Innhold

1 Introduksjon **1**

2 Grunnlag **3**
2.1 Optical Flow 8
2.1.1 Antagelser 8
2.1.2 Mulige metoder 9
2.1.3 Horn & Schunk's metode 12
2.2 Contour Tracking with Splines 15
2.2.1 Splines 17
2.2.2 Shape-space 18
2.2.3 Image features 19
2.2.4 Kurvetilpassning 21
2.2.5 Statistiske modeller 25
2.2.6 Dynamiske modeller for bevegelse 27
2.2.7 Temporær filtrering med Kalman-filter 28
2.2.8 Condensation 29
2.2.9 ICondensation 31
2.3 Dynamic Time Warping 34

3 Applikasjon / System **39**
3.1 Identifisering og tolkning av bevegelse 40
3.2 Identifisering av objekter og objektbevegelse 47
3.3 Tolkning av objektbevegelse 48

4 Eksperimentelle resultater **51**
4.1 Test av optisk flyt 51
4.2 Test av konturspor ing 55
4.3 Test av bevegelsestolkning 58

5 Konklusjon **69**

Figur 1: Eksempel på UAV: Rotormotion SR1B 800. (rotomotion.com)

1 Introduksjon

Et autonomt helikopter er et ubemannet helikopter som kan fly og utføre oppdrag på egen hånd uten hjelp fra mennesker. Utviklingen av såkalte *Unmanned Air Vehicles* (UAV) har foregått over lang tid, og i de siste 10-årene også autonome typer. For noen eksempler på slike prosjekt se [1], [2], [3] og [4]. På grunn av den stadige teknologiske utviklingen kan slike helikoptre realiseres enklere og billigere for hvert år som går. I dag kan man bygge forholdsvis kraftige systemer på små plattformer, noe som passer fjernstyrte helikoptre bra.

Grunnene til å utvikle slike helikoptre er mange; ettersøk, kartlegging, transport, overvåkning og sporing. Dette dokumentet tar for seg ett bestemt scenarie: Anta et svært trafikkert lyskryss eller en rundkjøring. Hver dag passerer det tusenvis av kjøretøy og noen av disse bryter sannsynligvis trafikkreglene, noe som kan føre til mer eller mindre alvorlige situasjoner. Hvordan kan man overvåke et slikt trafikkpunkt billig og enkelt? Et autonomt helikopter med nødvendig utrusting vil kunne overvåke slike punkt forholdsvis enkelt og det vil i tillegg kunne ta opp jakten på farlige og uansvarlige trafikanter. Det vil si; en UAV kan være stasjonert ved et knutepunkt og overvåke situasjonen kontinuerlig. Med en gang det detekteres en alvorlig hendelse kan dette rapporteres til ansvarlig myndighet og om nødvendig kan mistenkte spores til situasjonen er tatt hånd om. Helikopteret trenger nødvendigvis ikke å være stasjonert på ett bestemt sted hele tiden, men kan isteden fly rundt til forskjellige interessante lokasjoner og ta stikkprøver. Forskjellige lokasjoner kan være interessante på forskjellige tider av døgnet. (Om vi ønsker at slike helikoptre skal overvåke oss er et godt spørsmål, men jeg er bare interessert i bildebehandlingsaspektet.)

Hvordan kan den nødvendige informasjonen fra en videosekvens utvinnes og hvordan kan denne informasjonen brukes til å lære og tolke bevegelsesinformasjon? Bevegelsesinformasjon kan utvinnes på forskjellige måter fra en videosekvens. Lokale metoder vil i denne sammenheng si at alle pix-

ler i et bilde benyttes og at hvert piksel behandles uavhengig av hverandre. Globale[1] metoder vil si metoder som behandler pixler i grupper (nabopixler, klikk) slik at det blir en helhetlig tolkning av bildet. Lokale metoder er ofte raske og robuste, men gir ikke alltid et godt grunnlag for komplekse tokninger av bildet, som for eksempel deformasjon. Globale metoder er gir nettopp dette, men er ofte mer prosessorkrevende og utsatt for støy. (Se [5] for en diskusjon angående dette.) Eksempler på lokale metoder er *background substraction* og *piksel segmentering*. Eksempel på en global metode er *korrelasjon*.

Optisk flyt vil si hvordan lysintensitet forflytter seg i en videosekvens og er ganske interessant for oppgaver av denne typen. Det finnes flere forskjellige måter å kalkulere optisk flyt; *differensiale teknikker*, *korrelasjonsteknikker*, *energibaserte teknikker* og *fasebaserte teknikker* [6]. De forskjellige metodene har ulike fordeler og ulemper. Optisk flyt representeres som vektorkart der hver vektor forteller retning og hastighet for ett bestemt piksel. Det vil si at et bilde på 100×100 piksler vi være representert som et vektorkart på 100×100 vektorer. Når man har bestemt den optiske flyten vil man vite alle bevegelsene som finnes nøyaktig. Kvaliteten på vektorkartet måles som oftest med *average angular error* som vil si den gjennomsnittlige feilen vektorene har målt i grader.

Et vektorkart er bare en representasjon av bevegelse i rå form. For å forstå hva som egentlig foregår i en sekvens av bilder må man sette alle bevegelsesvektorene i sammenheng. Det vil for eksempel si å gruppere / segmentere (*cluster*) vektorer som har en spatial sammenheng. Når man har gjort dette kan man tolke hver gruppering som objekter og følge deres utvikling over tid. Problemet med denne framgangsmåten er at den er forholdsvis ressurskrevende. Den motsatte muligheten er å identifisere objektene man ønsker følge først og så beregne bevegelsene. (Dvs; generell bevegelse $\rightarrow$ clustering $\rightarrow$ objekt identifikasjon :vs: objekt identifikasjon $\rightarrow$ beregning av bevegelse.) Utfordringen med denne framgangsmåten ligger i selve identifiseringen av objektene. En mulighet er *contour tracking with splines*, som vil si å bruke forhåndsdefinerte splines til å identifisere objekter ut i fra konturene deres. Fordelen med å bruke splines, i motsetning til å bruke maler som bitmaps, er at de krever lite lagringsplass og er svært enkle å manipulere / deformere. Dette er viktig med tanke på robusthet. Splines gir også gode muligheter for å spare ressurser, noe som vi skal se senere.

Når man har identifisert en bevegelse, og eventuelt hva som beveger seg, kan man også bestemme bevegelsesnaturen til objektet. Det vil for eksempel si å identifisere om en bil tar til venstre eller høyre i ett kryss, eller om en bil kjører noen ekstra runder i en rundkjøring. Dette er infor-

[1] På engelsk blir gjerne *low-level* og *high-level* brukt isteden for *local* og *global*.

masjon som må læres og tolkes. En interessant framgangsmåte er å bruke *dynamic time warping* (DTW). DTW er kanskje først og fremst kjent innen *talegjenkjenning*, men kan egentlig brukes til å løse mange forskjellige typer problemer. Se [7], [8], og [9] for noen eksempler. I denne sammenheng vil det enkelt sagt si; representere bevegelser over tid som grafer, bygge opp en database over alle lovlige grafer, og så gjøre søk i denne databasen hver gang man vil identifisere en bevegelse. DTW greier dette selv om det er mye støy på dataene og grafene er strukket forskjellig ut i tid i forhold til hverandre.

Kapittel to tar for seg flere mulige metoder som kan brukes til å løse oppgaven. Kapittel tre presenterer den komplette applikasjonen i detalj. Kapittel 4 inneholder testresultatene.

2 Grunnlag

Den mest vanlige og enkleste metode for å registrere bevegelse i video er *background substraction* (BS). Det er blitt gjort svært mye forskning på metoder som baseres på BS. Se [10], [11], [12] og [13] for noen interessante eksempler på dette. Det er derfor naturlig å ta utgangspunkt i BS for å løse denne oppgaven.

La oss anta at en rundkjøring skal overvåkes av et autonomt helikopter. Helikopteret er enten midlertidig stasjonert (på et oversiktlig fast punkt) eller svever stille i lufta for en viss periode. Et videokamera montert på helikopteret gir kontinuerlig videodata.

Nivå 1: Detekter bevegelse.
Trekk siste bilde (fra videosekvensen) fra et referansebilde (pikselvis subtraksjon). (Referansebildet kan være et bilde av rundkjøringen uten trafikk eller forrige bilde i bildesekvensen.) Dersom det ikke er noe trafikk i øyeblikket vil man få et helt sort bilde med alle verdier lik null, eller nær null. Med en gang en bil kommer inn i synsfeltet oppdages dette ved at noen av verdiene blir enten svært negative eller positive avhenging om bilen er lysere eller mørkere enn bakgrunnen. Man kan altså bare bestemme en grenseverdi for maks avvik fra null for å oppdage bevegelse. Med denne metoden kan man produsere svart-hvitt bilder der hvitt indikerer bevegelse. (Allerede på dette nivået har man oppnådd noe nyttig. Dette er alt man trenger for å lage en optisk sensor ala det som finnes i enkle alarmsystemer.)

Nivå 2: Identifiser objektene som beveger seg.
Man har allerede svart-hvitt bilder der hvitt indikerer bevegelse. Nå tren-

ger man bare å segmentere de hvite områdene fra hverandre. En mulig framgangsmåte er å bruke *flood-fill* på hvite områder med forskjellige identifikatorer. Områder med forskjellige identifikatorer representerer forskjellige objekter. Så kan objektene matches i suksessive bilder basert på enten koordinater, form eller størrelse. Man kan for eksempel anta at det nærmeste objektet i neste bilde er det samme som det forrige. Mer interessant er det å identifisere hva objektene er ved hjelp av størrelse og form. Da kan man bestemme om det er en bil eller motorsykkel som beveger seg. (På dette nivået kan man bestemme hvor mange objekter som beveger seg i bildet og i hvilken retning de beveger seg, eventuelt hva objektene er.)

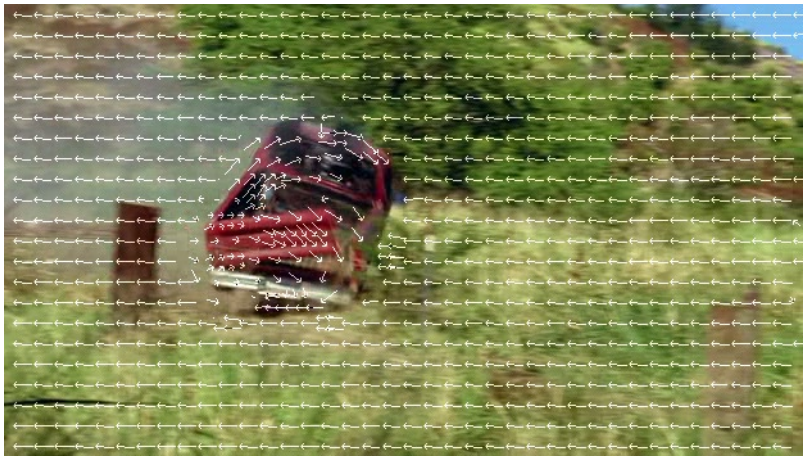
Nivå 3: Forstå bevegelsesmønstre.
På dette nivået er man ferdig med bildebehandling. Nå jobber man med rekker av koordinater. Som nevnt er det mulig og bruke DTW til dette. Enkelt sagt: produser en bevegelsesgraf og match denne opp mot forhåndslærte grafer. Det er ønskelig at grafen er uavhengig posisjon for at det skal være enklest mulig å matche. En mulighet er å plotte grafer der x-aksen er tid og y-aksen er retning (i grader eller medianer). Det er ønskelig å ta hensyn til hastighet også, slik at man kan identifisere fartsovertredelser, men dette er mulig å detektere direkte ut i fra fartsvektorene. (Nå har man faktisk utrustning for å fange opp og skille fra hverandre biler som f.eks. kjører 3 ganger rundt i en rundkjøring vs. en som tar 4 runder, eventuelt en som kjører i feil retning.)

Problemer

- Dersom kameraet ikke står helt stille virker ikke BS. Referansebildet blir ubrukelig i forhold til den nye posisjonen/retningen til kameraet. Dette kan kanskje løses med *phase correlation*[2], men dette krever en del ressurser på grunn av fouriertransformasjonene og er begrenset til forholdsvis enkel bevegelse (translasjon). Ved sporing/forfølging vil det sannsynligvis være umulig, eller svært vanskelig, å bruke BS. Det finnes imidlertid eksempler på framgangsmåter som bruker BS under svært vanskelige forhold. Ett av dem er [14].

- Dersom lysforholdene forandrer seg, som de gjerne gjør, blir referansebildet ubrukelig (igjen). Dette kan faktisk håndteres ved bruke dynamiske metoder for å oppdatere referansebildet. Isteden for å bruke et konstant referansebilde kan man f.eks. bruker bilder der hvert piksel er representert som normalfordelte modeller av lysintensiteten. Se [15] for ett godt eksempel.

[2]Se http://en.wikipedia.org/wiki/Phase correlation



Figur 2: Optisk flyt som *motion block vectors* i en MPEG-4 video.

- Dersom det begynner å snø (...) har man selvfølgelig et problem. Dette gjelder for alle former av alvorlig støy. Støy som snø vil skape et inntrykk av veldig mye bevegelse og det er svært vanskelig å skille denne bevegelsen fra all annen bevegelsene. Alle metoder som baseres på å finne bevegelse før identifiseringen av objektene vil ha dette problemet.

- Kameraposisjonen er også svært avgjørende om man benytter BS. Metoden fungerer ikke så godt dersom objektene overlapper hverandre. Så man er avhengig av at kameraposisjonen er forholdsvis høyt oppe og at retningen er bratt slik at objektene kan oppfattes med god klarering fra hverandre.

Alternativ 1: Optisk flyt
Optisk flyt (OF) vil som sagt si å beregne bevegelsesvektorer til alle pixler i ett bilde. Forskjellige metoder for dette diskuteres senere. Når et slikt vektorkart er beregnet har man egentlig alt man trenger for å løse nivå 1 og 2 samtidig som man løser noen av problemene forbundet med BS:

- Kamerabevegelse vil ikke være noe problem for selve algoritmen som kalkulerer optisk flyt. Den oppdages som all annen bevegelse. Ved å se på fordelingen av bevegelsesvektorene kan man for det første bestemme *om* kameraet beveger seg, for det andre bestemme *hvordan* kameraet beveger seg, og for det tredje *filtrere vekk* denne bevegelsen slik at den egentlige bevegelsen på bakken kan identifiseres. Dette vises senere.

- Lysforholdene spiller ingen rolle siden OF baseres på suksessive bilder og ikke ett statisk referansebilde. Svært raske lysforandringer, som f.eks flimring, vil selvfølgelig være ett problem. Slik støy vil oppfattes som bevegelse, men kan kanskje filtreres bort pga. den tilfeldige naturen. Dessuten er ikke slik støy så naturlig (med unntak av lynnedslag), det vil i så fall være pga. dårlig utstyr. (Den betydelige støyen fra helikopteret setter selvfølgelig høye krav til utstyret.)

- Værforhold som f.eks. snø er også en utfordring, men sannsynligvis enklere å håndtere enn med bruk av BS. Snødriv vil gi et ganske uniformt bevegelsesmønster og kan derfor filtreres vekk. Men det kan også forveksles med kamerabevegelse.

- Overlapping vil gi samme problem som med BS, men det vil være mulig å skille forskjellige objekter fra hverandre med OF dersom retningsvektorene assosiert med hvert objekt er forskjellige. (Se figur 3 for et eksempel på overlapping.)

- I forhold it BS er OF mye mer krevende og derfor må det forsøkes å minimere ressursforbruket på en eller annen måte. Det enkleste er å gå ned i oppløsning på bildene.

Med OF er det strengt tatt ikke nødvendig å gå til nivå 3 dersom man ikke trenger et så komplekst system. Man kan bygge opp statistiske modeller som bare forteller hvor man aksepterer bevegelse og hvilke retninger som aksepteres. Man tar altså ikke hensyn til *hva* som beveger seg, men sjekker bare *om* bevegelsen er lovlig. Da kan man selvfølgelig ikke oppdage om en bil kjører flere runder i en rundkjøring. For å ta hensyn til at flere retninger kan være lovlige på samme punkt, som vil være tilfellet i lyskryss, kan man bruke *multiple structure tensor fields* [16].

Alternativ 2: Kontursporing med Splines(KS)
Uttrykket 'snakes' blir brukt av Blake og Isard [17] for å forklare *spline tracking*. Ideen går ut på å lage såkalte *feature maps* (se figur 3), ved hjelp av konvolusjon, og se på de som landskapskart der slanger (definert som splines) kan bevege seg i. Lyse områder (topper) vil tiltrekke seg slangene og ved å bruke forskjellige masker under konvolusjonen kan man få de til å bevege seg over de trekkene man ønsker (se [18] for eksempler på forskjellige masker). På grunn av en slanges særegne anatomi kan den bevege seg over uregelmessige dumper / hull, samtidig som den ikke kan ta alt for skarpe svinger. Poenget er at når man matcher en spline til trekk som finnes i bilder vil de være resistente både mot støy og mot områder der det ikke finnes informasjon (f.eks. der faktiske kanter forsvinner fordi forgrunnsfargen er lik bakgrunnsfargen). Man kan si at splinen blir utsatt

Figur 3: Eksempel på *feature map*. Bilde av tungt trafikkert vei.

for interne krefter, fordi det er splinens natur (siden den er definert med forholdsvis få punkter), og ytre krefter fordi den tiltrekkes de ønskelige trekkene i bildet. Den endelige splinen vil derfor være et resultat av et kompromiss mellom disse kreftene og det er dette som gjør metoden robust.

Når man har 'matchet' en spline til de ønskelige trekkene i et bilde, dvs. identifisert et objekt, kan man matche den samme splinen i neste bilde. Siden bildene blir tatt i sanntid vil sannsynligvis ikke objektet ha rukket å bevege seg i veldig stor grad og man kan derfor søke etter objektet i nærheten av dens gamle posisjon. (Dette betyr også at man kan spare prosessorkraft.) Ved å registrere objektets posisjon over flere bilder kan man bestemme bevegelsen.

Som sagt må splinen være definert på forhånd. Hvis man vil spore en bestemt bil må man definere en spline som passer konturene til bilen. Dette er apriorisk informasjon og er selve grunnen til robustheten. Det betyr at kamerabevegelse, lysforhold, snøvær og overlapping ikke vil være et like stort problem for algoritmen (ift. BS og OF). Matchingsprosessen er ikke avhengig av et referansebilde, men baserer seg bare på informasjonen som finnes i det aktuelle bildet. Posisjonen til objektet som spores er som sagt med på å veilede neste søk, men en kamerabevegelse vil bare føre til en mindre eller større bevegelse av objektet.

Noen poeng:

- Man må definere alle splines på forhånd. Det vil si at man må vite hva man skal spore. Man kan ikke oppdage bevegelse som er basert på ukjente objekter. Så om det skulle komme en bil med en svært uvanlig fasong ville den rett og slett ikke bli registrert. Det kreves altså mer forhåndsarbeid og man vil kanskje aldri kunne garantere at man registrerer all bevegelse.

- Den kunne også være mer ressurskrevende pga. konvolusjonen, men dette løses ved at man innsnevrer søkeområdet. Isteden for å utføre komplette konvolusjoner av bildene kan man faktisk bare gjøre det langs normalene til splinen. Det vil si konvolusjon i bare én dimensjon i stedet for i to dimensjoner. Dette forklares i detalj senere.

- Den kan ha dårligere skaleringsegenskaper avhengig av hvordan man ser på det. OF er avhengig av oppløsningen bildene har. Jo større bilde jo mer krevende å kalkulere. På den andre side vil OF bestemme *all* bevegelse som finnes. KS sporer i utgangspunktet bare ett objekt, så jo flere objekter man vil spore, jo mer krevende er fremgangsmetoden. På den andre side har oppløsningen lite å si for ytelsen (i teorien).

OF og KS er altså to vidt forskjellige måter å løse oppgaven. OF er generell og bruker mye ressurser uansett. KS er spesifikk og kan være ganske billig dersom man ikke skal spore for mange objekter samtidig. KS er nok uansett mer robust mot støy enn OF. Senere skal vi se at det er en blanding av metodene som kanskje vil være det beste.

2.1 Optical Flow

2.1.1 Antagelser

For å kalkulere optisk flyt må man gjøre visse antagelser. De metodene som finnes i dag baseres hovedsaklig på tre forskjellige antagelser:

- Data Conservation: Lysstyrke er konstant over tid, dvs. at et objekt fremstår på samme måte over tid selv om dets posisjon forandrer seg.

- Spatial Coherence: Nabopixler tilhører samme objekt og har derfor lik bevegelse.

- Temporal Continuity: Objekters hastighet og retning er konstant eller forandrer seg gradvis.

I realiteten vil disse antagelsene ikke alltid være riktige, noe som kan føre til feilberegninger. På den andre side kan beregningene også være riktige eller i alle fall brukbare selv om ikke antagelsene stemmer.

Data Conservation For at man skal kunne oppdage bevegelse i det hele tatt må det være forandringer i lysintensitet. Man kan for eksempel ikke oppdage gjennomsiktige objekter, man kan heller ikke oppdage spesielle bevegelser dersom objektene ikke har noen tekstur. Det er for eksempel ikke lett å se om kuler med blank overflate står stille eller roter rundt sin egen akse. For at objekter skal oppdages må de ha en tekstur på overflaten

eller så må de bevege seg i forhold til bakgrunnen. Antagelsen er at teksturer til objekter er konstant eller forandrer seg gradvis / sent. På denne måten kan man *følge* lysintensitet som beveger seg over et bilde. Dvs. se hvordan hvert piksel "forflytter" seg. Men forandringer i lysintensitet kan også være resultat av andre ting enn bevegelse. Hvis sola forsvinner bak en sky, for eksempel, forandrer selvfølgelig lysintensiteten seg og da vil det oppfattes som bevegelse. Ett annet problem er sensorstøy på kameraet, men dette kan ofte filtreres vekk.

Spatial Coherence Siden man alltid vil spore flatelapper er det naturlig å anta at nabopixler oppfører seg likt. Slik er det imidlertid ikke i overgangen mellom forskjellige flatelapper og bakgrunn. Når en flatelapp translerer over bildet vil det "forsvinne" pixler under flatelappen og "dukke opp" pixler bak den. Det vil også være sterke motsetninger mellom pixlene på hver side av objektet (shear). Den vanlige måten å håndtere dette problemet er å glatte ut bildet slik at skarpe kanter forsvinner (blur). Dette fører selvfølgelig til at resultatet ikke blir helt nøyaktig og at noe viktig informasjon forsvinner.

Temporal Continuity Det er også naturlig å anta at objekter beveger seg med konstant hastighet eller i det minste akselererer gradvis. På et lavt nivå vil dette si at fartsvektoren til et punkt i bildet ikke vil forandre seg dramatisk på kort tid. Det betyr at man kan tolke bevegelse over lengre tid enn bare to påfølgende bilder. Ved å ta hensyn til dette kan man utvikle metoder som både er mer robuste mot støy og mindre krevende. Problemet er igjen at virkeligheten ikke er så enkel. Kamerabevegelse kan være spesielt problematisk. Ved vibrasjon, for eksempel, vil bevegelsene i bildet være svært urolige og uforutsigbare. Da er det vanskelig å benytte seg av temporær kontinuitet.

2.1.2 Mulige metoder

Differential Techniques Differensiale teknikker forsøker å relatere lokale forandringer i lysintensitet, uttrykt som spatiale og temporære derivater av lystintensitetsfunksjonen til bildedata, til optisk flyt [19]. Den grunnleggende antagelsen er altså Data Conservation. Ut i fra denne kan man utlede en lineær funksjon som beskriver hastighetsvektorene:

$$\frac{\delta I}{\delta x}u_x + \frac{\delta I}{\delta y}u_y + \frac{\delta I}{\delta t} = 0$$

Her relateres de spatiale ($\frac{\delta I}{\delta x}$ og $\frac{\delta I}{\delta y}$) og temporære ($\frac{\delta I}{\delta t}$) derivatene av lysintensiteten for hvert punkt i bildet til den optiske flyten (u_x, u_y). Fordi

det bare er én funksjon med to ukjente kan man ikke bestemme den optiske flyten uten å gjøre ytterligere antagelser. Dette blir kalt for *the aperture problem* og betyr at problemet er *ill posed*.

For å gjøre problemet *well-posed* gjør man en antagelse til. Uansett hva som velges sitter man som oftest igjen med et sett av lineære funksjoner som kan brukes til å finne den optiske flyten for hvert punkt:

$$Av = d$$

der $v = (u_x, u_y)$, A er en $p \times 2$ matrise som uttrykker de spatiale derivatene, og d uttrykker de temporære derivatene som en vektor lik p. Graden av A må være større enn to for at problemet skal kunne løses.

Region-Based Matching En annen vanlig måte å beregne optisk flyt er Region-Based Matching, også kalt korrelasjonsteknikk. Denne baseres også på Data Conservation, som Differential Techniques, men bruker en sammenligningsstrategi isteden for en lokal derivatstrategi. Gitt en region av et bilde, forsøk å finne denne regionen igjen i neste bilde. Dvs. bestem forskyvningen (u, v) (hastighetsvektoren) ved å minimalisere:

$$E_D(u, v) = \sum_{(x,y)\in R} [I(x, y, t) - I(x + u\delta t, y + v\delta t, t + \delta t)]^2 \tag{2.1}$$

Man tar altså en region / vindu fra et bilde og translerer det over neste bilde samtidig som man kalkulerer en verdi ($E_D(u, v)$) som forteller hvor god match regionen gir på forskjellige steder. Ligning 2.1 er den mest kjente metoden, *Sum-of-Squared-Differences* (SSD). (Det finnes utallige varianter av denne som alle forsøker å forbedre resultatet. Noen eksempler er LMeds, WLS, WTLS og LMSOD. [19])

Siden man må utføre søk er regionbaserte metoder mer ressurskrevende enn differensiale metoder selv om man kan bruke forskjellig heuristikk for å begrense søkeområdet. Region-baserte metoder er mye brukt innen komprimeringsteknikk (video, f.eks. MPEG-4), og blir da som oftest kalt *Motion Block Matching* [20]. Dersom bevegelse i en videosekvens kan uttrykkes med vektorer kan man spare mye plass ved å translere regioner i stedet for å lagre dem for hver eneste tidsenhet. (Man kan faktisk se disse blokkene bevege seg i videoklipp som er hardt komprimert.)

En utfordring med disse metodene ligger i selve defineringen av en region, eller et vindu. Ofte velger man rektangulære regioner av f.eks. 16x16 pixels, eller 32x32 pixels. Hvis regionen er for stor kan det være vanskelig å finne gode matcher i det hele tatt pga. deformasjoner og komplekse / overlappende bevegelser. Hvis regionen er for liten kan det bety for lite informasjon til å bestemme noe fornuftig samtidig som metoden blir mer

krevende. Ellers lider teknikker basert på korrelasjon av samme problemer som differensiale teknikker fordi de baseres på samme antagelser.

Energy-Based Methods Den tredje klassen av metoder for å kalkulere optisk flyt baseres på energieffekt avgitt fra filtre som er følsomme for hastighet. De blir også kalt for frekvensbaserte metoder fordi filtrene som benyttes er definert i Fourier-domenet. Fourier-transformasjonen av en translerende flatelapp er:

$$\hat{I}(k,\omega) = \hat{I}_0(k)\delta(\omega + v^T k)$$

der $\hat{I}(k,\omega)$ er Fourier-transformasjonen av $I(x,0)$, $\delta(k)$ er en Dirac delta funksjon, ω betegner temporær frekvens og $k = (k_x, k_y)$ betegner spatial frekvens. Dette betyr at all energi, større enn null, tilknyttet en translerende 2-D flatelapp ligger i planet som skjærer gjennom origo i frekvensdomenet.

Heeger [12] benytter seg av et sett med tolv Gabor filtre med forskjellige spatiale oppløsninger som trekker ut hastighetsinformasjon fra bildesekvenser. Ved å benytte seg av Gabor filtre, som samtidig finner spatiotemporære og frekvens lokaliseringen, oppnår man en ren band-pass representasjon. *Least square fit* blir så brukt på den resulterende distribusjonen i frekvensdomenet.

Denne framgangsmåten er verken spesielt intuitiv eller enkel å implementere.

Phase-Based Techniques Noen av de mest nøyaktige algoritmene som estimerer optisk flyt er basert på Fleet og Jepson's [21] fasebaserte fremgangsmåte. Fasebaserte metoder benytter seg av sett med *velocity-tuned* filtre for å utvinne lokale frekvensrepresentasjoner fra bildesekvenser slik som energibaserte metoder også gjør. Flyten blir estimert ved hjelp av gradientsøk i fasedomenet til de bestemte signaturene. Det er altså en differensial teknikk brukt på fase istedenfor intensitet.

Grunnlaget for denne framgangsmåten er basert på oppfatningen om at utviklingen til fasekonturer gir et godt grunnlag til estimering av bevegelse. Produktet av *band-pass* filtre er som oftest mer stabilt enn amplituden dersom deteksjon av sen bevegelse er viktig. Siden optisk flyt er lokalbasert er den ofte kjennetegnet med små forskyvninger. Derfor kan det være en fordel å utlede hastighet fra fase i stedet for størrelsesorden.

Oppsummering Estimering av optisk flyt er svært interessant og viktig, men de fleste metodene som finnes er forholdsvis begrensede. I sammenligninger av de forskjellige metodene, se [6] og [22], viser det seg at Fleet og Jepson's metode gir de beste resultatene, men den er også den mest tidkrevende metoden. En modifisert versjon Horn og Schunk's metode, som er

en av de eldste, gir noe dårligere resultat noe raskere, men er fremdeles tidkrevende. Alle metodene er uansett forholdsvis ressurskrevende, samtidig som at de er utsatte for støy. Og det å bestemme den optiske flyten er også bare en del av arbeidet. Det må alltid ekstra algoritmer til for å tolke informasjonen den gir. Det vil si at man må regne med en del ekstra prosessorforbruk for å få det komplette systemet til å virke i sanntid.

Jeg har valgt å bruke Horn og Schunk's metode først og fremst fordi den er forholdsvis enkel å implementere, men også fordi den gir godt kompromiss mellom kvaliteten på resultatet og kravene til ressurser. Med de aller beste metodene jeg har funnet, se for eksempel [23], snakker man ofte om sekunder per bilde isteden for bilder per sekund (computation time). Det er selvfølgelig svært vanskelig å bruke slike metoder i en applikasjon som skal kjøre i sanntid.

2.1.3 Horn & Schunk's metode

La $I(x, y, t)$ være lysintensiteten ved punkt (x, y) ved tiden t. Da kan antagelsen av at lysintensiteten holder seg konstant over tid uttrykkes som:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) = I(x + u\delta t, y + v\delta t, t + \delta t)$$

der (u, v) er den horisontale og vertikale hastigheten ved ett punkt. (Man legger til grunn at δt er liten.) Dette forteller bare at verdien til punkt (x, y) ved tid t er den samme ved en et senere tidspunkt, men da forflyttet ($\delta x, \delta y$) av den optiske flyten. Gradientbaserte metoder [24] tar Taylorrekkeutviklingen av høyre side av ligningen slik at:

$$I(x, y, t) = I(x, y, t) + I_x u\delta t + I_y v\delta t + I_t\delta t + e$$

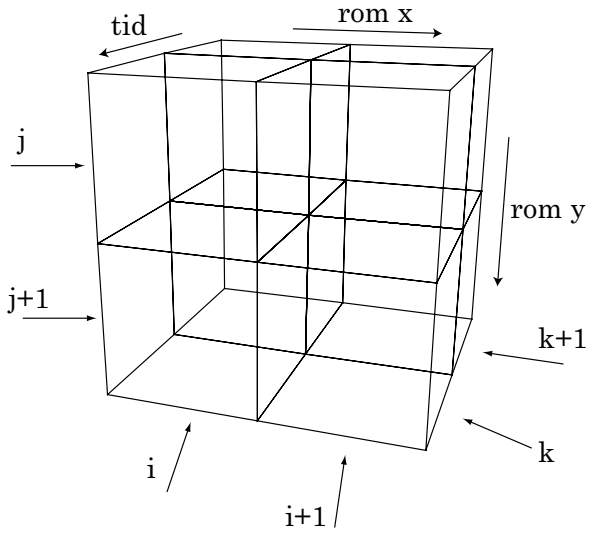
der I_x, I_y og I_t er de første ordens partielle deriverte av lysintensiteten I med hensyn på x, y og t, og der e inneholder de høyere ordens deriverte. Ved å trekke fra $I(x, y, t)$ og dele med δt får vi:

$$I_x u + I_y v + I_t = 0$$

som er den første begrensningen av problemet (man ser bort i fra e siden δt er liten). *The data conservation term* blir da:

$$E_D(u, v) = p(I_x u + I_y v + I_t)$$

Dette gir oss retningen til bevegelsen, men ikke hastigheten. Derfor trengs det som nevnt en begrensning til. En mulighet er å bruke *the smoothness constraint* (spatial koherens) som vil si at man antar at bevegelsen ved et punkt er lik bevegelsene til nabopunktene. Dersom man greier å sette et tall på forskjellen i disse bevegelsene ønsker man å minimalisere



$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$
$\frac{1}{6}$	-1	$\frac{1}{6}$
$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{12}$

Figur 4: Venstre: Indekser ved estimering av de partielle deriverte. Høyre: En *Laplacian mask*.

dette tallet. Dette tallet kan være summen av kvadratene til Laplace av x- og y-komponentene av den optiske flyten. Laplace av u og v er definert som:

$$\nabla^2 u = \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} \quad \nabla^2 v = \frac{\delta^2 v}{\delta x^2} + \frac{\delta^2 v}{\delta y^2}$$

Dette betyr at man straffer forskjeller i flyten.

Estimering av de partielle deriverte

$$\begin{aligned} I_x \approx \tfrac{1}{4}(&I_{i,j+1,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i+1,j,k} + \\ &I_{i,j+1,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i+1,j,k+1}) \\ I_y \approx \tfrac{1}{4}(&I_{i+1,j,k} - I_{i,j,k} + I_{i+1,j+1,k} - I_{i,j+1,k} + \\ &I_{i+1,j,k+1} - I_{i,j,k+1} + I_{i+1,j+1,k+1} - I_{i,j+1,k+1}) \\ I_t \approx \tfrac{1}{4}(&I_{i,j,k+1} - I_{i,j,k} + I_{i+1,j,k+1} - I_{i+1,j,k} + \\ &I_{i,j+1,k+1} - I_{i,j+1,k} + I_{i+1,j+1,k+1} - I_{i+1,j+1,k}) \end{aligned} \tag{2.2}$$

der k uttrykker tiden.

Estimering av Laplace av flyten

$$\nabla^2 u \approx \kappa(\bar{u}_{i,j,k} - u_{i,j,k}), \qquad \nabla^2 v \approx \kappa(\bar{v}_{i,j,k} - v_{i,j,k})$$

der

$$\begin{aligned}\bar{u}_{i,j,k} &= \tfrac{1}{6}(u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k}) + \\ &\quad \tfrac{1}{12}(u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}) \\ \bar{v}_{i,j,k} &= \tfrac{1}{6}(v_{i-1,j,k} + v_{i+1,j,k} + v_{i,j-1,k} + v_{i,j+1,k}) + \\ &\quad \tfrac{1}{12}(v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k})\end{aligned} \tag{2.3}$$

dersom $\kappa = 3$.

Minimalisering

$$E_D = I_x u + I_y v + I_t$$

$$E_S = \left(\frac{\delta u}{\delta x}\right)^2 + \left(\frac{\delta u}{\delta y}\right)^2 + \left(\frac{\delta v}{\delta x}\right)^2 + \left(\frac{\delta v}{\delta y}\right)^2$$

der E_D er data conservation og E_S er spatial koherens. De to forskjellige faktorene blir vektet med α. Total feil (error E) som skal minimaliseres er:

$$E^2 = \iint (\alpha^2 E_S^2 + E_D^2) dx dy$$

Minimaliseringen kan oppnås ved å finne passende verdier for hastigheten til den optiske flyten (u, v).

$$\begin{aligned} I_x^2 u + I_x I_y v &= \alpha^2 \nabla^2 u - I_x I_t \\ I_y^2 v + I_x I_y u &= \alpha^2 \nabla^2 v - I_y I_t \end{aligned}$$

Ved å bruke estimatet fra Laplace:

$$\begin{aligned} (\alpha^2 + I_x^2) u + I_x I_y v &= (\alpha^2 \bar{u} - I_x I_t) \\ (\alpha^2 + I_y^2) v + I_x I_y u &= (\alpha^2 \bar{v} - I_y I_t) \end{aligned}$$

Ved å løse for u og v:

$$\begin{aligned} (\alpha^2 + I_x^2 + I_y^2) u &= (\alpha^2 + I_y^2)\bar{u} - I_x I_y \bar{v} - I_x I_t \\ (\alpha^2 + I_x^2 + I_y^2) v &= (\alpha^2 + I_x^2)\bar{v} - I_x I_y \bar{u} - I_y I_t \end{aligned}$$

Forskjellen i flyt ved ett punkt ut i fra lokalt gjennomsnitt:

$$\begin{aligned} (\alpha^2 + I_x^2 + I_y^2)(u - \bar{u}) &= -I_x(I_x \bar{u} + I_y \bar{v} + I_t) \\ (\alpha^2 + I_x^2 + I_y^2)(v - \bar{v}) &= -I_y(I_x \bar{u} + I_y \bar{v} + I_t) \end{aligned}$$

Algoritme OPTISKFLYT

Gitt to bilder X_k og X_{k-1} i en bildesekvens ved tid $t = k$ og $t = k - 1$. Gitt to hastighetskart $u_{i,j}^{k-1}$ og $v_{i,j}^{k-1}$ fra tid $t = k - 1$. Bestem de nye hastighetskartene $u_{i,j}^{k}$ og $v_{i,j}^{k}$.

1. **For** alle kolonner (i X)
2. **For** alle rader (i X)
   (a) Bestem derivatene I_x, I_y og I_t med formel 2.2
   (b) Bestem de lokale gjennomsnittshastighetene $\bar{u}^{k-1}$ og $\bar{v}^{k-1}$ med formel 2.3
   (c) Bestem de nye hastighetene u^k og v^k med formel 2.4.

Iterativ løsning blir:

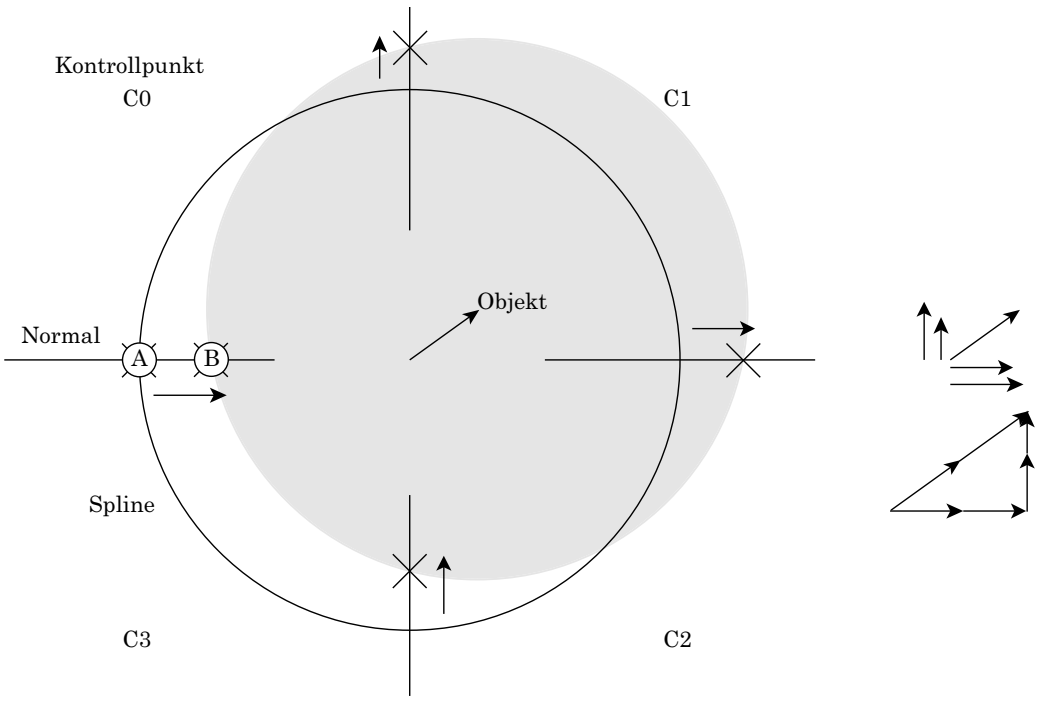
$$\begin{aligned} u^{k+1} &= \bar{u}^k - I_x \frac{I_x\bar{u}^k + I_y\bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \\ v^{k+1} &= \bar{v}^k - I_y \frac{I_x\bar{u}^k + I_y\bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \end{aligned} \tag{2.4}$$

OPTISKFLYT er den komplette algoritmen. Den er ganske enkel og består egentlig bare av konvolusjoner. Først en temporær / spatial differanse av bildet og så en laplace av hastighetene i x- og y-retning. Tidskompleksiteten er $O(n^2)$ dersom bredde og høyde på bildene er lik n. Plasskompleksiteten er $O(4n^2)$ siden algoritmen må opprettholde to bilder og to hastighetskart hele tiden.

2.2 Contour Tracking with Splines

Anta en bildesekvens av en sirkelflate som translerer rolig rundt i planet. Se figur 5. For å spore denne sirkelen definerer man først en spline som passer konturen til sirkelen. (Det vil si å definere en lukket spline med minst fire kontrollpunkter.) Så *legger* man denne splinen over første bilde i sekvensen og forsøker å *matche* den til sirkelen. Langs hele splinen beregner man normaler med bestemte mellomrom, og langs hver normal utfører man en 1-D konvolusjon for å finne kantinformasjon. Ved å sammenligne punkt A, som er der normalen krysser splinen, med punkt B, som er der kanten i bildet er, kan man si noe om forholdet mellom sirkelen i bildet og splinen. Det vil si forskyvningen mellom de. Dersom man beregner den

ønskelige forflyttningen for hver normal (en av de er altså B-A) og sammenligner disse kan man si noe om hvordan hele splinen burde forflyttes. For dersom man forflytter et og et kontrollpunkt vil splinen fort deformeres til det ugjenkjennelige.



Figur 5: Hvordan matche en spline til et objekt

Man ønsker selvfølgelig at deformasjon skal være mulig, men dette må gjøres på en kontrollert måte. Det vil si at man gir splinen visse friheter som translering, rotasjon, skalering osv. Jo mer kompleks bevegelse man vil spore, jo flere friheter må splinen nødvendigvis ha. Dersom man sporer en bil som velter vil splinen som følger den forandre seg ganske mye. Mulige / lovlige transformasjoner defineres i *shape-space*, som Blake og Isard [17] kaller det. To vanlige klasser er de euklidske og affine transformasjonene.

I figur 5 er bare translasjon nødvendig. Derfor holder det å regne ut gjennomsnittsvektoren for å kunne *låse* splinen på objektet. Hvis derimot objektet er større eller mindre enn splinen vil en gjennomsnittsvektor være ubruklig. Den vil rett og slett bare være en nullvektor siden vektorene nuller hverandre ut. Vi skal senere se hvordan man håndterer rotasjon, skalering og andre mer avanserte transformasjoner.

For å kunne låse en spline til et objekt i bildet trenger man kanskje flere iterasjoner for å greie det. I videosekvenser vil hver iterasjon typisk være

koblet til hvert enkelt bilde i sekvensen. Det vil si at splinen stadig er på jakt etter objektet siden objektet også kan / vil forflytte seg. Overgangen fra stillbilder til bildesekvenser er derfor ganske naturlig. Det er imidlertid visse begrensninger. Hvis objektet forflytter seg alt for fort, eller *hopper rundt* i bildet, vil det selvfølgelig være vanskelig for splinen å kunne følge etter fordi den alltid vil være på etterskudd og fordi den kan komme så langt unna at normalene ikke lenger krysser noe del av objektet i det hele tatt. For å kunne spore objekter i bevegelse må man altså ty til mer logikk. For det første; om splinen mister objektet helt kan den øke lengden på normalene helt til den fanger opp objektet igjen. Den kan også øke sin egen størrelse, skalere, helt til den finner objektet, og så forminske seg igjen ned på objektet. For det andre; når en spline har fulgt et objekt en stund, kanskje bare for noen få iterasjoner, kan den kanskje forutsi hvordan objektet vil forflytte seg i neste iterasjon og ta denne forflyttingen med i beregningen med en gang. Det vil si å alltid ha en formening om fartsvektoren til objektet og bruke denne informasjonen aktivt.

En annen utfordring er støy. Kanskje ikke kamerastøy, men støy i form av andre objekter og bakgrunnen. For å takle dette kan det benyttes mer avanserte statistiske modeller som hele tiden kan veilede splinen i dens jakt på objektet.

(Det meste av det jeg presenterer av konturspоring med splines er basert på boka *Active Contours* [17]. Jeg forsøker å gi en forenklet og intuitiv framstilling av metoden, men siden konseptet er forholdsvis omfattende og komplisert er min framstilling på ingen måte komplett.)

2.2.1 Splines

En spline er en spesiell kurve som er definert med en eller flere polynomer over forskjellige intervaller der hver overgang mellom polynomene er interpolert [25]. En spline med *orden* d bruker polynomer med orden d, eller *grad* $d-1$. Hvert intervall kalles for *spenn* eller *stykke*, og er knyttet sammen med *avbruddspunkt*. Man representerer ofte splines på parametrisk form fordi det er enkelt å håndtere. En parametrisk kurve kan defineres som $(x(s), y(s))$, der s er en parameter som stiger ettersom kurven traverseres, og x og y er funksjoner av s som representerer kurven i henholdsvis x- og y-retning. Vanligvis bruker man splines med orden 3 eller 4, som kalles henholdsvis *kvadratiske* og *kubiske* splines.

En B-spline er en spesiell utgave av splines og består av N_B vektede *basisfunksjoner*. Hver basisfunksjon består av d polynomer som hver er definert over sitt intervall (spenn) av s. Hvert spenn er koblet sammen med *knuter*. Dersom knutene er definert i jevne mellomrom er splinen *uniform*, hvis ikke er den *ikke-uniform*. En B-spline kan defineres formelt slik:

$$x(s) = \sum_{n=0}^{N_B-1} x_n B_n(s)$$

der x_n er vektene som blir anvendt på basisfunksjonene $B_n(s)$ og N_B er antall funksjoner / vekter. Disse vektene er som oftest kjent som *kontrollpunkt* (altså vekter i et 2-D plan). Definert på en annen måte:

$$\mathbf{r}(s) = (x(s), y(s)) = \sum_{n=0}^{N_B-1} B_n(s) q_n$$

for $0 \leq s \leq L$, der $q_n = (q_n^x, q_n^y)^t$ er kontrollpunkt. Isteden for å jobbe med hvert enkelt kontrollpunkt hver for seg representeres de heller som *kontrollvektorer*. Da kan man bruke:

$$\mathbf{r}(s) = U(s)\mathbf{Q}$$

der

$$U(s) = \begin{pmatrix} \mathbf{B}(s)^T & 0 \\ 0 & \mathbf{B}(s)^T \end{pmatrix}, \quad \mathbf{B}(s) = (B_0(s), B_1(s), \ldots, B_{N_B-1}(s))^T$$

og

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}^x \\ \mathbf{Q}^y \end{pmatrix}, \quad \mathbf{Q}^x = \begin{pmatrix} q_0^x \\ \cdots \\ \cdots \\ q_{N_B-1}^x \end{pmatrix}, \quad \mathbf{Q}^y = \begin{pmatrix} q_0^y \\ \cdots \\ \cdots \\ q_{N_B-1}^y \end{pmatrix}$$

2.2.2 Shape-space

Man kan som nevnt ikke manipulere hvert enkelt kontrollpunkt til en spline uavhengig av hverandre for da mister splinen sin *indre* styrke og kan bli deformert til det ugjenkjennelige. En *fri* spline har $2N_B$ frihetsgrader. Det vil si at en spline definert med bare 10 kontrollpunkt har 20 frihetsgrader. For å begrense antall frihetsgrader definerer man et *shape-space*.

Et shape-space $\mathcal{S} = L(W, \mathbf{Q}_0)$, er en lineær avbildning av en *shape-space vektor* $\mathbf{X} \in \mathbb{R}^{N_X}$ på en splinevektor $\mathbf{Q} \in \mathbb{R}^{N_Q}$:

$$\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0$$

der W er en $N_Q \times N_X$ *shape matrix*. Konstanten $\mathbf{Q}_0$ er en kurvemal / utgangspunktet som forskjellige former kan måles mot. Det vil si at kurver

som ligner på eller er like $\mathbf{Q}_0$ kan grupperes sammen vha. et definert shape-space S og en definert $\mathbf{X}$.

For euklidske likheter (translasjon, rotasjon, isotropisk skalering) er

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{pmatrix}$$

der $\mathbf{0} = (0, 0, \ldots, 0)^T$ og $\mathbf{1} = (1, 1, \ldots, 0)^T$.
Shape-space vektor $\mathbf{X}$ inneholder informasjon om de mulige transformasjoner:

1. $\mathbf{X} = (0, 0, 0, 0)^T$ vil si ingen transformasjon. $\mathbf{Q} = \mathbf{Q}_0$
2. $\mathbf{X} = (1, 0, 0, 0)^T$ vil si translasjon i x-retning. $\mathbf{Q} = \mathbf{Q}_0 + (1, 0)^T$
3. $\mathbf{X} = (0, 1, 0, 0)^T$ vil si translasjon i y-retning. $\mathbf{Q} = \mathbf{Q}_0 + (0, 1)^T$
4. $\mathbf{X} = (0, 0, 1, 0)^T$ vil si skalering. $\mathbf{Q} = 2\mathbf{Q}_0$
5. $\mathbf{X} = (0, 0, \cos\theta - 1, \sin\theta)^T$ vil si rotasjon θ°.

For affine likheter (translasjon, rotasjon, full skalering) er

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{0} \end{pmatrix}$$

og

1. $\mathbf{X} = (0, 0, 0, 0, 0, 0)^T$ vil si ingen transformasjon.
2. $\mathbf{X} = (1, 0, 0, 0, 0, 0)^T$ vil si translasjon i x-retning.
3. $\mathbf{X} = (0, 0, 1, 0, 0, 0)^T$ vil si en dobling i bredde.
4. $\mathbf{X} = (0, 0, 1, 1, 0, 0)^T$ vil si en dobling i størrelse.
5. $\mathbf{X} = (0, 0, \cos\theta - 1, \cos\theta - 1, -\sin\theta, -\sin\theta)^T$ vil si rotasjon θ°.

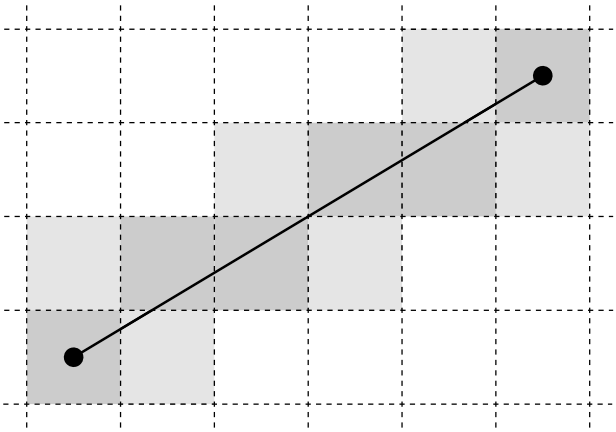
Det finnes flere klasser som for eksempel tredimensjonale transformasjoner. Det er også mulig å skreddersy shape-spaces til spesifikke problemer.

Oppsummering: $\mathbf{X}$ er *transformasjonen* uttrykt som en vektor, W er et *shape-space* representert som en matrise. $\mathbf{Q}_0$ er de originale *kontrollpunkt* (eller utgangspunktet). Antall *frihetsgrader* betegnes som N_X.

2.2.3 Image features

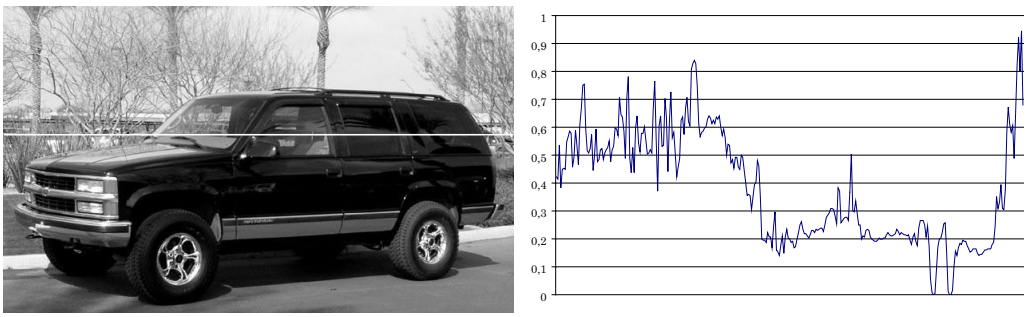
Langs hver normal som beregnes ut i fra splinen må man altså finne kantinformasjon. Først må man beregne selve linjene, f.eks. med Bresenhams linjealgoritme [26], og så kan man utføre forskjellige typer konvolusjoner for å finne de trekkene man ønsker. (Man lagrer altså linjen i en endimensjonal tabell isteden for å tegne den.) For å oppnå et godt resultat lønner det

seg å interpolere flere punkt, typisk 4 naboer. I figur 6 ser man et eksempel. Den svarte linjen forestiller en normal, de mørkeste pixlene er diskretiseringen av linjen og de lysegrå pixlene er de som kan interpoleres med for å få et bedre grunnlag til konvolusjonen.



Figur 6: Hvordan trekke ut linjeinformasjon fra et bilde

I figur 7 er det blitt trukket ut en horisontal linje fra et bilde av en bil. Det er forholdsvis lett å se på grafen hvor bilen er, men grafen viser også hvor mye støy som finnes i et naturlig bilde. Legg særlig merke til buskene til venstre i bildet og frontruta. Støyen fra bakgrunnen gjør det vanskelig å bestemme akkurat hvor frontruta begynner. Derfor er det viktig å bruke riktige konvolusjonsfiltre for at algoritmen senere skal få et best mulig grunnlag.



(a) **(b)**

Figur 7: Lysintensitet langs en horisontal linje

2.2.4 Kurvetilpassning

Kurvetilpassning er en videreutviklet, eller avansert form for, sammenligning av kurver. Som definert tidligere uttrykker $\mathbf{r}$ en kurve / spline som man ønsker å finne i et bilde. På samme måte kan man definere den egentlige konturen i bildet som $\mathbf{r}_f$ (f for feature). Det vil si den "ekte" kurven som er et resultat av (avbildningen av) objektet i bildet. Ved å sammenligne $\mathbf{r}$ med $\mathbf{r}_f$ kan man si noe om hvor godt en mulig kontur passer med det faktiske objektet i bildet.

Det vi ønsker er å transformere $\mathbf{r}$ slik at $\mathbf{r} \approx \mathbf{r}_f$. På den annen side ønsker vi også at $\mathbf{r}$ beholder sin originale form, dvs. at den ikke skal deformeres for mye. Siden $\mathbf{r}_f$ er et resultat av ikke-ideelle data kan den faktisk være ganske deformert. Deler av det avbildede objektet kan for eksempel være skjult eller ødelagt av støy. For å bøte på dette sammenligner man $\mathbf{r}$ med $\bar{\mathbf{r}}$ også, der $\bar{\mathbf{r}}$ vil si *malkurven* eller *gjennomsnittskurven* (den ideelle formen). Dette kalles for regularisering.

For å sammenligne to funksjoner er det mulig å bruke den såkalte L_2-*normen*:

$$||x|| = \sqrt{\frac{1}{L}\int_{s=0}^{L} |x(s)|^2 ds}$$

som egentlig bare vil si *root-mean-square* av $x(s)$ over $0 \leq s \leq L$. Dersom man definerer $x(s) = \mathbf{r}(s) - \mathbf{r}_f(s)$ får man et godt sammenligningstall for kurvene. Man kan derfor uttrykke kurvetilpassning som et minimaliseringsproblem av:

$$\alpha||\mathbf{r} - \bar{\mathbf{r}}||^2 + ||\mathbf{r} - \mathbf{r}_f||^2 \tag{2.5}$$

der α er en konstant som regulerer hvor stor innflytelse malkurven skal ha. Høyere α vil si at $\mathbf{r}$ blir mer lik $\bar{\mathbf{r}}$ fordi man straffer ulikheter med malkurven mer enn ulikheter med de observerte data. Overført til shapespace kan dette skrives:

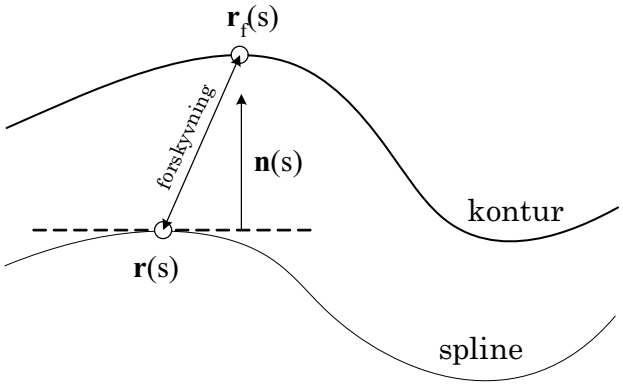
$$\alpha||\mathbf{X} - \overline{\mathbf{X}}||^2 + ||\mathbf{Q} - \mathbf{Q}_f||^2 \tag{2.6}$$

Man ønsker å holde $\mathbf{r}$ forholdsvis lik $\bar{\mathbf{r}}$, men det gjelder bare form, ikke posisjon eller orientering. Derfor innføres en vektmatrise $\overline{S}$ som avgrenser påvirkningen til bare form:

$$(\mathbf{X} - \overline{\mathbf{X}})^T \overline{S} (\mathbf{X} - \overline{\mathbf{X}}) + ||\mathbf{Q} - \mathbf{Q}_f||^2 \tag{2.7}$$

der

$$\overline{S} = \alpha\mathcal{H}, \quad \mathcal{H} = W^T \mathcal{U} W, \quad \mathcal{U} = \begin{pmatrix} \mathcal{B} & 0 \\ 0 & \mathcal{B} \end{pmatrix}, \quad \mathcal{B} = \frac{1}{L}\int_0^L \mathbf{B}(s)\mathbf{B}(s)^T ds$$



Figur 8: Normal Displacement

Det er kanskje ikke så enkelt å forstå hva $\overline{S}$ egentlig er. Enkelt sagt er den bare en begrensning av shape-space-matrisen W. $\overline{S}$ vil være en matrise med størrelse $m \times m$ der m er antall frihetsgrader i shape-spacet (dvs. kolonner i W). Derfor kan man si at venstre side (av 2.7) bidrar med bare formdata mens høyre side bidrar med først og fremst posisjon- og orienteringsdata, og noe formdata. (For en mer utfyllende forklaring se [17].)

L_2-normen fungerer ikke så bra dersom det er forskjeller i parameteriseringene til de forskjellige kurvene. Dvs. at kontrollvektorene er forskjøvet i forhold til hverandre. For å løse dette kan man definere en funksjon $g(s)$ som *mapper* punkt på r til de riktige punktene på $\mathbf{r}_f$. Isteden for $||\mathbf{r}(s) - \mathbf{r}_f(s)||$ brukes heller den alternative kurveforskyvningsmålingen $d(\mathbf{r}, \mathbf{r}_f)$:

$$d^2 = min_g \frac{1}{L} \int (\mathbf{r}(g(s)) - \mathbf{r}_f(s))^2$$

definert som minimum over alle mulige reparameteriseringer.

Nå er det ikke realistisk å teste alle mulige reparameteriseringer, derfor må man forenkle problemet litt. Forskyvingen $||\mathbf{r}(s) - \mathbf{r}_f(s)||$ ved punkt s kan uttrykkes som en sum av to komponenter parallelle med henholdsvis tangenten og normalen til kurven (se figur 8). Komponenten parallell med tangenten beskriver tilnærmelsesvis forskyving langs kurven og kan derfor også oppfattes som forskyvning mellom parameteriseringene til de to kurvene. Hvis man ser vekk fra denne komponenten, og bare bruker normalkomponenten, vil man altså oppnå parameteriseringsuavhengighet. (Dette er faktisk hva som er gjort i fig. 5.) Distansen mellom de to kurvene blir derfor:

$$d(\mathbf{r}, \mathbf{r}_f) \approx \frac{1}{L} \int [(\mathbf{r}(s) - \mathbf{r}_f(s)) \cdot \mathbf{n}(s)]^2 ds$$

der $\mathbf{n}$ er normalkomponenten. Denne brukes så til å definere en ny norm $||\cdot||_{\overline{\mathbf{n}}}$ som tar hensyn til malkurven $\overline{\mathbf{r}}$ og dens normaler $\overline{\mathbf{n}}$:

$$||\mathbf{r}||^2_{\overline{\mathbf{n}}} \equiv \frac{1}{L}\int [\mathbf{r}(s)\cdot\overline{\mathbf{n}}(s)]^2 ds \tag{2.8}$$

og har egenskapen:

$$||\mathbf{r}-\mathbf{r}_f||_{\overline{\mathbf{n}}} \approx d(\mathbf{r},\mathbf{r}_f)$$

gitt at både $\mathbf{r}$ og $\mathbf{r}_f$ er forholdsvis nære $\overline{\mathbf{r}}$. Denne alternative normen kalles *normal displacement*. Diskretisert:

$$||\mathbf{r}-\mathbf{r}_f||^2_{\overline{\mathbf{n}}} \approx \frac{1}{N}\sum_{i=1}^{N}[(\mathbf{r}_f(s_i)-\mathbf{r}(s_i))\cdot\overline{\mathbf{n}}(s_i)]^2 \tag{2.9}$$

I shape space:

$$\mathbf{r}(s_i) = U(s_i)(W\mathbf{X}+\mathbf{Q}_0)$$

$$||\mathbf{r}-\mathbf{r}_f||^2_{\overline{\mathbf{n}}} \approx \frac{1}{N}\sum_{i=1}^{N}\left(v_i - \mathbf{h}(s_i)^T[\mathbf{X}-\overline{\mathbf{X}}]\right)^2 \tag{2.10}$$

$$v_i = (\mathbf{r}_f(s_i)-\overline{\mathbf{r}}(s_i))\cdot\overline{\mathbf{n}}(s_i))$$

$$\mathbf{h}(s)^T = \overline{\mathbf{n}}(s_i)^T U(s_i) W$$

Ut i fra 2.7 og 2.10 blir det endelige uttrykket:

$$T = (\mathbf{X}-\overline{\mathbf{X}})^T\overline{S}(\mathbf{X}-\overline{\mathbf{X}}) + \sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left(v_i-\mathbf{h}(s_i)^T[\mathbf{X}-\overline{\mathbf{X}}]\right)^2 \tag{2.11}$$

Konstanten σ er *the measurement error*. TRIVIELLKURVETILPASSNING er den komplette algoritmen. For utledningen av $\hat{\mathbf{X}}$ se [17].

[I algoritmen over er v_i bare ett tall. Dersom shape-spacet har 4 frihetsgrader er W en 2×4 matrise, S_i er 4×4 og $\mathbf{h}(s_i)$, Z og $\mathbf{X}$ er 4×1.]

Det er viktig å understreke at $\mathbf{r}_f$ ikke er en definert spline. Den er bare en *tenkt* spline som representerer konturen til et objekt i et bilde. Siden man bruker *normal displacement* trenger man heller ikke å definere denne, noe som hadde vært ganske kostbart. I praksis bestemmer man $\mathbf{r}_f(s_i)$ ved å søke etter kanter langs normalene til $\overline{\mathbf{r}}$ ved hjelp av lineær konvolusjon, som vist tidligere. Tallet $v_i(s_i)$ uttrykker altså bare hvor langt ut på normalen $\mathbf{r}(s_i)$ ligger i forhold til kurven som skal tilpasses. Siden det er ingen garanti for at man finner den ekte kanten til objektet er det lurt å tolke v_i før man bruker den. Det vil si at dersom den er svært positiv eller

Algoritme TRIVIELLKURVETILPASSNING

Gitt et initielt kurveestimat $\overline{\mathbf{r}}(s)$ med normaler $\overline{\mathbf{n}}$, og regulariseringsvektmatrise $\overline{S}$ løs $min\ T$ (2.11):

1. Bestem $s_i, i = 1, \ldots, N$, $s_1 = 0, s_{i+1} = s_i + h, S_N = L$.
2. For hver i, benytt et filter langs en passene linje (f.eks. normal) som går gjennom $\overline{\mathbf{r}}(s_i)$, for å finne posisjonen til $\mathbf{r}_f(s_i)$.
3. Initialiser $\mathbf{Z}_0 = 0$, $S_0 = 0$.
4. **For** hver $i = 1, \ldots, N$

$$
\begin{aligned}
v_i &= (\mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i)) \cdot \overline{\mathbf{n}}(s_i) \\
\mathbf{h}(s_i)^T &= \overline{\mathbf{n}}(s_i)^T U(s_i) W \\
S_i &= S_{i-1} + \tfrac{1}{\sigma_i^2} \mathbf{h}(s_i)\mathbf{h}(s_i)^T \\
\mathbf{Z}_i &= \mathbf{Z}_{i-1} + \tfrac{1}{\sigma_i^2} \mathbf{h}(s_i) v_i
\end{aligned}
$$

5. Den totale observasjonsvektoren er $\mathbf{Z} = \mathbf{Z}_N$, med tilhørende statistisk informasjon $S = S_N$.
6. Den beste tilpassede kurve er $\hat{\mathbf{X}} = \overline{\mathbf{X}} + (\overline{S} - S)^{-1}\mathbf{Z}$.

negativ kan den være et resultat av en annen kant (til et annet objekt eller pga støy). Da kan det være lurt å ikke bruke den i det hele tatt, dvs. at algoritmen rett og slett hopper over en iterasjon dersom $|v_i|$ er stor (i FOR løkka). Problemet er å sette en fornuftig grense. Dette blir kalt *Validation Gate*.

2.2.5 Statistiske modeller

Med TRIVIELLKURVETILPASSNING kan man tilpasse kurver til konturer (objekter) i enkeltbilder. Man kunne kanskje ha sporet (tracket) objekter over flere bilder, men det ville bare virket med svært sene bevegelser. Algoritmen tolererer heller ikke særlig mye støy. Den er altså ikke spesielt robust. Derfor er det mulig å bruke statistikk og modellere fordelinger som beskriver hvordan form og bevegelse utvikler seg over tid. Det vil si at man ved en hver tid forsøker å forutsi hvordan objektet vil se ut og hvor det vil befinne seg i neste bilde basert på erfaring / læring.

Hva vi ønsker å estimere er $\mathbf{X}$ som beskriver en kurves forandring fra bilde til bilde basert på observasjonen $\mathbf{r}_f(s)$. I statistisk setting kan man uttrykke dette som $p(\mathbf{X}|\mathbf{r}_f)$, som vil si *hva er sannsynligheten for* $\mathbf{X}$ *gitt* $\mathbf{r}_f$. Dette er den betingede simultanfordeling, eller *aposteriorifordelingen*. Bayes formel sier at aposteriorifordelingen kan beregnes som produktet av *apriorifordelingen* $p_0(\mathbf{X})$ og *observasjonsfordelingen* $p(\mathbf{r}_f|\mathbf{X})$:

$$p(\mathbf{X}|\mathbf{r}_f) \propto p(\mathbf{r}_f|\mathbf{X})p_0(\mathbf{X})$$

Observasjonsfordelingen $p(\mathbf{r}_f|\mathbf{X})$ beskriver selve forandringen / deformeringen (degraderingen i bildeanalyse) av konturen som observeres over tid og kalles ofte for *likelyhoodfordelingen*. Apriorifordelingen $p_0(\mathbf{X})$ modellerer forhåndskunnskap og kan for eksempel inneholde L_2-normen eller den alternative normen (2.8).

En liten understrekning: $\mathbf{r}$ er splinen, som skal tilpasses konturen $\mathbf{r}_f$. $\mathbf{X}$ er transformasjonen som er nødvendig for å beregne $\mathbf{r}$ ut i fra et utgangspunkt $\mathbf{Q}_0$. $\overline{\mathbf{X}}$ er den ideelle formen (egentlig den ideelle transformasjonen ut i fra $\mathbf{Q}_0$) som brukes for å hindre at $\mathbf{r}$ blir deformert for mye. Poenget er at $p(\mathbf{X})$ kan oppfattes som en normalfordelt sannsynlighetsfunksjon med $\overline{\mathbf{X}}$ som forventningsverdi. Dvs. at dersom man ønsker å sample $\mathbf{X}$ så vil man få verdier (eller vektorer) som ligger nært $\overline{\mathbf{X}}$. $p(\mathbf{r})$ kan også oppfattes som normalfordelt med $\mathbf{r}_f$ som forventningsverdi. Disse to konkurrerende faktorene, som uttrykkes i ligning 2.5 og 2.6, kan altså smeltes sammen til én sannsynlighetsfunksjon som også er normalfordelt vha. Bayes formel. Og det er dette som er aposteriorifordelingen.

Vi ønsker altså å bestemme aposteriorifordelingen og for å gjøre dette må vi først bestemme likelyhoodfordelingen og apriorifordelingen. Når

man har kalkulert denne kan man bestemme hvor god en mulig X er eller, om man tar inversen av funksjonen, sample gode / sannsynlige versjoner av X. Målet er uansett å maksimere sannsynligheten og derfor kalles det ofte *Maximum A Posteriori* (MAP).

Apriorifordelingen for kurver i shape-space $\mathcal{S}$ konsistent med den generelle kvadratiske regularisatoren $(\mathbf{X} - \overline{\mathbf{X}})^T \overline{S} (\mathbf{X} - \overline{\mathbf{X}})$ er:

$$p_0(\mathbf{X}) \propto \exp -\frac{1}{2}(\mathbf{X} - \overline{\mathbf{X}})^T \overline{S} (\mathbf{X} - \overline{\mathbf{X}})$$

med kovarians

$$P_{\mathbf{r}}(s) = U(s) W P_0 W^T U(s)^T, \quad P_0 = \overline{S}^{-1}$$

Apriorifordelingen uttrykker altså forholdet mellom $\overline{\mathbf{r}}$ og $\mathbf{r}$, den indre styrken til splinen. Et gitt punkt $\mathbf{r}(s)$ kan derfor samples fra en todimensjonal normalfordeling $\mathcal{N}(\overline{\mathbf{r}}(s), P_{\mathbf{r}}(s))$.
Likelyhoodfordelingen uttrykker på samme måte den ytre styrken til splinen, dvs. forholdet mellom kurven $\mathbf{r}$ og de observerte data (features):

$$p(\mathbf{r}_f | \mathbf{X}) \propto \exp -\frac{N_X}{2\overline{\rho_f}^2} ||\mathbf{r} - \mathbf{r}_f||^2_{\overline{\mathbf{n}}}$$

$$p(\mathbf{r}_f | \mathbf{X}) \propto \exp -\frac{1}{\sigma^2} \sum_{i=1}^{N} [(\mathbf{r}_f(s_i) - \mathbf{r}(s_i)) \cdot \overline{\mathbf{n}}(s_i)]^2$$

$$\sigma = \overline{\rho_f} \sqrt{\frac{N}{N_X}}$$

Konstanten $\overline{\rho_f}$ uttrykker den gjennomsnittlige avstanden mellom $\mathbf{r}$ og $\mathbf{r}_f$, for alle punkt, kun basert på målefeil.

Aposteriorifordelingen kan uttrykkes som:

$$\mathbf{X} | \mathbf{Q}_f \sim \mathcal{N}(\hat{\mathbf{X}}, P), \quad P = S^{-1}$$

Dens forventningsverdi $\hat{\mathbf{X}}$ uttrykker det samme som i TRIVIELLKURVETILPASSNING, men nå modifisert med hensyn til observasjonsvariansen. (Se [17] for utledningen av aposteriorifordelingen.)

$$\hat{\mathbf{X}} = S^{-1} \left(\overline{S}\, \overline{\mathbf{X}} + \frac{N_X}{\overline{\rho_f}^2} \mathcal{H} \mathbf{X}_f \right)$$

der

$$S = \overline{S} + \frac{N_X}{\overline{\rho_f}^2} \mathcal{H}$$

2.2.6 Dynamiske modeller for bevegelse

På dette nivået er det blitt definert statistiske modeller for form, men koherensen mellom flere bilder i en bildesekvens er fortsatt ikke benyttet. Isteden for å bruke en konstant apriorifordeling for alle bilder, så er det mulig å bruke aposteriorifordelingen fra bilde $k-1$ som apriorifordeling for bilde k. Man innfører derfor en betinget fordeling $p(\mathbf{X}_k|\mathbf{X}_{k-1})$ som gir fordelingene til mulighetene for den k-ende formen $\mathbf{X}_k$ *gitt* den forrige formen $\mathbf{X}_{k-1}$. Dette betyr en første ordens Markov-kjede, som vil si at $\mathbf{X}_k$ bare er avhengig av dens forrige verdi selv om den i prinsippet er relatert til *alle* tidligere verdier $\mathbf{X}_1 \dots \mathbf{X}_{k-1}$:

$$p(\mathbf{X}_k|\mathbf{X}_1 \dots \mathbf{X}_{k-1}) = p(\mathbf{X}_k|\mathbf{X}_{k-1})$$

For å modellere mer komplekse bevegelser må man imidlertid benytte seg av andre ordens Markov modeller [17], som vil si at man tar hensyn til to foregående tilstander isteden for bare én. Den dynamiske modellen kan på en enkel måte uttrykkes som en andre ordens lineær differensligning:

$$\mathcal{X}_k - \overline{\mathcal{X}} = A(\mathcal{X}_{k-1} - \overline{\mathcal{X}}) + B\mathbf{w}_k \tag{2.12}$$

der

$$\mathcal{X}_k = \begin{pmatrix} \mathbf{X}_{k-1} \\ \mathbf{X}_k \end{pmatrix} \quad \overline{\mathcal{X}} = \begin{pmatrix} \overline{\mathbf{X}} \\ \overline{\mathbf{X}} \end{pmatrix}$$

og

$$A = \begin{pmatrix} 0 & I \\ A_2 & A_1 \end{pmatrix} \quad B = \begin{pmatrix} 0 \\ B_0 \end{pmatrix}$$

Matrisen A uttrykker den deterministiske delen av den dynamiske modellen, mens B uttrykker den stokastiske delen. Den deterministiske delen beskriver hvordan man forventer at bevegelsen til det sporede objekt arter seg. Man kan for eksempel anta en fast og retningsbestemt bevegelse eller en urolig / svingende bevegelse. Den stokastiske delen beskriver den tilfeldige naturen til bevegelsen, eller det som kan oppfattes som støy. Hver $\mathbf{w}_k$ er uavhengige vektorer som inneholder N_X uavhengige og tilfeldige variabler mellom 0 og 1. Altså tilfeldig støy. A og B kan enten bestemmes manuelt eller læres ut ifra trening.

For å vise at den dynamiske modellen ér en temporær Markov-kjede kan den uttrykkes slik:

$$p(\mathcal{X}|\mathcal{X}_{k-1}) \propto \exp -\frac{1}{2} \left\| B^{-1}((\mathcal{X}_k - \overline{\mathcal{X}}) - A(\mathcal{X}_{k-1} - \overline{\mathcal{X}})) \right\|^2$$

Forventningsverdi og varians er:

$$\hat{\mathcal{X}}_k = \mathcal{E}[\mathcal{X}_k], \quad \mathcal{P}_k = \mathcal{V}[\mathcal{X}_k]$$

der

$$\mathcal{E}[\mathbf{Y}] = \int_{\mathbb{R}^{N_Y}} p(\mathbf{Y}) d\mathbf{Y}, \quad \mathcal{V}[\mathbf{Y}] = \mathcal{E}[(\mathbf{Y} - \mathcal{E}[\mathbf{Y}])(\mathbf{Y} - \mathcal{E}[\mathbf{Y}])^T]$$

for en tilfeldig variabel $\mathbf{Y}$.

2.2.7 Temporær filtrering med Kalman-filter

Som nevnt kan den statistiske modellen som er beskrevet uttrykkes som en normalfordeling:

$$\mathcal{X}_k \sim \mathcal{N}(\hat{\mathcal{X}}_k, \mathcal{P}_k)$$

gitt både den dynamiske modellen for $\mathcal{X}_k$ og målingshistorien $\underline{\mathbf{Z}}_k$:

$$\underline{\mathbf{Z}}_k = (\mathbf{Z}_1, \ldots, \mathbf{Z}_k)$$

der $\mathbf{Z}_k$ er den samme som i TRIVIELLKURVETILPASSNING ved tid $t = k$. Den statistiske infoen er, på samme måte, S_k. Redefinisjon av $\hat{\mathcal{X}}_k$ og $\mathcal{P}_k$, nå aposteriorisk gjennomsnitt og varians:

$$\hat{\mathcal{X}}_k = \mathcal{E}[\mathcal{X}_k | \underline{\mathbf{Z}}_k], \quad \mathcal{P}(t_k) = \mathcal{V}[\mathcal{X}_k | \underline{\mathbf{Z}}_k]$$

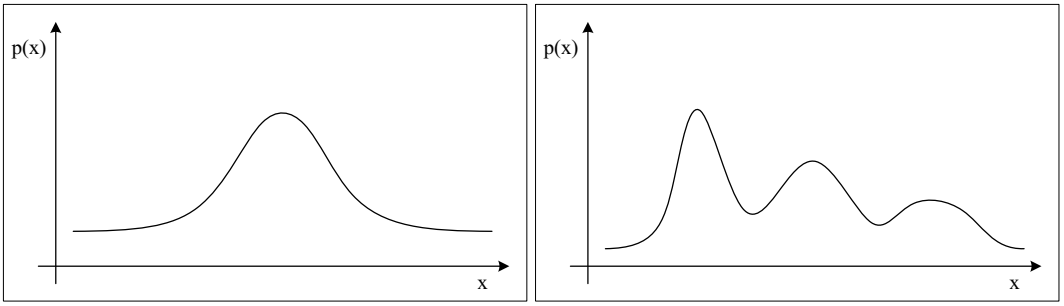
Siden X er normalfordelt kan man bruke Kalman-filter for å fjerne støy slik at metoden blir mer robust. Et Kalman-filter er et rekursivt filter som estimerer tilstanden til et dynamisk system fra en serie av målinger med støy. Det vil si at den fjerner støy i fra målingene ved å modellere virkeligheten. Den har to faser; *prediksjon* og *assimilasjon*. Man bruker estimatet fra tid $t = k - 1$ til å estimere den nye tilstanden ved tid $t = k$. Så sjekker man dette opp i mot hva man observerer (måler) og oppdaterer modellen slik at neste estimat forhåpentligvis er bedre. Dette kan oppfattes som et system med en innebygd treghet som gjør den robust mot støy.

Prediksjon

$$\tilde{\mathcal{X}}_k - \overline{\mathcal{X}} = A(\hat{\mathcal{X}}_{k-1} - \overline{\mathcal{X}})$$
$$\tilde{\mathcal{P}}_k = A\mathcal{P}_{k-1}A^T + BB^T$$

Assimilasjon (by Kalman gain)

$$\begin{aligned} \mathcal{K}_k &= \tilde{\mathcal{P}}_k H^T \left(S_k H \tilde{\mathcal{P}}_k H^T + I \right)^{-1} \\ \hat{\mathcal{X}}_k &= \tilde{\mathcal{X}}_k + \mathcal{K}_k \mathbf{Z}_k \\ \mathcal{P}_k &= (I - \mathcal{K}_k S_k H)\tilde{\mathcal{P}}_k \end{aligned}$$



Figur 9: Eksempel på en normalfordeling og en multimodal fordeling.

der

$$H = \begin{pmatrix} 0 & I \end{pmatrix}$$

(I er en identitetsmatrise.)

Med Kalman-filteret på plass har man alt som trengs for å spore objekter i bildesekvenser. Visuell datamåling, en dynamisk modell og temporær filtrering.

2.2.8 Condensation

Bruken av Kalman-filteret er basert på antagelsen av at $p(\mathbf{X})$ er normalfordelt. Men det er den nødvendigvis ikke i naturlige data. Dersom det er *clutter* i bildet, som vil si rot / uorden, vil fordelingen være mer komplisert. Hvis man for eksempel skal spore en bil som kjører forbi andre biler, som står parkert, hvordan kan man da forhindre at bilene ikke blandes sammen. Det vil si at sporingssplinen låser seg på en av de parkerte bilene isteden for å følge den som beveger seg? Frem til nå har $p(\mathbf{X})$ hatt forventningsverdi $\overline{\mathbf{X}}$, som kan oppfattes som at den bare opererer med én hypotese. Siden det er flere biler i bildet, eller andre objekter som kan forveksles med biler, må man vedlikeholde flere hypoteser for å oppnå ønskelig robusthet. Så, isteden for en normalfordeling, vil man ha en fordeling med flere lokale maksima der hvert maksima representerer en hypotese. Rent praktisk vil det si at alle biler i en scene vil være representert med hvert sitt lokale maksima i en flerdimensjonal fordeling. Fordelingen vil altså være multimodal (se figur 9).

CONDENSATION (CONDitional DENSity propagATION) [27][28] er en mer generell metode enn Kalman-filteret som kan håndtere multimodale fordelinger. Enkelt sagt går den ut på å sample flere tilfeldige splines fra en sannsynlighetsfordeling, sjekke hvor godt disse passer med virkeligheten, og så oppdatere fordelingen slik at bedre splines vil samples neste gang. Dette skjer i hver iterasjon for hvert eneste bilde i en videosekvens. I utgangspunktet kan man sample over hele bildeflaten med alle mulige

former, men på grunn av begrenset prosessorkapasitet må samplingen begrenses. Dersom man bare skal spore ett objekt og man vet utgangspunktet vil det være ganske greit siden man da kan opprettholde ganske stramme fordelinger. (Dvs; færre muligheter - større andel kan testes.)

Siden apriori- og observasjonsfordelingen nødvendigvis ikke lenger er Gaussiske finnes det ikke en enkel måte å beregne aposteriorifordelingen. *Factored sampling* er en metode for å *estimere* aposteriorifordelingen. Den genererer først et sett med samples $\{\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}\}$ fra apriorifordelingen $p(\mathbf{X})$ og så en indeks $n \in \{1, \ldots, N\}$ med sannsynlighet $\pi^{(n)}$;

$$\pi^{(n)} = \frac{p_z(\mathbf{s}^{(n)})}{\sum_{j=1}^{N} p_z(\mathbf{s}^{(j)})}$$

der

$$p_z(\mathbf{s}) = p(\mathbf{Z}|\mathbf{X} = \mathbf{s})$$

er den betingede observasjonsfordelingen. Verdien $\tilde{\mathbf{X}} = \mathbf{s}^{(n)}$ vil ha en fordeling som ligner aposteriorifordelingen $p(\mathbf{X}|\mathbf{Z})$. Den vil altså være et estimat av $p(\mathbf{X}|\mathbf{Z})$, betegnet $\tilde{p}(\mathbf{X}|\mathbf{Z})$, som vil bli bedre og bedre ettersom $N \to \infty$. Gjennomsnitt og varians kan bestemmes direkte ut i fra prøvene $\left\{\mathbf{s}^{(n)}\right\}$:

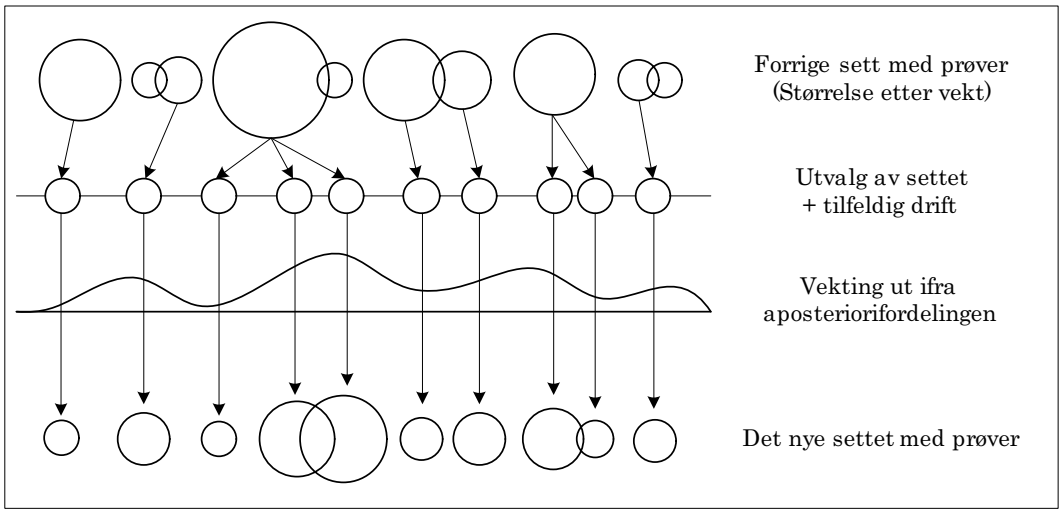
$$\mathcal{E}[g(\mathbf{X})|\mathbf{Z}] = \frac{\sum_{n=1}^{N} g(\mathbf{s}^{(n)}) p_z(\mathbf{s}^{(n)})}{\sum_{j=1}^{N} p_z(\mathbf{s}^{(j)})}$$

Gjennomsnittet beregnes om man setter $g(\mathbf{X}) = \mathbf{X}$, og variansen dersom $g(\mathbf{X}) = \mathbf{X}\mathbf{X}^T$.

CONDENSATION baseres på factored sampling. Factored sampling er i utgangspunktet begrenset til enkeltbilder, så Condensate er bare en videreutvikling for bruk på bildesekvenser. Den minner for så vidt på Kalmanfilteret, i og med at den består av lignende faser, men den er egentlig en enklere og mindre kostbar algoritme.

Prosessen ved hver tidsenhet er en uavhengig iterasjon av factored sampling, slik at resultatet av en iterasjon vil være et vektet og tidsbestemt sett med prøver (samples), uttrykt: $\{(\mathbf{s}_k^{(n)}, \pi_k^{(n)}), n = 1, \ldots, N\}$. Disse representerer altså den betingede fordelingen $p(\mathcal{X}_k|\underline{\mathbf{Z}}_k)$ ved tid $t = k$. Prosessen trenger apriorifordelingen $p(\mathcal{X}_k|\underline{\mathbf{Z}}_{k-1})$, som kan utledes fra resultatet fra forrige iterasjon: $\{(\mathbf{s}_{k-1}^{(n)}, \pi_{k-1}^{(n)}), n = 1, \ldots, N\}$. Første steg er å trekke N prøver fra $\mathbf{s}_{k-1}^{(n)}$ med sannsynlighet $\pi_{k-1}^{(n)}$. De elementene med høy sannsynlighet kan velges flere ganger, mens de med lav sannsynlighet velges kanskje ikke i det hele tatt. Siden det velges akkurat N prøver holdes settets størrelse konstant slik at algoritmen har en garantert kjøretid.

Neste steg er å utsette de valgte prøvene for prediksjon. Dvs. å bestemme $\{\mathbf{s}_k^{(n)}\}$ ved å trekke prøver fra fordelingen $p(\mathcal{X}_k|\mathcal{X}_{k-1})$ (se ligning 2.12).



Figur 10: CONDENSATION illustrert.

Det siste steget er å bestemme vektene $\{\pi_k^{(n)}\}$ ut i fra observasjonsfordelingen $p(\mathbf{Z}_k|\mathcal{X}_k)$. Dette gir $\{(\mathbf{s}_k^{(n)}, \pi_k^{(n)}), n = 1, \ldots, N\}$ for tid t_k. Algoritmen er presentert i egen ramme.

[Liten kommentar. Condensation minner egentlig om *genetisk programmering*. Det opprettholdes en populasjon (sett) over flere iterasjoner som utsettes for utvalg og tilfeldig påvirkning (permutasjon). Det som mangler er parring av forskjellige individer (for å blande egenskaper), noe som for så vidt er ganske viktig i genetisk programmering.]

2.2.9 ICondensation

Som sagt opprettholder CONDENSATION flere hypoteser om mulige objekter i et bilde og dette betyr i utgangspunktet at den kan spore flere objekter samtidig. Vanligvis sporer man bare ett objekt, som vil si at man enten velger den $\mathbf{s}^{(n)}$ med størst vekt $\pi^{(n)}$, eller velger det veide gjennomsnitt av hele settet med prøver. Dersom man vil spore flere objekter samtidig kan man oppnå dette ved å velge flere av prøvene. Altså de prøvene med de største vektene. Problemet er imidlertid at settet med prøver er begrenset pga. prosessorkapasitet. Dvs. at sannsynlighetsfordelingen er strammet inn slik at antall prøver i et sett ikke behøver å være alt for stort. Anta at man er godt i gang med å spore en bil. Da vil sannsynlighetsfordelingen være slik at alle splines man genererer vil ligge i nærheten av det stedet man så bilen sist (siste bildet). Det kan hende at noen av prøvene vil ligge et stykke unna, men det er det liten sannsynlighet for. Dette betyr at dersom det kommer en ny bil inn i bildet vil dette sannsynligvis ikke oppdages i det hele tatt. Så for å kunne spore flere biler samtidig må sø-

Algoritme CONDENSATION

Fra det gamle (forrige) settet $\left\{\mathbf{s}_{k-1}^{(n)}, \pi_{k-1}^{(n)}, c_{k-1}^{(n)}, n = 1, \ldots, N\right\}$ ved tid t_{k-1}, beregn det nye settet $\left\{\mathbf{s}_{k}^{(n)}, \pi_{k}^{(n)}, c_{k}^{(n)}, n = 1, \ldots, N\right\}$ for tid t_k. Beregn den n-te av N slik:

1. **Select** $\mathbf{s}_t'^{(n)}$:
   (a) generer ett tilfeldig tall $r \in [0, 1]$, fra en uniform fordeling.
   (b) bestem (by binary subdivisjon) den minste j slik at $c_{k-1}^{(j)} \geq r$.
   (c) sett $\mathbf{s}_k'^{(n)} = \mathbf{s}_{k-1}^{(j)}$
2. **Predict** ved sampling fra $p(\mathcal{X}_k | \mathcal{X}_{k-1} = \mathbf{s}_k'^{(n)})$ for å bestemme hver $\mathbf{s}_k^{(n)}$. Dersom dynamikken er bestemt av en lineær autoregressiv prosess, kan det nye eksemplaret beregnes som : $\mathbf{s}_k^{(n)} = A\mathbf{s}_k'^{(n)} + (I - A)\overline{\mathcal{X}} + B\mathbf{w}_k^{(n)}$ der $\mathbf{w}_k^{(n)}$ er en vektor av uniformfordelte tilfeldige variabler, og BB^T er kovariansen til prosesstøyen.
3. **Measure** og veie den nye posisjonen i termer av målte trekk $\mathbf{Z}_k$:

   $$\pi_t^{(n)} = p(\mathbf{Z}_t | \mathcal{X}_t = \mathbf{s}_t^{(n)})$$

   normaliser slik at $\sum_n \pi_k^{(n)} = 1$ og lagre det sammen med den kumulative sannsynligheten som $(\mathbf{s}_k^{(n)}, \pi_k^{(n)}, c_k^{(n)})$ hvor

   $$\begin{aligned} c_k^{(0)} &= 0, \\ c_k^{(n)} &= c_k^{(n-1)} + \pi_k^{(n)} \quad (n = 1, \ldots, N). \end{aligned}$$

   Når N utvalg har blitt bestemt: **estimer**, dersom ønskelig, sporede posisjoner ved tid t_k som

   $$\mathcal{E}[h(\mathbf{X}_k)] = \sum \pi_k^{(n)} h(\mathbf{s}_k^{(n)})$$

   og bestem, for eksempel, den gjennomsnittlige posisjonen med $h(\mathbf{X}) = \mathbf{X}$.

ket utvides uten at det blir for kostbart. Hvordan kan man oppnå dette? Det er her *importance sampling* kommer inn i bildet. Importance sampling er generell teknikk som er utviklet for å forbedre effektiviteten til factored sampling. Den kan benyttes når det finnes ekstra kunnskap i form av en importance-funksjon $g(\mathbf{X})$ som beskriver hvilke områder av tilstandsdomenet [!] som forteller mest om aposteriorifordelingen. Poenget er altså å generere prøver $\mathrm{s}^{(n)}$ i disse områdene ut i fra $g(\mathbf{X})$ isteden for aprioriforfordelingen $p(\mathbf{X})$. Det ønskelige er å unngå og generere prøver med lav vekt siden de er bortkastet, med tanke på at de ikke bidrar særlig i estimeringen av aposteriorifordelingen.

CONDENSATION kan benytte seg av importance sampling og kalles da for ICONDENSATION [29]. Importance-funksjonen betegnes $g_k(\mathcal{X}_k)$ og vektene er:

$$\pi_k^{(n)} = \frac{f_k(\mathbf{s}_k^{(n)})}{g_k(\mathbf{s}_k^{(n)})} p(\mathbf{Z}_k|\mathcal{X}_k = s_k^{(n)}) \tag{2.13}$$

der

$$\begin{aligned} f_k(\mathbf{s}_k^{(n)}) &\equiv \tilde{p}(\mathcal{X}_k = \mathbf{s}_k^{(n)}|\underline{\mathbf{Z}}_{k-1}) \\ &= \textstyle\sum_{j=1}^{N} \pi_{k-1}^{(n)} p(\mathcal{X}_k = \mathbf{s}_k^{(n)}|\mathcal{X}_{k-1} = \mathbf{s}_{k-1}^{(n)}) \end{aligned} \tag{2.14}$$

er det samme estimatet som i CONDENSATION. Poenget med brøken i ligning 2.13 er å bevare informasjonen om koherensen som finnes i den dynamiske modellen. Så selv om g_k er tatt med i beregningen vil distribusjonen fortsatt være et estimat av aposteriorifordelingen. Importance sampling skal bare forbedre effektiviteten, den endrer ikke på den probabilistiske modellen.

I praksis vil funksjonen g_k være et resultat av en mangelfull (ikke optimal) målingsprosess og derfor er det mulig at den utelater noen topper i $p(\mathbf{Z}_k|\mathcal{X}_k)$. Derfor kan det være lurt å generere noen prøver med standard factored sampling, og noen med importance sampling. Så hvis en av de feiler vil algoritmen fortsatt virke som helhet. Det er også lurt å inkludere en mekanisme for reinitialisering av den dynamiske modellen. Dvs. å godta reposisjonering av det sporede objekt uten å ta hensyn til historien $\underline{\mathbf{Z}}_k$. På denne måten kan trackeren låse seg på objekter som kommer inn i scenen/bildet, eller gjenfinne objekter som har vært skjult på grunn av andre objekter i scenen eller pga støy / målingsfeil.

I ICONDENSATION, se figur, blir konstantene q og r brukt til å velge samplingsmetode ut i fra et tilfeldig tall α. På grunn av dette vil algoritmen være både robust og agil. Dersom man ønsker å spore flere objekter samtidig må man gjøre litt tolkning av settet $\{(\mathrm{s}_k^{(n)}, \pi_k^{(n)})\}$. Om det er to objekter i bildet vil det komme til uttrykk som to grupperinger av prøver sentrert rundt hvert sitt punkt i bildeplanet. Man må altså identifisere de viktigste samlingene (spatialt-sett) av prøver og beregne gjennomsnittet av

de for å kunne spore flere objekter. Dersom man greier å gjenfinne de samme gruppene i suksessive bilder har man en multiobjektsporingsalgoritme!

Det som gjenstår da er: Hva er g_k? Det forklares i neste kapittel.

2.3 Dynamic Time Warping

Dynamic Time Warping (DTW) [30] er en metode for å beregne distansen mellom to tidsserier. En tidsserie er en liste med data fra målinger foretatt med bestemte mellomrom. Problemet kan defineres slik: Gitt to tidsserier X og Y, med lengde $|X|$ og $|Y|$,

$$\begin{aligned} X &= x_1, x_2, \ldots, x_i, \ldots, x_{|X|} \\ Y &= y_1, y_2, \ldots, y_j, \ldots, y_{|Y|} \end{aligned}$$

beregn en *warp path* W,

$$W = w_1, w_2, \ldots, w_K \quad \max(|X|, |Y|) \leq K < |X| + |Y|$$

der K er lengden av W og element nummer k er

$$w_k = (i, j)$$

der i er en indeks i tidsserien X og j er en indeks i tidsserien Y. To krav: W må starte i begynnelsen av begge tidsserier, dvs. $w_1 = (1, 1)$, og slutte der begge tidsserier ender, dvs. $w_k = (|X|, |Y|)$. Dette betyr at alle indekser i begge tidsserier vil bli tatt med i beregningen. Det andre kravet er at i og j øker monotont:

$$w_k = (i, j), w_{k+1} = (i', j') \quad i \leq i' \leq i + 1, j \leq j' \leq j + 1$$

En warp path er altså en mapping mellom elementene i X og elementene i Y. Den optimale warp path er den med minst distanse, der distansen er definert som

$$d(W) = \sum_{k=1}^{k=K} d(w_{ki}, w_{kj})$$

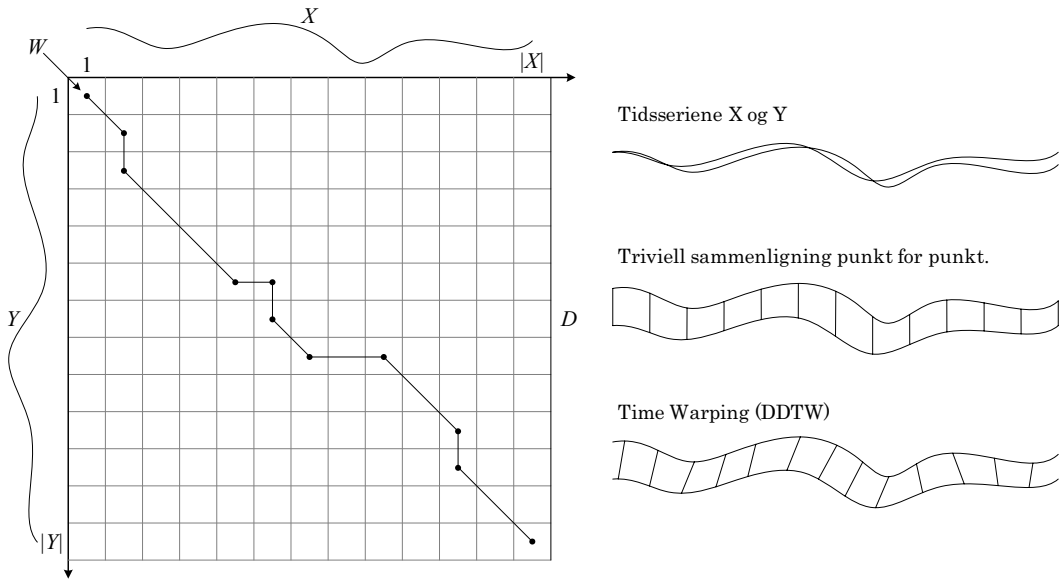
og $d(w_{ki}, w_{kj})$ er distansen mellom $x_{w_{ki}}$ og $y_{w_{kj}}$. (Det kan for eksempel være euklidsk distanse $(x_{w_{ki}} - y_{w_{kj}})^2$.)

For å beregne den optimale W brukes dynamisk programmering (derav navnet). Man genererer en todimensjonal kostmatrise D (se figur 11), med størrelse $|X| \times |Y|$, der verdien $D(i, j)$ er den billigste warp path som er mulig for tidsseriene $X' = x_1, \ldots, x_i$ og $Y' = x_1, \ldots, x_j$. $D(|X|, |Y|)$ vil derfor være den optimale warp path mellom X og Y. Alle celler i kostmatrisen må

Algoritme ICONDENSATION

Fra det gamle (forrige) settet $\left\{\mathbf{s}_{k-1}^{(n)}, \pi_{k-1}^{(n)}, n = 1, \ldots, N\right\}$ ved tid t_{k-1}, beregn det nye settet $\left\{\mathbf{s}_{k}^{(n)}, \pi_{k}^{(n)}, c_{k}^{(n)}, n = 1, \ldots, N\right\}$ for tid t_k. Importancefunksjonen $g_k(\mathcal{X}_k)$ antas for å være bestemt allerede. Sample den n-te av N slik:

1. **Velg** en samplingsmetode ved å trekke ett tilfeldig tall $\alpha \in [0, 1)$ fra en uniform fordeling
2. **Sample** fra den estimerte fordelingen $\tilde{p}(\mathcal{X}|\underline{\mathbf{Z}})$ slik:
   (a) **If** $\alpha < q$ bruk den initiale aprioriforde lingen. Bestem $\mathbf{s}_k^{(n)}$ ved å trekke fra $g_k(\mathcal{X}_k)$ og sett *the importance correction factor* $\lambda_k^{(n)} = 1$.
   (b) **If** $q \leq \alpha < q + r$ bruk importance sampling. Bestem $\mathbf{s}_k^{(n)}$ ved å trekke fra $g_k(\mathcal{X}_k)$ og sett $\lambda_k^{(n)} = f_k(\mathbf{s}_k^{(n)})/g_k(\mathbf{s}_k^{(n)})$ der
   $$f_k(\mathbf{s}_k^{(n)}) = \sum_{j=1}^{N} \pi_{k-1}^{(n)} p(\mathcal{X}_k = \mathbf{s}_k^{(n)} | \mathcal{X}_{k-1} = \mathbf{s}_{k-1}^{(n)})$$
   (c) **If** $q \geq q + r$ bruk standard CONDENSATION. Velg en $\mathbf{s}_{k-1}^{(i)}$ med sannsynlighet $\pi_{k-1}^{(i)}$, og så $\mathbf{s}_k^{(n)}$ ved å sample fra $p(\mathcal{X}_k | \mathcal{X}_{k-1} = \mathbf{s}_{k-1}^{(i)})$ og sett $\lambda_k^{(n)} = 1$.
3. **Measure** og veie den nye posisjonen ut i fra $\mathbf{Z}_k$ og *the importance correction factor term*:
   $$\pi_t^{(n)} = \lambda_k^{(n)} p(\mathbf{Z}_t | \mathcal{X}_t = \mathbf{s}_t^{(n)})$$
   normaliser slik at $\sum_n \pi_k^{(n)} = 1$ og lagre som $\{(\mathbf{s}_k^{(n)}, \pi_k^{(n)})\}$.



Figur 11: Eksempel på D. (Ikke basert på ekte data.)

beregnes og dette kan gjøres inkrementelt. Dvs. at man først kalkulerer $D(1,1)$ og så resten av cellene slik:

$$D(i,j) = \min \left\{ \begin{array}{l} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{array} \right\} + d(i,j)$$

En warp path til $D(i,j)$ må nødvendigvis gå via en av de tre naboene (i min) på grunn av de krav som er definert tidligere. Og siden disse naboene allerede er bestemt er det bare å velge den beste av dem og legge til den nye avstanden $d(i,j)$. For at dette skal virke må cellene i tabellen selvfølgelig beregnes i en bestemt rekkefølge. Man fyller ut kostmatrisen én kolonne om gangen fra topp til bunn, fra venstre til høyre. Når hele matrisen er ferdig vil den være et slags *høydekart* der $D(1,1)$ er det laveste punktet og den mest kostbare ruta er det høyeste punktet. Se figur 12 for eksempler. Dersom man kunne ha sluppet en klinkekule fra $D(|X|,|Y|)$ ville den rullet ned den optimale veien, dvs. den bratteste, og det er denne veien som er den beste warp path. Så for å bestemme den er det bare å gjøre et grådig søk fra $D(|X|,|Y|)$ der man hele tiden velger den naboen (N, NW, W) som har lavest verdi. Om man lagrer alle valgene ned til $D(1,1)$ i en liste vil W være inversen av denne.

DTW har en tids- og plasskompleksitet på $O(N^2)$ dersom $N = |X| = |Y|$. Hver celle i kostmatrisen må beregnes én gang og hver av disse beregningene gjøres i konstant tid. Dette blir utfordring etter hvert som problem-

Figur 12: Tre eksempler på ”høydekart”. Fra venstre: god korrelasjon, god korrelasjon i starten og dårlig på slutten, totalt ukorrelert. Alle er normalisert på verdien i $D(|X|, |Y|)$ (som vil si punktet nederst til høyre) og invertert.

størrelsene øker. Dersom distansene lagres som integers (32 bit) vil man trenge over 1 GB i minne allerede ved 16000 målinger (lengden av X og Y). Det finnes flere måter å begrense antall celler i D man må beregne. Hvis X og Y er like vil W være en rett diagonal linje fra $D(1, 1)$ til $D(|X|, |Y|)$. Det vil si at jo mer like to tidsserier er, jo nærmere vil W ligge diagonalen. Derfor er det mulig å utelate områdene rundt de motsatte hjørnene fordi det er lite sannsynlig at den optimale veien går der. Men siden det er alltid en mulighet for at den gjør nettopp det vil man ved enhver utelukking av celler risikere at den optimale veien ikke blir funnet.

En annen mulighet er beregne DTW på forkortede tidsserier, dvs. det samme som å undersample, men da blir resultatet mindre nøyaktig. Den siste muligheten er å bruke mer finurlige metoder til slå 'like' celler sammen slik at man bare behøver å beregne distansen én gang for alle celler i samme gruppe. Salvador og Chan [8] har foreslått en alternativ metode, kalt FastDTW, som benytter seg av en kombinasjon av de to første mulighetene. FastDTW har lineær tids og plasskompleksitet samtidig som den kan vise til svært lav feilrate.

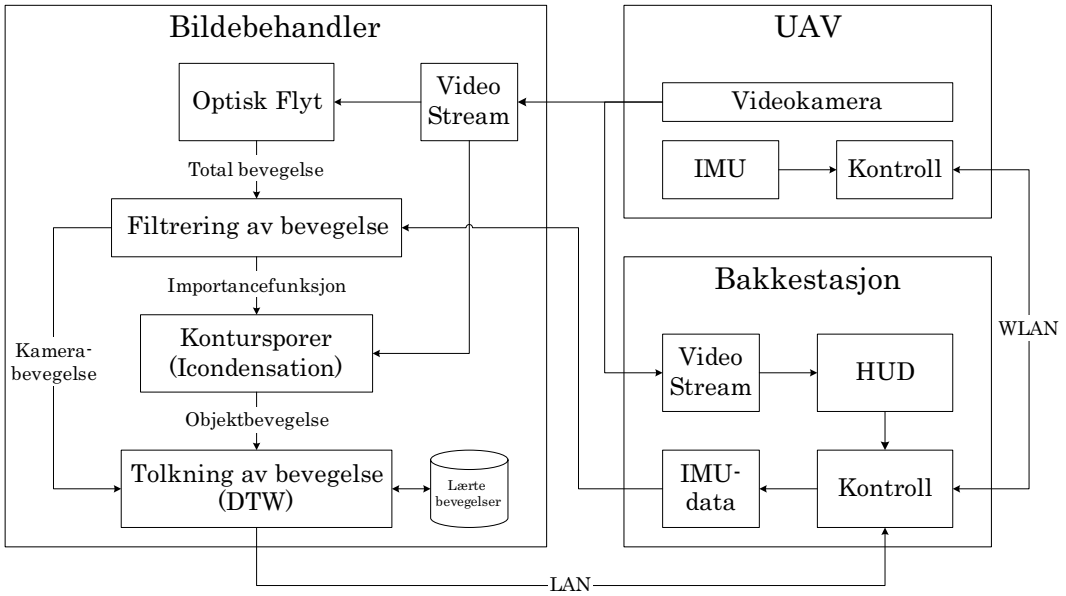
Spørsmålet til slutt er hvordan man definerer distansen $d(i, j)$. Den euklidske avstanden er nødvendigvis ikke alltid den beste. Dersom man bare sjekker tidsseriene punktvis mot hverandre kan det gi et svært mangelfullt syn på hva som passer sammen. En stigende trend i en tidsserie kan gi god korrelasjon mot en synkende trend i en annen tidsserie bare fordi grafene krysser hverandre på ett punkt. Dette er ikke ønskelig og for å unngå det kan man heller ta kvadratet av differansen mellom derivatene av kurvene på hvert punkt:

$$d(i, j) = (x'_i - y'_j)^2$$

der

$$x'_i = \frac{(x_i - x_{i-1}) + ((x_{i+1} - x_{i-1})/2)}{2}$$
$$y'_j = \frac{(y_j - y_{j-1}) + ((y_{j+1} - y_{j-1})/2)}{2}$$

Det er Keogh og Pazzani [7] som har foreslått dette og de kaller den alternative algoritmen for Derivative Dynamic Time Warping (DDTW).



Figur 13: Systemet i sin helhet

3 Applikasjon / System

Figur 13 viser hele systemet på en forenklet måte. Det består av tre hoveddeler; UAV (helikopteret), bakkestasjon og bildebehandler. Helikopteret er for så vidt helt uavhengig bakkestasjonen og bildebehandleren. Det vil si at det kan fly rundt på egenhånd ved å følge forhåndsbestemte ruter eller enkle algoritmer. Bakkestasjonen er der først og fremst for å kunne ta kontroll over helikopteret på en enkel måte. Helikopteret har et analogt videokamera ombord som sender videodata kontinuerlig slik at flyturen kan overvåkes fra bakken med et headup-display (HUD). Det gjør det også mulig å styre helikopteret manuelt med for eksempel en joystick eller et tastatur. Bakkestasjonen kontrollerer helikopteret via WLAN som betyr at rekkevidden er forholdsvis begrenset. (Video sendes uavhengig WLAN)

Videodataene som transmitteres kan også leses av bildebehandleren. Bildebehandleren kan egentlig være hvor som helst så lenge den er innenfor helikopterets eller bakkekontrollens rekkevidde. Den kan også være om bord selve helikopteret, men det er under forutsetning av at helikopteret har nok prosessorkapasitet. Uansett hvor bildebehandleren er; den mottar videodata og tolker den hovedsaklig i tre trinn; først bestemmer den kamerabevegelse, så objektbevegelse, og til slutt avgjør den om noen objekter har beveget seg ulovlig. Når den er ferdig med tolkningen sender den dataene til bakkestasjonen, eller direkte til helikopteret, som så kan ta en beslutning på hva som må gjøres.

Det viktigste arbeidet er selve sporingen av objekter og det blir gjort med ICONDENSATION. Den trenger en importancefunksjon for å fungere. En mulig måte er å bruke fargesegmentering (se [29]), men dette er ikke så enkelt siden biler ofte kommer i alle slags farger og nyanser. Dessuten finnes ofte de samme fargene naturlig i miljøet rundt. Derfor bruker jeg bevegelsesinformasjon i stedet, produsert av algoritmen for optisk flyt, som importancefunksjon. (Dvs. sample splines der det er bevegelse.) For det første; det er primært biler i bevegelse som skal spores. (Det er uansett ikke så vanskelig å spore objekter som står stille når man først har funnet dem.) For det andre; den optiske flyten er interessant uansett fordi den forteller om kamera- / helikopterbevegelsen som finner sted. Dersom man skal kunne bestemme om en bil kjører lovlig må man vite bevegelsen i forhold til bakken, ikke i forhold til helikopteret. Man må selvfølgelig regne med at helikopteret greier å holde seg forholdsvis rolig under overvåkningsfasen, men noe bevegelse vil det bli uansett.

Kameraets bevegelse kan også bestemmes ut i fra dataene som finnes i IMU'en ombord helikopteret. IMU står for *Inertial Measurement Unit* og er et lukket system som detekterer helikopterets stilling og bevegelse vha flere akselerometere og sensorer. Den er altså en svært viktig komponent og finnes på enhver UAV. Jeg er først og fremst opptatt av bildebehandlingen, men det er lurt å ta med informasjon fra andre kilder dersom den er relevant. I figuren er IMU-dataene tegnet inn som en parallell til dataene fra den optiske flyten. Det vil si at dataene fra de forskjellige kildene kan sjekkes opp mot hverandre slik at man oppnår en større sikkerhet. Data fra IMU'en kan også være med på å påvirke importancefunksjonen.

Når objektbevegelse relativ bakken er bestemt lagres de som grafer i en database under en opplæringsperiode. DTW brukes senere til å matche nye bevegelser mot de lovlige. Resultatet sendes til bakkestasjonen, eller helikopteret, som så kan ta en avgjørelse. Under forfølgninger vil DTW ikke være en del av systemet. Da vil posisjonen til objektet man følger bli sendt direkte til helikopteret. Utfordringen med en slik oppgave er å holde objektet man følger i sentrum av bildet. Ingen bevegelsestolkning av objektet vil være nødvendig. (Den vil optimalt være null.) Forfølgning er derfor en enklere oppgave for bildebehandleren.

3.1 Identifisering og tolkning av bevegelse

Begrensning av arbeid

Algoritmen OPTISKFLYT må kjøres for hvert eneste bilde i den kontinuerlige strømmen av bilder som helikopteret sender. Det vil si mellom 25 og 30 bilder i sekundet. Som sagt har algoritmen en tidskompleksitet som tilsvarer omtrent to konvolusjoner med 3×3 store masker. I tillegg må det utføres en *blurring* på forhånd for å få et godt resultat (se spatial coheren-

ce). Det vil si at arbeid tilsvarende tre konvolusjoner eller mer skal utføres på under (helst godt under) 40 millisekunder (25 fps). Og dersom algoritmen skal kjøres på samme prosessor som ICONDENSATION bør den helst bruke mindre enn halvparten av tiden slik at ICONDENSATION får minst like mye prosessorkraft til rådighet. Om arbeidet foregår på bakken er det selvfølgelig mulig å sette inn store ressurser, men det er uansett ønskelig å få et system som ikke er alt for tungt og som i teorien kan kjøres på 'lette' maskiner.

Det finnes flere måter å begrense arbeidet på. Selve algoritmen kan effektiviseres litt, noe som blir utdypet senere, men den aller viktigste framgangsmåten er å undersample i enten tid eller rom, eller begge deler. Er det nødvendig å bruke hvert eneste piksel i hvert eneste bilde i videoen? Oppløsning i tid er nok viktigere enn oppløsning i rom. Algoritmen i seg selv (differensial teknikk) fungerer i utgangspunktet ikke så godt for raske bevegelser. Om man halverer oppløsningen i tid, til f.eks. 15 fps, vil det bety at hastigheten til bevegelsene dobles. Derfor er det ønskelig å ha flest mulig bilder i sekundet. Den spatiale oppløsningen er derimot ikke så viktig. For det første; om et objekt beveger seg over 100 pixler med en hastighet på to pixler i sekundet vil en halvering i spatial oppløsning si en bevegelse på 50 pixler med en hastighet på ett piksel i sekundet. Altså til fordel for algoritmen. Bevegelsene blir tregere med hensyn til piksler per sekund. For det andre; det er allerede antatt at bevegelse er spatialt korrelert. Så om man slår sammen pixler vil ikke det føre til betydelige tap. For det tredje skal resultatet bare brukes til å *guide* ICONDENSATION. ICONDENSATION selv vil jobbe med de originale bildene i 'høy' oppløsning. Noen mulige oppløsninger (og antall punkt) er:

$$\begin{aligned} 640 \times 480 &= 307200 \\ 320 \times 240 &= 76800 \\ 160 \times 120 &= 19200 \\ 80 \times 60 &= 4800 \end{aligned}$$

160×120 vil nok egne seg best for beregning av den optiske flyten med tanke på ressursforbruk.

Håndtering av kamerastøy

En hver form for støy kan oppfattes som bevegelse. Som sagt må bildene glattes ut med et blur-filter, noe som tar en del av støyen, men finnes andre muligheter for å fjerne støy. Man kan for eksempel bruke medianfilter dersom støyen er sterk. Andre muligheter er å bruke maksimum- eller minimumsfilter (dilasjon og erosjon). Bildene behøver ikke å se pene ut, de skal

bare gi et godt grunnlag for beregningen av flyten. En annen framgangsmåte er å terskle hver enkelt piksel basert på forandringen av intensitet over tid. Det vil si å forkaste nye verdier dersom de er like de gamle (innen en viss grad) fordi de sannsynligvis er et resultat av støy. Dette strider mot utglattingen av bildene og paradoksalt nok er det den sterkeste støyen som overlever. Men det er en billig metode som gir et tilsynelatende godt resultat. Den kan i alle fall brukes for å fjerne litt støy slik at bildet blir roligere. Nå er ikke jevn og fin kamerastøy det aller største problemet. Selv om det helt klart vil føre til falsk bevegelse vil bevegelsesvektorene basert på støy være så tilfeldige og motstridende at de ikke vil ha noe stor innvirkning på den totale oppfatningen av flyten i bildet.

Det er også mulig å bruke gjennomsnittet av suksessive bilder, enten av to eller flere. Men dette tilsvarer litt grovt sagt en nedsampling i tid. Det hjelper godt mot støy, men de ekte bevegelsene vil bli raskere. Om man tar gjennomsnittet av TO suksessive bilder oppnår man altså bedre bilder på bekostning av halv oppløsning i tid. (Det er ikke helt sant siden alle bilder vil delta i to gjennomsnitt.) Ved svært rask bevegelse vil man oppleve *ghosting*, noe som ikke hjelper algoritmen i det hele tatt.

Den normale måten å behandle bildene er å beregne gjennomsnitt av både spatiale og temporære nabopixler. Ofte med en radius på 1,5 pixler. Da vil de spatiale naboene ha større innvirkning på gjennomsnittet enn de temporære naboene har.

Kostnader

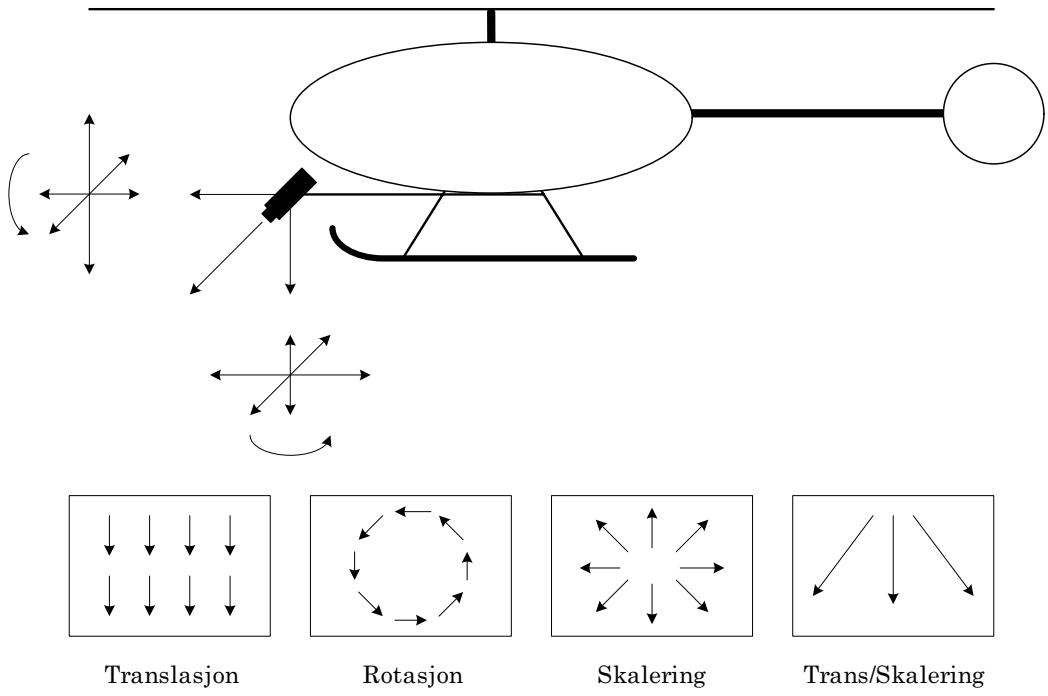
En vanlig blur algoritme har en tids- og plasskompleksitet på $O(n^2m^2)$ der n er høyde og bredde på bildet og m er høyde og bredde på masken. (Egentlig $O(n'^2m^2)$ der $n' = n - m$.) Min algoritme FASTAVERAGE tar vare på verdiene den beregner slik at den ikke trenger å gjøre samme utregning flere ganger. Tidskompleksiteten er $O(n(m^2 + nm))$, som man kan se i algoritmen. Dette sparer den en del tid på, i alle fall når det brukes store masker. Allerede ved 5×5 store masker sparer man 18 addisjoner i n^2 løkka i forhold til den vanlige fremgangsmåten ($m^2 - m - 2$). Ulempen er at man ikke kan vekte naboskap forskjellig. Men dette er heller ikke nødvendig siden algoritmen bare skal bare skal brukes til klargjøre bildene for beregning av den optiske flyten.

Den samme metoden kan brukes til å spare utregning med OPTICALFLOW. Men gevinsten blir ikke like betydelig. Laplace-operasjonen kan det ikke spares noe på, siden den nødvendigvis må være vektet. Ved beregningen av de partielle deriverte sparer man 6 addisjoner og 6 subtraksjoner av totalt 35 +- operasjoner (der tallene blir lest fra hovedminnet). Dette er ikke så betydelig men det merkes. Problemet kan ikke skaleres slik som med gjennomsnittsfiltrering. Det vil alltid være 23 operasjoner inne i n^2

Algoritme FASTAVERAGE

Gitt et bilde Y av størrelse $w \times h$. Gitt en grad g som definerer størrelsen på vindusfunksjonen. (Det vil si masken eller vektene. Alle vekter er like i denne algoritmen.)

1. Opprett et midlertidig bilde T av samme størrelse som Y.
2. Opprett en endimensjonal tabell A med lengde g og indeks a.
3. Sett $m = g/2$.
4. **For** alle i der $1 \leq i \leq w - g$
   (a) Sett $a = 0$ og $A = \{0, \ldots, 0\}$.
   (b) Beregn summen av alle verdier i vinduet med koordinatene $(i, 1)$. Lagre hver delsum (for hver rad) i A.
   (c) Lagre summen som $T_{i+m,1+m}$.
   (d) **For** alle j der $2 \leq j \leq h - g$
      i. Beregn summen av alle verdier i vinduet med koordinater (i, j) ved å trekke A_a fra $T_{i+m,j+m-1}$ og legge til summen av vindusraden $(i, j + g - 1)$, som også lagres i A_a. Inkrementer a med én så lenge den er mindre eller lik g, ellers sett den til 1.
      ii. Lagre summen som $T_{i+m,j+m}$.
5. Kopier alle verdiene i T til Y samtidig som de deles med g^2.



Figur 14: Kameravinkel og transformasjoner

løkka.

Identifisering av kamerabevegelse
Når den optiske flyten er beregnet må den tolkes. Det første man gjør er å identifisere kamerabevegelsen slik at den egentlige bevegelsen som finner sted kan bestemmes. Hovedsaklig vil man oppleve translasjon, rotasjon og skalering på grunn av at kameraet flytter seg. Anta at kameraet filmer rett ned, normalt på bakken; om helikopteret beveger seg rett langs en linje i det horisontale planet vil det oppfattes som translasjon. Om helikopteret henger stille i luften og snur rundt vil det oppfattes som rotasjon. Skalering (eller zoom) vil oppfattes dersom helikopteret øker eller minker høyden over bakken. Dersom kameraet filmer rett frem, horisontalt, blir det litt annerledes. Rett bevegelse i det horisontale planet blir skalering, rotasjon rundt den vertikale akse og forandring i høyde over bakken blir translasjon. Mest sannsynlig vil kameravinkelen være en mellomting av disse ytterkantene. Da vil man oppleve blandinger av de tre transformasjonene.

I en overvåkningssituasjon vil det være små kamerabevegelser og disse vil være hovedsaklig av translativ art. Når man forfølger objekter vil man oppleve alle de tre transformasjonene, men den translative vil som

oftest være den viktigste. Dersom man flyr rett over bilen man forfølger vil man oppleve hovedsaklig translasjon. Litt rotasjon vil det også være, men så lenge helikopteret holder en viss hastighet vil denne forsvinne i all translasjonen. Skalering vil man som sagt bare oppleve om høyden over bakken forandres. Dette vil også forsvinne i all translasjonen så lenge forandringen i høyde skjer gradvis. Poenget mitt er at man kommer langt med ganske enkel tolkning av fordelingen av alle fartsvektorer.

Den best representerte fartsvektoren, dvs. modus, vil være null dersom det ikke finnes noe kamerabevegelse. Den vil også være null så lenge det bare er små lokale bevegelser i bildet. Dersom det er translasjon vil modus uttrykke translasjonen i seg selv. Det vil si at den bevegelsen kan fjernes ved å trekke modus fra alle fartsvektorer. Men modus vil også ha en verdi ved rotasjon og skalering, men da vil den ikke utrykke noe fornuftig, og man kan heller ikke fjerne slik bevegelse enkelt. For å skille situasjonene kan man sammenligne modus med gjennomsnittet. Dersom modus og gjennomsnitt er forholdsvis like vil det si translasjon, dersom ikke vil det si annen mer kompleks bevegelse. Man kan også bruke standardavviket for å skape mer sikkerhet.

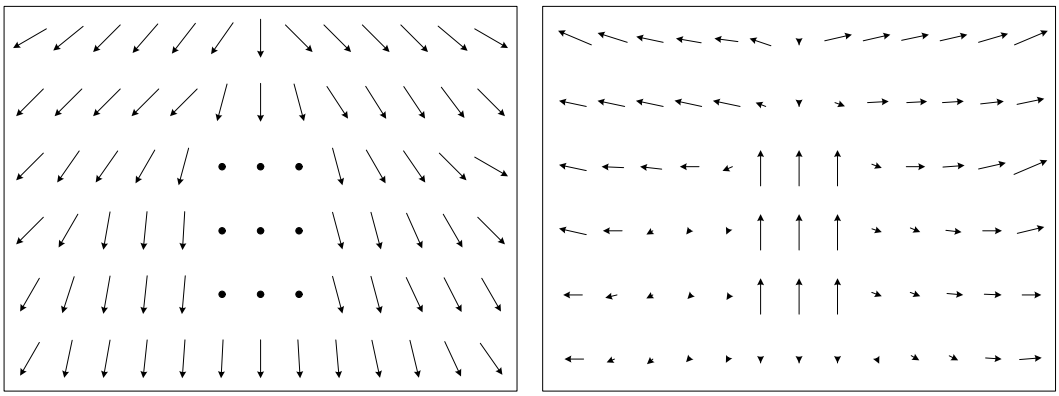
Generering av importancefunksjon

Når man har fjernet kamerabevegelse kan man lage ”kart” som vil være grunnlaget for (eller vil være i seg selv) importancefunksjonen. Dersom man beregner lengden av alle fartsvektorer, plotter disse i en todimensjonal tabell og normaliserer med den største verdien vil man få et bilde der 1 indikerer den største bevegelsen og 0 den minste. Så kan man terskle kartet med en bestemt verdi, for eksempel 0.5, slik at det vil bli segmentert i interessante og uinteressante områder. Man kan bruke erosjon og dilasjon for å fjerne de minste områdene som sannsynligvis bare er resultat av støy. Til slutt gir man alle avskilte hvite områder hver sin identifikator og sorterer de ut i fra størrelse og / eller styrke (dersom man har tatt vare på gradientinformasjon). Importancefunksjonen vil da definere disse områdene som viktige steder der *Icondensation*-algoritmen bør sample splines.

(For å skille de interessante områdene fra hverandre kan man bruke flood-fill. Alle pixler i det binære bildet flood-filles med distinktive identifikatorer. Dersom et piksel er 1 setter man en id (f.eks. 2) og sjekker alle nabopixlene. Dersom et piksel er ulik 1 gjør man ingen ting.)

Ett eksempel

La oss si at helikopteret følger en bil og at kameraet har en vinkel på 45° i forhold til den horisontale (eller vertikale) aksen. Anta også at helikopteret har nesa rett fram parallelt med fartsretningen. Da vil man oppleve en blanding av translasjon og skalering i bildedataene. Gjennomsnittsvekto-



Figur 15: Vektorkart ved sporing av bil. Kameravinkelen er $45°$. Til høyre er gjennomsnittsvektoren trukket fra vektorene til venstre.

ren vil helt riktig indikere en bevegelse rett ned i bildet (det vil si bevegelsen til bakken). Bilen vil ikke ha noen hastighet i forhold til helikopteret og derfor vil dens fartsvektorer være null. Dersom man nå trekker gjennomsnittsvektoren fra alle fartsvektorer vil man sitte igjen med fartsvektorene til bilen, som nå alle er negativen til gjennomsnittet, pluss en del vektorer med betydelige y-komponenter. Gjennomsnittet hadde ingen y-komponent og derfor vil alle de som har det 'overleve'. Dette er et problem fordi disse vil nå oppfattes som objektbevegelse. Se figur 15. Det er ikke så enkel å filtrere ut disse siden det er en spatial korrelasjon mellom dem. Men siden dette problemet helst vil oppstå i denne bestemte situasjonen kan man håndtere det eksplisitt. Man vet at fartsvektorene på bilen man følger vil ha en betydelig x-komponent. Derfor kan man bare finne de vektorene med de største x-komponentene og forkaste alle andre. Man kan også unngå dette problemet helt ved å bruke en stor vinkel (ift den horisontale aksen).

I situasjonen beskrevet er det også mulig å bare velge de minste fartsvektorene. Eventuelt kan man bruke CONDENSATION isteden for ICONDENSATION (dvs. ikke bruke importancefunksjon i det hele tatt). Man sporer bare ett objekt, man vet den initiale posisjonen (i alle fall om algoritmen startes manuelt), og objektet vil ha en svært enkel bevegelsesdynamikk. Utfordringen vil først og fremst ligge i kontrolleringen av selve helikopteret. Dersom bilen bremser opp vil det si at den beveger seg nedover i bildet mot den nedre kanten. Da må helikopteret bremse opp like fort for å kompensere. Oppgaven vil være å holde bilen midt i bildet hele tiden. Det er altså oppløsningen i tid og hvor raskt man greier å styre helikopteret som er avgjørende. Jo større avstand helikopteret har fra bilen, jo mer tid vil være til rådighet.

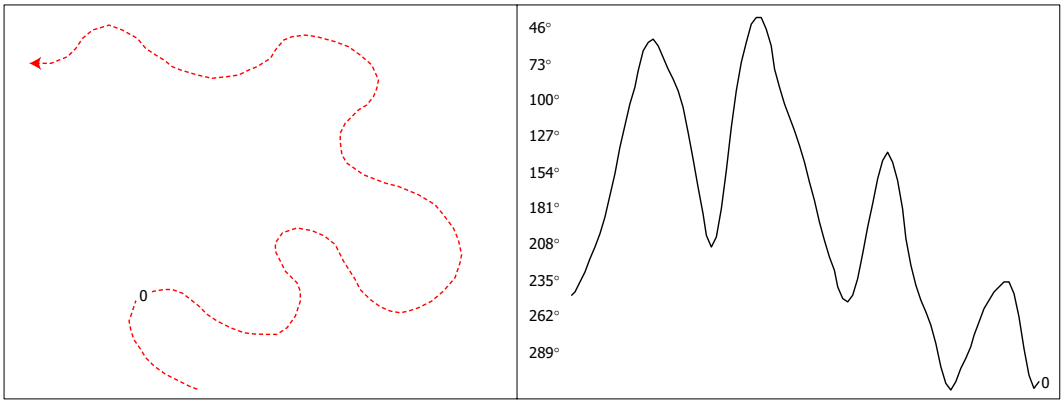
3.2 Identifisering av objekter og objektbevegelse

For å kunne spore objekter vha splines må man først definere alle splines som skal spores. Det vil si splines som representerer alle typer biler. Man må også definere frihetsgradene og den dynamiske modellen. Jo flere splines, frihetsgrader og mer avansert dynamisk modell, jo mer kostbar blir algoritmen. En måte å forenkle søkedomenet er å diskretisere problemet. ICONDENSATION sampler bare splines. Om den sampler fra en diskret samling eller fra en fordeling spiller ingen rolle. Poenget er at det er billigere å sample fra en diskret samling. Alle muligheter blir derfor definert på forhånd og lagret i en tabell. Det eneste som er viktig er å forsøke og skape en god lokal korrelasjon mellom hver mulighet slik at søket kan begrenses enkelt. (Dette vil si at man lagrer alle mulige splines i en tabell, splines av alle type bilfasonger, fra alle mulige retninger, på en slik måte at like splines ligger i nærheten av hverandre.) Størrelse (og selvfølgelig posisjon) er noe som beregnes under kjøring av algoritmen.

Når man har definert alle splines og har en importancefunksjon ligger alt til rette for sporing av objekter. Kostnaden til algoritmen er bestemt av antallet splines N som samples ved hver iterasjon. Jo flere splines som samples, jo bedre resultat og større kostnad. Det å spore flere objekter samtidig koster i utgangspunktet ikke mer enn å spore ett. Men siden det totale antallet splines som samples må fordeles på alle objektene som spores vil det allikevel ha betydning. Dersom man ikke øker N vil kvaliteten bli dårligere etter hvert som flere objekter skal spores. Dersom man øker N proporsjonalt med antall objekter, øker også prosessorforbruket proporsjonalt.

Man kan uansett ikke spore alt for mange biler samtidig. Hele tiden beregnes hvor godt hver spline som samples korrelerer med dataene i bildet. For at dette skal fungere godt må det være en viss størrelse på objektene man sporer. Dersom bilene blir svært små, dvs. mindre en et areal på rundt 1000 punkt, blir det svært vanskelig å spore med splines. Alle distinksjonene til splinen forsvinner etter hvert som objektet blir mindre. Siden det må være en viss størrelse på objektene begrenses også antallet objekter som kan være i bildet samtidig. Det begrenser også tolkningen av bevegelse siden det blir mindre plass til den.

For sporing av objekter er det som nevnt før enklest dersom kameraet filmer rett ned, normalt på bakken eller med minst mulig vinkel. Da forsvinner den tredje dimensjonen og man unngår de tyngste transformasjonene som er nødvendig ellers. Høyden over bakken (eller zoom) er også avgjørende for resultatet. Men grunnen til å bruke konturspоring er jo for å kunne spore under vanskelige forhold. Dersom man forenkler problemet tilstrekkelig kan man bare bruke den optiske flyten (dvs. importancefunksjonen).



Figur 16: Fra bevegelse (venstre) til graf (høyre).

3.3 Tolkning av objektbevegelse

For å kunne gjenkjenne bevegelser med DTW må man representere de som grafer. Målet er å produsere unike grafer slik at det er lett å skille forskjellige bevegelser fra hverandre. Det er først og fremst spatial ulikhet som må skilles, ikke hastighet, akselerasjon eller ventetider. Disse egenskapene er også interessante, men de kan sjekkes uavhengig DTW. Dersom man skiller de spatiale egenskapene fra de temporære vil man oppleve større frihetsgrad samtidig som søkedomenet blir mindre.

Jeg har derfor valgt å bruke retningen i grader til å lage bevegelsesgrafer. For hver posisjon som registreres beregnes en retningsvektor i forhold til den forrige posisjonen. Retningsvektoren plottes i et koordinatsystem der x-aksen er tid og y-aksen er grader. Dersom en posisjon er lik den forrige forkastes den. Det vil si at plottingen stopper opp når det sporede objekt stopper og fortsetter ikke før det er bevegelse igjen. På denne måten unngår man å skille mellom ventetider, som kan oppstå for eksempel i lyskryss og rundkjøringer. For å unngå skarpe kanter i overgangen fra $360°$ til $0°$ brukes heller relative grader. Det vil si at en overgang fra $355°$ til $5°$ blir $365°$. Når alle bevegelser er plottet, etter et bestemt antall grafer eller innen et bestemt tidsrom, kan alle grafene normaliseres i tid om nødvendig. På denne måten blir grafene uavhengige hastighet. En mulighet er å strekke grafene ut i tid slik at alle blir like lange som den lengste. Da kaster man i alle fall ikke bort data. For å glatte ut kurvene brukes et gjennomsnittsfilter.

Det finnes noen viktige poeng som er verdt å merke seg dersom man bruker retning som basis for grafene. For det første vil lineære bevegelser føre til grafer som har konstant y-verdi. Derfor vil y-verdien i seg selv være helt avgjørende for å skille slike bevegelser. Dersom man normaliserer i y-retning, som vil si amplituden til grafen, vil alle lineære bevegelser bli like.

Dette er ikke ønskelig. For det andre vil parallelle bevegelser ikke kunne skilles fordi retning er uavhengig posisjon når den først er beregnet. Det vil si at biler som kjører i samme retning, men i forskjellige filer, ikke skilles fra hverandre. Noe som nødvendigvis ikke er et problem. Man kunne alternativt plottet posisjonene direkte inn som bevegelsesgrafer. Men da må man finne en god måte blande x- og y-komponent av posisjonene sammen. Det kan gi en ganske streng tolkning av bevegelse.

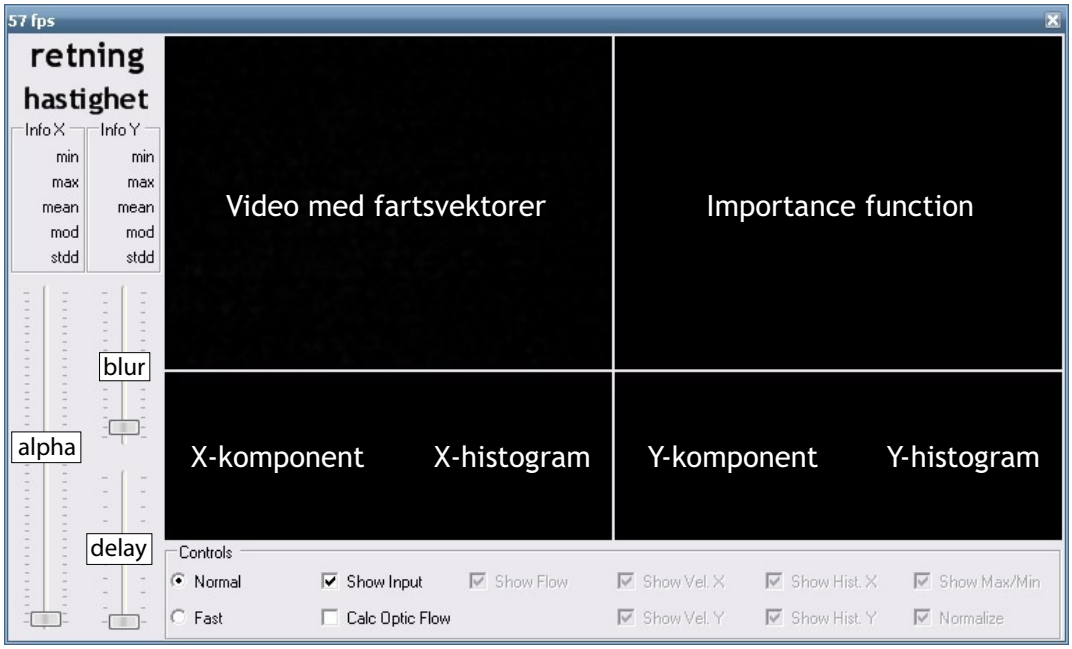
For å sjekke hastigheter bruker jeg en todimensjonal tabell som inneholder de maksimale hastighetene som er registrert på et hvert punkt. Det vil si at hver verdi i tabellen tilsvarer hastigheten på hvert punkt i bildeflaten. Ved å gjøre det på denne måten slipper man å beregne ekte hastigheter, som kan være vanskelig, og i tillegg vite hva som er fartsgrensen på enhver veistrekning. Men det forutsetter at hastighetene man registrerer under opplæring er lovlige. Under overvåkning reagerer man om det registreres en større hastighet på et punkt enn det som finnes i tabellen. På denne måten registreres det også om biler kjører av veien fordi der er fartsgrensen null.

Når man senere skal sjekke om en bevegelse er lovlig beregner man korrelasjonen mellom den nye bevegelsen og ALLE lovlige bevegelser med DTW. Lave tall vil si god korrelasjon. Det eneste som er interessant er hvor lite det minste tallet er. Dersom det er under en viss verdi kan man si at bevegelsen finnes og derfor er lovlig. Disse tallene vil være avhengige lengden på tidsseriene. Derfor er det lurt å normalisere korrelasjonstallene på denne lengden. Dersom man gjør dette vil verdiene være ganske universelle og bare én grenseverdi er nødvendig for alle forskjellige situasjoner.

Man må avgjøre om en bevegelse er lovlig eller ei før objektet som står for bevegelsen forsvinner ut av bildet, ellers er det ikke mulig å følge objektet. Det vil si at DTW må utføres før bevegelsesgrafen er ferdig. For å greie dette må man enten kutte de lærte grafene eller prøve å forutsi hvordan grafen som skal sjekkes vil se ut. Siden jeg tross alt bruker dynamisk programmering er det også mulig å beregne kostmatrisen samtidig som retningsgrafen bygges opp. Da er det ikke så enkelt å normalisere på tid siden man aldri vil vite hvor lange grafene vil bli. Men dette er kanskje ikke et problem.

For mitt problem er det nok ikke nødvendig med FastDTW. Dersom man antar at hver bil som skal spores er i bildet i 10 sekunder, og man gjør målinger av posisjon 30 ganger i sekundet, vil det si tidsserier med lengde på 300. Det er et ubetydelig tall, og 10 sekunder er nok mer enn det som vil være normalt. Selv om man skulle spore en bil i et helt minutt, som betyr en tidsserie på 1800 elementer, ville nok heller ikke *det* ha vært noe problem. Når det gjelder mål for distanse er euklidsk avstand mer enn godt nok av samme grunn (forholdsvis korte tidsserier). Og siden mye av informasjonen til grafene ligger i amplituden er heller ikke derivatet (DDTW)

det beste å bruke.



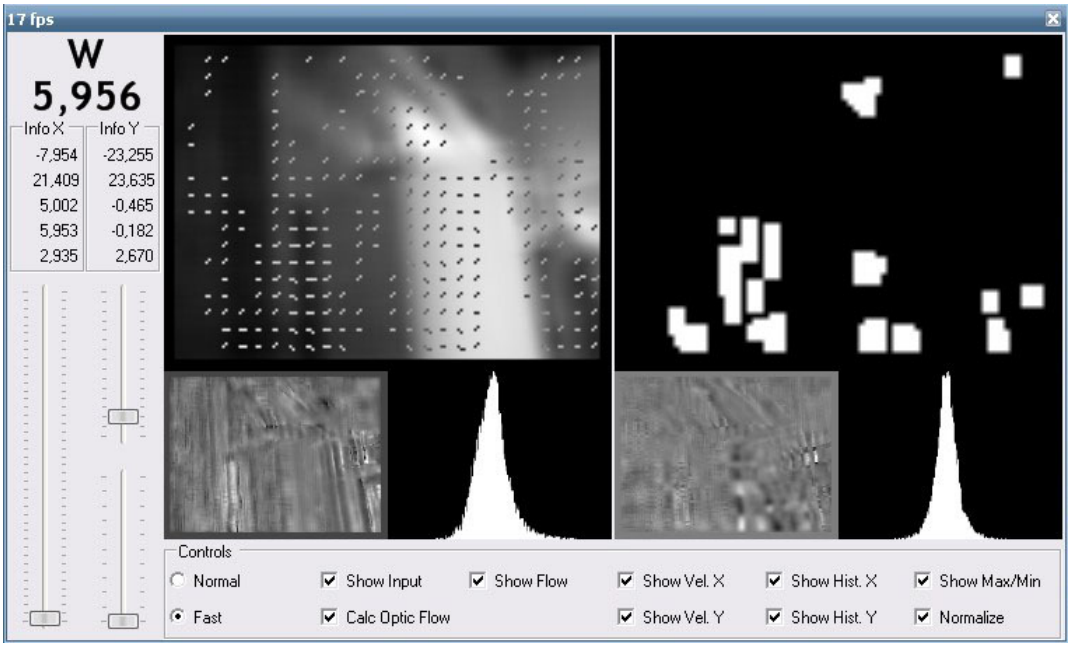
Figur 17: Forklaring

4 Eksperimentelle resultater

Helikopteret er ikke operativt ennå, og bildbehandleren ble heller ikke helt ferdig. Derfor er algoritmene testet hver for seg med ulike framgangsmåter. All testing er gjort på en maskin med 2.8 GHz P4, 1 GB RAM, WindowsXP og JDK1.5.0. Kameraet som ble brukt er et *Logitech QuickCam Sphere*.

4.1 Test av optisk flyt

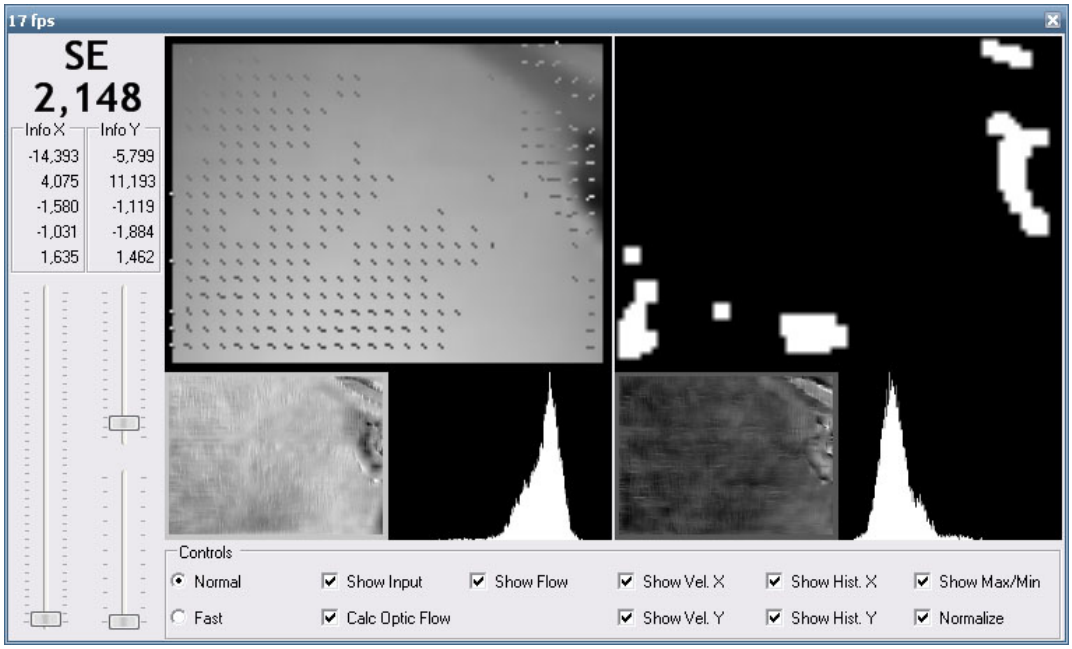
For å teste algoritmen som beregner den optiske flyten lagde jeg en enkel applikasjon som viser flere interessante tall og bilder av prosessen. Figur 17 er et bilde av dette programmet med forklaringer av hva de forskjellige komponentene i vinduet er. (Programmet er skrevet i C#). Applikasjonen kjører i utgangspunktet i sanntid, dvs. mer enn 25 bilder i sekundet, men på grunn av den grafiske oppdateringen som skjer etter hver iterasjon blir det litt mindre. Rundt 17-18 bilder i sekundet når oppløsningen er 160×120. Uten oppdatering av grafikken oppnår algoritmen 30 fps i *normal* modus og 34 i *fast*. (Grafikken er selvfølgelig ikke nødvendig i det endelige systemet.) Fast vil si at den bruker samme teknikk som FASTAVERAGE. Med en oppløsning på 320×240 greier algoritmen 7-8 fps uten grafikk. FASTAVERAGE gjør det mulig å blurre kraftig uten at det har stor betydning for ytelsen. Det er selvfølgelig ingen grunn til å blurre alt for hardt.



Figur 18: Kamerabevegelse

Figur 18 illustrerer *kamerabevegelse til venstre*. Man kan se tydelig at applikasjonen oppfatter dette helt riktig. Bevegelsesvektorene peker mot høyre og er forholdsvis like. Men en del vektorer mangler også. Det er fordi bevegelse oppdages først og fremst der det er kanter. Vektorene er også normaliserte, for at de ikke skal overlappe hverandre i bildet, og derfor forsvinner de korteste. Den sanne fordelingen av vektorene ser man i histogrammene. De er forholdsvis spisse og balanserte og underbygger tolkningen av bevegelsen. Gjennomsnittet til x-komponentene av vektorene er litt lavere enn modus pga. nullvektorene som trekker ned. Dette ser man også i histogrammet. I y-retning er modus ganske nært null, noe som stemmer med den ekte bevegelsen. Spredningen (max-min) er større enn for x-retning, men jevnt fordelt om null. (Merk at y aksen er opp-ned slik som i de fleste datasystemer.) Importance-funksjonen viser de områdene med de største hastighetene og er ikke så interessant i denne sammenhengen. Egentlig skal modusen trekkes fra før importancefunksjonen blir beregnet, men det er ikke gjort her.

Figur 19 viser det samme som forrige figur, men nå med kamerabevegelse ned til høyre. Kamerabevegelsen er også tregere og derfor er spredningen mindre. Forskjellen mellom gjennomsnitt og modus er mindre i x-retning og dette ser man også på vektorene. De er forholdsvis like. Etter en del tester av denne typen virker det som om algoritmen fungerer bra til å avgjøre kamerabevegelse. Jeg har svingt kameraet rundt og applikasjonen

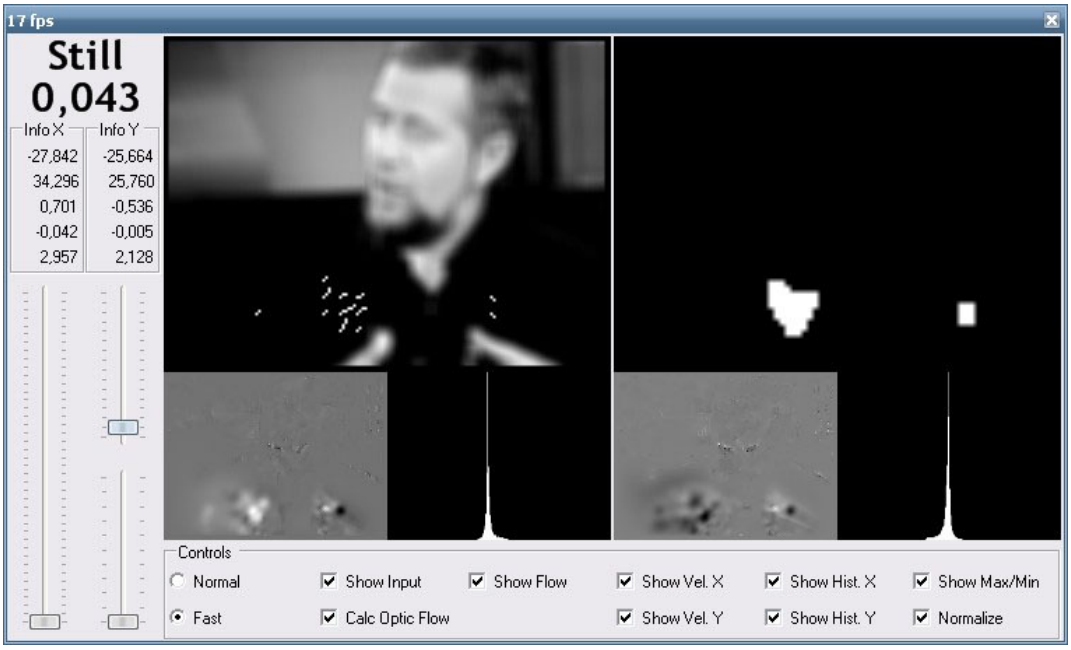


Figur 19: Kamerabevegelse

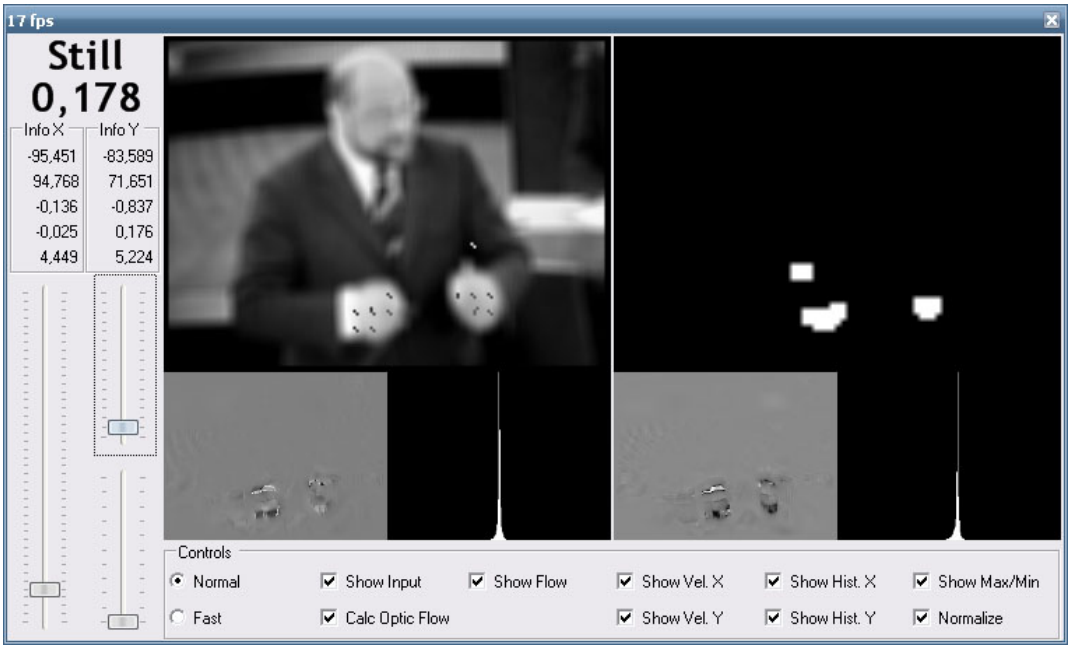
viser nesten alltid riktig retning. Men det finnes en viss treghet i systemet og denne tregheten blir større ettersom α økes (se kap 2). Treghet vil si å utrykke en bevegelse etter at kameraet har stoppet opp eller snudd retning. Men med en lav nok α greier algoritmen å oppdage dette ganske raskt. Fordelen med å bruke stor α er at algoritmen blir mer robust mot støy. Begge disse egenskapene kommer av at historien får større betydning etter som α økes.

I figur 20 følges et debattprogram med algoritmen. Kameraet står stille og det er først og fremst hendene og munnen til deltageren som beveger seg. Man ser tydelig i alle indikatorer (vektorer, importance-funk, hastighet x- og y-retning) hvor bevegelsene foregår. Bevegelsesvektorene og de hvite områdene i importancefunksjonen er faktisk ikke på hendene, men litt ovenfor. Dette er fordi at hendene beveger seg svært raskt. Det er områdene som hendene til stadighet beveger seg over som får betydelige utslag som vektorer. Hastighetskartene til hver komponent viser disse områdene mye klarere. Man kan også se at munnen/kjeven gir utslag i disse bildene, men ikke nok til at noen vektorer blir tegnet. Histogrammene er svært spisse og tynne. Modus er nær null (ingen kamerabevegelse), mens spredningen er stor (hastigheten til bevegelsen som finnes er forholdsvis stor). Figur 21 viser et eksempel på det samme, men nå med enda større hastigheter.

I figur 22 er det ikke like store hastigheter og derfor er spredningen



Figur 20: Hender som beveger seg.



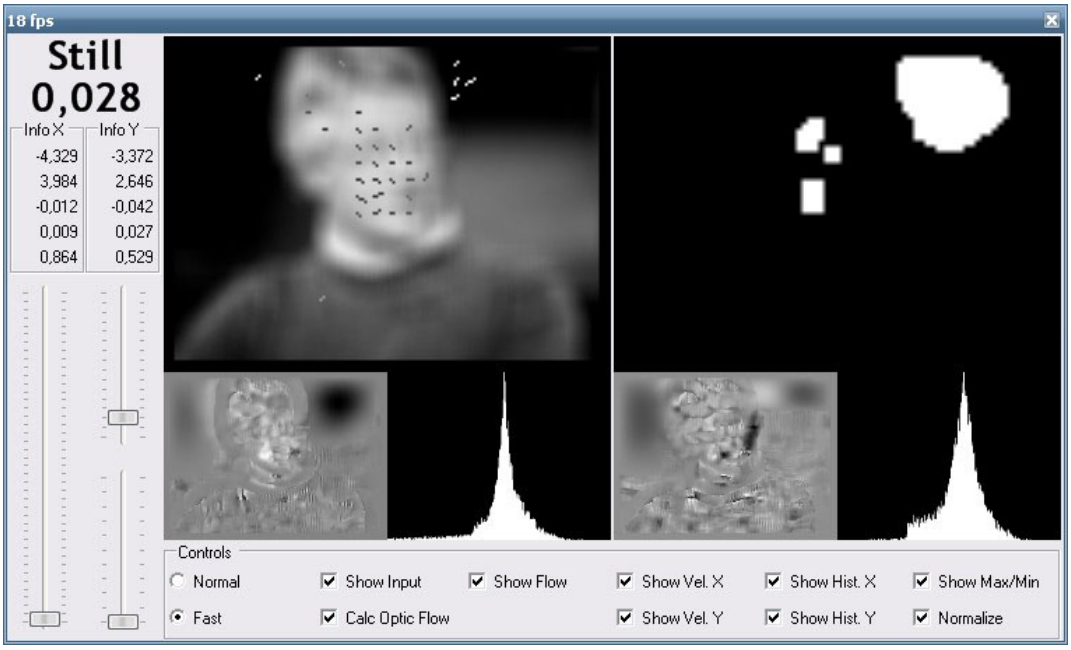
Figur 21: Hender som beveger seg igjen.

mindre og histogrammene ikke så spisse. Kamerabevegelsen er fortsatt null og modus er enda lavere en ved forrige bilde. Bevegelsen som finner sted er et hode som trekkes opp og til venstre samtidig som det snur seg. Dette vises både som vektorer og som flekker i importance-funksjonen. Men i importance-funksjonen er det også en stor hvit flekk som ikke skulle ha vært der. Altså en feil. Dette er en feil jeg opplevde ofte på helt hvite eller sorte områder, helst etter at noe har beveget seg over de. Problemet er teksturløse områder der derivatet blir null. Det er den samme grunnen til at noen områder av videooverflaten i figur 18 og 19 bare består av nullvektorer. For å løse dette problemet skal egentlig hastighetsvektorene som finnes kopieres utover flater som ikke har noe hastighetsinformasjon. Jeg trodde i utgangspunktet at jeg kunne spare prosessorkraft ved å utelate denne operasjonen. Om disse områdene bare hadde vært null ville det egentlig ikke utgjort noe problem for importance-funksjonen. Men siden bevegelse også kan "henge" seg opp i disse områdene må operasjonen utføres likevel. Det kan også virke som om støyen er en medvirkende årsak til dette problemet. Ved maksimum- og minimumsverdien av gråtonene vil all støy være ensidig fordi støyverdiene aldri kan overgå ytterpunktene. Støyen i disse områdene kan i utgangspunktet ikke resultere i bevegelse med én bestemt retning, men den kan kanskje være med på å opprettholde passerende bevegeleser. På den andre siden blir dette problemet større når α økes. Det vil si når historien får større innflytelse. Det kan altså være på grunn av en akkumulerende feil som etter hvert vokser seg stor nok til å bli betydningsfull.

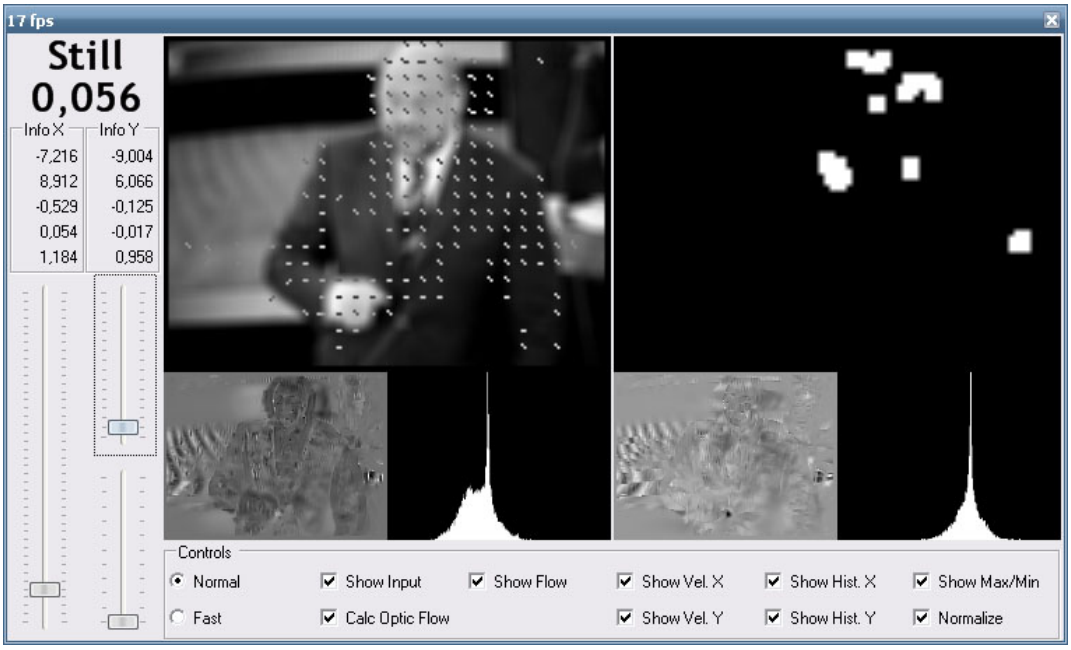
Figur 23 viser en person som lener seg rolig til høyre og litt tilbake. Selv om det er forholdsvis mye bevegelse i bildet, kanskje opp i mot 50 % av flaten, oppfattes dette ikke som kamerabevegelse. X-histogrammet i dette eksempelet skiller seg ganske godt ut fra de foregående eksemplene. Den lokale toppen til venstre for modus uttrykker den best representerte objektbevegelsen i bildet. Selv ved enda mer omfattende objektbevegelse vil modus fortsatt være null. Vektorene basert på objektbevegelse vil være mer eller mindre spredt utover mens nullvektorene vil være konsentrerte. (I figuren er modus representert over 200% bedre enn topp nummer to selv om det er bevegelse i nesten 50% av overflaten). Histogrammet er også et godt eksempel på hvorfor gjennomsnittet, som ligger et sted mellom de to toppene, ikke er en god indikator i seg selv på kamerabevegelse.

4.2 Test av konturspåring

Figur 24 inneholder en bildesekvens av en kurve som "låser" seg på et hode. Kurven er en forhåndsdefinert kvadratisk lukket b-spline med 27 kontrollpunkt (vist som sorte kryss). Langs hver normal ble den sterkeste kanten valgt som $\mathbf{r}_f$ (vist som blå sirkler) vha. masken {-0.375, -0.625, 0, 0.625,



Figur 22: Ansikt. Feil i importance-funksjon øverst til høyre.



Figur 23: Forholdsvis stor objektbevegelse.

0.375}. Det ble brukt én normal for hvert spenn (vist som gule linjestykker). Lengden på normalene er konstant. Lovlige transformasjoner var de affine og Kalman-filter ble brukt som temporært filter. Som man kan se holdt det med bare noen få iterasjoner for å finne objektet i bildet.

De neste figurene (25, 26 og 27) viser eksempler på henholdsvis translasjon, skalering og rotasjon. Jeg vil si at det er ganske gode resultater, men problemet var heller ikke av de vanskeligste selv om hastigheten ved translasjonen var forholdsvis stor. Applikasjonen, som er skrevet i C++, oppnådde rundt 50 fps under kjøring. Alle tester ble gjort med 320×240 i oppløsning fordi kameraet støttet bare 15 fps med 640×480. Det viste seg at det var driverne / koden til kameraet som brukte mesteparten av prosessorkraften. Ved å simulere objekter isteden for å bruke videodata greide applikasjonen rundt 125 fps. Ytelsen er selvfølgelig avhengig antall kontrollpunkt og frihetsgrader. Uansett vil jeg si at denne løsningen har mye å gå på når det gjelder ytelse.

Jeg gjorde de samme testene med condensation og fikk omtrent de samme resultatene. Forskjellen var det gav enda bedre resultat på raske bevegelser. I figur 25 kan man se at kurven ligger litt etter objektet. Dette taklet condensation bedre. Utfordringen ved bruk av condensation er at det kreves mer forhåndsarbeid. For at det skal virke må man definere de dynamiske modellene på forhånd. Med Kalman-filteret vil et sett med standard innstillinger virke for de fleste problemer så lenge det ikke er for mye rot i bildet. Det er bare til å definere en spline og starte opp trackeren. Condensation er også mer prosessorkrevende. Med bare 100 samples per iterasjon, spline med 7 kontrollpunkt og affine transformasjoner var algoritmen nede i 30 fps. Med 27 kontrollpunkt, slik som splinen vist i figurene har, greier algoritmen ikke mer enn 12-14 fps. (Uten videodata henholdsvis 55 fps og 25 fps.)

Jeg utførte også en del tester der trackeren feilet (både med Kalmanfilter og condensation). Det blir særlig problematisk når objektene som skal spores blir for små. I figurene kan man se at det ble brukt forholdsvis lange normaler til feature-søk. Det var nødvendig for å kunne spore raske bevegelser. Om splinen blir for liten (i areal) vil normalene "gå i hverandre" og da kan splinen kollapse helt. Det vil si at konturinformasjonen som utvinnes blir ubrukelig / irrelevant. Et annet problem er å finne gode filtre som kan virke for hele objektet. Bildene i figurene illustrerer dette: Det svarte håret gir et godt grunnlag for å bestemme konturer og derfor blir r_r i de områdene nesten alltid riktig. (Av og til velges kantene på skriften over hodet, som for øvrig er et godt eksempel på clutter.) Konturen av kinnet, derimot, er mer vanskelig å identifisere. Ofte forveksles kinnet med nesa. Det er mulig å bruke forskjellige filtre på forskjellige normaler (og forskjellig logikk), slik at den riktige kanten blir funnet for hver normal, men det er vanskelig å benytte seg av dette i praksis. Uansett viser jo dette litt av

robustheten som ligger i framgangsmåten. Så lenge flertallet av kantene identifiseres riktig vil splinen også bli riktig.

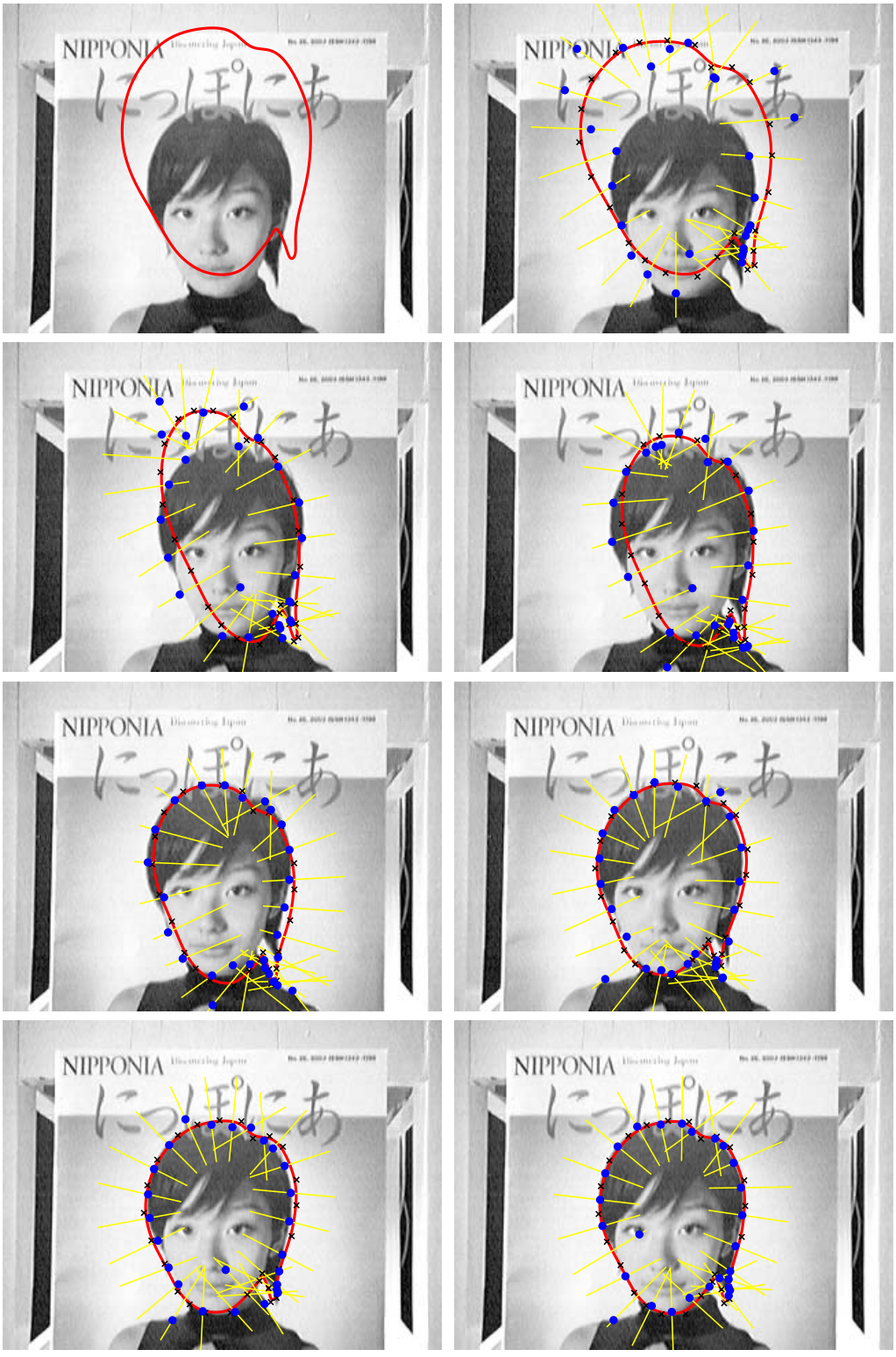
Icondensation ble jeg ikke ferdig med og har derfor ikke noen testresultater. Dette er egentlig ganske synd siden icondensation skal være en betydelig forbedring av condensation. Icondensation er også nødvendig, tror jeg, for å gjøre multitracking mulig i praksis.

4.3 Test av bevegelsestolkning

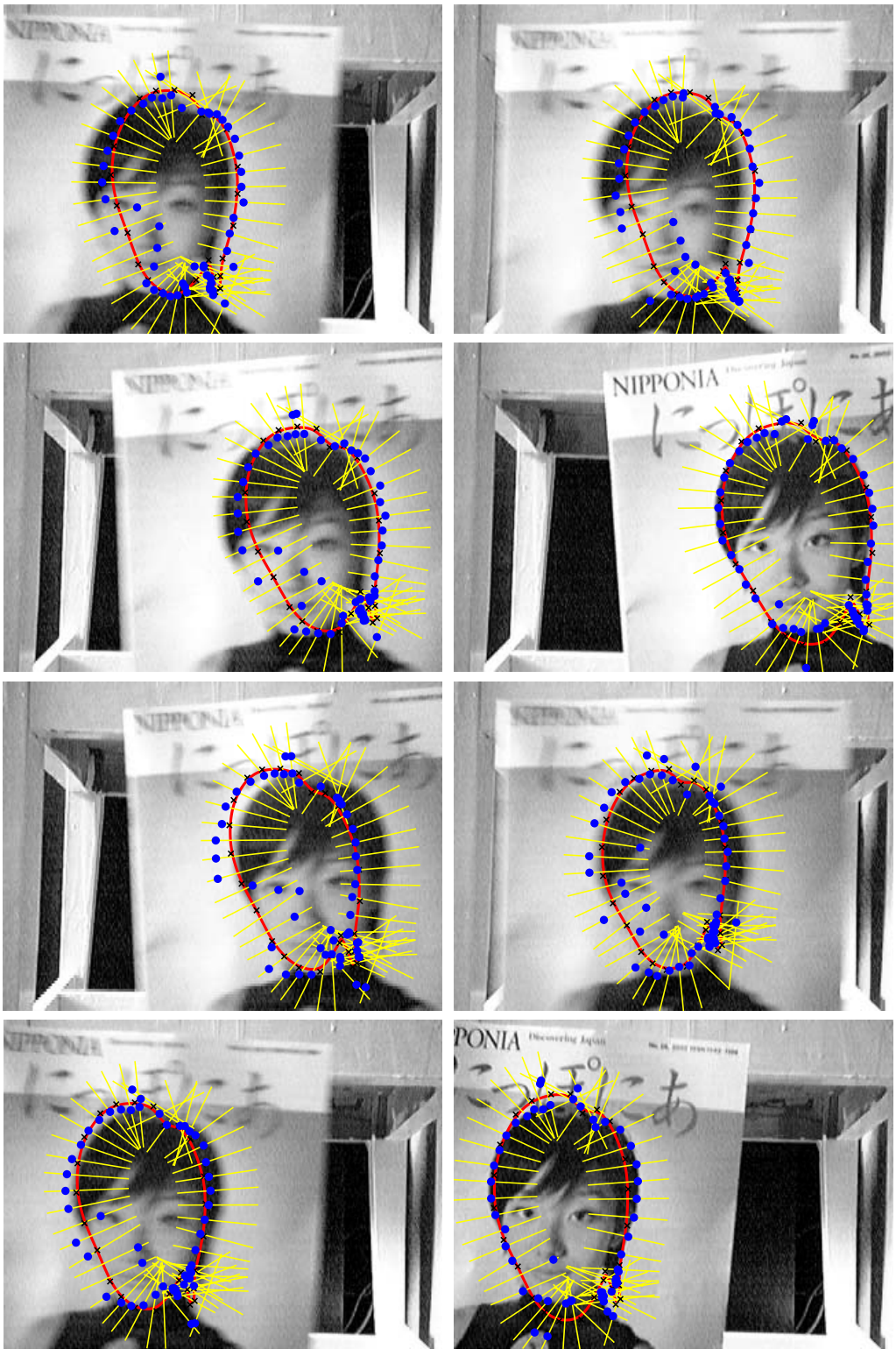
For å teste algoritmen som gjenkjenner bevegelse implementerte jeg en applikasjon der man kan simulere bevegeleser manuelt med musepeker. I figur 28 er det lært opp fire mulige ruter gjennom en rundkjøring. Rutene er tegnet på et bilde av en rundkjøring, til venstre, mens de resulterende retningsgrafene er tegnet på høyre side med grader som y-akse og tid som x-akse. De fire retningsgrafene skiller seg ganske godt ut fra hverandre. Figur 29 viser det samme men nå med fire like ruter oppå hverandre. De tilhørende retningsgrafene korrelerer ganske bra, men har litt forskjellig utstrekning i tid. I en opplæringssituasjon vil det være vanlig at den samme ruten læres opp flere ganger på denne måten. Det er selvfølgelig mulig å utelukke nye ruter som ligner eksisterende ruter vha DTW. I figur 30 er en ny rute testet mot de kjente rutene. Korrelasjonen til rute nummer 0 er svært lav og det er umulig at den skal blandes med en av de andre.

I figur 31 er 16 ruter lært opp. Det er alle nødvendige ruter gjennom rundkjøringen. Selv om det egentlig er lov å kjøre flere ganger rundt rundkjøringen antar vi nå at alle ruter uten om de 16 definerte er ulovlige. ID'ene er satt på et tilfeldig sted i begynnelsen av hver rute. Rute 0 starter i bunn og tar av første vei, rute 1 starter i bunn og tar av andre vei, osv. De tilhørende retningsgrafene skiller seg fortsatt ganske godt i fra hverandre. De ligner hverandre mest i begynnelsen og så sprer de seg utover. I figur 32 er rute nummer 6 testet. Algoritmen har ingen problemer med å identifisere den. I tabell 1 er alle rutene testet i rekke og rad. Den rette ruten kommer helt tydelig frem ut i fra korrelasjonstallene. Den største lovlige verdien er 28, den minste ulovlige verdien er 337. Jeg satte derfor grensen for lovlig / ulovlig på 100. I figur 33 er enda en lovlig rute testet, men nå med tidsnormaliserte grafer. Den eneste forskjellen er at retningsgrafene blir enda mer distinktive. Det blir derfor et større gap mellom de lovlige og ulovlige korrelasjonstallene. Men strengt tatt er det egentlig ikke nødvendig å normalisere i tid. Dette kommer selvfølgelig an på hvor store forskjeller (i lengde på rutene) man opererer med.

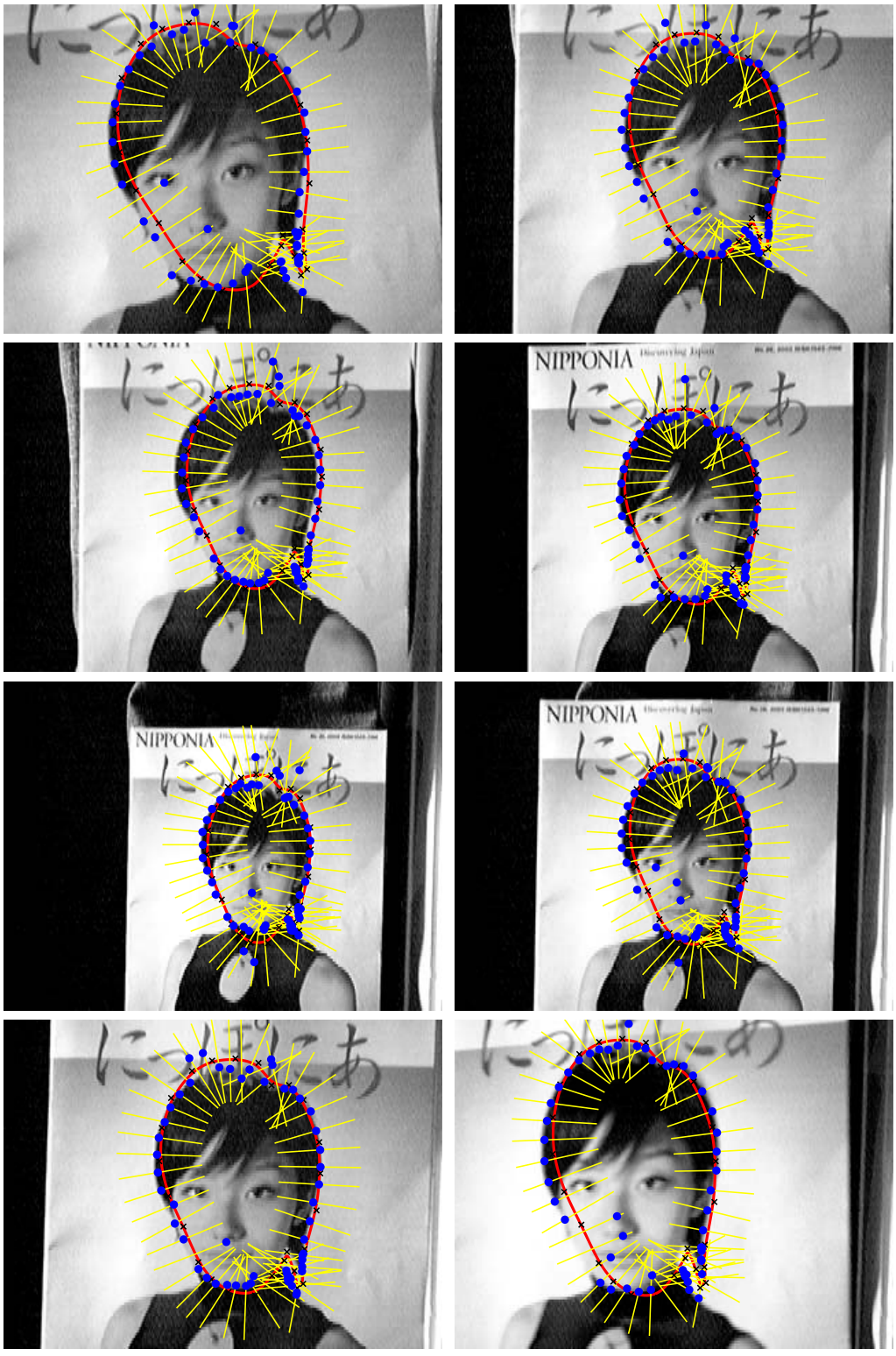
I figur 34 og 35 er to ulovlige ruter testet. Begge rutene gir dårlige korrelasjonstall. Tabell 2 viser alle korrelasjonstall da 16 ulovlige, men teknisk mulige, ruter ble testet. Den minste verdien er 160, altså over 100 som er grensen. Det er omtrent like god margin som den største lovlige verdien



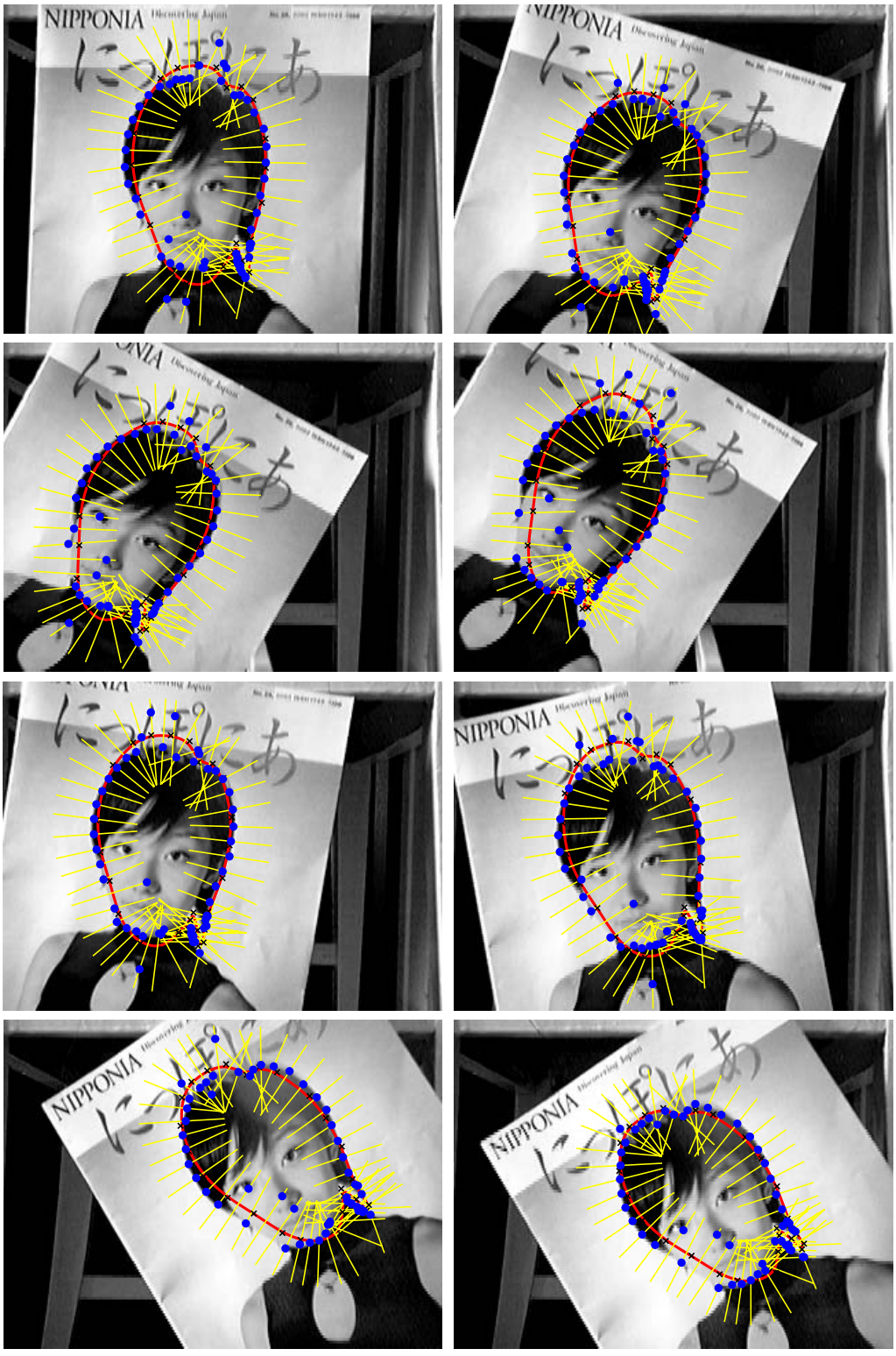
Figur 24: ”Lock on” etter bare 7 iterasjoner.



Figur 25: Test av translasjon.



Figur 26: Test av skalering.



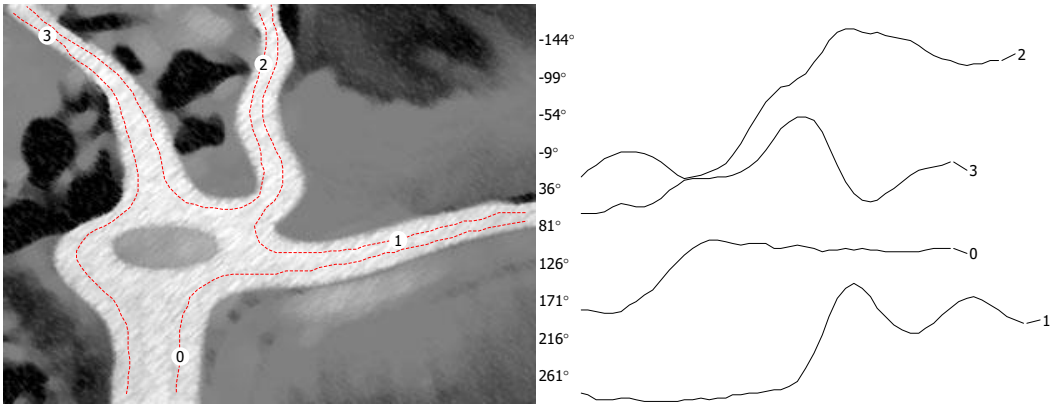
Figur 27: Test av rotasjon.

har. Ved tester med DDTW fikk jeg lavere verdier for alle målte tall. Såpass mye at det ble vanskeligere å skille rutene fra hverandre og lovlige fra ulovlige. Av og til ble det også feil. Dvs. at lignende ruter ble blandet sammen. Dette stemmer med hva jeg har skrevet før om amplituden til grafene.

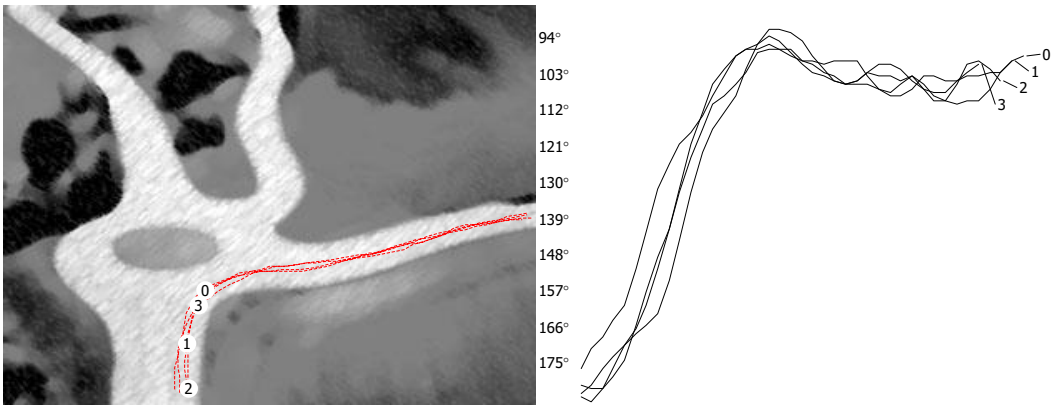
Når det gjelder ytelse er prosessorkraften som trengs ubetydelig. I de aller fleste tester bruker algoritmen 0 millisekunder (dvs. å teste en rute mot 16 lovlige). Av og til påstår programmet 16 ms. (Merk at systemet, om det er Java eller Windows, ikke har bedre oppløsning i tid ved måling.) For å teste skaleringen kopierte jeg opp rutene i figur 31. Ved 256 ruter oppgav den 16 eller 31 ms. Ved 512 ruter oppgav den 31 eller 47 ms. Prosessorforbruket blir altså merkbart etter hvert. Men nå er så mange ruter usannsynlig. Når det gjelder lengden på rutene ble de i testene sjeldent over 100. Men dette er avhengig den spatiale oppløsningen og hvor kompliserte bevegelsene er. Det er som oftest de mest fantasifulle av de ulovlige bevegelsene som blir lengre. Om man normaliserer i tid, slik som forklart i forrige kapittel, blir alle grafene lengre og da tar algoritmen mer tid.

Jeg vil si at metoden fungerer svært godt til å tolke bevegelse. Det eneste problemet med min framgangsmåte oppstår når den initiale retningen til en bevegelse ligger på grensen mellom $0°$ og $360°$. Jeg løste som nevnt denne overgangen ved å akkumulere verdiene, men dette er ikke mulig å gjøre i oppstarten av en rute. Dersom en opplært rute har initial retning på for eksempel $2°$, mens en rute som skal testes har $359°$ vil de få svært dårlig korrelasjon selv om de ellers er helt like. Ruten som skal testes vil komme ut på et helt annet nivå. Se figur 36 for et eksempel på dette. Jeg har ikke funnet noen god generisk løsning på dette problemet. Normalisering i y-retning ville selvfølgelig har virket, men det tar vekk for mye av den distinktive informasjonen til grafene. Det enkleste er kanskje å prøve å unngå denne retningen under overvåkningsfasen. Som i figuren er det bare retningen loddrett ned som er et problem (siden y-aksen er opp ned).

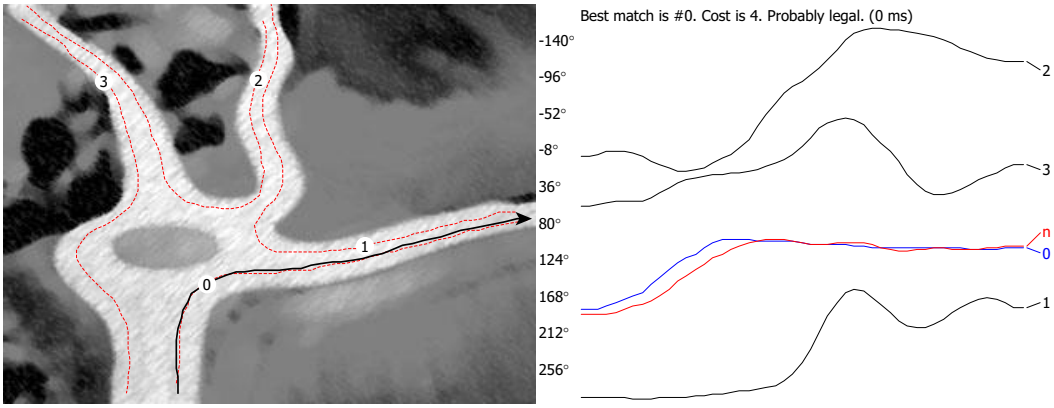
Den aller siste figuren illustrerer bare poenget om at retningsgrafene er posisjonsuavhengige. Rutene 1-3 vil tolkes som ulovlige fordi de går over områder der maksimal hastighet er (forhåpentligvis) satt lik null.



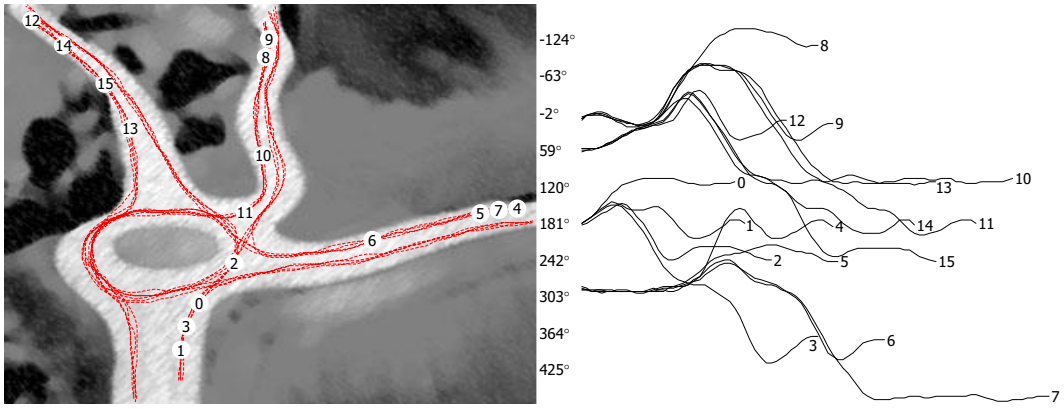
Figur 28: Fire mulige ruter gjennom en rundkjøring med tilhørende retningsgrafer til høyre.



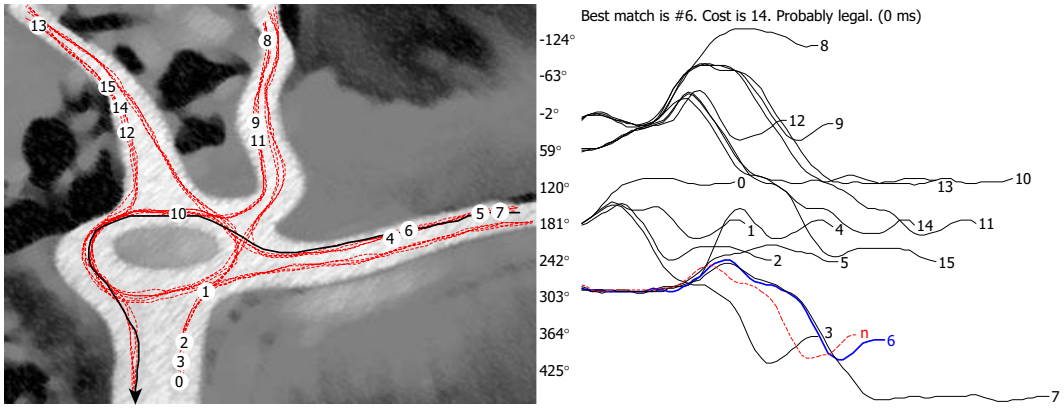
Figur 29: Fire like ruter med tilhørende retningsgrafer.



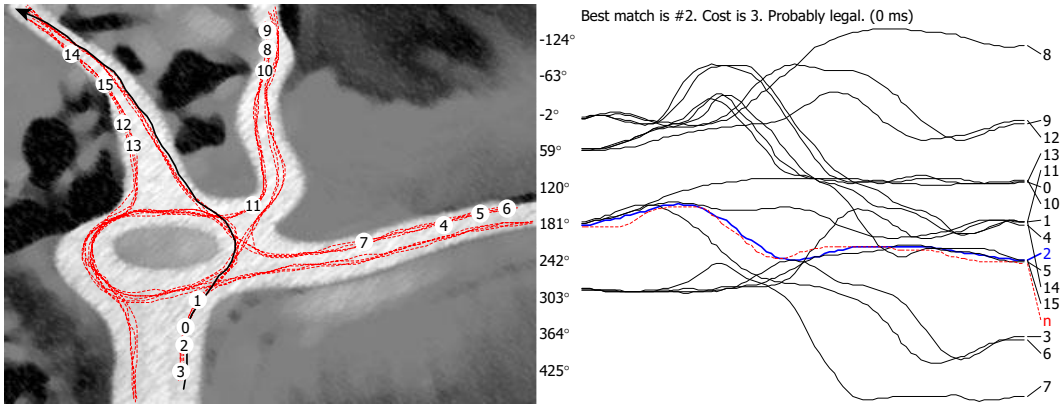
Figur 30: Test av en rute mot rutene i figur 28.



Figur 31: Alle mulige ruter (fornuftige) gjennom rundkjøringen.



Figur 32: Test av en lovlig rute mot de i figur 31.



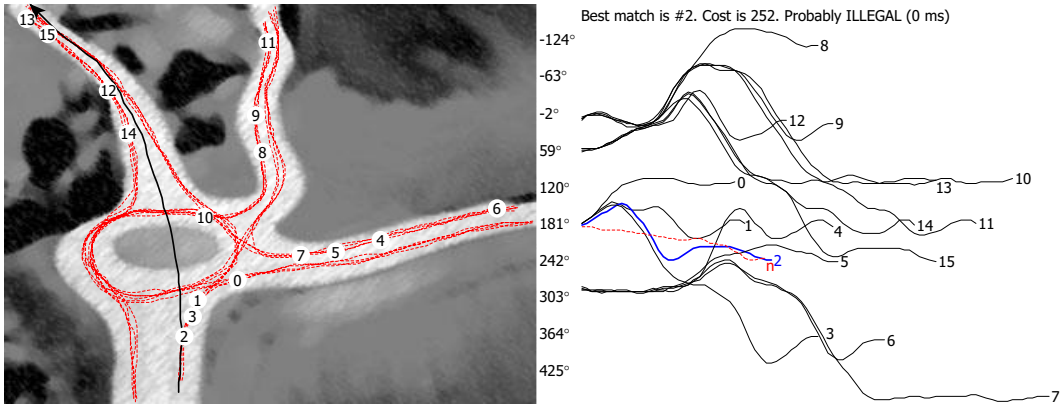
Figur 33: Samme som forrige figur men nå med tidsnormaliserte retningsgrafer.

vs	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	3519	9577	46431	11174	18216	52500	150273	51321	27447	26916	27855	12303	9794	12238	17948
1	3492	6	1013	21794	9008	11248	31154	108744	87478	51767	49024	42354	28780	22296	18416	18578
2	9833	874	13	12284	7964	6315	17890	78635	112772	67415	57424	45226	43115	29377	20184	18350
3	45071	18045	8131	21	20040	12461	5178	9486	150231	91982	69942	47831	76317	48377	29040	20834
4	11740	8122	7112	20267	19	992	15435	53197	95850	70170	69584	64514	51572	43337	40393	40020
5	19634	11673	6108	12506	846	14	7402	40566	109318	81225	87088	77368	63912	54039	47676	45726
6	50621	26940	13972	4863	12857	5869	18	4512	154614	109281	113281	93026	97378	75592	59898	51649
7	155510	98161	58721	7693	49065	37603	2871	10	320117	222441	110846	86465	243298	106031	79857	59314
8	49323	75707	87089	152029	99414	116063	188204	371267	17	8216	35306	54997	10124	33242	52982	73930
9	26902	45718	55301	91418	68933	80129	126038	266145	6920	15	4815	12245	702	5260	12759	23664
10	23538	36734	38993	59285	59067	69637	81443	127969	34604	3304	21	1314	4783	337	1771	4888
11	29338	37127	34395	41778	59379	68794	72978	93702	53764	10331	1496	16	13725	1872	519	986
12	12001	25774	36274	92073	57865	69290	123120	288061	12031	1081	6764	15729	6	5372	14399	27271
13	9321	19144	23326	46650	38710	48639	62951	115825	34622	5013	995	2253	4907	16	1363	4472
14	11925	14863	14515	26050	34558	41606	48168	86033	53152	11822	2918	1137	13238	1582	28	463
15	18396	16119	14123	19576	37429	40928	42457	71515	67908	18264	5496	1289	21770	4433	502	12

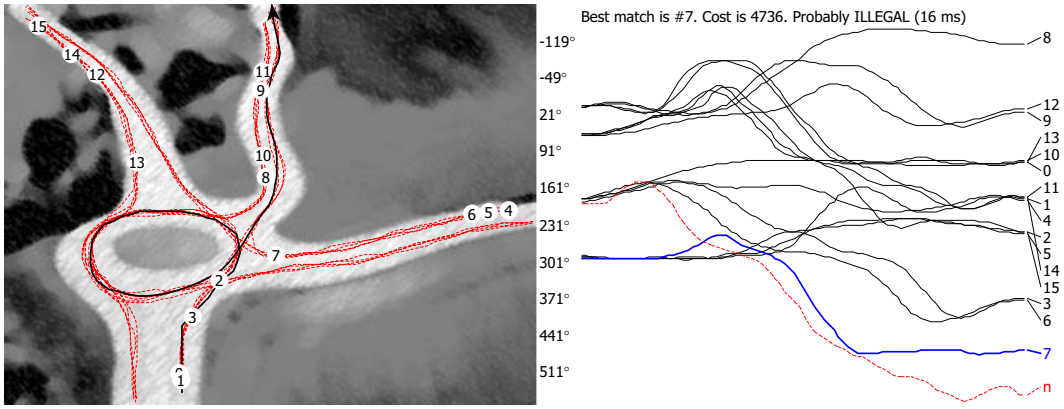
Tabell 1: Kostnader ved testing av alle mulighetene i figur 31 i kronologisk rekkefølge. Den rette ruten i hvert tilfelle kommer veldig tydelig frem. Største lovlige verdi er 28.

vs	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	8119	947	266	13263	7348	6106	18506	82220	117660	75578	75224	62506	44729	38839	30161	28368
1	5993	1006	374	13277	4461	4430	17743	82289	108647	66159	61598	52963	40018	31084	25534	24073
2	186127	116510	76182	14827	78401	59511	15952	7442	312960	203925	136847	94083	231181	119510	80103	60762
3	46364	18130	7500	946	16766	14568	11184	12216	145613	79032	33989	19659	84529	27305	14201	8014
4	107066	57806	34297	2338	44871	31034	6830	4373	215979	138582	105651	71216	143999	85502	54309	39520
5	16628	34262	54886	136280	93318	108693	187404	425215	10278	1833	17659	32672	835	15218	30501	50721
6	9714	23833	40577	108014	64139	77247	142323	333504	14948	2171	8832	19364	160	6152	16905	32378
7	27308	13667	7034	10148	3849	656	3999	35116	145559	107519	121146	103205	70328	75233	62695	57926
8	27386	10731	4619	628	17872	14124	10095	23239	135699	74920	55622	39231	58496	35556	21555	14983
9	8763	14470	18783	52328	47398	52148	84762	196293	32350	4634	2259	4222	3324	455	2569	8131
10	4266	11369	16636	42116	32645	37106	64545	156714	29280	6580	4795	6588	3449	571	2677	7486
11	7461	7765	8123	20763	30675	31837	46168	101305	47072	12181	5525	3267	11067	2086	267	841
12	41420	21505	11267	5376	11360	5225	443	9670	171022	128060	147284	122673	85622	94128	76323	68280
13	142699	88512	55452	11765	52340	35915	5987	1223	276570	185892	133234	104001	197376	105518	79648	65137
14	262259	177026	122201	36888	117602	93651	32691	18586	391260	262369	168308	121184	302815	151756	108024	85516
15	496	5067	10355	36804	6675	13241	39807	111533	41928	22778	21297	22607	13204	9042	11050	15212

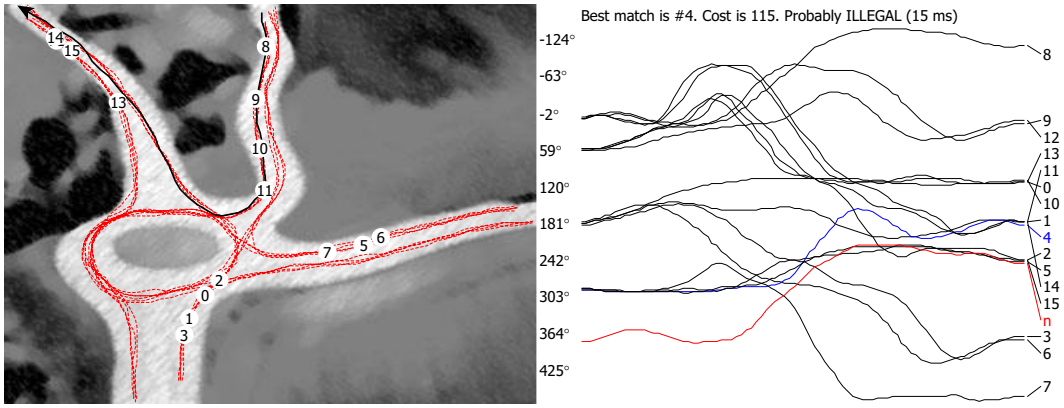
Tabell 2: Kostnader ved testing av 16 ulovlige men mulige ruter som de i figur 34 og 35. Den minste kostnaden er 160.



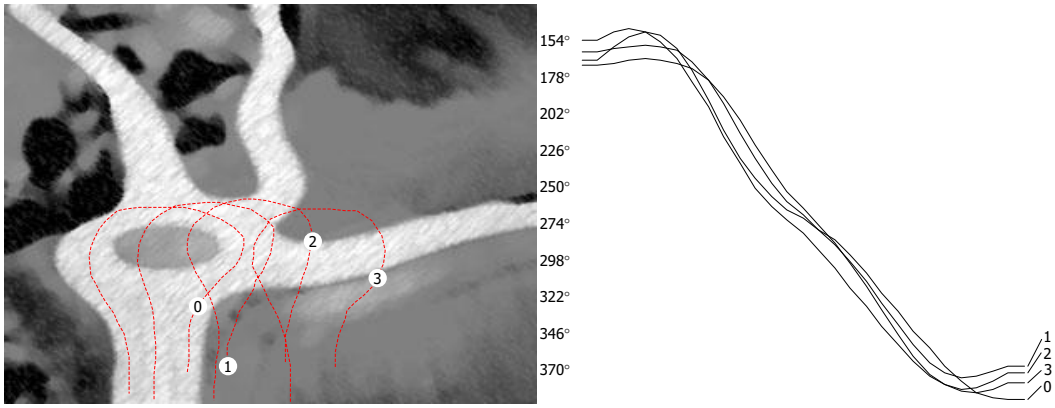
Figur 34: Test av en ulovlig rute.



Figur 35: Test av enda en ulovlig rute. Nå med tidsnormaliserte grafer.



Figur 36: En lovlig rute kommer helt feil ut på grunn av den initiale retningen. Den skulle ha matchet rute #8.



Figur 37: Fire forskjellige ruter som gir god korrelasjon som grafer.

5 Konklusjon

Tolkning av videodata er i seg selv ingen triviell oppgave og når man i tillegg må ta hensyn til kamerabevegelse åpnes det opp for helt nye problemer. Jeg har jobbet med forskjellige algoritmer for å finne gode løsninger på disse problemene. Algoritmene som er blitt brukt har, som normalt, vist både fordeler og ulemper.

Optisk Flyt Jeg valgte altså Horn og Schunck's metode for å kalkulere optisk flyt og sitter igjen med noe blandede følelser. Algoritmen i seg selv forholdsvis intuitiv, enkel å implementere og er mulig å bruke i sanntidssystemer. Den gir også greie resultater, men ikke de beste. Det nærmeste alternativet er nok *block search*. Fordelen med den metoden er at den gir flere muligheter for modifisering. Det vil si å kunne sette strengere krav til kvalitet, men også kunne utvikle heuristikk for å spare ressurser. Men det hadde nok uansett vært vanskelig å få den mindre krevende enn metoden jeg valgte. Som nevnt før er hardware som komprimerer video interessant i denne sammenheng. *Motion block* vektorene som blir brukt i MPEG, vist i figur 2, er jo optisk flyt. Dette er altså et mulig og realistisk alternativ.

Sporing med splines Det kan selvfølgelig diskuteres om hvor vanskelig teorien til denne metoden er, men jeg vil uansett påstå at den er svært omfattende. Den er ingen liten oppgave å implementere et slikt sporingssystem. Jeg kom heller ikke helt i mål selv om jeg brukte mye ferdig kode. Når det er sagt er nok metoden kanskje en av de mest robuste som finnes for sporing, i tillegg til at den ikke er alt for krevende. Men den har allikevel åpenbare begrensninger. De største problemene er størrelsene på objektene som skal spores og antallet som skal spores samtidig. Bildet av rundkjøringen som ble brukt til å teste DTW er ett godt eksempel. Med et slikt størrelsesforhold vil det nok være umulig å spore biler med splines. Biler i det eksempelet ville framstå som bitte små firkanter. Uten distinktive former og med rett og slett for små overflater for fremgangsmåten. Det er også helt opplagt at metoden ikke kan greie å spore det antallet med biler som det faktisk er mulig å ha i det omtalte bildet samtidig. Men når det gjelder oppgaver som forfølgning fungerer nok metoden svært godt. Ett objekt, enkel dynamikk, ingen begrensning på størrelse (med tanke på zoom) gir til sammen et godt grunnlag for framgangsmåten.

Bevegelsestolkning med DTW Denne framgangsmåten gir svært gode resultater samtidig som at den er lite ressurskrevende. Det er også stort rom for videre utvikling om det er ønskelig. Ved alle testene var det veldig gode marginer. Jeg har altså ingen problemer med å anbefale denne.

Oppsummering Dersom jeg skulle ha startet denne oppgaven på nytt ville jeg nok ha sett på alternative metoder til bruk i overvåkningsfasen. Den enkle løsningen hadde vært å begrense helikopteret til et fast punkt under denne fasen og unngått hele problemstillingen med bevegelig kamera. Alternativt kunne man ha utviklet forskjellige metoder til å fjerne kamerabevegelsen før videre tolkning ble gjort. Det er for så vidt det jeg har prøvd på med optisk flyt, men dette krever at den optiske flyten er av høy kvalitet. Dersom metoden jeg brukte hadde virket i sanntid med bedre oppløsning, si 640×480, ville nok situasjonen ha vært annerledes. Jeg ville uansett ha beholdt kontursporing til forfølgningsfasen og dtw til bevegelsestolkning.

Referanser

[1] M. Whalley, M. Freed, M. Takahashi, D. Christian, A. Patterson-Hine, G. Schulein, and R. Harris. The nasa/army autonomous rotorcraft project. In *American Helicopter Society 59th Annual Forum, Phoenix, AZ*, 5 2003. http://human-factors.arc.nasa.gov/apex/docs/papers/arpahs03/ARP_AHS03_v10.pdf.

[2] G. Buskey, J. Roberts, P. Corke, M. Dunbabin, and G.F. Wyeth. The csiro autonomous helicopter project. In *ISER*, 2002. http://www.itee.uq.edu.au/~wyeth/Publications/iser_article_aff.pdf.

[3] Eric N. Johnson and Paul A. DeBitetto. Modeling and simulation for small autonomous helicopter development. In *AIAA Modeling & Simulation Technologies Conference*, 1997. http://clubs.engr.arizona.edu/arc/DOCUMENTS/uarizona2002.pdf.

[4] K.Brock. Development of an autonomous aerial vehicle, 2002. http://www.ae.gatech.edu/~ejohnson/A97-3511.pdf.

[5] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Combining local and global optic flow methods. In *Pattern Recognition, Proc. 24th DAGM Symposium*, 2002. http://www.mia.uni-saarland.de/Publications/bruhn_dagm02_clg.pdf.

[6] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. *CVPR*, 92:236–242, 1994.

[7] E. Keogh and M. Pazzani. Derivative dynamic time warping. In *First SIAM International Conference on Data Mining*, 2001. http://www.cs.rutgers.edu/~mlittman/courses/lightai03/DDTW-2001.pdf.

[8] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *KDD Workshop on Mining Temporal and Sequential Data*, 2004. http://www.cs.fit.edu/~pkc/papers/tdm04.pdf.

[9] Toni M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 521–527, 2003. http://www.cs.fit.edu/~pkc/papers/tdm04.pdf.

[10] W.E.L. Grimson and Chris Stauffer. Adaptive background mixture models for real-time tracking. In *Proc. CVPR*, pages 246–252, 1999. http://www.cs.rutgers.edu/~mlittman/courses/lightai03/DDTW-2001.pdf.

[11] Anurag Mittal and Nikos Paragios. Motion-based background subtraction using adaptive kernel density estimation, 2004. http://www.umiacs.umd.edu/~anurag/cvpr2004.pdf.

[12] Jing Zhong and Stan Sclaroff. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. In *ICCV*, pages 44–50, 2003. http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0044_zhong.pdf.

[13] Thanarat Horprasert, David Harwood, and Larry S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *Proc. of International Conference on Computer Vision*, 1999. http://www.vast.uccs.edu/~tboult/FRAME/Horprasert/HorprasertFRAME99.pdf.

[14] Richard Souvenir, John Wright, and Robert Pless. Spatio-temporal detection and isolation: Results on the pets2005 datasets. In *Proceedings of the IEEE Workshop on Performance Evaluation in Tracking and Surveillance*, 2005. http://www.cs.wustl.edu/~pless/papers/souvenirWrightPless.pdf.

[15] Robert Pless, John Larson, Scott Siebers, and Ben Westover. Evaluation of local models of dynamic backgrounds. In *CVPR (2)*, pages 73–78, 2003. http://www.cs.wustl.edu/~pless/papers/plessDynamicBackgrounds.pdf.

[16] John Wright and Robert Pless. Analysis of persistent motion patterns using the 3d structure tensor. In *Proceedings of the IEEE Workshop on Motion and Video Computing*, 2005. http://www.cs.wustl.edu/~pless/papers/wright_j_tensor.pdf.

[17] Andrew Blake and Michael Isard. *Active Contours*. Springer, 1 edition, 1998. http://www.robots.ox.ac.uk/~contours/.

[18] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2 edition, 2002.

[19] Alireza Bab-Hadiashar and David Suter. Robust optic flow computation. *Int. J. Comput. Vision*, 29(1):59–77, 1998. http://www.ds.eng.monash.edu.au/suter_publications/ijcv.pdf.

[20] Simon Peter William Booth. A new fast motion search algorithm for block based video encoders, 2003. http://www.eng.uwaterloo.ca/~dclausi/Theses/SimonBoothMASc2003.pdf.

[21] David J. Fleet and Allan D. Jepson. Computation of component image velocity from local phase information. In *International Journal of Computer Vision*, pages 77–104, 1990. http://www.cs.rutgers.edu/~mlittman/courses/lightai03/DDTW-2001.pdf.

[22] Hongche Liu, Tsai-Hong Hong, Martin Herman, and Rama Chellappa. Accuracy vs. efficiency trade-offs in optical flow algorithms. In *European Conference on Computer Vision*, pages 174–183, 1996. http://www.isd.mel.nist.gov/documents/liu/ECCV96.pdf.

[23] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision - Proc. 8th European Conference on Computer Vision*, 2004. http://www.mia.uni-saarland.de/Publications/brox_eccv04_of.pdf.

[24] Berhold K.P. Horn and Brian G. Schunck. Determining optical flow. In *Artificial Intelligence*, pages 185–203, 1981. http://people.csail.mit.edu/people/bkph/papers/Optical_Flow_OPT.pdf.

[25] Alan Watt. *3D Computer Graphics*. Addison-Wesley, 3 edition, 2000.

[26] Donald Hern and M. Pauline Baker. *Computer Graphics with OpenGL*. Pearson Prentice Hall, 3 edition, 2004.

[27] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. In *Int. J. Computer Vision*, pages 5–28, 1998. http://research.microsoft.com/users/misard/papers/ijcv98.ps.gz.

[28] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *ECCV (1)*, pages 343–356, 1996. http://research.microsoft.com/users/misard/papers/condensation.ps.gz.

[29] Michael Isard and Andrew Blake. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proc 5th European Conf. Computer Vision*, pages 893–908, 1998. ftp://ftp.robots.ox.ac.uk/pub/ox.papers/VisualTracking/eccv98-importance.ps.gz.

[30] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1 edition, 1983.