Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Informasjonsteknologi – Automatisering og Signalbehandling	Vårsemesteret, 2017 Åpen
Forfatter: Kevin Alexander Østerhus	Kevin A. Østerhus (signatur forfatter)
Fagansvarlig: Ivar Austvoll Veileder: Ivar Austvoll	
Tittel på masteroppgaven: Implementering og evaluering av metoden ”EdgeFlow”, en optisk flyt metode for sanntidsbehandling. Engelsk tittel: Implementation and evaluation of the ”EdgeFlow” method, an optical flow method for real-time processing.	
Studiepoeng: 30	
Emneord: Droner, Optisk flyt, Implementering, Estimering, Blokk-matching	Sidetall: 45 + vedlegg: 11 + Zip Stavanger, 15. juni / 2017 dato/år

Sammendrag

I en utvikling hvor størrelsen på droner stadig reduseres, vil også mengden utstyr begrenses. Slike systemer er avhengige av sanntidsbehandling, hvor et kamera i kombinasjon med enkle og effektive algoritmer kan være løsningen. Hensikten med oppgaven er å se hvordan denne typen algoritmer klarer seg i forhold til ytelse og nøyaktighet.

Videre har droner de siste årene havnet i de fleste husholdninger, da ofte i små utgaver. Droner opererer i et utfordrende miljø der ressurser er begrenset. Det har derfor vært et fokus på å finne nye løsninger. Et kamera har mulighet til å hente inn informasjon om dronens omgivelser. Ved å lage enkle algoritmer i kombinasjon med et kamera, har man mulighet til å redusere prosessorkraften og på den måten øke flygetiden til dronen. I denne rapporten har en optisk flyt-metode for estimering av bevegelser i en videosekvens blitt implementert og til slutt evaluert. Den implementerte metoden er evaluert med tanke på ytelse, beregningseffektivitet og nøyaktighet. Resultatene viser at den implementerte metoden kommer godt ut når det kommer til beregningseffektivitet og ytelse. Nøyaktigheten påvirkes av søkemetoden og av at metoden benytter seg av heltallig estimering.

Forord

Rapporten er skrevet som et avsluttende prosjekt innenfor studiet *Informasjonsteknologi, automatisering og signalbehandling* ved Universitetet i Stavanger (UIS).

Jeg vil gjerne rette en takk til veileder Ivar Austvoll for muligheten, og for rådgivningen underveis i prosjektet.

Innhold

Figurliste . . . vii

Tabelliste . . . viii

1 Innledning **1**

2 Teori **2**

2.1 Datasyn . . . 2

2.1.1 Filter . . . 2

2.1.2 Kant-histogram . . . 3

2.1.3 Estimering av bevegelse . . . 5

2.1.4 Blokk-matching (BM) . . . 6

2.2 Optisk flyt . . . 10

2.2.1 Metode - *EdgeFlow* . . . 12

2.3 Evalueringsverktøy . . . 13

2.3.1 MATLAB Profiler . . . 14

2.3.2 Kvalitetssjekk . . . 15

2.3.3 Nøyaktighet . . . 16

2.4 Teknisk informasjon . . . 17

3 Implementering **18**

3.1 MATLAB . . . 18

3.1.1 Mappestruktur . . . 18

3.1.2 Implementering av *calEdgeFlow.m* . . . 19

3.2 Nøyaktighet . . . 23

4 Resultat **26**

4.1 Evaluering av *calEdgeFlow* - Ytelse . . . 26

4.2 Sammenligning av BM-algoritmer . . . 29

4.2.1 Forsøk med *threshold* 34
4.3 Evaluering av *calEdgeFlow* - Nøyaktighet 38
4.4 Diskusjon 40

5 Konklusjon 42

A Skisse av *calEdgeFlow*. 46

B EdgeFlow: Steg for steg 47

C Resultat av *calEdgeFlow*. 49

D Resultat av *Kanade-Lucas-Tomasi dense optical flow*. 53

E Zip File 56

Figurer

1.1 Illustrerende figur til innledningen. 1

2.1 Illustrerende figur for avsnittet Datasyn. 2

2.2 Filter-kjernene til en Sobel-operator. 3

2.3 Bildene er brukt i forbindelse ved forklaringen til kant-histogrammene. Bildet til venstre inneholder tre objekter som forflytter seg 20 piksler i positiv x-retning. Bildet til høyre viser resultatet etter forhåndsbehandlingen. 4

2.4 Resulterende kant-histogram av figur 2.3. 4

2.5 Eksempel på piksel- og blokkbasert bevegelsesestimering. 5

2.6 Det grunnlegende prinsippet for BM. 6

2.7 Prinsippet for BM med calEdgeFlow. 7

2.8 Blokkstørrelsen. 7

2.9 Algoritme for fullsøk (venstre). Algoritme for calEgdeFlow (høyre). 8

2.10 Vise hvordan avstanden til objekter påvirker lengden på den optiske flytvektoren. 10

2.11 Bildene representerer ulike situasjoner hvor optisk flyt forekommer. 11

2.12 Kamera montert på drone henholdsvis under og foran. 12

2.13 Eksempel: Rapport fra *Profiler*. 14

2.14 Illustrerende figur for metodene *Angular error* (AE) og *endpoint error* (EE). 16

2.15 Bildesekvensene: RubberWhale, Hydrangea og Dimetrodon. . . 16

3.1 Mappestrukturen til metoden calEdgeFlow i MATLAB. 19

3.2 Et enkelt eksempel på hvordan blokk-matchingmetoden fungerer. 22

3.3 Fra venstre: Et av bildene som skal evauleres, ground truth og fargekartet til den optiske flyten. 23

3.4 Algoritmen for evaluering av *endpoint error* (EE). 23

3.5 Sparsom og tett optisk flyt. 24

3.6 Deler av bildet blir utelukket ved oppdeling i mindre blokker. . 24

4.1 Rapport fra MATLAB *Profiler*. 26

4.2 Detaljert *Profiler* rapport av funksjonen *calHist*: **Code Analyzer results**. 27

4.3 Resultat av å kjøre listing 4.1. 28

4.4 Resultat av å kjøre listing 4.2. 28

4.5 Detaljert rapport for funksjonen *calSADBlockMatching* i seksjonen *Lines where the most time was spent"*. 28

4.6 Detaljert rapport for funksjonen *calSADBlockMatching* i seksjonen *Function listing"*. 29

4.7 Mappestrukturen til *calEdgeFlow*, hvor de fremhevede funksjonene representerer BM-delen av algoritmen. 29

4.8 Et bilde fra hver videosekvens som er listet opp i tabell 4.1 . . . 30

4.9 Resultat: Kvalitetssjekk av estimatet ved bruk av metoden PSNR. 31

4.10 Resultat: Gjennomsnittsmål på antall søk som blir utført for hver blokk. 32

4.11 Resultat: Tiden det tar å utføre de ulike BM-algoritmene. 33

4.12 Søkeprinsippet for BM i *calEdgeFlow*. 34

4.13 Bildet er forskjøvet 5 piksler mot høyre, og estimeringsfeil som legger seg i de ytterste indeksene er markert og til slutt fjernet. 34

4.14 Grafisk fremstilling for økningen av estimeringsfeil. 35

4.15 Sammenligning av metoden uten/med *threshold* - PSNR - Videosekvensen *caltrain* ble brukt. 36

4.16 Bildene skal illustrere situasjonen hvor objekter i bildet har ulike avstander - Videosekvensen *garden* ble brukt. 37

4.17 Studering av bildesekvensen *RubberWhale* med resultater fra *calEdgeFlow*. 38

4.18 Bildesekvensene: RubberWhale, Hydrangea og Dimetrodon. . . 39

4.19 Bildene som brukes for å evaluere nøyaktigheten fra videosekvensen **RubberWhale**. 39

A.1 Skisse av algoritmen calEdgeFlow. 46

B.1 Resultat av Matlab-filen *calImageGradient.m.* 47
B.2 Resultat av Matlab-filen *preProImagesBM.m.* 47
B.3 Resultat av Matlab-filen *calHIST.m.* 47
B.4 Resultat av Matlab-filen *calSADBlockMatching.m.* 48
B.5 Resultat av Matlab-filen *calLS.m* og plot av resultatet. 48

C.1 Resultat på **RubberWhale** ved *calEdgeFlow.* 50
C.2 Resultat på **Hydrangea** ved *calEdgeFlow.* 51
C.3 Resultat på **Dimetrodon** ved *calEdgeFlow.* 52

D.1 Resultat på **RubberWhale** ved *Kanade-Lucas-Tomasi dense optical flow.* 53
D.2 Resultat på **Hydrangea** ved *Kanade-Lucas-Tomasi dense optical flow.* 54
D.3 Resultat på **Dimetrodon** ved *Kanade-Lucas-Tomasi dense optical flow.* 55

Tabeller

3.1 Informasjon om bildesekvensene fra middlebury benchmark [1]. . 25

3.2 Inneholder informasjon om hvordan ulike blokkstørrelser påvirker resultatet av *calEdgeFlow.m*. 25

4.1 Informasjon om de ulike videosekvensene som er brukt i rapporten, hentet fra [2]. 30

4.2 Inneholder simuleringsresultater: PSNR. 31

4.3 Inneholder simuleringsresultater: Antall søk 32

4.4 Inneholder simuleringsresultater: Tid 33

4.5 Tabellen inneholder resultater før og etter bruk av "*threshold*". . 35

4.6 Inneholder evalueringsresultater for *Endpoint error (EE)*. 39

1. Innledning

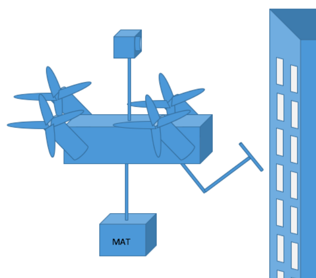
Droner er i stadig utvikling, og miljøet rundt teknologien kan bidra til nye metoder for å utføre arbeidsoppgaver. Teknisk ukeblad (TU) har de siste årene publisert en rekke artikler på emner hvor droner har spilt en sentral rolle under utviklingen av nye arbeidsmetoder. Noen eksempler på dette er en norsk nettbutikk, Kolonial, som selger matvarer og som ønsker å utvikle en drone som kan transportere produktene hjem til forbrukerene [3]. KTV Group driver med vedlikehold og ønsker derfor å utvikle en drone som kan vaske høyblokker [4]. Det danske selskapet Bygkontrol ønsker å utvikle en drone som kan overvåke byggeplasser for å oppdage feil som kan oppstå under byggeprosessen [5].

Prosjektene har noe ulikt fokus, men utfordringene knyttet til droneteknologi er ofte de samme. Størrelsen på dronen spiller en sentral rolle når det kommer til begrensninger. Begrensningene bygger på hvor mye prosessorkraft (Flygetid) og hvilken mengde utstyr man har mulighet for å bruke. Det er nødvendig å bruke forskjellige typer sensorer for å kunne utføre nødvendige beregninger tilknyttet dronens stabilitet, posisjon og hastighet.

Produktet har de siste årene havnet i de fleste husholdninger, da ofte i mindre utgaver. Disse blir ofte klassifisert som innendørsversjoner på grunn av produktets begrensninger. Disse versjonene har ikke så mye prosessorkraft, noe som reduserer flygetiden og mengden utstyr. Dette gjør at produktet ikke er like egnet for industriell bruk.

Det har derfor vært et fokus rundt å øke prosessorkraften ved å se på nye teknologier. Et kamera har mulighet til å hente inn nødvendig informasjon om dronens omgivelser. Ved å lage enkle algoritmer i kombinasjon med et kamera, har man mulighet til å redusere prosessorkraften og på den måten øke flygetiden til dronen.

En optisk flyt-metode for sanntidsbehandling har i denne rapporten blitt implementert og evaluert. Bakgrunnen for denne studien er å få et innblikk i hvordan denne typen algoritmer klarer seg i forhold til ytelse og nøyaktighet.



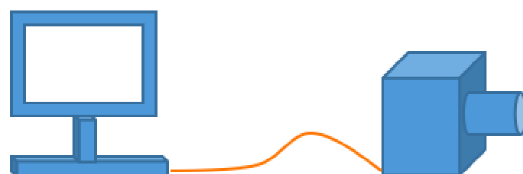
Figur 1.1: Illustrerende figur til innledningen.

2. Teori

I denne delen av rapporten vil den nødvendige teoretiske delen rundt den implementerte metoden presenteres. Begrepet «optisk flyt» vil bli gjennomgått, hvor også status på feltet vil foreligge. I siste del av rapporten vil den implementerte metoden presenteres, samt metodene for å evaluere den.

2.1 Datasyn

Et kamera er et apparat som har mulighet til å ta bilder. Et bilde inneholder mye informasjon som kan være nyttig for ulike applikasjoner. Ulike applikasjoner er ute etter forskjellig informasjon, og det er derfor nødvendig å benytte seg av et bildebehandlingsverktøy. I denne delen av rapporten vil vi gjennomgå bildebehandlingsverktøy som er inkludert i den implementerte metoden. Det vil gi et innblikk i metodens oppbygning, funksjonalitet og restriksjoner.



Figur 2.1: Illustrerende figur for avsnittet Datasyn.

2.1.1 Filter

Filter er et felles verktøy for både bilde- og signalbehandling. Bruken av kameraer har eksplodert de siste årene i kombinasjon med roboter som gjør dem mer bevisste på sine omgivelser. I denne rapporten brukes et filter for å finne kanter/linjer, og for å forbedre kontraster.

I bildebehandling er det ofte nødvendig å forhåndsbehandle bildene før man henter ut informasjon. Det fins mange ulike filtre som kan brukes til å fjerne støy og for å bevare eller fjerne detaljer. En metode for kantdeteksjon har i denne rapporten blitt brukt for å filtrere ut detaljer i bildet, for deretter å bevare kanter. Kanter i bildet oppstår når man har en stor overgang i fargeintensiteten. En kant kan karakteriseres ved [6](s.596) kantstyrke og retning gitt ved

$$Strength = E(i,j) \equiv \sqrt{[\Delta f_x(i,j)]^2 + [\Delta f_y(i,j)]^2} \tag{2.1}$$

$$Orientation : a(i,j) \equiv tan^{-1} \frac{\Delta f_y(i,j)]}{\Delta f_x(i,j)]} \tag{2.2}$$

hvor $\Delta f_x(i,j)$ og $\Delta f_y(i,j)$ er henholdsvis x- og y-komponentene til bildegradienten. En mye brukt metode for å beregne bildegradienten er *Sobel-operatoren*, som kombinerer en virtuell differanse og utjevning normalt på gradienten. *Sobel-operatorene* er gitt av to *3 x 3* filter-kjerner, som vist i figur 2.2.

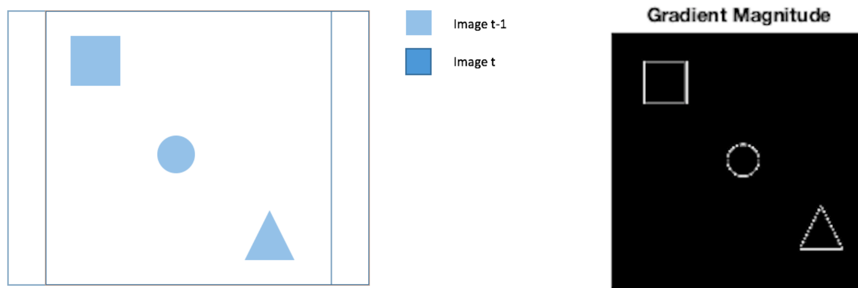
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figur 2.2: Filter-kjernene til en Sobel-operator.

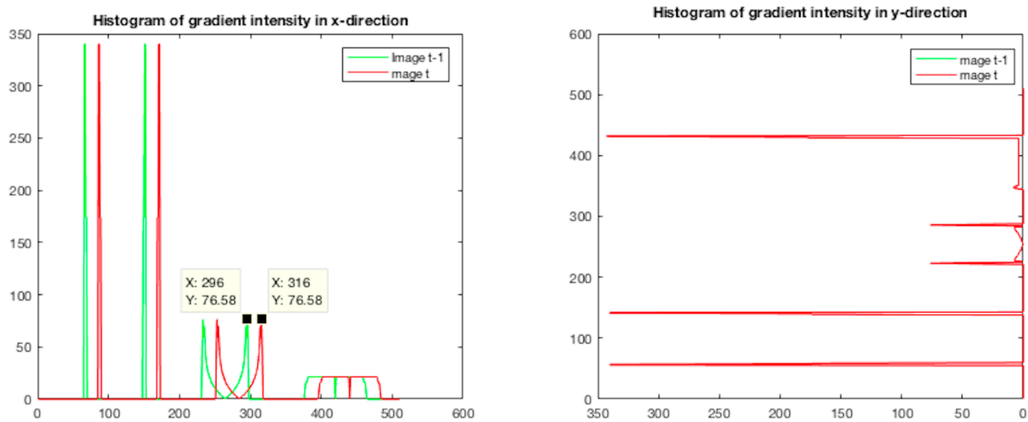
2.1.2 Kant-histogram

Histogram er et velkjent verktøy som brukes for å fremstille data på en enkel og ryddig måte. Verktøyet tas ofte i bruk og kan settes opp på ulike måter med tanke på oppsett, funksjonalitet og design [7]. I rapporten har bruken av histogram et litt annet formål enn å presentere data. Den implementerte metoden bruker histogrammer for å detektere bevegelser i bildene.

Bildene går igjennom en forhåndsbehandling hvor kanter og linjer lokaliseres, vist ved figur 2.3. Bildene deles deretter opp i mindre blokker, hvorfra blokkene summeres sammen til histogrammer. Histogrammene inneholder nå informasjon om hvor i blokkene kanter og linjer forekommer, vist ved figur 2.4. Ut ifra histogrammene har en mulighet for å detektere to typer bevegelser: inn og ut av scenen, samt forflytninger i horisontal og vertikal retning. I denne rapporten har det i første omgang vært et fokus på bevegelser i horisontal og vertikal retning. Denne bevegelsen finner man ved å skyve histogrammene over hverandre for å finne den beste «matchen». Bevegelser ut og inn i scenen detekteres ved å studere høydene på dataene som foreligger i histogrammene.



Figur 2.3: Bildene er brukt i forbindelse ved forklaringen til kant-histogrammene. Bildet til venstre inneholder tre objekter som forflytter seg 20 piksler i positiv x-retning. Bildet til høyre viser resultatet etter forhåndsbehandlingen.



Figur 2.4: Resulterende kant-histogram av figur 2.3.

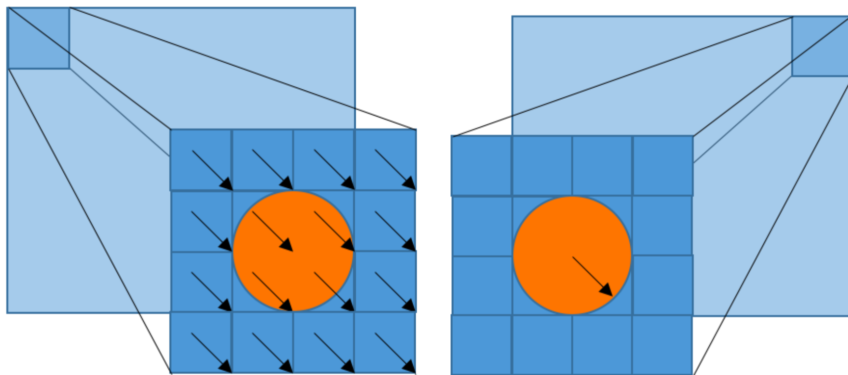
2.1.3 Estimering av bevegelse

I en videosekvens forekommer det bevegelser av ulike slag. Hvilke elementer som forflytter seg bestemmes ofte av hvilken type applikasjon kameraet inkluderes i. Dersom et kamera har en fast posisjon, vil kun objekter som forflytter seg registreres som om de var i bevegelse. Hvis kameraet ikke har en fast posisjon, men forflytter seg, så vil også omgivelsene (stillestående objekter) være i bevegelse.

Når bevegelser i bildet skal estimeres så er det to grunnleggende metoder som kan brukes [8](s.13):

- Pikselbasert bevegelsesestimering
- Blokkbasert bevegelsesestimering

Metodene har ulike karakteristiske elementer som gjør dem egnet for ulike applikasjoner. Den førstnevnte metoden går ut på at man skal finne en forflyttelsesvektor for hver piksel som er inkludert i bildet, vist ved figur 2.5(Venstre). Denne metoden blir ofte ressurskrevende, og er derfor ikke beskrevet noe ytterligere i denne rapporten. Blokk-matching er en rask og effektiv metode som bygger på at man deler et bilde inn i mindre blokker. Blokkene blir deretter enkeltvis sammenlignet med det neste bildet i sekvensen for å finne et matchende område. Resultatet av dette er at man får en forflyttelsesvektor for hver blokk, det vil si et større område av bildet. Dette er vist ved figur 2.5(Høyre), som i dette eksempelet er en 4x4-blokk.

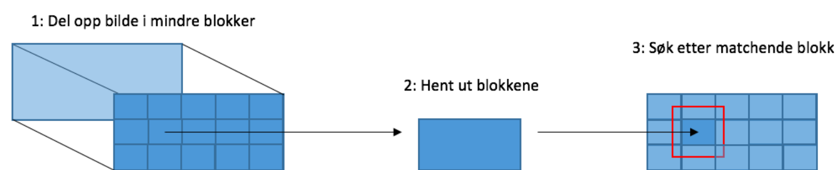


Figur 2.5: Eksempel på piksel- og blokkbasert bevegelsesestimering.

2.1.4 Blokk-matching (BM)

Blokk-matching (heretter avkortet BM) er en metode som ofte tas i bruk ved estimering av bevegelser, spesielt ved videokoding [8](s.13). Metoden har i nyere tid også blitt anvendt i videokomprimering [9]. Det grunnleggende prinsippet for metoden er beskrevet i [8](s.14-16) og vist ved figur 2.6. Det er viktig å se på de ulike parametrene som inngår i metoden for å forstå hvordan de påvirker ytelsen og det endelige resultatet, nemlig nøyaktigheten. Det er totalt tre parametre som må bestemmes for algoritmen, og disse vil bli utredet noe ytterligere i de neste delpunktene [8](s.16):

- Blokkstørrelsen.
- Størrelsen på søkeområdet.
- Kriterium for blokksammenligning.



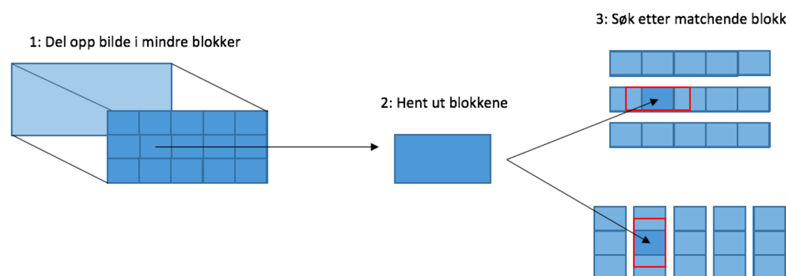
Figur 2.6: Det grunnlegende prinsippet for BM.

For å forstå hvordan metoden er implementert i denne rapporten, må man først se på det grunnleggende prinsippet for BM. Det grunnleggende prinsippet bygger på at man henter ut to bilder i en videosekvens, (I_{t-1} og I_t), for deretter å dele I_{t-1} inn i mindre blokker. En matematisk beskrivelse av dette vil være en matrise. Hvor mange elementer matrisen vil ende opp med bestemmes av størrelsen som settes på blokkene. Elementene i matrisen blir deretter enkeltvis sammenlignet med I_t. Dette gjøres ved at de skyves over et begrenset område, som er representert med et rødt kvadrat i figur 2.6. Antall søk bestemmes av størrelsen på det begrensede området, samt hvilken søkemetode som tas i bruk. Det enkleste prinsippet dreier seg om å utføre et søk over hele det begrensede området, et såkalt 2D-søk. Det er ofte denne delen av algoritmen som skiller seg ut. Det er blitt utført mange eksperimenter på ulike søkemetoder. Disse kan studeres nærmere i [10]. Det blir deretter utført en sammenligning hvor man leter etter det området som passer best med den blokken som er brukt under søket. Ulike metoder kan tas i bruk for å bestemme den beste matchen. Området som matcher best representerer da forskyvningen fra (I_{t-1} til I_t), som kan utrykkes ved 2.3 og 2.4:

$$I_{t-1}(x, y) \tag{2.3}$$

$$I_t(x \pm u, y \pm v) \tag{2.4}$$

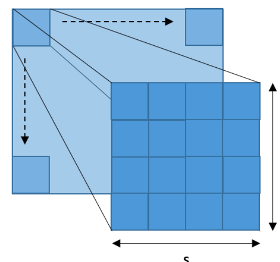
BM-algoritmen som er implementert i denne rapporten benytter seg av et 1-dimensonalt (1D) søk, istedenfor 2D-søk. Søket blir utført horisontalt og vertikalt, vist ved figur 2.7, som er en enkel representasjon av prosessen.



Figur 2.7: Prinsippet for BM med calEdgeFlow.

Blokkstørrelsen

Når det kommer til hvilken størrelse som skal settes på blokkene, så dreier det seg om å kompensere. Små blokker foretrekkes på grunn av at det vil redusere muligheten for at to objekter som forflytter seg i ulike retninger havner i samme blokk. Derimot vil større blokker redusere antall søk og redusere prosessorkrafen. Den blir derfor ofte satt til 16 x 16 [8](s.18).



Figur 2.8: Blokkstørrelsen.

I rapportens metode er en avhengig av at bildesekvensene inneholder en del detaljer, i form av kanter. Dersom en blokk i bildet ikke skulle inneholde noen detaljer, vil det påvirke nøyaktigheten til metoden. Videre vil dimensjonene på bildene spille en sentral rolle, hvor større bilder vil inneholde flere blokker som resulterer i flere søk, som igjen påvirker prosessorkraften.

Størrelsen på søkeområdet

Størrelsen på søkeområdet påvirker nøyaktigheten og prosessorkraften som metoden skulle trenge under estimeringen av forflyttelsesvektorene. Det er i denne delen det er mulighet for å spare mest med tanke på prosessorkraften. Parameteren **d** bestemmer det maksimale søkeområdet som blokkene kan skyves over, vist ved figur 2.9. En liten **d** vil begrense størrelsen på mulige bevegelser som kan forekomme i en videosekvens. Hvis det skulle forekomme store forflyttelser i bildet, vil en liten **d** ikke kunne kompensere for dette. Ved å sette **d** til en

stor verdi vil man øke bevegelsesområdet, men samtidig øke antall søk, som igjen påvirker prosessorkraften.

Figur 2.9 illustrerer søkeområdet for metoden, hvor hele området gjennomgås sammen med søkeområdet for metoden som er implementert i denne rapporten. Ved å ta utgangspunktet i eksempelet under vil man kunne redusere antall søk med hele 86,7 prosent. Dette vil redusere prosessorkraften, men da på bekostning av nøyaktigheten til algoritmen.

Sammenligning:

*Formlene som er brukt i eksempelet er hentet fra figur 2.9 og brukes for å sammenligne algoritmene. I dette eksempelet har bildet en dimensjon på 512 x 512 (M x N), størrelsene på blokkene settes til å være 16 x 16 (s x s) og **d** settes til å være lik 7. Ut ifra informasjonen vil en få en matrise på 32 x 32 blokker, som tilsvarer totalt 1024 blokker.*

Algoritmen for fullsøk:

$$\text{Én enkel blokk: } (2d+1)^2 = (2*7+1)^2 = 225 \tag{2.5}$$

$$\text{Totalt antall søk: } 225*1024 = 230\ 400 \tag{2.6}$$

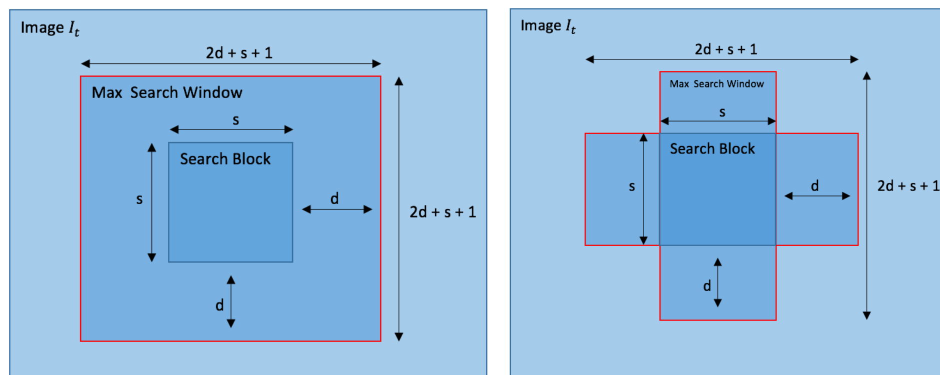
Algoritmen for calEdgeFlow:

$$\text{Én enkel blokk: } (2d+1)*2 = (2*7+1)*2 = 30 \tag{2.7}$$

$$\text{Totalt antall søk: } 30*1024 = 30\ 720 \tag{2.8}$$

Forskjellen i prosent:

$$\frac{230\ 400 - 30\ 720}{230\ 400} * 100 \approx 86,7\% \tag{2.9}$$



Figur 2.9: Algoritme for fullsøk (venstre). Algoritme for calEgdeFlow (høyre).

Kriterium for blokksammenlikning

Kriterium for blokksammenligning er metoden som brukes for å evaluere søkeområdet i I_t, som passer best med blokken som er hentet fra I_{t-1}. Ulike metoder kan tas i bruk, men de må evalueres med tanke på antall og type operasjoner. For hvert søk må denne metoden utføre de operasjonene som inngår i metoden, noe som vil påvirke prosessorkraften direkte. Eksempler på metoder som kan tas i bruk er vist ved listen under:

- Mean Square Error (MSE)
- Sum Of Absolute Difference (SAD)

Hver av de nevnte metodene over har sine ulike egenskaper og operasjoner. I artiklene som omhandler den implementerte metoden [11, 12] benyttes SAD. Den blir også benyttet i en av artiklene som metoden er inspirert fra [13], men i artikkelen [14] benyttes kun MSE. Det er derfor nødvendig å evaluere begge de nevnte metodene for å se hvilke av disse som skiller seg ut med tanke på prosessorkraft. Formlene 2.10 og 2.11 er hentet fra [8](s.17), og det som skiller dem er at man ikke utfører noen multiplikasjon i SAD-delen. Det betyr at for den implementerte metoden så vil SAD passe bedre med tanke på antall operasjoner.

$$MSE(d_x, d_y) = \frac{1}{NxN} \sum_{(i=x)}^{x+N-1} \sum_{(j=y)}^{y+N-1} [I_t(i,j) - I_{t-1}(i+d_x, j+d_y)]^2 \tag{2.10}$$

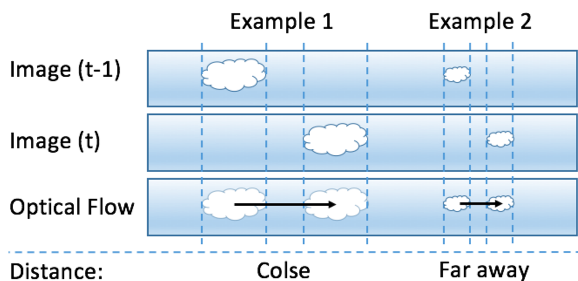
$$SAD(d_x, d_y) = \sum_{(i=x)}^{x+N-1} \sum_{(j=y)}^{y+N-1} |I_t(i,j) - I_{t-1}(i+d_x, j+d_y)| \tag{2.11}$$

2.2 Optisk flyt

Uttrykket «*optisk flyt*» *(eng.* «*optical flow*»*)* defineres ulikt i ulike sammenhenger. *Geoffrey Barrows* er en av forfatterene bak artikkelen [13], som er en av artiklene den implementerte metoden hentet inspirasjon fra. Han startet bedriften *Centeye* [15], som produserer og designer små kameraer med en oppløsning på et par tusen piksler. Bedriften henter inspirasjon fra dyrelivet, noe som gjenspegler seg i prosjektene de utvikler. De har laget et lite og lett helikopter som bruker disse kameraene til å kunne holde seg i en fast posision i lufta. Videre har de brukt teknologien i andre sammenhenger for å kunne unngå hindringer. På grunn av kameraets størrelse, og det faktum at kameraet har et lavt antall piksler, så trengs det ikke mye prosessorkraft for å behandle dataene fra disse kameraene [16]. *Centeye* refererer til *Optisk flyt* som et "visuelt fenomen som du opplever hver dag"[17]. De utdyper dette noe hvor "*Optiske flyt* er den tilsynelatende visuelle bevegelsen du opplever når du beveger deg igjennom verden"[17].

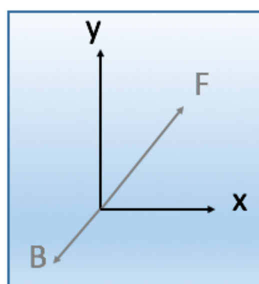
Det handler om hvordan man opplever bevegelser i synsfeltet i løpet av korte tidsperioder. Figur 2.11 viser hvordan den optiske flyten oppleves ved ulike situasjoner. Bildene til venstre viser eksempler på bevegelser i horisontal og vertikal retning. Figur 2.11(a) viser hvordan bevegelser oppleves når kameraet skyves mot venstre. Innholdet i bildet vil skyves i motsatt retning, og det er denne bevegelsen man uttrykker med optisk flyt. Bildene til høyre representerer en annen situasjon, der kameraet skyves framover og bakover. Bevegelsene oppleves her på en helt annen måte, hvor den optisk flyten trekkes mot midten i 2.11(d) og mot alle kanter i 2.11(f).

Så hva er det som gjør at optisk flyt integreres i videoapplikasjoner? Ved å studere bilder hvor den optiske flyten er synlig, vil man legge merke til at objekter tildeles ulike lengder på de optiske flytvektorene. I dette ligger det at objekter som er nærmere kamera vil oppleves å flytte seg over et større område enn objekter som er langt ifra, vist ved figur 2.10.

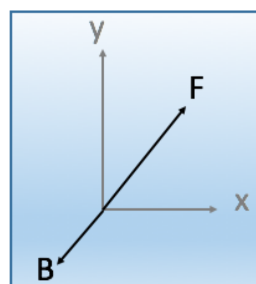


Figur 2.10: Vise hvordan avstanden til objekter påvirker lengden på den optiske flytvektoren.

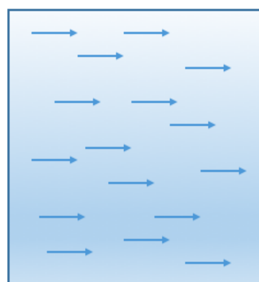
I enkle applikasjoner har en mulighet for å benytte seg av de ulike optiske flytmønstrene til å navigere, unngå hindringer og regulere ens hastighet [17]. En algoritme for optisk flyt blir ofte brukt i kombinasjon med andre målinger for å løse ulike problemstillinger, avhengig av typen applikasjon en ønsker å utvikle. Ved å integrere en avstandsmåler og måle avstanden til objekter som forflytter seg i bildet, vil en kunne beregne seg frem til hastighetene ved hjelp av enkle matematiske formler. Det må da bli tatt hensyn til om kameraet er i bevegelse eller ikke. I denne rapporten har det vært et fokus å kun se på et kamera i kombinasjon med enkle algoritmer for å beregne den optiske flyten som forekommer i en bildesekvens.



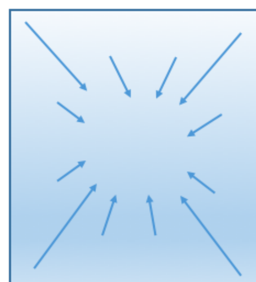
(a) Bevegelse i x- og y-retning: Vises med koordinater.



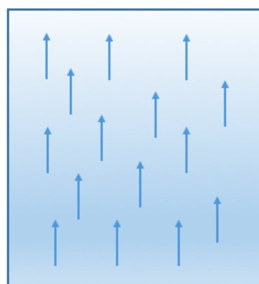
(b) Bevegelse ut og inn av scenen: Vises med koordinater.



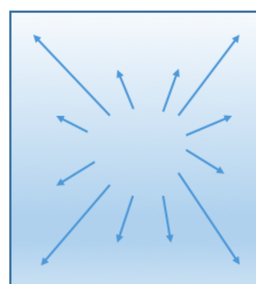
(c) Bevegelse i horisontal retning: Kamera skyves fra høyre mot venstre.



(d) Bevegelse bakover i scenen.



(e) Bevegelse i vertikal retning: Kamera skyves nedover.

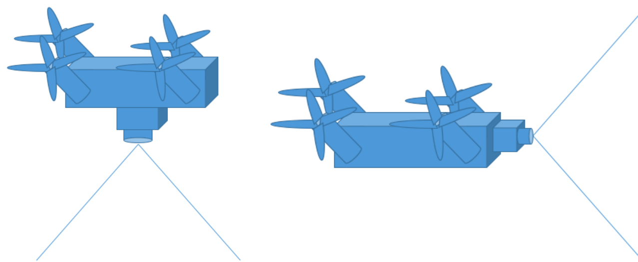


(f) Bevegelse fremover i scenen.

Figur 2.11: Bildene representerer ulike situasjoner hvor optisk flyt forekommer.

2.2.1 Metode - *EdgeFlow*

I prosjekter hvor et kamera forekommer finner man ofte en utredning av en metode for optisk flyt som er implementert i applikasjonen. Dette er en voksende trend og byr på ulike utfordringer med tanke på dagens teknologi. I denne rapporten har det vært et fokus å se på små droner, noe som betyr at metodene som skal brukes må være enkle for å minimere prosessorkraften. Algoritmen som er implementert i denne rapporten blir presentert i [11] og [12], hvor en lomme-drone er brukt. I [11] er kamera montert under dronen for hastighetskontroll, samt for å holde dronen stabil i horisontal retning. I [12] er kamera montert i fronten for å kunne unngå hindringer. Figuren 2.12 viser de forskjellige monteringene. Algoritmen *EdgeFlow* er inspirert fra [13, 14], og begge disse artiklene handler om hvordan dronen skal kunne unngå hindringer.



Figur 2.12: Kamera montert på drone henholdsvis under og foran.

Algoritmen *EdgeFlow* er presentert i [11] og [12]. Algoritmen er bygd opp av ulike metoder som ble presentert under avsnittet 2.1 *Datasyn*. Det er nødvendig å se på de ulike stegene som inngår i algoritmen for å få et helhetlig bilde av algoritmens funksjonalitet. Ut ifra en videosekvens vil en hente ut to bilder, som begge inneholder informasjon om dronens omgivelser. Bildene går først igjennom en for-prosessering hvor kanter og linjer i bildet lokaliseres. Bildene summeres sammen for å produsere to kant-histogram. Histogrammene brukes videre i en enkel BM-metode, i kombinasjon med summen av absolutt differanse (SAD), for å estimere bevegelsen i bildet (lokal flyt). Det blir deretter laget en lineær modell av forskyvningene ved å bruke minste kvadraters metode. Det vil gi et estimat på kameraets forflytning (global flyt).

2.3 Evalueringsverktøy

Rapporten har et fokus på enkle og effektive algoritmer, noe som medfører at metoden må evalueres med tanke på ytelse og nøyaktighet. Før man starter med å analysere kode, er det viktig at man har et førsteutkast. I dette ligger det at man ikke skal legge tid ned i å analysere ulike deler av koden hver for seg. Grunnen til dette er at man kan ende opp med å bruke mye tid på deler av koden som i det store bildet har minimal innvirkning på ytelse og nøyaktighet.

Før man starter med å endre på kode, er det viktig å evaluere tidsforbruket ved eventuelle endringer opp imot hvor mye tid man kan spare i det lange løpet. Kode kan deles inn i ulike nivåer [18](s.3), vist ved listen under. Ut ifra dette kan man enkelt evaluere om man skal legge tid i endringer, eller om man skal bruke tid andre steder.

- Nivå 1: Kode som blir brukt én gang.
- Nivå 2: Kode som blir brukt ofte.
- Nivå 3: Kode som blir brukt av andre.

I første omgang skal ytelsen analyseres. Ved å analysere programmer kan man enkelt identifisere ulike problemstillinger [18](s.25) listet under.

- Identifisere tidsforbrukende kodelinjer.
- Identifisere kodelinjer som ikke tas i bruk.
- Identifisere hvor programmet kan effektiviseres med tanke på CPU, minne eller I/O.

MATLAB har noen innebyggede kommandoer og funksjoner som kan benyttes [19]. Funksjonen som er benyttet til dette formålet er beskrevet noe ytterligere under avsnittet 2.3.1 *MATLAB Profiler*. Den implementerte metoden benytter seg av BM for å estimere forflyttelsen i bildet. Denne delen blir analysert hvor ulike BM-algoritmer sammenlignes med hensyn til kvaliteten på estimeringsresultatene, avsnitt 2.3.2 *Kvalitet*. Videre skal metoden evalueres med tanke på nøyaktighet. Metoden som er benyttet i denne rapporten er beskrevet under avsnitt 3.2 *Nøyaktighet*.

2.3.1 MATLAB Profiler

For å aktivere analyseverktøyet benyttes koden som vist under listing 2.1. Dette er et eksempel hvor MATLAB-verktøyet *Profiler* aktiveres, koden analyseres, før verktøyet avsluttes og viser frem resultatet.

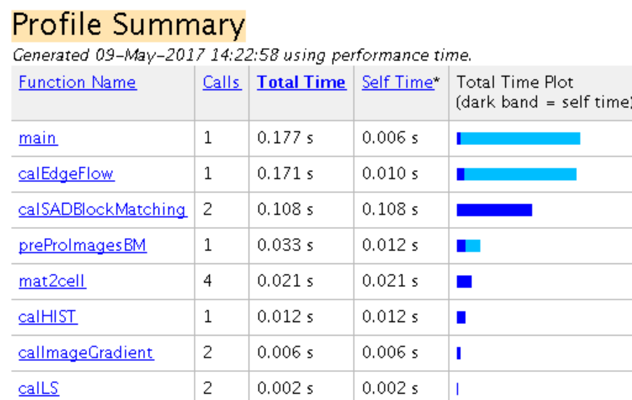
Listing 2.1: Oppsettet for å kjøre MATLAB-funksjonen *Profiler*.

```
% Turn on the Profiler
profile on

[...] = calEdgeFlow(...); % -> Code to be analysed

% Turn off the Profiler
profile off
% Show Profiler report
profile viewer
```

MATLAB-funksjonen *Profiler* er et analyseverktøy som viser resultatet i form av et grafisk brukergrensesnitt (Graphical User Interface - GUI) [20], vist ved figur 2.13. I denne delen av rapporten er figur 2.13 bare en illustrasjon for å kunne forklare hva som kan leses ut av resultatet. Resultatet er satt opp i tabellform, hvor de ulike funksjonene (MATLAB-filene) er listet opp helt til venstre. I dette eksemplet ble MATLAB-filen *calEdgeFlow.m* kjørt fra hovedfilen *main.m*. Den totale tiden det tok for å kjøre metoden var 0,174 sekunder (s). Her brukte den kun 0,01s i sin egen fil, som vist under **Self Time**. Funksjonen som brukte lengst tid var *calSADBlockMatching*, som i denne analysen brukte 0,109s. I tillegg vises det hvor mange ganger de ulike funksjonene blir kalt på. For en mer detaljert rapport kan man klikke seg inn på de ulike funksjonene. Det foreligger en grundig forklaring i [18](s.28-31) med bildeeksempler.



Profile Summary

Generated 09-May-2017 14:22:58 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	0.177 s	0.006 s	
calEdgeFlow	1	0.171 s	0.010 s	
calSADBlockMatching	2	0.108 s	0.108 s	
preProImagesBM	1	0.033 s	0.012 s	
mat2cell	4	0.021 s	0.021 s	
calHIST	1	0.012 s	0.012 s	
calImageGradient	2	0.006 s	0.006 s	
calLS	2	0.002 s	0.002 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Figur 2.13: Eksempel: Rapport fra *Profiler*.

2.3.2 Kvalitetssjekk

I [10] ble syv ulike BM-algoritmer beskrevet, Exhaustive Search (ES), Three Step Search (TSS), New Three Step Search (NTSS), Simple and Efficient TSS (SES), Four Step Search (4SS), Diamond Search (DS), og Adaptive Rood Pattern Search (ARPS). Algoritmene ble sammenlignet med tanke på antall søk, tidsforbruk og kvaliteten på estimatet. Ved å utføre denne sammenligningen vil en kunne se på hvordan rapportens integrerte metode klarer seg i forhold til [10]. *Peak signal-to-noise ratio* (PSNR) er metoden som er brukt for å sammenligne kvaliteten på estimatet av forskyvningen i bildet, gitt ved

$$PSNR = 10 \cdot log_{10}(\frac{255^2}{MSE}) \tag{2.12}$$

Metoden blir mye brukt innenfor videokoding ved sammenligning av kvaliteten mellom komprimerte og dekomprimerte videosekvenser [21](s.19). Metoden gir et utilstrekkelig resultat når det kommer til nøyaktigheten på bevegelser i bildet, og brukes derfor kun for å sammenligne BM-delen av algoritmen. Metoden blir også benyttet i [22], hvor tolv ulike algoritmer ble samenlignet, hvorav noen er nevnt over [10]. I tillegg til å beregne PSNR i [22], ble det utført en sammenligning der ES ble brukt som referanse [22]. Det vil gi en indikasjon på hvordan de ulike algoritmene gjør det i forhold til hverandre målt i prosent, gitt ved

$$D_{PSNR} = -(\frac{PSNR_{ES} - PSNR_{BM}}{PSNR_{ES}}) \cdot 100\% \tag{2.13}$$

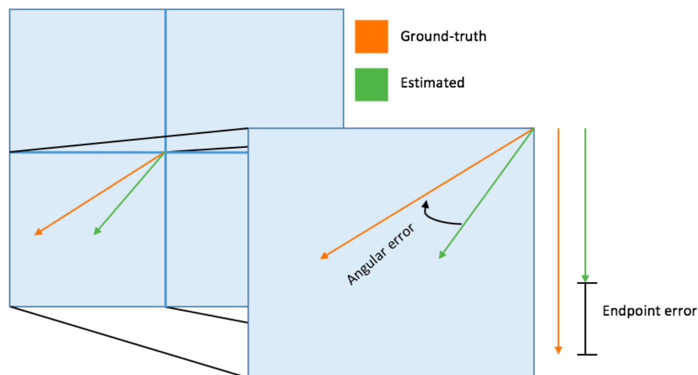
Filen **BlockMatchingAlgoMPEG** ble lastet ned hvor mappen inneholdt de nødvendige MATLAB-filene som kreves for å utføre sammenligningene [10]. MATLAB-koden som er vist under listing 2.2 er hentet fra [10] og viser hvordan 2.12 og 2.13 blir beregnet, i dette eksempelet for metoden ES. Først blir forflyttelsesvektorene estimert for BM-algoritmen. Estimatet skal deretter kvalitetsskjekkes ved at et nytt bilde rekonstrueres. Dette blir gjort ved hjelp av originalbildet og de estimerte vektorene. Ved å se på differansen mellom det rekonstruerte bildet og bildet som ble brukt for å estimere vektorene, vil en få et forhold som representerer kvaliteten på estimatet (PSNR).

Listing 2.2: Oppsettet for de ulike BM-algoritmene - hentet fra filen *motionEstAnalysis.m* [10].

```
1 % Exhaustive Search
2     [motionVect, computations] = motionEstES(imgP,imgI,mbSize,p);
3     imgComp = motionComp(imgI, motionVect, mbSize);
4     ESpsnr(i+1) = imgPSNR(imgP, imgComp, 255);
5     EScomputations(i+1) = computations;
```

2.3.3 Nøyaktighet

Angular error (AE) og *endpoint error* (EE) er to utbredte metoder som brukes for å evaluere nøyaktigheten til de estimerte forflyttelsesvektorene [23](s.16), vist ved figur 2.14.



Figur 2.14: Illustrerende figur for metodene *Angular error* (AE) og *endpoint error* (EE).

I denne rapporten har en valgt å se bort ifra AE fordi denne metoden behandler bevegelser i forskjellige områder ulikt [23](s.16). EE er et mål på differansen mellom de estimerte forflyttelsesvektorene og den "*sanne forflyttelsen*" (ground truth), gitt ved

$$EE = \sqrt{(u - u_{GT})^2 + (v - v_{GT})^2} \tag{2.14}$$

hvor (u, v) og (u_{GT}, v_{GT}) er de estimerte ground truth-vektorene. Ground truth-data er ikke perfekt [23](s.10), men i sammenheng med sammenligning vil det kunne gi en god indikasjon på hvordan de ulike metodene klarer seg i forhold til hverandre. I denne rapporten er evalueringsdata fra middlebury benchmark [1] benyttet i de tilfeller hvor dataene inneholder ground truth-informasjon. Bildesekvensene som er hentet fra [1] og som er brukt under evalueringen er RubberWhale, Hydrangea og Dimetrodon, vist ved figur 2.15. Bildesekvensene inneholder skjulte teksturer.

Figur 2.15: Bildesekvensene: RubberWhale, Hydrangea og Dimetrodon.

2.4 Teknisk informasjon

I denne rapporten har en ikke benyttet seg av fysiske droner til å hente inn videoer. Rapportens resultater er bygget på videoer fra internett som inneholder informasjon om den virkelige forflyttelsen. Teknisk informasjon om material og programvare er listet opp nedenfor for gjenskaping av resultat.

PC - MacBook Air (13-inch, Mid 2012)

- Prosessor: 2 GHz Intel Core i7
- Minne: 8 GB 1600 MHz DDR3
- Grafikk: Intel HD Graphics 4000 1536 MB

Program - MATLAB

- Version: R2016a

3. Implementering

I denne delen av rapporten vil teknisk informasjon gjennomgås. Det vil foreligge informasjon om utstyr, teknikker og programvare. Først vil ulike teknikker for implementering av metoden i MATLAB presenteres. Deretter vil implementeringen av evalueringsmetoden for nøyaktighet gjennomgås.

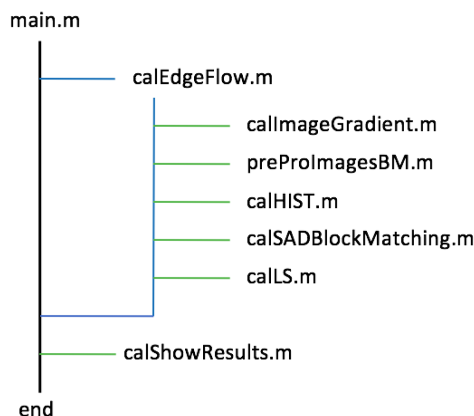
3.1 MATLAB

Når et program, funksjon eller metode skal implementeres så kan ulike teknikker tas i bruk. En god praksis er å lage en overordnet skisse av metoden. Skissen skal være en enkel representasjon av metodens hovedelementer, fra start til slutt. Dette vil gi et enkelt, men klart bilde av metodens oppbygning og funksjonalitet. Videre er det viktig å bygge opp metoden på en ryddig og enkel måte, slik at *neste bruker* enkelt kan sette seg inn i programmets oppbygning. For å lage et ryddig og oversiktlig program, vil beskrivende tekst samt oppdeling i mindre funksjoner tas i bruk. Det vil bidra til å forebygge tiden en ville ha brukt under testing. Det vil også gi mulighet til å kunne bytte ut eksisterende funksjoner.

I denne rapporten er algoritmen *calEdgeFlow* implementert i MATLAB, hvor den overordnede skissen av metoden er vist ved vedlegg A. Skissen er et resultat av informasjonen som er gitt om metodens oppbygning i [11] og [12]. Metoden inneholder hovedelementene som ble presentert i 2.1 *Datasyn*.

3.1.1 Mappestruktur

Når metoden skal implementeres i MATLAB, er det viktig å tenke på hvordan mappestrukturen skal settes opp. Påliteligheten til programmet kan styrkes ved å bruke tid på å lage en konstruktiv mappestruktur. Det finnes mange forskjellige måter å sette dette opp på. I denne rapporten er mappestrukturen satt opp som vist ved figur 3.1. Strukturen bygger på at man har en hovedfil *main.m* hvor de overordnede operasjonene behandles. I denne rapporten vil det dreie seg om å hente ut to bilder fra en videosekvens, for deretter å kalle på metoden *calEdgeFlow.m* og presentere resultatet. I filen *calEdgeFlow.m* vil funksjonene knyttet til metoden bli kalt på for å utføre nødvendige utrekninger og operasjoner. Oppsettet av mappestrukturen bygger på at man ikke skal utføre noen beregninger i hovedfilen, men i funksjoner en kaller på. Samtidig ønsker en ikke å bevege seg for langt bort fra hovedfilen.



Figur 3.1: Mappestrukturen til metoden calEdgeFlow i MATLAB.

3.1.2 Implementering av *calEdgeFlow.m*

Mappestrukturen til *calEdgeFlow* ble presentert i figur 3.1. Metoden er oversiktlig, og hoveddeler som inngår er implementert i egne funksjoner. Det gir muligheten til å teste funksjonene hver for seg, før en setter alt sammen. Det vil også være enklere å oppdatere/bytte ut de ulike funksjonene i etterkant.

Under implementering og testing ble matrisene som er vist under listing 3.1 og 3.2 tatt i bruk. Matrisene skal representere to bilder hvor en har en forskyvning på to piksler mot høyre. Når en skal teste programmer er det viktig å gjøre dette under regulerte forhold. Metoden *calEdgeFlow* benytter seg av en del matriseregning, og vanlige bilder har ofte en stor dimensjon. Det kan derfor ofte bli vanskelig å holde styr på hva som blir utført i de ulike programdelene. Ved å benytte seg av disse bildene vil en kunne navigere seg igjennom programmet på en enkel og ryddig måte, hvor en kan sjekke at summeringer og søkemetoder fungerer slik de skal.

Listing 3.1: Bilde I1 er brukt under testing av metoden *calEdgeFlow*.

```
1  %% This image is just used for
2  %% testing purposes: I(t-1)
3  I1 = [1 2 3 4 5 6 7 8 9 10;...
4        1 2 3 4 5 6 7 8 9 10;...
5        1 2 3 4 5 6 7 8 9 10;...
6        1 2 3 4 5 6 7 8 9 10;...
7        1 2 3 4 5 6 7 8 9 10;...
8        1 2 3 4 5 6 7 8 9 10;...
9        1 2 3 4 5 6 7 8 9 10;...
10       1 2 3 4 5 6 7 8 9 10;...
11       1 2 3 4 5 6 7 8 9 10;...
12       1 2 3 4 5 6 7 8 9 10;];
```

Listing 3.2: Bilde I2 er brukt under testing av metoden *calEdgeFlow*.

```
14 %% This image is just used for
15 %% testing purposes: I(t)
16 I2 = [11 11 1 2 3 4 5 6 7 8;...
17       11 11 1 2 3 4 5 6 7 8;...
18       11 11 1 2 3 4 5 6 7 8;...
19       11 11 1 2 3 4 5 6 7 8;...
20       11 11 1 2 3 4 5 6 7 8;...
21       11 11 1 2 3 4 5 6 7 8;...
22       11 11 1 2 3 4 5 6 7 8;...
23       11 11 1 2 3 4 5 6 7 8;...
24       11 11 1 2 3 4 5 6 7 8;...
25       11 11 1 2 3 4 5 6 7 8;];
```

Når metoden *calEdgeFlow* kalles på, vil den motta to bilder fra en videosekvens. Metoden vil også trenge noen parametre. Parametrene den vil trenge er størrelsen på blokkene *(s x s)* som bilde I(t-1) skal deles opp i, samt det maksimale søkeområdet i bildet I(t). I første omgang blir den deriverte i henholdsvis x- og y-retning beregnet, i MATLAB-filen *calImageGradient.m*. Dette blir gjort ved bruk av sobel-kjernen som ble presentert i teorien, figur 2.2. Et eksempel er vist ved figur B.1. Bildene går så igjennom en for-prosessering i MATLAB-filen *preProImagesBM.m* hvor bildene blir delt opp i mindre blokker, vist ved figur B.2. Filen er på mange måter todelt hvor den i første omgang sjekker hvor mange blokker bildet kan deles opp i. Hvis dimensjonene på bildene og blokkene ikke går opp, vil deler av bildene bli utelukket, deretter nedover samt mot høyre. Ved å gå igjennom denne prosessen vil koden bli mer fleksibel og robust med tanke på ulike størrelser på bildene/blokkene. Etter at dimensjonene på bildene er sjekket opp imot størrelsen på blokkene, vil prosessen ved å dele opp bildet starte. Et eksempel fra kode-filen *preProImagesBM.m* er vist under listing: 3.3. Her hentes den nye dimensjonen på bildet. Det blir så sjekket hvor mange blokker bildet vil kunne deles opp i. Til slutt brukes MATLAB-funksjonen **mat2cell** for å konvertere bildet I*(m x n)* til blokker av størrelsen *(s x s)*.

Listing 3.3: MATLAB(*preProImagesBM.m*): Deler opp bildet i mindre blokker.

```
37 %% Get the new dimensjon of the Image
38 [tempRow, tempCol, ~] = size(tempGx);
39
40 % Find the max number of blocks in horisontal and vertical direction
41 numOfRowBlocks = floor(tempRow/blockSize);
42 numOfColBlocks = floor(tempCol/blockSize);
43
44 %% Divaide image_x I(t-1) into (s x s) blocks
45 rowDim = blockSize*ones(1,numOfRowBlocks);
46 % rowDim = [r r]
47 colDim = blockSize*ones(1,numOfColBlocks);
48 % colDim = [c c]
49
50 pre_I1x = mat2cell(tempGx, rowDim, colDim)';
51 %  I(n x m) => [r x c][r x c]
52 %             [r x c][r x c]
53
54 pre_I1x = pre_I1x(:);
55 %  I(n x m) => [r x c]
56 %             [r x c]
57 %             [r x c]
58 %             [r x c]
```

Det neste steget i algoritmen er å lage histogram av de oppdelte bildene. Dette blir gjort ved å summere sammen verdiene i de ulike retningene. Det blir utført i MATLAB-filen *calHIST.m* og resultatet av dette er vist ved figur B.3.

Det er nå klart for å utføre blokk-matching, hvor SAD blir brukt for å estimere forskyvningen av de ulike blokkene i bildet. For å forstå hvilke utfordringer som kan forekomme er det viktig å sette seg inn i hvordan metoden fungerer. Det er derfor gitt et enkelt eksempel *Blokk matching 1D* med en illustrerende figur 3.2. I tillegg er en del av koden hentet ut fra MATLAB-filen *calSADBlockMatching.m* vist under listing 3.4. Fra eksempelet ser vi at under første søk, vil startposisjonen (Index = 0) ende opp utenfor bildet. Dette er noe som vil skje rundt alle kantene i bildet. Det er ulike teknikker som kan tas i bruk for å hindre/løse dette problemet. I denne rapporten ble dette løst ved å sette verdien **Inf** hvis posisjonen skulle ende opp utenfor bildet. Det betyr at det ikke vil forekomme noen form for matching i dette området. I listing 3.4 blir dette sjekket på kodelinje 59. De neste søkene er innenfor bildet og SAD blir utført. Dette blir utført på linje 62-63 i listing 3.4. Søket som har den laveste verdien matcher best og blir lagret. Deretter blir neste blokk hentet, og et nytt søk utføres. I første omgang kalkuleres forskyvningen i horisontal retning, og deretter i vertikal retning. Resultatet av å kalle på funksjonen *calSADBlockMatching.m* er vist ved figur B.4. Til slutt blir det laget en lineær modell av forskyvningene i MATLAB-filen *calLS*, hvor en global forflyttelse blir kalkulert ved hjelp av minste kvadraters metode.

Listing 3.4: MATLAB(*calSADBlockMatching.m*): Utfører blokk-matching.

```
47  % Get match block from I(t-1)
48  matchBlock = hist_previous{counterMatchBlocks};
49  % Get search block I(t)
50  searchBlock = hist_current{counterSearchBlocks};
51
52  for counterSR = -searchRange:searchRange
53
54          % Set the starting point and the end point in I(t)
55          index1 = 1+counterSR + jumperBlock*(sizeWindow);
56          index2 = counterSR + (1+jumperBlock)*sizeWindow;
57
58          % Check if the search area is outside of the image
59          if index1 <= 0 || index2 > length(searchBlock)
60              SAD_temp(counterSR+searchRange+1) = Inf;
61          else
62              SAD_temp(counterSR+searchRange+1) = ...
63                 sum(abs(matchBlock - searchBlock(index1:index2)));
64          end
65          % Count the number of computations
66          computations = computations + 1;
67  end
```

Blokk-matching 1D

Dette er et enkelt eksempel på hvordan blokk-matching fungerer i calEdgeFlow. De tidligere stegene i algoritmen er utført og situasjonen er nå som vist på figur 3.2. Blokk nr 1 er nå hentet ut ifra henholdsvis bilde I(t-1) og I(t). Størrelsene på blokkene er satt til (4 x 4) og det maksimale søkeområdet er satt til én piksel. Utifra dette kan en rekne oss ut til at det vil bli utført totalt 3 søk.

Søk nr. 1:
- Start pos. = 0
- Slutt pos. = 3

$$\text{INF (INDEX OUT OF BOUNDS!)} \tag{3.1}$$

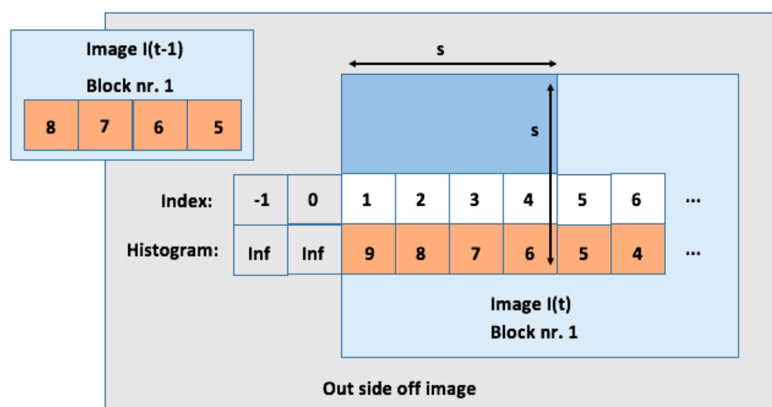
Søk nr. 2:
- Start pos. = 1
- Slutt pos. = 4

$$|(8 + 7 + 6 + 5) - (9 + 8 + 7 + 6)| = 4 \tag{3.2}$$

Søk nr. 3:
- Start pos. = 2
- Slutt pos. = 5

$$|(8 + 7 + 6 + 5) - (8 + 7 + 6 + 5)| = 0 \tag{3.3}$$

Resultat:
Det søket som har den laveste verdien matcher best. Utifra resultatet kan en se at bildet har forskjøvet seg 1 piksel.



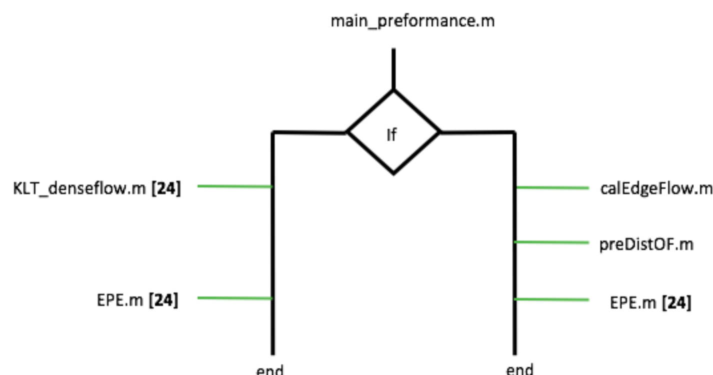
Figur 3.2: Et enkelt eksempel på hvordan blokk-matchingmetoden fungerer.

3.2 Nøyaktighet

For å kunne evaluere den implementerte metoden ble bilder, ground truth-data og nødvendige MATLAB-filer fra middlebury benchmark [1] lastet ned. Middlebury benchmark bruker farger for å representere den optiske flyten i bildet, vist ved figur 3.3. Evalueringsresultat av ulike optiske flytmetoder er tilgjengelig på Middlebury benchmark [1]. Ground truth-dataene til de evaluerte metodene er ikke tilgengelige, og en må da sende inn metoden for evaluering. Det er noen testsekvenser hvor ground truth-data foreligger, hvor en har mulighet for å selv evaluere metoden. Metoden Lucas-Kanade ble derfor inkludert. Denne metoden ble også brukt i [11] for å sammenligne metoden *EdgeFlow*, hvor metoden ble presentert. Metoden som er brukt i denne rapporten er hentet fra [24] Kanade-Lucas-Tomasi, hvor den enkleste formen ble brukt. I dette ligger det at man har mulighet til å sette ulike parametre for å tilpasse metoden. I denne rapporten er metoden kun implementert i sammenligningssammenheng.

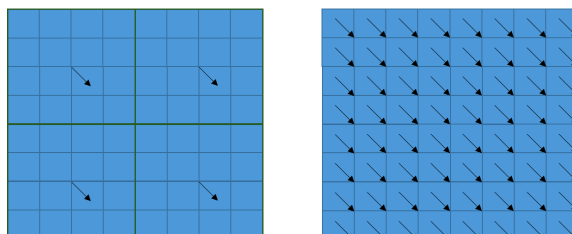
Figur 3.3: Fra venstre: Et av bildene som skal evauleres, ground truth og fargekartet til den optiske flyten.

Figur 3.4 viser algoritmen som ble brukt for å evaluere metoden. Fra hovedfilen *main_preformance.m* har en mulighet for å kjøre metodene *calEdgeFlow.m* og *KLT_denseflow.m* [24]. Evalueringen bygger på metoden *endpoint error* (EE), som sammenligner differansen mellom de estimerte forflyttelsesvektorene med ground truth-dataen. Det blir gjort i MATLAB-filen *EPE.m* [24].



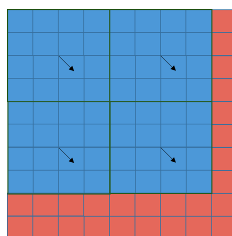
Figur 3.4: Algoritmen for evaluering av *endpoint error* (EE).

Metoden *KLT_denseflow.m* estimerer forflyttelsesvektorer for hver piksel i bildet (tett), og det foreligger også for ground truth-data. Den implementerte metoden estimerer en forflyttelsesvektor for et større område (sparsom), vist ved figur 3.5.



Figur 3.5: Sparsom og tett optisk flyt.

Dette gjør at resultatet for *calEdgeFlow.m* inneholder mindre informasjon, som derfor ikke vil kunne sammenliknes med ground truth-dataene. I denne rapporten ble det løst ved å sette den samme estimerte forflyttelsesvektoren i hver piksel for den representerte blokken, *preDistOF.m*. Bevegelse i mindre områder har en tendens til å bevege seg i samme retning og på denne måten vil en få en optisk flyt-vektor for hver piksel. Videre er metoden fleksibel, noe som betyr at *calEdgeFlow.m* prøver å dele opp bildet i x antall blokker, der resterende vil bli utelukket, vist ved figur 3.6. Tabell 3.1 lister opp teknisk informasjon for de ulike bildesekvensene. Bildene er av samme størrelse (*584 x 388*), og tabell 3.2 viser hvor stor del av bildet som blir utelukket ved ulike blokkstørrelser. Dette påvirker resultatet når metoden *calEdgeFlow.m* skal evalueres, og resultatet vil inneholde noe mindre informasjon sammenlignet med *KLT_denseflow.m*. Det utelukkede området vil ikke kunne overskride størrelsen som blir satt på blokkene *(s x s)*, som er vist ved det røde området på figur 3.6.



Figur 3.6: Deler av bildet blir utelukket ved oppdeling i mindre blokker.

Tabell 3.1: Informasjon om bildesekvensene fra middlebury benchmark [1].

Bildesekvens	**Antall bilder**	**Fargekode**	**Format**	**Maks. Flow**
RubberWhale	2	Grå	584×388	4.62
Hydrangea	2	Grå	584×388	11.12
Dimetrodon	2	Grå	584×388	4.67

Tabell 3.2: Inneholder informasjon om hvordan ulike blokkstørrelser påvirker resultatet av *calEdgeFlow.m*.

Bilde Størrelse (584 x 388)
Antall piksler
226 592

Blokkstørrelse (12 x 12)		**Blokkstørrelse (14 x 14)**		**Blokkstørrelse (16 x 16)**	
Antall piksler	**%**	**Antall piksler**	**%**	**Antall piksler**	**%**
5 408	2.39	9 620	4.25	5 408	2.39

Blokkstørrelse (18 x 18)		**Blokkstørrelse (20 x 20)**		**Blokkstørrelse (22 x 22)**	
Antall piksler	**%**	**Antall piksler**	**%**	**Antall piksler**	**%**
8 864	3.91	6 192	2.73	12 664	5.59

4. Resultat

Delen av prosjektet som omhandlet datasyn ble delt inn i tre ulike stadier. Det første trinnet gikk ut på å implementere metoden *calEdgeFlow* i MATLAB. Metoden skulle så evalueres med hensyn til ytelse. Dette ble gjort ved hjelp av ulike innebygde metoder som finnes i MATLABs biblioteker. Det ble så utført en kvalitetssjekk av estimeringsresultatene til BM-delen av den implementerte metoden. Til slutt skulle metoden evalueres med hensyn til nøyaktighet. Tanken bak de ulike stadiene var å få et innblikk i utfordringene og begrensningene som kan påløpe i de ulike prosessene.

4.1 Evaluering av *calEdgeFlow* - Ytelse

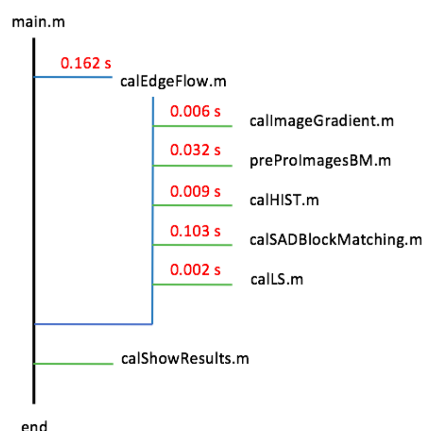
Metoden *calEdgeFlow* er nå implementert i MATLAB og det er på tide å analysere ytelsen til metoden. For å kunne analysere metoden trengs det to bilder hvorav det ene er forskjøvet. Det er derfor tatt utgangspunkt i eksempelet i vedlegg B, hvor bildet er forskjøvet fem piksler mot høyre. Blokkstørrelsene er nå endret fra *100 x 100* til *18 x 18* og søket på 10 piksler forblir det samme. Programmet analyseres ved å aktivere funksjonen *Profiler* i MATLAB. Resultatet etter analysen er vist ved figur 4.1, hvor tidene er satt opp i mappestrukturen for å vise hvor i metoden det blir brukt mest tid. Ut ifra resultatet kan en se at funksjonen *calSADBlockMatching* tar opp 63,5 % av tiden. Dette var forventet da det er i denne funksjonen det blir utført flest utregninger. Det vil derfor være naturlig å gå igjennom denne funksjonen.

Profile Summary

Generated 09-May-2017 21:47:39 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	0.168 s	0.006 s	
calEdgeFlow	1	0.162 s	0.010 s	
calSADBlockMatching	2	0.103 s	0.103 s	
preProImagesBM	1	0.032 s	0.013 s	
mat2cell	4	0.019 s	0.019 s	
calHIST	1	0.009 s	0.009 s	
calImageGradient	2	0.006 s	0.006 s	
calLS	2	0.002 s	0.002 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.



Figur 4.1: Rapport fra MATLAB *Profiler*.

I forkant av analysen ble det utført noen forbedringer. Det ble utført en analyse hvor hver funksjon ble gjennomgått med tanke på avsnittet *Code Analyzer results*, i den detaljerte *Profiler* rapporten. Denne delen av *Profiler* rapporten tar for seg forslag til endringer som kan forbedre og øke hastigheten til funksjonene. Figur 4.2 viser hva som kan bli gjort for å effektivisere funksjonen *calHist* med tanke på hastighet.

Code Analyzer results

Line number	Message
33	The variable 'I1_GxHist' appears to change size on every loop iteration. Consider preallocating for speed.
39	The variable 'I1_GyHist' appears to change size on every loop iteration. Consider preallocating for speed.
45	The variable 'I2_GxHist' appears to change size on every loop iteration. Consider preallocating for speed.
51	The variable 'I2_GyHist' appears to change size on every loop iteration. Consider preallocating for speed.

Figur 4.2: Detaljert *Profiler* rapport av funksjonen *calHist*: **Code Analyzer results**.

I funksjonen *calHist* skal de oppdelte bildene konventeres til histogrammer. Dette ble gjort i en sløyfe som er vist ved listing 4.1. Problemet rundt denne løsningen er at **I1_GxHist**, som skal holde på histogrammene, blir større for hver runde som går. Det betyr at MATLAB leter etter nye plasser i minnet for å lagre den voksende vektoren **I1_GxHist**. Ved å sette størrelsen på **I1_GxHist** før en begynner å legge til histogrammene vist ved listing 4.2, så vil MATLAB i forkant dedikere plass i minnet som vil redusere tiden en bruker på å dedikere nye plasser.

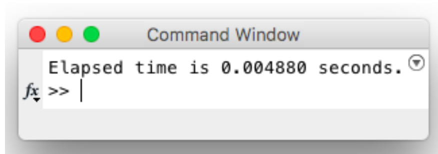
Listing 4.1: Funksjonen *calHist.m* før endringer.

```
31      tic
32
33      for i = 1:numBlocks_x
34          I1_GxHist{i} = ...
35                  sum(pre_I1x{i});
36      end
37      toc
```

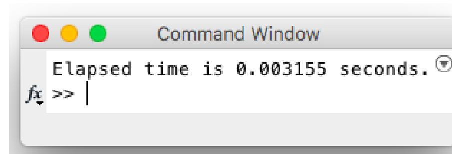
Listing 4.2: Funksjonen *calHist.m* etter endringer.

```
31      tic
32      I1_GxHist = cell(1,numBlocks_x);
33      for i = 1:numBlocks_x
34          I1_GxHist{i} = ...
35                  sum(pre_I1x{i});
36      end
37      toc
```

MATLAB-kommandoene **tic** og **toc** kan brukes for å finne ut hvor lang tid ulike deler av programmet bruker. De kan samenliknes med en stoppeklokke hvor **tic** er start og **toc** er stopp [19]. Figur 4.3 og 4.4 viser resultatet av å kjøre MATLAB-koden som er vist i listing 4.1 og 4.2. Ut ifra resultatet kan vi se at tiden det tar å kjøre igjennom koden nå har blitt redusert med rundt 35%.



Figur 4.3: Resultat av å kjøre listing 4.1.



Figur 4.4: Resultat av å kjøre listing 4.2.

Som nevnt tidligere så tar funksjonen *calSADBlockMatching* opp ca. 63,5 % av tiden til funksjonen *calEdgeFlow*. Det betyr at det er denne funksjonen en burde gå igjennom for å sjekke om det er mulighet for å gjøre noen endringer. Ved å åpne den detaljerte rapporten til *calSADBlockMatching* og se på seksjonen *Lines where the most time was spent"*, vist ved figur 4.5, vil en få en oversikt over de kodelinjene som tar opp mest tid. Linje 63 tar opp 74,4% av tiden og det er under søkeprosessen hvor blokker fra hvert bilde sammenlignes. Videre kan en lese ut at denne linjen blir kjørt totalt 32 256 ganger. De andre linjene utgjør en så liten del av det totale bildet og trenger ikke å sees på i første omgang.

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
63	sum(abs(matchBlock - searchBlo...	32256	0.077 s	74.4%	
70	[value,index] = min(SAD_temp);	1568	0.007 s	6.5%	
67	end	32928	0.003 s	2.9%	
59	if indexS <= 0 || indexE &g...	32928	0.002 s	2.1%	
55	indexS = 1+counterSR + jumperB...	32928	0.001 s	1.4%	
All other lines			0.013 s	12.6%	
Totals			0.103 s	100%	

Figur 4.5: Detaljert rapport for funksjonen *calSADBlockMatching* i seksjonen *Lines where the most time was spent"*.

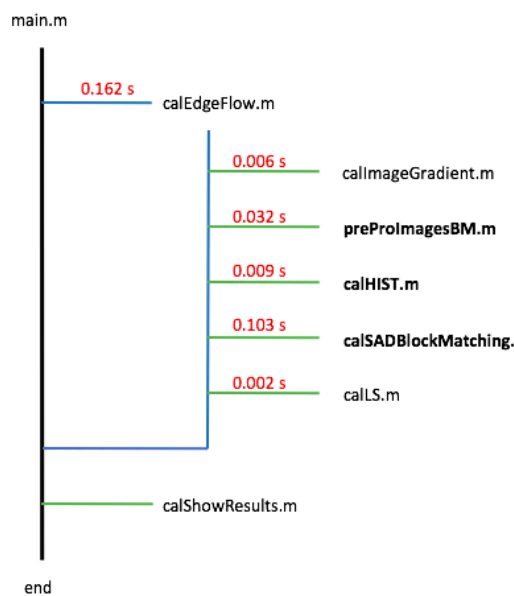
Det neste steget vil være å studere den nedre delen av den detalijerte rapporten. Et utdrag fra denne er vist ved figur 4.6, hvor kodelinjen som tar opp mest tid er merket med mørkrød farge. Kodelinjen ligger inne i en sløyfe som kjøres 1 568 ganger. Denne verdien representerer antall blokker som skal sjekkes. Videre skal det utføres 32 928 søk hvorav 672 ligger utenfor bildet. Det betyr at 32 356 søk ble utført innenfor bildet og representerer da linje 63. Kodelinjen har en total tid på rundt 0,08s fordelt på 32 356 søk. Ut ifra disse opplysningene falt valget for å ikke utføre noen endringer.

```
< 0.01    1568   52   for counterSR = -searchRange:searchRange
                 53
                 54       % Set the starting point and the end point in I(t)
< 0.01   32928   55       index1 = 1+counterSR+jumperBlock*(sizeWindow);
< 0.01   32928   56       index2 = counterSR+(1+jumperBlock)*sizeWindow;
                 57
                 58       % Check if the search area is outside of the image
< 0.01   32928   59       if index1<=0 || index2>length(searchBlock)
< 0.01     672   60           SAD_temp(counterSR+searchRange+1) = Inf;
< 0.01   32256   61       else
  0.08   32256   62           SAD_temp(counterSR+searchRange+1) = ...
         32256   63                       sum(abs(matchBlock-searchBlock(index1:index2)));
< 0.01   32256   64       end
                 65       % Count the number of computations
< 0.01   32928   66       computations = computations+1;
< 0.01   32928   67   end
```

Figur 4.6: Detaljert rapport for funksjonen *calSADBlockMatching* i seksjonen *Function listing"*.

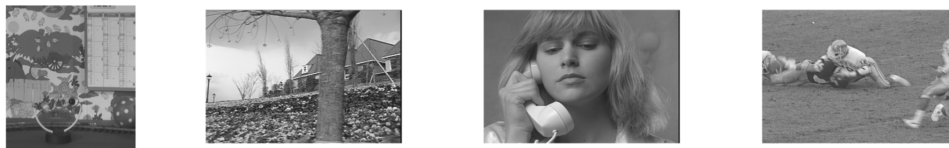
4.2 Sammenligning av BM-algoritmer

BM er metoden som brukes for å estimere forflyttelsen som forekommer i bildet. Ut ifra analysen som ble utført med MATLAB *Profiler*, var det denne delen av algoritmen som brukte mest tid. I *calEdgeFlow* er det tre funksjoner som brukes for å estimere forflyttelsen i bildet. I *preProImageBM.m* blir bildene delt opp i mindre blokker, som deretters summeres sammen til blokkhistogrammer, *calHIST.m*. Blokkhistogrammene blir så sammenlignet i *calSADBlockMatching.m*. Funksjonene er fremhevet i figur 4.7.



Figur 4.7: Mappestrukturen til *calEdgeFlow*, hvor de fremhevede funksjonene representerer BM-delen av algoritmen.

Fra en videosekvens blir det tatt ut to bilder med et mellomrom på to bilder. I denne rapporten ble sekvensene *caltrain* [2], *garden* [2], *susie* [2] og *football* [2] brukt. Videosekvensene er listet opp i tabell 4.1, hvor figur 4.8 viser et bilde fra hver av dem.



Figur 4.8: Et bilde fra hver videosekvens som er listet opp i tabell 4.1

Tabell 4.1: Informasjon om de ulike videosekvensene som er brukt i rapporten, hentet fra [2].

Videosekvens	**Format**	**Antall bilder**
caltrain	512x400	33
garden	352x240	61
susie	352x240	75
football	352x240	60

Resultatene etter simulering er listet opp i tabellene 4.2, 4.3 og 4.4. Resultatene av å simulere videosekvensen *caltrain* er vist grafisk ved figurene 4.10, 4.9 og 4.11. Tabell 4.2 viser kvaliteten på estimatene, hvor en høyere verdi representerer et bedre estimat [25]. I denne analysen havnet den integrerte metoden i nedre del av grafen. Algoritmen kom dårligst ut der den endet opp med rundt 9% estimeringsfeil i forhold til ES. Algoritmen ARPS kom best ut, som den også gjorde i [10], tett etterfulgt av NTSS. Resultatene som er listet opp i tabell 4.3 representerer et gjennomsnittsmål på antall søk utført for hver blokk. Metoden som er integrert i denne rapporten havner øverst på den fremhevede grafen, figur 4.10. Metoden vil alltid utføre det samme antall søk for hver blokk, og i denne simuleringen var søkeområdet satt til syv piksler, som resulterer i 30 søk per blokk. Det ble også utført tidsmålinger på de ulike BM-algoritmene. Til å måle tiden ble MATLAB-funksjonen **timeit** brukt. Oppsettet er vist ved listing 4.3 [26].

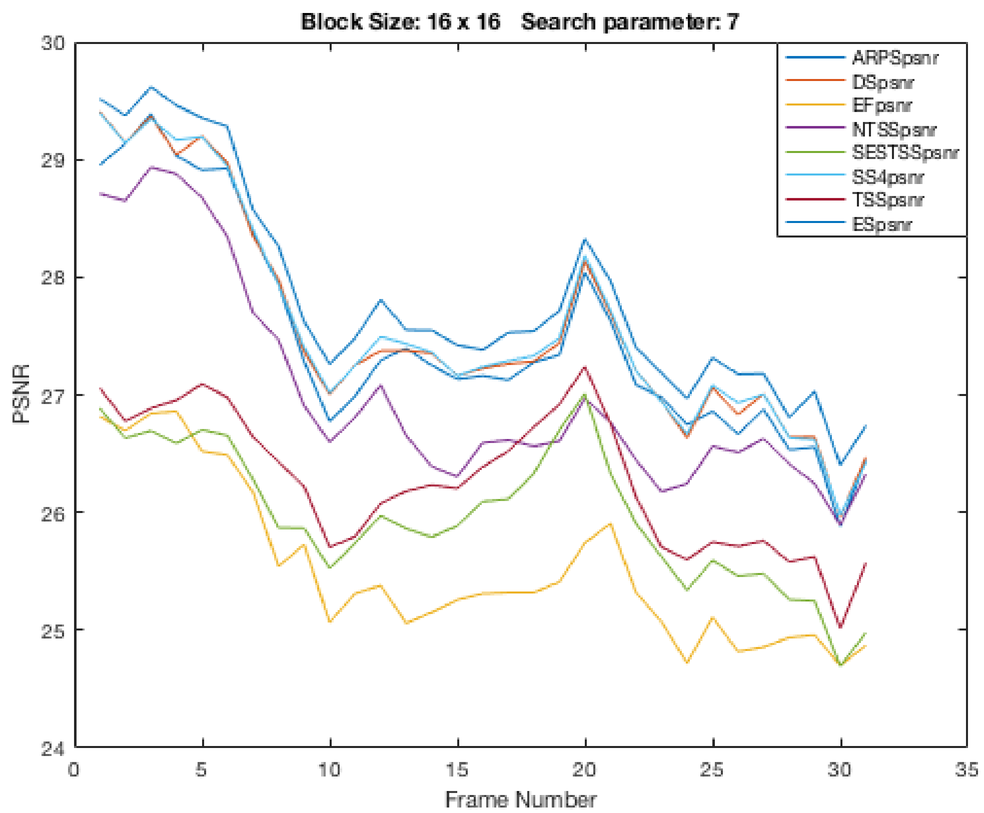
Listing 4.3: MATLAB-funksjonen *timeit*.

```
1 ESfunction = @() motionEstES(imgP,imgI,mbSize,p);
2 EStime(i+1) = timeit(ESfunction);
```

Tabell 4.4 viser resultatet av hvor lang tid de ulike algoritmene brukte. I denne grafen havnet den integrerte metoden nederst, som betyr at denne er den raskeste av de representerte algoritmene.

Tabell 4.2: Inneholder simuleringsresultater: PSNR.

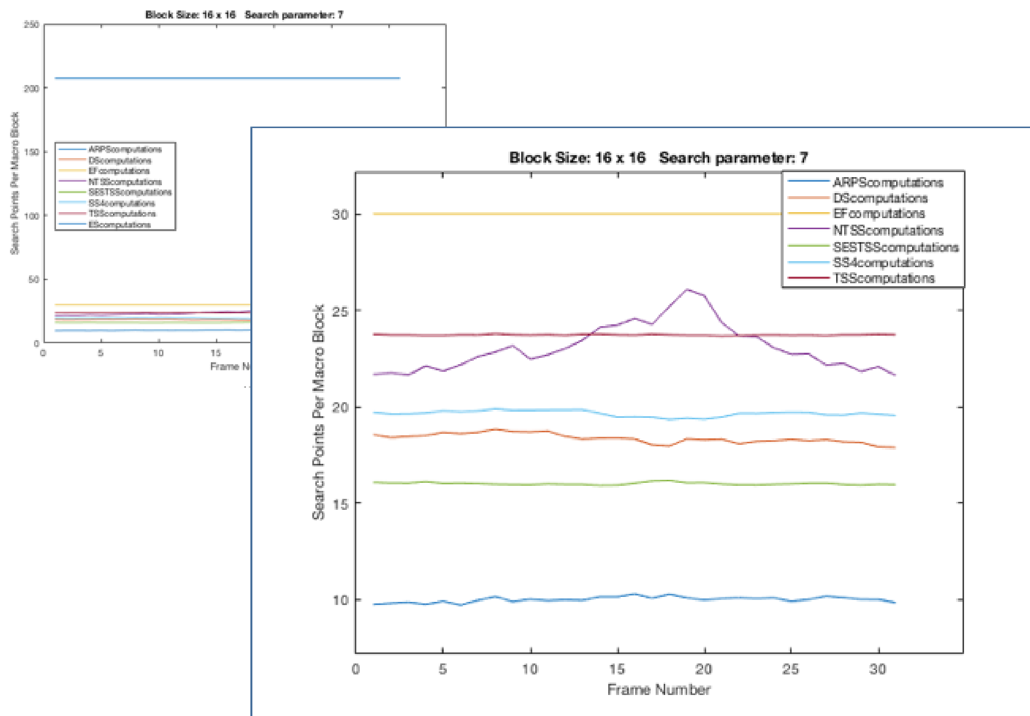
Algoritme	caltrain (Søkeparameter: 7)		garden (Søkeparameter: 7)		susie (Søkeparameter: 7)		football (Søkeparameter: 7)		Gjennomsnitt
	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	PSNR	D_{PSNR}	(D_{PSNR})
ES	27.83	0	21.07	0	31.85	0	20.5	0	0
TSS	26.27	-5.63	20.29	-3.72	31.05	-2.49	20.26	**-1.17**	-3.25
NTSS	27.02	-2.91	20.57	-2.39	31.55	**-0.92**	20.22	-1.35	-1.89
SES	25.97	-6.69	20.13	-4.47	30.21	-5.14	19.70	-3.86	-5.04
4SS	27.59	**-0.86**	19.83	-5.91	31.22	-1.96	20.09	-1.99	-2.68
DS	27.57	-0.93	19.57	-7.11	31.48	-1.16	20.04	-2.26	-2.87
ARPS	27.48	-1.25	20.57	**-2.35**	31.49	-1.13	20.04	-2.27	**-1.75**
calEdgeFlow	25.52	-8.29	18.83	-10.66	28.60	-10.18	19.17	-6.5	-8.91



Figur 4.9: Resultat: Kvalitetssjekk av estimatet ved bruk av metoden PSNR.

Tabell 4.3: Inneholder simuleringsresultater: Antall søk

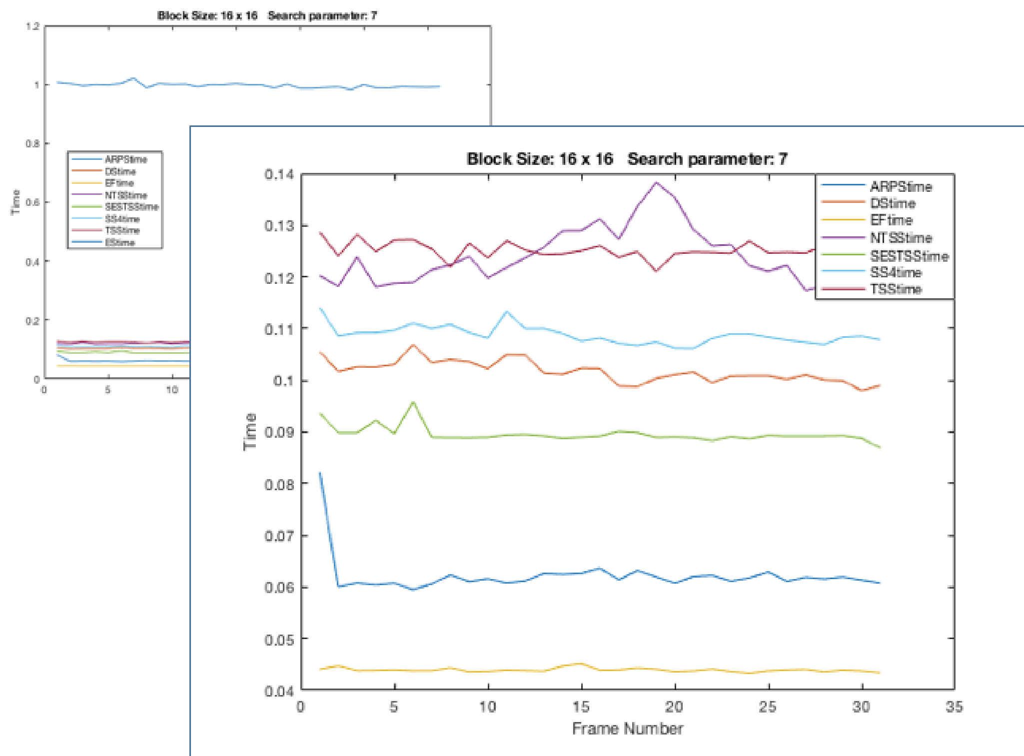
Algoritme	caltrain (Søkeparameter: 7) Antall søk	garden (Søkeparameter: 7) Antall søk	susie (Søkeparameter: 7) Antall søk	football (Søkeparameter: 7) Antall søk	Gjennomsnitt (Antall søk)
ES	207.41	202.05	202.05	202.05	203.39
TSS	23.73	23.31	23.15	23.14	23.33
NTSS	23.1	25.8	21.5	22.35	23.19
SES	16.02	15.45	16.04	15.95	15.87
4SS	19.65	20.25	18.84	19.12	19.47
DS	18.37	19.64	17.3	18.03	18.34
ARPS	**10.02**	**10.32**	**9.48**	**10.65**	**10.12**
calEdgeFlow	30	30	30	30	30



Figur 4.10: Resultat: Gjennomsnittsmål på antall søk som blir utført for hver blokk.

Tabell 4.4: Inneholder simuleringsresultater: Tid

Algoritme	caltrain (Søkeparameter: 7) Time	garden (Søkeparameter: 7) Time	susie (Søkeparameter: 7) Time	football (Søkeparameter: 7) Time	Gjennomsnitt (Time)
ES	1.0306	0.6041	0.6080	0.5791	0.7055
TSS	0.1219	0.0725	0.0720	0.0725	0.0847
NTSS	0.1220	0.0820	0.0687	0.0722	0.0862
SES	0.0886	0.0519	0.0538	0.0536	0.0620
4SS	0.1075	0.0674	0.0619	0.0630	0.0750
DS	0.1015	0.0658	0.0583	0.0616	0.0718
ARPS	0.0607	0.0378	0.0358	0.0411	0.0439
calEdgeFlow	**0.0438**	**0.0259**	**0.0263**	**0.0272**	**0.0308**

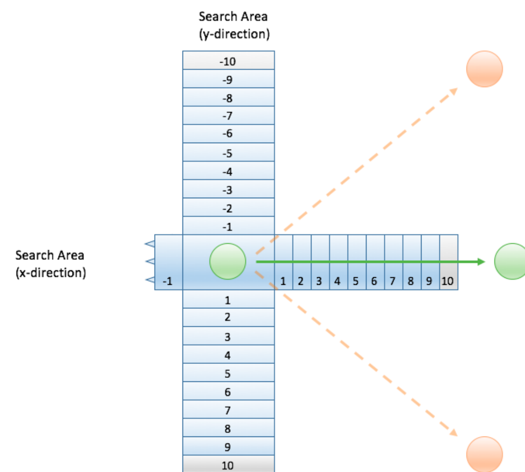


Figur 4.11: Resultat: Tiden det tar å utføre de ulike BM-algoritmene.

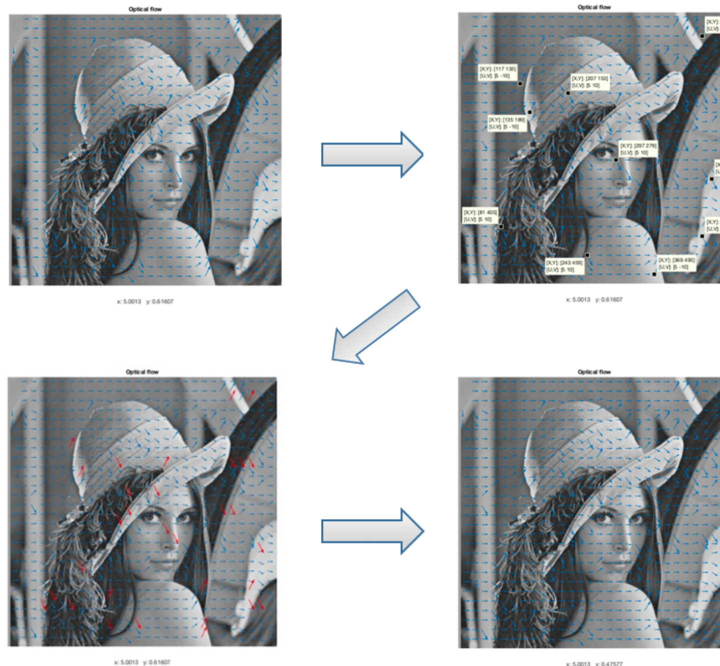
4.2.1 Forsøk med *threshold*

BM-delen av *calEdgeFlow* har noen svakheter når det gjelder metoden en bruker under søkeprosessen. Metoden søker kun i horisontal og vertikal retning, som vist ved figur 4.12. Den er ikke adaptiv og vil utføre det samme antall søk for hver blokk.

Figur 4.12 representerer to ulike situasjoner, vist ved de ulike fargene på sirklene. I situasjonen som er vist ved de oransje sirklene vil sannsynligheten for en perfekt match reduseres. Området som skal sammenlignes kan endres drastisk. Da den grønne situasjonen ble testet la en merke til at jo mer sirklen forskjøv seg i form av piksler, dess flere estimeringsfeil endte en opp med. Estimeringsfeilene hadde en tendens til å lande i endene på søkemetoden, som er markert med grå farge.



Figur 4.12: Søkeprinsippet for BM i *calEdgeFlow*.

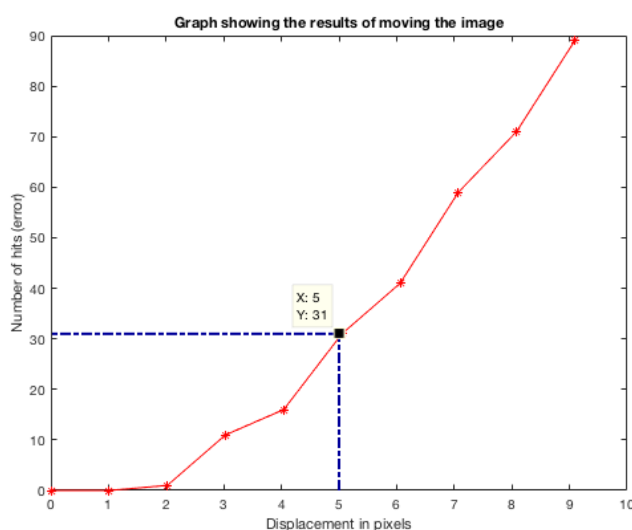


Figur 4.13: Bildet er forskjøvet 5 piksler mot høyre, og estimeringsfeil som legger seg i de ytterste indeksene er markert og til slutt fjernet.

Situasjonen som er vist ved figur 4.13 ble analysert, hvor parametersettingen var lik som i seksjonen 4.1. Bildet ble forskjøvet én piksel mot høyre, og estimeringsfeil som la seg på den ytterste indeksen ble lagt sammen. Tabell 4.5 viser resultatet av hvordan estimeringsfeilen øker og figur 4.14 viser situasjonen grafisk. En metode som ofte benyttes og som ikke krever drastiske endringer av metoden, går ut på å implementere en "*threshold*". Det innebærer at man setter en grense, og dersom en gå over eller under grensen, har en mulighet til å lage ulike utfall. En "*threshold*" ble integrert i funksjonen *calSADBkockMatching*, hvor verdien null ble satt om estimatet la seg på de ytterste indeksene, markert grå i figur 4.12. Resultatet av dette er vist i tabell 4.5.

Tabell 4.5: Tabellen inneholder resultater før og etter bruk av "*threshold*".

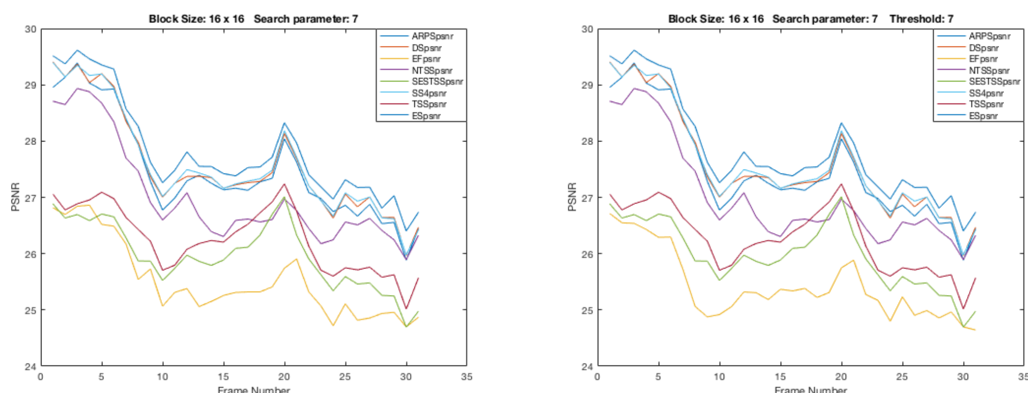
		Estimert Global Flyt (før)		**Estimert Global Flyt (etter)**	
Forskyvning (piksler)	**Antall treff (feil)**	**x-retning**	**y-retning**	**x-retning**	**y-retning**
0	0	0	0	0	0
1	0	0.98724	0.002551	0.98724	0.002551
2	1	2.0242	0.081633	2.0242	0.068878
3	11	3.0204	0.21939	3.0204	0.20663
4	16	4.014	0.43367	4.0013	0.33163
5	31	5.0013	0.61607	5.0013	0.47577
6	41	5.9949	0.60969	5.9949	0.54592
7	59	6.9885	0.65051	6.963	0.5102
8	71	7.8495	0.74872	7.8495	0.5574
9	89	8.801	0.7398	8.5204	0.625



Figur 4.14: Grafisk fremstilling for økningen av estimeringsfeil.

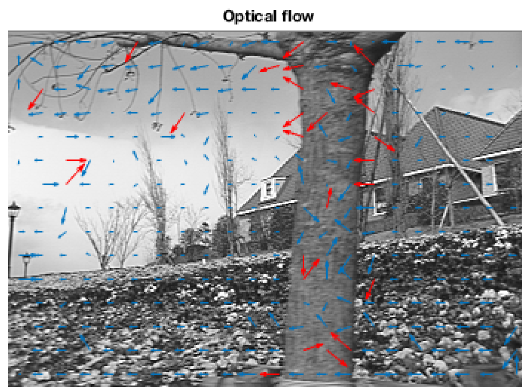
Ut ifra resultatet i tabell 4.5 kan en se at den horisontale retningen endrer seg minimalt. Ved "*threshold*" ender man opp med å klassifisere noen av vektorene som går horisontal som feil, og resultatet blir noe dårligere mot slutten av tabellen. Videre kan en se at det er en liten forbedring i den vertikale retningen. Resultatene bygger på den estimerte globale flyten i bildet. Ved å bruke "*threshold*" i denne søkemetoden, ender man opp med å bruke ca. 10% av søkeområdet som en form for kontroll. Det betyr at en går ifra 10-pikslet søk i hver retning til 9 piksler. Dette ble testet under regulerte forhold hvor det samme bildet ble forskjøvet.

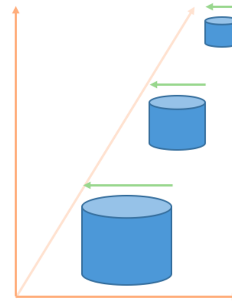
Det ble deretter utført en ny analyse av metoden, hvor parametrene var like som de under sammenligningen av BM-algoritmene. Figur 4.15 viser resultatet med og uten "*threshold*". Det var ingen betydelige endringer, og situasjonen hvor en brukte "*threshold*" kom noe dårligere ut.



Figur 4.15: Sammenligning av metoden uten/med *threshold* - PSNR - Videosekvensen *caltrain* ble brukt.

En grunn til at metoden kom noe dårligere ut kan skyldes at avstanden i bildet kan variere. I dette ligger det at objekter som er nærmere kamera vil forflytte seg over større områder av bildet sammenlignet med objekter som oppleves som langt ifra, vist ved en illustrerende figur 4.16. I figur 4.16 er det også hentet ut et bilde fra videosekvensen *garden*, hvor avstandene i bildet kommer klart frem. Treet som står forholdsvis nærme kamera vil oppleves som at det beveger seg fortere, og metoden med "*threshold*"vil i dette tilfellet feilklassifisere disse vektorene. Teksturene i treet byr også på utfordringer, da man i dette området ser at det forekommer mange estimeringsfeil. Tanken bak å bruke "*threshold*" var å finne feil som er vist ved vektoren i det hvite området (skyen), helt til venstre i figur 4.16. Området er utfordrende med tanke på informasjonen man finner i dette område, noe som bidrar til at en ofte ender opp med estimeringsfeil.

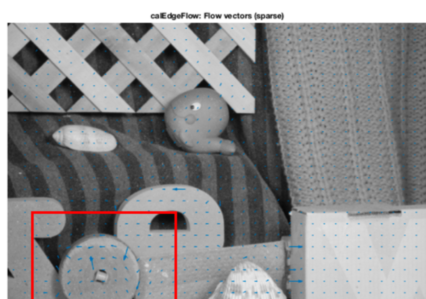




Figur 4.16: Bildene skal illustrere situasjonen hvor objekter i bildet har ulike avstander - Videosekvensen *garden* ble brukt.

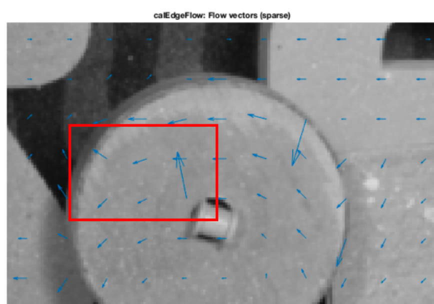
4.3 Evaluering av *calEdgeFlow* - Nøyaktighet

Metoden som er implementert i denne rapporten beregner en forflyttelsesvektor for et større område, vist ved figur 4.17(a). I denne rapporten har metoden blitt betegnet som sparsom, og for å kunne evaluere denne så ble det omgjort til en tett estimering, vist ved figur 4.17(b). Ved å studere disse bildene kan en se at dette har noen konsekvenser. Figurene 4.17(c)(e) inneholder en klar estimeringsfeil, som i 4.17(d)(f) blir satt over et større område, over hele den representerte blokken. Dette påvirker resultatet når evalueringen av *endpoint error (EE)* skal utføres.

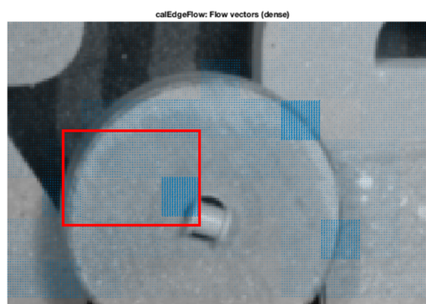


(a) *calEdgeFlow*: sparsom (nr. 1).

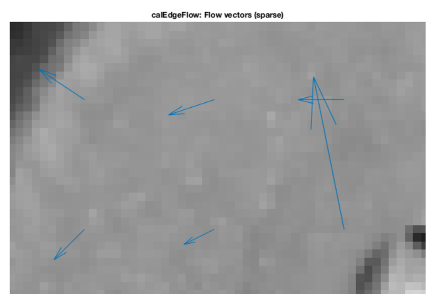
(b) *calEdgeFlow*: tett (nr. 1).



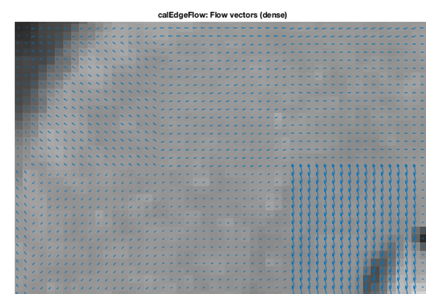
(c) *calEdgeFlow*: sparsom (nr. 2).



(d) *calEdgeFlow*: tett (nr. 2).



(e) *calEdgeFlow*: sparsom (nr. 3).



(f) *calEdgeFlow*: tett (nr. 3).

Figur 4.17: Studering av bildesekvensen *RubberWhale* med resultater fra *calEdgeFlow*.

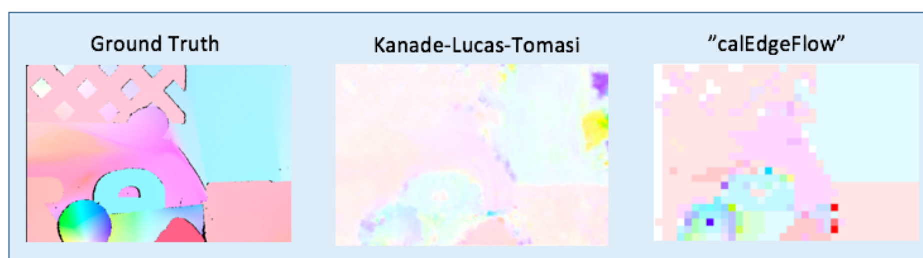
Figur 4.18 viser bildene som ble brukt under evalueringen. Bildene inneholder ulike typer objekter og bevegelser. I bildet **RubberWhale** ser man ulike objekter som forflytter seg i ulike retninger med relativt små bevegelser. Bildet **Dimetrodon** viser et dyreobjekt hvor bakgrunnen inneholder en del refleksjoner. Bildet inneholder små bevegelser som er rettet mot venstre. Det siste bildet **Hydrangea** viser detaljer av en blomst som roterer foran en glatt bakgrunn, som roterer med blomsten og som oppleves å bevege seg mot høyre. Bevegelsene i dette bilde er noe større en i de to andre bildesekvensene.

Figur 4.18: Bildesekvensene: RubberWhale, Hydrangea og Dimetrodon.

Tabell 4.6 inneholder evalueringsresultatene mellom Kanade-Lucas-Tomasi [24] og den implementerte metoden *calEdgeFlow* med hensyn til *endpoint error (EE)*. Bilderesultater er lagt ved i vedlegg C og D. Metoden Kanade-Lucas-Tomasi [24] kom noe bedre ut med tanke på gjennomsnittet i *endpoint error (EE)*. Den implementerte metoden bruker derimot 3,73 ganger mindre tid for å estimere den optiske flyten i bildet. Parametrene for *calEdgeFlow* ble satt hvor blokkstørrelsen var 16 x 16 og søkeparameteren var 10, for Kanade-Lucas-Tomasi [24] var den enkleste form tatt i bruk.

Tabell 4.6: Inneholder evalueringsresultater for *Endpoint error (EE)*.

	Kanade-Lucas-Tomasi [24]		**calEdgeFlow**	
	Endpoint error (EE)	**time**	**Endpoint error (EE)**	**time**
RubberWhale	0.7723	0.3669 s	0.5124	0.0989 s
Hydrangea	0.7710	0.3618 s	0.8723	0.0975 s
Dimetrodon	0.4755	0.3644 s	0.6860	0.0943 s
Gjennomsnitt:	**0.6729**	0.3644 s	0.6902	**0.0969 s**



Figur 4.19: Bildene som brukes for å evaluere nøyaktigheten fra videosekvensen **RubberWhale**.

4.4 Diskusjon

I denne rapporten har det vært et fokus på enkle og effektive algoritmer. Den implementerte metoden er evaluert med tanke på ytelse, beregningseffektivitet og nøyaktighet.

Analyseringen av ytelsen til den implementerte metoden ble utført i to omganger. Først ble metoden implementert, og deretter evaluert for seg selv. Det ble så utført en sammenligning med en kjent optisk flyt-metode med hensyn til nøyaktighet, hvor også ytelsen ble evaluert. MATLAB-funksjonen *Profiler* er et analyseverktøy som ble tatt i bruk, og resultatene ble vist i form av et grafisk brukergrensesnitt *(Graphical User Interface - GUI)*. I denne analysen ble metoden **0,162s** brukt for å estimere den optiske flyten i bildet. Ut ifra analysen fikk man et klart bilde av hvilken del av algoritmen som brukte mest tid. BM-delen av algoritmen brukte hele **63,5%** av metodens totale tid. Det er denne delen som sammenligner blokkene, og det er her flest beregninger utføres. Simuleringsresultatene viser at metoden brukte **0,08s** på **32 356** søk.

Det ble på grunnlag av dette utført en sammenligning av ulike BM-metoder. Ut ifra resultatene kom metodens fordeler og ulemper forholdsvis klart frem. BM-metoden som er implementert i denne rapporten vil ikke kunne forbedres med tanke på antall søk per blokk. Antall søk er avhengig av størrelsen som blir satt for søkeområdet. BM-delen i *calEdgeFlow* blir utført i 1D og ikke i 2D, noe som bidrar til at algoritmen gjør det svært bra med tanke på tidsforløpet, vist ved figur 4.11. Under denne analysen brukte BM-delen i gjennomsnitt 30 søk per blokk. Den kom best ut blant BM-metodene med hensyn til tidsbruk, som innebærer at metoden er beregningseffektiv. Videre kom metoden dårligst ut med hensyn til kvaliteten på estimatene, vist ved figur 4.9. Den implementerte metoden søker kun i horisontal og vertikal retning, noe som byr på utfordringer ved større bevegelser, eller om bevegelsene skulle forekomme diagonalt. Kvaliteten på estimatene påvirkes også av at den implementerte metoden baserer seg på heltallig estimering.

Til slutt ble metoden evaluert med tanke på nøyaktighet. Til dette formålet brukes i denne rapporten metoden *endpoint error (EE)*. Det er et mål på differansen mellom de estimerte forflyttelsesvektorene og vektorene fra ground truth-dataene. En velkjent optisk flyt-metode ble også inkludert for å kunne sammenligne resultatene, nemlig Kanade-Lucas-Tomasi [24]. Den enkleste formen ble brukt hvor kun ett nivå i pyramiden ble aktivert. Artikkelen som den implementerte metoden stammer ifra [11] brukte også en versjon av *Lucas-Kanade* når metoden skulle analyseres, hvor også denne var av den enkleste form. I [11] kom metoden *EdgeFlow* bedre ut sammenlignet med *Lucas-Kanade*. I denne rapporten kom metodene nokså likt ut med tanke på sammenligningen av *endpoint error (EE)*. Den implementerte metoden benytter seg av heltallsestimat, hvor *EdgeFlow* har en tilleggsfunksjon som gjør at metoden baserer seg på

desimal-estimering. Dette gjør at metoden kommer noe bedre ut i [11], sammenlignet med metoden i denne rapporten. I [11] ble andre tenkikker benyttet for å evaluere metoden. Utgangspunktet for evalueringene var også veldig ulikt, hvor [11] brukte 440 bilder, bildestørrelse 128x96, blokkstørrelse 18 og hvor søkeområdet var satt til 10 piksler. For denne rapporten ble det brukt to bilder under evalueringen, hvor blokkstørrelsen var satt til 16 piksler. Størrelsen ble satt på blokkene for ikke å måtte utelukke så mye informasjon fra bildet. Søkeområdet ble satt til det samme, men bildestørrelsene var noe større, 584x388, noe som betyr at bildene var 18 ganger større. Det vises igjen når tidsmålingene skal sammenlignes. I [11] utførte metoden *EdgeFlow* estimatene for den optiske flyten fem ganger raskere enn *Lucas-Kanade*. Rapportens implementerte metode var **3,7** ganger raskere enn Kanade-Lucas-Tomasi [24]. Resultatene viser at denne typen algoritmer bruker mindre ressurser når den optiske flyten skal estimeres. Samtidig er det viktig å ta med at *Lucas-Kanade* estimerer den optiske flyten for hver piksel i bildet (tett), hvor *EdgeFlow* og den implementerte metoden benytter seg av en sparsom metode. I dette ligger det at metodene estimerer en forflytningsvektor for et større område.

Simuleringer er utført på en personlig datamaskin ved hjelp av MATLAB, som er et kraftig verktøy for utvikling og testing. Plattformen blir mye brukt innenfor forskning. MATLAB er et godt utgangspunkt siden plattformen både er oversiktelig og har en mengde innebygde funksjoner som kan tas i bruk. Det vil være nødvendig å videre overføre den implementerte metoden til en annen type plattform som for eksempel C, Java eller python, hvor dette er typiske programmeringsmiljø for mikroprosessorer.

5. Konklusjon

En optisk flyt-metode for sanntidsbehandling har i denne rapporten blitt implementert og evaluert. Droner opererer i et utfordrende miljø hvor teknologien er i stadig utvikling. Resultatene som er presentert i denne rapporten er generert på en PC og ikke på en aktiv drone. Diskusjonen baserer seg derfor på resultater som er gjort «offline». Resultatene viser at den implementerte metoden kommer godt ut når det kommer til beregningseffektivitet og ytelse. Nøyaktigheten påvirkes av søkemetoden og av at metoden benytter seg av heltallig estimering. En videre utvikling vil være å inkludere desimalestimering.

Bibliografi

[1] [Online]. Available: http://vision.middlebury.edu/flow/

[2] Y. Cho. Cipr sequences. [Online]. Available: http://www.cipr.rpi.edu/resource/sequences/index.html

[3] M. Valle. De vil bruke droner til å levere maten hjem til deg. [Online]. Available: https://www.tu.no/artikler/de-vil-bruke-droner-til-a-levere-maten-hjem-til-deg/367186

[4] ——. Den norske dronen skal vaske høyblokker. [Online]. Available: https://www.tu.no/artikler/den-norske-dronen-skal-vaske-hoyblokker/349557

[5] ——. Vil overvåke byggeplassen med droner. [Online]. Available: https://www.tu.no/artikler/vil-overvake-byggeplassen-med-droner/349527

[6] M. Petrou and C. Petrou, *Image Processing: The Fundamentals*, 2nd ed. John Wiley & Sons, Ltd, 5 2010. [Online]. Available: http://amazon.com/o/ASIN/047074586X/

[7] MatWorks. Histogram properties. [Online]. Available: https://se.mathworks.com/help/matlab/ref/histogram-properties.html?searchHighlight=histogram&s_tid=doc_srchtitle

[8] S. Metkar and S. Talbar, *Motion Estimation Techniques for Digital Video Coding*. Springer Publishing Company, Incorporated, 2013.

[9] S. Murthy and B. Sujatha, "An introduction to high efficiency video coding (hevc) standard in comparison to advanced video coding (a vc) standard," 2013.

[10] A. Barjatya. Block matching algorithms for motion estimation. [Online]. Available: https://se.mathworks.com/matlabcentral/fileexchange/8761-block-matching-algorithms-for-motion-estimation

[11] K. McGuire, G. C. H. E. de Croon, C. D. Wagter, B. Remes, K. Tuyls, and H. J. Kappen, "Local histogram matching for efficient optical flow computation applied to velocity estimation on pocket drones," *CoRR*, vol. abs/1603.07644, 2016. [Online]. Available: http://arxiv.org/abs/1603.07644

[12] K. McGuire, G. de Croon, C. D. Wagter, K. Tuyls, and H. J. Kappen, "Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone," *CoRR*, vol. abs/1612.06702, 2016. [Online]. Available: http://arxiv.org/abs/1612.06702

[13] R. J. D. Moore, K. Dantu, G. L. Barrows, and R. Nagpal, "Autonomous mav guidance with a lightweight omnidirectional vision sensor," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3856–3861.

[14] D.-J. Lee, R. W. Beard, P. C. Merrell, and P. Zhan, "See and avoidance behaviors for autonomous navigation," in *Mobile Robots*, 2002.

[15] Centeye. Introduction. [Online]. Available: http://www.centeye.com

[16] J. Biggs. Tc makers: Centeye creates insect-like flying robots in a dc basement. [Online]. Available: https://techcrunch.com/2012/08/19/tc-makers-centeye-creates-insect-like-flying-robots-in-a-dc-basement/

[17] Centeye. Optical flow. [Online]. Available: http://www.centeye.com/technology/optical-flow/

[18] Y. Altman, *Accelerating MATLAB Performance: 1001 tips to speed up MATLAB programs.* Taylor & Francis, 2014. [Online]. Available: https://books.google.no/books?id=9oqZBQAAQBAJ

[19] MatWorks. Performance and memory. [Online]. Available: https://se.mathworks.com/help/matlab/performance-and-memory.html

[20] ——. Profile to improve preformance. [Online]. Available: https://se.mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html

[21] I. Richardson, *Video Codec Design: Developing Image and Video Compression Systems.* John Wiley & Sons, 2002. [Online]. Available: https://books.google.no/books?id=8jxbbRKVbkIC

[22] E. Cuevas, D. Zaldivar, M. Pérez, H. Sossa, and V. Osuna-Enciso, "Block matching algorithm for motion estimation based on artificial bee colony (ABC)," *CoRR*, vol. abs/1407.0061, 2014. [Online]. Available: http://arxiv.org/abs/1407.0061

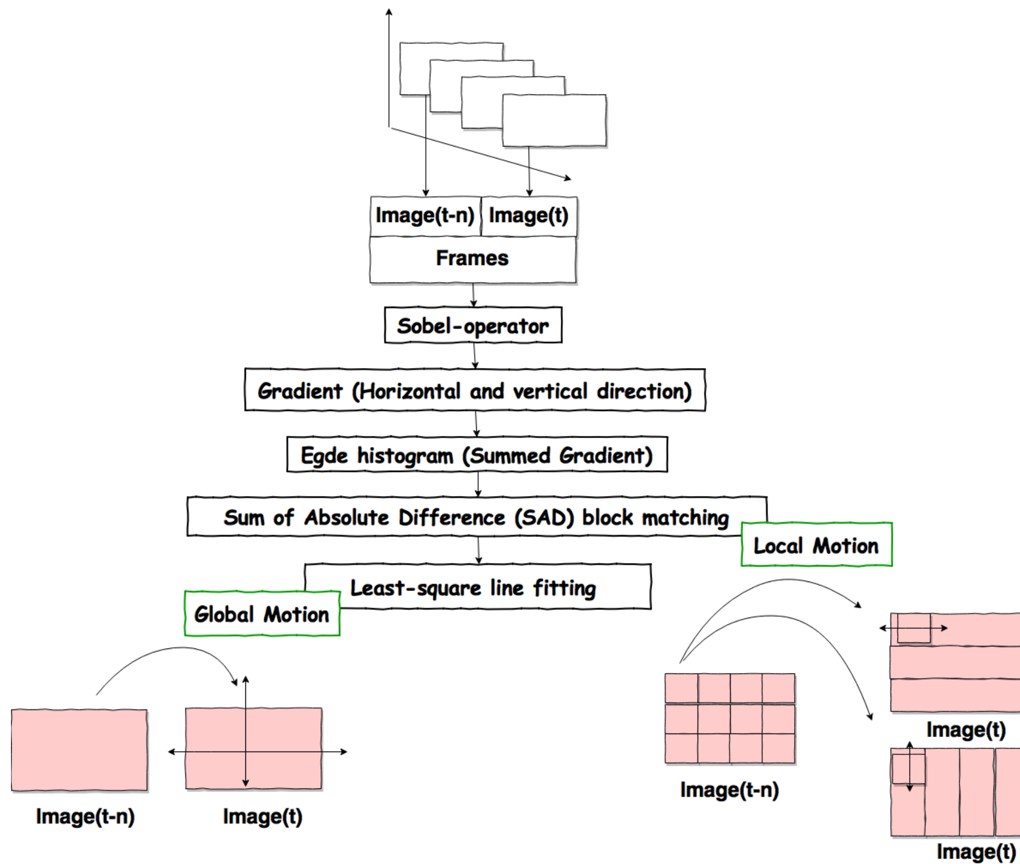
[23] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *Int. J. Comput. Vision*, vol. 92, no. 1, pp. 1–31, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1007/s11263-010-0390-2

[24] S. Smets, T. Goedemé, and M. Verhelst, "Custom processor design for efficient, yet flexible lucas-kanade optical flow," in *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Oct 2016, pp. 138–145.

[25] MatWorks. Psnr. [Online]. Available: https://se.mathworks.com/help/vision/ref/psnr.html#bqh7dwu-1

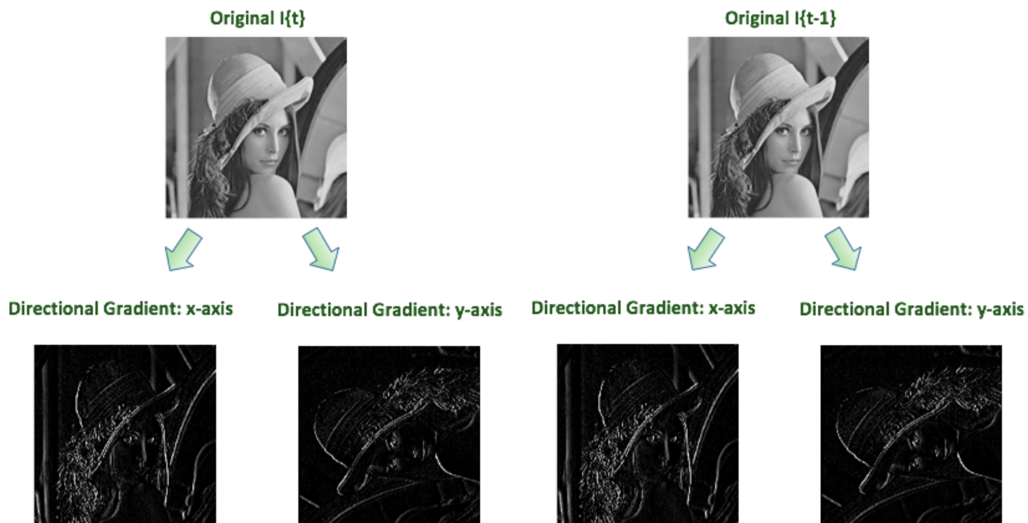
[26] ——. timeit. [Online]. Available: https://se.mathworks.com/help/matlab/ref/timeit.html

A. Skisse av *calEdgeFlow*.

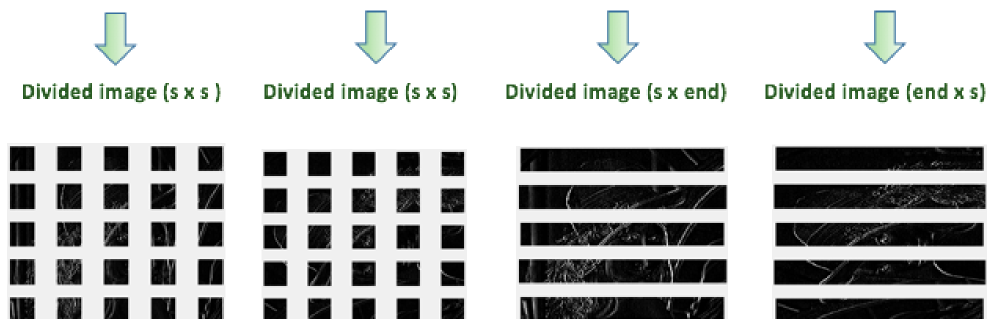


Figur A.1: Skisse av algoritmen calEdgeFlow.

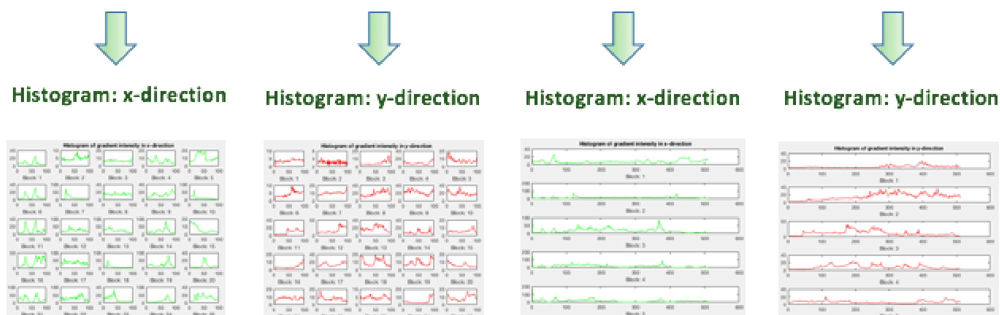
B. EdgeFlow: Steg for steg



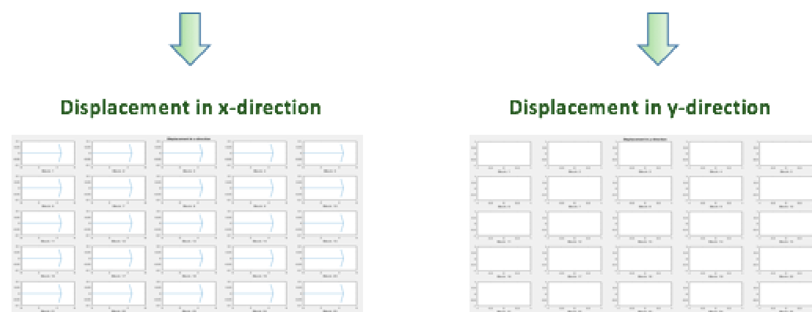
Figur B.1: Resultat av Matlab-filen *calImageGradient.m*.



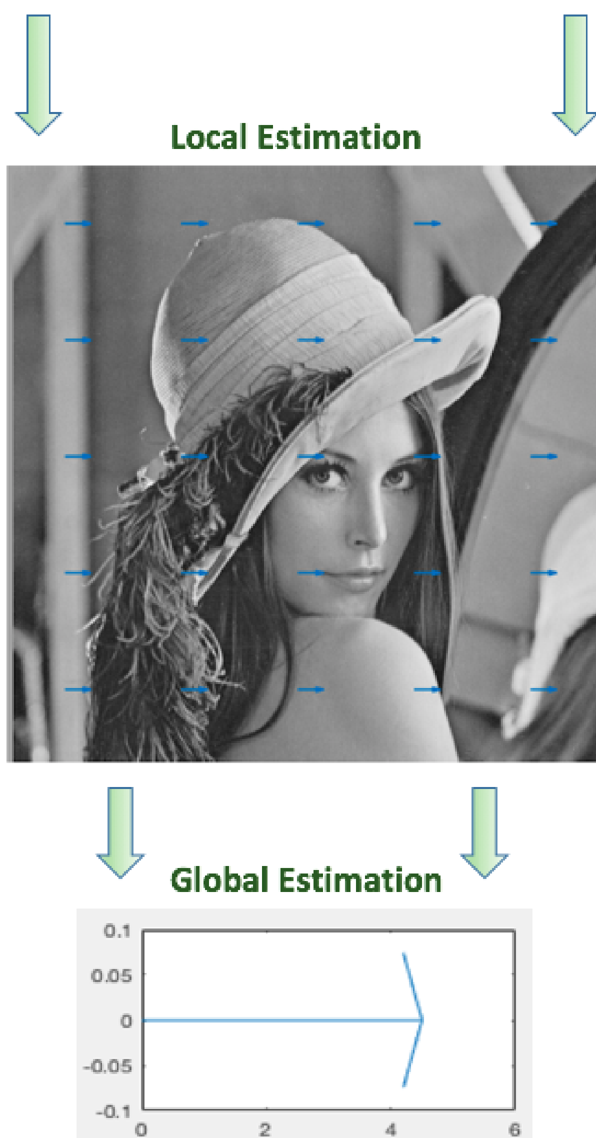
Figur B.2: Resultat av Matlab-filen *preProImagesBM.m*.



Figur B.3: Resultat av Matlab-filen *calHIST.m*.



Figur B.4: Resultat av Matlab-filen *calSADBlockMatching.m*.

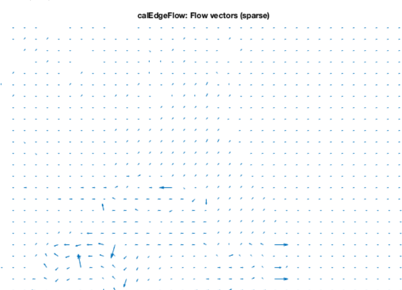


Figur B.5: Resultat av Matlab-filen *calLS.m* og plot av resultatet.

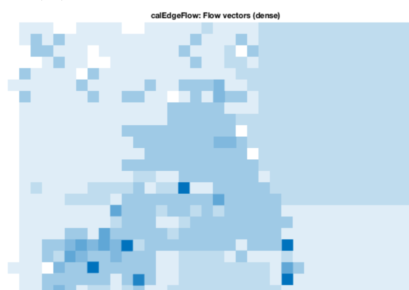
C. Resultat av *calEdgeFlow*.

(a) RubberWhale: Image 10.

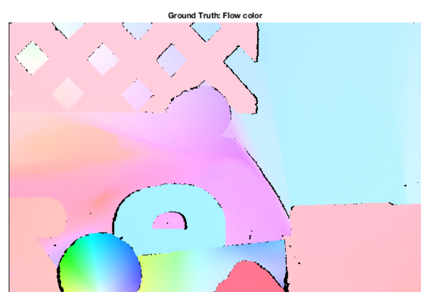
(b) RubberWhale: Image 11.



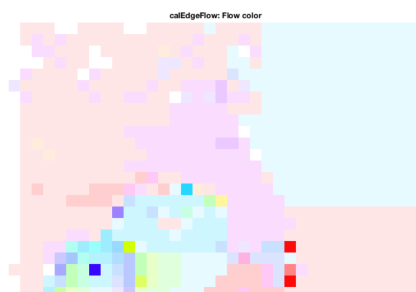
(c) calEdgeFlow: Flow vectors sparse.



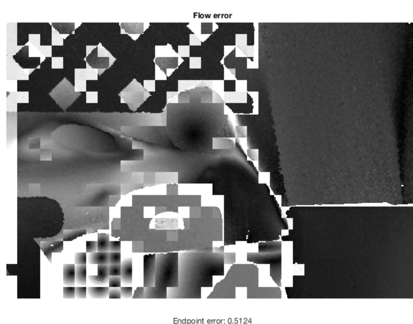
(d) calEdgeFlow: Flow vectors dense.



(e) Ground Truth: Flow color.

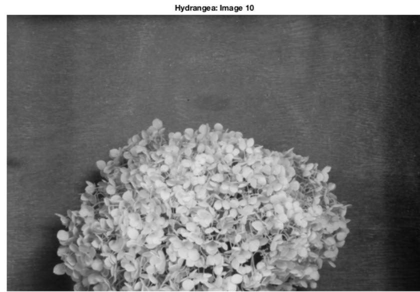


(f) calEdgeFlow: Flow color.

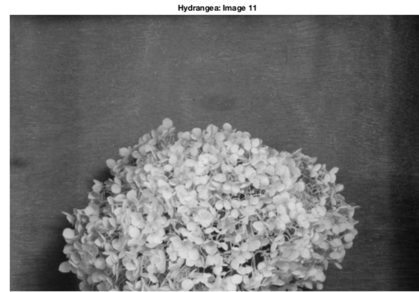


(g) Endpoint error: 0.5124.

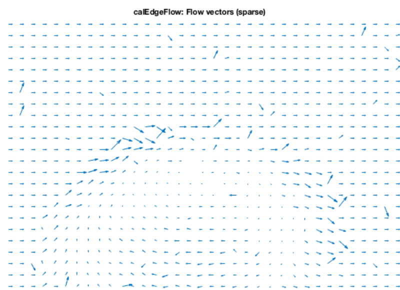
Figur C.1: Resultat på **RubberWhale** ved *calEdgeFlow*.



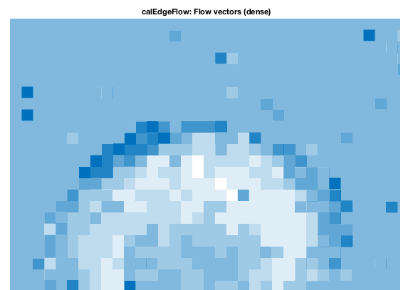
(a) Hydrangea: Image 10.



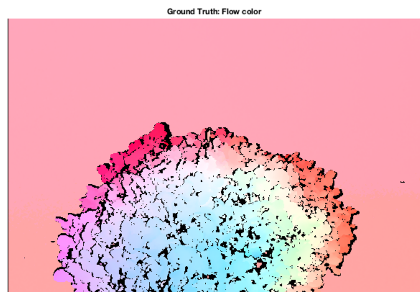
(b) Hydrangea: Image 11.



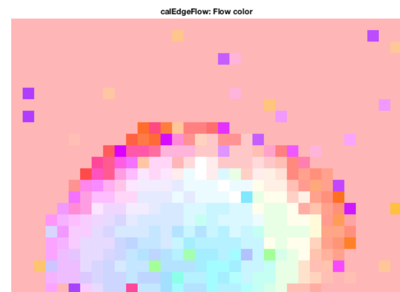
(c) calEdgeFlow: Flow vectors sparse.



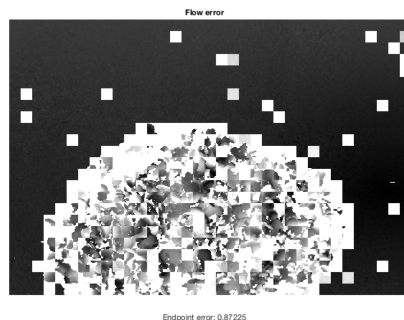
(d) calEdgeFlow: Flow vectors dense.



(e) Ground Truth: Flow color.



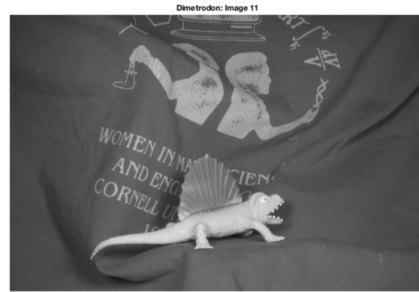
(f) calEdgeFlow: Flow color.



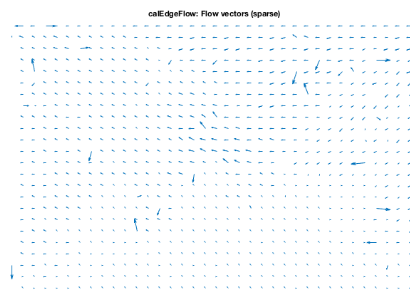
(g) Endpoint error: 0.87225.

Figur C.2: Resultat på **Hydrangea** ved *calEdgeFlow*.

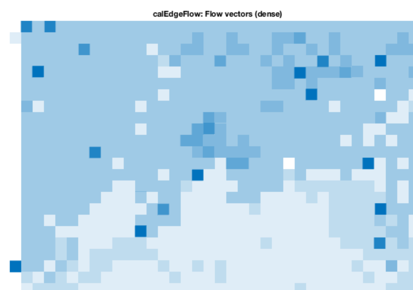
(a) Dimetrodon: Image 10.



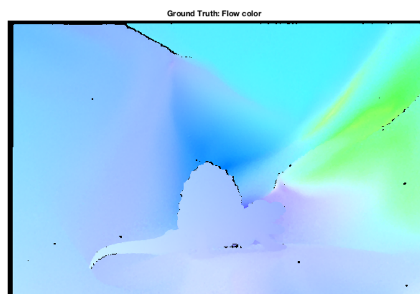
(b) Dimetrodon: Image 11.



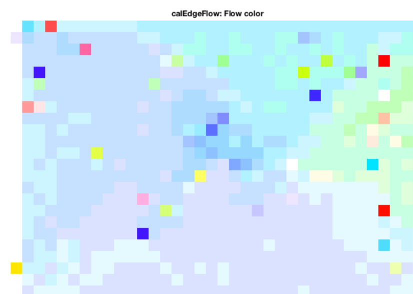
(c) calEdgeFlow: Flow vectors sparse.



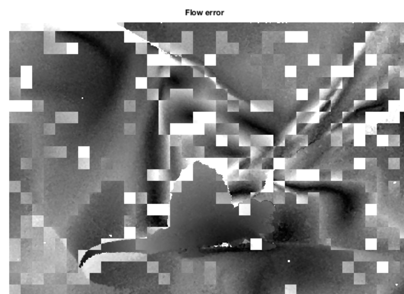
(d) calEdgeFlow: Flow vectors dense.



(e) Ground Truth: Flow color.



(f) calEdgeFlow: Flow color.



(g) Endpoint error: 0.6860.

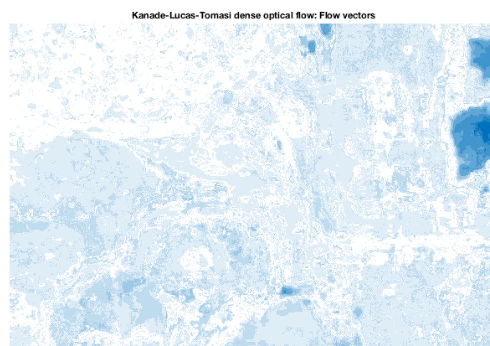
Figur C.3: Resultat på **Dimetrodon** ved *calEdgeFlow*.

D. Resultat av *Kanade-Lucas-Tomasi dense optical flow.*

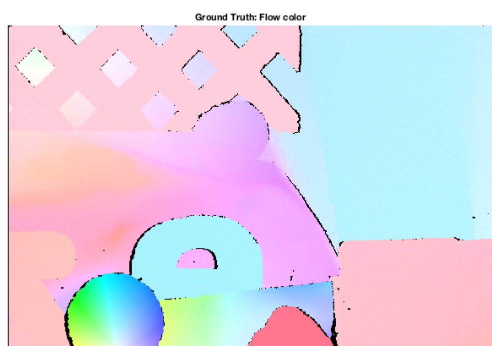
(a) RubberWhale: Image 10.

(b) RubberWhale: Image 11.

(c) Kanade-Lucas-Tomasi dense optical flow: Flow color.



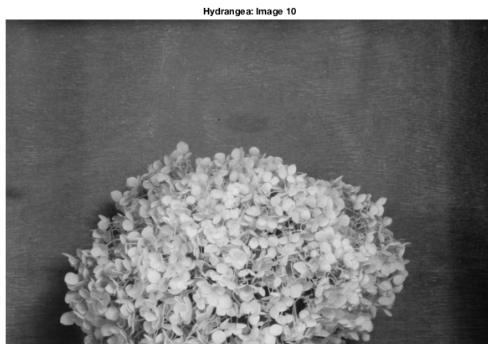
(d) Kanade-Lucas-Tomasi dense optical flow: Flow vectors.



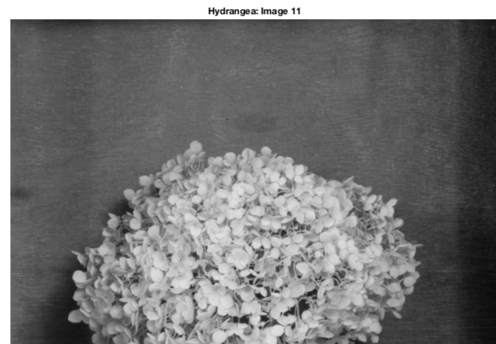
(e) Ground Truth: Flow color.

(f) Endpoint error: 0.77229.

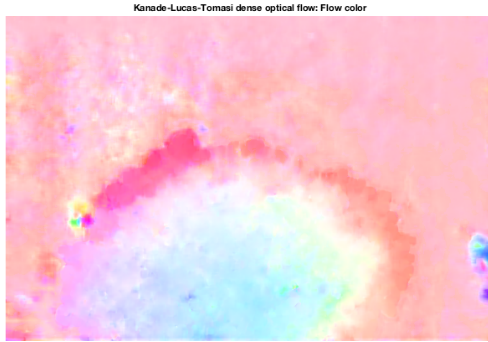
Figur D.1: Resultat på **RubberWhale** ved *Kanade-Lucas-Tomasi dense optical flow.*



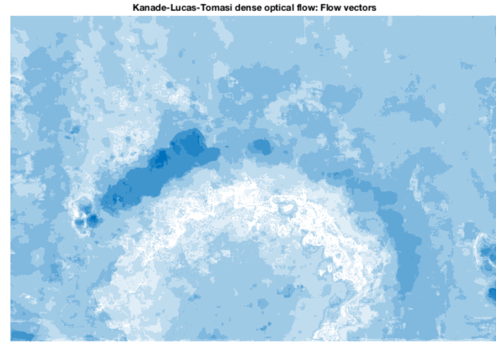
(a) Hydrangea: Image 10.



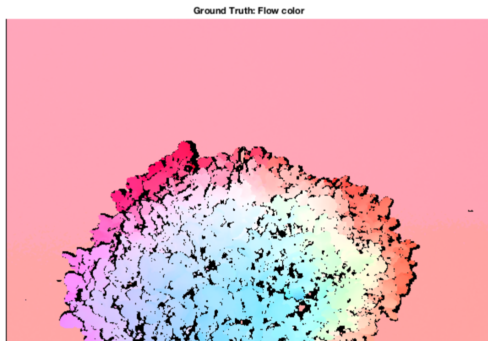
(b) Hydrangea: Image 11.



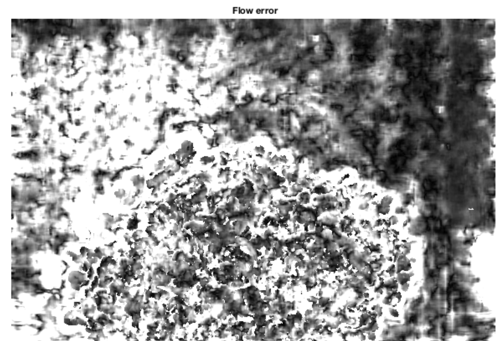
(c) Kanade-Lucas-Tomasi dense optical flow: Flow color.



(d) Kanade-Lucas-Tomasi dense optical flow: Flow vectors.



(e) Ground Truth: Flow color.

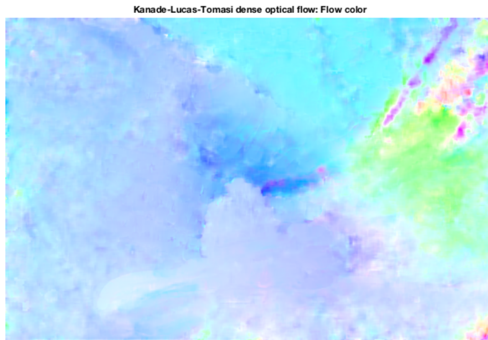


(f) Endpoint error: 0.77103.

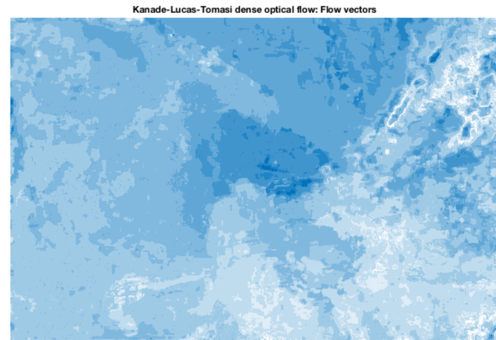
Figur D.2: Resultat på **Hydrangea** ved *Kanade-Lucas-Tomasi dense optical flow.*

(a) Dimetrodon: Image 10.

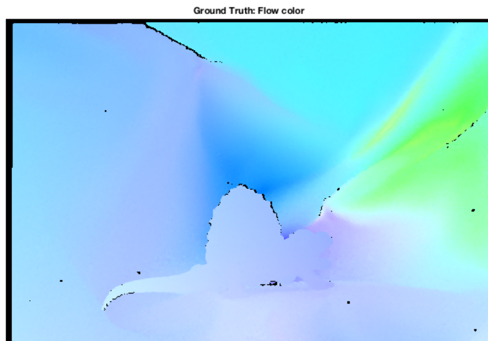
(b) Dimetrodon: Image 11.



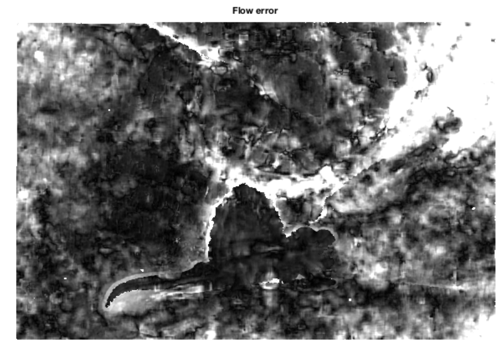
(c) Kanade-Lucas-Tomasi dense optical flow: Flow color.



(d) Kanade-Lucas-Tomasi dense optical flow: Flow vectors.



(e) Ground Truth: Flow color.



(f) Endpoint error: 0.47546.

Figur D.3: Resultat på **Dimetrodon** ved *Kanade-Lucas-Tomasi dense optical flow.*

E. Zip File

- MATLAB-kode
  - BlockMatchingAlgoMPEG
    - Nødvendige filer må lastes ned fra [10]
  - CIPRSequences
    - Nødvendige filer må lastes ned fra [2]
  - Fixed-point Lucas-Kanade optical flow
    - Nødvendige filer må lastes ned fra [24]
  - middleburyBenchmark
    - Inneholder filer fra [1]
  - testFunctions
    - Inneholder filer for testing
  - calEdgeFlow
    - calEdgeFlow.m
    - calHIST.m
    - calImageGradient.m
    - calLS.m
    - calSADBlockMatching.m
    - preDistOF.m
    - preProImagesBM.m
  - calShowResults.m
  - main.m
  - main_performance.m
- Plakat
  - masterPlakat.pdf