HOVEDOPPGAVE

HASTIGHETSMÅLINGER MED DIGITAL VIDEO

av

Ramón Kristian Arellano

Høsten 1999

Institutt for Teknisk kybernetikk
Fakultet for Elektroteknikk og telekommunikasjon
Norges teknisk-naturvitenskapelige universitet

FORORD

Denne rapporten er skrevet som den avsluttende hovedoppgaven i sivilingeniørutdannelsen ved Norges teknisk-naturvitenskapelig universitet i Trondheim, institutt for Teknisk kybernetikk, fakultet for Elektroteknikk og telekommunikasjon. Den er skrevet ved Universitetsstudiene på Kjeller (UniK).

Oppgaven omhandler bruk av digitale videokameraer for hastighetsmåling og identifisering av kjøretøy på vei. Den tar i hovedsak for seg metoder for opptak med kamera montert på bro og i bil. Billedanalyse, matematisk teori, og nøyaktighet blir behandlet. I tillegg er et program implementert og beskrevet for tilfellet med kamera på bro.

Jeg vil gjerne takke min faglige veileder ved UniK, Oddvar Hallingstad, for god hjelp underveis. Det har også vært nyttig å diskutere faglige spørsmål med min medstudent Gudbrand E. Arnesen. Til slutt vil jeg takke alle mine medstudenter på UniK som har skapt et særdeles hyggelig arbeidsmiljø.

Kjeller, 17. desember 1999

Ramón Kristian Arellano

SAMMENDRAG

Oppgaven *Hastighetsmålinger med digital video* beskriver metoder for hastighetsmåling og identifisering av kjøretøy på vei. Målet er å vurdere mulighetene for automatisering av trafikkontroll ved hjelp av digitale videokameraer.

Det er utviklet metoder for å analysere opptak av motkommende kjøretøy. Det er lagt vekt på metoder for opptak der kameraet er montert på bro og i bil. Opptakene omgjøres til enkeltbilder og analyseres deretter av en datamaskin. I denne rapporten utvikles to metoder for beregning av hastighet, den ene bruker kjøretøyenes vertikale posisjon i bildet, og den andre bruker kjøretøyenes dimensjoner. Begge metodene benytter billedanalyse og inverskinematikk for å beregne avstanden til kjøretøyene ved forskjellige tidspunkt. I tillegg utvikles det en metode for å ekstrahere og filtrere nummerskilt, slik at de er leselige for OCR-programmer. Matlab-kode er implementert for analyse av opptak fra bro.

Først gjennomgås all billedanalysen i et eget kapittel, deretter følger et kapittel om den matematiske analysen. Kapitlet tar for seg inverskinematikken i metodene, og kommenterer simuleringer for analyse av nøyaktighet. Fjerde kapittel dokumenterer Matlab-programmet som er implementert. Konklusjon og videre arbeid er behandlet i femte kapittel.

Rapporten konkluderer at metoden som benytter kjøretøyenes vertikale posisjon i bildet på opptak fra bro, gir med dagens digitale kameraer en tilfredstillende nøyaktighet i hastighetsmålingene. Metoden som benytter kjøretøyets dimensjoner, krever høyere oppløsning enn det som er vanlig i dag. I begge tilfellene bør det jobbes videre for å finne metodenes følsomhet for feil i oppsettet. Kjøring av det implementerte programmet på et kort opptak fra bro, viser at billedanalysen, inkludert ekstrahering og lesing av nummerskilt, fungerer i praksis.

INNHOLD

FIGURLISTE

1 INNLEDNING

2.1 BAKGRUNN

Den eksplosive utviklingen i datateknologi har åpnet for nye muligheter på mange fronter. To områder som har utviklet seg dramatisk de siste årene er prosessorkraft i PC'er og teknologien i digitale videokameraer. Fremskrittet på disse områdene gjør det fristende å undersøke muligheter for automatisering av trafikkontroll, og da i første omgang hastighetsmålinger på vei. Ideelt sett kan man se for seg at et kamera monteres i en patruljebil. Bilen kjører en runde mens kameraet filmer kjøretøy i motsatt kjøreretning. Den digitale videoen mates deretter inn i en PC, som analyserer kjøretøyenes hastighet. I tillegg til automatiseringen av hastighetsmåling kan det tenkes at identifiseringen av de kjøretøy som overstiger fartsgrensen også gjøres av en PC. Dette er bare en av flere mulige bruksområder for denne teknologien.

De nye digitale videokameraene lagrer video i DV-format (Digital Video). Dette formatet er fullstendig digitalt og gjør det derfor enkelt å overføre video til harddisk. I dag krever dette bare installering av et ekstra hardware-kort i PC'en som benyttes. Overføringen av data foregår med IEEE-1394, eller Firewire, som har en overføringskapasitet på 3.5 MBps. Hvis et system som analyserer video skulle vært utviket tidligere, måtte de analoge videosignalene først blitt diskretisert. Dette ville ikke bare vært en svært tidkrevende prosess, men kvaliteten på den digitaliserte videoen ville heller ikke være god nok for analyse av enkeltbilder. Med bruk av DV-kameraer kan man i fremtiden også tenke seg at video blir sendt direkte til en datamaskin via mobilnettet eller på et fast nett. Direkte overføring for fastmonterte kameraer innebærer at filmen ikke trenger å hentes fysisk. Dette gjør at bevisforspillelse grunnet hærverk reduseres.

En stor fordel med DV-formatet er at filmen blir lagret som enkeltbilder. Dette i motsetning til andre digitale videoformat som for eksempel MPEG, der kun forandringene fra et bilde til neste blir lagret. Enkeltvis lagring av bilder er meget gunstig for billedanalysen og hastighetsberegningene. Det er mulig å ekstrahere et bilde midt i filmen uten at filmen må avspilles til denne posisjonen. DV-formatet blir mer fullstendig behandlet i Appendiks A – Beskrivelse av DV.

Høy prosessorkraft er også viktig i en slik analyse. Når man skal beregne hastighet bruker man først et program for å analysere filmen bilde for bilde, og trekke ut den informasjonen som er interessant. Til dette benyttes billedbehandlingsprosesser som kantdetektering, objektgjenkjenning og filtrering. Slike relativt vanlige operasjoner krever mye regnekraft, og opptil flere slike operasjoner er nødvendige for å analysere hvert enkelt bilde. En time med video som er lagret med en bilderate på 12,5 bilder per sekund (DV med progressiv scan), tilsvarer 45000 bilder. Derfor er man avhengig av høy prosessorkraft for gjøre analysen praktisk gjennomførbar.

2.2 BRUKSOMRÅDER

Det finnes mange oppsett hvor et digitalt videokamera kan automatisere trafikkontroll som i større eller mindre grad gjøres manuelt i dag. I det følgende blir det beskrevet flere muligheter som innebærer enten hastighetsmåling, identifisering, eller begge deler. Oppsettene deles opp i to hovedområder, kamera i ro og kamera i bevegelse.

2.2.1 Kamera i ro

Et alternativ er montering av kamera på bro for å filme ned på veibanen under. Dette oppsettet er svært gunstig med tanke på inverskinematikken og nøyaktighet. Fordi kameraet står i ro er det flere parametere som kan måles med god nøyaktighet. Blant disse parameterne er kameraets posisjon og orientering. Parameterne trenger heller ikke å måles for hver gang utstyret settes opp. Kameraets posisjon kan merkes av på broen, og oppmålte avstander kan merkes av på veibanen, slik at måling av kameraets orientering kan gjøres ved hjelp av programvare. En annen fordel ved denne metoden er at bakgrunnen i bildene hele tiden er den samme. Kjøretøyet i et bilde kan isoleres fra bakgrunnen ved å fremheve alle områder som er forskjellige fra bilder uten kjøretøy. Denne type isolering gir god nøyaktighet for måling av kjøretøyets posisjon i bildet, som igjen gir økt nøyaktighet for hastigheten. At kameraet står høyt over veibanen, og direkte ovenfor det feltet hvor kjøretøyene blir filmet, forenkler de trigonometriske beregningene. Dette blir gjennomgått i detalj i kapitlet om matematisk analyse.

Et annet alternativ er å sette kameraet i en fastmontert boks ved siden av veibanen. Dette kan effektivisere dagens bruk av fotobokser. Kameraet trenger da ikke å filme kontinuerlig. En sensor lenger fremme i veien kan fortelle kameraet når det skal gjøre opptak, eller elektronikk i boksen kan gi signal når en bil kommer inn i bildet. Direkte sending av bildene via et nettverk gjør at man unngår noe av den ramponeringen av bokser som foregår i dag. Dette oppsettet gir omtrent de samme gunstige forholdene som for kamera på bro.

2.2.2 Kamera i bevegelse

En mulighet er som tidligere nevnt at kameraet monteres i kupeen på en patruljebil. Kameraets orientering og åpningsvinkel (zoom) kan kalibreres mot et fastmontert bilde på en vegg. Kameraet filmer kjøretøy i motsatt kjørebane. Hastigheten beregnes ved å bruke billedanalyse og inverskinematikk på en serie med bilder fra filmen. I et slikt oppsett må man ta hensyn til flere faktorer som kan bidra til feil. Veibanen er ikke nødvendigvis regelmessig, svinger, bakker og dumper vil påvirke beregningene. Også små vibrasjoner kan være en betydelige feilkilde. Dette krever at metoden som brukes for å beregne avstand er robust ovenfor disse faktorene.

En annen mulighet er å montere kamera på taket av en patruljebil. Da unngår man mulige forstyrrelser ved filming gjennom frontrute, og begrensninger på kameraets orientering. Det vil også være mer plass til å montere kameraet på en stabiliserende plattform. Den økte høyden over veibanen muliggjør dessuten en orientering av kameraet som gir større nøyaktighet i målingene, avhengig av hvilken beregningsmetode som blir valgt.

Et spørsmål som blir tatt opp i denne rapporten, er om beregningsmetodene kan gjøres mer robuste dersom man tar i bruk kunnskap om kjøretøyets dimensjoner. Med kjøretøyets dimensjoner menes avstanden mellom kjøretøyets lykter, eller eventuelt bredden på nummerskiltet. Disse dimensjonene kan oppdrives ved oppslag i databaser, etter at kjøretøyets nummerskilt har blitt identifisert.

2 BILLEDANALYSE

All informasjonen man trenger for å beregne et kjøretøys hastighet ligger lagret i enkeltbildene som hentes fra den digitale videoen. Beregningene gjøres ved å først finne avstanden til kjøretøyet ved forskjellig tidspunkt. Dette kan gjøres ved å måle kjøretøyets posisjon i enkeltbildene. Har man kunnskap om kjøretøyets dimensjoner kan man også beregne avstanden ved å måle disse dimensjonene i bildet. Enten man velger å bruke posisjon eller størrelse, trenger man tradisjonelle billedbehandlingsoperasjoner som regionssegmentering, objektgjenkjenning og kantdetektering i analysen.

Dette kapitlet forklarerer først hvordan digitale bilder representeres i en datamaskin. Deretter gjennomgås to metoder for å finne kjøretøyets posisjon bildet, en for video tatt fra bro, og en for video tatt fra bil. Til slutt presenteres en metode for å ekstrahere nummerskilt fra et bilde med kjøretøy.

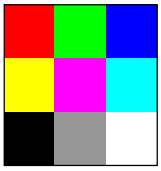
2.1 REPRESENTASJON AV DIGITALE BILDER

Bildebehandlingen som gjennomgås i dette kapitlet tar utgangspunkt i digitale bilder i tre forskjellig format. Det første formatet er for fargebilder av samme type som blir ekstrahert fra AVI-film. Fargebilder representeres med tre matriser. Hver av matrisene representerer en av fargene rød, grønn, blå, og de har alle samme antall linjer og kolonner som bildet. Hvert element i matrisene har en verdi mellom 0 og 1, som gir fargens intensitet for denne pikselen. Fargeblandinger foregår ved å kombinere verdiene for rød, grønn og blå intensitet. Figur 2.1 viser et bilde på 3×3 piksler, med sine tilhørende intensitetsmatriser. Hvit og svart representeres henholdsvis med (0, 0, 0) og (1, 1, 1).

1.0	0.0	0.0
1.0	1.0	0.0
0.0	0.5	1.0

0.0	1.0	0.0
1.0	0.0	1.0
0.0	0.5	1.0

0.0	0.0	1.0
0.0	1.0	1.0
0.0	0.5	1.0



Figur 2.1. Representasjon av fargebilde med rød, grønn og blå intenstitesmatrise.

Selv om man bruker desimalverdier mellom 0 og 1 for å representere intensitet, så betyr ikke det at det finnes et uendelig antall intensitetsnivåer. Antallet nivåer er avhengig av hvor mange bit som brukes ved lagring hvert piksel. Bit per piksel forandrer seg fra med lagringsformat og kapasiteten til skjermkortet. Her vil det bare bli operert med de desimale verdiene. Ved visning av et bilde på skjerm blir verdiene uansett rundet av til nærmeste nivå.

I noen billedanalyseoperasjoner er vi bare interessert i bildets intensitet, og trenger ikke informasjon om fargefordelingen. I slike tilfeller brukes et gråskalaformat som består av én matrise. Matrisen har fortsatt samme antall linjer og kolonner som bildet det representerer, og også her er elementverdiene mellom 0 og 1. For å gå fra et fargebilde til et gråskalabilde trenger man bare å ta gjennomsnittet av rød-, grønn- og blå-verdiene for hvert piksel. Fordi man mister informasjon om fargefordelingen når man tar gjennomsnittet, vil det ikke være mulig å reversere prosessen.

Det siste formatet er for binære bilder. De består i likhet med gråskalabilder av én matrise, men elementverdiene i binære bilder kan bare være 0 eller 1. Fordelen med dette formatet er at det har klare avgrensninger og muliggjør derfor bildebehandlingsfunksjoner som telling av objekter, fylling av hull, erodering og lignende. For å gå fra et bilde i gråskala til et binært bilde, brukes terskling. Det vil si at alle elementverdier i gråskalbildet som er høyere enn den valgte terskelen (settes ofte til 0.5) blir til enere i det binære bildet. Verdier som er lavere enn terskelen blir til nuller.

Figur 2.2 viser samme bilde i de tre forskjellige formatene.

Figur 2.2. Bilde av vei i fargeformat, gråskalaformat og binærformat.

Innføringen av gråskala og binære bilder gjør at mange billedanalyseoperasjoner går betydelig rasker og krever mindre minne. For en mer detaljert behandling av digital bilderepresentasjon refereres det til boken *Digital Image Processing*, av Gonzalez og Woods **[1]**.

2.2 KJØRETØYETS POSISJON I BILDET

Kjennskap til hva man leter etter kan med fordel benyttes når kjøretøyenes posisjon skal analyseres. Det er for eksempel ikke nødvendig å lete etter et kjøretøy ovenfor horisonten eller utenfor veikantene. Man vet dessuten at et kjøretøys lykter vil skille seg ut på bildet ved sin farge og intensitet. Den kompliserende faktoren i billedanalysen er at man ikke på forhånd vet nøyaktig hvordan bildene vil se ut. For eksempel vil veibanens farge og intensitet variere med asfaltens struktur, og om den er tørr eller våt. Lysforholdene, som igjen er bestemt av vær og veiens belysning, er avgjørende for bildets egenskaper, og kjøretøyenes utseende er naturligvis også sterk varierende. For at en og samme algoritme skal fungere under alle forhold, må man bruke fleksible og

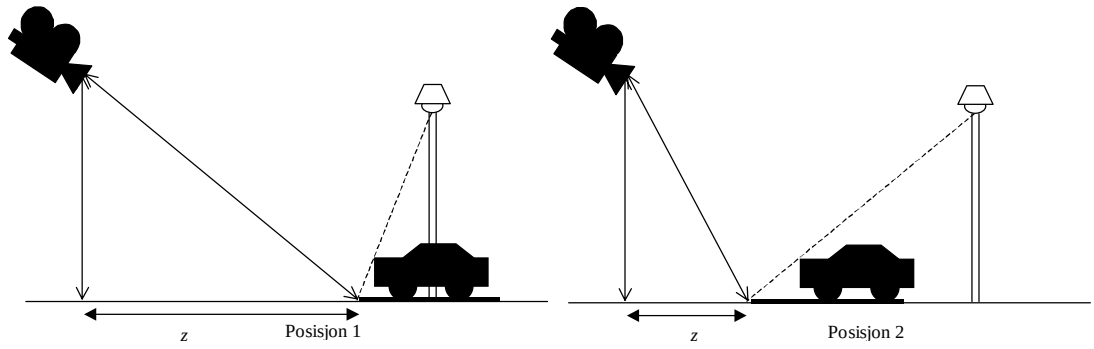
muligens selvkalibrerende metoder. I dette kapitlet blir det lagt vekt på to forskjellige scenarioer med hver sin løsningsmetode. I det ene scenarioet står kameraet i ro på en bro, og i det andre er kameraet montert i en bil.

2.2.1 Kamera på bro

I dette scenarioet filmer kameraet fra en bro og ned på veibanen under. Det gjør det gunstig å beregne avstanden til kjøretøy ut fra posisjon i bildet. Grunnen er at kameraet står relativt høyt over veien, og bildene får litt "fugleperspektiv". Det medfører at en endring i et kjøretøys posisjon langs veien gir en relativt stor endring i kjøretøyets posisjon i bildet. Motsetningen ville være om kameraet lå på nivå med veibanen. Kjøretøyets posisjon langs veibanen ville da ikke gi utslag på posisjonen i bildet. En sterkere matematisk begrunnelse er gitt i den matematiske analysen.

2.2.1.1 Skyggers innvirkning

Med kjøretøyets posisjon i bildet menes den vertikale posisjonen for overgangen mellom veibanen og kjøretøyet. Lysforholdene vil være avgjørende for om det er kjøretøyets forhjul eller kjøretøyets skygge som gir denne overgangen. I sollys er det ikke viktig for beregningene om vi måler avstanden til skyggen i stedet for hjulene. Skyggen vil forholde seg i samme posisjon relativt til kjøretøyet, uavhengig av kjøretøyets bevegelse. Hvis skyggen derimot skyldes lykter langs veien, er situasjonen annerledes. Det kan forekomme at skyggen ligger lenger foran kjøretøyet ved en posisjon i forhold til en annen, slik det er vist i Figur 2.3.



Figur 2.3. Feilaktig avstandsmåling på grunn av endrende skygge.

Under slike forhold vil måling av overgangen mellom veibane og skygge gi feil i hastighetsberegningene. Det er derfor viktig at man bruker metoder som måler overgangen mellom veibane og forhjul.

2.2.1.2 Bakgrunnsbilde

Fordi kameraet står og i ro er bakgrunnen alltid den samme. Det gjør det naturlig å først skille kjøretøy fra bakgrunnen, og deretter finne kjøretøyenes posisjon i bildet. Bilder som bare inneholder bakgrunn og ingen kjøretøyer trenger ikke å være eksakt like. Små vibrasjoner, tilfeldige lysglimt og elektronisk støy kan medføre at bilder i videoen som teoretisk sett burde vært like, ikke er det. Disse effektene reduseres ved å bruke gjennomsnittet av en serie bilder.

For å skjønne hvordan man finner et gjennomsnittsbilde bør man først vite at bilder ekstrahert fra en AVI film har RGB format, og består dermed av tre matriser. Beregningen foregår ved summere matrisenes fargeverdier elementvis. Deretter deles verdiene med antallet bilder i serien. De tre resulterende matrisene representerer gjennomsnittet av bildene.

2.2.1.3 Differansebilde

Neste steg er å sammenligne det gjennomsnittelige bakgrunnsbildet som ble funnet, med andre bilder fra videoen. Kjøretøy vil utmerke seg som områder der piksler er forskjellige fra de respektive pikslene i bakgrunnsbildet. Spørsmålet er hvordan man avgjør om en piksel er forskjellig fra en annen.

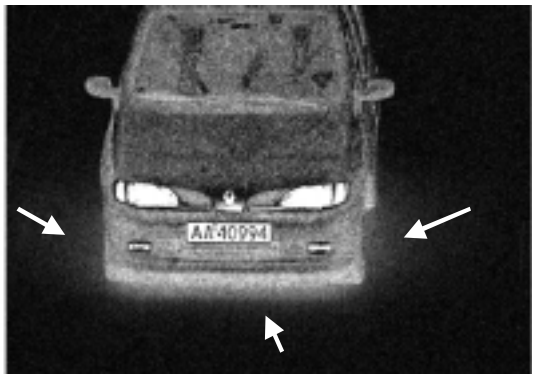
Figur 2.4. Gjennomsnittelig bakgrunnsbilde, og bilde med kjøretøy som skal isoleres.

Ett alternativ er å se på forskjellen i pikslenes intensitet. Det vil si at man beregner gjennomsnittet av pikselens rød-, grønn- og blåverdi, og finner den absolutte differansen mellom disse. Kravet blir da:

$$abs\left(\frac{P(rød)+P(grønn)+P(blå)}{3}-\frac{P_b(rød)+P_b(grønn)+P_b(blå)}{3}\right)>Terskelverdi \qquad \textbf{(2.1)}$$

Her er P en piksel i bildet som blir testet, og P_b den respektive pikselen i bakgrunnsbildet.

Problemet med denne metoden er at nettopp skygger kommer veldig godt frem som områder med stor forskjell i intensitet. Dette fremkommer også på Figur 2.5, som viser et bilde med absolutte intensitetsdifferanser for bildene på Figur 2.4. De lyse områdene foran og på siden av bilen skyldes skygge.



Figur 2.5. Intensitetsdifferanser. Pilene viser hvor skygge gir utslag.

En måte å unngå problemet med skygger på, er ved å analysere forskjeller i pikslenes farge istedenfor intensitet. Da er det forholdene rød/grønn, rød/blå og grønn/blå som benyttes. Man finner differansen i disse forholdene for piksler med samme posisjoni bildene. Summen av absoluttverdien for differansene representer størrelsen på den fargemessige forskjellen. Det matematiske kravet blir:

$$\sqrt{\left(\frac{P(\text{rød})}{P(\text{grønn})}-\frac{P_b(\text{rød})}{P_b(\text{grønn})}\right)^2+\left(\frac{P(\text{rød})}{P(\text{blå})}-\frac{P_b(\text{rød})}{P_b(\text{blå})}\right)^2+\left(\frac{P(\text{grønn})}{P(\text{blå})}-\frac{P_b(\text{grønn})}{P_b(\text{blå})}\right)^2} > Terskelverdi \quad \textbf{(2.2)}$$

Forbedringene ved bruk av denne metoden kan sees i Figur 2.6. De lyse områdene foran og ved siden av bilen har nå blitt mindre eller forsvunnet helt.

Figur 2.6. Fargedifferanser.

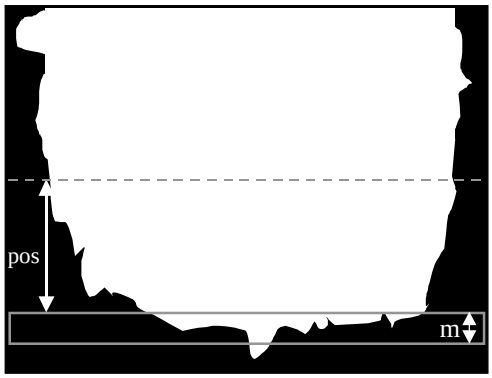
Men sammen med skyggene forsvinner også de deler av bilen som har samme fargesammensetning som den grå asfalten. Både den svarte støtfangeren og de hvite lyktene kommer dårlig frem. Dette kan i noen situasjoner være uønsket. Det er dermed naturlig at en kombinasjon av de øvrige metodene vil gi det beste resultatet. Hvis differansebildene omgjøres til binære bilder, og terskelverdiene settes høyt slik at det blir lite støy rundt kjøretøyet, kan det benyttes en *eller*-funksjon for å kombinere dem. Videre behandling av det resulterende differansebildet kan være:

- Fjerne støy rundt kjøretøyet ved hjelp av erosjon etterfulgt av dilatasjon.
- Fylle hull, det vil si områder med svarte piksler som er omringet av hvite piksler.
- Bevare av ett eller to av de største objektene i bildet og fjerne alle andre.

En fullstendig demonstrasjon av slik behandling er gitt i kapitlet Implementering.

2.2.1.4 Posisjon

Selve beregningen av posisjonen er svært enkel når man har isolert kjøretøyet. I et binært bilde hvor et kjøretøy har blitt isolert, vil alle piksler som utgjør bakgrunn ha verdien 0, og alle piksler som utgjør kjøretøy vil ha verdien 1. Posisjonsberegningen kan gjøres ved å starte nederst i bildet og summere antall hvite piksler i de *m* nederste linjene av bildet. Hvis summen overstiger en bestemt terske *T,* registreres den vertikale posisjonen. Hvis ikke så prøver man igjen en linje lenger opp. Begrunnelsen for å summere over *m* rader og ikke bare én, er å unngå at en liten ”tarm” i kjøretøyobjektet skal gi en feilaktig måling. Figur 2.7 viser at posisjonen måles til midten av bildet. Dette begrunnes i den matematiske analysen.



Figur 2.7. Måling av posisjon i differansebildet.

2.2.2 Kamera i bil

Når kameraet er montert i eller på taket av en bil, oppstår en annerledes problemstilling. Kameraet står ikke lenger høyt over veien. Nøyaktigheten ved å bruke kjøretøyets vertikale posisjon i bildet blir redusert. Et mulig alternativ er å måle avstanden mellom lyktene i bildet. Denne avstanden er ikke avhengig av kameraets høyde over bakken, og den forteller også noe om avstanden til kjøretøyet. Kapitlet Matematisk Analyse går nærmere inn på dette.

Det blir også vanskeligere å isolere kjøretøyet i slike tilfeller fordi bakgrunnen endrer seg kontinuerlig. Det gjør det umulig å bruke samme algoritme som tidligere. Det som befinner seg rundt kjøretøyet i bildet kan dessuten være alt fra asfalt og andre biler, til himmel og skog. Målet blir derfor å finne overgangen mellom kjøretøyets forhjul og asfalten, direkte.

2.2.2.1 Identifisering av veibane

For å finne kjøretøyets posisjon er det nødvendig finne ut hvilke deler av bildet som utgjøres av veibanen. I motsetning til tidligere har man ikke en fast bakgrunn. Det er derfor ikke mulig å sammenligne et bilde med kjøretøy mot et gjennomsnittelig bakgrunnsbilde. Man må istedenfor sammenligne med en gjennomsnittelig veibane-piksel. Denne pikselen finnes ved å velge et område hvor man ut fra kameraets orientering vet at det bare er veibane. Deretter beregnes den gjennomsnittelige verdien for piksler i dette området. I Figur 2.8 er et slikt område avmerket med en rød ramme i nedre høyre hjørne.

Figur 2.8. Bilde tatt fra bil. Det merkede området brukes for å finne en gjennomsnittelig verdi for veibanepiksler.

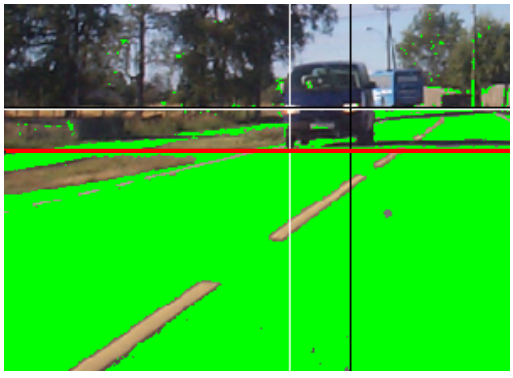
2.2.2.2 Lyktenes posisjon i bildet

Enten man benytter kjøretøyets vertikale posisjon eller avstanden mellom lyktene for å beregne avstand, er det gunstig å først finne lyktenes posisjon i bildet. Den første lyktens posisjon beregnes ved å søke etter en samling med piksler som har høy gjennomsnittsintensitet. Størrelsen på samlingen som brukes bør være avhengig av hvor i området man leter. Stor samling langt nede på bildet, mindre samling lenger oppe. Dette har direkte sammenheng med at objekter langt nede på bildet er nærmere og dermed større enn objekter lenger opp.

Den andre lykten i paret finnes ved å lete til venstre og høyre for den første lykten. Det bør være et krav at begge lyktene har omtrentlig samme gjennomsnittintensitet. På denne måten unngår man å forveksle en lykt med gjenskinn fra sol og lignende.

2.2.2.3 Posisjon

Når veibanen har blitt identifisert og lyktenes posisjon er funnet, er det en enkel oppgave å finne overgangen mellom kjøretøy og veibane. Først trekkes en linje mellom lyktene. Om denne linjen inneholder noe annet enn veibanepiksler flyttes den en linje ned, og testen gjøres på nytt. Figur 2.9 viser resultatet for analyse av bildet i Figur 2.8. Bildet er analysert med et Matlab-program der metodene beskrevet ovenfor er implementert. Veibanen, lyktposisjonen og overgangen mellom kjøretøy og veibane er merket av.



Figur 2.9. Bilde etter analyse med Matlab-program.

2.3 NØYAKTIGHET

Man kan spørre seg om hvor nøyaktig kjøretøyets eller lyktenes posisjon blir målt. Til tross for avrundingen som skyldes kvantiseringen så er det teoretisk bare én pikselposisjon som er den riktige med hensyn til projeksjonsteorien. Denne presisjonen

er ikke bare avhengig av at man har en god algoritme som er korrekt kalibrert, den er også avhengig av hvordan bildet ble til.

2.3.1 Lukkertid

Bildets skarphet er selvfølgelig avhengig av riktig fokus, men den er også sterkt avhengig av kameraets lukkertid. Dersom man filmer passerende kjøretøy med lav lukkertid vil kjøretøyet rekke å bevege seg mens bildet blir tatt. Resultatet er et kjøretøy som er "dratt" i bildet. Enkel matematikk tilsier at med en lukkertid på 1/50 sekund vil et kjøretøy med en hastighet på 80 km/t bevege seg $s = 80/3.6*(1/50) = 0.444$ meter. Denne bevegelsen vil i de fleste tilfeller være godt synlig, og dermed vil ikke én piksel alene beskrive kjøretøyets posisjon. Dessuten vil lengden som kjøretøyet beveger seg i bildet endre seg med avstanden mellom kamera og kjøretøy. I tillegg til å skape unøyaktighet i posisjon vil uskarpe bilder også gjøre det vanskelig eller umulig å gjenkjenne kjøretøyenes nummerskilt. Lukkertiden bør ideelt sett være rundt 1/500 sekund. Et kjøretøy med hastighet på 80 km/t vil da bare rekke å bevege seg drøyt 4 cm.

I moderne DV-kameraer som Sony TRV-900 kan lukkertiden justeres. Likevel kan man ikke uten konsekvenser bruke høy lukkertid. Høy lukkertid krever god belysning, ellers blir bildene mørke. Et mørkt bilde vil på samme måte som et uskarpt bilde, gjøre det vanskelig å finne den riktige overgangen mellom bil og veibane. Trolig vil framtidens digitale kameraer ha lysfølsomme chiper som tillater høy lukkertid til tross for dårlig belysning, men med dagens utstyr er sollys et krav for å kunne bruke lukkertid over 1/500 sekund.

2.3.2 Komprimering

I DV-format er enkeltbildene komprimert for å bruke mindre plass. Metoden for komprimering består av å først beregne bildenes frekvenskomponenter ved å bruke en diskret kosinustransform (DCT). Deretter fjernes de høyeste frekvenskomponentene i bildet. Denne typen komprimering benytter seg av at høyfrekvente detaljer ikke er godt synelige for det menneskelige øyet. Dermed kan noe av den høyfrekvente informasjonen forkastes uten at det har store konsekvenser for bildenes kvaliteten.

Om denne komprimeringen påvirker posisjonsmålingene i betydelig grad, er uvisst. Komprimeringen har en rate på 5:1, men sammenlignet med raten som er vanlig for JPEG-bilder, 1:10, så er ikke dette mye. I videre arbeid bør dette undersøkes.

2.3.3 Progressiv contra interlaced scan

På noen digitale kameraer kan man velge mellom to forskjellige modus for opptak, progressiv og interlaced scan. Modusen bestemmer i hvilken rekkefølge linjene i den

lysfølsomme chipen skal leses. I interlaced modus veksler man mellom å lese oddetallslinjene og partallslinjene mellom hvert bilde. Et nytt bilde leses inn hvert 1/25 sekund, men bare halvparten av linjene forandrer seg mellom hvert bilde. Interlaced scan er skapt for å simulere høyere bilderate for TV, slik at bevegelse skal se mer naturlig ut. Progressiv scan benytter en annen metode. Her leses alle linjene i kontinuerlig rekkefølge, men fordi datastrømmen må være like stor som for interlaced scan, skapes det to og to like bilder.

Dermed har hver av modusene sine fordeler og ulemper. Interlaced scan har en bilderate på 24 bilder i sekundet, men en virkelig vertikal oppløsning på 240 linjer. Progressiv scan har en reell bilderate på 12.5 bilder per sekund, men en vertikal oppløsning på 480 linjer.

Hvilken modus som bør velges for hastighetsmålinger avhenger av metoden som benyttes for å beregne avstand. For en metode som ikke er følsom for bildets vertikale oppløsning er valget enkelt, interlaced scan vil gi flere bilder å beregne hastigheten med, og dermed større presisjon. Bruker man derimot en metode som avhenger av den vertikale oppløsningen, slik som er beskrevet for kamera på bro, er det mulig det lønner seg å finne kjøretøyets vertikale posisjon i bildet med større presisjon. Hvilken modus som gir størst nøyaktighet for hastighetsberegningen avhenger da av oppsettet, og må vurderes i hvert enkelt tilfelle.

2.4 NUMMERSKILTETS POSISJON I BILDET

For å automatisere hastighetsovervåkingen er man avhengig av at datamaskinen kan lese kjøretøyets skilter fra et bilde. Det finnes allerede mange programmer som tar seg av tekstgjenkjenning, såkalte OCR-programmer. Felles for disse er at de krever å få binære bilder hvor bare teksten vises. Tar man utgangspunkt i et bilde med kjøretøy på en veibane, må man først finne en metode for å ekstrahere den delen som viser nummerskiltet. Bildet må i tillegg filtreres for å at det skal være i binært format. Denne filtreringen bør være selvkalibrerende, slik at man er sikker på at bare bakgrunnen blir hvit, og bare teksten blir svart. Nummerskiltbildene kan deretter lagres på disk og behandles av et OCR-program.

2.4.1 Metode

Når man har beregnet et kjøretøys hastighet til å være høyere enn den lovlige fartsgrensen må et bilde av kjøretøyet sendes til en programmodul som ekstraherer skiltene. Man kan tenke seg mange forskjellige metoder for å ekstrahere skiltene fra resten av bildet. Et eksempel er å først finne frem til kjøretøyets lykter. Disse skiller seg godt ut i bildet med sin sterke intensitet og er derfor lette å finne. Deretter finnes skiltets posisjon ved å søke etter et område som inneholder hovedsaklig svarte og hvite

piksler. En slik metode har svakheter som at skiltet må være plassert midt mellom lyktene og at kjøretøyets lykter må være på.

En bedre metode baserer seg på at man har kjennskap til skiltets dimensjoner. En har også kjennskap til kjøretøyets avstand fra kamera, og dens relative orientering i forhold til kamera. Med disse målene er det relativt enkelt regne seg fram til hvilken høyde og bredde skiltet vil ha i bildet, målt i piksler. Det neste steget blir dermed å lete etter et objekt i bildet som har disse dimensjonene.

For å gjøre denne letingen enklest mulig bør man først bruke kantdeteksjon på det aktuelle bildet. Nummerskiltet vil som oftest skille seg ut fargemessig fra bakgrunnen, men selv et hvitt skilt på hvit bil skiller seg ut med sterkere intensitet. Dette skjer fordi skiltets overflate er refleksbehandlet. Etter kantdeteksjon står man derfor igjen med et binært bilde der skiltets omriss er markert med enere og bakgrunnen består av nuller.

Algoritmer for kantdeteksjon baserer seg på analyse av bildets gradient. Et bilde representeres av en matrise der hvert element har en intensitetsverdi *f(x,y)*. I bildet er *x* linjeposisjonen og y kolonneposisjonen. Bildets gradient definert som:

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \qquad \textbf{(2.3)}$$

Ved kantdetektering er det vektorens magnitude som er interessant.

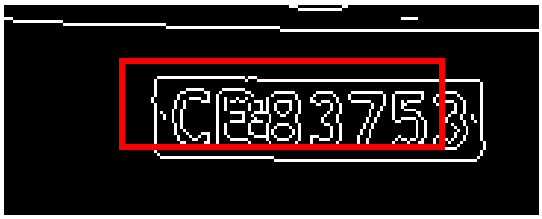
$$\nabla f = mag(\nabla \mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2} \qquad \textbf{(2.4)}$$

Magnituden er lik den maksimale økningsraten per lengdeenhet i retningen av $\nabla \mathbf{f}$, og forteller dermed hvor det skjer store endringer i bildet. En mer detaljert behandling av kantdeteksjon finnes i boken av Gonzalez og Woods **[1]**.

De to utsnittene nedenfor viser resultatene av kantdeteksjon utført i Matlab med funksjonen `edge()`.

Figur 2.10. Eksempler på kantdeteksjon.

Etter kantdeteksjonen gjenstår det å finne det rektanglet i bildet som har dimensjonene til et nummerskilt. Letingen foregår ved at en rektangulær ramme med to pikslers bredde blir flyttet rundt på bildet. Rammens dimensjoner får man fra projeksjonsberegninger. For hver gang rammen flyttes summerer man antall enere, dvs kantpiksler, som befinner seg under rammen (NB! Ikke innenfor, men under). Den posisjon som gir størst sum, og dermed best overensstemmelse mellom ramme og kant, gir skiltets plassering i bildet. For å forbedre algoritmen kan man merke seg at de vertikale kantene er kortere og derfor gir et mindre bidrag til summen. Det er minst like viktig at disse kantene er koordinert med rammen, så for å kompensere kan summen for de horisontale kantene ganges med en faktor større enn 1.



Figur 2.11. Søking etter skilt.

2.4.2 Effektivisering ved bruk av konvolusjon

En algoritme med flere nøstede *for-løkker* som går igjennom alle bildets posisjoner og summerer under rammen, vil være svært tidkrevende. Det samme problemet kan løses mer effektivt ved å bruke diskret konvolusjon i to dimensjoner. Dette gjøres ved å lage en matrise G med nummerskiltets dimensjoner, $c \times d$. Denne matrisen blir kalt en maske. Matrisen skal ha enere langs ytterkanten og nuller i midten. For å vektlegge de vertikale kantene gis disse høyere verdier. Deretter utføres konvolusjon mellom denne matrisen og kantdeteksjonen F av det originale bildet (dimensjoner $a \times b$).

$$C = F(x,y) * G(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(m,n) G(x-m, y-n) \qquad \textbf{(2.5)}$$

Dette gir en matrise C med dimensjoner $M = a+c-1$ og $N = b+d-1$. Posisjonen (x', y') til det største elementet i C er posisjonen for den beste overensstemmelsen mellom masken G og kantdeteksjonen F.

Enkelt eksempel:

Maske-matrise (3×3) $G =$

1	1	1
2	**0**	2
1	1	1

Det markerte elementet i midten er definert til å være origo. Dette er viktig for å finne posisjonen som gir best overensstemmelse senere. Matrisen har toere på de vertikale sidene for å vekte disse sterkere enn de horisontale.

Bildematrise (6×6) $F =$

0	1	0	0	1	0
1	0	0	0	0	0
0	0	1	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	0	0	0	0	1

Matrisen F representerer kantdeteksjonen av bildet. Den har en 3×3 ramme av enere i nedre høyre hjørne, som vi forventer skal gi best match med maske-matrisen. Dimensjonene til konvolusjonsmatrisen blir $M = 3+6-1 = 8$ og $N = 3+8-1 = 8$.

$$C = F(x,y) * G(x,y) = \sum_{m=0}^{8}\sum_{n=0}^{8} F(m,n)G(x-m, y-n) = \quad \textbf{(2.6)}$$

0	1	1	1	1	1	1	0
1	3	1	2	2	0	2	0
2	1	4	3	4	3	2	0
1	1	4	3	6	3	3	0
0	0	4	4	**10**	4	4	0
0	0	3	3	6	4	4	1
0	0	1	2	3	4	1	2
0	0	0	0	0	1	1	1

Den resulterende matrisen C er markert med grått der maskematrisens origo har vært innenfor matrisen F. Elementet med svart ramme har den høyeste verdien i matrisen, og viser dermed hvilken posisjon som ga best match. I dette tilfellet er det når maske-matrisens origo ligger over posisjon (4, 5) i F–matrisen (leses av fra det grå området). Dette stemmer overens med forventningene. En omfattende behandling av konvolusjon for objektgjenkjenning finnes i boken av av Jain, Kasturi og Schunk **[2]**.

2.5 IDENTIFISERING AV NUMMERSKILT

En av de store fordelene med fartsmåling ved bruk av digital video er muligheten for å automatisere identifiseringsprosessen. Med dagens fotobokser må filmen fra et analogt stillbildekamera fremkalles, og en person avlese kjøretøyenes nummerskilt. Med digital video kan en datamaskin ekstrahere nummerskilt fra digitale bilder, og deretter avlese disse automatisk. Det finnes mange avanserte OCR-programmer (Object Character

Recognition) for å lese skrift fra bilde. Felles for disse er kravet om at bildet som skal analyseres må være i binært format, altså bare inneholde hvite og svarte piksler.

Fargeinformasjonen i bildet er ikke relevant for å finne skriften, man kan med fordel omgjøre fargeinformasjon til intensitet. Det vil si at man tar gjennomsnittet av rød-, grønn- og blå-verdien for hver enkelt piksel, og lagrer disse gjennomsnittelige verdiene i en ny matrise av samme størrelse.

Det neste steget er å gå fra intensitet til svart-hvitt, eller binært format. Den enkleste metoden for slik filtrering er basert på at piksler med intensitet høyere enn en bestemt terskelverdi blir enere, og resten blir nuller. Problemet er at en terskel som fungerer godt for ett bildet ikke nødvendigvis fungerer godt for et annet. Den optimale terskelverdien er sterkt avhengig av lysforholdene under opptaket. Figur 2.12 gir et eksempel der det ikke er mulig å finne én terskelverdi som fungerer godt for alle tre bilder. En terskelverdi tilpasset både for det venstre og det midtre bildet, vil gjøre bildet til høyre fullstendig svart. På samme måte vil en terskelverdi tilpasset for bildet til venstre og bildet til høyre gjøre at bildet i midten blir fullstendig hvitt.

Figur 2.12. Skilter med sterkt varierende intensitetsfordeling.

Filtreringen bør derfor være selvkalibrerende. En enkel løsning er å benytte seg av kunnskapen om mengden skrift som finnes i et nummerskilt. Ved å filtrere en håndfull mengde skilter med manuell kalibrering, kan en finne den gjennomsnittelige andelen svart skrift i nummerskilt. Dette forholdet mellom antall svarte piksler og det totale antallet piksler betegnes *a*. Målet er nå å lage en algoritme som alltid gir dette forholdet, uavhengig av hvordan det originale bildet av nummerskiltet ser ut.

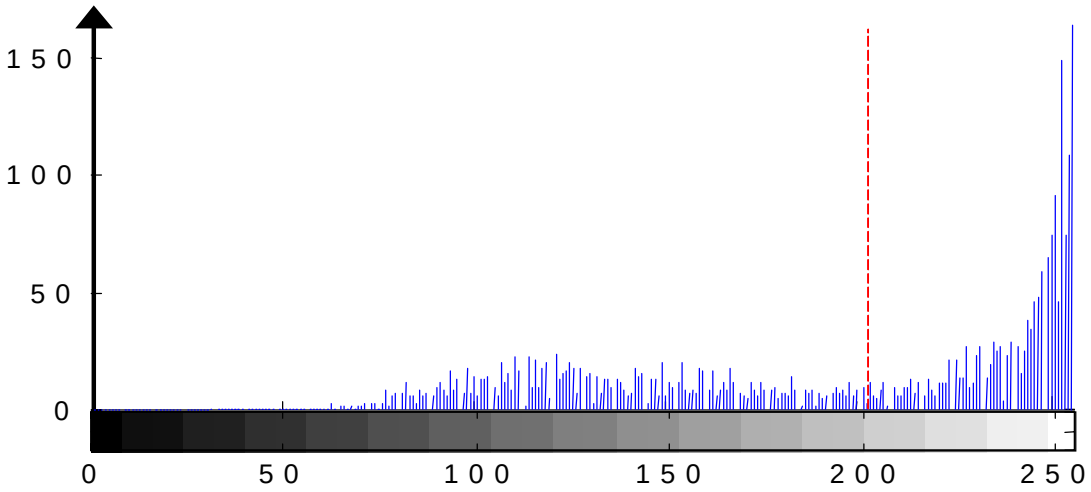
Til dette formålet er det nødvendig å beregne histogram. Et histogram for et gråskalabilde med L intensitetsnivåer er gitt av den diskrete funksjonen $p(r_k) = n_k / n$, hvor r_k er det k'te intensitetsnivået, n_k er antallet piksler med denne intensiteten, n er det totale antallet piksler i bildet, og k = 0,1,2…L-1.

For å finne den ønskede terskelen summerer man opp antallet piksler i de laveste båsene i histogrammet inntil forholdet mellom summen og det totale antallet piksler er lik *a*. Terskelverdien vil da være gitt av antallet båser som må summeres sammen.

$$\frac{\sum_{i=1}^{T} p(r_i)}{n} = a \qquad \textbf{(2.7)}$$

Her er *T* den ønskede terskelverdien.

Figur 2.13 viser et histogram for det første bildet i Figur 2.12, der intensitetene er fordelt på 255 båser. Den stiplede linjen viser terskelverdien for dette bildet.



Figur 2.13. Histogram for første bilde i Figur 2.12.

3 MATEMATISK ANALYSE

Dette kapitlet tar for seg inverskinematikken bak hastighetsanalysen. For å danne grunnlag for analysen, blir teorien bak bildeprojeksjon forklart. Kapitlet tar dernest for seg to metoder for avstandsberegning. Neste delkapittel tar for seg bruken av minste kvadraters metode for å finne hastighet ut fra avstander. Til slutt er det gjort simuleringer med et Matlab-program, for å undersøke metodenes nøyaktighet, og hvilke krav som må stilles til utstyret.

3.1 BILDESKAPNINGSPROSESSEN

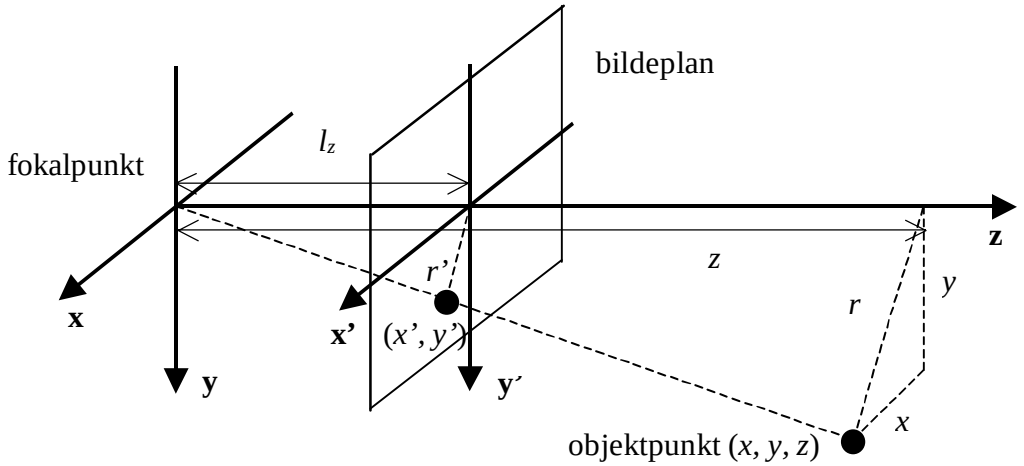
For å beregne en avstand langs en vei ut fra et digitalt bilde av veien, trenger man først å forstå hvordan bildeskapningsprosessen foregår. Enkelt forklart er den en projeksjon av punkter i et 3-dimensjonalt rom til et diskret 2-dimensjonalt plan. Dette underkapitlet tar for seg to viktige aspekter ved bildeskapningsprosessen:

1. Det geometriske aspektet, som avgjør hvor et punkt i et 3-dimensjonal rom blir projisert i bildeplanet.
2. Kvantiseringen av koordinatene i det 2-dimensjonale bildeplanet.

3.1.1 Bildegeometri

Figur 3.1 viser en enkel modell av hvordan et objektpunkt blir projisert i bildeplanet. Det er en slik modell med et 3-dimensjonalt koordinatsystem (**x, y, z**) og origo i fokuspunktet som er brukt igjennom rapporten. Fokalavstanden l_f er definert som avstanden mellom fokalpunktet og bildeplanet, og z er avstanden mellom fokalpunktet og objektpunktet. Y-aksen er orientert med positive verdier nedover for å få bedre overensstemmelse med det 2-dimensjonale koordinatsystemet (**x', y'**) som beskriver bildeplanet. Dette er i samsvar med beskrivelsen av projeksjon i boken av Jain, Kasturi og Schunk **[2]**.

Et punkt i det 3-dimensjonale rom blir "avbildet" dersom en strek som trekkes mellom dette punktet og fokalpunket også går igjennom bildeplanet.



Figur 3.1. Projeksjon av et punkt i 3D rom til et bildeplan.

For å finne koordinatene (*x'*, *y'*) til punktet i bildeplanet bruker man trekantulikheter. Fra Figur 3.1 ser man at følgende forhold gjelder:

$$\frac{f}{z} = \frac{r'}{r} \qquad \text{og} \qquad \frac{x'}{x} = \frac{y'}{y} = \frac{r'}{r} \qquad \textbf{(3.1)}$$

Kombinert og løst for posisjonene i bildeplanet, *x'* og *y'*:

$$x' = \frac{f}{z}x \qquad \text{og} \qquad y' = \frac{f}{z}y \qquad \textbf{(3.2)}$$

Disse enkle formlene for projeksjon gjelder bare når objektpunktet er beskrevet i koordinatsystemet (**x, y, z**) med origo i fokuspunktet. Det er sjeldent tilfellet. For å finne avstand til kjøretøy langs en vei er det gunstig å ha et koordinatsystem hvor z-aksen ligger langs veibanen. Denne z-aksen er ikke sammenfallende med z-aksen i Figur 3.1, som alltid peker langs kameraets sikteretning. Objektpunktet må transformeres fra veibanens koordinatsystem til kameraets koordinatsystem. Det gjøres ved hjelp av en transformasjonsmatrise.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} c_\varphi c_\vartheta c_\psi - s_\varphi s_\psi & -c_\varphi c_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta & d_x \\ s_\varphi c_\vartheta c_\psi + c_\varphi s_\psi & -s_\varphi c_\vartheta s_\psi - c_\varphi c_\psi & s_\varphi s_\vartheta & d_y \\ -s_\vartheta c_\psi & s_\vartheta s_\psi & c_\vartheta & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad \begin{aligned} c_\varphi &= \cos(\varphi) \\ s_\vartheta &= \sin(\vartheta) \end{aligned} \qquad \textbf{(3.3)}$$

Ligning (3.3) beskriver transformasjonen av et punkt fra koordinatsystem (**x, y, z**) til koordinatsystemet (**x', y', z'**). Vinklene φ, θ og ψ beskriver rotasjonen fra det ene systemet til det andre, mens d_x, d_y og d_z beskriver translasjonen. Transformasjonen består altså av en translasjon og en rotasjon. Translasjonen beskriver forflyttingen av

origo mellom koordinatsystemene, og rotasjonen beskriver rotasjonen rundt en eller flere av koordinatsystemets akser. En fullstendig beskrivelse av transformasjoner finnes i kompendiet *Matematisk modellering av dynamiske system*, av Hallingstad **[3],** og i *Modeling and control of robot manipulators*, av Sciavicco og Siciliano **[4]**.

3.1.2 Kvantisering

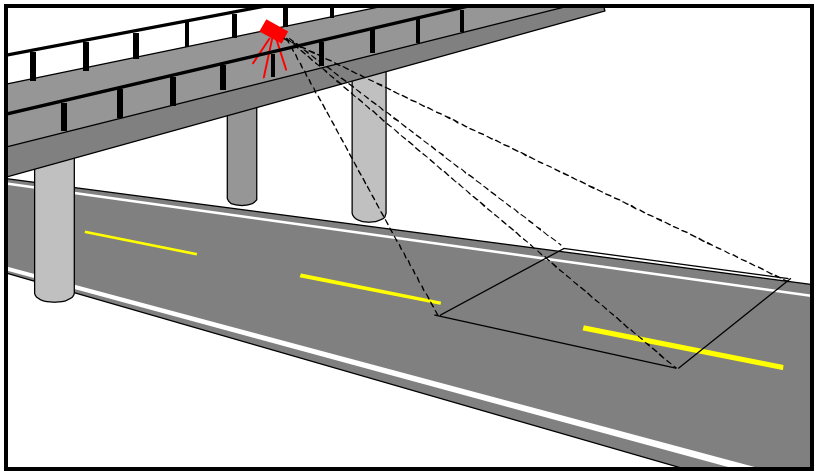
I et digitalt videokameraet blir lys projisert via linsen til en lysfølsom chip. Chipen er en matrise av ørsmå lysfølsomme transistorer. Allerede her skjer diskretiseringen av bildet. Tettheten av transistorer i chipen er avgjørende for pikseloppløsningen. Når et punkt i rommet blir projisert til bildeplanet forgår en form for avrunding i posisjonen. Reversering av prosessen for å beregne punktets posisjon i rommet, vil ikke gi nøyaktig overensstemmelse med den opprinnelige posisjonen, men økt pikseloppløsning i bildet gir økt nøyaktighet.

En naturlig tanke er kanskje å øke oppløsningen ved å sette inn ekstra rader og kolonner i bildet som allerede er tatt, og deretter bruke interpolasjon for å gi disse pikslene verdier. En slik økning av oppløsningen øker ikke mengden informasjon i bildet, og er derfor ubrukelig. Informasjonen i bildet er avhenging av samplingen som gjøres av den lysfølsomme chipen.

3.2 AVSTANDSBEREGNING

3.2.1 Kamera på bro

Å sette kameraet på en bro gir betydelige fordeler i avstandberegningen. Høyden over veibanen gjør det mulig å orientere kameraet slik kjøretøyene kjører inn i bildet ved øvre kant av skjermbildet, og ut igjen ved nedre kant. Dette betyr ta man får en direkte sammenheng mellom et kjøretøys vertikale posisjon i skjermbildet og avstanden mellom kjøretøyet og kameraet. Om man kjenner kameraets orientering og høyde over veibanen, kan avstanden til kjøretøyet beregnes ved relativt enkle trigonometriske operasjoner.



Figur 3.2. Oppsett av kamera på bro.

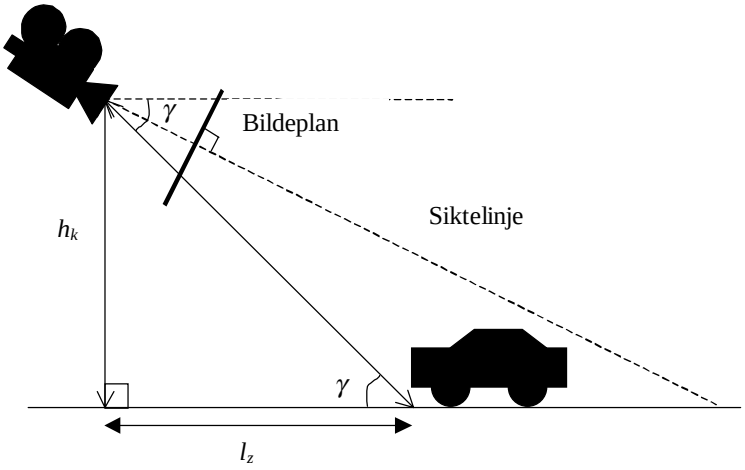
3.2.1.1 Kameraets orientering

Ideelt sett kan man se for seg at kameraet sikter vinkelrett ned på veibanen. Posisjonen i skjermbildet vil da være proporsjonal med posisjonen langs veibanen. For målingenes nøyaktighet er det ønskelig at en liten endring i kjøretøyets posisjon langs veibanen skal utgjøre en stor endring i posisjonen på skjermbildet. Dette gjelder alltid. Dersom en liten avstand i rommet blir projisert til en stor avstand i bildeplanet, vil avrundingen til pikselkoordinater bety lite ved reversering av prosessen. Dessverre er det mange andre praktiske faktorer som spiller inn på hvilken orientering og åpningsvinkel kameraet kan ha:

- Kameraet må orienteres såpass horisontalt at nummerskilt kan leses
- Bare en liten del av veibanen kan være i bildet for at skilt skal bli ”store” nok
- Kameraet må orienteres slik at kjøretøy langt framme ikke dekker kjøretøy som er lenger bak
- Man bør få flere bilder av hvert kjøretøy som passerer, for å redusere eventuelle feil ved måling av posisjon i bildet
- Føreren bør være synlig for eventuell identifisering

3.2.1.2 Kalibrering

Det antas her at kameraets høyde over veien er kjent. Denne avstanden kan enkelt måles ute i felten. Det antas også at vinkelen mellom kameraets sikteretning og veibanen er kjent. Denne vinkelen kan enten måles fysisk, eller den kan beregnes ut fra kjente punkter i bildet. Metoden forutsetter i tillegg at veien er forholdsvis flat. Dersom kjøretøyet man ønsker å finne avstanden til befinner seg på en forhøyning i veien eller i begynnelsen av en bakke så vil det være en kilde til feil. Fordelen ved denne metoden er at den ikke krever kjennskap til kjøretøyets dimensjoner.



Figur 3.3. Oppsett av kamera på bro, sett fra siden.

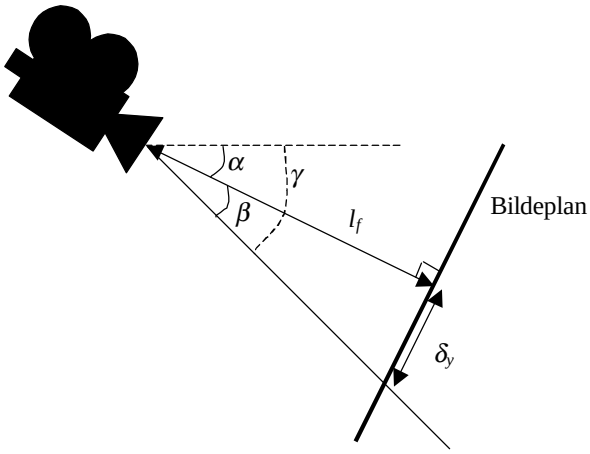
h_k Kameraets høyde over veibanen (måles)
γ Vinkel mellom veibane og en rett linje fra kameralinsen til kjøretøyets front (beregnes)
l_z Avstanden mellom kameraet og kjøretøy (beregnes)

Figur 3.3 viser at høyden h_k, lengden l_z og den rette linjen mellom kameralinsen og bilens front, utgjør en vinkelrett trekant. I denne trekanten mangler bare vinkelen γ for å kunne beregne l_z.

$$\tan\gamma = \frac{h_k}{l_z} \qquad \textbf{(3.4)}$$

$$l_z = \frac{h_k}{\tan\gamma} \qquad \textbf{(3.5)}$$

Figur 3.3 viser også at γ kan finnes igjen som vinkelen mellom trekantens hypotenus og horisontalplanet igjennom kameralinsen.



Figur 3.4. Vinkler mellom kamera, bildeplan og horisontalplan.

α	Vinkel mellom kameraets sikteretning og veibanen (kjent)
β	Vinkel mellom kameraets sikteretning og en rett linje fra kameralinsen til kjøretøyets front projisert i bildeplanet (beregnes)
l_f	Fokalavstanden, dvs. avstanden mellom kameralinsen og bildeplanet (kjent)
δ_y	Avstand fra bildeplanets sentrum til kjøretøyets front i bildeplanet (måles)

Figur 3.4 viser at γ er summen av α *og* β. Vinkelen α er kjent ettersom den er bestemt av kameraets sikteretning og horisontalplanet. Vinkelen β kan beregnes ved hjelp av *fokalavstanden* og lengden δ_y. Lengden δ_y er den som måles i bildet. Den tilsvarer kjøretøyets vertikale posisjon i bildeplanet, målt fra midten av bildet. Hvor nøyaktig δ_y kan måles avhenger av bildets oppløsning og algoritmen som benyttes i billedanalysen.

$$\gamma = \alpha + \beta \qquad \textbf{(3.6)}$$

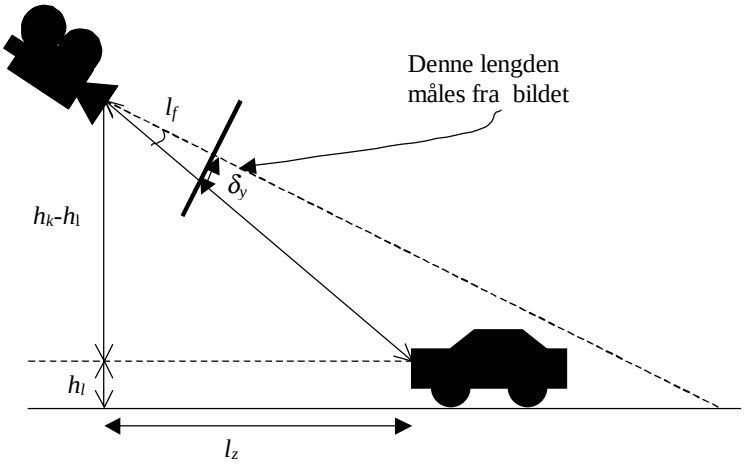
$$\gamma = \alpha + \arctan\left(\frac{\delta_y}{l_f}\right) \qquad \textbf{(3.7)}$$

Den endelige ligningen for beregning av avstand l_z blir dermed ligning (3.5) og (3.7) kombinert:

$$l_z = \frac{h_k}{\tan\left(\alpha + \arctan\left(\frac{\delta_y}{l_f}\right)\right)} \qquad \textbf{(3.8)}$$

3.2.1.3 Mulig forbedring

Dersom vi forutsetter at lyktenes høyde over veien er kjent kan det brukes en forbedret metode. Dette er ikke urimelig å forutsette, da kjennskap til dette målet kan oppnås ved å bruke nummerskiltet for å slå opp kjøretøyet i et register. Ut fra bilens merke og årsmodell finner man lyktenes høyde over veien. Dette vil gi en metode med større nøyaktighet, fordi lyktenes vertikale posisjon i bildet lar seg måle med større presisjon enn overgangen mellom kjøretøyet og veibanen.



Figur 3.5. Oppsett ved måling av lyktenes posisjon i bilet.

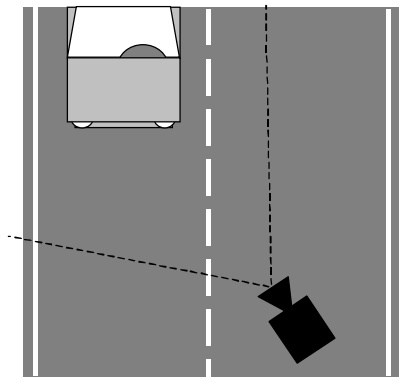
Ligningene vil bli de samme som tidligere bortsett fra at man istedenfor å bruke høyden h_k bruker høyden h_k-h_l, der h_l er lyktenes høyde over veibanen.

3.2.2 Kamera i bil

Metoden ovenfor tar i bruk kjøretøyets vertikale posisjon i avstandsberegningene. Samme prinsipp kan også benyttes når kameraet er montert i en bil, men nøyaktigheten reduseres fordi kameraet ikke befinner seg like høyt over bakken. I dette delkapitlet skal det fokuseres på et alternativ som ikke er avhengig av kameraets høyde. Istedenfor den vertikale posisjonen i bildet benyttes avstanden mellom kjøretøyets lykter. Dette krever at den virkelig avstanden mellom kjøretøyets lykter er kjent. Dette målet kan finnes på samme måte som lyktenes høyde over bakken, slik det er forklart tidligere. Dessuten kan bredden på nummerskiltet, som mer sannsynlig er kjent, også benyttes til dette formål. Grunnen til at avstanden mellom lyktene er å foretrekke, er fordi den vil utgjøre flere piksler bildet, og dermed gi større nøyaktighet i beregningene.

Figur 3.6 viser at kameraet bør være vinklet mot den motsatte kjørebanen. Dette kompliserer avstandsberegningene noe, men til gjengjeld kan kameraets åpningsvinkel

gjøres mindre. Det innebærer at kjøretøyene vil utgjøre en større del av bildet, som igjen betyr høyere nøyaktighet.

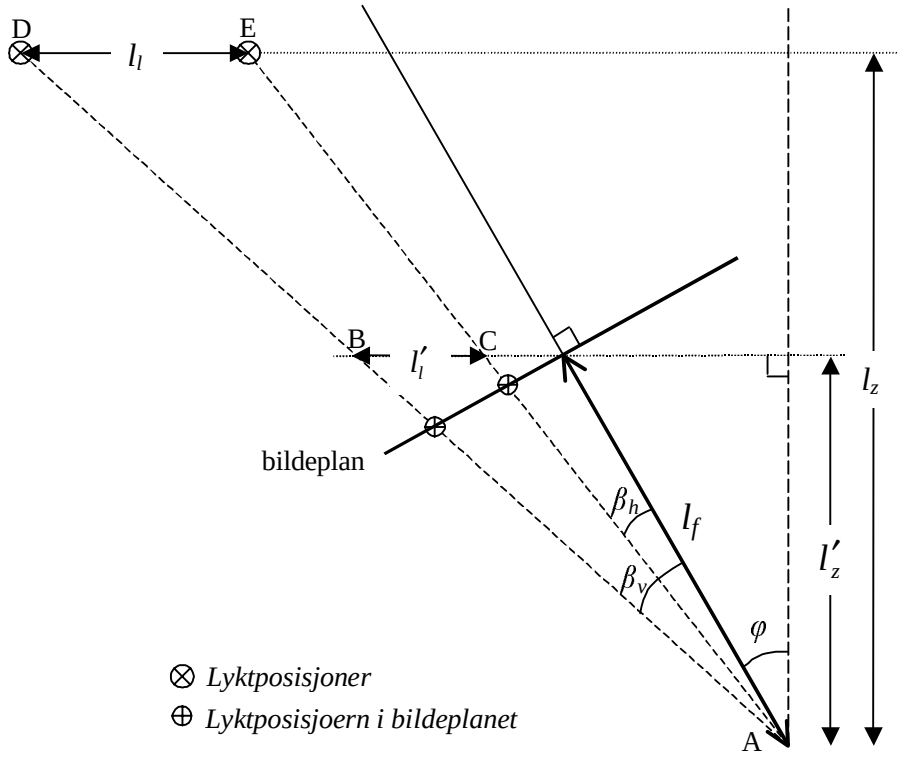


Figur 3.6. Enkel illustrasjon av oppsettet.

Å ha kameraet montert i en bil innebærer at egen hastighet må trekkes ifra den hastigheten som beregnes. Det antas her at informasjon om egen hastighet blir matet til det digitale kameraet under opptak. Slik informasjon kan for eksempel legges på lydsporet i DV-videoen.

3.2.2.1 Kalibrering

For å kunne beregne kjøretøyets hastighet må man først beregne avstanden til kjøretøyet ved forskjellige tidspunkter. Det er essensielt at det er avstanden parallelt med veibanen som berenges, og ikke avstanden mellom kjøretøyet og kameraet i luftlinje. Denne lengden er markert som l_z i Figur 3.7. Endringen i denne avstanden over tid vil gi kjøretøyets hastighet langs veien.

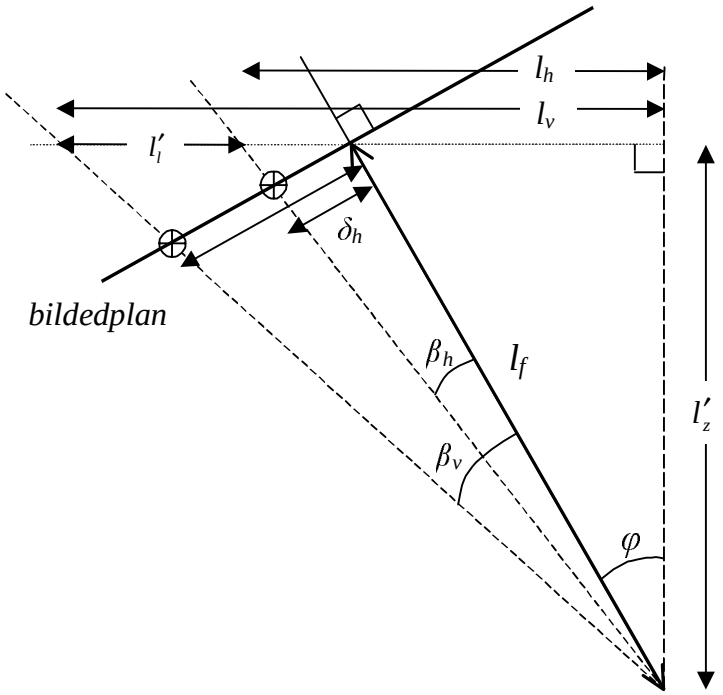


Figur 3.7. Trigonometriske forhold i beregning av avstand til kjøretøy.

For å beregne l_z benyttes formlikheten mellom trekantene *ABC* og *ADE*. Høydene i disse trekantene er henholdsvis l_z' og l_z. Grunnlinjene er henholdsvis l_l' og l_l. Dette gir forholdet:

$$\frac{l_z}{l_z'} = \frac{l_l}{l_l'} \Leftrightarrow l_z = \frac{l_l}{l_l'} l_z' \qquad \textbf{(3.9)}$$

Lengden l_l er avstanden mellom kjøretøyets lykter, og antas å være kjent. Dermed gjenstår det å beregne lengdene l_z' og l_l'.



Figur 3.8. Detaljert utsnitt fra nedre del av Figur 3.7.

Lengden l_z' er enkel å finne da den utgjør den lengste kateten i en trekant hvor vi kjenner hypotenusen og den ene vinkelen. Dette er vist i Figur 3.8. Hypotenusen l_f er fokalavstanden, og vinkelen φ, som er kameraets sikteretning i forhold til veibanen, kan enten måles fysisk eller beregnes fra bildet ved hjelp av selvkalibrerende metoder. Lengden l_z' er dermed gitt av ligning (3.10).

$$l_z' = l_f \cos(\varphi) \qquad \textbf{(3.10)}$$

For å beregne lengden l_l' må man først finne vinklene β_v og β_h. Figur 3.8 viser at disse er gitt av fokalavstanden og lengdene δ_v og δ_h ved følgende forhold:

$$\beta_v = -\arctan\left(\frac{\delta_v}{l_f}\right) \quad \text{og} \quad \beta_h = -\arctan\left(\frac{\delta_h}{l_f}\right) \qquad \textbf{(3.11)}$$

Lengdene δ_v og δ_h er horisontale avstander i bildedplanet. De måles fra midten av bildet og ut til projeksjonen av henholdsvis venstre og høyre lykt. Minus foran inverstangens-funksjonene kommer av retningen som vinklene β er definert med i figuren.

Det er nå mulig å beregne lengden l_l' ved å benytte følgende ligning:

$$l_l' = l_v - l_h = l_z' \tan(\varphi + \beta_v) - l_z' \tan(\varphi + \beta_h) \qquad \textbf{(3.12)}$$

Når ligning (3.12) settes inn i ligning (3.9), viser det seg at lengden l_z' faller bort, og man blir stående igjen med:

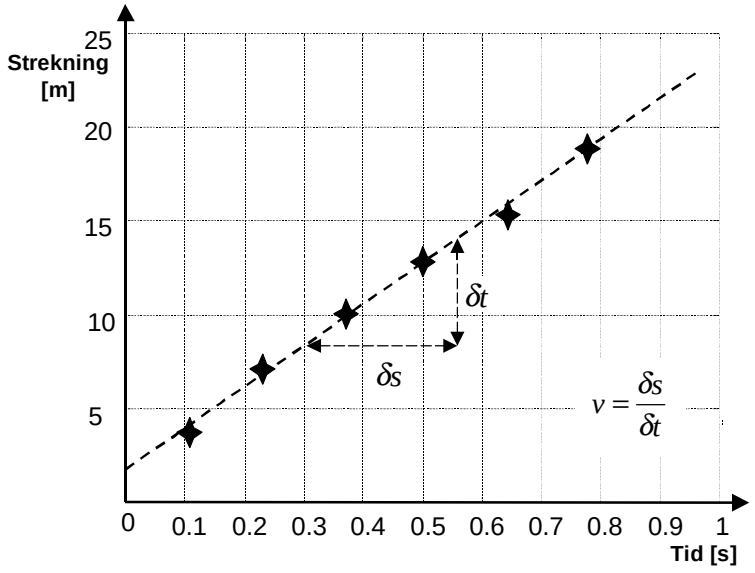
$$l_z = \frac{l_l}{\tan(\varphi + \beta_v) - \tan(\varphi + \beta_h)} \qquad \textbf{(3.13)}$$

Hvor β_v og β_h gitt i ligning (3.11). Dermed kan den horisontale avstanden mellom kamera og kjøretøy beregnes ut fra kun kjente og målbare verdier.

3.3 HASTIGHETSBEREGNING

Enten kamera står på en bro eller er plassert i en bil, vil det bli lagret mer enn ett bilde av hvert kjøretøy som passerer. Avstanden som blir funnet i hvert av bildene vil ikke være nøyaktig. Både diskretiseringen som foregår i filmingen og den billedanalysen som benyttes, vil være kilder til feil. Ved å benytte informasjonen fra flere bilder i hastighetsberegningene kan man redusere effekten av feilkildene.

Avhengig av hastigheten til kjøretøyet og hvordan utstyret er satt opp vil man få mellom 5 og 20 bilder hvor avstanden kan beregnes. Disse avstandene kan plottes som punkter på en graf med tid på x-aksen og strekning på y-aksen. Hastigheten vil være stigningen til den rette linjen som passer best til de punktene som er funnet, slik Figur 3.9 viser.



Figur 3.9. Tilpasning ved hjelp av minste kvadraters metode.

For å finne den beste tilpasningen benyttes minste kvadraters metode. Det vil si at man finner den hastighet *v* slik at summen av kvadratavvikene minimeres.

$$\min_{v} \sum_{i=1}^{N} (s_i - vt_i)^2 \qquad \textbf{(3.14)}$$

Her er N antallet punkter som brukes i beregningene.

Det er verdt å merke seg at kjøretøy beveger seg mer i bildet når de er nærme kameraet. Dette innebærer større endringer i pikselposisjon per tid, dermed vil avstander beregnet fra bilder hvor kjøretøy er nærme ha større nøyaktighet. Ytterligere optimalisering kan derfor oppnås ved å vektlegge korte avstander mer enn lange i hastighetsberegningene.

3.4 NØYAKTIGHET

3.4.1 Relevante faktorer

Metodene for avstands og hastighetsberegning som er blitt beskrevet, er ikke eksakte. Deres nøyaktighet påvirkes i forskjellig grad av:

- Motkommende kjøretøys hastighet
- Kameraets posisjon, orientering og åpningsvinkel (grad av zoom)
- Antall bilder per sekund i AVI-filmen
- Antall bilder som brukes i beregningene
- Hvilken informasjon vi har om motkommende kjøretøy
- Kameraets bildeoppløsning
- Hvor presist pikselposisjonen for kjøretøy og lykter blir beregnet i billedanalysen

De fleste av disse faktorene er det vanskelig å gjøre noe med. Motkommende kjøretøys hastighet har man ingen kontroll over, og kameraoppsettet bestemmes i høy grad av fysiske forhold og at skilt skal være leselige. Heller ikke antall bilder som brukes i beregningene kan manipuleres fritt fordi den er avhengig av de tre første faktorene.

I det følgende skal det sees på tre spesifikke oppsett. I det første tilfellet står kamera på bro og motkommende kjøretøys vertikale posisjon i bildet brukes for å beregne avstand. I tilfelle nummer to er kameraet i en bil. Avstanden mellom lyktene i bildet blir brukt for å beregne avstand. I det tredje tilfellet brukes samme metode som i det første tilfelle, men høyden over veien er redusert og det er lagt til en egenhastighet for kameraet på 80 km/t.

I alle tilfellene er det valgt ”fornuftige” verdier for kameraets orientering, posisjon og åpningsvinkel. Antall bilder per sekund i AVI-filmen og kameraets bildeoppløsning er

valgt som for et SONY TRV-900 kamera med progressiv scan (se side 12). Det vil si 12.5 bilder per sekund og en vertikal oppløsning for chipen på 640×480 punkter. TRV-900 er et relativt vanlig konsument-produkt i 1999.

3.4.2 Simuleringer

For å teste nøyaktigheten i de tre tilfellene er et simuleringsprogram utviklet i Matlab. Programmet kjører tester for fire forskjellige hastighetsområder, 60-80 km/t, 80-100 km/t, 100-120 km/t og 120-140 km/t. I de to siste scenarioene er det lagt til 80 km/t for kameraets egen hastighet. I hver hastighetsserie blir det gjennomført 1000 kjøringer med gradvis økende hastighet for kjøretøyet som kommer imot. Programmet beregner først posisjonene langs veien som kjøretøyet vil ha under filmingen. Deretter projiseres posisjonene inn i bildeplanet. En egen test forteller hvilke bilder i filmen som faktisk viser kjøretøyet, slik at bildet kan brukes til avstandsberegninger. Videre blir kjøretøyets posisjoner i projeksjonen diskretisert. Denne diskretiseringen tilsvarer samplingen som foregår i den lysfølsomme chipen i et digitalt kamera.

Kjøretøyets vertikale pikselposisjon og lyktposisjoner blir lagret og deretter sendt til en programmodul som analyserer avstanden til kjøretøyet. Til slutt analyseres hastighet, og det beregnes verdier som beskriver nøyaktigheten i analysen. Den viktigste av disse verdiene er hastighetens standardavvik. Den alene forteller hvilken nøyaktighet man statistisk sett kan forvente av beregningene. Kildekode for Matlab-programmet er dokumentert i Appendiks B.

I det første tilfellet med kamera på bro brukes følgende parameterverdiene for kameraets posisjon og orientering:

- Horisontal posisjon: Midt over kjørefeltet som filmes
- Vertikal posisjon: 7.9 m over veibanen
- Sikteretning i vertikalplanet : 11 graders vinkel ned mot veibanen
- Sikteretning i horisontalplanet: 0 grader, dvs. langs veibanen
- Vinkelåpning: 5 grader

Resultater fra kjøring i Matlab er gjengitt i Tabell 1.

	60–80 km/t	80–100 km/t	100–120 km/t	120–140 km/t
Gjennomsnittelig avvik [km/t]	0.0865	0.1119	0.1369	0.1678
Standardavvik for avviket [km/t]	0.0554	0.0741	0.0933	0.1088
Maksimalt avvik under 1000 kjøringer [km/t]	0.2632	0.3375	0.4833	0.5287
Gjennomsnittelig antall bilder brukt	8.4735	6.3237	4.9840	4.1339

Tabell 1. Resultater fra nøyaktighetsanalysen. Kamera på bro.

Tabell 1 viser at et opptak fra bro, som er gjort av en lysfølsom chip med oppløsning 640×480, i verste fall får et standardavvik for hastigheten på 0.1088 km/t. Det vil si at kjøretøy med hastighet mellom 120 og 140 km/t i 99.9% av tilfellene måles med en feil mindre enn 0.36 km/t, og i 99.99% av tilfellene vil feilen være mindre enn 0.42 km/t. I denne testen er det riktignok ikke tatt hensyn til at billedanalysen ikke nødvendigvis returnerer kjøretøyets eksakte pikselposisjon i bildet. Likevel må resultatene sies å være svært lovende.

Man bør legge merke til at nøyaktigheten synker betydelig med økt hastighet. Dette har sammenheng med at kjøretøyet passerer skjermbildet raskere, og at det derfor er færre bilder å beregne hastigheten ut fra.

I det andre tilfellet er kameraet i bil, og følgende parameterverdier brukes for kameraets posisjon og orientering:

- Horisontal posisjon: Midt i høyre kjørefelt
- Vertikal posisjon: 1.5 m over veibanen
- Sikteretning i vertikalplanet : 2 graders vinkel ned mot veibanen
- Sikteretning i horisontalplanet: 10 grader mot venstre kjørefelt
- Vinkelåpning: 12 grader
- Egenhastighet 80 km/t

Resultater fra kjøringene er gjengitt i Tabell 2.

	60–80 km/t	80–100 km/t	100–120 km/t	120–140 km/t
Gjennomsnittelig avvik [km/t]	1.1465	1.3184	1.4565	1.6241
Standardavvik for avviket [km/t]	0.8246	0.9535	1.0631	1.1660
Maksimalt avvik under 1000 kjøringer [km/t]	3.5780	4.5031	4.7665	4.9593
Gjennomsnittelig antall bilder brukt	8.5495	7.4086	6.5305	5.8452

Tabell 2. Resultater fra nøyaktighetsanalysen. Kamera i bil.

Fra Tabell 2 ser man at selv uten feil i målingene av lyktenes posisjon i bildet vil hastigheten ha et standardavvik på 1.166 km/t. Dette innebærer at man i 0.1% av tilfellene gjør en feil større enn 3.0 km/t, og i 0.01% prosent av tilfellene er feilen større enn 3.9 km/t. Dette beveger seg på kanten av hva som kan tolereres, og enda er det mange usikkerheter som ikke er tatt med i beregningene.

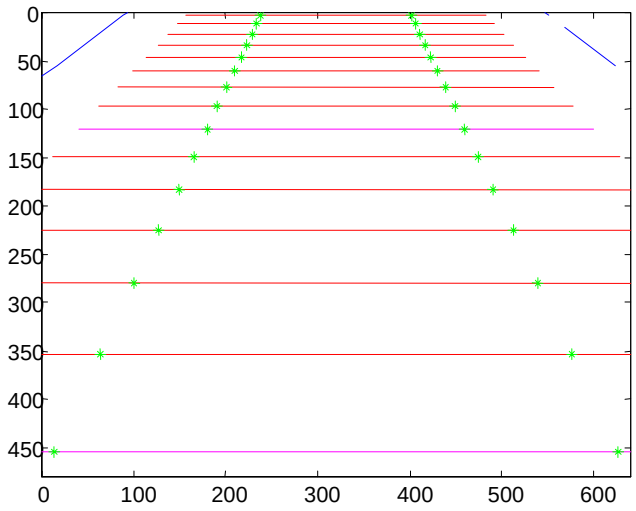
I det tredje tilfellet benyttes avstandsberegning ut fra vertikal posisjon i bildet, men høyden over veien er satt til 1.5 m. I tillegg er det lagt til 80 km/t for kameraets egenhastighet. Formålet med disse simuleringene er å finne sammenhengen mellom nøyaktighet og kamerahøyde for denne metoden. Man bør merke seg at å endre

kameraets høyde innebærer også at man må forandre kameraets orientering, derfor er ikke resultatene fullstendig sammenlignbare.

Følgende parameterverdier er brukt:

- Horisontal posisjon: Midt over kjørefeltet som filmes
- Vertikal posisjon: 1.5 m over veibanen
- Sikteretning i vertikalplanet : 5 graders vinkel ned mot veibanen
- Sikteretning i horisontalplanet: 0 grader (rett frem)
- Vinkelåpning: 4 grader
- Egenhastighet 80 km/t

Figur 3.10 viser den simulerte projeksjonen for disse parameterverdiene. De blå diagonale linjene i øvre del av bildet representerer kjørefeltets avgrensninger. Røde linjer markerer kjøretøyets posisjoner ved forskjellige tidspunkter, og grønne stjerner markerer kjøretøyets bredde. I de to foregående tilfellene ble alle bilder hvor kjøretøyet var synlig, brukt i beregningene. På grunn av den noe spesielle kameravinkelen i dette tilfellet er det valgt å konsekvent bruke de 7 bildene hvor kjøretøyet er nærmest. Første og siste posisjon er markert med fiolette linjer i bildet.



Figur 3.10. Simulert kameraprojeksjon.

Resultater:

	60–80 km/t	80–100 km/t	100–120 km/t	120–140 km/t
Gjennomsnittelig avvik [km/t]	0.4162	0.5247	0.2702	0.6310
Standardavvik for avviket [km/t]	0.2516	0.3392	0.4069	0.4511
Maksimalt avvik under 1000 kjøringer [km/t]	1.1746	1.1969	1.5794	1.8795
Gjennomsnittelig antall bilder brukt	7	7	7	7

Tabell 3. Resultater fra nøyaktighetsanalysen. Samme metode som for kamera på bro. Kameraets høyde over vei er satt til 1.5 meter.

Tabell 3 viser at for de høyeste hastighetene og med nøyaktig måling av kjøretøyets vertikale posisjon i bildet, beregnes hastigheten med et standardavvik på 0.451. Dette er omtrent fire ganger større enn ved kamerahøyde på 8 meter, men det er fortsatt halvparten så stort som ved bruk avstanden mellom lyktene i bildet. Dette tyder på at metoden som bruker kjøretøyenes vertikale posisjon i bildet, bør vurderes også for filming fra bil.

3.5 KRAV

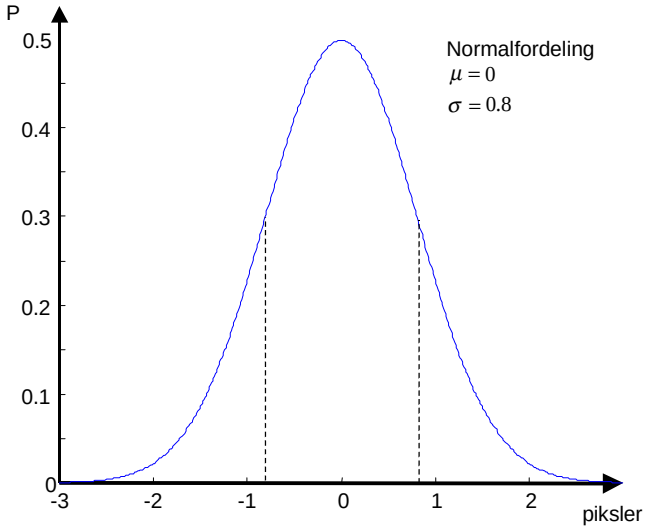
Simuleringene ovenfor forteller hvilke resultat man i beste fall kan forvente med et gitt oppsett og bildeoppløsningen på dagens DV-kameraer. De tar ikke hensyn til at billedanalysen ikke nødvendigvis finner den riktige pikselposisjonen for kjøretøyet eller lyktene. Her vendes det på problemstillingen. To parametere skal varieres, den ene er bildeoppløsningen, og den andre er standardavviket for kjøretøyets pikselposisjon vertikalt i bildet (målt i piksler). Dette standardavviket er et mål for hvor nøyaktig billedanalysen beregner kjøretøyets posisjon i bildet.

Simuleringen foregår på følgende måte. Først beregnes standardavvik for hastighetsberegningen for forskjellige verdier av bildeoppløsningen, mens standardavviket for pikselposisjon holdes ved en fast verdi. Deretter beregnes standardavviket for hastighetsberegningen for forskjellige verdier av standardavviket for pikselposisjon, mens bildeoppløsningen holdes konstant.

Også her benyttes Matlab-program for å analysere et tilfelle med kamera på bro og et tilfelle med kamera i bil. Paremeterverdiene for disse to scenarioene er identiske med parameterverdiene i de to første tilfellene som ble brukt i nøyaktighetsanalysen.

Når scenarionene analyseres for et fast standardavvik for pikselposisjonen, er standaravviket satt til 0.8 piksler. Denne verdien er valgt ut fra antagelsen om at posisjonen i de fleste tilfeller måles med tre pikslers nøyaktighet eller bedre. Mer nøyaktig betyr det at billedanalysen finner riktig pikselposisjon i 47% av tilfellene,

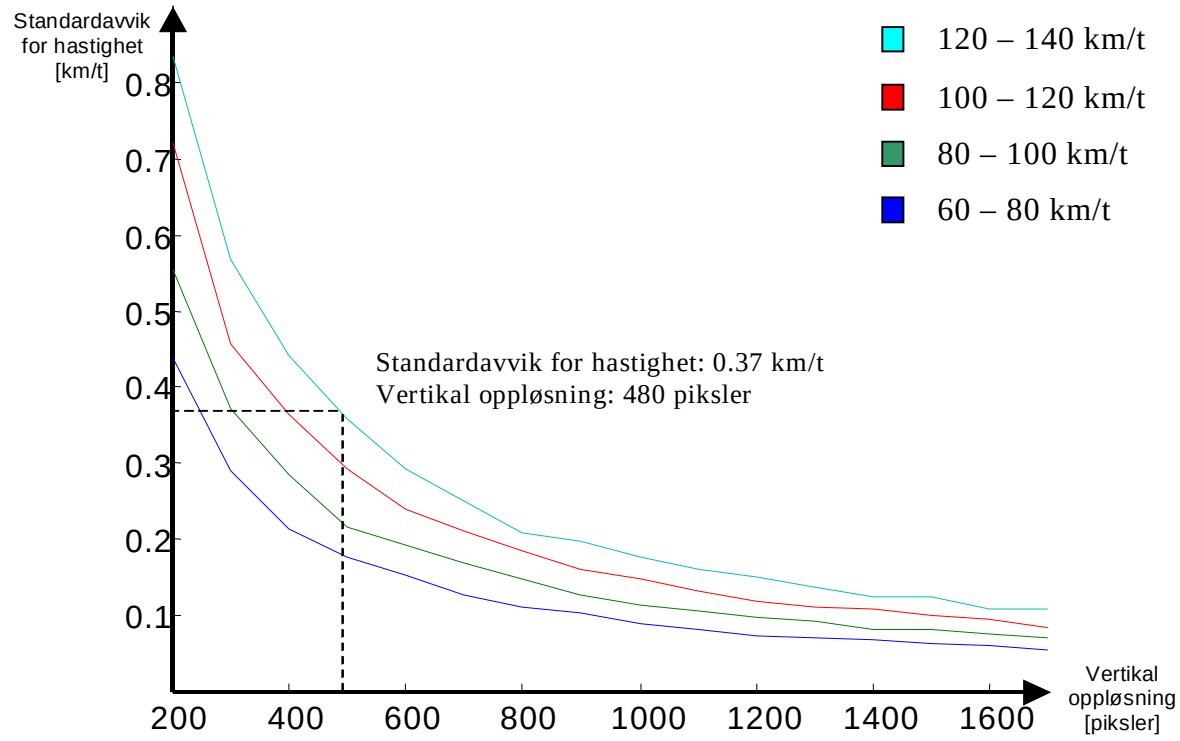
bommer med en piksel i 47% av tilfellene og bommer med to piksler i 5.9 % tilfellene. Figur 3.11 viser sannsynlighetsfordelingen.



Figur 3.11. Sannsynlighetsfordeling for avviket i pikselposisjon beregnet i billedanalysen.

3.5.1 Kamera på bro

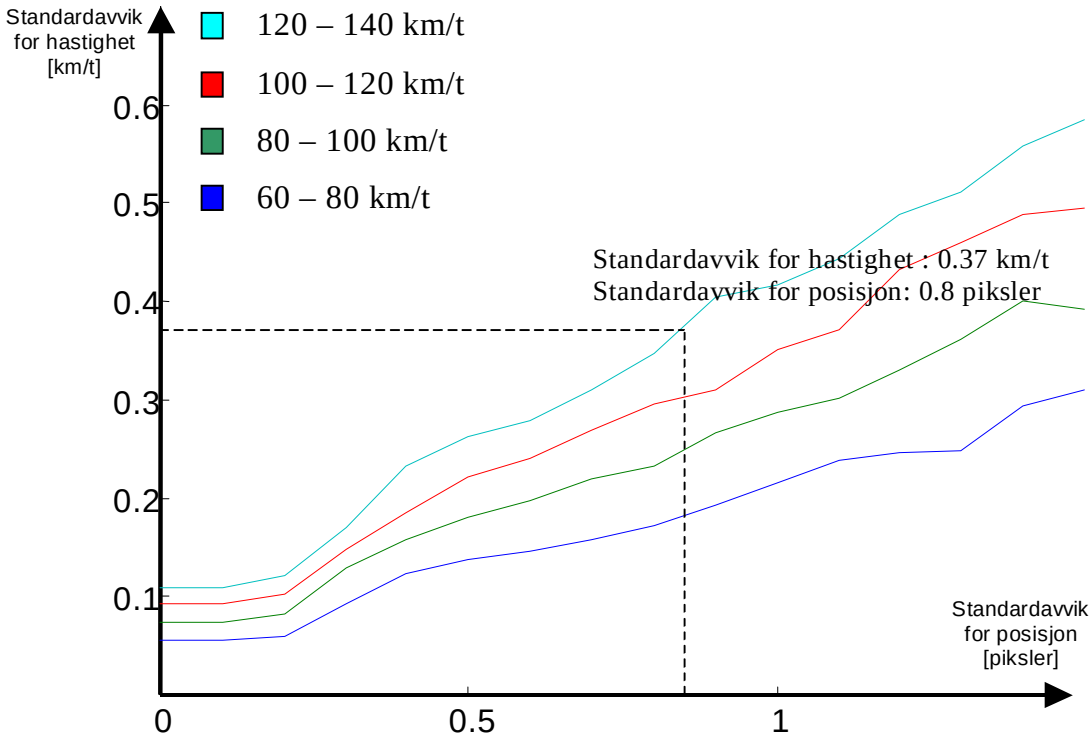
Som tidligere blir hastigheten beregnet ut fra kjøretøyets vertikale posisjon i bildet. Nøyaktigheten er da kun avhengig bildets vertikale oppløsning. Resultat for oppløsninger mellom 200 og 1600 piksler er gitt i Figur 3.12. Hastighetens standardavvik for en oppløsning på 480 piksler er merket av med stiplet linje. Disse kjøringene er som tidligere nevnt gjort for en pikselposisjon i bildet med standardavvik på 0.8 piksler.



Figur 3.12. Varierende vertikal oppløsning. Standardavvik for posisjon: 0.8 piksler.

Fra grafen kan vi se at nøyaktigheten kan økes betydelig ved en fordobling av den vertikale oppløsningen til 960 piksler. Høyere vertikal oppløsning enn dette gir ikke like stort utbytte i form av nøyaktighet, fordi kurvene ser ut til å ha et negativ eksponentielt forløp. Med vertikal oppløsning på 480 piksler er standardavviket for hastigheten 0.37 km/t. Dette gir i 99.9% av tilfellene en feil mindre enn 1.22 km/t, og i 99.99% av tilfellene en feil mindre enn 1.45 km/t. Dette betyr at dagens videokameraer gir god nøyaktighet i målingene.

Figur 3.13 viser resultater for varierende standardavvik i pikselposisjon og konstant vertikal oppløsning på 480 piksler. Her er et standardavviket for hastigheten merket av for et standardavvik i pikselposisjon på 0.8 piksler.

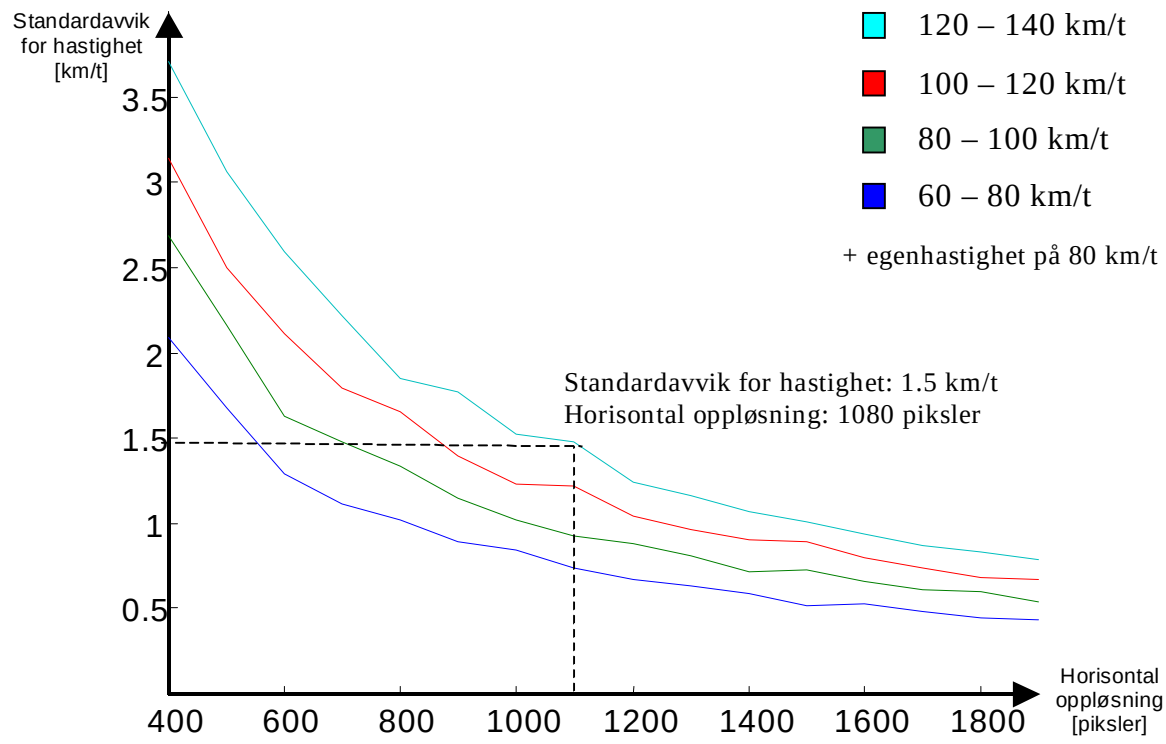


Figur 3.13. Varierende standardavvik for pikselposisjon. Vertikal oppløsning: 480 piksler.

Disse kurvene ser ut til å ha et lineært forløp etter at standardavviket har kommet over 0.2 piksler. Nøyaktigheten blir åpenbart mer følsom for hastigheten ved økende standardavvik (kurvene skilles mer og mer), men selv for et standardavvik på 1.5 piksler og hastigheter mellom 120 og 140 km/t beregnes hastigheten i 99.99% av tilfellene med en feil mindre enn 2.35 km/t.

3.5.2 Kamera i bil

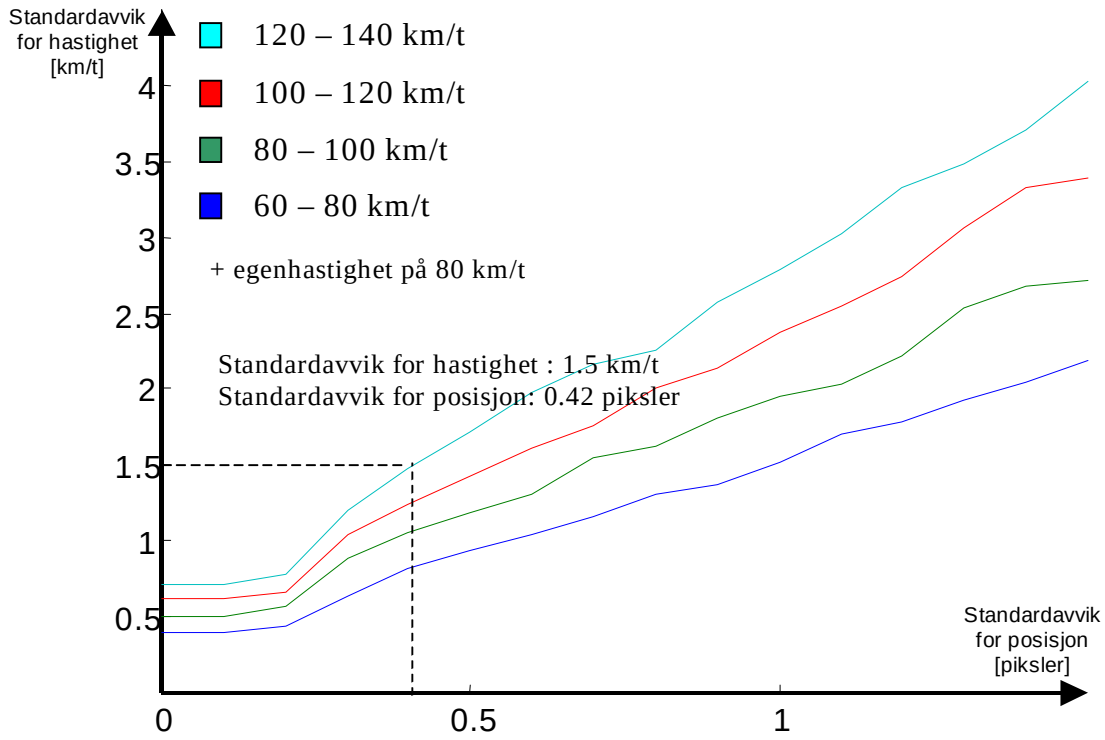
Her benyttes den horisontale avstanden mellom lyktene i bildet for å beregne avstand. Denne metoden er ikke følsom for endringer av oppløsningen i vertikal retning, men den er følsom for oppløsningen i horisontal retning. I dette scenarioet er det lagt til en egenhastighet for kameraet på 80 km/t. Resultater for varierende horisontal oppløsning, og et konstant standardavvik for målingen av lyktenes horisontale posisjoner på 0.8 piksler, er gitt i Figur 3.14.



Figur 3.14. Varierende horisontal oppløsning. Standardavvik for posisjon: 0.8 piksler.

Grafen viser at dersom man har et standardavvik for målingen av lyktenes posisjon i bildet på 0.8 piksler, og det settes krav om at hastigheten må beregnes med et standardavvik mindre enn 1.5 km/t, så må man bruke et kamera med horisontal oppløsning på minimum 1080 piksler. Dette er betydelig høyere enn det som er vanlig i dagens DV-kameraer, men det viser at metoden sannsynligvis vil være god nok i fremtiden.

Figur 3.15 viser resultater for varierende standardavvik i pikselposisjon og konstant horisontal oppløsning på 640 piksler. I figuren er en verdi for hastighetens standardavvik på 1.5 km/t merket. Dette kan sees på som et minimumskrav for nøyaktigheten til hastighetsberegningene.



Figur 3.15. Varierende standardavvik for pikselposisjon. Horisontal oppløsning: 640 piksler.

Her viser grafen at dersom den horisontale oppløsningen er på 640 piksler, slik som på dagens DV-kameraer, må lyktenes posisjon finnes med et standardavvik mindre enn 0.4 piksler. Dette betyr at billedanalysen må finne den eksakte pikselposisjonen i 79% av tilfellene, hvilket ikke er særlig realistisk.

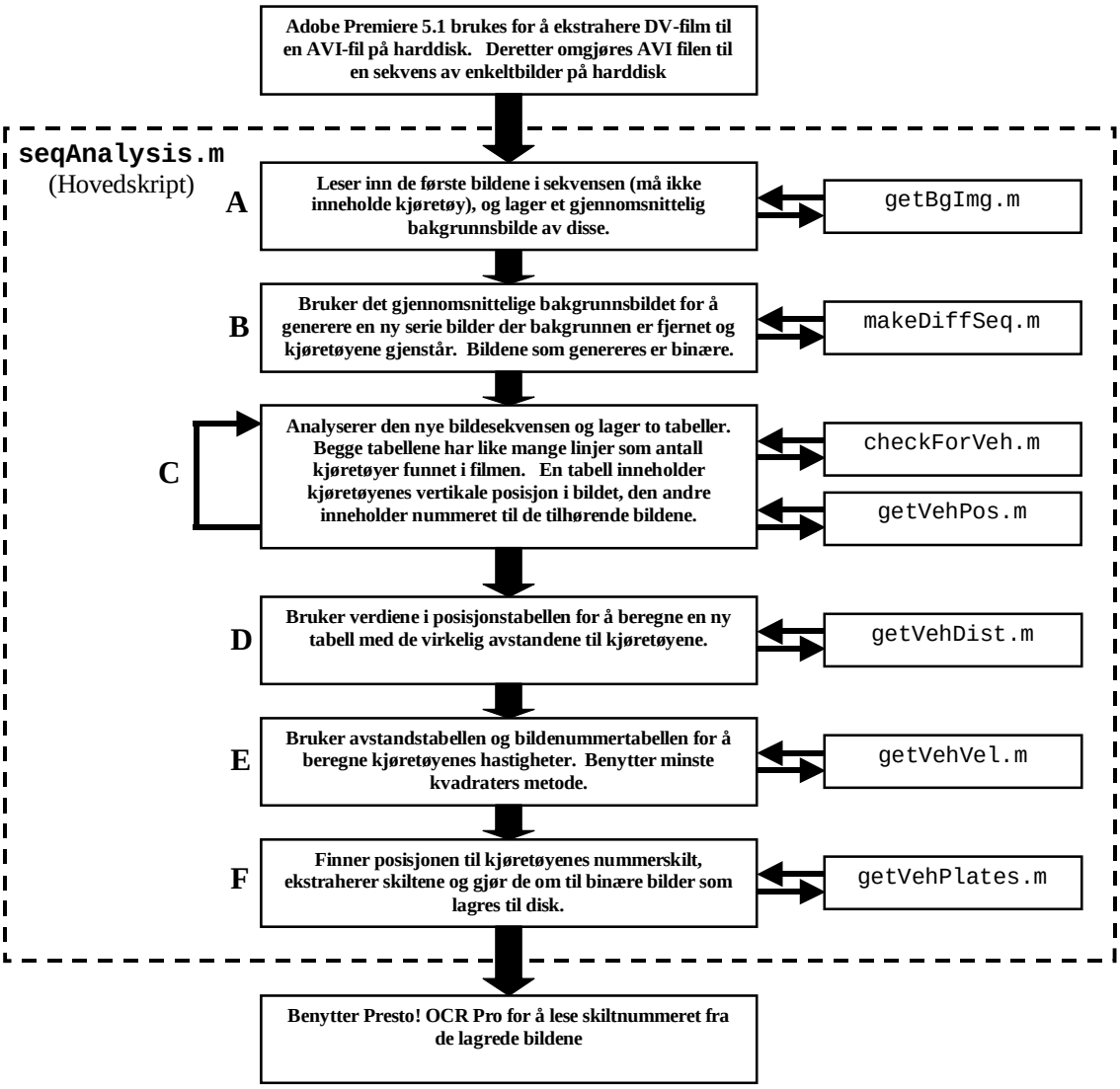
4 IMPLEMENTERING

Dette kapitlet dokumenterer implementeringen av et program som analyserer opptak fra bro. Programmet er implementert i Matlab, og tar utgangspunkt i enkeltbilder som er ekstrahert fra AVI-video. Bildene blir filtrert og analysert for å finne kjøretøy og tilhørende posisjoner. Programmet returnerer kjøretøyenes hastighet og lagrer bilder av nummerskiltene på disk. Nummerskiltene er klargjort for lesing av et OCR-program.

4.1 STRUKTUR

Figur 4.1 viser et flytskjema for hele prosessen. Delprosessene innenfor den stiplede boksen utføres av et skript som er implementert i Matlab. De to delprosessene utenfor den stiplede boksen utføres av eksterne programmer. Den første prosessen er ekstraheringen av DV-film fra kameraet, til AVI-film på harddisk, og i tillegg omgjøringen til enkeltbilder. Til dette brukes Adobe Premiere 5.1 ®. Premiere er et kommersielt program for redigering av digital video, som har ferdig definerte funksjoner for ekstrahering av enkeltbilder. Den andre eksterne prosessen er tekstgjenkjenningen for lesing av nummerskilt. Til dette benyttes OCR-programmet Presto! OCR Pro ®. Dette er også et kommersielt program. Presto tar inn svart-hvitt bilder av type TIFF og returnerer hvis mulig, skriften i tekstformat.

Det er valgt å bruke eksisterende programmer i disse delene av prosessen for å kunne benytte mer tid til bearbeiding av billedanalysedelen og den matematiske teorien i problemstillingen. Om man i fremtiden skulle ønske å videreutvikle denne ideen kommersielt, vil det være nødvendig å utvikle disse delene selv.



Figur 4.1. Flytskjema for den fullstendige prosessen.

4.2 PROGRAMDELER

Til implementering av prosessene innenfor den stiplede boksen er Mathworks Matlab versjon 5.3.0 benyttet. Foruten Matlabs standardfunksjoner er det også tatt i bruk en del funksjoner fra Image Processing Toolbox versjon 2.2. For detaljerte beskrivelser av Matlabs funksjoner refereres det til *Image processing toolbox user's guide* **[7]** og *Matlab online manuals* **[8]**.

4.2.1 Hovedskript

Hovedskriptet har navnet `seqAnalyser.m`, og er delt opp ti hoveddeler. Disse er merket på følgende måte i kildekoden (Kildekoden finnes i Appendiks B):

```
*** DELER AV PROGRAMMET SOM SKAL KJØRES ***
************* GLOBALE VARIABLE ************
********** PARAMETERE FOR OPPSETT *********
********** PARAMETERE FOR PROGRAM *********
***************** PROSESS A *****************
***************** PROSESS B *****************
***************** PROSESS C *****************
***************** PROSESS D *****************
***************** PROSESS E *****************
***************** PROSESS F *****************
```

Hvilke av prosessene A til F som skal kjøres bestemmes av variablene som står under den første delen av skriptet. Verdi 1 betyr at prosessen kjøres, verdi 0 betyr at den blir oversett. Dette kan spare en del tid hvis man ønsker å finjustere parametere. Etter programmet er kjørt en gang vil alle variable fortsatt ligge i arbeidsområdet. De som ikke påvirkes av justeringen trenger ikke å beregnes på nytt.

Andre del av skriptet definerer de globale variablene. Det blir brukt globale variable for følgende verdier:

- Kameraets orientering og posisjon i forhold til veibanen.
- Kameraets åpningsvinkel og fokuslengde
- Navn på katalogen hvor de originale enkeltbildene fra filmen ligger
- Navn på katalogen hvor bildene med isolerte kjøretøy skal ligge
- Bildeformatet til alle enkeltbilder
- Bildenes oppløsning i vertikal og horisontal retning

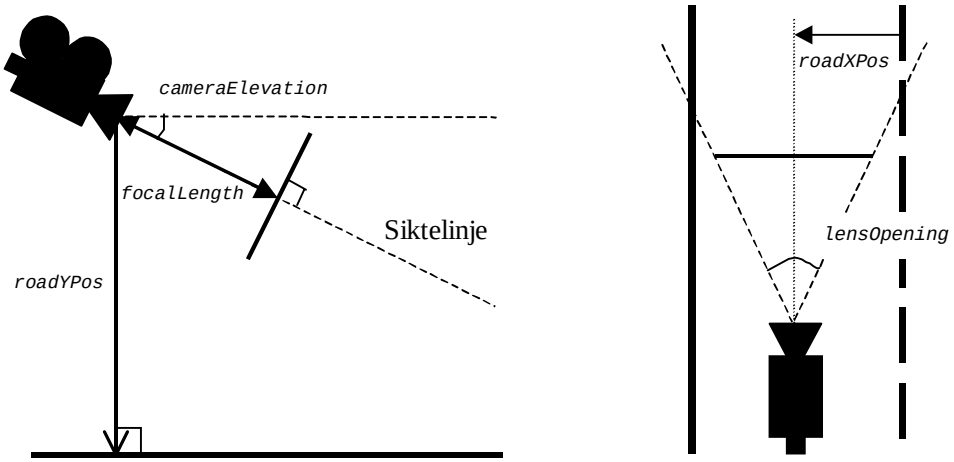
Grunnen til at disse variablene gjøres globale er at de tas i bruk av flere funksjoner under kjøring. En forandring i verdi for en av disse variablene, vil gjelde for alle funksjoner som kalles av hovedskriptet.

Neste del gir initialverdier til parameterne for oppsett. Med det menes de parameterne som sier noe om kamera og veibanen. Følgende av disse parameterne kan være interessante å endre på ved bruk av nye oppsett i fremtiden (Alle posisjoner er definert ut fra et koordinatsystem med origo i kameralinsen):

`roadXPos`	Midtstripens posisjon i forhold til kamera (på tvers av vei)
`roadYPos`	Veibanens vertikale posisjon i forhold til kamera.
`cameraElevation`	Kameraets sikteretning i forhold til veibanen (vertikalt)
`cameraAsimuth`	Kameraets sikteretning i forhold til veibanen (horisontalt)

`focalLength`	Fokallengde (avstand mellom linse og bildeplan)
`lensOpening`	Åpningsvinkel (grad av zoom)
`resolutionX`	Den lysfølsomme chipens horisontale oppløsning
`resolutionY`	Den lysfølsomme chipens vertikale oppløsning
`frameRate`	Bilder per sekund i AVI-filmen

Figur 4.2 illustrerer hvor parameterne inngår i oppsettet.



Figur 4.2. Oppsett med parametere.

Hver av prosessene A, B, D, E, F (ikke C) i Figur 4.1 har én enkelt funksjon knyttet til seg. Disse funksjonene kjøres kun én gang, og de returnerer enten verdier, tabeller eller bilder som behandles videre av neste prosess. Prosess C skiller seg ut. Den inneholder nøstede løkker for leting og posisjonsmåling av kjøretøy. Grundigere forklaring av denne prosessen er gitt lenger ut i dette kapitlet.

Ved start forventer hovedskriptet at de originale enkeltbildene fra AVI-videoen skal ligge lagret på en kjent katalog. Ved slutt vil de filtrerte bildene av kjøretøyenes ekstraherte nummerskilt være lagt på disk, og tabellene med avstander, hastigheter, og respektive bilder vil ligge i arbeidsområdet.

4.2.2 Prosess A - Generering av bakgrunnsbilde

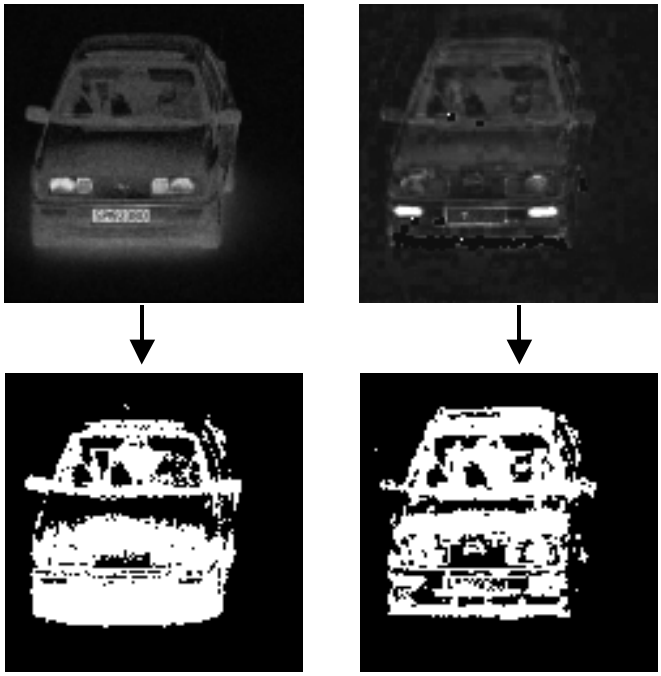
For å generere bakgrunnsbilde er det lagd en funksjon som heter `getBgImg()`. Denne funksjonen blir kalt av hovedskriptet og tar to parametere. Den første er bildenummeret for det første bildet i serien som skal brukes. Den andre parameteren er antall bilder som skal brukes for å generere bakgrunnsbildet.

Funksjonen lager et midlertidig bilde bestående av tre matriser, en for hver av fargene rød, grønn og blå. Matrisene har samme dimensjoner som bildene fra sekvensen. Alle bildets verdier blir i utgangspunktet satt til null. For å lese inn bilder fra harddisk brukes Image Toolbox-funksjonen `imread()`. Fargeintensitetene i det første bildet legges til verdiene i det midlertidige bildet. Deretter hentes neste bildet i sekvensen og verdiene legges til. Når alle bildene er blitt summert deles verdiene i det midlertidige bildet på antallet bilder summert. Dette gir det gjennomsnittelige bilde som returneres til hovedskriptet. Funksjonen har ingen parametere som er aktuelle å justere.

4.2.3 Prosess B – Isolering av kjøretøy

Når hovedskriptet kaller funksjonen `makeDiffSeq()` sender den med bakgrunnsbildet beregnet i prosess A, som en parameter. Den sender også med nummeret for det første og siste bildet i sekvensen. Funksjonen går igjennom annethvert bilde mellom det første og siste bildet, og filtrerer disse. Grunnen til at bare annethvert bilde bearbeides er at to og to bilder i den originale sekvensen er like. Dette er alltid tilfellet når man jobber med DV-film som er tatt opp med progressiv scan (se side 12).

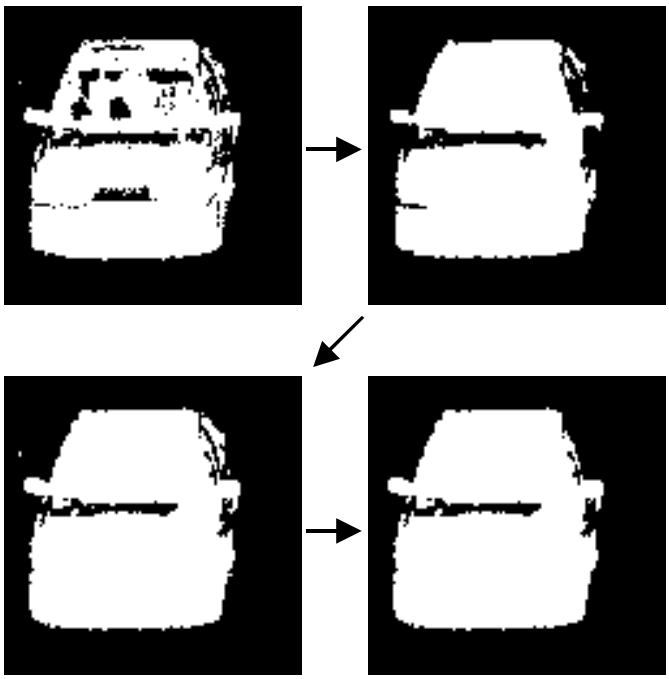
Selve filtreringen gjøres av funksjonen `getDiffImg()`, som mottar et bilde fra den originale sekvensen og bakgrunnsbildet. Metoden for å generere en fargedifferansematrise og en intensitetsdifferansematrise for disse to bildene er nøye forklart i billedanalysen, så her skal det fokuseres på hvilke Matlab-funksjoner som er benyttet i implementeringen.



Figur 4.3. Del 1 av filtreringsprosessen.

De to øverste bildene i Figur 4.3 er henholdsvis intensitetsdifferansen og fargedifferansen mellom det gjennomsnittelige bakgrunnsbildet og et bilde med kjøretøy. De nedre bildene er binære versjoner av de øvre bildene. Det vil si at de har blitt filtrert med et filter som gjør alle intensitetsverdier høyere enn en bestemt terskel til enere, og de som er lavere enn terskelen til nuller. Til dette er Matlab-funksjonen `im2bw()` benyttet. Tersklene som er brukt for de to tilfellene er gitt av variablene `intTol` og `colTol`. Disse tersklene må være mellom 0 og 1. Høye verdier lar mer hvitt passere, lavere verdier lar mindre hvitt passere.

I overgangen til bildet øverst til venstre i Figur 4.4, er en *eller*-funksjon utført mellom de to binære bildene. Det vil si at piksler som er på i minst et av bildene, vil være på i det resulterende bildet.



Figur 4.4. Del 2 av filtreringsprosessen.

Mellom de to øverste bildene i Figur 4.4 er alle svarte områder som var omringet av hvitt blitt fylt med kommandoen `bwfill(image,'holes')`. I tillegg er bildet erodert. Det vil si at alle hvite objekter har mistet piksler i ytterkant. I Matlab brukes funksjonen `erode()`. I neste overgang har bildet blitt dilatert. Det innebærer at man legger hvite piksler rundt alle objekter med kommandoen `dilate()`. Erosjon etterfulgt av dilatasjon glatter objekter og fjerner støy. Til slutt fjernes alle hvite objekter bortsett fra det største. Dette gjøres av funksjonen `getBig()`.

Funksjonen `getbig()` er ikke en standard kommando i Matlab. Den finner det største objektet i et bilde ved å benytte Image Toolbox-kommando `bwlabel()`. Kommandoen

`bwlabel()` tar inn et binært bilde og returnerer antall objekter i bildet og en matrise. Matrisen har samme dimensjoner som bildet og inneholder objektene merket med sine respektive objektnummer. Dette er illustrert i Figur 4.5.

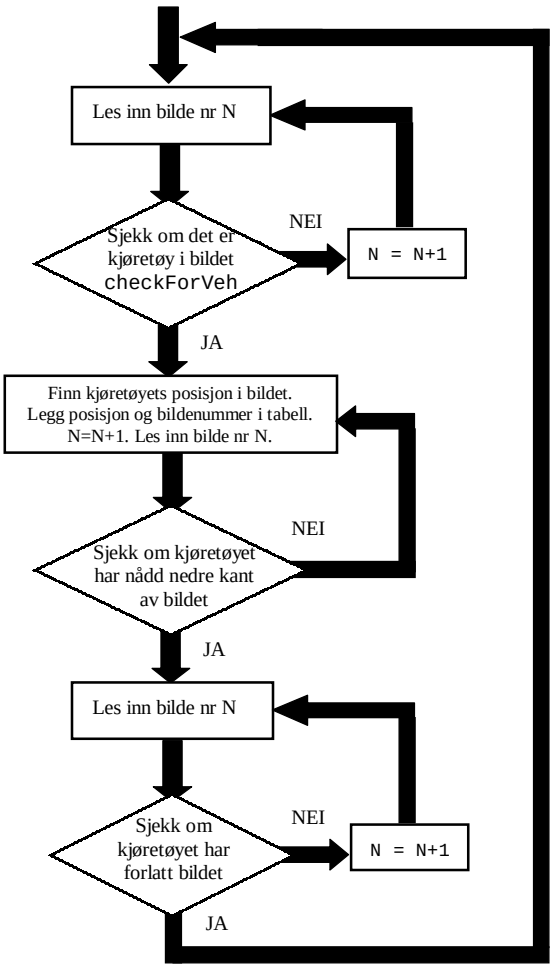
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 \\ 3 & 3 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \end{bmatrix}$$

Figur 4.5. Eksempel ved bruk av `bwlabel()`.

Funksjonen `getBig()` teller antall elementer som har samme tallverdi i matrisen ved å bruke `find()`, og finner på den måten det største objektet i bildet. Den nye bildet som bare inneholder det største objektet genereres av kommandoen `bwselect()` som tar matrisen og objektnummer som inn-parametere. I `getBig()` er det en justerbar parameter `minNumOfPixels`. Dersom det største objektet består av færre piksler enn denne parameterverdien, så forkastes objektet. Dette er for unngå støy i bilder som ikke inneholder kjøretøy.

4.2.4 Prosess C – Generering av posisjons- og bildetabell

Virkemåten for denne prosessen forklareres best med et flytskjema.



Figur 4.6. Flytskjema for generering av posisjon- og bildenummertabell.

Figur 4.6 viser at prosessen er tredelt. Første del benytter funksjonen `checkForVeh()` for å finne frem til neste bilde i filmen hvor det er kjøretøy. Funksjonen `checkForVeh()` er svært enkel, den teller antall hvite piksler i bildets nederste linjer. Hvis antallet piksler er høyere enn terskelen `pixelTol`, returnerer funksjonen 1, ellers returnerer den 0. Antallet linjer som det summeres over er satt av variabelen `linesToCheck`.

Når et kjøretøy blir funnet går prosessen videre til neste del. Denne tar for seg serien av bilder fra bilen kommer inn i øvre kant av skjermbildet, og fram til den når nedre kant. For hvert bilde kalles funksjonen `getVehPos()`, som leter seg oppover i bildet inntill den finner kjøretøyet. Virkemåten for denne funksjonen er nesten den samme som for `checkForVeh()`, med den forskjell at `getVehPos()` prøver igjen en linje lenger opp på bildet for hver gang testen returnerer 0. Når testen slår til returnerer funksjonen kjøretøyets vertikale posisjonen i bildet. Posisjoner og bildenummer blir lagret i hver sin tabell. Tabellene har én linje for hvert kjøretøy som blir funnet, og én kolonne for

hver posisjon og bilde som registreres. Funksjonens justerbare parametere er de samme som for `checkForVeh()`.

Den siste delen av prosessen leter seg videre i bildeserien inntill `checkForVeh()` returner null. Det vil si at kjøretøyet har kjørt ut av skjermbildet i nedre kant. Deretter blir hele prosessen gjentatt. Dette foregår helt til alle bilder i serien er analysert.

4.2.5 Prosess D – Avstandsberegning

Denne prosessen kaller funksjonen `getVehDist()`. Funksjonen tar posisjonstabellen som innparameter og returnerer en tabell med avstander. Det matematiske grunnlaget for beregning av avstander ut fra vertikal posisjon i bilde er gjennomgått i kapitlet Matematisk Analyse. Det er ligning (3.8) som er implementert i `getVehDist()`.

4.2.6 Prosess E – Hastighetsberegning

Det eneste som mangler for å kunne beregne hastigheten ved hjelp av minste kvadraters metode er å lage en tidsvektor og en strekningsvektor. Tidsvektoren beregnes ut fra bildevektoren. Hvis ingen bilder er uleselige så vil denne bildevektoren inneholde verdier som øker med 2 for hvert element (bare annethvert bilde blir brukt). DV-film har en bilderate på 25 bilder per sekund, dvs. 0.04 sekunder mellom hvert bilde. Det første tidspunket i tidsvektoren defineres til å være 0, og de resterende elementene beregnes med følgende formel:

$$tid[i] = tid[i-1] + \frac{bildeNr[i+1] - bildeNr[i]}{bilderate} \qquad \textbf{(4.1)}$$

Strekningsvektoren er lik den beregnede avstandsvektoren, men defineres i likhet med tidsvektoren til å være null for første element. Det vil si at strekningen er null ved tid null. Strekningsvektoren beregnes ved å trekke den første avstanden i avstandsvektoren fra alle avstandene som er registrert.

$$strekning[i] = avstand[i] - avstand[1] \qquad \textbf{(4.2)}$$

Tidsvektoren og avstandsvektoren for hvert kjøretøy mates inn i Matlabs funksjon for løsning av lineære problem med minste kvadraters metode. Funksjonen heter `lsqlin()`. Den tar vektorene *C, d, A, b* og løser,

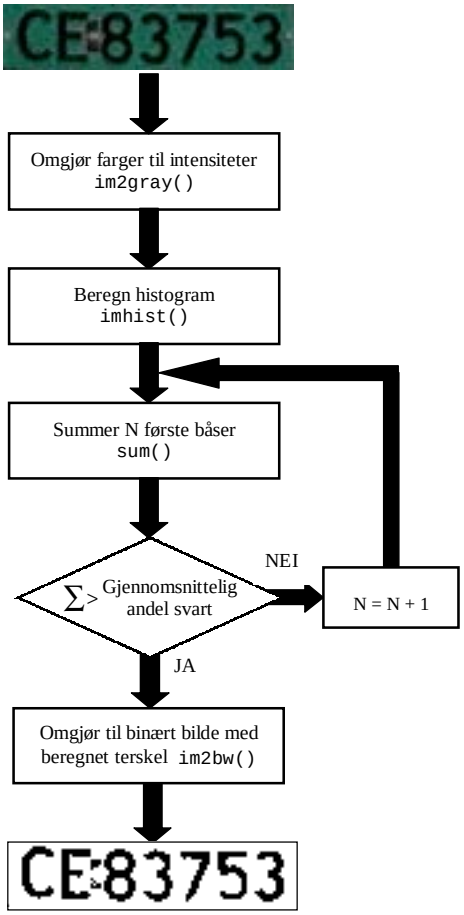
$$\min_x 0.5 \cdot \left(\left|C \cdot x - d\right|\right)^2 \quad \text{med} \quad A \cdot x \le b \qquad \textbf{(4.3)}$$

A og *b* beskriver begrensninger som ikke er nødvendige her, så disse settes til 0. Variabelen *x* som ligningen minimeres over, tilsvarer i dette tilfellet hastigheten.. *C* og *d* er henholdsvis tidsvektoren og strekningsvektoren. Funksjonen returnerer den minimerende verdien for *x*.

4.2.7 Prosess F – Ekstrahering og filtrering av nummerskilt

Fremgangsmåten for å finne nummerskilt i et bilde er nøye forklart i kapitlet Billedanalyse. Implementeringen som er gjort her bruker to nøstede løkker som går igjennom samtlige av bildets pikselposisjoner. Dette er ikke en effektiv metode. I framtiden bør Matlab-kommandoen `conv2()` benyttes. Den utfører en 2-dimensjonal konvolusjon mellom to matriser, *A* og *B*. Matrise *A* er en kantdeteksjon av det aktuelle bildet, og finnes med kommandoen `edge()`. Matrise *B* har høyden og bredden som skiltet beregnes å ville ha ved den aktuelle avstanden. Den bør som tidligere nevnt ha verdien 1 i øvre og nedre kant, verdier større enn 1 i høyre og venstre kant, og 0 midten. Kommandoen `conv2()` returnerer en matrise *C* hvor den største verdien indikerer posisjonen som gir best match mellom matrise *A* og *B*. Dette er forklart i mer detalj på side 15.

Før det ekstraherte nummerskiltet eksporteres til et OCR-program må det filtreres. Flytskjemaet i Figur 4.7 forklarer fremgangsmåten i filtreringen. Boksene forteller i tillegg hvilke funksjoner som er brukt i Matlab-implementeringen.



Figur 4.7. Flytskjema for filtrering av nummerskilt.

4.3 RESULTATER VED GJENNOMKJØRING AV DV

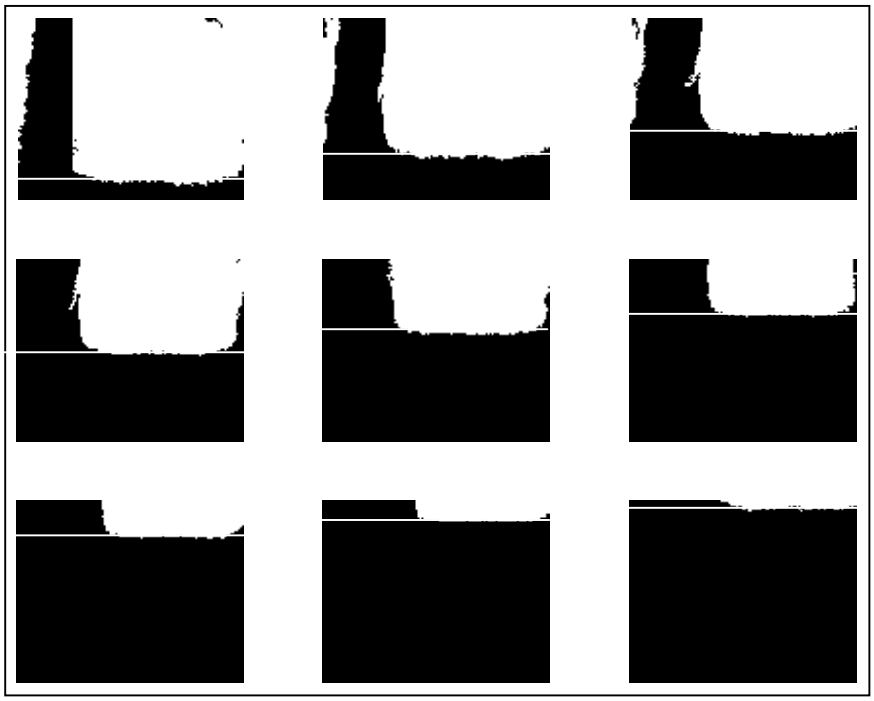
Matlab-programmet som har blitt beskrevet er brukt for å analysere en DV-film bestående av 255 bilder. Filmen viser veibane med passerende biler filmet fra en bro. Kameraet som er brukt er et Sony TRV-900, og progressiv scan er benyttet. Filmen er omgjort til enkeltbilder av Adobe Premiere før den behandles av programmet. Figur 4.8 viser ett bilde av hvert kjøretøy som inngår i filmen.

Figur 4.8. Bilder av de fem kjøretøyene som inngår i filmen.

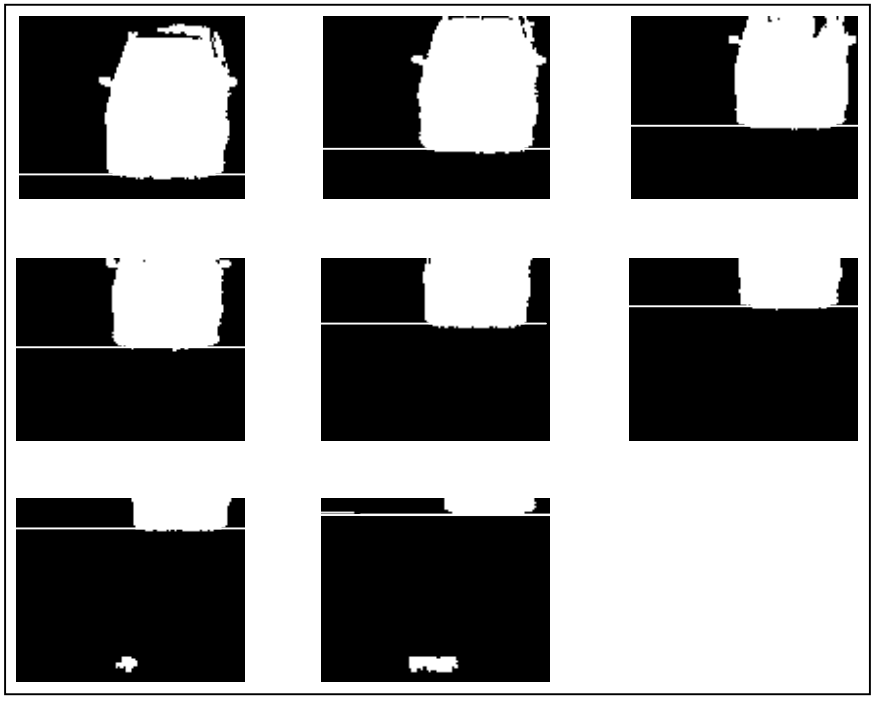
Parameterverdiene brukt i kjøringen er som følger:

- Horisontal posisjon: Midt over midtstripen
- Vertikal posisjon: 7.9 m over veibanen
- Sikteretning i vertikalplanet : 11 graders vinkel ned mot veibanen
- Sikteretning i horisontalplanet: 5 graders mot venstre kjørefelt
- Vinkelåpning: 5 grader
- Egenhastighet 0 km/t
- Bildenes oppløsning: 720×576 piksler

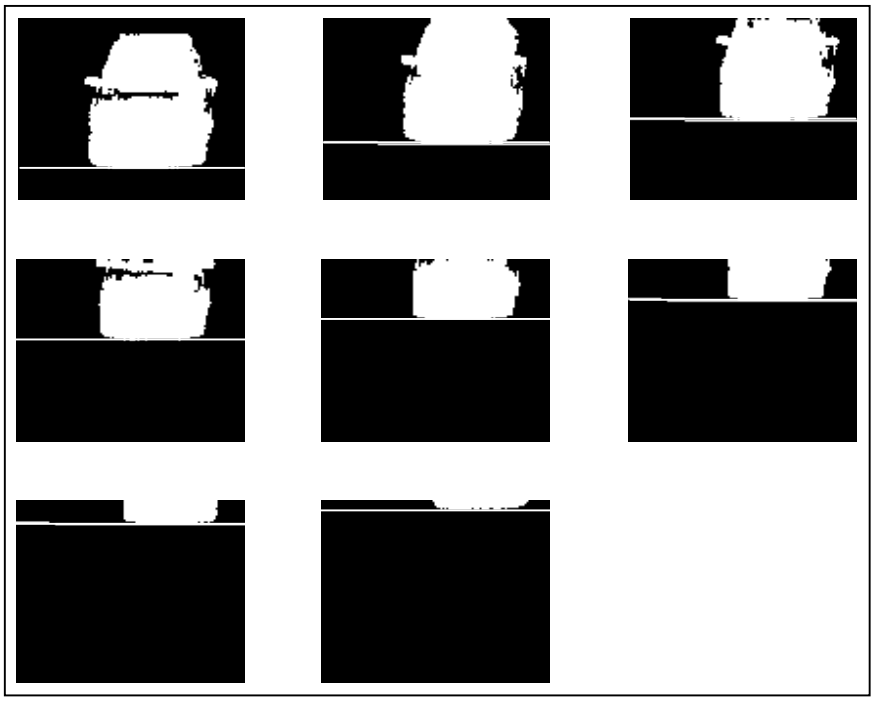
Figur 4.9 til Figur 4.13 viser bildene som er brukt i posisjonsmålingene av kjøretøyene. Bildene er binære og viser passeringen av de isolerte kjøretøyene. Den målte posisjonen er markert med en horisontal hvit linje.



Figur 4.9. Bildene som er brukt for posisjonsmåling av kjøretøy nr 1.

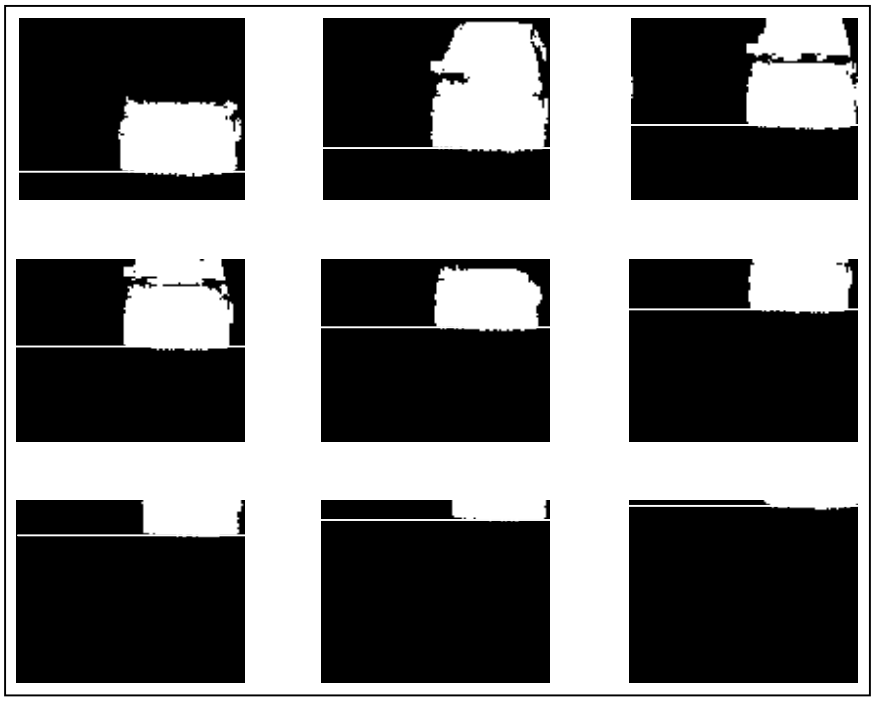


Figur 4.10. Bildene som er brukt for posisjonsmåling av kjøretøy nr 2.



Figur 4.11. Bildene som er brukt for posisjonsmåling av kjøretøy nr 3.

Figur 4.12. Bildene som er brukt for posisjonsmåling av kjøretøy nr 4.



Figur 4.13. Bildene som er brukt for posisjonsmåling av kjøretøy nr 5.

Tabell 4 til Tabell 7 viser resultatene av analysen.

Kjøretøy 1	27	25	23	21	19	17	15	13	11
Kjøretøy 2	83	81	79	77	75	73	71	69	
Kjøretøy 3	135	133	131	129	127	125	123	121	
Kjøretøy 4	159	157	155	153	151	149	147		
Kjøretøy 5	205	203	201	199	197	195	193	191	189

Tabell 4. Nummer på bildene brukt i analysen.

Verdiene i Tabell 4 forteller hvilke bilder som er brukt i posisjonsmålingene for hvert kjøretøy. Fordi bare annethvert bildet i filmen blir brukt er alle bildenummer oddetall.

Kjøretøy 1	538	456	382	312	249	190	130	79	42
Kjøretøy 2	525	442	367	296	231	170	114	62	
Kjøretøy 3	488	409	335	265	201	142	86	41	
Kjøretøy 4	515	433	355	283	216	154	103		
Kjøretøy 5	509	432	363	297	235	179	126	75	38

Tabell 5. Vertikal posisjon i bildet for kjøretøyene, målt i piksler.

Tabell 5 viser de vertikale pikselposisjonene i bildet beregnet av prosess C i skriptet. Verdiene er monotont synkende langs kolonnene, og befinner seg innenfor bildet, slik som forventet.

Kjøretøy 1	34.9483	36.6385	38.3042	40.0194	41.6944	43.3904	45.2572	46.9704	48.2940
Kjøretøy 2	35.2062	36.9429	38.6597	40.4324	42.1981	43.9959	45.7815	47.5697	
Kjøretøy 3	35.9606	37.6799	39.4398	41.2564	43.0642	44.8715	46.7279	48.3308	
Kjøretøy 4	35.4071	37.1411	38.9487	40.7740	42.6269	44.4921	46.1488		
Kjøretøy 5	35.5287	37.1633	38.7556	40.4063	42.0852	43.7214	45.3872	47.1101	48.4415

Tabell 6. Horisontal avstand fra kamera til kjøretøy.

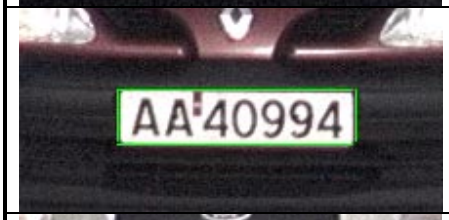
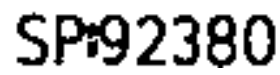
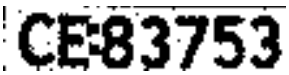
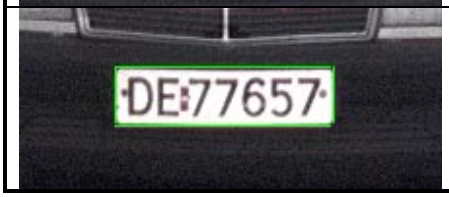
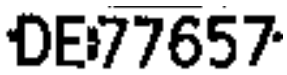
Tabell 6 viser avstandene som er beregnet ut fra pikselposisjonene.

	[m/s]	[km/t]
Kjøretøy 1	21.1756	76.2322
Kjøretøy 2	21.9853	79.1472
Kjøretøy 3	22.2079	79.9485
Kjøretøy 4	22.4827	80.9375
Kjøretøy 5	20.4311	73.5519

Tabell 7. Beregnet hastighet.

De endelig resultatene av hastighetanalysen er gitt i Tabell 7. Dessverre var ikke kjøretøyenes virkelige hastigheter tilgjengelig for sammenligning.

Tabell 8 viser resultatene av nummerskiltanalysen. Kolonnen til venstre viser omrisset av skiltposisjonene i grønt. Kolonnen i midten viser nummerskiltene etter at de har blitt ekstrahert og filtrert. Kolonnen til høyre viser teksten som OCR-programmet Presto returnerer etter behandling av bildene i midtre kolonne.

Posisjon	Ekstrahert og filtrert	OCR resultat
		AYP 812
		AA-40994
		SP192380
		CE83753
		DE177657

Tabell 8. Resultat av nummerskiltanalyse.

Det første skiltet er dekket av et gitter som skaper svært ugunstige forhold. Dette til tross, blir skiltet avlest med bare en bokstav feil. Selv denne feilen kunne vært unngått hvis OCR-programmet var optimalt justert for denne typen tekst. Resten av skiltene er avlest med stort hell. I noen tilfeller skaper oblaten for betalt veiavgift misforståelser, men dette er enkelt å gjøre noe med. For eksempel kan området mellom bokstaver og tall fjernes før OCR-analysen.

5 KONKLUSJON OG VIDERE ARBEID

5.1 VURDERING AV NØYAKTIGHET OG KRAV

5.1.1 Kamera på bro

Simuleringene fra analysen med kamera på bro ser svært lovende ut. Alle disse simuleringene beregner hastighet ut fra kjøretøyets vertikale posisjon i bildet. Resultatene i Figur 3.12 og Figur 3.13 viser at med en vertikal oppløsning på 480 piksler, hvilket er vanlig for dagens DV-kameraer, og et standardavvik for målingen av kjøretøyets vertikale posisjon i bildet på 0.8 piksler, beregnes hastigheten med et standardavvik på 0.37 km/t. Dette gir i 99.9% av tilfellene en feil mindre enn 1.22 km/t, og i 99.99% av tilfellene en feil mindre enn 1.45 km/t, hvilket er akseptabelt. En slik fordeling tilsier at man praktisk talt aldri beregner hastigheten til å være for høy hvis 3 km/t trekkes fra den beregnede verdien.

Resultatene viser i tillegg at nøyaktigheten kan forbedres etter hvert som kameraene i fremtiden får høyere oppløsning, men dette vil bare gi betydelig forbedring for vertikal oppløsning opptil omtrent 1000 piksler. Høyere oppløsning vil ikke gjøre store utslag på nøyaktigheten, fordi kurvene ser ut til å ha et negativt eksponentielt forløp.

Resultatene for simuleringer med fast oppløsning viser at nøyaktigheten øker nærmest proporsjonalt med standardavviket for kjøretøyets målte pikselposisjon i bildet, men selv for et standardavvik på 1.5 piksler og hastigheter mellom 120 og 140 km/t beregnes hastigheten i 99.99% av tilfellene med en feil mindre enn 2.35 km/t. Dette vil si at man kan tillate betydelige feilmålinger i billedanalysen.

Konklusjonen må være at denne metoden er svært robust for et dagens DV-kamera montert på en bro 6-12 meter over veibanen, og øvrige parametervalg som gjør nummerskilt godt leselige.

5.1.2 Kamera i bil

Resultatene fra simuleringer med kamera i bil og bruk av avstand mellom lykter i bildet for å beregne avstand, ser ikke fullt så lovende ut. Intuitivt kan vi forvente dårligere nøyaktighet for denne metoden av flere grunner. I dette tilfellet er det to pikselposisjoner som skal måles i bildet, en for hver lykt. Dessuten er ikke forandringen i avstand mellom et kjøretøys lykter ved en forandring i avstand mellom kamera og kjøretøyet, altfor stor. Dette sammenlignet med forandringen i kjøretøyets vertikale posisjon i bildet under filming fra bro.

Resultatene i Figur 3.14 viser at med et standardavvik for målingen av lyktenes posisjon i bildet på 0.8 piksler, og krav om at hastigheten skal beregnes med et standardavvik mindre enn 1.5 km/t, må kameraets lysfølsomme chip ha en horisontal oppløsning på minimum 1080 piksler. Dette er en høyere oppløsning enn det som er vanlig i dagens kameraer, men det vil utvilsomt være tilgjengelig i fremtiden.

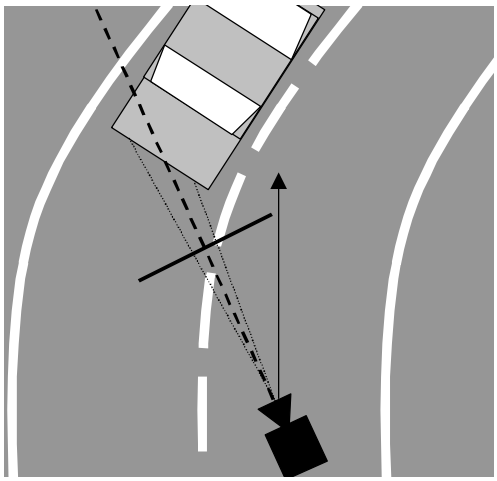
Resultatene viser også at med den oppløsningen som er vanlig i dagens kameraer, må lyktenes posisjon finnes med et standardavvik mindre enn 0.4 piksler. Dette betyr at billedanalysen må beregne den eksakte pikselposisjonen i 79% av tilfellene, hvilket ikke er særlig realistisk.

Konkluderer derfor med at denne metoden avhenger av bedre oppløsning enn det som er tilgjengelig i dagens DV-kameraer for at den skal gi en akseptabel nøyaktighet for beregnet hastighet. Det er likevel viktig å merke seg at denne metoden i motsetning til bruk av kjøretøyets vertikale posisjon i bildet, ikke lar seg påvirke av bakker og dumper i veibanen, og heller ikke av kameraets posisjon. Den er med andre ord mer robust med hensyn til oppsettet, og bør derfor ikke forkastes.

5.2 VIDERE ARBEID

I det videre arbeidet bør det legges vekt på analysen av feilkilder. Denne oppgaven tar for seg metodenes følsomhet for feil i posisjonsmålinger som gjøres i billedanalysen. I tillegg beregnes følsomheten for bildenes oppløsning, men det er flere faktorer som bør tas hensyn til.

I første rekke bør beregningenes følsomhet for ujevnheter og sving i veibanen undersøkes. Metoden for hastighetsberegning som bruker kjøretøyenes vertikale posisjon i bildet, er mest følsom for bakker og dumper. Slike ujevnheter vil medføre endring i kjøretøyets vertikale posisjon i bildet uten at det har sammenheng med forandring i avstanden mellom kamera og kjøretøy. Metoden som bruker avstanden mellom lyktene i bildet er først og fremst følsom for svinger. I en sving vil ikke kjøretøyets front og veibanens lengderetning stå vinkelrett på hverandre. Dette medfører at den projiserte avstanden mellom lyktene blir mindre enn den ellers ville vært. Dette er vist i Figur 5.1.



Figur 5.1. Måling av avstand mellom lykter i sving.

En annen feilkilde som bør behandles er feilmåling eller feilkalibrering av kameraets posisjon og orientering. For kamera på bro vil posisjonen og orienteringen måles med hensyn til veibanen, i tilfellet med kamera i bil vil det være i forhold til bilen. Det er viktig å beregne metodens følsomhet for feilmålinger i disse parameterne.

Det bør også jobbes videre for å kartlegge hvordan kameraets lukkertid og valg av algoritme i billedanalysen, påvirker sannsynlighetsfordelingen for målingen av kjøretøys og lykters posisjon i bildet.

Dersom denne ideen skal videreutvikles til et ferdig produkt kan systemet gjøres mer brukervennlig ved å innføre selvkalibrering. Det vil si at man ikke trenger å måle kameraets posisjon og orientering hver gang det skal gjøres opptak. Kalibreringen kan gjøres av programvare ved å identifisere kjente punkter i bildet. For tilfellet på bro kan de kjente punktene være malt på veibanen ved førstegangs måling. For tilfellet med kamera i bil kan bilens settes foran en vegg med avmerkede punkter.

6 REFERANSER

[1] Gonzalez, R. C., Woods, R. E., *Digital image processing*, USA: Addison Wesley-Wesley Publishing Company, 1993, ISBN 0-201-60078-1

[2] Jain, R., Kasturi, R., Schunk, B. G., *Machine vision*, USA: The McGraw-Hill Companies Inc., 1995, ISBN 0-07-032018-7

[3] Hallingstad, O., *Matematisk modellering av dynamiske system*, Norge: UniK, 1998

[4] Sciavicco, L., Siciliano, B., *Modeling and control of robot manipulators*, USA: The McGraw-Hill Companies Inc., 1996, ISBN 0-07-057217-8

[5] Kreyzig, E., *Advanced engineering mathematics*, Canada: John Wiley & Sons Inc., 1993, ISBN 0-471-59989-1

[6] Rottman, K., *Matematisk formelsamling*, Danmark: Bibliographisches Institut & F.A. Brockhaus, 1991, ISBN 82-7822-005-0

[7] Mathworks, *Image processing toolbox user's guide*, The Mathworks Inc., 1993-1998, [Online] Tilgjengelig fra: http://www.mathworks.com/access/helpdesk/help/pdf_doc/images/images_tb.pdf

[8] Mathworks, *Matlab online manuals*, The Mathworks Inc., [Online] Tilgjengelig fra: http://www.mathworks.com/access/helpdesk/help/fulldocset.shtml

[9] Adamwilt.com, *The DV, DVCAM, & DVCPRO formats - Technical details*, [Online] Tilgjengelig fra: http://www.adamwilt.com/DV-tech.html

[10] Simpson, D., *DV coding – How it works with IEEE-1394*, [Online] Tilgjengelig fra: http://desktopvideo.miningco.com/compute/desktopvideo/library/weekly/aa032698.htm

[11] Microsoft, *DV data in the AVI file format*, [Online] Tilgjengelig fra: http://asia.microsoft.com/directx/dxm/help/ds/filtdev/DV_Data_AVI_File_Format.htm

APPENDIKS A – BESKRIVELSE AV DV

6.1 HVA ER DV?

DV, først kjent som DVC (Digital Video Cassette), er en internasjonal standard for digital video, skapt i et samarbeid mellom ti store elektronikkonsern. I dag har samarbeidet utviklet seg til å innbefatte 60 organisasjoner. DV bruker 6,35mm metallbånd for lagring av høykvalitets digital video. Video samples med samme rate som Digital Betacom video, det vil si 720 piksler per scanlinje, men med lavere fargesampling (fargesampling blir diskutert senere).

Den samplede videoen komprimeres ved bruk av DCT (diskret kosinustransform), det vil si samme metoden som for JPEG-bilder. Fordelen med metoden brukt i DV er at den tillater mer lokal optimalisering av kvantiseringstabeller i bildet. Dette gir bedre kvalitet for den nominelle 5:1 kompresjonen, enn det et JPEG-bilde ville gi.

DV bruker såkalt *intraframe-kompresjon*. Det vil si at komprimeringen av hvert bilde, eller frame, i videoen, avhenger bare av bildet selv, og ikke av data fra tidligere eller etterfølgende bilder. Ved bruk av interlaced opptak (forklart i billedanalysen), brukes *interfield-kompresjon*. Dette innbærer at dersom lite forskjell detekteres mellom interlace-feltene i et frame, så blir disse feltene komprimert sammen. I de tilfellene hvor feltene komprimeres sammen brukes den ekstra lagringsplassen for å øke kvaliteten. I teorien betyr dette at stillestående områder i et bilde får mer detalj enn områder med mye bevegelse. I praksis innebærer det degenerering av bildet rundt objekter som er i bevegelse.

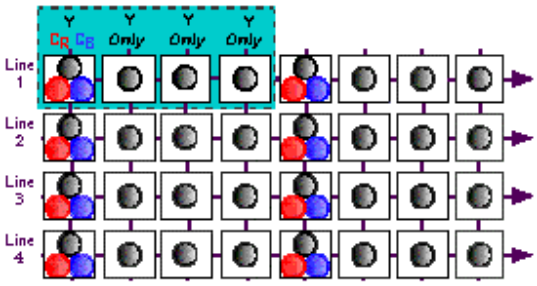
DV-video overføres nominelt i en datastrøm på 25 megabit per sekund (Mbps). Med audio, tidskoding, *Insert and Track Information* (ITI) og feilkorrigering, kommer den totale datastrømmen på 36 Mbps.

6.2 FARGESAMPLING

Fargesampling for DV foregår med *4:2:0* koding for PAL standard, og med *4:1:1* koding for NTSC-standard. *4:2:2* og *4:1:1* er forkortelser for to forskjellige samplingsstrukturer for digital video med 13,5 MHz samplingsfrekvens og 720 piksler per linje. Det første tallet referer til samplingsraten for intensitet (illumninans), som i begge disse tilfellene er fire ganger større enn bærefrekvensen for farger i NTSC og PAL.

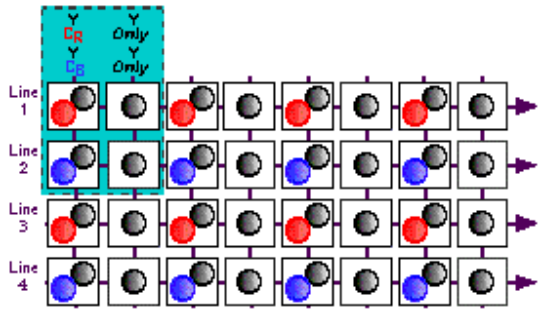
De to andre tallene refererer til samplingsraten for fargedifferansesignalene rød minus gul (R-Y) og blå minus gul (B-Y). Disse fargedifferansene er kalt Cr og Cb i det digitale domenet.

I *4:1:1* systemer foregår fargesamplingen med en fjerdedel av frekvensen til intensitetssamplingen, og begge fargesamplingene lagres på samme sted. Dette er vist i Figur 0.1.



Figur 0.1. *4:1:1* sampling, brukt i 525 linjers / 59.94 Hz, NTSC-DV.

4:2:0 notasjonen er enda mindre intuitiv, den betyr at fargesamplingen foregår med halve frekvensen av intensitetssamplingen, både i horisontal og vertikal retning. Det vil si at fargeinformasjon samples 360 ganger per linje, men bare på annenhver linje. Teorien er at samplingsmetoden skal gi en mer balansert struktur en *4:1:1*, der den vertikale samplingen ser ut til å være fire ganger høyere enn den horisontale. Til tross for at *4:2:0* fungerer bra for PAL-koding og kringkasting, er den mer utsatt for degenerering ved multigenerasjonsarbeid enn *4:1:1*. Strukturen er illustrert i Figur 0.2.



Figur 0.2. *4:2:0* sampling, brukt i 625 linjers / 50 Hz, PAL-DV.

For mer informasjon angående DV-formatets struktur refereres det til *The DV, DVCAM, & DVCPRO Formats - Technical Details* **[9]**, og *DV Coding – How it works with IEEE-1394* **[10]**.

6.3 DV-DATA I AVI-FORMAT

Når DV blir overført til PC ved bruk av Fire wire og et videoredigeringsprogram, blir det generert AVI-filer. AVI (Audio Video Interleave) er et fleksibelt format for lagring av lyd og bilde, og det er i utstrakt bruk på PC'er med Windows-platform.

Microsoft har utviklet to lagringsformat for DV-video i AVI-filer. Det ene formatet inneholder kun én *DV-stream*, og er det formatet som bruker minst plass. Ulempen med dette formatet er at det ikke er bakoverkompatibelt med programmer som *Video for Windows*. Det andre formatet inneholder DV som en *vids-stream* og en *auds-stream*. Det betyr at bilde og lyd er lagret hver for seg.

For informasjon om hvordan DV i AVI-format skal dekodes refereres det til Microsoft sine internettsider om *DV in the AVI file format* **[10]**.

APPENDIKS B – KILDEKODE

6.4 PROGRAM FOR ANALYSE AV DV

6.4.1 Hovedskript – seqAnalyser.m

```
%Sequence Analyzer
%Ramon Kristian Arellano
%Last modified: 29.11.99

%Clearing and formatting
%clear all;
format compact;

%***************** DELER AV PROGRAMMET SOM SKAL KJØRES ********************
getBackgroundImage        = 0;    %PROSESS A
makeDifferenceSequence    = 0;    %PROSESS B
findVehicles              = 1;    %PROSESS C
getVehicleDistances       = 1;    %PROSESS D
getVehicleVelocities      = 1;    %PROSESS E
getVehiclePlates          = 1;    %PROSESS F

%************************* GLOBALE VARIABLE **************************
global roadYPos;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
global frameRate;
global orgSeqName  ;
global diffSeqName;
global imgFmt;


%************************ PARAMETERE FOR OPPSETT *************************
%(All positions are relative to camera position [0, 0, 0]);
roadYPos              = 7.9;                    %Vertical position of road
cameraElevation       = 11*(2*pi/360);          %Camera is pointing 10 degrees downwards
cameraAsimuth         = 0;                      %Camera is pointing 10 degrees to the
left
cameraTransf          = [cameraElevation, cameraAsimuth, 0, 0, 0, 0]; % rotation around
xyz
focalLength       = 0.1;
lensOpening       = 5*(2*pi/360);
resolutionX       = 720;

resolutionY       = 576;
imageWidth            = 2*focalLength*tan(lensOpening/2);
imageHeight       = imageWidth*resolutionY/resolutionX;
frameRate             = 25;                     %Images per second

%************************ PARAMETERE FOR PROGRAM *************************
orgSeqName     = 'C:\windows\profiles\ramon\My
documents\diplom\bilder\sarpsborg_klipp1\original\.tif';
diffSeqName    = 'C:\windows\profiles\ramon\My
documents\diplom\bilder\sarpsborg_klipp1\difference\seq';
```

```
imgFmt         = 'tif';
firstImg       = 1;       %This is the first image of the sequence to be used (including
images only showing empty background)
lastImg        = 225;     %This is the last image of the sequence to be used
numOfBgImgs= 8;       %This is the number of images at the (starting with 'firstimage')
that only contain empty background
maxVehPos      = 24;      %Max number of positions per vehicle

%****************************** PROSESS A ******************************
%Get and show an average background image
if getBackgroundImage == 1
   disp('Getting background image');
   bgImg = getBgImg(firstImg, numOfBgImgs);
end;

%****************************** PROSESS B ******************************
%Write a new sequence to disk containing difference images
if makeDifferenceSequence == 1
   disp('Making Difference Sequence');
   makeDiffSeq(bgImg, firstImg, lastImg);
end;

%****************************** PROSESS C ******************************
%Finds vehicles in sequence and creates one table containing positions and one talbe
containing
%the corresponding image numbers
if findVehicles
   vehNum      = 0;           %Vehicle number
   imgNum      = firstImg;%Image number
   vehPresent  = 0;           %True if vehicle is present at bottom of image
   endOfSeq= 0;           %True if image number is greater than last image number;
   vehPos = zeros(1, maxVehPos);

   while endOfSeq ~= 1,
      %Goes through images until it finds one with a vehicle at reaching the bottom
      disp(strcat('Waiting for vehicle #', int2str(vehNum+1), ' to come into screen'));
      while checkForVeh(imgNum) == 0,
         if imgNum >= lastImg-2
            endOfSeq = 1;
            break;
         end;
         imgNum = imgNum + 2
      end;

      %Calls the function getVehPos which returns the vectors with positions and image
numbers
      disp(strcat('Vehicle #', int2str(vehNum+1), ' found. Analysing positions'));
      if endOfSeq ~= 1
         vehNum = vehNum + 1;
         [vehPosVec, vehImgVec] = getVehPos(vehNum, imgNum-2, maxVehPos);
         vehPos(vehNum, :) = vehPosVec;
         vehImg(vehNum, :) = vehImgVec;
      end;

      %Goes through images until the vehicle in the images has gone out at the bottom
      disp(strcat('Waiting for vehicle #', int2str(vehNum+1), ' to exit screen'));
      while checkForVeh(imgNum) == 1,
         if imgNum >= lastImg
            endOfSeq = 1;
            break;
         end;
         imgNum = imgNum + 2
      end;
   end;
end;

%****************************** PROSESS D ******************************
%Calculate real distances to vehicel from positions found in images
if getVehicleDistances == 1
   vehDist = getVehDist(vehPos);
end;
```

```
%****************************** PROSESS E ******************************
%Calculates each vehicles velocity solving a least squares problem : min sum((s(i)-
v*t(i))^2)
if getVehicleVelocities == 1
   vehVel = getVehVel(vehDist, vehImg, frameRate);
end;

%****************************** PROSESS F ******************************
%Extract the license plate from the first image in each series
if getVehiclePlates == 1
   getVehPlates(vehImg, vehDist);
```

6.4.2 end;Funksjon – makeDiffSeq.m

```
function finished = makeDiffSeq(bgImg, firstImg, lastImg);

%Global variables
global orgSeqName  ;
global diffSeqName;
global imgFmt;
global resolutionY;
global resolutionX;

for imgNum = firstImg:2:lastImg
   imgNum
   tempImg = double(imread(getImgName(orgSeqName, imgNum, imgFmt), imgFmt));
   diffImg = uint8(getDiffImg(tempImg, bgImg));
   imwrite(diffImg, getImgName(diffSeqName, imgNum, imgFmt), imgFmt);
end;
finished = 1;
```

6.4.3 Funksjon – getDiffImg.m

```
function diffImg = getDiffImg(orgImg, bgImg);
%Get Difference Image

global resolutionY;
global resolutionX;
colTol = 0.09;
intTol = 0.2;

%Create matric containing values taht represent the intensity difference between orgImg
and teh background image
intDiffNonNormImg = rgb2gray(abs(orgImg - bgImg));

%Normalize the difference image
intDiffImg = intDiffNonNormImg / max(max(max(intDiffNonNormImg )));
%writeThumb(intDiffImg, 1);

%Create binary image
binIntImg = im2bw(intDiffImg, intTol);
%writeThumb(binIntImg, 3);

%Create a matrix containing values that represent the relative color difference between
orgImg and the background image
orgRelt12 = reshape(orgImg(:,:,1)./orgImg(:,:,2), resolutionY, resolutionX,1);
orgRelt13 = reshape(orgImg(:,:,1)./orgImg(:,:,3), resolutionY, resolutionX,1);
orgRelt23 = reshape(orgImg(:,:,2)./orgImg(:,:,3), resolutionY, resolutionX,1);
bgRelt12 = reshape(bgImg(:,:,1)./bgImg(:,:,2), resolutionY, resolutionX,1);
bgRelt13 = reshape(bgImg(:,:,1)./bgImg(:,:,3), resolutionY, resolutionX,1);
bgRelt23 = reshape(bgImg(:,:,2)./bgImg(:,:,3), resolutionY, resolutionX,1);
```

```
colDiffImg = sqrt((orgRelt12-bgRelt12).^2 + (orgRelt13-bgRelt13).^2 + (orgRelt23-
bgRelt23).^2);
%writeThumb(colDiffImg, 2);

%Create binary image
binColImg = im2bw(colDiffImg, colTol);
%writeThumb(binColImg, 4);

%Combine images
binImg = binColImg | binIntImg;
%writeThumb(binImg, 5);

%Clean away lonely pixels
cleanImg = bwmorph(binImg, 'clean');
%writeThumb(cleanImg, 6);

%Fill
fillImg = bwfill(cleanImg, 'holes');
%writeThumb(fillImg, 7);

%Erode
erodeImg = bwmorph(fillImg, 'erode');
%writeThumb(erodeImg, 8);

%Dilate with 5x5
mask = ones(5,5);
dilImg = dilate(erodeImg, mask);
%writeThumb(dilImg, 9);

%Fill
fillImg = bwfill(dilImg, 'holes');
%writeThumb(fillImg, 10);

%Get the two biggest objects remaining
diffImg = getBig(fillImg);
%writeThumb(diffImg, 11);
```

6.4.4 Funksjon – checkForVeh.m

```
function vehPresent = checkForVeh(imgNum)
%Checks if a vehicle is present at the bottom of image number imgNum in the sequence.
Returns 1 if true.

global diffSeqName;
global imgFmt;
global resolutionY;
global resolutionX;
global bgColor;

%Parameters
linesToCheck       = 13;          %Number of lines from bottom of image to be checked
pixelTol           = 200;         %Number of pixels that must be different from background
to return true

img = imread(getImgName(diffSeqName, imgNum, imgFmt), imgFmt);
pixelCounter = sum(sum(img(resolutionY-linesToCheck:resolutionY-3, :)));

if pixelCounter > pixelTol
   vehPresent = 1;
else
   vehPresent = 0;
end;
```

6.4.5 Funksjon – getVehPos.m

```
function [vehPosVec, vehImgVec] = getVehPos(vehNum, imgNum, maxVehPos)
%Finds the positions for series of images with decreasing image number until no vehicle
%can be seen in the image

global diffSeqName;
global imgFmt;
global resolutionY;
global resolutionX;

linesToCheck       = 3;
pixelTol           = 200;
bgColor        = [255, 0, 0];
posNum             = 1;
finished           = 0;
imgsBtwnChecks = 2;
iCounter           = resolutionY-linesToCheck;

vehPosVec = zeros(1, maxVehPos);
vehImgVec = zeros(1, maxVehPos);

%Temp
figure(vehNum);
img = imread(getImgName(diffSeqName, imgNum, imgFmt), imgFmt);
imshow(uint8(img));

while finished ~= 1,
   img = imread(getImgName(diffSeqName, imgNum, imgFmt), imgFmt);

   while sum(sum(img(iCounter-linesToCheck:iCounter, :))) < pixelTol
      iCounter = iCounter - 1;
      if iCounter == linesToCheck
          finished = 1;
          break;
      end;
   end;

   if finished ~= 1
      vehPosVec(posNum) = iCounter-linesToCheck;
      vehImgVec(posNum) = imgNum;

      %Temporary
      figure(vehNum+10);
      subplot(4,3,posNum);
      img(iCounter, :, 1) = 255;

      imshow(uint8(img));

      posNum = posNum + 1;
      imgNum = imgNum - imgsBtwnChecks
    end;
end;
```

6.4.6 Funksjon – getVehDist.m

```
function vehDist = getVehDist(vehPos)
%Calculates the distance from along road from camera to vehicle
%Uses the vertical position of the vehicle found from the images
%and the measured constants such as camera position and orientation

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
```

```
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
global frameRate;

numOfVehs = length(vehPos(:, 1));
numOfPos  = length(vehPos(1, :));

vehDist = zeros(numOfVehs, numOfPos);

for vehNum = 1:numOfVehs,
   for posNum = 1:numOfPos,
      if vehPos(vehNum, posNum) ~= 0
         %Calculate distance
         dy = vehPos(vehNum, posNum)*imageHeight*1.0005/resolutionY-
(imageHeight*1.0005/2);
         gamma = cameraElevation + atan2(dy, focalLength);
         vehDist(vehNum, posNum) = roadYPos/tan(gamma);
      end;
   end;
end;
```

6.4.7 Funksjon – getVehVel.m

```
function vehVel = getVehVel(vehDist, vehImg, frameRate)
%Calculates each vehicles velocity solving a least squares problem : min sum((s(i)-
v*t(i))^2)

global distVec;
global timeVec;

numOfVehs = length(vehDist(:, 1));
%Create a new vector with one element for each vehicle
vehVel = zeros(numOfVehs, 1);

for vehNum = 1:numOfVehs,
   numOfDist = max(find(vehDist(vehNum, :)));
   %Create a vector contaning the distances the car has moved relative to the first image
   distVec = (vehDist(vehNum, 1:numOfDist)-vehDist(vehNum, 1))';
   %Create a vector containing the time passed relative to the first image
   timeVec = -(vehImg(vehNum, 1:numOfDist)-vehImg(vehNum, 1))'/frameRate;
   %Solve for velocity using a linear least squares method
   vehVel(vehNum) = lsqlin(timeVec, distVec, 0, 0);
end;
```

6.4.8 Funksjon – getVehPlates.m

```
function getVehPlates(vehImg, vehDist)
%Extracts the vehicles license plates from the first image in the series
%Uses knowledge of the camera geometry to calculate the size of the plates
```

```
global imgFmt;
global orgSeqName  ;

%Measured constants
%(All positions are relative to camera position [0, 0, 0]);
roadYPos              = 7.9;                    %Vertical position of road
cameraElevation       = 11*(2*pi/360);          %Camera is pointing 10 degrees downwards
focalLength       = 0.1;
lensOpening       = 5*(2*pi/3);
resolutionX       = 720;

resolutionY       = 576;
imageWidth            = 2*focalLength*tan(lensOpening/2);
imageHeight       = imageWidth*resolutionY/resolutionX;

plateImgName = 'C:\windows\profiles\ramon\My
documents\diplom\bilder\sarpsborg_klipp1\plates\veh';
numOfVeh = length(vehImg(:,1));
areaHeight = 20;
areaWidth  = 30;

plateVariation = 2;

for vehNum = 2:2, %numOfVeh,
   vehNum

   switch vehNum
   case 1
      plateWidth = 106;
      plateHeight   = 24;
   case 2
      plateWidth= 120;
      plateHeight   = 26;
   case 3
      plateWidth= 110;
      plateHeight   = 26;
   case 4
      plateWidth= 112;
      plateHeight   = 28;
   case 5
      plateWidth= 110;
      plateHeight   = 28;
   end;

   %Read first image in sequence
   img = imread(getImgName(orgSeqName, vehImg(vehNum, 1), imgFmt), imgFmt);
   imgGray = rgb2gray(img);
   edgeImg = edge(imgGray, 'sobel');

   maxMatches = 0;
   for iCounter = round(resolutionY/2):resolutionY-plateHeight-plateVariation,
   iCounter
      for jCounter = 1:resolutionX-plateWidth-plateVariation,
         %intensity = sum(sum(sum(img(iCounter:iCounter+areaHeight,
jCounter:jCounter+areaWidth ,:))));
         matches = sum(sum(edgeImg(iCounter:iCounter+plateVariation,
jCounter:jCounter+plateWidth)));
         matches = matches +
sum(sum(edgeImg(iCounter+plateHeight:iCounter+plateHeight+plateVariation,
jCounter:jCounter+plateWidth)));
         matches = matches + 4*sum(sum(edgeImg(iCounter:iCounter+plateHeight,
jCounter:jCounter+plateVariation)));
         matches = matches + 4*sum(sum(edgeImg(iCounter:iCounter+plateHeight,
jCounter+plateWidth:jCounter+plateWidth+plateVariation)));

         if matches > maxMatches
            maxMatches = matches;
            iMax = iCounter;
            jMax = jCounter;
         end;
```

```
      end;
   end;

   img(iMax, jMax:jMax+plateWidth, 1) = 0;
   img(iMax, jMax:jMax+plateWidth, 2) = 255;
   img(iMax, jMax:jMax+plateWidth, 3) = 0;

   img(iMax+plateHeight+plateVariation, jMax:jMax+plateWidth, 1) = 0;
   img(iMax+plateHeight+plateVariation, jMax:jMax+plateWidth, 2) = 255;
   img(iMax+plateHeight+plateVariation, jMax:jMax+plateWidth, 3) = 0;

   img(iMax:iMax+plateHeight, jMax, 1) = 0;
   img(iMax:iMax+plateHeight, jMax, 2) = 255;
   img(iMax:iMax+plateHeight, jMax, 3) = 0;

   img(iMax:iMax+plateHeight, jMax+plateWidth+plateVariation, 1) = 0;
   img(iMax:iMax+plateHeight, jMax+plateWidth+plateVariation, 2) = 255;
   img(iMax:iMax+plateHeight, jMax+plateWidth+plateVariation, 3) = 0;

   plateImg = img(iMax+plateVariation:iMax+plateHeight-plateVariation,
jMax+plateVariation:jMax+plateWidth-plateVariation, :);
   binPlateImg = getBinPlateImg(plateImg);
   imWrite(binPlateImg, getImgName(plateImgName, vehNum, 'tif'), 'tif');

   figure(20+vehNum);
   imshow(uint8(img));
end;
```

6.4.9 Funksjon – getBinPlateImg.m

```
function binPlateImg = getBinPlateImg(plateImg)

%Constants
textPercentage = 0.35;

%Convert color image to grayscale
grayImg = rgb2gray(plateImg);

%Create a histogram with one 256 bins
histogram=sum(hist(double(grayImg),256),2);

%Calculate total number of elements
numOfElements = length(grayImg(1,:))*length(grayImg(:,1));

%Calculate how many bins must be summed before the number of elements
%(in percentage of the total number of elements), exceeds the average text percentage.
threshold = 0;
while sum(histogram(1:1+threshold))/numOfElements < textPercentage,
   threshold = threshold + 1;
end;

binPlateImg = im2bw(grayImg, threshold/255);
```

6.5 PROGRAM FOR NØYAKTIGHETSANALYSE

6.5.1 Hovedskript – errorSim.m (kamera på bro)

```
%Ramon Kristian Arellano
%22.11.1999
```

```
%Nøyaktighetsanalyse
%Kamera står på bro

clear all;
format compact;

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;

%Variables
%(All positions are relative to camera position [0, 0, 0]);
roadWidth             = 8.5;                    %Total width of both lanes
roadXPos              = roadWidth/4;        %Horizontal position of middle line
roadYPos              = 7.9;                    %Vertical position of road
vehicleXPos        = roadXPos-roadWidth/4;   %Horizontal position of car in left lane
vehicleYPos        = roadYPos;           %Vertical position of car wheels.
vehicleZPos        = 100;                    %Distance between camera in vehicle along
road
vehicleLightYPos       = roadYPos-0.5;           %Vertical position of car lights
vehicleWidth           = 1.5;                    %Distance between lights.
cameraElevation        = 11*(2*pi/360);          %Camera is pointing 10 degrees downwards
cameraAsimuth          = 0;                      %Camera is pointing 10 degrees to the
left
cameraTransf           = [cameraElevation, cameraAsimuth, 0, 0, 0, 0]; % rotation around
xyz, and translation along xyz
focalLength        = 0.1;
lensOpening        = 5*(2*pi/360);
resolutionX        = 640;

resolutionY        = 480;
imageWidth             = 2*focalLength*tan(lensOpening/2);
imageHeight        = imageWidth*resolutionY/resolutionX;
speed                  = 100/3.6;
acceleration           = 0;
frameRate              = 1/0.08;

%Parts of script to run
showImageAsPlot        = 1;
doCalculations     = 1;

%Create Projection Image
if(showImageAsPlot == 1)

   %Road lines
   nrOfLinePoints = 100;
   lineSegLength = 10;
   lineBrkLength = 3;
   for counter1=1:3,
      for counter2=1:nrOfLinePoints,
         %line format : line(line nr, line point nr, point position (4*1 vector))
         line(counter1, counter2, :) = [(roadWidth/2)*(counter1-2)+roadXPos, roadYPos,
counter2*lineSegLength-8, 1];
      end;
   end;
```

```
   %Vehicle Positions
   zPos = vehicleZPos;
   frameNr = 1;
   while zPos > 0,
      zPositions(frameNr) = zPos;
      time = frameNr / frameRate;
      zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
      frameNr = frameNr +  1;
   end;

   %Find first frame
   y = -1000;
   frameNr = 1;
   while y < -imageHeight/2,
      pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]', cameraTransf);
      [x, y] = xyProjection(pos, focalLength);
      frameNr = frameNr + 1;
   end;
   firstFrame = frameNr-1;

   %Find last frame
   y = -1000;
   frameNr = firstFrame;
   while y < imageHeight/2,
      pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]', cameraTransf);
      [x, y] = xyProjection(pos, focalLength);
      frameNr = frameNr + 1;
   end;
   lastFrame = frameNr-2;

   img = zeros(resolutionY, resolutionX);
   figure(3);
   hold off;

   %Draws road lines
   for counter1=1:3,
      for counter2=1:nrOfLinePoints-1,
            if counter1 == 2
               brkLength = lineBrkLength;
            else
               brkLength = 0;
            end;

            pos1 = transform(reshape(line(counter1, counter2, :), 4, 1), cameraTransf);
            pos2 = transform(reshape(line(counter1, counter2+1, :), 4, 1)-
[0,0,brkLength,1]', cameraTransf);
            [x1, y1] = xyProjection(pos1, focalLength);
            [x2, y2] = xyProjection(pos2, focalLength);
            j = ([x1, x2]+imageWidth/2)/imageWidth*resolutionX;
            i = ([y1, y2]+imageHeight/2)/imageHeight*resolutionY;

            plot(j, i, 'b');
            hold on;
      end;
   end;

   %Draws vehicle positions
   for frameNr=1:length(zPositions),
      pos1 = transform([vehicleXPos-vehicleWidth, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
      pos2 = transform([vehicleXPos+vehicleWidth, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
      pos3 = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
      pos4 = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);

      [x1, y1] = xyProjection(pos1, focalLength);
      [x2, y2] = xyProjection(pos2, focalLength);
      [x3, y3] = xyProjection(pos3, focalLength);
```

```
      [x4, y4] = xyProjection(pos4, focalLength);

      j = ([x1, x2]+imageWidth/2)/imageWidth*resolutionX;
      i = ([y1, y2]+imageHeight/2)/imageHeight*resolutionY;
      lightj = ([x3, x4]+imageWidth/2)/imageWidth*resolutionX;
      lighti = ([y3, y4]+imageHeight/2)/imageHeight*resolutionY;

      if(frameNr == firstFrame | frameNr == lastFrame)
         color = 'm';
      else
         color = 'r';
      end;
      plot(j, i, color);

      plot(lightj(1), lighti(1), 'g*');
      plot(lightj(2), lighti(2), 'g*');

      if(frameNr == firstFrame | frameNr == lastFrame)
         color = 'b';
      else
         color = 'g';
      end;
      plot(lightj, lighti, color);
   end;

   axis ij;
   axis([0, resolutionX, 0, resolutionY]);
end;


%Calculate distance and velocity from image information
%Parameters
if(doCalculations == 1)
   nrOfSets       = 4;
   nrOfTests          = 1000;
   resultTable        = zeros(nrOfTests, nrOfSets);

   for testSpeedSet=1:nrOfSets,
      switch testSpeedSet
         case 1
            minSpeed = 60/3.6;
            maxSpeed = 80/3.6;
            disp('Speed 60 - 80 km/h');
         case 2
            minSpeed = 80/3.6;
            maxSpeed = 100/3.6;
            disp('Speed 80 - 100 km/h');
          case 3
            minSpeed = 100/3.6;
            maxSpeed = 120/3.6;
            disp('Speed 100 - 120 km/h');
         otherwise
            minSpeed = 120/3.6;
            maxSpeed = 140/3.6;
            disp('Speed 100 - 140 km/h');
      end;

      maxDif   = 0;
      testNr   = 0;

      for speed=minSpeed:(maxSpeed - minSpeed)/nrOfTests: maxSpeed,
         testNr    = testNr + 1;
         zPos      = vehicleZPos;
         frameNr   = 1;

         %Vehicle Positions
         clear zPositions;
         while zPos > 0,
            zPositions(frameNr) = zPos;
            time = frameNr / frameRate;
            zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
```

```
            frameNr = frameNr +  1;
         end;

         %Find first frame
         y = -1000;
         frameNr = 1;
         while y < -imageHeight/2,
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         firstFrame = frameNr-1;

         %Find last frame
         y = -1000;
         frameNr = firstFrame;
         while y < imageHeight/2,
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         lastFrame = frameNr-2;

         vehPos = zeros(1, lastFrame-firstFrame);
         vehImg = zeros(1, lastFrame-firstFrame);

         for frameCounter=firstFrame:lastFrame,
            %Calculate pixel y-position
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameCounter), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);
            vehPos(1, frameCounter-firstFrame+1) =
round((y+imageHeight/2)/imageHeight*resolutionY);

            vehImg(1, frameCounter-firstFrame+1) = frameCounter;
         end;

         %Calculate distances
         vehDist = getVehDist(vehPos+0.00001);

         %Calculate velocities
         vehVel = getVehVel(vehDist, vehImg, frameRate);

         %Insert results in table
         nrOfFrames(testNr, testSpeedSet) = lastFrame-firstFrame;
         resultTable(testNr, testSpeedSet) = abs(vehVel - speed);
      end;
   end;

   disp('Average error [km/h]')
   mean(resultTable)*3.6

   disp('Standard deviation [km/h]')
   std(resultTable)*3.6

   disp('Max error [km/h]');
   max(resultTable)*3.6

   disp('Average number of frames used');
   mean(nrOfFrames)
end;
```

6.5.2 Hovedskript – errorSim.m (kamera i bil)

```
%Ramon Kristian Arellano
```

```
%10.09.1999
%Kamera i bil

clear all;
format compact;

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;


%Variables
%(All positions are relative to camera position [0, 0, 0]);
roadWidth             = 8.5;                    %Total width of both lanes
roadXPos              = -roadWidth/4;           %Horizontal position of middle line
roadYPos              = 1.5;                    %Vertical position of road
vehicleXPos        = roadXPos-roadWidth/4;   %Horizontal position of car in left lane
vehicleYPos        = roadYPos;           %Vertical position of car wheels.
vehicleZPos        = 100;                    %Distance between camera in vehicle along
road
vehicleLightYPos      = roadYPos-0.5;           %Vertical position of car lights
vehicleWidth          = 1.6;                    %Distance between lights.
cameraElevation       = 2*(2*pi/360);           %Camera angle downwards
cameraAsimuth         = 10*(2*pi/360);          %Camera angle the left
cameraTransf          = [cameraElevation, cameraAsimuth, 0, 0, 0, 0]; % rotation around
xyz, and translation along xyz
focalLength        = 0.1;
lensOpening        = 12*(2*pi/360);
resolutionX        = 640;

resolutionY        = 480;
imageWidth            = 2*focalLength*tan(lensOpening/2);
imageHeight        = imageWidth*resolutionY/resolutionX;
speed                 = 180/3.6;
acceleration          = 0;
frameRate             = 1/0.08;

%Parts of script to run
showImageAsPlot       = 1;
doCalculations     = 1;


%Create Projection Image
if(showImageAsPlot == 1)

   %Road lines
   nrOfLinePoints = 100;
   lineSegLength = 10;
   lineBrkLength = 3;
   for counter1=1:3,
      for counter2=1:nrOfLinePoints,
         %line format : line(line nr, line point nr, point position (4*1 vector))
         line(counter1, counter2, :) = [(roadWidth/2)*(counter1-2)+roadXPos, roadYPos,
counter2*lineSegLength-8, 1];
      end;
```

```
   end;

   %Vehicle Positions
   zPos = vehicleZPos;
   frameNr = 1;
   while zPos > 0,
      zPositions(frameNr) = zPos;
      time = frameNr / frameRate;
      zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
      frameNr = frameNr +  1;
   end;

   %Find first frame
   deltaH = 1000;
   frameNr = 1;
   while deltaH > imageWidth/2,
      pos = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos, zPositions(frameNr),
1]', cameraTransf);
      [deltaH, y] = xyProjection(pos, focalLength);
      frameNr = frameNr + 1;
   end;
   firstFrame = frameNr-1;

   %Find last frame
   deltaV = 1000;
   frameNr = firstFrame;
   while deltaV > -imageWidth/2,
      pos = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos, zPositions(frameNr),
1]', cameraTransf);
      [deltaV, y] = xyProjection(pos, focalLength);
      frameNr = frameNr + 1;
   end;
   lastFrame = frameNr-2;

   img = zeros(resolutionY, resolutionX);
   figure(3);
   hold off;

   %Draws road lines
   for counter1=1:3,
      for counter2=1:nrOfLinePoints-1,
            if counter1 == 2
               brkLength = lineBrkLength;
            else
               brkLength = 0;
            end;

            pos1 = transform(reshape(line(counter1, counter2, :), 4, 1), cameraTransf);
            pos2 = transform(reshape(line(counter1, counter2+1, :), 4, 1)-
[0,0,brkLength,1]', cameraTransf);
            [x1, y1] = xyProjection(pos1, focalLength);
            [x2, y2] = xyProjection(pos2, focalLength);
            j = ([x1, x2]+imageWidth/2)/imageWidth*resolutionX;
            i = ([y1, y2]+imageHeight/2)/imageHeight*resolutionY;

            plot(j, i, 'b');
            hold on;
      end;
   end;

   %Draws vehicle positions
   for frameNr=1:length(zPositions),
      pos1 = transform([vehicleXPos-vehicleWidth, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
      pos2 = transform([vehicleXPos+vehicleWidth, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
      pos3 = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
      pos4 = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
```

```
      [x1, y1] = xyProjection(pos1, focalLength);
      [x2, y2] = xyProjection(pos2, focalLength);
      [x3, y3] = xyProjection(pos3, focalLength);
      [x4, y4] = xyProjection(pos4, focalLength);

      j = ([x1, x2]+imageWidth/2)/imageWidth*resolutionX;
      i = ([y1, y2]+imageHeight/2)/imageHeight*resolutionY;
      lightj = ([x3, x4]+imageWidth/2)/imageWidth*resolutionX;
      lighti = ([y3, y4]+imageHeight/2)/imageHeight*resolutionY;

      if(frameNr == firstFrame | frameNr == lastFrame)
         color = 'm';
      else
         color = 'r';
      end;
      plot(j, i, color);

      plot(lightj(1), lighti(1), 'g*');
      plot(lightj(2), lighti(2), 'g*');

      if(frameNr == firstFrame | frameNr == lastFrame)
         color = 'b';
      else
         color = 'g';
      end;
      plot(lightj, lighti, color);
   end;

   axis ij;
   axis([0, resolutionX, 0, resolutionY]);
end;


%Calculate distance and velocity from image information
%Parameters
if(doCalculations == 1)
   nrOfSets       = 4;
   nrOfTests          = 1000;

   resultTable        = zeros(nrOfTests, nrOfSets);

   for testSpeedSet=1:nrOfSets,
      switch testSpeedSet
         case 1
            minSpeed = 140/3.6;
            maxSpeed = 160/3.6;
            disp('Speed 140 - 160 km/h');
         case 2
            minSpeed = 160/3.6;
            maxSpeed = 180/3.6;
            disp('Speed 160 - 180 km/h');
          case 3
            minSpeed = 180/3.6;
            maxSpeed = 200/3.6;
            disp('Speed 180 - 200 km/h');
         otherwise
            minSpeed = 200/3.6;
            maxSpeed = 220/3.6;
            disp('Speed 200 - 220 km/h');
      end;

      maxDif   = 0;
      testNr   = 0;

      for speed=minSpeed:(maxSpeed - minSpeed)/nrOfTests: maxSpeed,
         testNr    = testNr + 1;
         zPos      = vehicleZPos;
         frameNr   = 1;

         %Vehicle Positions
         clear zPositions;
```

```
         while zPos > 0,
            zPositions(frameNr) = zPos;
            time = frameNr / frameRate;
            zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
            frameNr = frameNr +  1;
         end;

         %Find first frame
         deltaH = 1000;
         frameNr = 1;
         while deltaH > imageWidth/2,
            pos = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
            [deltaH, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         firstFrame = frameNr-1;

         %Find last frame
         deltaV = 1000;
         frameNr = firstFrame;
         while deltaV > -imageWidth/2,
            pos = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
            [deltaV, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         lastFrame = frameNr-2;

         vehLeftPos = zeros(1, lastFrame-firstFrame);
         vehRightPos = zeros(1, lastFrame-firstFrame);
         vehDist      = zeros(1, lastFrame-firstFrame);
         vehImg = zeros(1, lastFrame-firstFrame);

         for frameCounter=firstFrame:lastFrame,
            %Calculate left light position
            pos = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameCounter), 1]', cameraTransf);
            [deltaV, y] = xyProjection(pos, focalLength);
            vehLeftPos(1, frameCounter-firstFrame+1) =
round((deltaV+imageWidth/2)/imageWidth*resolutionX);

            %Calculate right light position
            pos = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameCounter), 1]', cameraTransf);
            [deltaH, y] = xyProjection(pos, focalLength);
            vehRightPos(1, frameCounter-firstFrame+1) =
round((deltaH+imageWidth/2)/imageWidth*resolutionX);

            vehImg(1, frameCounter-firstFrame+1) = frameCounter;
         end;

         %Calculate distances
         vehDist = getVehDist(vehLeftPos+0.00001, vehRightPos+0.00001);

         %Calculate velocities
         vehVel = getVehVel(vehDist, vehImg, frameRate);

         %Insert results in table
         nrOfFrames(testNr, testSpeedSet) = lastFrame-firstFrame;
         resultTable(testNr, testSpeedSet) = abs(vehVel - speed);
      end;
   end;

   disp('Average error [km/h]')
   mean(resultTable)*3.6

   disp('Standard deviation [km/h]')
   std(resultTable)*3.6

   disp('Max error [km/h]');
```

```
   max(resultTable)*3.6

   disp('Average number of frames used');
   mean(nrOfFrames)
end;
```

6.5.3 Funksjon – transform.m

```
function  newPos = transform(oldPos, transformation)

xAngle = transformation(1);
yAngle = transformation(2);
zAngle = transformation(3);
xTranslation = transformation(4);
yTranslation = transformation(5);
zTranslation = transformation(6);

cx = cos(xAngle);
cy = cos(yAngle);
cz = cos(zAngle);
sx = sin(xAngle);
sy = sin(yAngle);
sz = sin(zAngle);

T = [   cz*cy   cz*sy*sx-sz*cx     cz*sy*cx+sz*sx     xTranslation
        sz*cy   sz*sy*sx+cz*cx     sz*sy*cx-cz*sx     yTranslation
        -sy     cy*sx                  cy*cx          zTranslation
        0           0                     0              1           ];

newPos = T*oldPos;
```

6.5.4 Funksjon – xyProjection.m

```
function [x, y] = xyProjection(xyzPos, focalLength)

x1 = xyzPos(1);
y1 = xyzPos(2);
z1 = xyzPos(3);

x = x1*focalLength/z1;
y = y1*focalLength/z1;
```

6.6 PROGRAM FOR KRAVANALYSE (KAMERA PÅ BRO)

6.6.1 Hovedskript – criteriaSim.m (kamera på bro)

```
%Ramon Kristian Arellano
%22.11.1999
%Krav-analyse
%Kamera står på bro

clear all;
format compact;

%Global variables
```

```
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
global frameRate;

%Variables
%(All positions are relative to camera position [0, 0, 0]);
roadWidth             = 8.5;                    %Total width of both lanes
roadXPos              = roadWidth/4;        %Horizontal position of middle line
roadYPos          = 7.9;                    %Vertical position of road
vehicleXPos       = roadXPos-roadWidth/4;   %Horizontal position of car in left lane
vehicleYPos       = roadYPos;               %Vertical position of car wheels.
vehicleZPos       = 100;                    %Distance between camera and vehicle
vehicleLightYPos  = roadYPos-0.5;           %Vertical position of car lights
vehicleWidth      = 1.5;                    %Distance between lights.
cameraElevation   = 11*(2*pi/360);          %Camera is pointing 10 degrees downwards
cameraAsimuth     = 0;                      %Camera is pointing 10 degrees to the left
cameraTransf      = [cameraElevation, cameraAsimuth, 0, 0, 0, 0]; % rotation and transl.
focalLength       = 0.1;
lensOpening       = 5*(2*pi/360);
resolutionX       = 640;

resolutionY       = 480;
imageWidth        = 2*focalLength*tan(lensOpening/2);
imageHeight       = imageWidth*resolutionY/resolutionX;
acceleration      = 0;
frameRate         = 1/0.08;

%Parts of script to be run
findResYCriteria = 1;
findPixDevCriteria = 1;

%Constant parameters
nrOfTests = 1000;     %Number of tests per run


%Set pixel deviation and calculate for different values of vertical resolution
if findResYCriteria == 1
   pixelStdDev = 0.8;
   runNum = 1;

   for resolutionY = 200:100:1700,
      stdDev = getPrecision(pixelStdDev, resolutionY, nrOfTests);
      resolutionVec(runNum) = resolutionY;
      stdDevVec1(runNum, 1:4) = stdDev;
      runNum = runNum + 1;

      %Temporary
      resolutionY
      stdDev
   end;

   figure(1);
   plot(resolutionVec, stdDevVec1);
end;
```

```
%Set vertical resolution and calculate pixel standard deviation criteria
if findPixDevCriteria == 1
   resolutionY = 480;
   runNum = 1;
   for pixelStdDev = 1.5:-0.1:0,
      stdDev = getPrecision(pixelStdDev, resolutionY, nrOfTests);
      pixelStdDevVec(runNum) = pixelStdDev;
      stdDevVec2(runNum, 1:4) = stdDev;
      runNum = runNum + 1;

      %Temporary
      pixelStdDev
      stdDev
   end;

   figure(2);
   plot(pixelStdDevVec, stdDevVec2);
end;
```

6.6.2 Funksjon – getPrecision.m (kamera på bro)

```
function stdDev = getPrecision(pixelStdDev, resolutionY, nrOfTests)

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

%global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
global frameRate;

   nrOfSets        = 4;
   resultTable         = zeros(nrOfTests, nrOfSets);

   for testSpeedSet=1:nrOfSets,
      switch testSpeedSet
         case 1
            minSpeed = 60/3.6;
            maxSpeed = 80/3.6;
            %disp('Speed 60 - 80 km/h');
         case 2
            minSpeed = 80/3.6;
            maxSpeed = 100/3.6;
            %disp('Speed 80 - 100 km/h');
          case 3
            minSpeed = 100/3.6;
            maxSpeed = 120/3.6;
            %disp('Speed 100 - 120 km/h');
         otherwise
            minSpeed = 120/3.6;
            maxSpeed = 140/3.6;
```

```
            %disp('Speed 100 - 140 km/h');
      end;

      maxDif   = 0;
      testNr   = 0;

      for speed=minSpeed:(maxSpeed - minSpeed)/nrOfTests: maxSpeed,
         testNr    = testNr + 1;
         zPos      = vehicleZPos;
         frameNr   = 1;

         %Vehicle Positions
         clear zPositions;
         while zPos > 0,
            zPositions(frameNr) = zPos;
            time = frameNr / frameRate;
            zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
            frameNr = frameNr +  1;
         end;

         %Find first frame
         y = -1000;
         frameNr = 1;
         while y < -imageHeight/2,
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         firstFrame = frameNr-1;

         %Find last frame
         y = -1000;
         frameNr = firstFrame;
         while y < imageHeight/2,
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameNr), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         lastFrame = frameNr-2;

         vehPos = zeros(1, lastFrame-firstFrame);
         vehImg = zeros(1, lastFrame-firstFrame);

         for frameCounter=firstFrame:lastFrame,
            %Calculate pixel y-position
            pos = transform([vehicleXPos, vehicleYPos, zPositions(frameCounter), 1]',
cameraTransf);
            [x, y] = xyProjection(pos, focalLength);

            %Rounding (quantization) is done
            vehPos(1, frameCounter-firstFrame+1) =
round((y+imageHeight/2)/imageHeight*resolutionY);

            %Pixel deviaton is added
            pixelDev = round(randn(1)*pixelStdDev);
            vehPos(1, frameCounter-firstFrame+1) = vehPos(1, frameCounter-firstFrame+1) +
pixelDev;

            vehImg(1, frameCounter-firstFrame+1) = frameCounter;
         end;

         %Calculate distances
         vehDist = getVehDist(vehPos+0.00001);

         %Calculate velocities
         vehVel = getVehVel(vehDist, vehImg, frameRate);

         %Insert results in table
         resultTable(testNr, testSpeedSet) = abs(vehVel - speed);
```

```
      end;
   end;

%   disp('Standard deviation [km/h]')
stdDev = std(resultTable)*3.6;
```

6.6.3 Hovedskript – criteriaSim.m (kamera i bil)

```
%Ramon Kristian Arellano
%22.11.1999
%Krav analyse
%Kamera står i bil
clear all;
format compact;

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
global frameRate;

%Variables
%(All positions are relative to camera position [0, 0, 0]);
roadWidth              = 8.5;                    %Total width of both lanes
roadXPos               = -roadWidth/4;           %Horizontal position of middle line
roadYPos               = 1.5;                    %Vertical position of road
vehicleXPos        = roadXPos-roadWidth/4;   %Horizontal position of car in left lane
vehicleYPos        = roadYPos;           %Vertical position of car wheels.
vehicleZPos        = 100;                    %Distance between camera in vehicle along
road
vehicleLightYPos       = roadYPos-0.5;           %Vertical position of car lights
vehicleWidth           = 1.6;                    %Distance between lights.
cameraElevation        = 2*(2*pi/360);           %Camera angle downwards
cameraAsimuth          = 10*(2*pi/360);          %Camera angle the left
cameraTransf           = [cameraElevation, cameraAsimuth, 0, 0, 0, 0]; % rotation around
xyz, and translation along xyz
focalLength        = 0.1;
lensOpening        = 12*(2*pi/360);
resolutionX        = 640;

resolutionY        = 480;
imageWidth             = 2*focalLength*tan(lensOpening/2);
imageHeight        = imageWidth*resolutionY/resolutionX;
acceleration           = 0;
frameRate              = 1/0.08;

%Parts of script to be run
findResYCriteria = 1;
findPixDevCriteria = 1;
```

```
%Constant parameters
nrOfTests = 1000;      %Number of tests per run

%Set pixel deviation and calculate for different values of vertical resolution
if findResYCriteria == 1
   pixelStdDev = 0.8;
   runNum = 1;

   for resolutionX = 400:100:1900,
      stdDev = getPrecision(pixelStdDev, resolutionX, nrOfTests);
      resolutionVec(runNum) = resolutionX;
      stdDevVec1(runNum, 1:4) = stdDev;
      runNum = runNum + 1;

      %Temporary
      resolutionX
      stdDev
   end;

   figure(3);
   plot(resolutionVec, stdDevVec1);
end;


%Set vertical resolution and calculate pixel standard deviation criteria
if findPixDevCriteria == 1
   resolutionX = 640;
   runNum = 1;
   for pixelStdDev = 1.5:-0.1:0,
      stdDev = getPrecision(pixelStdDev, resolutionX, nrOfTests);
      pixelStdDevVec(runNum) = pixelStdDev;
      stdDevVec2(runNum, 1:4) = stdDev;
      runNum = runNum + 1;

      %Temporary
      pixelStdDev
      stdDev
   end;

   figure(4);
   plot(pixelStdDevVec, stdDevVec2);
end;
```

6.6.4 Funksjon – getPrecision.m (kamera i bil)

```
function stdDev = getPrecision(pixelStdDev, resolutionX, nrOfTests)

%Global variables
global roadWidth;
global roadXPos;
global roadYPos;
global vehicleXPos;
global vehicleYPos;
global vehicleZPos;
global vehicleLightYPos;
global vehicleWidth;
global cameraElevation;
global cameraAsimuth;
global cameraTransf;
global focalLength;
global lensOpening;
%global resolutionX;

global resolutionY;
global imageWidth;
global imageHeight;
global acceleration;
```

```
global frameRate;

   nrOfSets       = 4;
   tempTable          = zeros(nrOfTests, nrOfSets);
   resultTable        = zeros(nrOfTests, nrOfSets);

   for testSpeedSet=1:nrOfSets,
      switch testSpeedSet
         case 1
            minSpeed = 60/3.6;
            maxSpeed = 80/3.6;
            %disp('Speed 60 - 80 km/h');
         case 2
            minSpeed = 80/3.6;
            maxSpeed = 100/3.6;
            %disp('Speed 80 - 100 km/h');
          case 3
            minSpeed = 100/3.6;
            maxSpeed = 120/3.6;
            %disp('Speed 100 - 120 km/h');
         otherwise
            minSpeed = 120/3.6;
            maxSpeed = 140/3.6;
            %disp('Speed 100 - 140 km/h');
      end;

      maxDif   = 0;
      testNr   = 0;

      for speed=minSpeed:(maxSpeed - minSpeed)/nrOfTests: maxSpeed,
         testNr    = testNr + 1;
         zPos      = vehicleZPos;
         frameNr   = 1;

         %Vehicle Positions
         clear zPositions;
         while zPos > 0,
            zPositions(frameNr) = zPos;
            time = frameNr / frameRate;
            zPos = vehicleZPos - (speed*time + 0.5*acceleration*time^2);
            frameNr = frameNr +  1;
         end;

         %Find first frame
         deltaH = 1000;
         frameNr = 1;
         while deltaH > imageWidth/2,
            pos = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
            [deltaH, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         firstFrame = frameNr-1;

         %Find last frame
         deltaV = 1000;
         frameNr = firstFrame;
         while deltaV > -imageWidth/2,
            pos = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameNr), 1]', cameraTransf);
            [deltaV, y] = xyProjection(pos, focalLength);
            frameNr = frameNr + 1;
         end;
         lastFrame = frameNr-2;

         vehLeftPos = zeros(1, lastFrame-firstFrame);
         vehRightPos = zeros(1, lastFrame-firstFrame);
         vehDist       = zeros(1, lastFrame-firstFrame);
         vehImg = zeros(1, lastFrame-firstFrame);

         for frameCounter=firstFrame:lastFrame,
```

```
            %Calculate left light position
            pos = transform([vehicleXPos-vehicleWidth/2, vehicleLightYPos,
zPositions(frameCounter), 1]', cameraTransf);
            [deltaV, y] = xyProjection(pos, focalLength);

            %Rounding
            vehLeftPos(1, frameCounter-firstFrame+1) =
round((deltaV+imageWidth/2)/imageWidth*resolutionX);

            %Pixel deviaton is added
            pixelDev = round(randn(1)*pixelStdDev);
            vehLeftPos(1, frameCounter-firstFrame+1) = vehLeftPos(1, frameCounter-
firstFrame+1) + pixelDev;

            %Calculate right light position
            pos = transform([vehicleXPos+vehicleWidth/2, vehicleLightYPos,
zPositions(frameCounter), 1]', cameraTransf);
            [deltaH, y] = xyProjection(pos, focalLength);

            %Rounding
            vehRightPos(1, frameCounter-firstFrame+1) =
round((deltaH+imageWidth/2)/imageWidth*resolutionX);

            %Pixel deviaton is added
            pixelDev = round(randn(1)*pixelStdDev);
            vehRightPos(1, frameCounter-firstFrame+1) = vehRightPos(1, frameCounter-
firstFrame+1) + pixelDev;

            vehImg(1, frameCounter-firstFrame+1) = frameCounter;
         end;

         %Calculate distances
         vehDist = getVehDist(vehLeftPos+0.00001, vehRightPos+0.00001);

         %Calculate velocities
         vehVel = getVehVel(vehDist, vehImg, frameRate);

         %Insert results in table
         resultTable(testNr, testSpeedSet) = abs(vehVel - speed);
      end;
   end;
%   disp('Standard deviation [km/h]')
   stdDev = std(resultTable)*3.6;
```