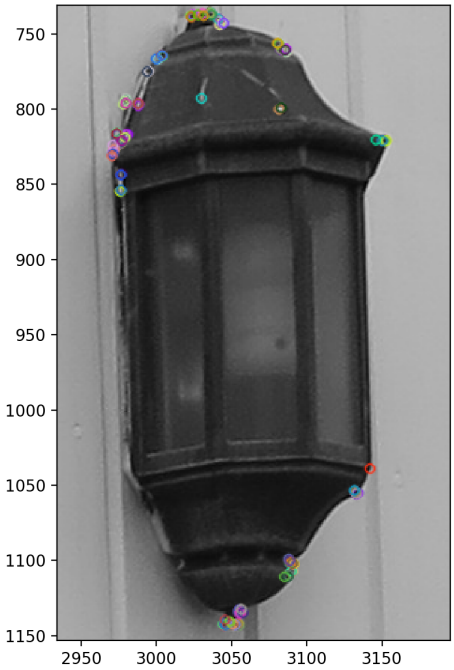
(b) ORB nøkkelpunkter og matcher - sentrert rundt bildets senter.

Figur 6.1: Sammenligning av SIFT og ORB [Karami et al., 2015]

6.2.2 Funksjoner til objektbasert matching

Dette avsnittet gir en kort beskrivelse av de funksjonene som benyttes til å løse denne delen av oppgaven. Teorien er hentet fra dokumentasjonssidene til OpenCV. Merk at funksjonene stort sett heter det samme uavhengig av hvilken algoritme som brukes.

Detect og Compute (ORB)
Dette er funksjonene som detekterer og beregner nøkkelpunkter og deskriptorer i et bilde. Figur 6.2 viser de detekterte nøkkelpunktene på lampehuset i et av testbildene.



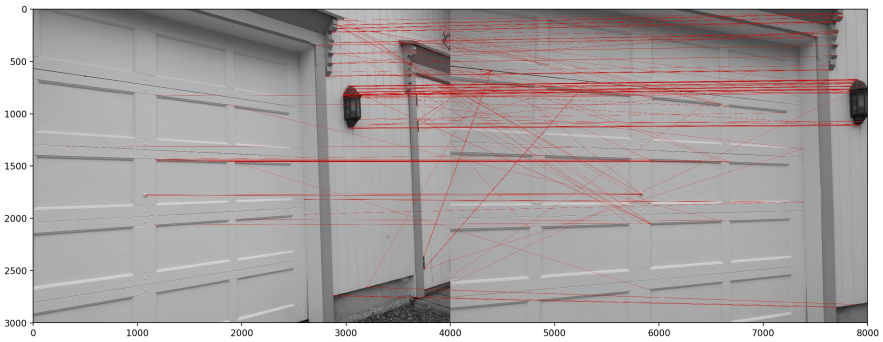
Figur 6.2: Detekterte nøkkelpunkter på lampehuset.

BFMatcher og FlannBasedMatcher

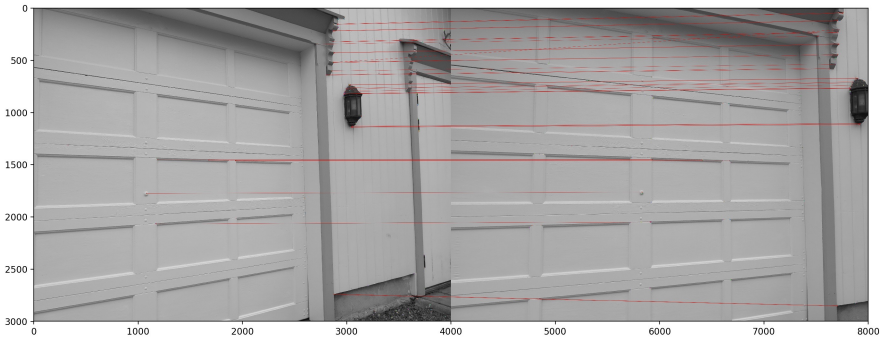
Dette er funksjoner som matcher deskriptorene i to bilder. BFMatcher benytter en algoritme som tester absolutt alle deskriptorene i hvert bilde mot hverandre. For hver deskriptor blir avstanden til alle de andre deskriptorene beregnet. Den korteste avstanden blir definert som match. Merk at det ikke er snakk om geometrisk avstand, men om avstanden i det 128 dimensjonale rommet til deskriptorvektorene som ble beskrevet avsnitt 5.2.2. Altså differansen mellom deskriptorvektorene. Dette er en krevende prosess, og vil som regel ikke være nødvendig med mindre endringen mellom to bilder er veldig stor. Ulempen er at deskriptorer som er like risikerer å bli matchet med hverandre selv om de ikke tilhører samme nøkkelpunkt.

FlannBasedMatcher er derimot basert på nærmeste-nabo prinsippet, så denne vil ikke teste alle punktene mot hverandre. Dette er en mer effektiv metode som i de fleste tilfeller vil gi gode nok resultater. Dessverre har det vært pro-

blemer med å få denne funksjonen til å fungere skikkelig i kombinasjon med siste versjon av OpenCV og Python[6].
Figur 6.3 viser to bilder som er matchet med hverandre med BFMatcher. De røde strekene symboliserer matchede punkter, og det er tydelig at enkelte av disse er feilaktige. De feilaktige matchene kan enkelt sorteres bort ved å for eksempel sette et krav om maksimum geometrisk avstand mellom punktene.



(a) Objektmatching - alle potensielle matcher.



(b) Objektmatching - 50 beste matcher (basert på plassering).

Figur 6.3: Matching med BFMatcher.

findEssentialMat og recoverPose

Den relative orienteringen i OpenCV gjøres med to funksjoner. Den første, findEssentialMat, er basert på fempunkts algoritmen som ble beskrevet i delkapittel 5.3, og krever dermed et kalibrert kamera. RANSAC brukes til å plukke ut de matchene som gir best resultat (se avsnitt 5.3.2) for orienteringen. Resultatet fra funksjonen er en essensiell matrise. Fra den essensielle

[6]https://github.com/opencv/opencv/issues/5667

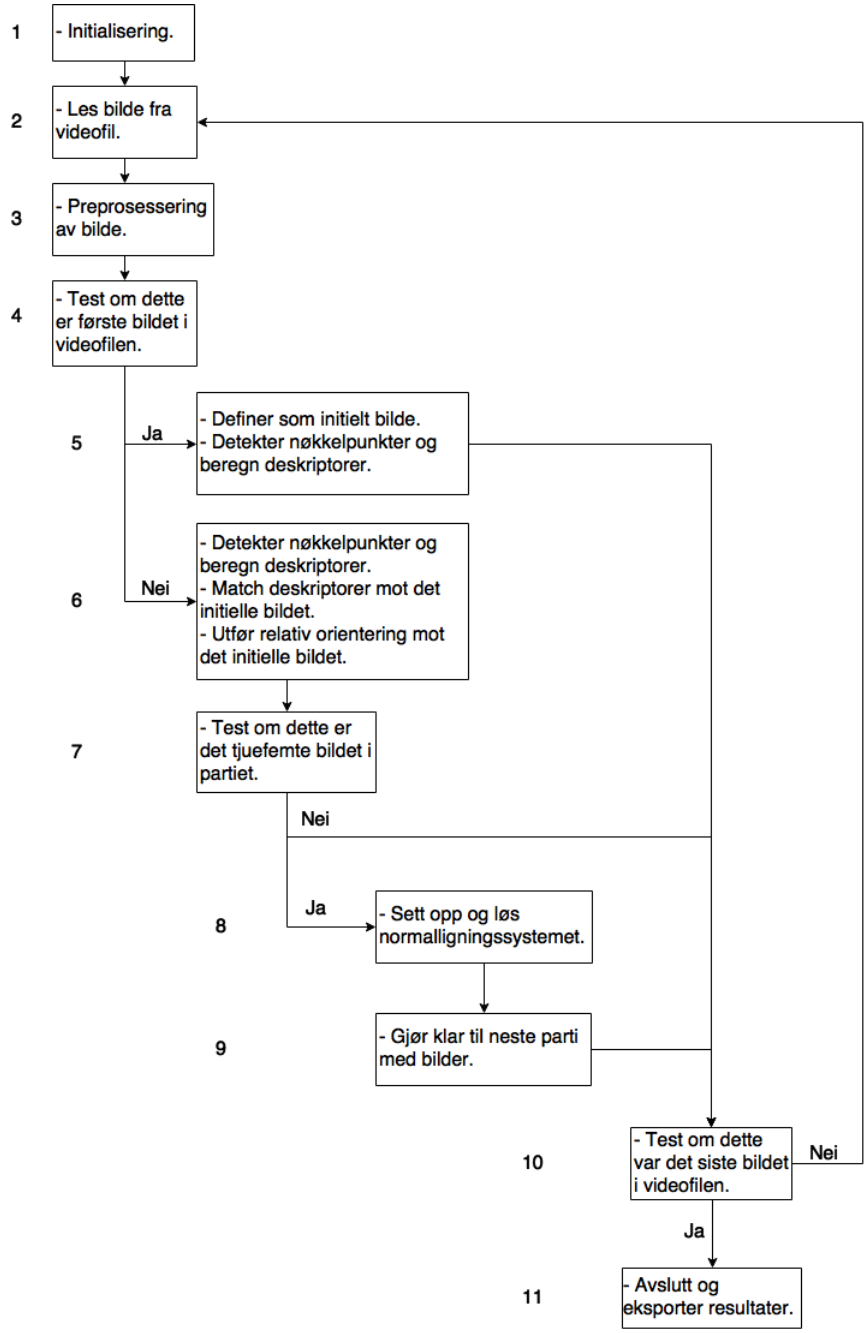
matrisen kan funksjonen recoverPose ta ut rotasjonsmatrisen og translasjonsvektoren mellom de to bildene.

6.2.3 Alternative funksjoner

Et alternativ til deskriptormatching er **KLTracker**. Dette er en funksjon laget spesifikt til å tracke allerede detekterte nøkkelpunkter. I motsetning til deskriptormatching trengs det ikke detektere nøkkelpunkter i hvert bilde, og den er dermed noe mer effektiv. Funksjonen vil tracke nøkkelpunktene så lenge de er synlige, og det vil kun detekteres nye nøkkelpunkter dersom det er for få igjen. Resultatet er også relativ orientering mellom to og to bilder slik at resten av metoden er lik den som benyttes i denne oppgaven.

6.3 Beskrivelse av metoden

Som i Del 1 er også dette problemet (forsøkt) løst med programmering i Python. Dette delkapittelet vil dermed gi en beskrivelse av programmeringsgangen. Programmet baseres på funksjonene som ble beskrevet i avsnitt 6.2 og teorien i kapittel 5. Med utgangspunkt i disse funksjonene fra OpenCV er det laget et script som leser bilde for bilde fra en videofil. Hvert bilde som leses blir matchet mot det første bildet, og deretter relativt orientert. For hvert tjuefemte bilde (hvert sekund) blir alle observasjonene og de beregnede rotasjonene tatt inn og løst i et normalligningssystem. Det tjuefemte bildet blir så satt som initielt bilde for de neste tjuefem bildene som leses inn. På den måten blir alle partier med bilder og observasjoner koblet sammen, og resultatet blir en sammenhengende *trajectory*. Figur 6.4 gir en visuell oversikt over dataflyten i programmet, og avsnitt 6.3.1 gir en punktvis gjennomgang av alle programmets deler.



Figur 6.4: Dataflyt visuell odometri.

6.3.1 Punktvis beskrivelse av programmet

Initialisering.
I denne første funksjonen defineres følgende parametere:

- Kameraets indre orienteringselementer. Det forutsetter at disse er estimert fra produsenten eller med kamerakalibrering.
- Initiell rotasjonsmatrise og translasjonsvektor. Rotasjonsmatrisen settes lik en identitetsmatrise, og translasjonsvektoren settes lik null.
- Videofilen som skal leses inn.
- Ny bildehøyde som senere benyttes til å kutte vekk nederste del av bildene som leses inn. Dette er nødvendig dersom bilens dashbord og panser er synlig nederst i bildet.
- Antall nøkkelpunkter og deskriptorer som ORB-algoritmen detekter. Som standard settes denne til fem hundre, men i dette tilfelle blir det mye å ha med i normalligningssystemet. Samtidig er det viktig å ha med nok punkter til å kunne matche bildet med tjuefire andre bilder og bestemme den relative orienteringen til hver av disse. Etter noe testing ble funksjonen satt til å detektere hundre nøkkelpunkter. Det ga en god balanse mellom prosesseringstid og kvalitet.

2: Les bilde fra videofil.
Dette er den ytre løkken i programkoden, og denne leser inn bilde for bilde med funksjonen VideoCapture.

3: Preprosessering av bilde.
Når bildet er lest inn blir den nye bildehøyden fra initialiseringen lagt til. Som nevnt er dette for å kutte vekk bilens panser og dashbord. Tidspunkt og bildenummer blir også definert her.

4: Test om dette er første bilde i videofilen.
Hvis dette er det aller første bildet i videofilen må det detekteres nøkkelpunkter og beregnes deskriptorer. Alle nøkkelpunktene som blir detektert her blir tatt med i løsningen av normalligningssystemet. Den initielle rotasjonsmatrisen og translasjonsvektoren blir også tatt inn her og definert som orienteringen til det første bildet.

5: Dersom dette er det første bildet.

- **Definer som initielt bilde.** Med initielt bilde menes det bilde som alle påfølgende bilder skal testes mot.
- **Detekter nøkkelpunkter og beregn deskriptorer.** Med ORB-algoritmen blir det detektert og beregnet hundre nøkkelpunkter og deskriptorer. Alle disse blir lagret.

6: Dersom dette ikke er det første bildet.

- **Detekter nøkkelpunkter og beregn deskriptorer.** Med ORB-algoritmen blir det detektert og beregnet hundre nøkkelpunkter og deskriptorer.
- **Match deskriptorer mot det initielle bildet.** Deskriptorene blir matchet med deskriptorene fra det initielle bildet. Matchene blir sortet etter avstand (altså punktets koordinatforskjell i de to bildene). De ti matchene med kortest geometrisk avstand blir luket ut, da det er sannsynlig at disse ligger svært langt unna kameraet. Videre lagres de førti beste matchene for å luke ut matcher som åpenbart er feil.
- **Utfør relativ orientering mot det initielle bildet.** De abeste matchene blir brukt til å beregne den relative orienteringen mellom bildene. Til dette brukes funksjonene findEssentialMat og recoverPose. Disse baseres på fempunkts-algoritmen. For å teste mot uteliggere benyttes RANSAC (avsnitt 5.3.2).

7: Test om dette er det tjuefemte bildet i partiet.
Hvis dette er det tjuefemte bildet i partiet løses normalligningssystemet. Så blir funksjonene forberedt på et nytt parti med bilder.

8*: Sett opp og løs normalligningssystemet.
Dette punktet er dessverre ikke fullført. Se delkapittel 5.4 for teori, og avsnitt 6.3.2 for hvordan det kunne blitt gjort.

9: Gjør klar til neste parti med bilder.
Det siste bildet i partiet blir satt som initielt bilde for neste parti med bilder. Det vil si at alle de hundre nøkkelpunktene som ble detektert i det siste bildet tas med til det neste partiet.

10: Test om dette var det siste bildet i videofilen.
Hvis dette var det siste bildet i videofilen avbrytes den ytre løkken fra og prøve å lese inn neste bilde. Dette er for at programmet ikke skal prøve å lese inn bilder som ikke finnes, men heller gå inn i den avsluttende fasen.

11*: Avslutt og eksporter resultater.
Ettersom løsningen av normalligningssystemet ikke ble fullført har det heller ikke blitt noen resultater for denne delen av oppgaven. Se avsnitt 6.3.2 for hvordan resultatene kunne blitt eksportert, og hvordan det hadde passet inn i et kalmanfilter.

6.3.2 Gjenstående arbeid

På grunn av tidsmangel ble ikke normalligningssystemet fullført, og det er dermed ingen resultater fra denne delen av oppgaven. Her gis en beskrivelse av det gjenstående arbeidet.

Løsning av normalligningssytemet
Som beskrevet tidligere blir den relative orienteringen til hvert bilde beregnet relativt det første bildet i partiet. Hvert parti består av tjuefem bilder. Alle detekterte nøkkelpunkter i alle tjuefem bilder blir transformert til det første bildets referansesystem. Nøkkelpunktene kan sees på som observasjoner i form av bildemålinger. Alle bildemålingene og alle projeksjonssenterene tas med i en utjevning for å forbedre de foreløpige verdiene. Det er kun posisjonen til de endelige projeksjonssenterene og rotasjonene som behøver å eksporteres.

Kobling mot navigasjonssystemet
For at de beregnede rotasjonene og translasjonene skal kunne gi et fornuftig bidrag til navigasjonssystemet må målestokken bestemmes. Kombinasjon med et hjulodometer vil kunne gi målestokken direkte. Dersom odometermålinger ikke er tilgjengelig (for eksempel på jernbane) må GNSS/INS benyttes til å bestemme den kjørte avstanden. Siden samtlige bilder er koblet sammen gjennom utjevningen trenger ikke målestokken være kjent i mer enn et par bilder. Det betyr at systemet vil klare seg med sporadisk god GNSS-dekning.

Utjevningen gjøres for hvert tjuefemte bilde, det vil si hvert sekund. Det betyr at de beregnede rotasjonene og translasjonene kan tas inn i kalmanfilteret som støttemålinger hvert sekund. Hvor godt dette vil fungere i praksis er avhengig av kvaliteten på bildenes tidsstempling. Målestokken kan også legges inn som en omtrentlig betingelse i normalligningssystemet (avsnitt 5.4.3) og innføres som en ekstra ukjent i Kalmanfilteret.

Delayed State

Det er verdt å bemerke at målingene fra visuell odometri vil være inkrementelle. Det betyr at hver måling er avhengig av tilstanden ved tidspunkt k og k-1. Dette kan skrives slik:

$$z = Hx_k + Jx_{k-1} + e \tag{6.1}$$

Dette er kjent som *delayed state*. Den optimale løsningen for dette problemet er presentert i [Brown and Hwang, 1997, s. 350 - 353]:

Tilstandsoppdatering:

$$\hat{x}_k = \tilde{x}_k + K_z(z_k - \tilde{z}_k) \tag{6.2}$$

der

$$\tilde{z_k} = H_k\tilde{x}_k + J_k\hat{x}_{k-1} \tag{6.3}$$

Kalman Gain:

$$K = [\tilde{P}_k H_k^T + \Phi_{k-1}P_{k-1}J_k^T][H_k\tilde{P}_k h_k^T + R + J_k P_{k-1}\Phi_{k-1}^T H_k^T + H_k\Phi_{k-1}^T P_{k-1}J_k^T + J_k P_{k-1}J_k^T]^{-1} \tag{6.4}$$

Kovariansoppdatering:

$$P_k = \tilde{P}_k - K_k L_k K_k^T \tag{6.5}$$

der

$$L_k = H_k\tilde{P}_k H_k^T + R_k + J_k P_{k-1}\Phi_{k-1}^T H_k^T + H_k\Phi_{k-1}P_{k-1}J_k^T + J_k P_{k-1}J_k^T \tag{6.6}$$

Formen på disse ligningene er den samme som for de vanlige Kalmanfilterligningene. Den eneste forskjellen er noen ekstra ledd i uttrykket for gainen og kovariansoppdateringen. Merk at det ikke er noen kjent løsning for avhengighet av tilstanden ved tidspunkt k-2 eller lenger bak.

6.4 Oppsummering Del 2

I denne delen har visuell odometri blitt testet for å se om det kan brukes som støtte til treghetsnavigasjonssytemet. Dette gjøres ved å detektere nøkkelpunkter og så enten tracke eller matche disse gjennom flere bilder. På den måten kan den relative orienteringen mellom hvert bilde beregnes. I teorien skulle dette gitt forholdsvis gode resultater, og dermed fungere godt som en støtte. Dessverre ble det ikke tid til å fullføre normalligningssystemet, så det er ingen resultater som viser hvor godt dette fungerer i praksis. Derimot har tidligere forskning vist til gode resultater (se avsnitt 5.1.1).

Kapittel 7

Konklusjon for Del 1 og Del 2

7.1 Konklusjon

I denne oppgaven har det blitt testet om et kamera kan fungere som en støttesensor i et treghetsnavigasjonssystem. Dette har blitt gjort i to deler: nullhastighetsoppdatering og visuell odometri. For begge problemstillingene var formålet å teste ut om dette faktisk fungerer. Det er dermed ikke lagt mye fokus i å teste ulike datasett for å få flere resultater. For å detektere nullhastighet ble problemet løst på en enkel måte ved å anta at liten endring mellom flere bilder på rad indikerte at systemet var i ro. Dette viste seg å stemme, men med noe usikkerhet knyttet til tidsstemplingen av bildene. Den samme usikkerheten ville vært gjeldende også for den visuelle odometrien. For å kunne bruke bildenes relative orientering til å estimere kameraets relative posisjon må usikkerheten til tidsstemplingen være på samme nivå som i systemet ellers. Om dette er mulig å få til med et actionkamera av typen som er benyttet i denne oppgaven er uvisst. For å finne ut av dette måtte kameraet blitt testet i langt større grad enn det har blitt tid til i denne oppgaven. Det mest optimale hadde vært om bildeeksponeringen ble styrt av navigasjonssystemet. Dessverre ble ikke delen om visuell odometri fullført. Andre artikler viser riktignok til at det er mulig å oppnå gode resultater med visuell odometri.

Konklusjonen blir dermed at et kamera kan benyttes som en støttesensor til et navigasjonssystem, men det må en del testing til for å oppnå gode resultater. Deteksjon av nullhastighet kan løses ved å benytte metoden i denne oppgaven, eller ved å benytte visuell odometri. Sistnevnte er naturligvis det mest interessante, siden det også gir mulighet til å estimere posisjonsendring.

Bibliografi

Andersen, O. (1994). *Bilde-Tall-Terreng.*

Andersen, O. (2003). *Orientering i stereoinstrument.*

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359.

Brown, R. and Hwang, P. (1997). *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions.* Number v. 1 in Introduction to Random Signals and Applied Kalman Filtering: With MATLAB Exercises and Solutions. Wiley.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg. Springer-Verlag.

El-Sheimy, N. (1996). *The Development of VISAT - A Mobile Survey System for GIS Applications.* PhD thesis, The University of Calgary.

Farrell, J. (2008). *Aided Navigation: GPS with High Rate Sensors.* McGraw-Hill professional engineering: Electronic engineering. McGraw-Hill Education.

Hofmann-Wellenhof, B., Lichtenegger, H., and Wasle, E. (2007). *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more.* Springer Vienna.

Jekeli, C. (2001). *Inertial Navigation Systems with Geodetic Applications.*

Walter de Gruyter.

Karami, E., Prasad, S., and Shehata, M. (2015). Image matching using sift, surf, brief and orb: Performance comparison for distorted images. In *Conference: Conference: Newfoundland Electrical and Computer Engineering Conference, IEEE, Newfoundland and Labrador Section, At St. John's, NL*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110.

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee.

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, pages 430–443, Berlin, Heidelberg. Springer-Verlag.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA. IEEE Computer Society.

Scaramuzza, D. and Fraundorfer, F. (2012). Visual odometry, part ii: Matching, robustness, optimization, and applications. *IEEE Robotics and Automation Magazine*.

Tegedor, J. (2015). *Multi-Constellation Satellite Navigation: Precise Orbit Determination and Point Positioning*. PhD thesis, Norwegian University of Life Science, Ås (NMBU).

Titterton, D., Weston, J., and of Electrical Engineers, I. (2004). *Strapdown Inertial Navigation Technology*. Electromagnetics and Radar Series. Institution of Engineering and Technology.

Woodman, O. J. (2007). An introduction to inertial navigation. Technical

Report 696, University of Cambridge - Computer Laboratory.

Zuliani, M. (2009). Ransac for dummies. *With examples using the RANSAC toolbox for Matlab and more.*

Norges miljø- og biovitenskapelig universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway