Reduksjon av bildedata ved identifisering av menneskeskapte objekter

---

Tor Nordseth

Våren 2005

Sammendrag

Algoritmen i oppgaven bruker egenskapsidentifisering til å finne deler av landskapsbilder som inneholder menneskeskapte objekter. Bildene er tatt i det infrarøde spekteret og viser varmestrålingen fra landskapet. Ut ifra testdata ble det manuelt identifisert fire egenskaper som for det meste tilhørte menneskeskapte objekter. Fem delealgoritmer for egenskapsidentifisering er utviklet og testet (hvorav to identifiserer den samme egenskapen):

1) Lyse områder definert ved statisk terskel
   Denne gav poeng for områder som hadde høyere stråling enn normalt for generell bakgrunn.
2) Lyse områder definert ved dynamisk terskel
   Denne gjorde det samme som den forrige, men ved å tilpasse terskelen gav den poeng til flere menneskeskapte objekter og færre naturlige områder.
3) Markante rette kanter
   Denne gav poeng til områder med rette kanter som var sterkere enn en fastsatt terskel.
4) Parallelle linjer
   Denne gav poeng til områder med par av kanter som var parallelle.
5) Uniforme arealer ved rette kanter
   Denne gav poeng til områder med rette kanter liggende inntil uniforme flater.

Den første delealgoritmen for egenskapsidentifisering ble forkastet fordi den andre delealgoritmen gjorde tilsvarende nytte, men med bedre resultater. Den fjerde delealgoritmen for egenskapsidentifisering ble forkastet fordi den ikke klarte å skille mellom antydninger til parallelle linjer i områder med bakgrunn og i tilfeller ved menneskeskapte objekter.

Det er brukt et representativt utvalg av bilder med forskjellige egenskaper for å vise at algoritmen skilte godt ut områder med menneskeskapte objekter og forkastet mye uinteressant bildedata. Som følge av de egenskapene det er valgt å bruke, er algoritmen avhengig av at de menneskeskapte objektene enten stråler mye varme eller inneholder rette kanter lange nok til å skille seg fra støy i bakgrunnen. De menneskeskapte objektene som ikke gjenkjennes av algoritmen som menneskeskapte er derfor små og med liten varmestråling.

Innholdsfortegnelse

Figurliste

Forord

Denne oppgaven er skrevet som en diplomoppgave ved fakultetet for informasjonsteknologi, matematikk og elektronikk ved Norges Teknisk- Naturvitenskaplige Universitet. Oppgaven er utført i samarbeid med Kongsberg Defence and Aerospace AS. I den anledning ønsker jeg å takke gjengen på KDA Kjeller, for testdata og andre resurser til prosjektet, og for et ellers godt samarbeid gjennom hele prosjektet.

En spesiell takk til Arne Jan Rødland for tilrettelegging og veiledning.

Kapittel 1: Innledning

I dette kapitlet utdypes oppgaven utover den korte oppgaveteksten. Først forklares begrepet *drone*, og hvorfor disse kan ha stor nytte av å komprimere bildedata med den metoden som er gitt i oppgaveteksten. Dette delkapitlet er ment å gi større innsikt i hvordan og hvorfor oppgavens algoritme skal brukes. Denne oversikten vil gjøre senere avveininger forståelig i forhold til hvordan algoritmen er ment brukt. Deretter følger ett delkapittel som gir bakgrunnsinformasjon om testdataene som er brukt gjennom rapporten. Det er disse som utgjør ”en serie digitale landskapsbilder med tilhørende opptaksinformasjon, tatt med en drone” som det uttrykkes i oppgaveteksten. Det blir satt lys på egenskaper ved disse som kan være relevante for algoritmen og hvordan denne fungerer på dataene. Til slutt i denne innledningen vil det bli gitt en oversikt over hvordan resten av rapporten er bygd opp. Dette vil si en oversikt over hvilke kapitler som er med og kort hva hver av disse inneholder.

1.1 Algoritmens anvendbarhet i droner

En drone forklares i [w1] som et ubemannet fly, gjerne utstyrt med sensorer som radar og kameraer. Droner betegnes ofte med den engelske forkortelsen *UAV* (*Unmanned Aerial Vehicle*). Dette var oppslagsordet i [w1], samt det mest brukte uttrykket i faglitteraturen jeg har lest (se kildelisten bak i rapporten). I denne rapporten vil det derimot konsekvent bli brukt *drone*, for å unngå unødige engelske forkortelser.

Blant bruksfeltene til dronene nevner [w1] overvåkning og fotografering. Dette kan være med både militære og sivile hensikter. At dronen er ubemannet og kan dekke relativt store arealer i forhold til tiden den bruker, gjør den svært godt egnet til rekognosering i fiendtlig territorium. Ved en flytur med regelmessig fotografering av bakken vil dronen fort samle inn en vesentlig mengde data. Hvor stor denne mengden er, avhenger av faktorer som bildestørrelse, bilderate og lagringsformat. Som et illustrativt eksempel kan det nevnes at en bildestørrelse på 1024 * 1024 piksler tatt med en bilderate på ett bilde annethvert sekund, og hvor hver piksel lagres med 3 byte (en byte for hver av de tre fargekomponentene rød, grønn og blå), gir en datastrøm på 1,5 MB/sek. Dette er rådata som må tas vare på, enten prosessert eller som rådata. I tillegg kan man tenke seg muligheten til samtidsovervåkning ved hjelp av en drone. Dette vil si at bildene blir sendt tilbake til en operatør rett etter at de er tatt, slik at operatøren til enhver tid kan se hva som foregår under dronen.

Noen droner er spesialisert på å fly i store høyder og er utstyrt med kameraer med svært god forstørrelse. Dette gir relativt gode bilder av bakken i forhold til avstanden. Andre droner er konstruert for å fly svært lavt. Disse er gjerne små og raske for å unngå å bli skutt ned. For disse dronene er det vanskelig å opprettholde en høy båndbredde siden den lille flykroppen ikke tillater store antenner. Mindre antenner gir mindre muligheter til å styre signalet i en bestemt retning, noe som igjen gjør at styrken på signalet faller raskt med avstanden til antennen. Etter hvert som signalets styrke blir mindre i forhold til støyen i signalet, blir det nødvendig med lavere rate for å bevare signalet. For ikke å bli oppdaget av eventuelle fiendtlige sensorer er det ofte også ønskelig med lav signalstyrke.

For slike droner er båndbredden en flaskehals for hvor mye data som kan sendes. Høy komprimering av dataene blir dermed en viktig del av effektiviteten til dronen. Jo høyere

komprimering som foretas, desto mer data kan sendes, og dronen kan rekke å sende bilder med finere oppløsning og/eller flere bilder per tidsintervall.

Ofte er det de menneskeskapte objektene i bildene som er interessante for den som foretar rekognoseringen. Det meste av dataene som samles viser derimot landskap mellom slike objekter. Dette avhenger selvfølgelig av byggetettheten på bakken, men så lenge det ikke flys inne i tettbebygde strøk, vil det være store andeler uinteressante områder i bildene, som skog, jorder og vann. Hvis all bildedata behandles likt vil store andeler av lagrings- eller kommunikasjonskapasiteten bli brukt på uinteressant informasjon. Her ligger det et potensial til å bedre utnyttelsen av lagrings- og/eller kommunikasjonskapasiteten ved å kun behandle informasjon som kan være interessant for operatøren.

Siden områdene i bildene skal kunne klassifiseres som interessant eller uinteressant i dronen må klassifiseringen foregå helautomatisk. I tillegg har ikke klassifiseringsproblemet et entydig svar. Det er ikke gitt hva enhver operatør vil finne interessant av områdene under dronens flybane. Det er heller ikke gitt hva som bør karakteriseres som menneskeskapte objekter. Av bildene kan det være umulig å avgjøre en slik karakterisering, både automatisk og manuelt. Det er heller ikke gitt noen nedre grense for hvor mye som kan filtreres vekk som uinteressant av bildedataene. Man kan tenke seg tilfeller der hele bilder er fylt med potensielt interessante objekter for operatøren. I dette tilfellet kan klassifiseringsarbeidet være fåfengt, siden man ikke oppnår noen komprimering av dataene. I så måte kan problemet karakteriseres som et optimaliseringsproblem der sikkerheten for å få med alle mulige potensielt interessante områder vurderes opp mot komprimeringseffektiviteten ved å forkaste så mange og store områder av bildedataene som mulig.

Denne komprimeringsalgoritmen skiller seg fra mer tradisjonelle komprimeringsalgoritmer på ett vesentlig punkt. Mens vanlige komprimeringsalgoritmer som oftest koder en mengde informasjon i en mindre mengde data, minker denne algoritmen selve informasjonen som skal sendes. Dette gjør det mulig å bruke andre komprimeringsalgoritmer på den resterende informasjonen for dermed å oppnå en videre reduksjon av datamengden. Som eksempel kan JPEG2000 nevnes som en tradisjonell bildekomprimeringsalgoritme. Selv om den er forholdsvis ny og blant de mest moderne bildekomprimeringsalgoritmene, kan den karakteriseres som tradisjonell her på bakgrunn av dens anvendelse: Komprimere et bilde i minst mulig data. Denne algoritmen vil kunne brukes i tillegg til oppgavens algoritme. Oppgavens algoritme vil først kunne trekke ut relevant informasjon fra bildedataene. Deretter kan de resterende bildedelene komprimeres til JPEG2000 format. Dermed vil man kunne utnytte begge algoritmene for størst mulig kompresjon av informasjonen. Bildedelene som klassifiseres som interessante kan presenteres for en eventuell tradisjonell bildekomprimeringsalgoritme på forskjellige måter, som for eksempel ved å sette alle andre piksler til en fast gråtone (eventuelt svart), eller som mange forskjellige mindre bilder. Dette må sees i sammenheng med den tradisjonelle bildekomprimeringsalgoritmen og hvordan denne best mulig får komprimert bildene videre. Mer om JPEG2000 kan leses i [a1].

Siden oppgavens algoritme og tradisjonelle komprimeringsalgoritmer ikke er gjensidig utelukkende men kan anvendes i sekvens på samme datamengde, er det irrelevant å sammenligne kompresjonseffektiviteten mellom disse for å finne den beste. For å gi et inntrykk av kompresjonseffektiviteten til oppgavens algoritme bør man heller se på andelen data forkastet som uinteressant i forhold til den totale datamengde. Skal dataene lagres på disk og

datamengdens plass på disken er det man ønsker å minimalisere, vil den totale datamengden være alle bildene i opptaket og den forkastede andelen all forkastet data gjennom hele opptaket. Skal derimot dataene sendes fortløpende over en kommunikasjonslink vil det være mer relevant å se på forkastet data i forhold til total datamengde for hvert bilde, slik at ett bilde med lite forkastet data ikke tetter flaskehalsen som en treg link vil representere i systemet. Brukes det en form for buffer får man en mellomting mellom de to ekstreme eksemplene nevnt, og antall bilder som går i bufferen samtidig vil utgjøre den totale datamengden i effektivitetsbetraktningen. Vil man gi et videre inntrykk av hvor godt algoritmen fungerer må man også vurdere hvorvidt algoritmen ivaretar interessant data.

1.2 Testdata

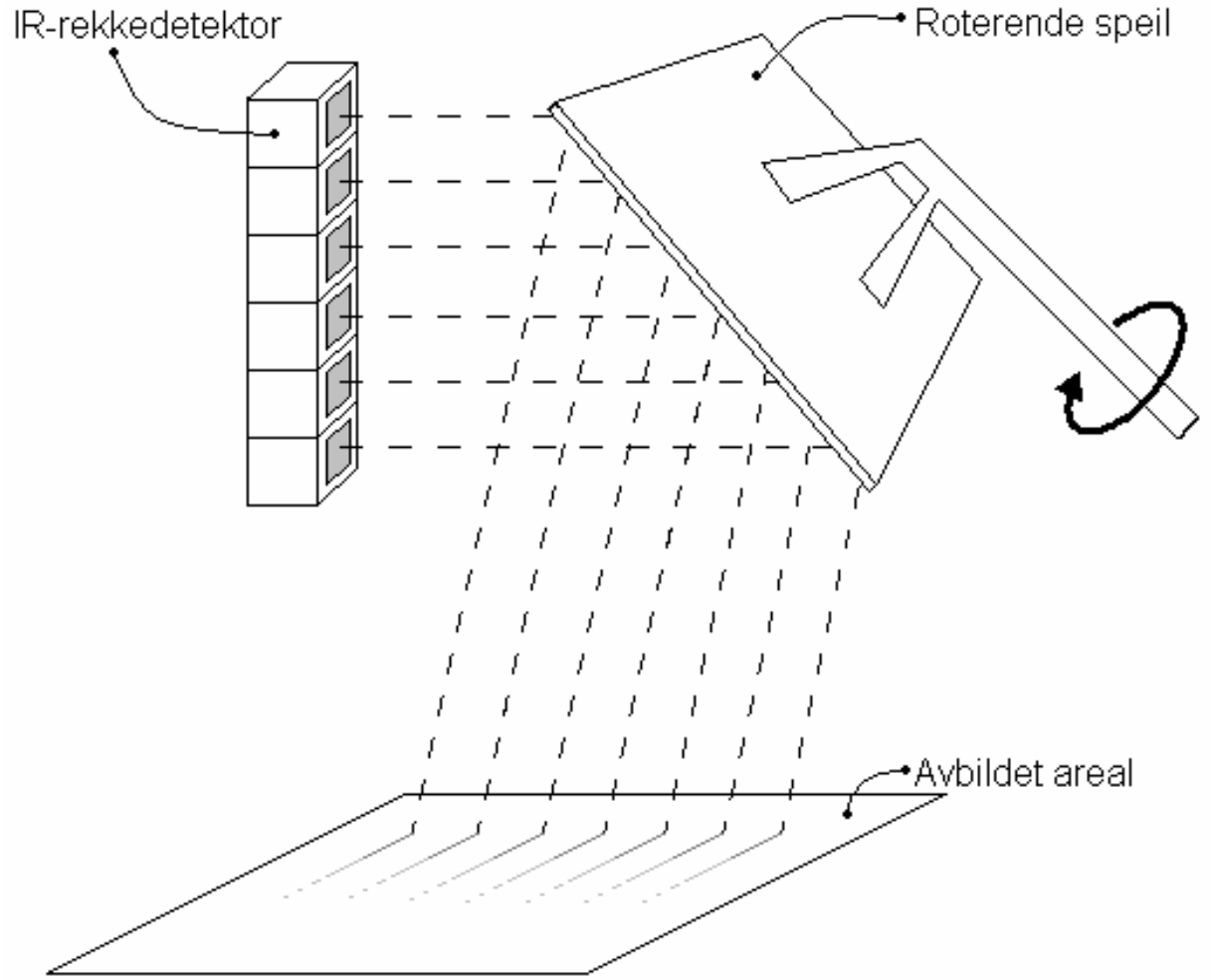
Til denne oppgaven har Kongsberg Defence and Aerospace AS (KDA) gitt meg tilgang til opptak tatt med utstyr til en typisk lavtflygende, liten drone. Opptakene er tatt med testutstyr festet på undersiden av et jagerfly som har drevet lavtflygning langs utvalgte baner. Disse banene er begrenset til å være innenfor de definerte lavtflygningssonene i Norge. Disse sonene er igjen områder der lavtflygning vil være til sjenanse for færrest mulig, noe som for det meste vil si fjellstrøk. I tillegg er det lavtflygningssoner rundt flyplassene av opplagte grunner. Dermed kan man tenke seg at det er noe mindre bebyggelse og andre menneskeskapte objekter i disse opptakene enn hva det normalt ville vært i et reelt rekognoseringsopptak. Flybanene er derimot lagt ut med tanke på å få opptak med interessante objekter, og skulle på den måten dekke relevante områder.

Til hvert av bildene registreres det også navigasjonsdata. Denne gir en global posisjonering av kameraet til ethvert bilde, samt orienteringen av kameraet.

Beskrivelsen av testdataene i dette delkapitlet er delt i to underdeler. Siden typen av sensor har mye å si for rådataene, beskrives sensorene først, og deretter egenskaper ved selve dataene.

1.2.1 Sensoren

Kameraet som er brukt fanger stråling i den infrarøde (IR) delen av det elektromagnetiske spekteret. Kameraet har en skannende IR-rekkedetektor. Denne rekkedetektoren består av like mange elementer som det er piksler i høyden på bildene, og kan dermed sample en bildekolonne av gangen. Rekkedetektoren ser ut av kameraet via et speil, slik figur 1 viser. Ved å dreie speilet ser rekkedetektoren ut i verden med forskjellige asimutvikler. For å bygge opp et bilde dreies speilet en liten vinkel mellom hver gang rekkedetektoren sampler en bildekolonne. Denne kameraoppbygningen gir mulighet for et synsfelt med stort spenn i asimutvinkel og høy oppløsning.



Figur 1: Skannende IR-detektor

IR-sensoren er DC-koblet. Dette betyr at lik innstråling til sensoren gir den samme utgangsverdien hver eneste gang sensorene brukes.

I tillegg til rådataene fra IR-rekkedetektoren, blir det til hvert bilde registrert metadata om opptakssted og retning for sensoren i det bildet ble tatt. Siden KDA ikke kan publisere all informasjon om den registrerte metadata, vil verdien til en del sentrale parametere ikke bli oppgitt i utregninger i eksemplene. Dette legger ingen hindring for forståelsen av utregningene eller hvordan algoritmene fungerer, siden det av parametrenes navn vil fremgå hva slags verdi den inneholder. Eksempler på slike parametere er opptaksbildenes oppløsning, synsvinkelen til kameraet, opptaksfrekvensen, absoluttverdien til pikslene i råbildene og fullstendig innhold av metadataen. Verdiene vil i koden bli lest fra de innlastede opptakene eller metadataen, og vil i så måte representere en mer generell algoritme, og ikke kun tilpasset dette datasettet. Bildene fra opptakene gjengitt i denne rapporten vil også kun være utklipp, for ikke å gjengi bildenes format.

1.2.2 Egenskaper

Verdiene til pikslene bestemmes av mengden IR stråling fra den retningen sensoren er rettet. Strålingen et objekt sender ut øker med temperaturen til objektet, men forskjellige materialer har

forskjellige evne til utstråling ved samme temperatur. Temperaturen til objektene bestemmes av to ledd: Oppvarming utenfra og indre oppvarming.

Oppvarming utenfra vil for det meste si oppvarming som følge av sola. Alle objektene i testdataene har et minimum av utstråling som følge av solas oppvarming av kloden. I de tilfeller der objektene ligger ute i solsteiken, som for eksempel steiner eller husvegger, kan temperaturen til objektene stige til uvanlige høyder.

Den indre oppvarmingen forekommer som regel i forbindelse med menneskeskapte objekter. Hus er, i hvert fall i de kaldere årstider, gjerne oppvarmet innenfra og holder en høyere temperatur enn omgivelsene. Hvor mye huset som objekt er oppvarmet sett utenfra avhenger av hvor godt isolert bygningen er. Vinduer representerer gjerne flater som isolerer dårlig, og dermed stråler mye.

I tillegg til utstrålingen som følge av objektets temperatur (emittert stråling) reflekteres innkommende stråling på objekter. Evnen til å reflektere stråling varierer mye med forskjellige materialer. Med en kameravinkel ovenfra og nedover, slik tilfellet er med testbildene, reflekterer takflater stråling fra himmelen. Noen takflater reflekterer dårlig, og viser nesten kun emittert stråling, mens andre reflekterer godt. Disse viser lite stråling med mindre synsvinkelen skulle slumpe til å treffe solas speilbilde i taket.

Sensorene brukt for å ta bildene i testdataene kan skille den detekterte strålingen i mange flere intervaller enn de 256 som er vanlig å bruke på bilder i datamaskiner. Disse 256 forskjellige verdiene tar utgangspunkt i hvor fine gråtonenyanser det menneskelige øyet klarer å skille mellom. Siden dette er godt under 256, er det ingen hensikt å bruke mer enn 8 bit på å lagre verdien til en piksel i et bilde som kun skal vises for mennesker. Når bildene her skal prosesseres av en maskin, som praktisk sett kan skille mellom uendelig mange gråtoneverdier, er det verdt å ta vare på alle de nyansene detektorene kan skille mellom. Derfor brukes det flere enn 8 bit til å representere gråtonene i testbildene, og pikselverdiene overstiger 256.

Siden KDA heller ikke kan publisere informasjon om sensorenes sensitivitet, vil verdien av selve pikselverdier og parametere med verdier i samme spekter, være utelatt fra eksempler og den vedlagte koden i vedlegg A. Heller ikke dette legger noen hindring for forståelsen av hvordan algoritmene fungerer, siden det av parametrenes navn vil fremgå hva slags verdi den inneholder.

Når bildene er vist i rapporten er gråtoneverdiene flyttet til mellom 0 og 256. Konverteringen har foregått ved å bevare forholdene mellom pikselverdiene, men dratt pikselverdiene til å dekke spekteret mellom 0 og 256. Dermed får bildene best mulig kontrast uten å forskyve forholdet mellom pikslene, som er viktig for forståelsen av bildet. Siden én pikselverdi i forskjellige bilder vil kunne få forskjellig verdier etter konvertering, er bildene ikke ukritisk sammenlignbare med hverandre. Det må da tas hensyn til at spekteret kan være dratt forskjellig for å oppnå best mulig kontrast i hvert bilde, samt at absoluttverdiene kan være senket forskjellig for å få minste pikselverdi ned til null.

Selve innholdet i bildene blir grundigere forklart i kapittel 2.1 i sammenheng med valg av metode for algoritmen i denne oppgaven.

1.3 Rapportens oppbygning

Denne rapporten er delt i fem kapitler, hvorav denne innledningen er det første. Her er det sett på oppgavens forutsetninger i form av oppgavens innhold og testdataene. Med dette som grunnlag blir det i neste kapittel vurdert ulike metoder som skal danne grunnlaget for oppgavens algoritme. Den best egnede, egenskapsuttrekning, blir valgt. I det tredje kapitlet velges det egenskaper som passer denne metoden, på bakgrunn av hvordan menneskeskapte objekter ser ut i testdataene. For hver av disse egenskapene blir det i kapitel fire utledet algoritmer som sammen trekker ut de delene av bildet som inneholder disse egenskapene. Kapittel fem viser tester av disse algoritmene. Det neste kapitlet inneholder forslag til nyttige videre arbeidsområder innen oppgaven, før det siste og sjuende kapitlet inneholder en helhetlig konklusjon på oppgaven.

Kapittel 2: Valg av metode

I dette kapitlet begrunnes valget av metoden som ligger til grunn for algoritmen utviklet senere i oppgaven. Med metode menes det her en type algoritme. Det er tidligere blitt brukt mange forskjellige typer algoritmer for gjenkjenning av objekter i bilder. De forskjellige metodene har forskjellige egenskaper og egner seg til forskjellig problemer og data. For å komme frem til den beste algoritmen til denne rapportens oppgave vil det i dette kapitlet bli vurdert forskjellige metoder tidligere brukt til objektgjenkjenning. Dette er gjort ved å gå gjennom tidligere publiserte objektgjenkjenningsarbeider, se hvilke typer algoritme som er mye brukt og sammenligne disses egenskaper med egenskapene til denne oppgaven og de gitte testdata.

Kapitlet er delt inn i tre delkapitler. Det første tar for seg de gitte testdata og trekker frem egenskaper ved disse. Det blir gitt eksempler på innholdet i dataene i den hensikt å gi leseren en generell forståelse av innholdet. Dette innebærer både eksempler på menneskeskapte objekter og typiske bakgrunner. Det andre delkapitlet presenterer de vanligste metodene innen objektgjenkjenning fra tidligere arbeider. Disse blir presentert med tanke på egenskaper som har betydning for anvendelsen i denne oppgavens problem og data. Til slutt kommer et delkapitel der det tas et valg av metode på bakgrunn av informasjonen i de to foregående delkapitlene.

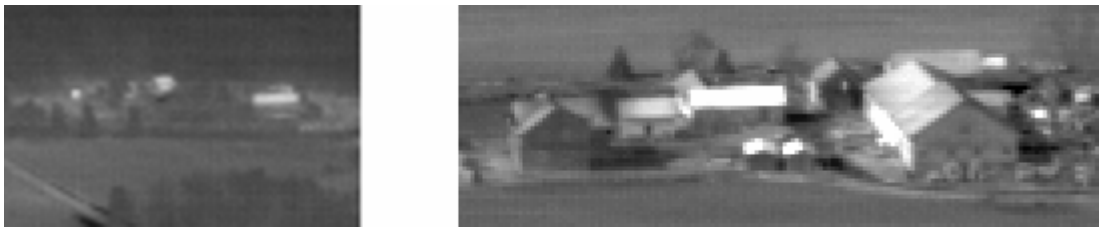
2.1 Innholdet i testdataene

Testdataene består som tidligere nevnt av flere opptak. Felles for opptakene er utstyret som er brukt, noe som gjør at bildefrekvens, pikslenes synsvinkel og andre utstyrsavhengige variable forblir konstante. Hastigheten på kameraet under opptakene er også noenlunde lik. Dette gir like stor forskyvning mellom hvert bilde. Når kameraet beveger seg i en høyde rundt 100 meter over terrenget, som er vanlig i opptakene og normalt for dronen testutstyret tilhører, beveger kameraet seg så langt mellom hvert bilde at det er liten overlapp mellom to sekvensielle bilder. En overlapp på under 50 % er vanlig. Dette gjør serien av bilder lite egnet for filmkompresjoner, som for eksempel MPEGX[1], som bygger på at det er små endringer mellom hvert bilde. Den delen av bildet som bevares til neste bilde endrer seg også en del. Dette kommer av at kameraet ikke er rettet rett nedover, men på skrå fremover. I det påfølgende bildet har dermed kameraet ikke lenger samme synsvinkel til objektet, som nå også ligger mye nærmere kameraet. Derfor er det trolig heller ikke verdt å benytte ideer fra filmkompresjon på denne delen av bildet.

Alle opptakene er gjort på vinteren (mars og desember 2004), noe som muligens kan gi en bedre kontrast i IR-bildene mellom oppvarmede menneskeskapte objekter og heller kjølig omliggende natur.

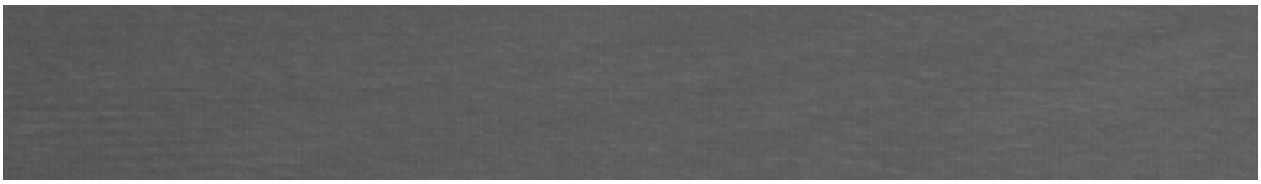
Det er også forskjeller mellom opptakene. Innholdet av opptakene er generelt forskjellige som følge av ulike flyhøyder under opptagning. Dette gir forskjellig representasjon av lignende menneskeskapte objekter ved at de kan fremstå med ulike størrelse. Figur 2 viser hvor forskjellig to gårder kan fremstå som konsekvens av dette.

[1] MPEG er en engelsk forkortelse for *Moving Picture Expert Group*. Disse har utviklet en serie standarder for lyd- og bildekompresjoner hvorav de viktigste er MPEG1, MPEG2, MPEG4 og MPEG7. Mer om MPEG kan leses på [w2].

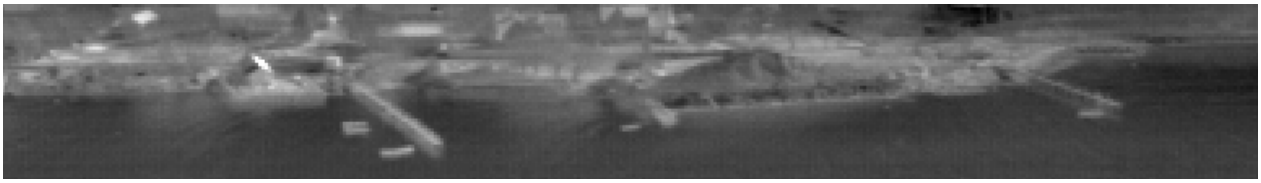


Figur 2: Bygninger fotografert fra forskjellig distanser

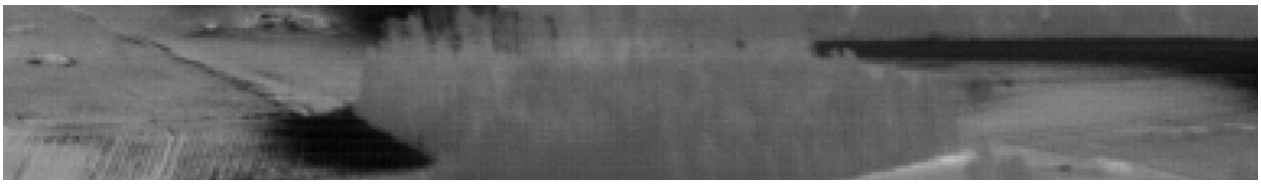
Opptak tatt på forskjellige steder gir generelt forskjellig bakgrunn. Noen opptak er tatt over fjellstrøk der bakgrunn skiller mellom skog, mindre vann, snø og stein. Andre opptak tatt over jordbruksområder med jorder avskilt med veier, bebyggelse, små skoger og små vann. Også opptak over sjø og kystlinjer er representert. I figur 3 til og med figur 8 er en del av disse eksemplene gjengitt i bilder. Dette er ment å gi leseren en generell oppfattning av hvordan innholdet i testdataene ser ut.



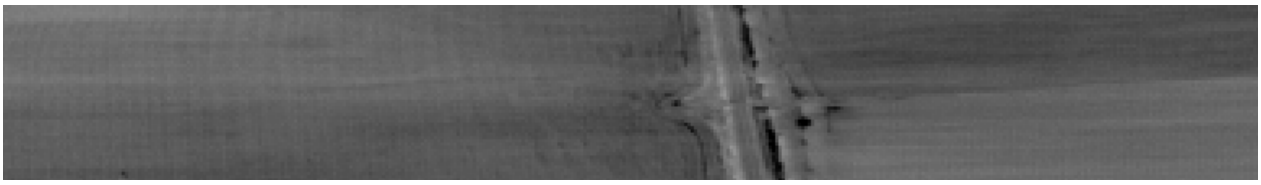
Figur 3: Hav



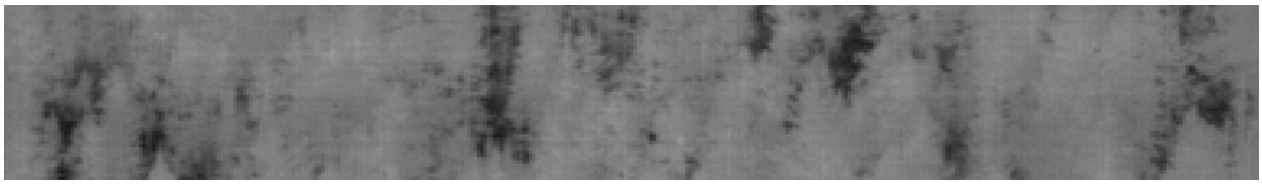
Figur 4: Kystlinje



Figur 5: Jorde med skogholt



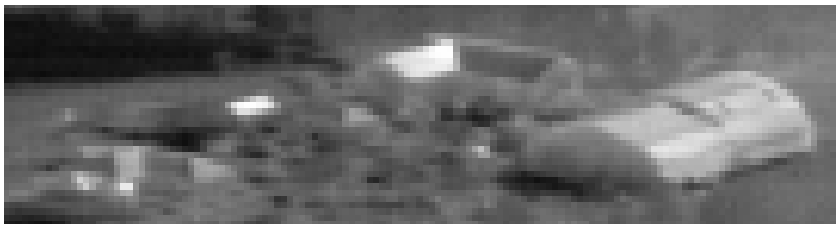
Figur 6: Jorde delt av jernbanelinje



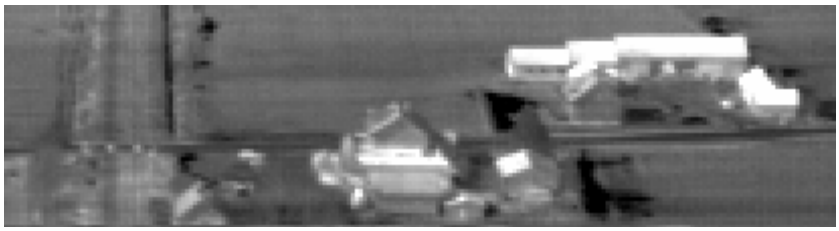
Figur 7: Skog

Figur 8: Skog med tjern

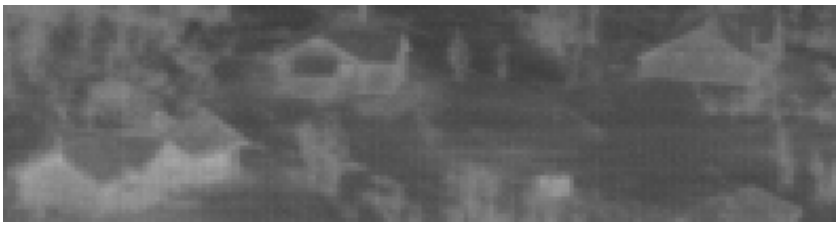
For å gi et inntrykk av spekteret av menneskeskapte objekter og hvordan disse arter seg i bildene er det i figur 9 til og med figur 14 gitt eksempler fra opptakene.



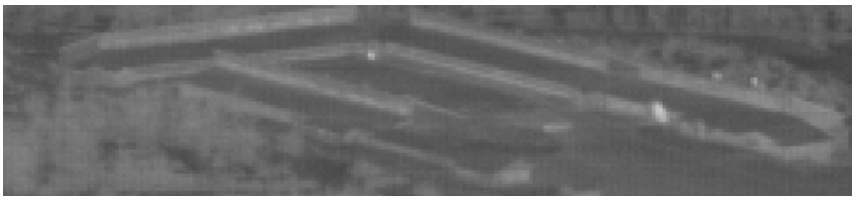
Figur 9: Gård



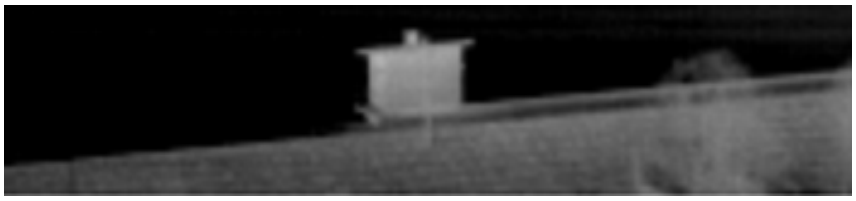
Figur 10: Bolighus, vei og jernbanelinje



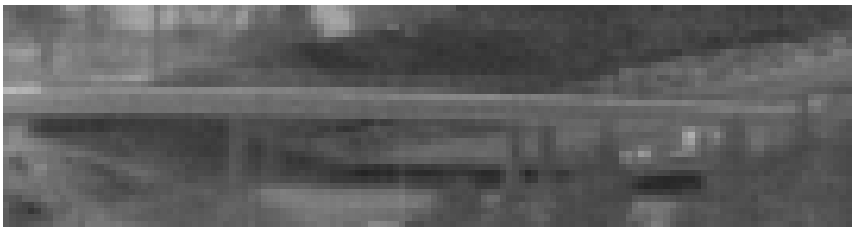
Figur 11: Utdrag av hyttefelt



Figur 12: Ukjent stor bygning (stadion?)



Figur 13: Demning



Figur 14: Bru

2.2 Tidligere arbeider

I søk etter tidligere utførte arbeider innen digital objektgjenkjenning har jeg i stor grad basert meg på store internasjonale vitenskaplige artikkeldatabaser på Internett. Disse databasene består av:

- IEEE (Institute of Electrical and Electronic Engineers Inc.) sin artikkeldatabase på deres hjemmeside [w3]. Her kan man også lese mer om IEEE. Søkene i denne databasen er foretatt gjennom søkegrensesnittet på [w4].
- SPIE (The International Society for Optical Engineering) sin artikkeldatabase på deres hjemmeside [w5]. Her kan man også lese mer om SPIE. Søkene i denne databasen er foretatt gjennom søkegrensesnittet på [w6].
- ACM (Association for Computing Machinery) sin artikkeldatabase på deres hjemmeside [w7]. Her kan man også lese mer om ACM. Søkene i denne databasen er foretatt gjennom søkegrensesnittet på [w8].

I tillegg er Google [w9], en generell søkemotor for Internettsider, brukt. I søkene ble det brukt nøkkelfraser som ”object recognition”, ”feature extraction” og ”man made objects” i sammenheng med ”image processing” og ”IR”. De 75 første treffene ble vurdert. Det ble ikke funnet relevante dokumenter i løpet av de 25 siste treffene av disse 75, så dette ble regnet som mange nok.

I tillegg er det lest en del trykte artikler. Dette innebærer både samling av artiklene til SPIEs konferanse *Automatic Target Recognition XIII* 22.-24. april 2003, samt tilhørende kursnotater, og dokumentering av tester gjort ved KDA.

Videre nevnes sentrale fremgangsmetoder som går igjen innen objektgjenkjenning, og det blir diskutert hvorvidt disse passer denne oppgaven.

2.2.1 Nevrale nettverk

Nevrale nett er testet mye for gjenkjenning av objekter i digitale bilder. Kort forklart går metoden ut på å simulere ett nettverk av nevroner. Den er inspirert av metoden biologiske hjerner fungerer på, som er svært effektive til å prosessere bilder. Nevronene får opptil flere inngangssignaler og sender et utgangssignal som funksjon av en vektet sum av inngangssignalene og en egen terskel dette nevronet har. Alle vektene og tersklene i nettverket utgjør nettverkets parametere. På inngangen til de første nevronene kobles egenskaper fra bildene, som for eksempel pikselverdier. På utgangene av de siste nevronene leser man av resultatverdier etter nettverkets prosessering av bildet. Ved å justere nettverkets parametere slik at man får de ønskede resultatene på nettverkets utgang(-er) ved testing på bilder med kjent ønsket resultat, tillegges nettverket den ønskelige funksjonalitet. Dette kalles å trene nettverket. Etter å ha trent nettverket med tilstrekkelig mange representative bilder håper man å få riktige resultater også ved kjøring av nettverket på bilder ikke brukt under treningen.

Metodens ulemper er blant annet at nettverket for mer avansert prosessering fort blir stort, og man mister oversikten over hvilke egenskaper som brukes og blandes i hvilke nevroner. Derfor finnes det algoritmer som automatisk justerer nettverkets parametere slik at feilen mellom ønsket utgangsverdier og de faktiske resultatene fra treningen blir stadig mindre[2]. Dette medfører at man mister oversikten over hvordan nettverket kommer frem til sitt resultat, og dermed også kontrollen over når og hvorfor algoritmen kan feile.

Dette trenger ikke alltid være et stort problem i oppgaver uten eksakte svar, slik definering av interessante og uinteressante områder i bildene. I slike oppgaver må man tåle ett visst avvik fra optimal løsning, i og med at denne endrer seg fra anvendelse til anvendelse av den ferdigprosesserte datamengden.

[a2] viser et eksempel fra lignende bruksområde som denne oppgaven. Bilder fra en drone prosesseres for å skille fly og narremål. Til klassifiseringen brukes det et nevralt nett. Inngangsverdiene til nettet er diverse egenskaper til objekter som er identifisert i bildet. Til forskjell fra denne oppgaven, har bildene i denne artikkelen en langt mer uniform bakgrunn. Bakgrunnen bak flyet og eventuelle narremål består av himmel med eventuelle skyer. Dermed blir det langt enklere å identifisere objekter fra bakgrunnen, som siden blir karakterisert som interessante (fly) eller uinteressante (narremål). Selv om forfatterne oppnår en meget god treffprosent på sine klassifiseringer mellom interessante og uinteressante objekter, kan dette ikke brukes på bilder der bakgrunnen er for broket til at mulige interessante objekter skiller seg klart ut med en definert grense. Til å skille ut objektene brukes det i [a2] en enkel form for egenskapsidentifisering, en metode som blir nærmere beskrevet i eget avsnitt.

KDA har tidligere testet nevrale nettverk for gjenfinning av objekter i lignende opptak. Disse opptakene ble tatt med samme utstyr, men målene var begrenset til båter og bakgrunnen begrenset til vann/hav. Det ble vurdert lettere for programmereren å definere forskjellen mellom områder i bildet som tilhørte skip og områder som tilhørte hav, enn å lære opp et nevralt nettverk

[2] For eksempel algoritmen *Backpropegation*, som er utførlig forklart i kapittel 3.3 i [b1].

til å gjøre det samme [p1]. Derfor ble ikke nevrale nettverk valgt som metode den gang. I tilfellet med generelle menneskeskapte objekter og forskjellige typer bakgrunn blir det vanskeligere å definere forskjellen, i og med begge kategoriene arter seg på så mange forskjellige måter. Dette gjør nevrale nettverk mer aktuelle igjen, og kan med fordel testes på nytt på dette nye datagrunnlaget.

2.2.2 Genetiske algoritmer

Genetiske algoritmer er algoritmer som utvikler seg som en følge av testing og kombinering av de bedre algoritmene. Med utgangspunkt i darwinistisk utviklingslære er målet at en mengde algoritmer stadig skal forbedre gjennomsnittsytelsen ved stadig å bytte ut mengden algoritmer med de beste algoritmene fra mengden, kombinasjoner av disse og kopier med små tilfeldige endringer. I likhet med automatisk konfigurerte nevrale nett mister også her utvikleren lett oversikten over hvordan algoritmen fungerer. Dermed blir det vanskelig å forutsi når og hvorfor algoritmen kan feile, annet enn ved erfaring fra grundig testing.

[a3] viser et eksempel svært likt denne oppgaven, der det brukes en genetisk utviklet algoritme for å finne interessante områder i bilder fra en drone. Dessverre er utviklingen av den genetiske algoritmen svært overfladisk beskrevet, men enkelte relevante forskjeller i oppgavene kan nevnes her. Til forskjell fra testdataene i denne oppgaven er bildene i [a3] tatt over ørkenlandskap. I tillegg er bildene tatt fra vesentlig større høyde. Disse to egenskapene gjør bakgrunnen i bildene, altså ørkensanden rundt militære installasjoner ute i ørkenen, langt mer uniform enn skog, fjell og vann slik de fremstår i denne oppgaven. Den uniforme bakgrunnen gjør, som tidligere nevnt under nevrale nettverk, at objektene er lettere å skille ut, og dermed kan egenskaper ved disse objektene brukes til å avgjøre hvorvidt den er interessant eller ikke. Til å skille ut objektene brukes det også her en form for egenskapsidentifisering.

2.2.3 Korrelasjon med mal

Korrelasjon med mal (eng: *Template matching*) går ut på å anvende en mal av objektet man ønsker å gjenfinne i bildene og regne ut korrelasjonen mellom malen og bildet for mulige plasseringer av malen i bildet. Mulige plasseringer innebærer i tillegg til posisjon også orientering og skalering hvis ikke disse er kjent. Høy korrelasjon angir at bildet og malen ligner, og man kan anta at objektet malen representerer befinner seg på det stedet i bildet som korrelerer godt med malen.

[a4] viser godt sentrale problemer ved bruk av korrelasjon med mal som metode for å gjenkjenne objekter i bilder. Artikkelen tar for seg både bakkemål og luftmål. Det viser godt hvor mye korrelasjonen varierer ved små forskjeller i orienteringen av malen. Små rotasjoner av en forholdsvis liten mal (i denne artikkelen er malene inneholdt i bilder på 32x32 piksler) kan man se at til dels mange piksler endrer verdi. I artikkelen ble det regnet ut korrelasjon for hver 10 grads endring langs to rotasjonsakser. Dette gav 648 forskjellige maler for luftmål og halvparten for landmål (siden det ble antatt at man aldri ville se landmålene fra undersiden). I tillegg kommer alle mulige posisjoner og størrelser for hver rotasjon, hvis alle muligheter skal beregnes. Selv om [a4] foreslår tiltak for å minke mengden beregninger fremstår metoden som meget resurskrevende i tilfeller der løsningsrommet utspent av mulige orienteringer, plasseringer og størrelser er stort.

2.2.4 Aktive konturmodeller

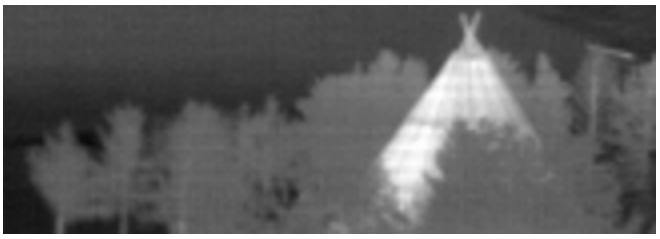
Aktive konturmodeller (eng: *Active Contour Models*), eller *slanger* som de også blir kalt, bruker egenskaper i bildene til å flytte konturer mot kanten av interessante områder. Punkter rundt på konturen flyttes som følge av kalkulerte krefter, regnet som funksjoner av egenskaper i bildet, kjente egenskaper ved områdene man ønsker konturen rundt og eventuelle andre faktorer. Metoden krever opprinnelig en initiering av konturen, men forskjellige metoder for å unngå dette er foreslått opp igjennom tidene.

Metoden brukes ofte til å definere posisjon av organer i medisinske bilder. [a5] viser ett enkelt eksempel der hele bildet blir initiert med små konturer som vokser seg sammen og legger seg rundt ryggvirvelledd, som er de ønskede objektene.

2.2.5 Egenskapsidentifisering

Egenskapsidentifisering (eng: *Feature Extraction*) er allerede nevnt som brukt i sammenheng med nevrale nett og genetiske algoritmer. Metoden består av å finne beregne mengden av en eller flere egenskaper for deretter å sammenligne med hva en forventer fra objekter av den typen man ønsker å gjenkjenne. Brukbarheten til metoden avhenger av hvilke egenskaper som beregnes. Egenskapene bør skille mest mulig mellom de interessante objektene og bakgrunn.

Av de menneskeskapte objektene presentert tidligere i kapitlet kan man se at disse har visse egenskaper som skiller dem fra bakgrunnen. Dette er de egenskapene som gjør det mulig å mistenke noen objekter for å være menneskeskapte uten å gjenkjenne hva slags objekt det er. figur 15 viser et eksempel. Uten å vite hva som faktisk er avbildet, er det mulig å mistenke det ene objektet som menneskeskapt. Grunnen til dette er enkelte egenskaper som er vanlige hos slike objekter, slik som oppvarmet i forhold til omgivelsene (lysere i bildet) og rette kanter.



Figur 15: Ukjent menneskeskapt objekt

Metoden passer godt til oppgavens bruksområde med tanke på de menneskeskapte objektenes store variasjon, men med enkelte felles egenskaper. Det vil allikevel fortsatt være et problem å finne objektene for å kunne regne ut egenskapene. En mulig løsning på dette er å dele opp bildet i små områder på størrelse med de menneskeskapte objektene man ønsker å finne, for deretter å regne ut egenskaper for hver av disse. Dette vil igjen skape problemer for enkelte egenskaper, siden det i grunnlaget for beregningene av egenskapene også vil forekomme en god del piksler tilhørende bakgrunnen. Som eksempel kan det nevnes at mange menneskeskapte objekter har en jevn overflate med nær uniform stråling. Dette gir seg utslag som et område i bildet med jevn gråtone. Variansen mellom alle pikselverdiene i et område vil gi en verdi for hvor uniformt områdets farge er, og kunne tenkes brukt som en parameter på jevne flater i metoden. Dette ville fungere dårlig på tilfeller der man i beregningsgrunnlaget får med vesentlige deler

bakgrunnspiksler, slik som for eksempel i figur 15, siden bakgrunnens mer spraglete farge vil ha betydelig innvirkning på den beregnede verdien. Ved valg av egenskaper og beregningsmetode for disse må slike hensyn tas. Som et eksempel på en bedre egenskap kan lengden av rette kanter nevnes. Ved identifisering av disse trenger man ikke vurdere hvorvidt hvilke piksler som tilhører objektet og hvilke som tilhører bakgrunnen.

En del av de andre nevnte metodene har vært ansett som svært resurskrevende. Resurskravene til egenskapsidentifisering avhenger av algoritmene som brukes til å beregne hvor sterkt fremtredende egenskapene er. Dette gjør at denne metodens resurskrav kan kontrolleres ved valg av egenskaper som beregnes, samt hvordan dette gjøres. En begrensning på tilgjengelig resurser til dette formålet vil selvfølgelig begrense hvilke egenskaper som kan beregnes, og hvor nøye disse beregningene kan være.

2.3 Valg av metode

I denne oppgaven velges det å teste egenskapsidentifisering som metode. Denne metoden ansees som den mest aktuelle fordi de interessante objektene i testdataene skiller seg ut ved kombinasjoner av egenskaper. Egenskapsidentifisering tilbyr en oversiktlig og kontrollert metode for å rangere områder med disse egenskapene. I tillegg passer det godt at resursforbruket er justerbart ved valg av identifiseringsalgoritmene som brukes.

Nevrale nett og genetiske algoritmer kunne også vært brukt i denne oppgaven, men mangel på forutsigbarhet og oversikt gjør de til mindre attraktive enn egenskapsidentifisering. Det ville allikevel vært interessant å sammenligne ytelsen av systemer med disse metodene mot systemet utviklet i denne oppgaven, men dette er det av resursmessige årsaker ikke gjort her.

Korrelasjon med mal er uaktuell fordi man ikke kan definere en mal for menneskeskapte objekters utseende. Disse kan være vidt forskjellige (hus, bil, demninger, industrianlegg, bruer og så videre), og det finnes så mange at det ikke vil være mulig å lage maler for alle. For maler av de vanligste gjengangerne av menneskeskapte objekter i testdataene, ville det være mange mulige posisjoner, orienteringer og størrelser for malene i bildene. Dette vil medføre lang beregningstid for hvert bilde og kan komme til å bli tregere enn eventuell annen flaskehals i systemet (som nevnt i kapittel 1.1).

Aktive konturmodeller er heller ikke aktuell i denne oppgaven, siden områdene som ønskes definert i testdataene ikke lett lar seg beskrive i form av krefter. De forskjellige menneskeskapte objektene har vidt forskjellige konturer. Videre vil trolig nye menneskeskapte objekter som kan dukke opp i kommende opptak ikke har konturer lignende de som er representert i testdataene. Slike ukjente konturer vil vanskelig la seg favne av eventuelle generelt definerte konturkrefter for menneskeskapte objekter. I tillegg krever metoden særdeles mye beregning per bilde, noe som vil være uforenelig med dette systemets anvendelser.

Kapittel 3: Valg av egenskaper

I dette kapitlet velges det egenskaper ved de menneskeskapte objektene som skal beregnes i algoritmen. Grunnlaget for utvelgelsen har vært en gjennomgang av testdataene, der menneskeskapte objekter manuelt er identifisert og sammenlignet.

I valget av egenskaper er det også tatt i betraktning at egenskapene skal la seg beregne av algoritmer med en relativ enkel kjøretid. Med dette menes kjøretider som ikke utelukker de bruksområdene av algoritmen som skisseres i denne rapportens innledning, og dermed bruk av de metodene som i forrige kapittel ble ansett som for resurskrevende. Derfor er det også viktig å ha en formening om hvordan egenskapen skal identifiseres, når man velger egenskaper.

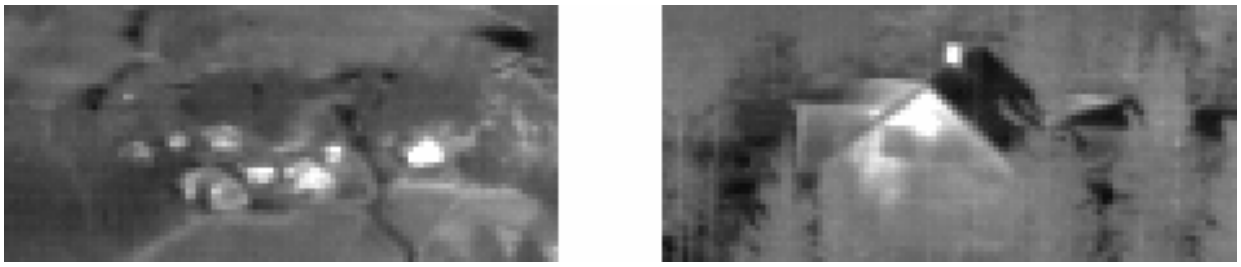
De valgte egenskapene presenteres i dette kapitlet hver for seg i hvert sitt delkapittel. Hver presentasjon er igjen delt i tre deler. Først presenteres eksempler fra testdataene som viser hvordan egenskapen skiller menneskeskapte objekter fra naturen rundt. Det legges vekt på hvordan egenskapen fremtrer og hvilke fordeler og ulemper den kan ha. Deretter vises eksempler på hvordan egenskapen kan opptre i naturlige elementer, som primært ikke bør skilles ut av algoritmen. Dette er viktig for å vise hvorfor bruk av flere egenskaper er viktig. Til slutt gis en enkel pekepinn på hvilken algoritme som kan skille ut egenskapen. Siden det er viktig at egenskapene kan identifiseres innen rimelig kjøretid, antydes det i denne siste delen hvilke algoritmer som er tenkt brukt. Selve algoritmene er utviklet og forklart i detalj i kapittel 4.

3.1 Lyse områder

Det mest iøynefallende ved de menneskeskapte objektene i testdataene er at de fleste har en lysere gråtone enn omgivelsene.

3.1.1 Eksempler på egenskapen hos menneskeskapte objekter

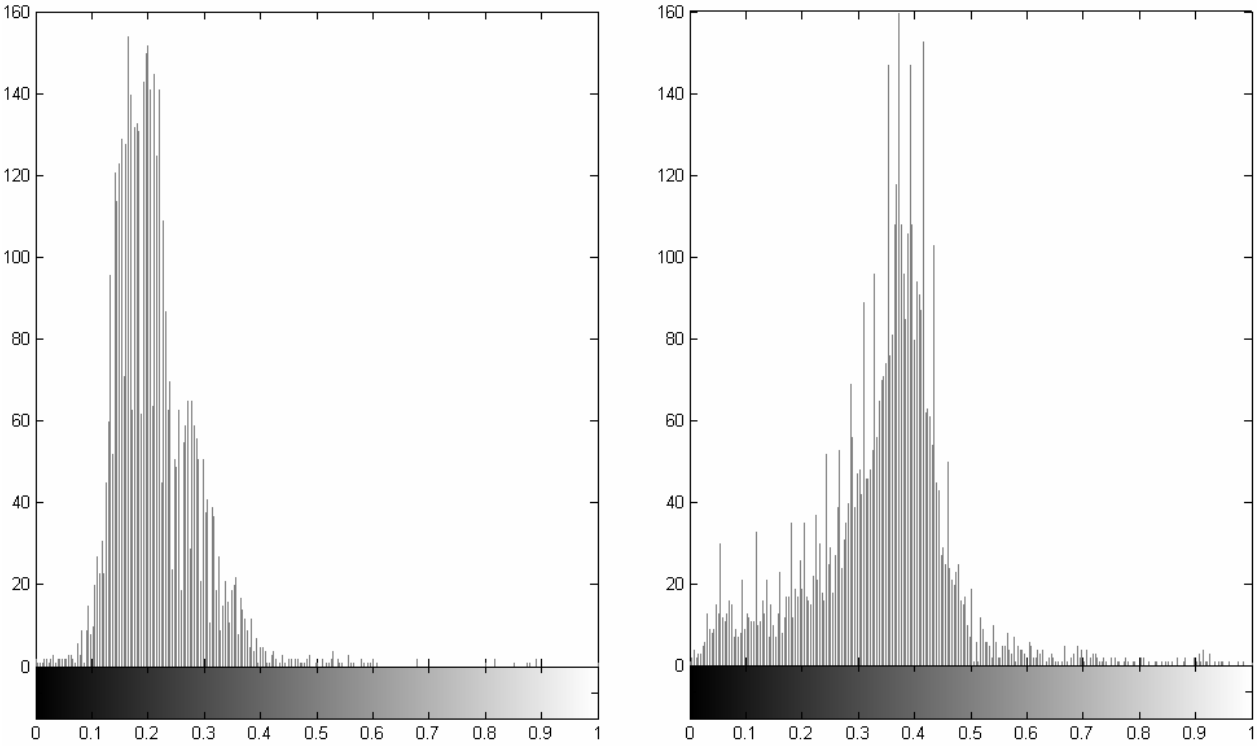
Figur 16 viser to eksempler på bygninger som klart skiller seg ut som lysere enn naturen rundt. Bildet til venstre i figuren viser en klynge hus fra en større avstand, mens bildet til høyre viser overdelen av et hus som stikker opp av omliggende skog.



Figur 16: Lyse bygninger, uansett distanse til kamera

At bygninger skiller seg ut som lysere i IR-bildene er ikke merkverdig. Lysere områder representerer områder som har større varmeutstråling, og mange menneskeskapte objekter varmes lett opp når de står ute i sola, slik som solveggen på disse bygningene.

Selv om de lyse områdene lett skiller seg ut for det blotte øyet kan de være vanskeligere å identifisere automatisk. De lysere områdene inneholder gjerne piksler med verdier over et større spekter, og går ofte glidende over i gråere områder på en eller annen side. Et slikt tilfelle kan sees på bildet til høyre der nærmeste vegg går over i gråere busker nedover veggen. Figur 17 viser histogrammene til bildene i figur 16 (venstre histogram tilsvarer venstre bilde, og motsatt).



Figur 17: Histogrammer til bildene i forrige figur[3]

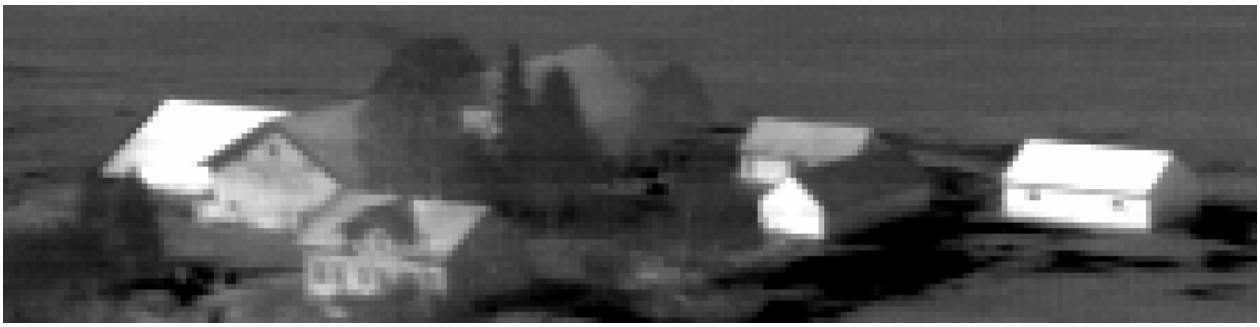
Man kan se at mesteparten av pikslene tilhører en mørk og mellomgrå farge, mens noen få er lysere. Det er ingen klar grense mellom pikslene til objektene og bakgrunnen, men de lyseste pikslene tilhører objektene. Figur 18 viser dette ved å angi plasseringene til pikslene med intensitet over henholdsvis 0,5 og 0,6.

Figur 18: Plassering av de lyseste pikslene

[3] Pikselverdiene er normalisert til mellom 0 og 1 etter laveste og lyseste piksel i de fulle bildene som utklippene i Figur 16 er hentet ifra. Derfor trenger ikke bildet til venstre være mørkere enn bildet til høyre selv om de høyeste søylene i Figur 17 ligger lavere på intensitetsskalaen. Dette kommer av normaliseringen med forskjellig maksimums- og minimumsverdi i de to bildene.

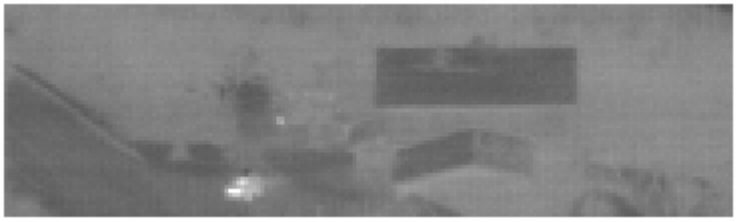
Sammenlignes figur 16 og figur 18 ser man at de lyse pikslene tilhører menneskeskapte objekter.

Figur 19 viser flere eksempler på bygninger som skiller seg ut som lyse. Figuren viser godt noen variasjoner vedrørende denne egenskapen. Først og fremst viser den hvordan forskjellige bygninger kan ha ulik varmeutstråling ved like forhold. Bygningen helt til høyre i bildet er vesentlig lysere enn for eksempel bygningen nærmest kamera. Dette kan komme av at bygningene består av forskjellig materialer, som har forskjellig evne til å varmes opp. Man kan også lett se hva solas oppvarming har noe å si på bygningenes gråtone. Skyggen til bygget nærmest kameraet avslører at sola står nesten rett mot dette byggets kortvegg vendt vekk fra kameraet, med litt stråling inn på langveggen vendt mot kameraet. Forskjellen mellom gråtonen på disse veggene og de mørkere kortveggene kan viser solas innvirkning.



Figur 19: Gårdstun med mer eller mindre lyse bygninger

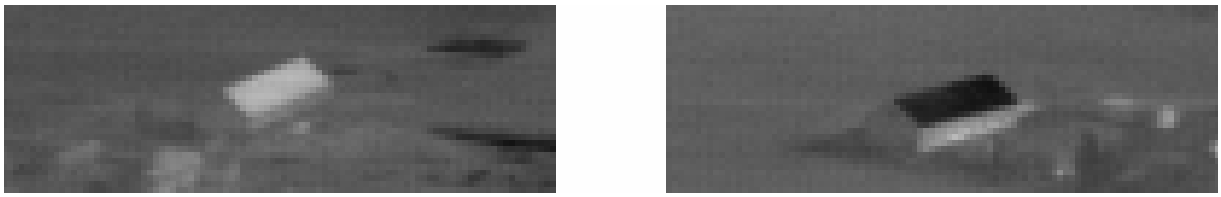
Man kan se at det ikke er stor forskjell i gråtone mellom veggene som er vendt vekk fra sola og bakgrunnen. Figur 20 viser hvordan dette utarter seg for en bygning filmet en overskyet dag.



Figur 20: Bygning fotografert en overskyet dag

Figuren viser at veggene går i ett med bakgrunnen. Slike tilfeller forventes ikke å avsløres som menneskeskapte objekter av denne egenskapen, men viser at egenskapen må kombineres med andre for å skille ut alle menneskeskapte objekter.

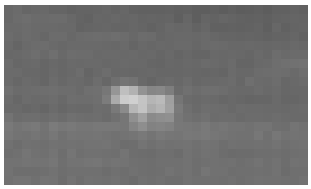
Figur 20 viser også at taket på bygningen skiller seg ut som mørkt, i motsetning til for eksempel bygget lengst til høyre på figur 19. Ut ifra synsvinkelen ovenfra og skrått nedover er det naturlig nok ofte taket av bygningene som utgjør størsteparten av bygget på bildene. Figur 21 illustrerer forskjellen godt, der to bygninger under like forhold viser forskjellige egenskaper vedrørende utstråling fra taket.



Figur 21: Bygninger med lyst og mørkt tak

Som tidligere nevnt reflekterer noen tak innkommende varmestråling godt, uten selv å bli oppvarmet. Med synsvinkelen ovenfra og ned fanges da speilbildet av himmelen, som nesten ikke stråler varme inn mot jorden, om det ikke skulle slumpe til å treffe sola. Derfor blir taket nærmest svart i bildene. Andre tak reflekterer derimot strålingen dårlig, men blir i stedet varmet opp av innkommende varmestråling og fremtrer dermed som varme, lyse arealer på bildene.[4] Ut ifra bildene i figur 21 har antakelig bygningen til venstre takstein, som absorberer mye av den innkomne strålingen, og bygningen til høyre blikktak, som reflekterer mye av den innkomne strålingen. Å trekke ut mørke områder som en egenskap ved menneskeskapte objekter er ikke dermed noen god idé, siden slike områder forekommer veldig ofte i bakgrunnen.

En siste bemerkning ved figur 20 er den lille lyse flekken til venstre for bygningen. Denne virker lys nok til å bli plukket. Dette er en bil, oppvarmet av indre varmeutvikling i motoren. Dette er uten tvil et menneskeskapt objekt, og viser igjen at lyse områder er en god egenskap for å mistenke objekter for å være menneskeskapte. Figur 22 viser derimot et utklipp fra et jorde. Det er vanskelig å se hva det lille lyse objektet er, men det kan muligens være ei ku. Dette viser igjen at egenskapen kan skille ut andre typer objekter enn menneskeskapte. En mulig unngåelse av dette storfeet ville vært å sette en minimumsgrense for hvor mange piksler objektet må strekke seg ut over (antall piksler som må være lyse), men dette ville fort ha ekskludert bilen fra figur 20. Dessuten er det mulig objektet i figur 22 ikke er ei ku, og at det kan være best å inkludere objektet som interessant. Hvorvidt det er ønskelig å bevare litt for mye eller litt for lite bildemateriale vil være avhengig av anvendelsen.

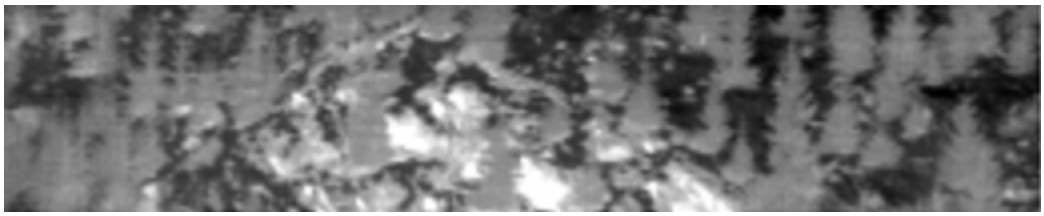


Figur 22: Storfe?

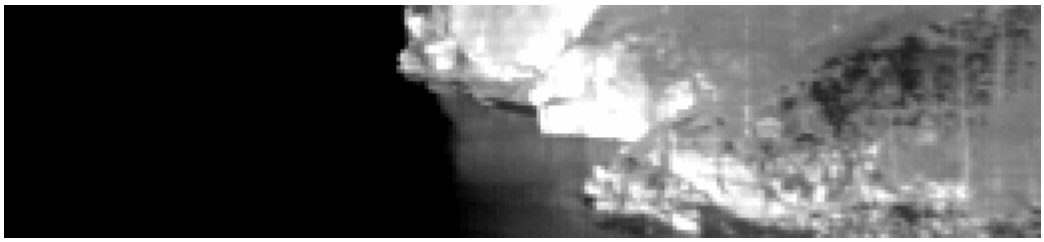
3.1.2 Eksempler på egenskapen i naturen

Lyse områder kan også være klart villedende i form av naturlige elementer som lar seg varme opp av sola. I testdataene er det funnet bergknauser og stein som eksempler på dette. Figur 23 og figur 24 viser hvordan dette kan arte seg i bildene. Figurene viser henholdsvis en fjellknaus som stikker frem inne i skog og bart fjell i en vannkant.

[4] Av termodynamikkens grunnlover øker objekter som absorberer mer stråling enn det emitterer, økte temperatur.



Figur 23: Bergknaus i skog



Figur 24: Fjell ned i vann

3.1.3 Utkast til algoritme

Å skille de lyseste områdene i bildet kalles terskling og foregår ved simpelthen ved å sammenligne piksel for piksel med en terskelverdi og bevare de som er over. Disse kan for eksempel telles for å finne områder som inneholder tilstrekkelig mye lyse områder til å betegnes som interessant.

Fastsettelsen av terskelen kan gjøres på forskjellige måter. Under utviklingen av algoritmen vil det bli testet både en statisk terskel, satt etter prøving og feiling, og en dynamisk terskel som endrer seg underveis. De forskjellige tersklene har forskjellige egenskaper, som blir diskutert under utvikling og testingen av algoritmen i de påfølgende kapitlene.

Selv om alle de viste områdene kan virke mer eller mindre like lyse på figurene, har de forskjellige verdier i forhold til hvor varme de er. Dermed er det mulig å skille noe mer eksakt enn det blotte øye klarer ut ifra disse figurene. Spørsmålet om hvilke verdier man ønsker å karakterisere som interessante må defineres ut ifra algoritmens anvendelse.

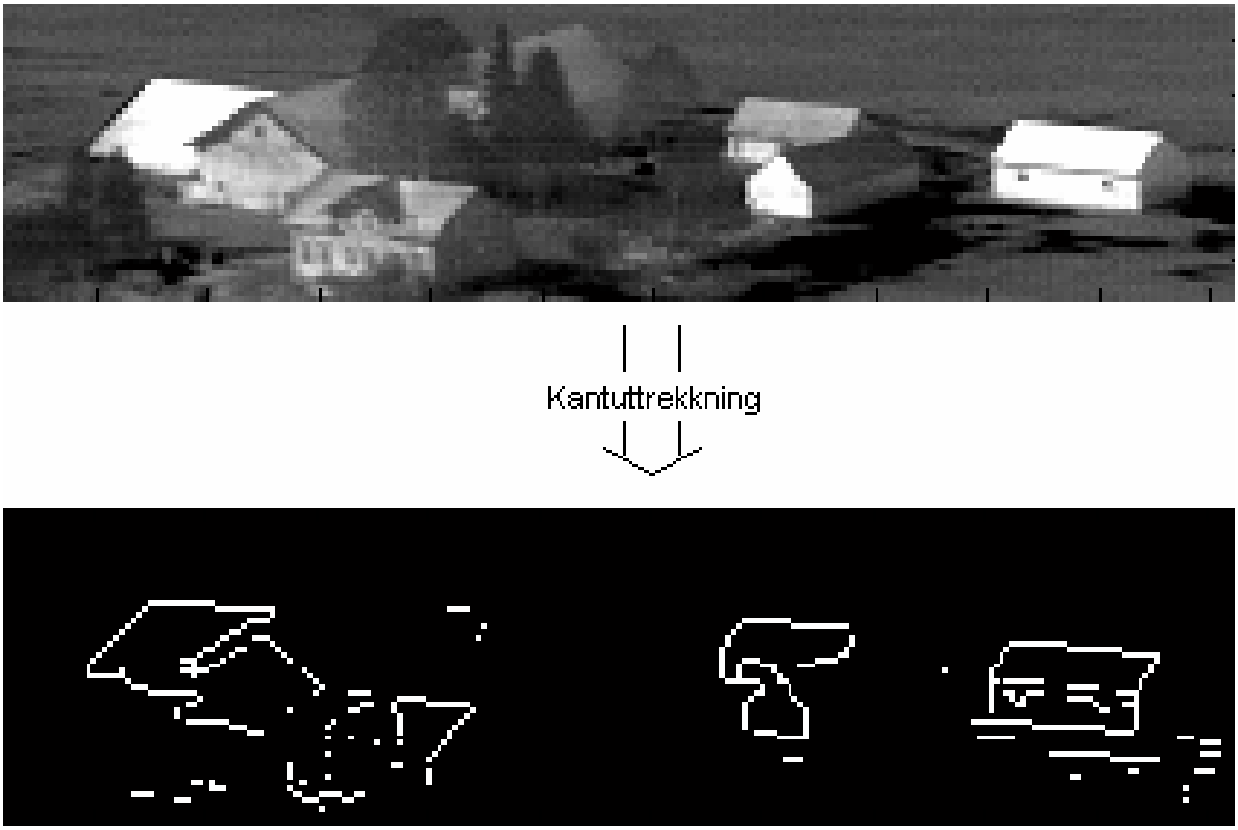
3.2 Markante rette kanter

For å klare å utelukke lyse felter som ikke representerer menneskeskapte objektene må man lete etter egenskaper som skiller de menneskeskapte og de naturlige eksemplene i forrige delkapittel. En egenskap som trer frem i denne sammenheng er kanten mellom de lyse feltene og den mørkere bakgrunnen. Omrisset rundt de lyse menneskeskapte objektene består ofte av rette kanter, slik som møne, vegger og kant mellom vegg og tak. Rette kanter forekommer sjelden i naturlige objekter.

Med en kant menes det en forskjell i verdien til to piksel som ligger ved siden av hverandre. Jo mindre forskjell i verdi, desto svakere kant. Med markante kanter menes kanter som har stor forskjell i gråtone mellom det mørke området på ene siden av kanten og det lyse området på den andre siden.

3.2.1 Eksempler på egenskapen hos menneskeskapte objekter

De fleste bygninger har en silhuett som består av rette kanter. Dette kan lett sees i de figurene fra forrige delkapittel som inneholder bygninger (figur 19 til og med figur 21). For illustrativt å vise hvor de mest markante kantene i figur 19 ligger, viser figur 25 den samme figuren sammen med et kantbilde. Kantbildet er generert ved MATLABs metode *edge*, for kantuttrekning[5].

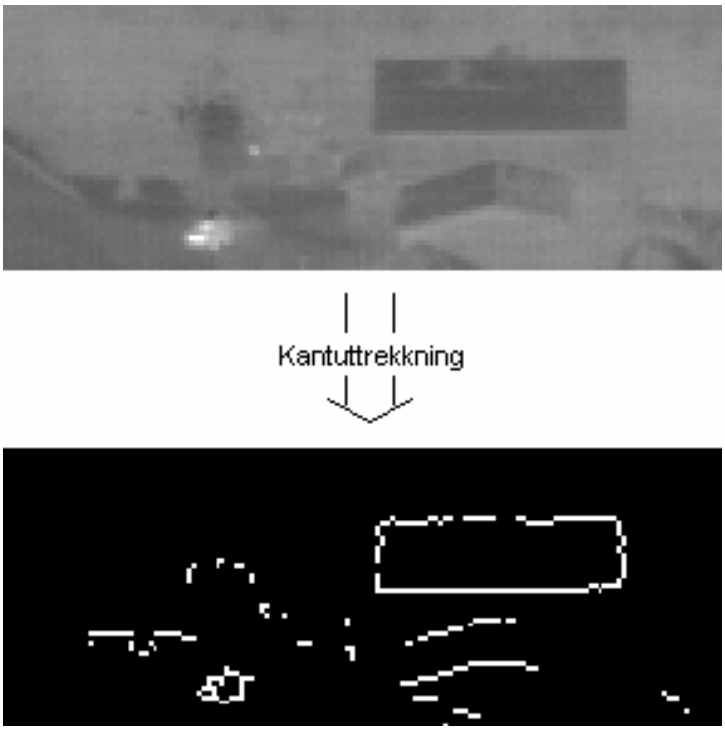


Figur 25: Kantuttrekning av bildet av gårdstun

Figuren viser at de sterkeste kantene i bildet ligger langs rette linjer langs overgangen mellom menneskeskapte objekter og bakgrunnen.[6]

Figur 26 viser videre hvordan metoden også fungerer på mørkt tak, og der sola ikke varmer opp bygget. I dette tilfellet kunne ikke lyse felter avsløre det menneskeskapte objektet, men som figuren viser er det en klar forbindelse mellom de rette kantenes beliggenhet og objektets beliggenhet.

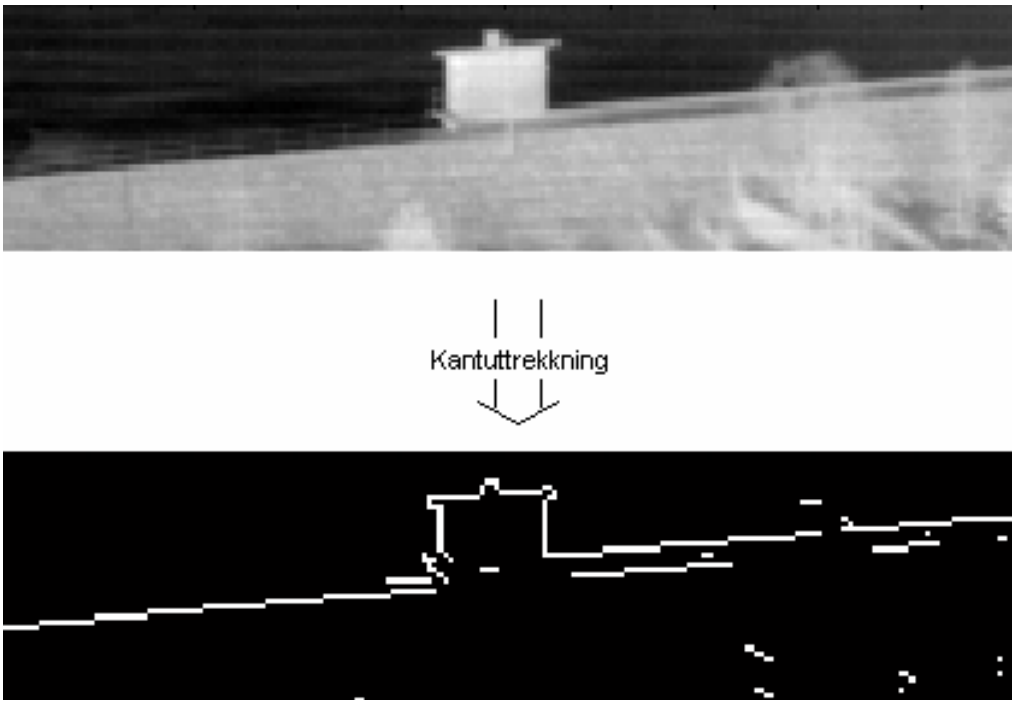
[5] I dette tilfellet brukte metoden sobeloperatoren, siden dette ble spesifisert i parameter til metodekallet. Metoden kalkulerte automatisk en passende terskel for hvilke kanter som skulle være tydelige nok til å inkluderes i kantbildet.
[6] Med linje vil det i denne rapporten konsekvent refereres til en lengde uten bredde, lik den matematiske definisjonen. En kant i kantbildet består derimot av kantpiksler og har dermed minst én piksels bredde. Der kan dermed defineres flere linjer i én kant, noe som er sentralt siden i rapporten.



Figur 26: Kanuttrekning av bygning fotografert en overskyet dag

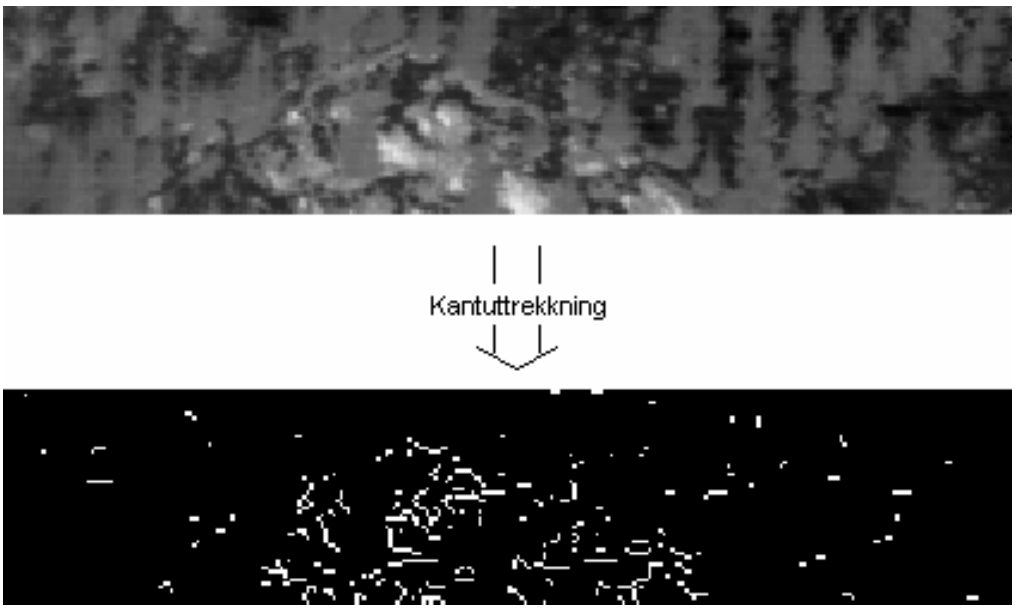
Figuren viser at begge takpartiene avsløres ved de rette kantene, mens det samme ikke kan sies om bilen. Dette kommer av at bilen i bildet er for liten til at dens linjer strekker seg over flere enn et fåtalls piksler, noe som ikke er nok til å anslå hvorvidt linjene faktisk er rette eller ikke. Bilen ble derimot avslørt av sine lyse felter i forrige omtalte egenskap, så disse egenskapene kan sies å være utfyllende. Dette viser seg i høyeste grad på akkurat dette eksempelet (figur 20 og figur 26), hvor de avslører hver sine menneskeskapte objekter.

Demningen fra figur 13 er nok et godt eksempel på hvordan rette linjer avslører menneskeskapte objekter ute i naturen. Figur 27 viser hvordan kantbildet arter seg i dette tilfellet.



Figur 27: Kantuttrekning av demningen

Til sammenligning med disse bildene av menneskeskapte objekter med rette kanter viser figur 28 kantuttrekning av typisk bakgrunn, her representert ved de lyse steinene i figur 23.

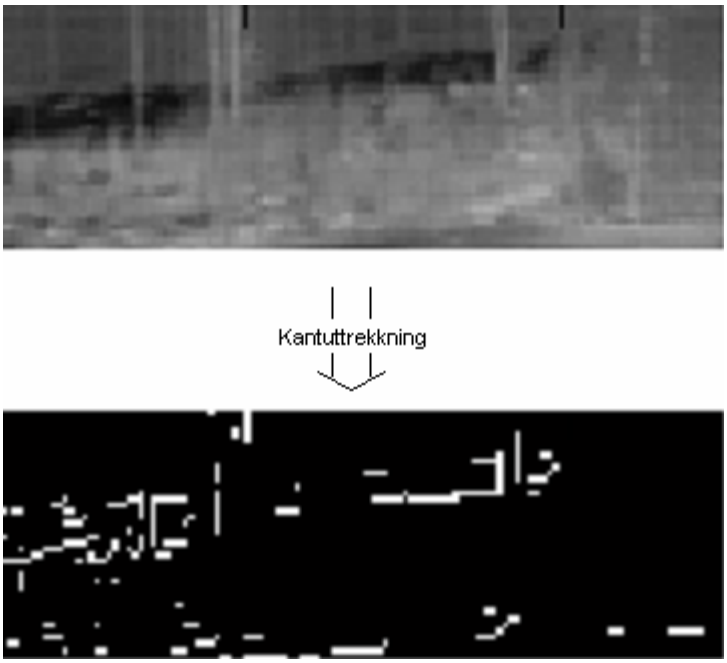


Figur 28: Kantuttrekning av bergknausen

I denne figuren viser godt fraværet av rette kanter. Kantene til disse lyse feltene gjengir kun et spraglete, mer tilfeldig kantbilde.

3.2.2 Eksempler på egenskapen i naturen

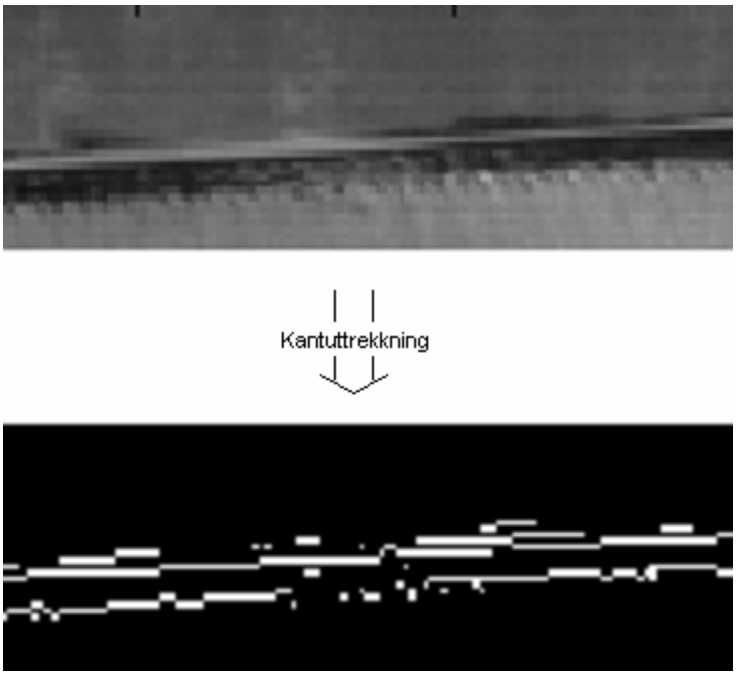
Som nevnt er det sjeldent rette kanter forekommer ute i naturen. Enkelte tilfeller kan dog forekomme. Ett eksempel kan sees i figur 24 der fjellet stikker ned i vannet. I denne skjæringen oppstår det en noenlunde rett kant, i og med at steinen har en såpass flat side mot kameraet. Trær med lange rette stammer uten greiner kan også fremkalle noen rette kanter i bilder, men slike små detaljer er forholdsvis sjeldne. Figur 29 viser hvordan slike trestammer arter seg i kantbilder. Noen spredte vertikale kanter kan finnes, men disse er ikke meget fremtredende i bildene.



Figur 29: Naturlig rette kanter i trestammer

I tillegg til de rent naturlige rette kantene kommer en gruppe kanter som strengt tatt er menneskeskapt, men som i denne sammenheng ikke kvalifiserer som interessante. Et typisk eksempel på slik kant er enden på et jorde. Denne enden er ofte rett, mens både jordet og tilliggende område, ofte skog, kvalifiserer som typiske områder som ønskes fjernet fra bildedataene. Dermed representerer ikke kanten mellom disse områdene tilstedeværelsen av interessante objekter.

Hvis utstrålingen fra jordet og det tilstøtende området, ofte skog, ikke er vesentlig forskjellig, vil det ikke synes noen endring. Ofte ligger det vann, snø eller kanskje en vei langs jordekanten som endrer strålingen, og som legger seg inntil den rette jordekanten. Også trærnes skygge som faller innover jorde kan lage et mørkere område med en rett kant inn mot skogen. Figur 30 viser en jordekant. Området nederst på bildet er et ferdigpløyd jorde (derav rillene). Denne støter inntil et mørkt område som kommer av vann langs jordekanten. Vannet støter inntil en veiskulder, som er relativt rett. På andre siden av veien er det skog.



Figur 30: Kant langs jorde

Bakken, representert ved jordet og veien, samt skogen har noenlunde lik gråtone, mens vannet speiler den lave strålingen fra himmelen, og er mørkere. Dermed får bildet en lang rett kant mellom veien og vannet, samt en ruglete kant mellom vannet og det pløyde jordet.

3.2.3 Utkast til algoritme

Når kantene i bildet skal defineres er det her essensielt at styrken av kanten bevares, siden det er denne som skal være grunnlaget for beregningen av poengsummen til denne egenskapen. Derfor kan en enkel vertikal og horisontal derivering av bildet være egnet.

For videre å finne rette kanter i kantbildet egner Hough-transformasjonen[7] seg. Denne finner hvor mange piksler det finnes i alle mulige kanter i kantbildet. De høyeste verdiene i den resulterende matrisen gjengir de lengste rett linjene i kantbildet. Det vil derfor være naturlig å benytte seg av Hough-transformasjonen i denne sammenhengen. Det blir også her essensielt å bevare styrken av kantene gjennom transformasjonen.

3.3 Parallelle linjer

En annen egenskap ved mange menneskeskapte objekter er at de inneholder en form for parallellogram. Som nevnt i dette kapitlets første delkapittel, utgjør taket på ett bygg ofte store deler av det området bygget opptar i bildet. Dette kommer naturlig av den synsvinkelen en drone vil ha mot bakken. Takene består ofte av rektangulære flater. Når disse blir projisert til bildeflaten utgjør de tilnærmede parallellogrammer. Med forholdsvis liten forskjell i avstand mellom kamera og de forskjellige punktene på rektangelet, gjøres det ikke stor feil i denne tilnærmingen. Dessverre er det vanskelig å detektere parallellogrammer, men parallelle linjer i bildet gir en god indikasjon.

[7] Mer om Hough-transformasjonen kan leses i kapittel 10.2.2 i [b2].

Nå kan det settes spørsmålstegn ved om menneskeskapte objekter med parallelle kanter ikke alt er oppdaget av algoritmen for markante rette kanter, siden et objekt med parallelle kanter nødvendigvis også ha rette kanter. Det er to grunner til at begge er tatt med. For det første gis det her poeng til rette kanter på bakgrunn av hvorvidt det finnes andre kanter som ligger parallelt i stedet for hvorvidt kanten er markant eller ikke. Dermed er det mulig at algoritmene finner menneskeskapte objekter med litt forskjellige egenskaper. For det andre vil det under testingen være interessant å se hvilke av algoritmene som best utmerker menneskeskapte objekter med gode poengsummer. Dersom de konsekvent plukker ut de samme objektene vil det være interessant å vite hvilken algoritme som skiller mest mellom menneskeskapte objekter og bakgrunnen, og som på dette grunnlaget minst sannsynlig vil fortsette å skille ut menneskeskapte objekter i reelle bilder siden. Dermed kan den ene algoritmen forkastes. I en slik avveining vil også resursbruk være aktuelt å vurdere. Kort sagt vil det i denne oppgaven være interessant å se hvordan algoritmene yter, noe som er vist i testkapitlet.

3.3.1 Eksempler på egenskapen hos menneskeskapte objekter

Figur 25 og figur 26 viser bilder av bygninger, samt tilhørende kantbilder. Figurene viser hvordan bygningenes motsatte sider utgjør parallelle linjer i kantbildet. Som eksempel kan bygningen helt til høyre på figur 25 fremheves. Overkanten av bygget på bildet, utgjort av byggets møne, samt underkanten på veggen vendt mot kameraet, utgjør linjer som synes parallelle i kantbildet. Også kanten mellom veggen og taket, midt på bygningen, lager en linje i kantbildet som er tilnærmet parallell med de to nevnte linjene. Linjene utgjøres av hver sine kanter av rektangulære flater i scenen, nærmere bestemt taket og veggen. Ikke alltid vises skillet mellom to slike flater på ett bygg, i og med at de ofte har tilnærmet lik utstråling, og dermed tilnærmet lik gråtone i bildet. Her er det overhenget fra taket over veggen som kaster en skygge, som igjen lager skillet. De to andre sidene på rektanglene, både taket og veggen, utgjør også parallelle linjer, om enn noe kortere.

Kantene etter bygningen på figur 26 bør også bemerkes. I bildet kan taket klart sees som et mørkt rektangel med rette kanter. I kantbildet er overkanten på rektangelet noe ruglete som følge av at skillet ligger mellom to piksellinjer i bildet, og kantuttrekningsmetoden varierer mellom å vurdere hvem av de to linjene som egner seg best til å plassere kanten i. Dermed varierer kanten mellom de to piksellinjene. Noen steder faller linja også bort. Dette er en følge av at bildet er såpass diffust at overgangen noen steder anses som glidende mer enn en kant. Hvor markante overgangene mellom to piksler skal være før det regnes som en kant kan justeres, men settes den for lavt blir for mye av bakgrunnen regnet som kanter. Justering av denne parameteren blir videre diskutert i forbindelse med utviklingen av algoritmene i neste kapittel. Bildet i figur 26 er etter justeringer vurdert som en passende uttrekning av kanter og viser godt hvordan kantbildene ofte ser ut. I tillegg til det store taket er det også et mindre tak rett under. Dette gjenspeiler seg i kantbildet som kun to parallelle linjer og ikke noe parallellogram. Dette er også vanlig i kantbilder av bygninger. Siden de to endene av parallellogrammet utgjør en overgang til bakken og et annet takparti, utgjør ikke skillet i gråtone nok til at det blir betraktet som kanter. Skillet mellom veggen under og taket, samt mellom taket og veggen bak vises derimot som to parallelle kanter i kantbildet. Dette utgjør nok en god grunn til å se etter parallelle linjer fremfor fullstendige parallellogram.

3.3.2 Eksempler på egenskapen i naturen

Som tidligere nevnt forekommer det sjeldent rette linjer ute i naturen. Dermed er det enda sjeldnere det oppstår to parallelle rette linjer. Enkelte tilfeller kan allikevel forkomme. Når det nå presenteres eksempler på denne egenskapen i naturen fra bildene som allerede er brukt i rapporten, betyr ikke dette at denne egenskapen forekommer ofte og er tilfeldigvis å finne i bilder fra tidligere eksempler. Tvert imot er egenskapene sjeldne, og disse bildene plukket med akkurat den hensikt å kunne vise flest mulige egenskaper.

Ser man nøye på kantbildet i figur 25 ser man en ekstra linje under bygningen lengst til høyre. Denne linjen kommer som en følge av mørkere partier på bakken foran bygningen, trolig forårsaket av skygge eller vann. Denne linja ligger omtrent parallelt med linjer i bygget og kan forårsake en antagelse om at er en bygningsflate i bildet mellom skyggen og veggen. Til tross for at dette ikke er riktig vil det ikke føre til noen gal oppførsel av algoritmen som en helhet; Å anta at denne delen av bildet inneholder menneskeskapte objekter vil uansett være sant, så lenge den ene av de parallelle linjene er en del av et menneskeskapt objekt.

De forekomster av denne egenskapen som kan forårsake problemer utgjøres derfor av parallelle linjer der begge linjene forekommer i områder der det ikke er noe menneskeskapte objekter, og dermed ikke skal karakteriseres som interessante områder. Figur 28 viser hvordan naturlige områder kan vise seg i kantbilder. Kantene er korte og mange fremfor rette og lange. Selv om små deler av kanter kan utgjøre små rette linjer som er parallelle, kan disse lett utelukkes ved å sette et krav til minimumslengde på kantene.

Figur 29 viser et vanskeligere problem. Trestammene utgjør vertikale linjer i kantbildet, noe som lett kan settes sammen til par av parallelle linjer. En mulig forbedring av algoritmen vil være å regne et mål på hvor broket arealet mellom de parallelle linjene er. I forbindelse med bygninger har parallellogrammene ofte noenlunde uniform gråtone (noen unntagelser kommer av vinduer og dører på veggene, eller busker som stikker opp foran rektangelet). I tilfellet med trærne vil det være generell bakgrunn mellom linjene, noe som oftest er mye mer broket enn tilfellene med menneskeskapte objekter.

En annen måte parallelle linjer kan opptre i naturen vises i figur 30. De lengre horisontale linjene i kantbildet kommer ikke fra en generell broket bakgrunn, men fra skyggen eller vannet som ligger langs veien mellom skogen og jordet (som diskutert i kapittel 3.2.2). Selv om skyggen/vannets kant mot jordet er noe ruglete som følge av pløyningsrillene gir deler av kanten rette linjer som hver for seg ligger parallelt med kanten mot veien. Disse lange kantene tilhører heller ikke områder som bør karakteriseres som interessante. De fleste menneskeskapte objektene har lengder mindre enn lengden av vei eller jordekant, men å sette en grense for maksimumslengden på en kant skal vise seg å være vanskeligere enn å sette en minimumsgrense. Begge deler er forsøkt under utvikling av algoritmen i neste kapittel.

3.3.3 Utkast til algoritme

Hough-transformasjonen er en generell metode for å finne hvor i kantbilder geometriske figurer befinner seg, og kunne vært brukt for å finne parallellogrammer. Dessverre har et parallellogram for mange frihetsgrader; Hele seks. Disse kan for eksempel betegnes som lengden på de to

sidene, vinkelen mellom dem, plassering i x og y retning, samt rotasjon. Dette vil gi en altfor stor kjøretid på Hough-transformasjonen til å brukes i denne sammenheng.

Linjene hver for seg er mye enklere å bestemme. Ei linje har to frihetsgrader som kan betegnes som vinkelen mellom linjen og én akse, samt linjens skjæringspunkt med aksen. Siden det alt er beregnet Hough-transformasjonen av kantbildet (se forrige egenskap; Markante rette linjer), og denne inneholder vinkelen mellom en akse og de mest markante linjene i bildet, var ideen å utnytte allerede utregnede egenskaper. Ved å sammenligne vinklene kunne det fastslåes hvilke linjer som var parallelle. Ved å sammenligne andre egenskaper, som avstanden mellom linjene, kunne mistanken om parallellogram ytterligere testes.

3.4 Uniforme arealer ved rette linjer

Områdene som utgjør menneskeskapte objekter har ofte en uniform gråtone. Dette gjelder spesielt i forhold til bakgrunn av vegetasjon, som er mer broket i sin fremtoning i gråtonebildene. Ett vanlig eksempel kan finnes i figur 7, som viser skog i form av relativt lyse trær med sprekker ned mot en relativt mørk skogbunn. Dessverre kan man ikke beregne variasjonen i gråtoneverdiene til pikslene knyttet til det menneskeskapte objektet i og med at man ikke vet hvilke piksler som tilhører sådanne. I behandlingen av rette kanter i forbindelse med de andre egenskapene burde pikslene på minst en av sidene av kanten tilhøre et menneskeskapt objekt. Hvis ikke dette er tilfelle burde ikke området kanten befinner seg i karakteriseres som interessant. Dermed burde variasjonen av pikslene på minst én av sidene av kanten være liten.

Siden egenskapen skal vise seg å ha en del tvilsomme tilfeller der menneskeskapte objekter ikke alltid er vesentlig mer uniforme enn bakgrunnen, er egenskapen først og fremst ment som en siste egenskap for å tippe i den ene eller den annen retning i tvilstilfeller. Hvorvidt den egner seg til dette vil fremgå av testingen av algoritmen i kapittel 5.

3.4.1 Eksempler på egenskapen hos menneskeskapte objekter

Denne egenskapen viser seg først og fremst blant de menneskeskapte objektene som er bygninger. Siden dette utgjør de fleste av de menneskeskapte objektene som kan sees i testdataene, kan dette vise seg å være en egenskap det er verdt å bruke. Bygninger med relativt uniforme gråtoneverdier i forhold til bakgrunnen kan sees i de fleste figurene av menneskeskapte objekter brukt hittil i rapporten. Som eksempler kan det trekkes frem taket på bygningen øverst til høyre på figur 10, områdene rundt kantene på det menneskeskapte objektet i figur 12 og bygningen lengst til høyre på figur 19.

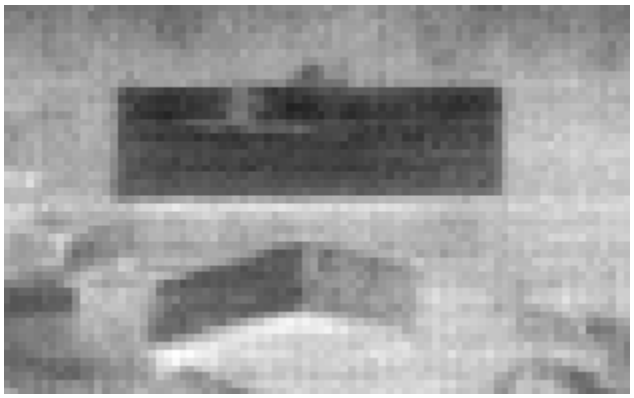
Vel så viktig kan det være å se på mangelen av denne egenskapen i tilfellene der det finnes rette linjer ute i naturen. Figur 29 viser at trestammene er relativt tynne på bildet, og pikslene på hver side av stammen varierer stort i gråtone. Videre kan man se på linjene i kantbildet i figur 30. Der er også gråtonene på begge sider av linjene relativt varierte i forhold til områdene på bygningene nevnt i avsnittet over. Både i skyggen/vannet, på jordet og i skogen i figur 30 overgår de nevnte bygningene angående variasjon i gråtoneverdier.

3.4.2 Eksempler på egenskapen i naturen

Selv om bakgrunnen generelt har større variasjon i gråtoneverdier finnes det enkelte problemer. Jo større avstand det er mellom kamera og objektet, desto mindre detaljer klarer kameraet å fange

opp. Dette medfører igjen at flater oppstår mindre spraglete jo større høyde bildene er tatt ifra. Dermed kan jorder, tett skog og andre naturlige områder fotografert fra store høyder fremstå vel så uniforme som overflater på bygninger fotografert på nært hold.

Andre bilder er så diffuse at den spraglete bakgrunnen kan fremstå som like uniform som bygninger. Figur 20 viser dette. Fenomenet er noe forsterket i dette bildet, siden den meget varme bilen strekker spekteret av strålingsintensitet i bildet. Når dette skal konverteres til 256 gråtoneverdier, som er tilfelle i disse bildene, blir det mindre verdier til å bruke på de andre strålingsintensitetene. Figur 31 viser det samme bildet, der bilen er utelatt og de gjenværende strålingsintensitetene er strukket over 256 gråtoneverdier. Også her synes det at bakgrunnen fremstår som noenlunde like uniform som bygningsflatene.



Figur 31: Bygning fotografert en overskyet dag

Det bør også nevnes at langt fra alle bygningene har uniforme områder ved kantene. Som eksempel på bygningsflate kan muren i demningen på figur 13 trekkes frem. Siden steinene i muren er varmere og dermed lysere enn mellomrommet mellom dem, virker hele muren broket, om enn i regelmessig mønster.

3.4.3 Utkast til algoritme

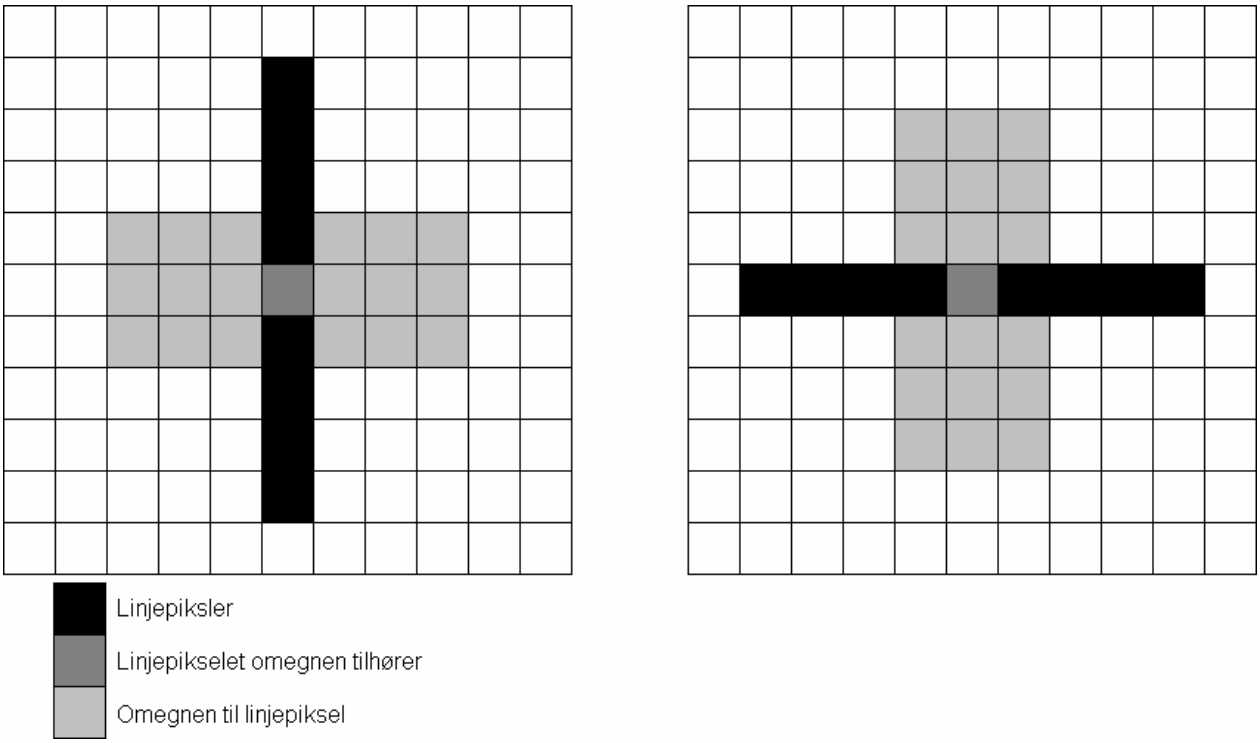
For å sette et mål på variasjonen i pikselverdier finnes det flere muligheter. Fouriertransformasjonen gir oversikt over variasjonen som funksjon av frekvensen i variasjonene. Hvis pikselverdiene endrer seg gradvis over større områder viser Fouriertransformasjonen større innflytelse fra store frekvenser. Jo fortere pikselverdiene endrer seg over et like stort område, desto lavere frekvenser utmerker seg i Fouriertransformasjonen[8]. På grunn av at det her er snakk om forholdsvis små områder å kalkulere variasjonen på (noen titalls piksler er vanlig skal det vise seg), blir det få frekvenser som kan gjøre seg gjeldende, og Fouriertransformasjonen vil være uegnet. Derimot er det naturlig å bruke varians. I tillegg til å være matematisk enkel og liten beregningstid, finnes det i MATLAB en egen metode for å beregne variansen, noe som gjøre den enkel å benytte her. Ett tredje alternativ ville være å bruke Wavelets[9]. Denne representerer en avveining mellom romslig plassering i bildet og variasjon i verdiene som funksjon av frekvensen. Denne er mer matematisk komplisert og har en større beregningstid. Derfor er det valgt å bruke

[8] Mer om Fouriertransformasjonen kan leses i kapittel 4 i [b2].
[9] [w10] inneholder en enkel og strukturert forklaring av Wavelets.

varians her. Tilsvarende forsøk med Wavelets for sammenligning hadde vært interessant, men er ikke gjort i denne oppgaven.

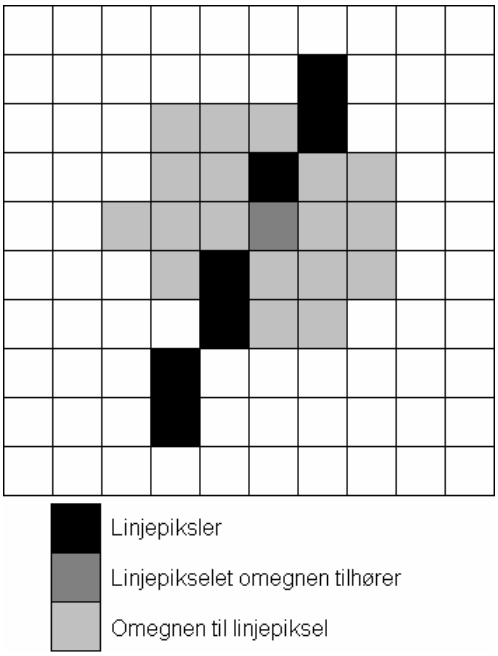
Siden det ikke er gitt hvilken side av kanten som tilhører det menneskeskapte objektet, må det beregnes for begge sider. Den siden med minst varians tilhører trolig det menneskeskapte objektet. Ut ifra variansen fra denne siden kan en poengsum genereres ved hjelp av en funksjon som favoriserer lave varianser, og dermed lave variasjoner.

Å plukke ut pikselverdier til beregningen er ikke trivielt. Det er naturlig å benytte ett areal på siden av kanten. Så lenge kanten er vertikal eller horisontal kan det lett defineres et areal ved å ta *N* x *M* piksler, slik som foreslått i figur 32.



Figur 32: Omegn rundt linjepiksler fra rett linje

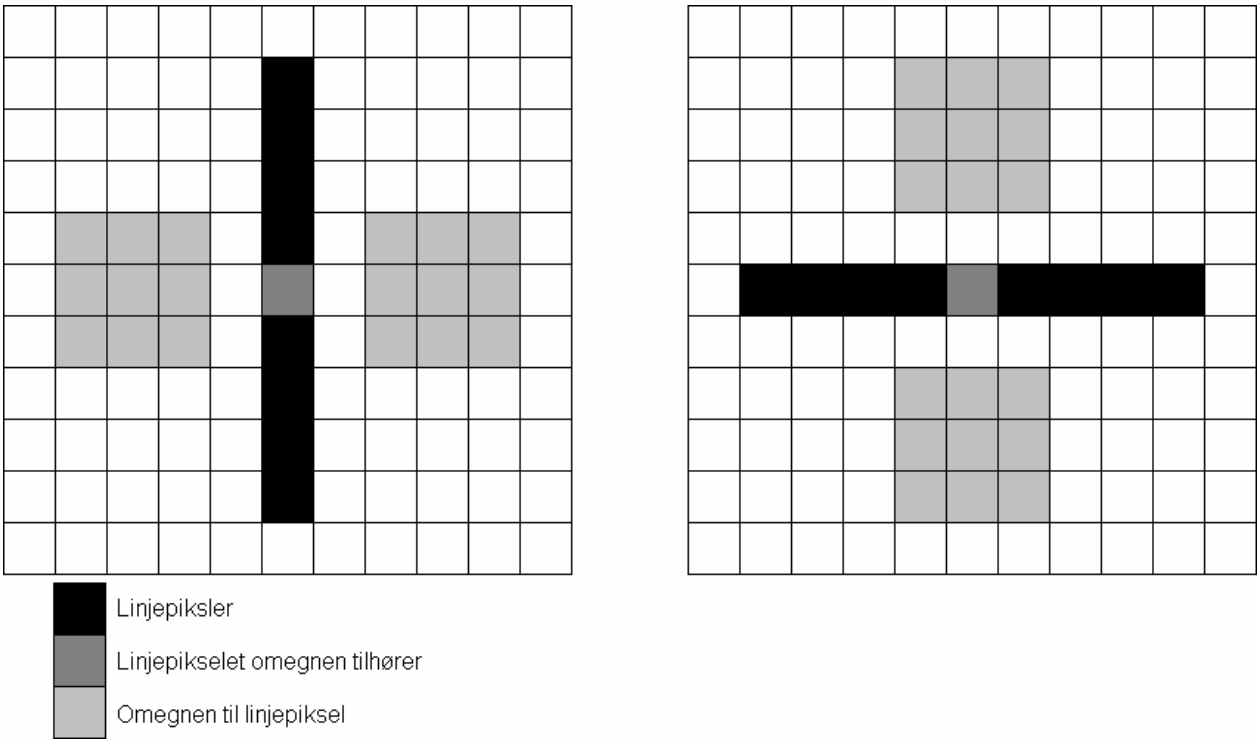
I tilfeller der kanten ikke går i en av disse retningene må det flere beregninger til for å angi en mengde piksler som godt representerer den ene siden av linjen, slik figur 33 foreslår (ett forslag basert på like stort areal som i figur 32, dannet av de nærmeste pikslene på hver side).



Figur 33: Omegn rundt linjepiksler fra skrå linje

Dette ansees som unødvendig nøyaktig for denne sammenheng. Derfor kan det ganske enkelt kun beregnes vertikale og horisontale kanter (enkelt derivert i de to retninger) for så å benytte den enkle firkantede pikselmengden foreslått i figur 32. Linjer som ikke går rett vertikalt eller rett horisontalt vil fremgå som kombinasjoner av slike i kantbildet.

I denne oppgaven er det valgt å bruke et område litt vekk fra kanten, slik som vist i figur 34. Dette er gjort for å minke risikoen for å inkludere kantpiksler i beregningen av variansen. Skjeve kanter har her mindre mulighet for komme inn i området brukt til variansberegningen. Hvis kanten skulle opptre noe diffust i bildet, vil også dette ha mindre innvirkning på variansberegningen når området er flyttet litt vekk fra kanten.



Figur 34: Arealer godt innenfor kanten

I figur 32 og figur 34 er det foreslått et areal på 3*3 piksler, men dette kan selvfølgelig endres. Dette må sees i sammenheng med hvor store objektene er, slik at arealet ikke krysser en ny kant på andre siden av det menneskeskapte objektet.

Kapittel 4: Algoritmer

I dette kapitlet forklares den utviklede algoritme i detalj. Kapitlet er ment å gi innsikt i hvordan algoritmen deler bilder i interessante og uinteressante områder. Først beskrives algoritmen overordnet, før sentrale metodekall gjennomgåes i hvert sitt delkapittel. Med de sentrale metodekallene regnes blant annet de metodene som kalkulerer egenskaper.

En viktig parameter for valget av algoritme har vært resurskravet. For at algoritmen skal kunne brukes om bord på droner må den kunne prosessere bilder i sann tid. Dette har det vært tatt hensyn til gjennom utviklingen av algoritmen. Kjøretidskompleksiteten er ikke målt[10], men algoritmene er tenkt å kunne kjøre i sann tid gitt effektivisering av koden i form av omskrivning til et raskere programmeringsspråk og noen omformuleringer foreslått utover i kapitlet. Grunnen til at disse omformuleringene ikke er tatt med denne gangen har vært for å gjøre koden så tilpasset testing som mulig, siden det er det den her skal brukes til.

Det har under utviklingen vært forsøkt å gjøre algoritmene generelle med tanke på at de skal kunne brukes i systemer med forskjellige sensorer og lignende. Dette har vært en avveining mot å gjøre for mange parametere avhengige av å settes av operatør. Det er ønskelig at systemet skal være lettest mulig å betjene. Dette vil si at det inneholder færrest mulige parametere som operatøren må finjustere for hvert system. Spesielt er det parametere som må settes i samme skala som pikselverdiene. Disse avhenger av hvor fin inndeling sensorene kan skille mellom, og hvor godt denne nøyaktigheten er bevart i eventuelle formateringer av verdiene i senere kalkulasjoner og konverteringer. Siden det ikke bare er innedlingen, men også spekteret av registrert stråling som varierer med sensorene, er det ikke mulig å angi parametrene som prosentvis innen det registrerte spekteret. Derfor finnes det noen slike parametere som må settes av operatør. Det er ikke meningen disse må justeres for hvert opptak algoritmen brukes på, men for hvert system av sensorer det brukes med. Siden alle opptakene i testdataene benyttet her er samlet med det samme systemet, vil disse parametrene ha den samme verdien gjennom alle testene. Verdien på parametrene er ikke oppgitt siden de ligger i samme spekteret som pikselverdiene, som KDA ikke kan publisere. Parametrene er derimot utførlig forklart, slik at det skal være mulig å forstå hvilken verdi som vil være passelig til forskjellige systemer.

MATLAB[11] er brukt som utviklingsmiljø. Det er flere grunner til det. For det første er programmeringsspråket MATLAB lettforståelig for de fleste som har drevet med imperative programmeringsspråk eller objektorienterte språk som ligner[12]. Dette gjør den vedlagte kildekoden ved denne rapporten leselig for mange, og fungerer som pseudokode for de forskjellige algoritmene i dette kapitlet. For det andre har MATLAB en svært enkel håndtering av bilder, med innebygde konverteringer til enkle tallmatriser, metoder for visning av bilde på skjerm og lignende. I tillegg hadde jeg tilgang til ekstra bildebehandlingsfunksjoner gjennom tilleggspakken ”Image Processing”. All den innebygde funksjonaliteten gjorde utviklingen raskere og mer konsentrert rundt oppgaven. For det tredje hadde KDA brukt dette miljøet før, og

[10] Hovedgrunnen til at kjøretidskompleksiteten ikke er målt er fordi den har så mange parametere i tillegg til bildestørrelsen har stor påvirkning på kjøretiden.
[11] Mer spesifikt MATLAB & SIMULINK studentversjon, utgave 14 med service-pakke 1. For mer informasjon om MATLAB, se www.mathworks.com
[12] Som for eksempel C, C++ eller Java.

kunne låne meg rammeverk for lesing av testdataene (både bildeseriene og de tilhørende navigasjonsdataene), som har ett særegent format. Språket er ikke særlig raskt i logiske løkker. Dette gjør en del av algoritmene utviklet her relativt langsomme. Med tanke på hastigheten til algoritmene burde de nok vært implementert i et annet språk (for eksempel *C++*), men for å vise at og hvordan algoritmen virker er MATLAB et passende språk.

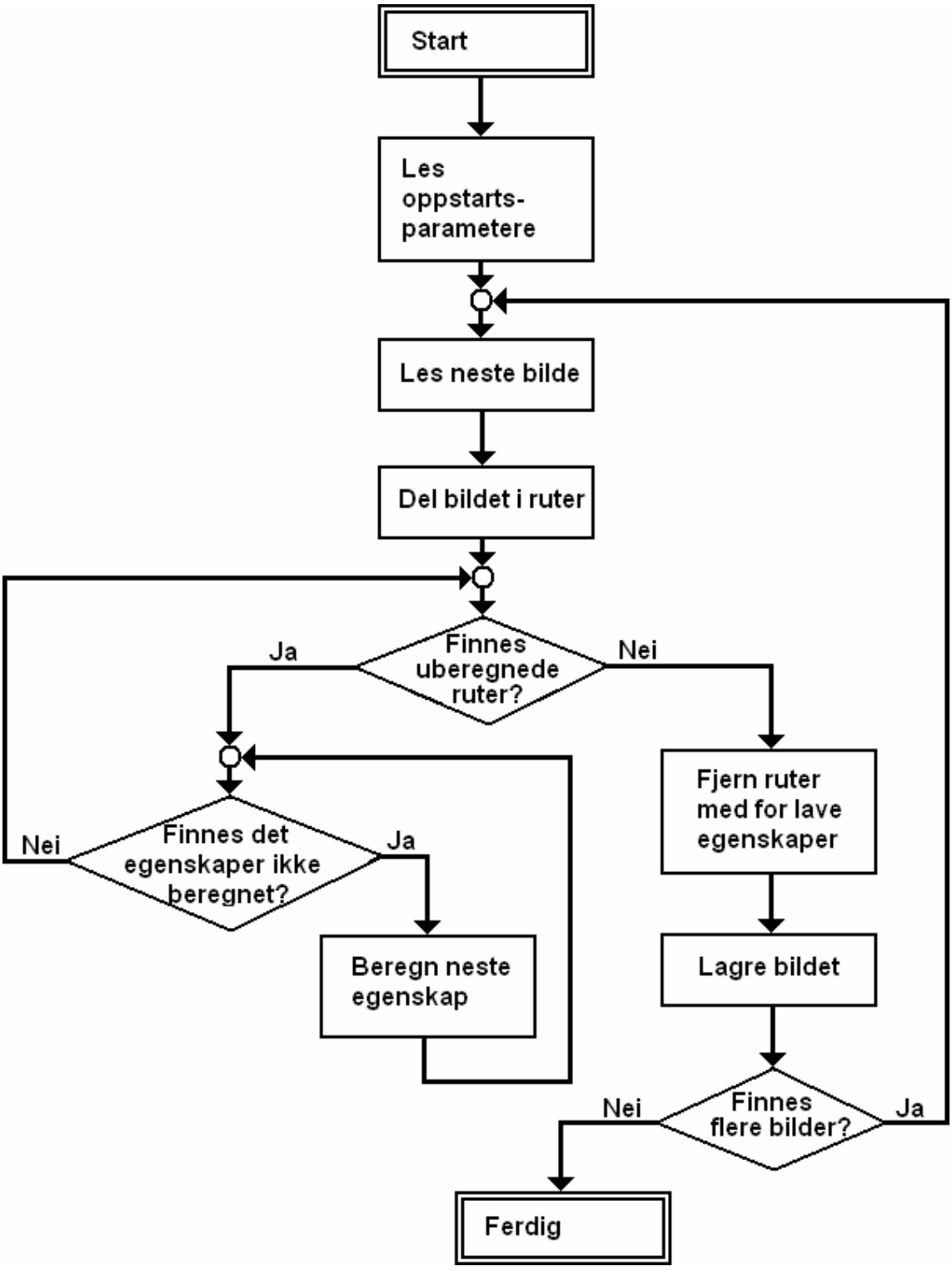
I dette kapitlet er det nevnt noen funksjoner som er hentet fra MATLAB sitt funksjonsbibliotek. For mer informasjon om disse henvises det til MATLAB sin egen dokumentasjon.

Utskrift av kildekoden ligger vedlagt dette dokument i vedlegg A. Her finnes all kode som er omtalt og brukt for å fremskaffe viste resultater. Når det refereres til kodefiler ved navn kan disse finnes ved å slå opp i vedlegget. Noen generelle kommentarer til kodefilene finnes aller først i vedlegget. Før sentrale algoritmer er det referert til hvor i koden den aktuelle algoritmen kan finnes, slik at de som ønsker å lese pseudokode kan slå opp her. Dermed er heller ikke forklaringen i dette kapitlet brutt opp av bolker med slik kode.

4.1 Overblikk over algoritmen

Skriptet som utgjør programmet som trekker ut interessante områder fra testdataene ligger i filen *Main*. Alle de andre filene er metoder som kalles, direkte eller indirekte, fra dette skriptet.

Innholdet i skriptet gjenspeiler valget av egenskapsidentifisering som algoritmemetode. Figur 35 viser kontrollflyten.



Figur 35: Kontrollflytdiagram

Først leses det inn en del parametere som ikke beregnes men fastsettes av operatør for hver anvendelse. Ett eksempel er lokasjonen til testdata på disken.

Deretter starter en større løkke gjennom alle bildene, slik vist i figur 35. I kronologisk rekkefølge lastes hvert bilde med tilhørende navigasjonsdata. Navigasjonsdataene inneholder posisjonen til kameraet i det bildet ble tatt og brukes i oppdelingen av bildet i mindre ruter. Selve innlesningen av dataene vil være svært forskjellige for hver anvendelse av algoritmen. I ett sanntidssystem vil typisk bildedataene komme rett ifra opptaksenheten mens navigasjonsdataene kommer fortløpende fra dronens navigeringsenhet. Testdataene brukt i denne rapporten er allerede lagt på disk, og leses derfra. For andre anvendelser av algoritmen vil denne delen gjerne byttes ut, og legges ikke vekt på i denne rapporten.[13] Den aktuelle koden er ikke utskilt i egne moduler som lett kan byttes ut, siden det er antatt at all koden vil bli skrevet på nytt for tilpassning og optimalisering ved anvendelse i et reelt system.

Neste oppgave er å dele bildet opp i mindre ruter. I denne rapporten vil det med ruter konsekvent henvises til disse bestanddelene som bildet deles opp i. Hoveddelen av dette består i å beregne størrelsen på rutene. Dette beskrives nærmere i delkapittel 4.2.

Deretter følger en løkke over alle rutene i bildet, slik figur 35 viser. Før løkken starter lages det en poengmatrise med en rad for hver egenskap og like mange kolonner som ruter i bildet. I denne matrisen lagres poengsummene for hver egenskap til hver rute. Jo mer fremtredende egenskap, desto større poengsum. Deretter beregnes alle egenskapene. Siden det i denne oppgaven er testet to forskjellige metoder for å identifisere lyse områder, statisk og dynamisk terskling, er det fem egenskapsberegninger for hver rute:

1. Lyse områder, beregnet med statisk terskel (Denne bolken er videre forklart i delkapittel 4.3)
2. Lyse områder, beregnet med dynamisk terskel (Denne bolken er videre forklart i delkapittel 4.4)
3. Markante linjer (Denne bolken er videre forklart i delkapittel 4.5)
4. Parallelle linjer (Denne bolken er videre forklart i delkapittel 4.6)
5. Uniforme arealer ved rette linjer (Denne bolken er videre forklart i delkapittel 4.7)

Resultatet av beregningene legges i poengmatrisen, og løkken over rutene avsluttes når alle egenskapenes fremtreden i alle rutene er beregnet (igjen, se figur 35).

Poengmatrisen inneholder nå informasjon om egenskapene i de forskjellige rutene av bildet, og brukes til å avgjøre hvilke ruter som skal forkastes som uinteressante. Det er forskjellige metoder for å avgjøre hvilke ruter som skal karakteriseres som interessante på bakgrunn av poengene. I kapittel 4.8 forklares forskjellige fremgangsmetoder med fordeler og ulemper. Her vil det også bli valgt en passende metode for de gitte testdata.

4.2 Oppdeling av bildet

For å kunne utelate enkelte områder av bildene må bildet deles opp i mindre enheter som klassifiseres som enten interessante nok til å bevare eller ikke. Dette delkapitlet er delt opp i to deler. Den første begrunner valget av størrelsen på rutene og hvilke konsekvenser valget vil

[13] I koden er innlesning og konvertering av data gjort ved metoder gitt av KDA. Disse er tilpasset rådataenes sære format og ikke lagt ved denne rapporten.

kunne føre til. Den andre delen forklarer hvordan algoritmen som beregner denne størrelsen virker.

4.2.1 Valg av rutestørrelse

Størrelsen og plasseringen av disse rutene vil kunne ha stor innflytelse på resultatet i og med at de definerer hvilke piksler som blir behandlet sammen som et kandidatområde, og dermed hvilke enheter som potensielt kan fjernes fra bildet. Små ruter vil gi muligheten til å forkaste større områder enn store ruter, siden alle ruter som inneholder hele eller deler av menneskeskapte objekter bør bevares. På den annen side vil flere ruter for enkelte egenskaper gi mer beregninger og dermed tregere algoritme. Dette gjelder egenskaper som krever beregninger som gjøres på hver rute, men som ikke effektiviseres vesentlig av at rutene blir mindre. Det er også en fordel å ha rutene store nok til å omfavne hele de menneskeskapte objektene. Få egenskaper ved disse objektene kan tilegnes alle pikslene som utgjør objektet i bildet. For eksempel kan det være lett å avdekke et varmt vindu på en vegg eller kanten langs et møne, mens hele bygningen ønskes bevart i bildet. Dermed kan det være en fordel å omfavne området rundt den avdekkede egenskapen i den hensikt å få med hele objektet. Ett annet betegnende eksempel er avdekningen av parallelle linjer i en rute. Her er det åpenbart vesentlig at begge linjene ligger i samme rute, og siden linjene gjerne ligger på motsatt side av objektet (for eksempel motstående vegger på en bygning) er det viktig å omfavne hele objektet i samme rute. I tillegg vil det som regel være fordelaktig for en operatør som skal se på dataene siden den nærliggende omegnen rundt objektet er bevart. Dette gjør objektene lettere å identifisere. Ett eksempel er bilen i figur 20, som ville vært vanskelig å gjenkjenne som en bil uten de nærmeste områdene rundt. Med veien og innkjørselen som referanse kan man ane størrelsen på objektet samt dets anvendelse (den brukes på veien), noe som gjør det lettere å gjenkjenne objektet som en bil.

Det er altså ønskelig med så små ruter som mulig, men store nok til å omfavne alle menneskeskapte objekter. For at rutene skal dekke de menneskeskapte objektene, som kan dukke opp hvor som helst i bildet, er det også viktig at rutene blir plassert slik at ikke objektene kan havne i skillet mellom to ruter. Dette betyr at det må være en viss overlapp mellom rutene.

For å holde rutene så små som mulig ville det være optimalt å holde størrelsen på rutene til et estimat av det største menneskeskapte objektet man kan tenke seg å finne i bildene. Denne ruten vil formodentlig kunne dekke alle menneskeskapte objekter som dukker opp i bildene. Men hvis ruten ikke mer enn akkurat dekker objektet må den plasseres på alle mulige punkter i bildet for å sikre at ethvert objekt fullstendig faller innenfor ruten. En slik rute kalles en glidende rute og flyttes rundt omkring i hele bildet, en piksel av gangen. Dette gir veldig stort krav til beregninger, som skal foretas for hver plassering av ruten. Dette gjør det ønskelig å minke antall posisjoner ruten kan anta i bildet. En reduksjon av dette vil gå på bekostning av størrelsen av ruten, hvis man fortsatt ønsker å være sikker på at ethvert objekt av den antatte største mulige størrelse fortsatt skal falle innenfor ruten.

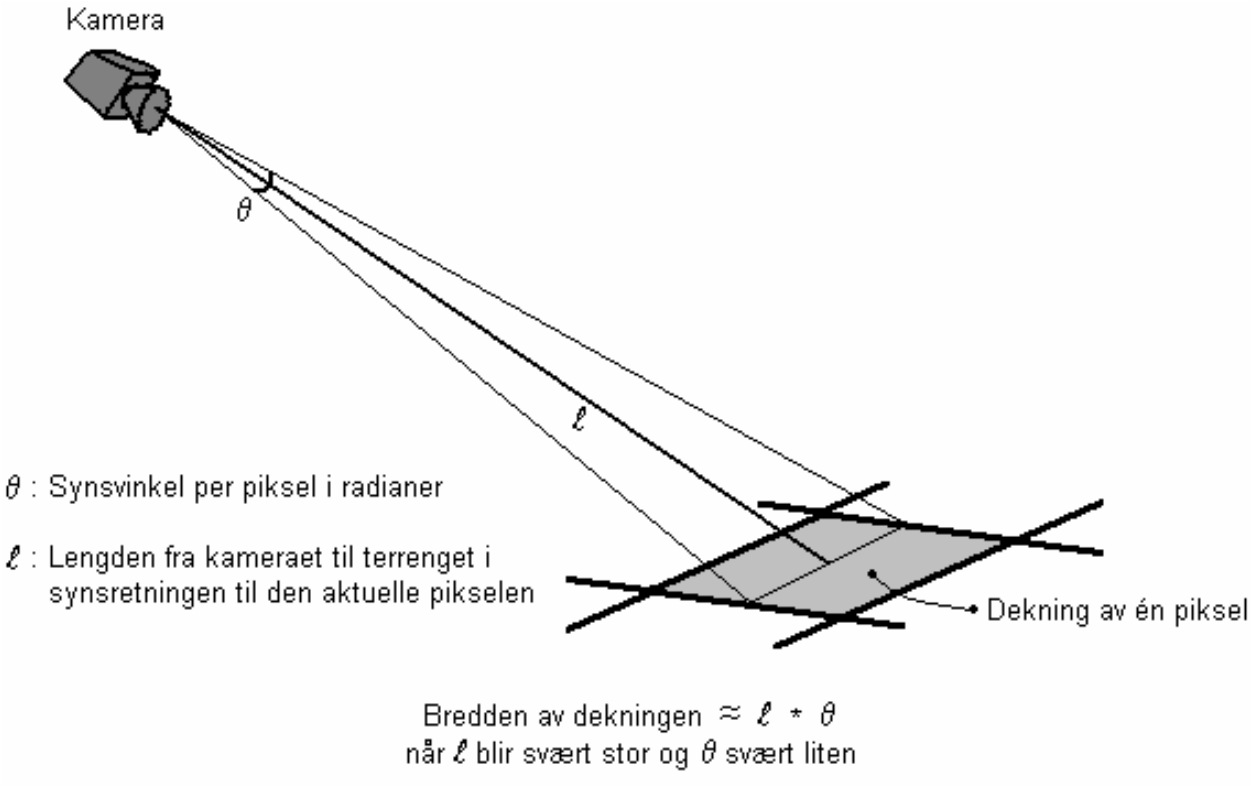
I denne oppgaven er det valgt en rute dobbelt så stor som det største menneskeskapte objektet man forventer å kunne oppdage. Rutene er plassert med 50 % overlapp, noe som gjør at alle punkter i bildet ligger innenfor akkurat to ruter. Denne plasseringen av ruter vil gjøre at alle objekter som ikke er større enn den antatte maksimale størrelsen, og som ligger kun delvis innenfor en rute vil i sin helhet ligge innenfor den neste ruten. Siden testbildene brukt i denne

rapporten er vesentlig bredere enn de er høye er det valgt å kun dele bildene opp i bredden. Det vil si at det kun er en rute i høyden, og hver rute har samme høyde som bildet. Algoritmen viser allikevel prinsippet for oppdelingen og denne kan godt utvides til å gjelde både i høyde og bredde der dette passer med dimensjonene på bildene.

4.2.2 Algoritmen for beregning av rutestørrelsen

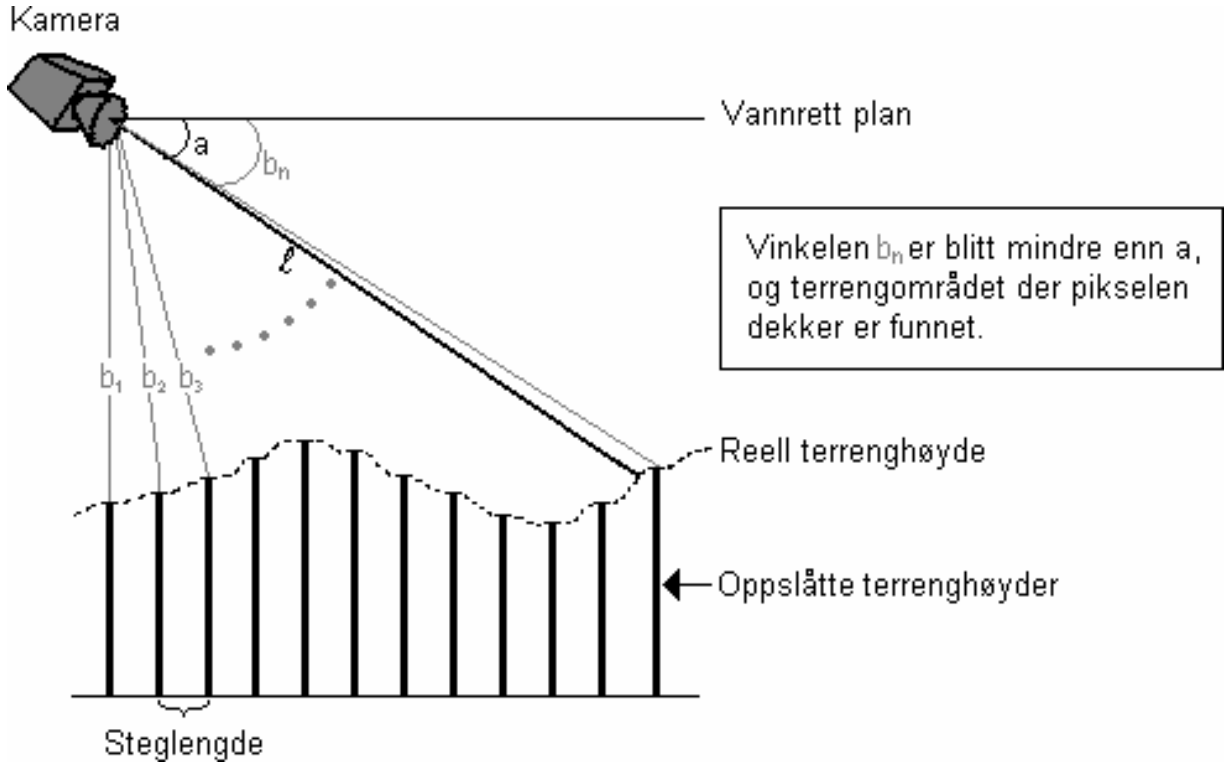
Siden den maksimale størrelsen man forventer å finne på menneskeskapte objekter vil tilsvare forskjellige størrelser i bildene alt etter hvilken høyde bildene er tatt fra, vil rutenes størrelse variere fra bilde til bilde. Derfor er koden for beregning av denne plassert i løkken som blar gjennom alle bildene. Denne koden kan finnes under overskriften *Calculation of size of windows* i *Main*-skriptet.

For å beregne hvilken størrelse disse største objektene kan ha i bildene ut ifra et estimat i meter, brukes navigasjonsdata sammen med kartdata. Kartdataene gir høyden på terrenget for en gitt posisjon. Navigasjonsdataene gir posisjonen og retningen av kameraet i det bildet ble tatt. For hver piksel i bildet gir kamerainnstillinger og -egenskaper retningen til den innkomne strålen. Ved å slå opp i kartdataene kan man finne høyden på terrenget under den banen strålen har fløyet. Lengden på denne banen, fra kameraet til den treffer terrenget, angir hvor langt det er mellom kameraet og terrenget avbildet i akkurat denne pikselen. Vet man i tillegg hvor stor synsvinkel hver piksel har i kameraet som er brukt, kan man ved enkel trigonometri finne hvor mange meter hver piksel dekker på bakken. Figur 36 viser denne sammenhengen.



Figur 36: Pikseldekning på bakken

Funksjonen som brukes til å beregne avstanden mellom kamera og bakken en piksel dekker, er kalt *dist_to_pixel* i koden. Funksjonen tar utgangspunkt i vinkelen fra et vannrett plan til synsretningen for den aktuelle pikselen. Denne vinkelen er merket *a* på figur 37. Deretter beregnes høyden til terrenget punkter langs bakken under strålens bane, med høyden rett under kameraet først og deretter stegvis utover. For hver høyde beregnes fortløpende vinkelen mellom et vannrett plan og retningen fra kamera til terrenget. Disse linene som utgjør vinkelen med det vannrette plan er merket med *b* i figur 37, og hver *b* har indeks som angir rekkefølgen de blir beregnet. Vinkel *a* og *b* sammenlignes, og når *b* er blitt mindre enn *a* kan avstanden mellom kameraet og bakken finnes enkelt trigonometrisk. Steglengden er også markert på figuren. Denne kan justeres av operatør for ønsket nøyaktighet, og bør stilles hvis for eksempel brukes høydedata med vesentlig dårligere oppløsning. En økning av steglengden vil også redusere prosesseringstiden, hvis dette skulle være nødvendig.



Figur 37: Beregning av distanse til terreng bak piksel

Parameteren for maksimal størrelse for de menneskeskapte objektene trenger ikke stilles nøye. For å kunne komme frem til en nøyaktig bredde på rutene trengs tilsvarende nøyaktig kartdata og posisjonsdata. Kartdataene brukt under utviklingen og testingen i denne rapporten har vært høydedatabasen *Digital Terrain Elevation Data level 1* (DTED)[14]. Denne inneholder maksimumshøyde for hver 90 x 90 meter, noe som kan påvirke resultatets nøyaktighet. Hvor mye nøyaktigheten påvirkes, avhenger av hvor mye bakken innenfor bildet avviker fra den registrerte høyden i kartdataene. Innenfor så små områder som 90 x 90 meter er det sjeldent terrenget har så stor høydeforskjell at objektene på bakken blir flyttet vesentlig mye nærmere eller lenger fra kameraet.

[14] Mer om denne kartdatabasen kan leses på [w11].

I testdataene var det bygninger som utmerket seg som de største menneskeskapte objektene. Størrelsen på disse varierte en god del, men holdt seg innenfor 30 meter, som er brukt som maksimal størrelse. For større objekter er det antatt at hver 30 meter lang del av objektet hver for seg inneholder nok egenskaper til å kunne gjenkjennes som menneskeskapte objekter. Denne parameteren leses i starten av skriptet, sammen med alle de andre anvendelsesavhengige parameterne (se figur 35).

Siden beregningen av rutestørrelsen ikke trenger å være svært nøyaktig er ikke dekningsbredden til alle pikslene i hvert bilde regnet ut. For hvert bilde beregnes dekningsbredden kun for noen godt spredte piksler. Gjennomsnittet av disse blir deretter brukt som generell dekningsbredde for pikslene i bildet. Hvor unøyaktig dette er, avhenger igjen av hvor kupert terrenget er i bildet, og hvor stor forskjell dette gjør for objektene plassert på de forskjellige bildeposisjonene. I koden er antallet piksler som beregnes gitt i en parameter med to tall; Ett for antall piksler i høyden som skal beregnes og ett for antall i bredden. Disse spres rundt i bildet med like avstander i høyde og i bredde. For hver av disse pikslene beregnes bredden av dekningsfeltet som nevnt tidligere. Tettheten av pikslene bør settes i forhold til hvor kupert terreng bildene inneholder og størrelsen på bildene. Parameteren bør derfor settes av operatør for hver anvendelse og leses derfor i starten av skriptet.

Til bilder der det er stor forskjell i avstanden mellom kameraet og forskjellige punkter i terrenget i bildet passer det dårlig med en fast rutestørrelse. Dette forekommer der det tas bilder over en bakketopp eller lignende, slik at bakketoppen forkommer svært nær i enkelte deler av bildet, mens terrenget bortenfor kan skimtes i området over bakketoppen. Her vil en gjennomsnittlig vindusstørrelse passe dårlig til begge områdene. Derfor er det en ekstra sjekk på resultatet av distansene regnet for de forskjellige pikslene, der for stor forskjell i distansene utløser en advarsel. Hvor stor forskjell som ønskes avdekket og beregningen eventuelt avbrutt er opp til enhver operatør å tilpasse sitt bruksområde av algoritmen, og kan bestemmes ved å sette variabelen *stdLimit* som angir hvor mange prosent distansene kan avvike fra gjennomsnittet. I slike tilfeller kunne man tenke seg ruter av to forskjellige størrelser, men dette er resurskrevende og vanskelig å lage en god løsning for, ikke minst fordi kartdataene ikke er nøyaktige nok til å finne akkurat hvilke piksler som tilhører de to delene av bildet. I koden er det allikevel brukt ruter beregnet ut ifra gjennomsnittsdistansen, men det skrives ut en advarsel for å klargjøre hvorfor rutene i bildet kan ha en noe unaturlig størrelse.

Til slutt beregnes rutebredden, som er antallet piksler som skal til for å dekke et dobbelt så stort areal i terrenget som den antatte største menneskeskapte objektet man kan regne med å observere.

I denne metoden ligger det en mulighet for effektivisering. Ved kjent fast flyhøyde og noenlunde flatt terreng gjennom opptakene, kan rutestørrelsen med fordel beregnes på forhånd, for så å bruke denne gjennom hele flyturen. I denne rapporten er ikke dette testet. Ved å definere en statisk rutestørrelse for alle bildene på forhånd slipper man all beregningen ovenfor, samt kartdataene. Nøyaktige kartdata for større flygebaner krever en del lagringsplass. I tillegg kreves det en del prosessering ved kalkulering av avstandene til pikselens dekning i terrenget.

4.3 Lyse områder, definert med statisk terskel

I denne beregningen gis det poeng for lyse områder i bildene. Hva som bør karakteriseres som lyst er ikke alltid lett definerbart. Statisk terskel er en måte å definere dette på.

Dette kapitlet har tre deler. Først forklares hva som menes med en statisk terskel. I den neste delen forklares hvordan den statiske terskelens verdi er bestemt. I den siste delen forklares hvordan det er beregnet en poengsum på grunnlag av det tersklede bildet. Selve tersklingsmetoden forventes kjent; Piksler over terskelen blir satt til én, mens alle andre piksler blir satt til null.

Koden som beregner denne egenskapens fremtreden i rutene ligger under overskriften *Thresholding* i løkken gjennom alle rutene i skriptet *Main*. Koden for selve tersklingen er i metoden *threshold*.

4.3.1 Statisk terskel

Med en statisk terskel menes en terskel som ikke endrer verdi i løpet av algoritmen. Terskelens verdi settes i starten av skriptet sammen med de andre parametrene, og forblir uendret. Selve verdien til terskelen er utelatt fordi KDA ikke ønsker å oppgi absoluttverdiene til piksler. Siden sensoren brukt ved opptak av testdataene som tidligere nevnt er DC-koblet, er absoluttverdiene til piksler[15] fra forskjellige opptak sammenlignbare. Dette gjør at den statiske terskelen bør kunne gjelde for alle opptakene så lenge objektene man søker er forventet å stråle like mye i de forskjellige opptakene.

4.3.2 Valg av statisk terskelverdi

For å bestemme hvilken verdi som burde brukes er det sett på bilder med typiske lyse områder det var ønskelig å skille ut, samt lyse områder man ikke ønsket å plukke ut. Typiske eksempler på områder man helst ville unngå å plukke ut vises i figur 23 og figur 24, der lyse områder forekommer naturlig, mens figur 16, figur 19, figur 20 og figur 22 viser typiske lyse områder som ønskes plukket ut. Det var noe overlapp mellom verdiene fra objektene man ønsket å registrere og objektene man ikke ønsket. Dette gjør at disse kategoriene ikke kan skilles enkelt ved å bruke en terskel. Det er heller ikke mulig å sette krav til størrelsen av de lyse områdene for å skille de to kategoriene. Det viste seg at de unaturlig lyse områdene ofte var de største, men dette varierte. Derimot kunne det fastslåes at de lyseste naturlig lyse områdene forekom relativt sjeldent, og besto nesten utelukkende av stein og fjellpartier som stikker opp i dagen. Ut ifra en vurdering av at det ikke var noen stor ulempe å plukke med disse få tilfellene ble terskelen satt til inkludere slike sære tilfeller. Dermed kunne metoden brukes til å plukke med det meste av de bygninger som skiller seg ut som lyse.

4.3.3 Kalkulering av poengsum for statisk terskling

Siden egenskapsidentifiseringene (her av lyse områder) skal resultere i én poengsum, er det valgt å sette denne til antallet piksler i ruten som er over terskelen. Dette er ikke et godt kriterium for å vurdere hvor trolig det er menneskeskapte objekter i ruten, siden menneskeskapte objekter kan ha få, men lyse piksler, slik som bilen i figur 20. Denne poengberegningen er allikevel valgt i

[15] Med absoluttverdier menes pikslenes verdier før eventuelle normaliseringer til verdier mellom for eksempel 0 og 1 eller 0 og 256.

mangel på noe bedre. Konsekvensen av dette er tatt ved bruk av poengsummen senere. Da vil ruter med minst én så varm piksel (det vil si poengsum på minst én) bli karakterisert som interessante områder med hensyn på lyse områder. Slik bevares områder som inneholder uvanlig varme elementer, uavhengig av objektets størrelse, slik som ønsket.

Denne bruken av poengsummen gjør det mulig å korte atskillig ned på koden og algoritmen. Algoritmen kunne i dette tilfellet returnert umiddelbart etter å ha funnet én piksel i ruten som oversteg terskelen, men for å åpne for andre anvendelser på andre data er koden fullstendig.

4.4 Lyse områder, definert med dynamisk terskel

I denne beregningen gis det som nevnt også poeng for lyse områder i bildene, men her er det brukt en annen metode for å definere de lyse områdene.

Dette kapitlet har også tre deler. Først forklares hva som menes med en dynamisk terskel. I den neste delen forklares hvordan den dynamiske terskelens verdi beregnes. I den siste delen forklares hvordan det er beregnet en poengsum av det tersklede bildet.

Koden som beregner denne egenskapen ligger under overskriften *Thresholding* i løkken gjennom alle rutene i skriptet *Main*. Koden for selve tersklingen ligger som tidligere nevnt i metoden *threshold*, mens beregningen av den dynamiske terskelen er plassert under overskriften *Maintaining dynamic threshold* i løkken gjennom bildene.

4.4.1 Dynamisk terskel

Den statiske terskelen tar ikke hensyn til at de generelle forholdene under bildeopptakene kan endre seg. Hvis man under flygningen først skulle fotografere i et område sola har varmet opp fra en skyfri himmel i en lengre periode, for så å fly inn under ett skydekke og fotografere områder som har ligget i skygge over lengre perioder. Dette siste området vil da fremstå med lavere verdier enn det forrige, og objekter som sannsynligvis ville ligget over terskelen i intensitet om de hadde ligget i det oppvarmede området, kan nå ha intensitet under terskelen. Siden objektet er like interessant å bevare hadde det vært ønskelig med en terskel som endret seg som følge av det generelle lysnivået i bildene.

Som et alternativ kan man tenke seg å trekke ut de områdene av hvert bilde som skiller seg ut som vesentlig lysere enn resten. En metode ville være å regne ut gjennomsnittet og standardavviket for alle pikselverdiene i bildet. Deretter kunne man sette en terskel på antall standardavvik over gjennomsnittet som skal betraktes som vesentlig lysere. Ulempen med denne metoden er at en terskel som regnes ut ifra et grunnlag på kun ett bilde vil være meget følsom for særegne bilder. Skulle det for eksempel komme ett bilde med så mange lyse bygninger at gjennomsnittet og standardavviket trekkes vesentlig opp, vil ikke lenger de lyse områdene ligge mange nok standardavvik unna gjennomsnittet og alle bygningene betraktes som for mørke. Slike tilfeller er ikke urimelige, siden hus gjerne ligger i klynger mellom skog, jorder og annen naturlig bakgrunn. For å unngå at slike særegne bilder skal ha for stor innvirkning på terskelen bør terskelen genereres ut ifra flere påfølgende bilder. Ved å bruke de *n* siste bildene som grunnlag for utregningen av terskelen, må en generell endring i bildene vare i over *n* bilder før terskelen undergraver endringen. En slik generell endring kan for eksempel være at alle pikselverdiene har blitt vesentlig mindre som følge av overskyet vær.

Av testdataene ser det ut til at omtrent 10 bilder vil være et greit antall bilder å legge til grunnlag for at det har skjedd en generell endring i bildegrunnlaget. Ett mindre antall vil gjøre terskelen følsom for klynger med hus og andre uvanlig lyse felter som varer opptil 10 bilder, mens ett større antall vil gjøre tilpassningen av terskelen tregere i de tilfellene der det faktisk skjer en generell endring i grunnlaget. Selv om 10 bilder kan høres tregt ut, går dette fort under opptakene gitt en opptaksfrekvens på flere bilder i sekundet. Gjennomgangen av testdataene viser at større lyse felter som strekker seg over flere bilder, men som det ikke ønskes å undergrave av terskelen, har passert i løpet av 10 bilder.

4.4.2 Beregning av dynamisk terskel

Kapittel 12.9 i [b3] viser hvordan rekursive filtre kan brukes til å filtrere bort den innvirkningen impulser med visse frekvenser har på en parameter som utvikles på bakgrunn av både nye verdier og parameterens tidligere verdier. Denne metoden kan brukes til å unngå at terskelen blir sterkt påvirket av generelle endringer i bildene som ikke varer opp til 10 påfølgende bilder. Generelt kan slike filtre brukes til å kutte impulser i ethvert frekvensintervall med den nøyaktighet som ønskes. Høyere nøyaktighet får man ved å anvende filtre av høyere orden, men også prosesseringskravene og kjøretiden øker med ordenene. Derfor er det vanligere å holde seg til filtre av relativt lave ordener. I denne oppgaven tilsvarer frekvensen den inverse av antallet bilder man ønsker å overse endringer i, altså 0,1. Nøyaktigheten på denne grensen er ikke bedre enn nøyaktigheten på anslaget om 10 bilder som øvre grense på hva det her er ønsket å overse av endringer i bildene. Siden denne ble anslått fritt etter observasjoner i testdataene, regnes kravet til nøyaktighet som minimal. Dette fører til at det ikke trengs noen høy orden på filteret, men at det klarer seg med et filter av 1.orden, som gir en enkel utregning for hvert steg.

Følgende formel viser et lavpassfilter med tidskonstant 0,1 (det vil si at det filtrerer bort påvirkninger lavere enn frekvensen 0,1):

$$T_n = 0{,}1 \cdot x_n + 0{,}9 \cdot T_{n-1} \qquad (1)$$

der:

- T er den dynamiske terskelen
- Indeksen nede til høyre angir hvilket bilde tegnet gjelder for. For eksempel står T_5 for den dynamiske terskelen til det femte bildet.
- x står for den utregnede terskelverdien til ett bestemt bilde. I dette tilfellet er denne regnet ut som et visst antall standardavvik over gjennomsnittet til pikselverdiene i bildet.

Konstantene på 0,9 og 0,1 er valgt for å gi den ønskede tidskonstanten på filteret, som igjen gir knekkfrekvensen på 0,1.

For å slippe å regne gjennomsnitt og standardavvik fra hele bilder (noe som fort kan overstige flere titalls tusen pikselverdier), brukes 1 % av pikslene som beregningsbakgrunn. Disse trekkes tilfeldig. I og med at gjennomsnittet og standardavviket ikke trenger å være nøyaktig, regnes dette utvalget som tilstrekkelig. Fra utvalget regnes en ny terskelverdi for akkurat dette bildet ved å legge et visst antall standardavvik til gjennomsnittet. Denne terskelen brukes deretter i beregningen av den nye dynamiske terskelen, som regnes ut etter ligning (1). Utprøving av

forskjellige antall standardavvik på testdataene viste at en terskel på 5 standardavvik fra snittet gav den dynamiske terskelen som best skilte ut menneskeskapte objekter.

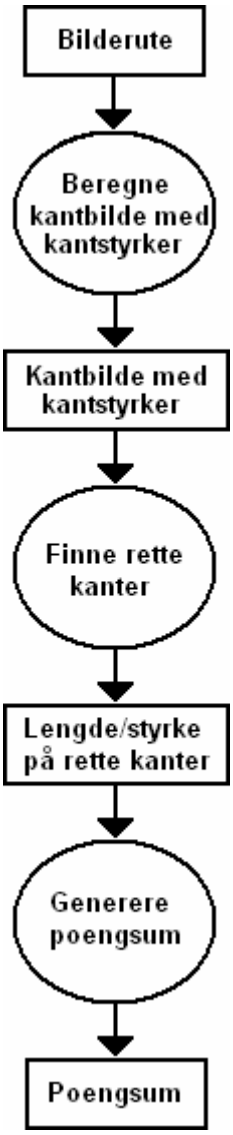
4.4.3 Kalkulering av poengsum for dynamisk terskling

Til denne kalkulasjonen er det brukt samme metode som ved statisk terskling; Poengsummen er lik antallet piksler over terskelen. Begrunnelsen er også den samme; Man ønsker kun å vite om det fantes noen lyse piksler innen ruten, siden de lyse feltene på et menneskeskapt objekt er ofte små, og alle tilfeller av slike lyse punkter bør inkluderes i bildene.

4.5 Markante rette kanter

I denne egenskapsidentifiseringen er det meningen å gi poeng for de sterkeste rette kantene i bildet. Poengkalkulasjonene blir foretatt under overskriften *Sharp edges* i løkken gjennom alle rutene i *Main*-skriptet. All kalkulasjonen for denne egenskapen er lagt i metoden *sharpEdgeExtraction*.

Denne egenskapsberegningen består av tre hoveddeler, som her blir forklart grundig i hvert sitt delkapittel. Først beregnes det et kantbilde av den aktuelle ruten. Siden det her legges vekt på styrken til kantene er det viktig at denne informasjonen bevares i kantbildet. Deretter identifiseres rette kanter i det genererte kantbildet. Igjen er det viktig å bevare informasjon om styrken til informasjon om styrken til de rette kantene. Til slutt genereres en poengsum ut ifra informasjonen om styrken og lengden av de rette kantene som er funnet. Figur 38 viser disse tre delene av beregningen og dataene mellom dem.



Figur 38: Dataflytdiagram for beregning av poengsum for markante rette kanter

4.5.1 Beregning av kantbilde med kantstyrker

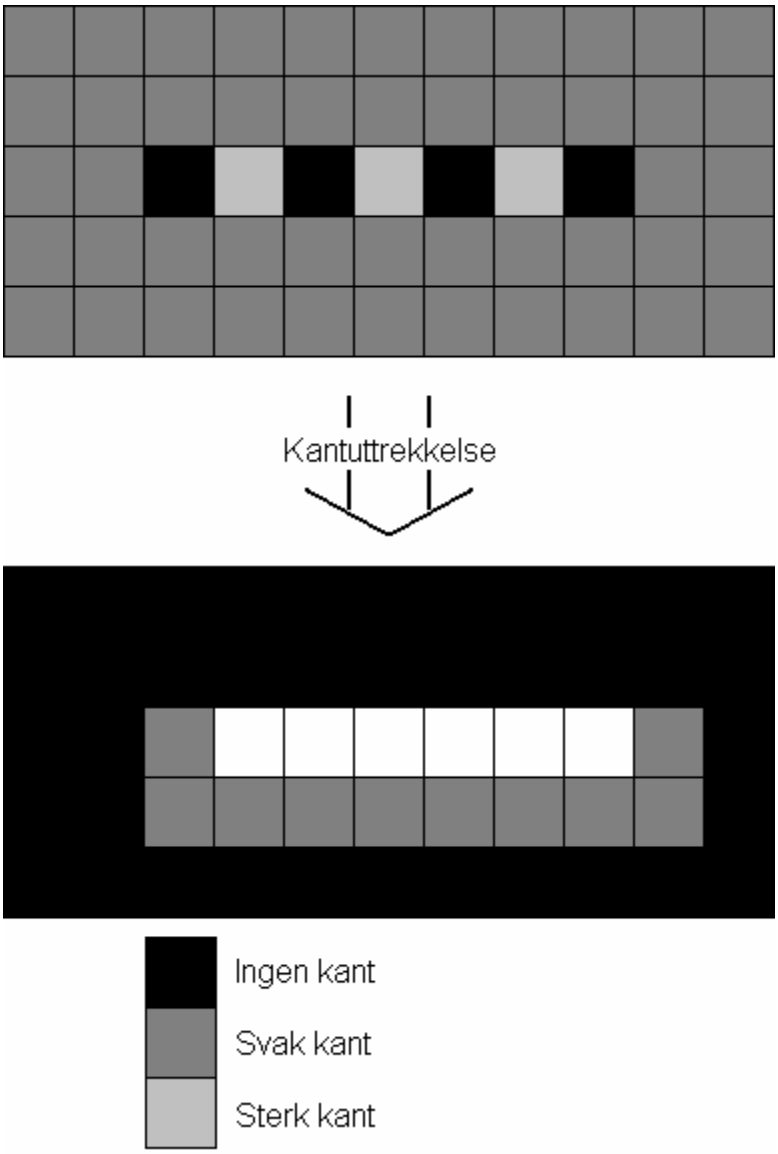
For å identifisere kanter i bildene brukes kantdeteksjonsalgoritmer. MATLAB har et utvalg slike i sitt bibliotek av metoder, men ingen av disse bevarer styrken på kanten, noe som er essensielt i denne sammenheng, der man ønsker å gi sterke kanter mer poeng. Derfor ble det nødvendig å bruke en egen kantdeteksjon i denne sammenhengen. Den enkleste metoden for å definere en kant er enkel derivasjon langs radene og kolonnene i bildet.[16] Siden derivasjonen også gir et uttrykk for kantens styrke i sin absolutte verdi, passer metoden fint i denne sammenheng. Å benytte derivasjon langs to retninger gir som omtalt i forrige delkapittel en forenkling der skrå kanter blir registrert som kombinasjoner av horisontale og vertikale kanter. Ett alternativ ville vært å detektere skrå kanter i tillegg til horisontale og rette, for eksempel med Roberts

[16] Det er her anvendt den enkleste form for derivasjon: Pikselvis differanse.

kryssgradientoperatorer[17] eller lignende. Videre kunne det vært utarbeidet operatorer for kanter med andre stigninger. Dette ville egnet seg for å avdekke kanter i disse retningene som er for svake til å utgjøre noen stor kant i rent horisontal eller vertikal retning, men siden det her søkes etter akkurat sterke kanter regnes horisontal og vertikal derivasjonen som tilstrekkelig.

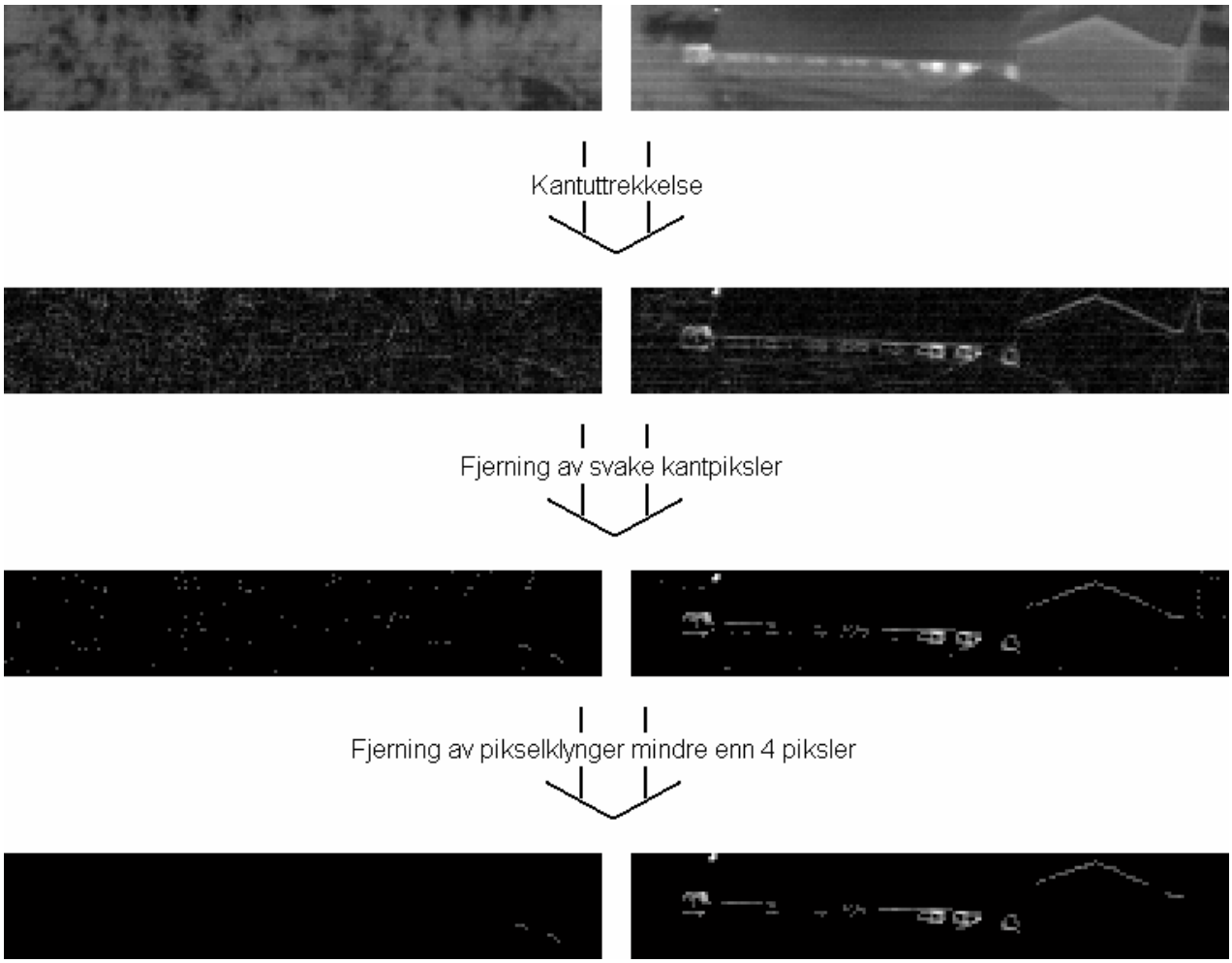
Ved å addere de to kantbildene (vertikalt og horisontalt kantbilde) får man et bilde som viser størrelsen på kantegenskapen for hver piksel, mens retningen på kanten ikke lenger kan bestemmes. Når det allikevel er valgt å gjøre det slik i denne algoritmen er det for slippe å foreta resten av prosesseringen i denne metoden på flere bilder, men kun på ett. Forenklingen kan gjøre at piksler med stor vertikalt kantbidrag, liggende langs en horisontal linje, vil fremstå som en lettere diffus horisontal kant, slik figur 39 illustrerer. Selv om spraglete bakgrunn er vanlig i bildene, er dette et nokså unaturlig mønster å finne. Det kommer av at pikselverdiene må endre seg mye for at den illuderte kanten skal kunne konkurrere med sterke menneskeskapte kanter, slik som de fremstår for eksempel i figur 25 og figur 27, i tillegg til at pikslene skal ligge så nært hverandre. Derfor regnes dette ikke som noe stort problem i denne algoritmen.

[17] Roberts kryssgradientoperatorer er definert i [b2] som $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ og $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$



Figur 39: Horisontal og vertikal derivert av en spraglete linje

For å minke mengden av urelevant informasjon i kantbildet nullstilles de pikselverdiene i kantbildet som har mindre kant enn en viss grense. Grensen som brukes settes av operatøren, siden verdien må justeres i forhold til spekteret på pikselverdiene, og dermed størrelsesordenen på hva som regnes som markante kanter i bildene. Deretter fjernes bidrag av kanter som ikke er lengre enn tre piksler. Denne metoden er kalt *remTriples* i koden og baserer seg på telling av antall nabopiksler. Slike små klynger av kantpiksler forekommer ofte i bildene som resultat av spraglete bakgrunn. Det kunne gjerne vært fjernet kanter av noen flere piksler enn tre, siden kanter med fire piksler også gjerne tilhører støy i denne sammenheng. Grunnen til at det kun er kanter med tre eller færre piksler som fjernes her, er at algoritmen blir vesentlig mer komplisert og resurskrevende når antallet piksler økes. Figur 40 viser effekten av disse metodene på et bilde med en spraglete bakgrunn av busker og et bilde av en bygning.



Figur 40: Fjerning av støy i kantbilde

Av figuren ser man at den spraglete bakgrunnen nærmest forsvinner fra kantbildet, mens de sterke kantene i forbindelse med bygningen bevares.

4.5.2 Identifisering av rette kanter i kantbildet

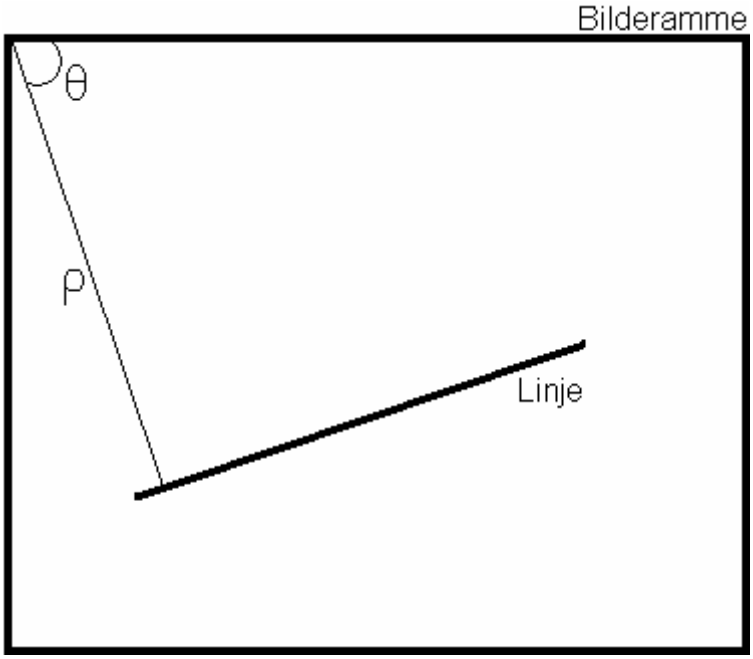
Det neste steget i algoritmen er å lete etter rette kanter i kantbildet. Til dette er som tidligere nevnt Hough-transformasjonen egnet. MATLAB sin implementering av transformasjonen tar utgangspunkt i et binært bilde. I kantbildet som her er utarbeidet tilsvarer verdien til kantpikslene styrken til kanten. En konvertering til et binært bilde vil derfor fjerne informasjon om styrken til kantene, som er en sentral del av den egenskapen som skal identifiseres. Derfor er det her benyttet en egenutviklet variant av Hough-transformasjonen som tar utgangspunkt i et kantbilde med styrkeverdier i kantpikslene og som bevarer informasjonen om denne styrken til de sterkeste rette kantene skal identifiseres. Denne Hough-transformasjonen er kodet i metoden *magnHough*, og er definert som følger:

$$mH(\theta,\rho) = \sum_{\{x,y\}|_{\rho = x\cdot\cos(\theta)+y\cdot\sin(\theta)}} [k(x,y)]$$

der *mH* angir det nye parameterbildet og *k* angir kantbildet.

Den fungerer akkurat som en vanlig Hough-transformasjon for rette linjer, men i stedet for å gi ei linje poeng etter hvor mange kantpiksler som befinner seg på den, får den summen av styrken til de kantpikslene som ligger her[18]. Dermed får kantene poeng for både hvor lange og hvor sterke de er.

Når Hough-transformasjonen her traverserer alle mulige linjer i bildet (ikke bare alle kantene, men alle mulige linjer), representerer den linjene med to parametere. Som nevnt er dette tilstrekkelig for linjer, som har to frihetsgrader. Parametrene som brukes er vinkelen mellom normalen til linja og bildets øvre horisontale kant (kalt θ), samt minste avstanden mellom linja og øvre venstre hjørne av bildet (kalt ρ). Figur 41 viser disse parametrene. θ ligger i intervallet -90º til 90º, mens ρ ligger mellom den negative og den positive verdien av lengden på diagonalen av bildet.

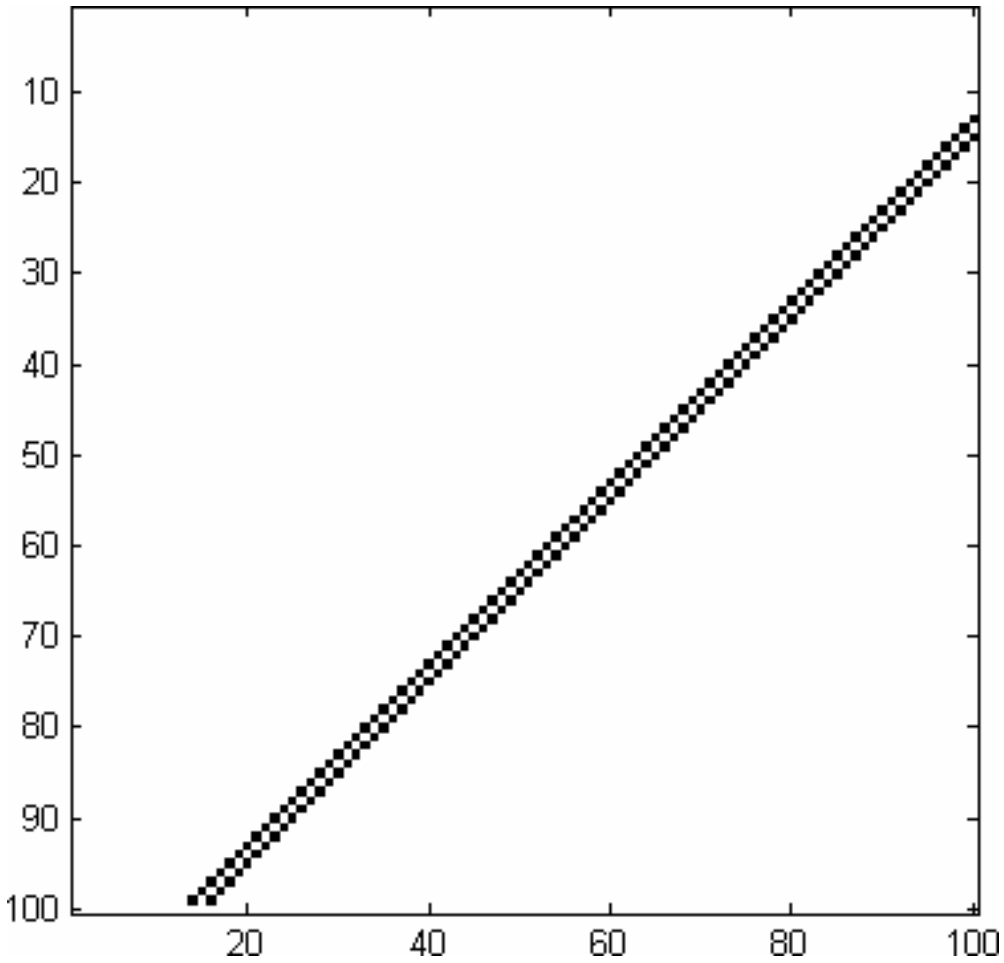


Figur 41: Parametrene θ og ρ brukt i Hough-transformasjonen

Oppløsningen på disse parametrene er en sentral del av Hough-transformasjonen. Disse oppløsningene er de to siste parametere til metodekallet (henholdsvis θ-oppløsning og ρ-oppløsning), og angir hvor mye disse parametrene skal endre seg mellom hver linje. Optimalt sett skal transformasjonen traversere alle mulige linjer i bildet, det vil si alle kombinasjoner av θ og ρ. Siden disse verdiene er flytende, kan de anta uendelig mange verdier innen intervallene sine. Derfor blir bestemmelsen av oppløsingene enn avveining mellom prosesseringsmengde og muligheten for å treffe kantene i bildet med linjene. Jo lavere oppløsning, desto mer prosessering må til, og desto større blir sannsynligheten til å dekke kantene i bildet med linjene. Siden bildet er

[18] I Hough-transformasjonen er hver linje representert ved to parametere, og verdien av pikselen disse parametrene spesifiserer i parameterdomenet er det som betegnes som linjas poeng.

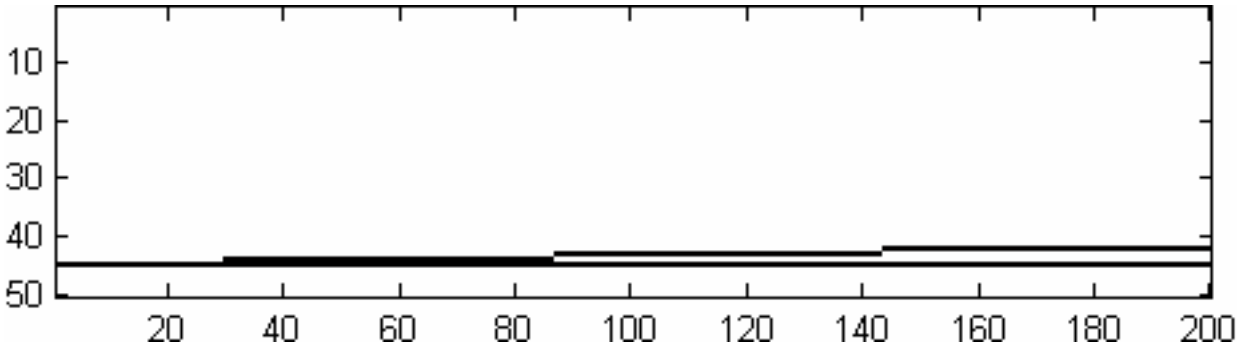
delt inn i piksler, er det en begrensning på antallet mulige linjer i bildet, selv om denne er høy. Man kunne kanskje forvente at en ρ-oppløsning på én piksel ville være nok for å dekke alle linjer. Dette er ikke tilfellet. For linjer som går skrått i bildet vil en avstand på én piksel faktisk hoppe over én mulig linje, slik som figur 42 viser. Her er det tegnet inn to linjer med θ=45º og ρ=80 og 81.



Figur 42: Linjer med avstand lik én piksel

Figuren viser at distansen her burde vært halvvert. Derfor er det brukt 0,5 som ρ-oppløsning i stedet for 1. Figur 43 viser tilsvarende hvordan to linjer med 1º forskjell i θ-verdi spriker mot høyre kant av bildet. Som figur 43 også viser, øker avstanden mellom linjene jo lengre bildet er. Derfor bør θ-oppløsningen vurderes etter hvor store bildene er i enhver anvendelse av algoritmen. Dersom den må settes veldig lavt for å få brukbare resultater, vil kjøretiden til algoritmen øke og det bør vurderes hvorvidt egenskapen er verdt å ta med i egenskapsidentifiseringen.

Etter testing ble det vurdert en θ-oppløsning på 0,2 i denne oppgaven. En så liten verdi var nødvendig for at sannsynligheten for å treffe kantene i bildet med linjene skulle være tilfredsstillende. De lave parametrene på oppløsningene øker prosesseringstiden for algoritmen merkbart, men ble allikevel valgt siden prosesseringstiden er underforstående i denne oppgaven.



Figur 43: Linjer med 1º mellomliggende vinkel

4.5.3 Kalkulasjon av poengsum for markante rette kanter

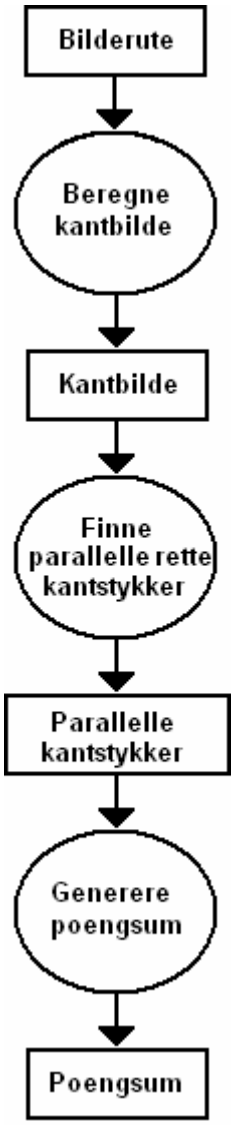
Til slutt i metoden genereres en generell poengsum for hele bildet (som tilsvarer bildet i en rute, siden metoden kalles på hver rute) på bakgrunn av parameterbildet. Dette gjøres ved å legge sammen poengene fra alle linjer med mer enn en viss poengsum. Denne grensen er også avhengig av pikselverdienes spekter, og må derfor settes av operatøren. Den leses inn sammen med alle andre slike parametere i starten av skriptet. Verdien tilsvarer lengden av den minste kanten man ønsker å registrere multiplisert med minste styrke på kanten. Verdien vil dermed avhenge av blant annen hvilke spekter pikselverdiene til rådataene har. Jo bredere dette spekteret er, desto større forskjell er det mellom pikslene som bør karakteriseres som markante, som igjen fører til at grensen må heves. Når det her brukes en grense uavhengig av størrelsen til ruten, kreves det samme kombinasjon av lengde og styrke på kantene for bilder med forskjellig avstand til objektene. Fjerne objekter må dermed ha mye lengre rette kanter enn nære objekter, for at de skal fremstå like lange i bildene. Denne favoriseringen av nære objekter er gjort fordi naturlig bakgrunn har en tendens til å inneholde en del tilsynelatende rette kanter når den kommer på avstand. Småruglete kanter som rundt skog og lignende fremstår ofte rett når ujevnhetene blir for små i forhold til pikseloppløsningen. En grense avhengig av rutestørrelsen (som er proporsjonal med avstanden til objektene i ruten) viste seg å inkludere for mange naturlige områder i små ruter.

Poengsummen til ruten skaleres til slutt ned for å komme i samme størrelsesorden som poengsummene til de andre egenskapene. Siden poengsummene fra alle egenskapsberegningene lagres i samme matrise er det en fordel at de ikke ligger i helt forskjellig størrelsesorden, siden alle tallene i matrisen lagres i samme dataformat. Dataformatene passer igjen til forskjellige størrelsesordner. I skaleringen tas det hensyn til størrelsen til ruten, for å kompensere litt for at større ruter generelt vil, uavhengig av innhold, ha mer poeng enn mindre ruter før skaleringen.

4.6 Parallelle kanter

I denne egenskapsidentifikasjonen er det meningen å gi poeng for parallelle linjer som kan representere parallellogrammer i bildene, siden bygninger ofte utgjør parallellogrammer i bildene. I koden kan poengkalkulasjonene finnes under overskriften *Parallell edges* i løkken gjennom alle rutene i *Main*-skriptet. All kalkulasjonen for denne egenskapen er lagt i metoden *parallellExtraction*.

Som den forrige egenskapsberegningen består også denne av tre hoveddeler, som blir forklart i hvert sitt delkapittel. Først beregnes det et kantbilde av den aktuelle ruten. Det vil her bli valgt en kantuttrekningsmetode. Deretter identifiseres rette kanter i det genererte kantbildet. Det er her ikke ønskelig å bevare kantstyrkene som ved forrige egenskapsidentifisering. Til slutt genereres en poengsum ut ifra hvorvidt det finnes kanter som ligner parallellogrammer. Figur 44 viser disse tre delene av beregningen og dataene mellom dem.



Figur 44: Dataflytdiagram for beregning av poengsum for parallelle kanterstykker

4.6.1 Valg av kantuttrekningsmetode

MATLAB har i sitt bibliotek av metoder en generell kantuttrekningsmetode kalt *edge*. I denne er det implementert følgende kantuttrekningsmetoder:

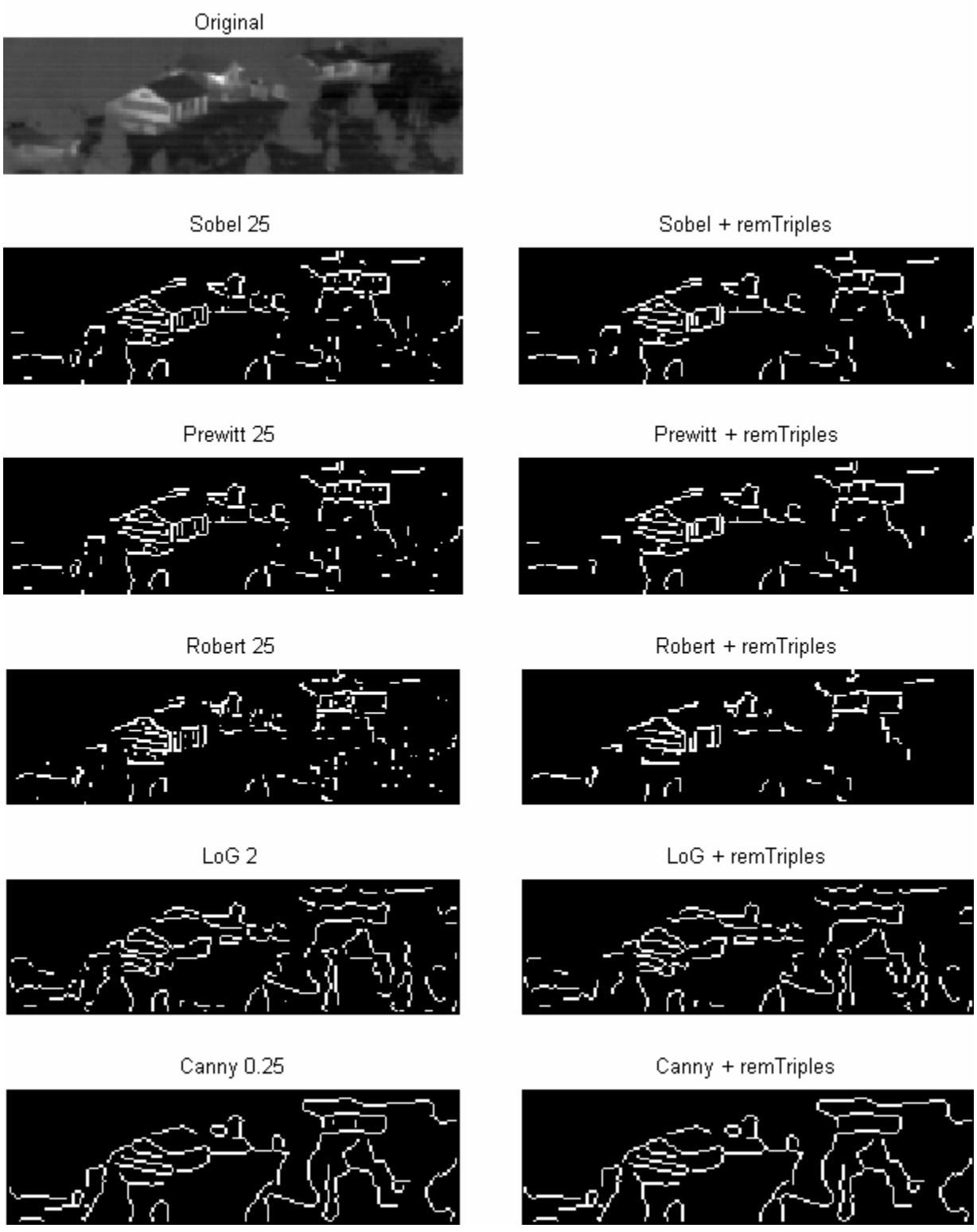
- Sobel
- Prewitt

- Robert
- Logaritmen av gausisk kurve (LoG)
- Canny
- Nullkryssning av derivert av valgt filter [19]

Alle disse metodene er testet unntatt den siste. Grunnen til at denne ikke er testet er at den krever at utvikleren utformer et filter som passer bildedataene. Siden det ble funnet tilfredsstillende kantuttrekningsmetoder blant de gjenværende metodene var dette sett som overflødig.

Figur 45 viser de forskjellige kantuttrekningsmetodene brukt på det samme utklippet fra testdataene. Alle metodene bruker en parameter til å bestemme hvor store endringer i gråtonen som skal defineres som kanter i bildet[20]. Denne parameteren er gitt i antall standardavvik fra gjennomsnittet i LoG og Canny, og antall pikselverdier i de andre (Nullkryssningsmetoden ikke medregnet). Overskriften til bildene, foruten originalbildet, viser hvilken metode som er brukt, samt verdien til terskelen. Til høyre for bildet er det samme bildet gjengitt etter at metoden *remTriples*, som fjerner pikselklynger mindre enn fire piksler, er benyttet. Siden bildene skal brukes etter å ha gjennomgått denne metoden er det mer relevant å se hvordan dette bildet kan brukes, fremfor det til høyre. Dette vil si at det ikke er viktig om det kommer kantelementer mindre enn tre sammenhengende piksler som ikke skal benyttes i kantbildet, siden disse allikevel fjernes før kantbildet skal brukes.

[19] For mer informasjon om de forskjellige kantuttrekningsmetodene, se kapittel 10.3 i [b2].
[20] Canny-metoden bruker egentlig også en nedre terskel, men denne settes til 40 % av øvre terskel hvis ikke noe annet er oppgitt. Etter lettere testing ble denne 40 % -verdien funnet like god som noe annet, og dermed valgt.

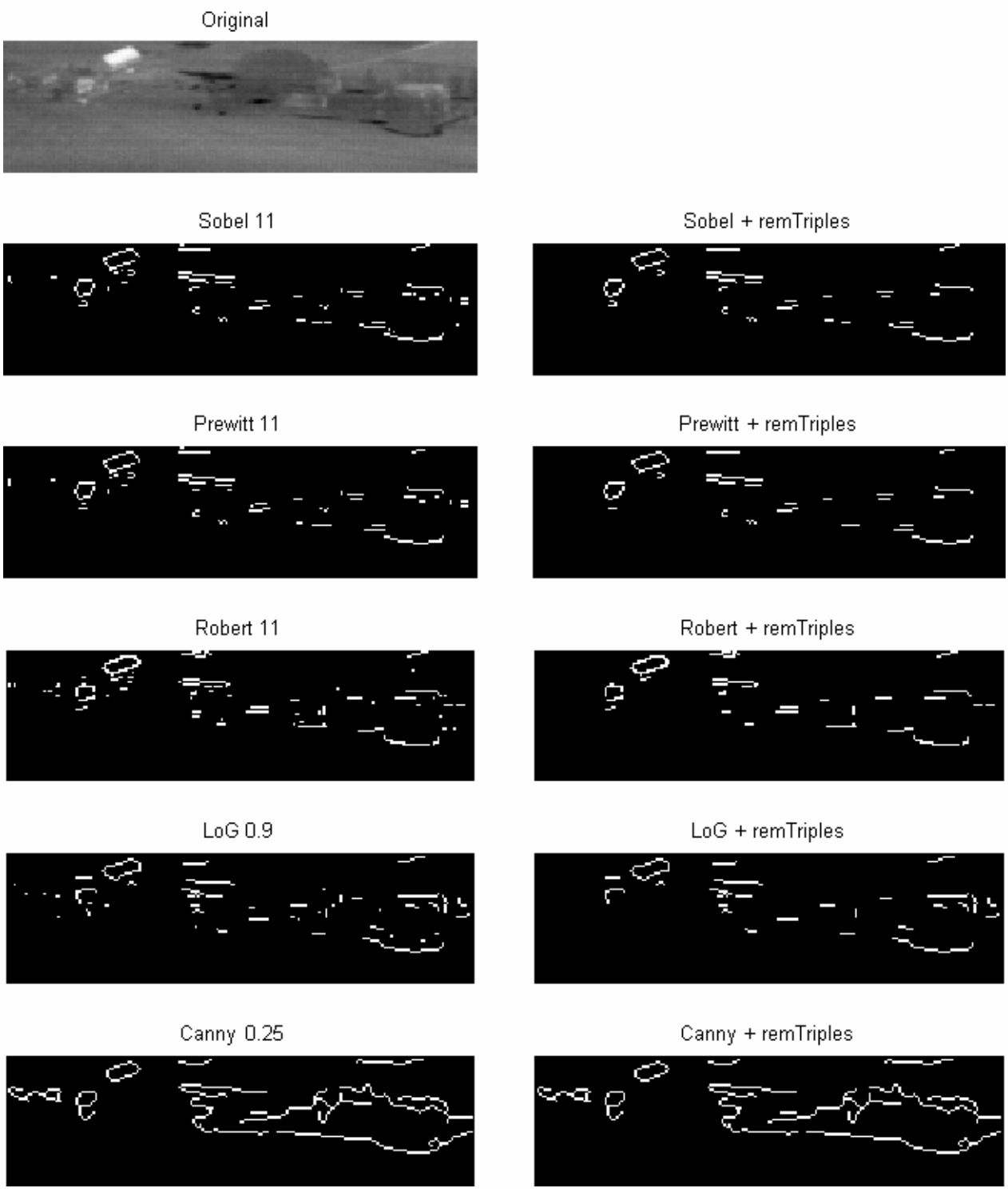


Figur 45: Kantbilder fra forskjellige metoder og med tilpassede terskler

Som figuren viser gir de tre første metodene svært like resultater, mens kantbildene til LoG og Canny ser litt annerledes ut. Forskjellen kommer av at de to gruppene bruker to forskjellige

grunnprinsipper for kantdeteksjon. Mens Sobel, Prewitt og Robert bruker varianter for derivering av bildet, bruker LoG og Canny gausiske filtre. Dette gjør de i bedre stand til å finne lengre men svakere kanter, noe som kan sees på figur 45. Dette trenger ikke være en fordel i denne sammenheng, siden lengre linjer som slynger seg rundt i bildet som oftest er naturlige og ikke menneskeskapte, slik som skillet mellom trærne og bakgrunnen i figur 45.

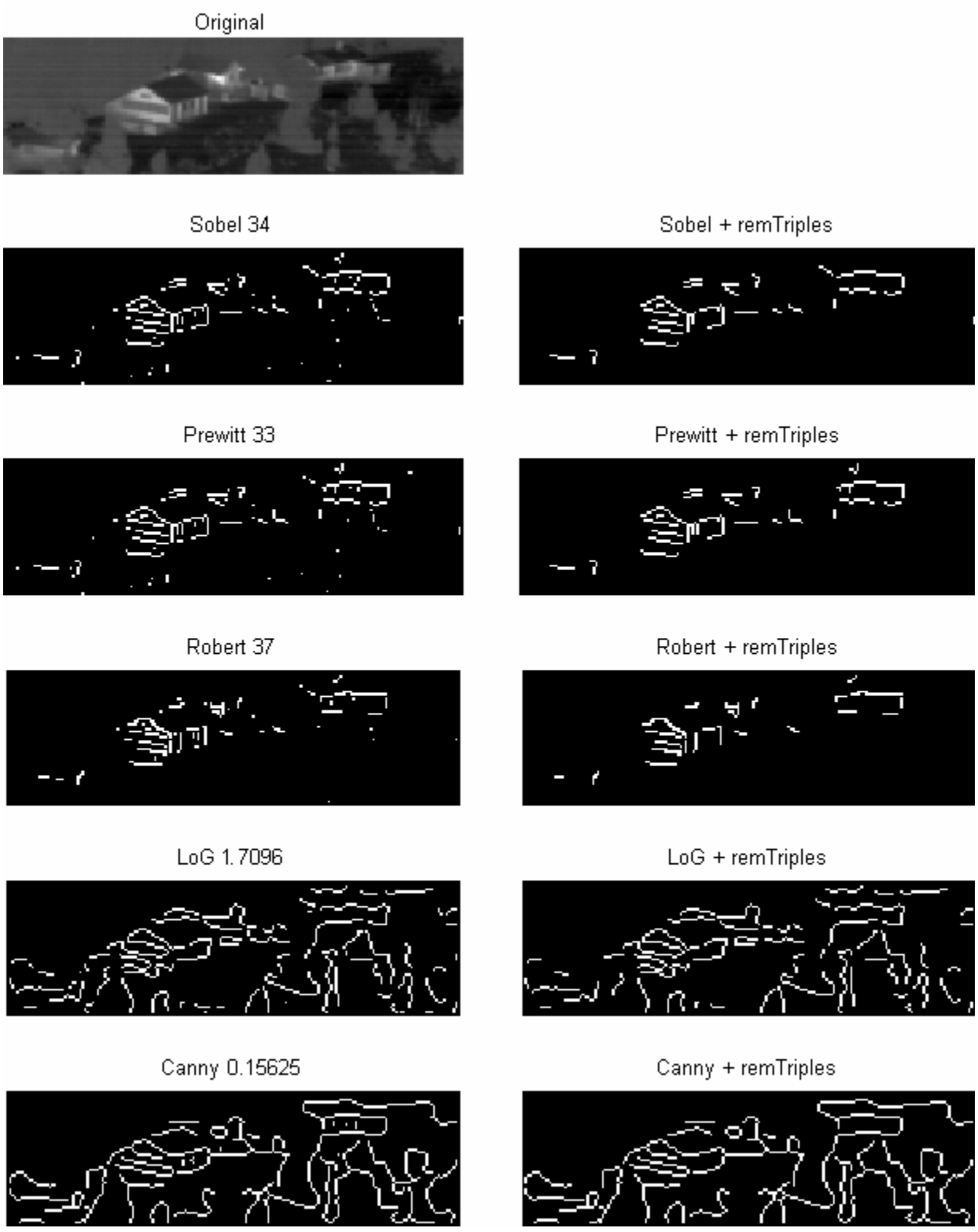
Terskelen brukt i de forskjellige metodene er funnet ved prøving og feiling. Resultatet i figuren er de terskelverdiene som gav det beste resultatet for det gitte bildet. Med det beste resultatet menes flest rette kanter ved de menneskeskapte objektene og minst mulig andre kanter. I tillegg var det viktig å få frem kanter som er parallelle hos de menneskeskapte objektene, siden det er disse som skal finnes i kantbildene. Figur 46 viser det samme som figur 45, men med et annet utklipp. Også her er tersklene funnet ved prøving og feiling.



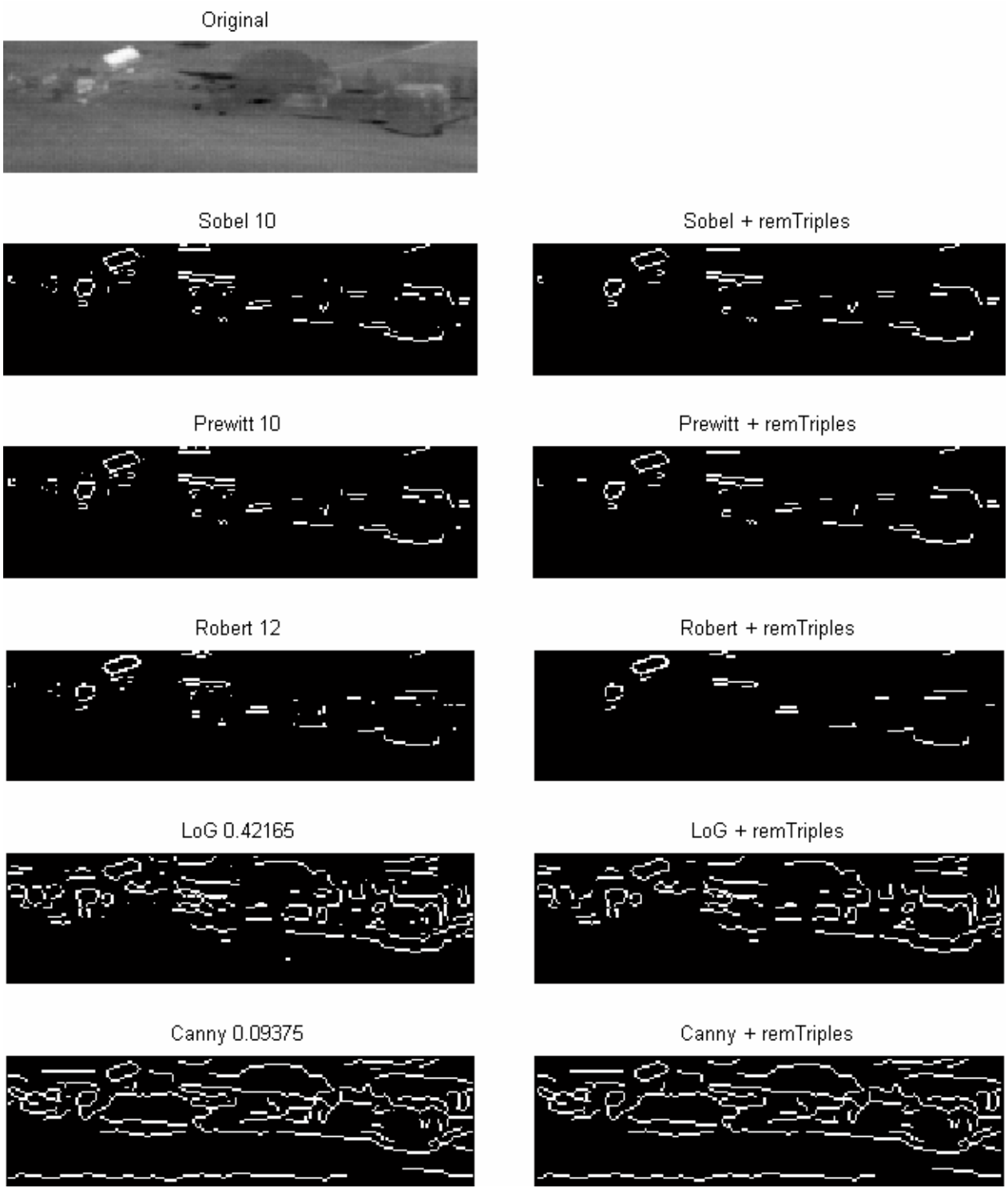
Figur 46: Kantbilder fra forskjellige metoder og med tilpassede terskler

Sammenlignes tersklene i figur 45 og figur 46 ser man at forskjellige terskler passer til forskjellige bilder. Ved anvendelse av terskelen fra figur 45 på utklippet i figur 46 blir det resulterende kantbildet helt svart, uten en eneste kant. Ved motsatt tilfelle (bruk av tersklene i figur 46 på utklippet i figur 45) blir det så mange kanter i kantbildet at det ikke kan brukes.

Derfor er bestemmelsen av terskelen vel så viktig som metodevalget når kantbildet skal genereres. MATLAB sin *edge*-metode tilbyr også en automatisk setting av tersklene, som baserer seg på gråtoneforholdene i bildet. Denne fastsettelsen av terskelen er sentral i denne metoden, men er dessverre ikke dokumentert i MATLAB sin dokumentasjon av metoden, og kan dermed ikke redegjøres for her. Figur 47 og figur 48 viser det samme som figur 45 og figur 46, men med terskler generert av *edge*-metoden.



Figur 47: Kantbilder fra forskjellige metoder og med automatisk genererte terskler



Figur 48: Kantbilder fra forskjellige metoder og med automatisk genererte terskler

Som figurene viser finner metoden automatisk en passende terskel for Sobel, Prewitt og Robert metodene. For LoG og Canny tar de med langt flere kanter enn det som behøves i denne sammenheng. Jo færre kanter som tas med, desto lettere blir det å finne kantene som tilhører de menneskeskapte objektene i bildet, gitt at disse kantene er med. Derfor ansees de tre første

metodene som mest brukbare. Robert-metoden, som bygger på en mindre maske enn de to andre (2x2 piksler i forhold til 3x3), plukker noe færre kanter enn Sobel og Prewitt. Sobel og Prewitt opptrer nesten helt likt i alle testene og ingen er foretrukket fremfor den andre. Her velges Sobel.

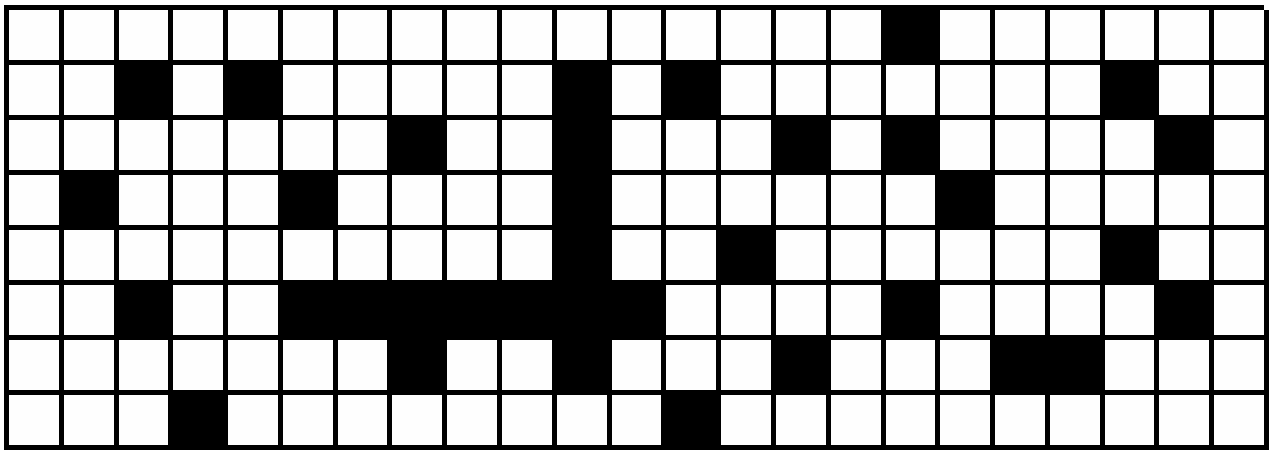
Som ved forrige egenskap fjernes kanter som ikke er lenger enn tre piksler. Siden kantbildet produsert av den valgte algoritmen er et binært bilde, og alle kanter dermed er like sterke, har det ingen mening å fjerne de svakeste kantene, slik det ble gjort ved beregning av markante rette kanter.

4.6.2 Identifisering av parallelle rette kantstykker

En av ideene bak beregningen av denne egenskapen var å dra nytte av informasjon allerede beregnet i forbindelse med markante rette kanter. Nærmere bestemt var det Hough-transformasjonen som skulle anvendes videre. Siden denne transformasjonen måtte endres noe i utviklingen av algoritmen for markante trette kanter, passer det ikke lenger å bruke den her. Når det her letes etter linjer som er parallelle er det ikke ønskelig med et kantbilde som viser hvor sterke kantene er, men kun hvorvidt det er en kant eller ikke. Derfor trengs det et kantbilde som er binært i stedet for gråtonekantbildet brukt i forrige Hough-transformasjonen. Når det i løpet av beregningen av parallelle linjer igjen beregnes en Hough-transformasjon dras det dermed ikke nytte av tidligere beregnet informasjon. Det kunne vært utviklet en kombinasjon av de to versjonene av Hough-transformasjon, slik at beregningen av de to til sammen ble mer effektiv, men det er ikke gjort her. Grunnen til dette har med oppdeling av koden å gjøre. For at hver enkelt egenskap skal kunne beregnes og testes hver for seg, er det lagt vekt på at beregningene skal være uavhengige. Dette gjør at hver egenskapsberegning kan kommenteres ut i løkken gjennom rutene. Dermed var det hensiktsmessig å ikke blande de to versjonene av Hough-transformasjonen i denne utviklings- og testingskoden. For en eventuell realisering av systemet for reell anvendelse bør dette vurderes på nytt, sammen med all annen effektivisering av koden. Dermed er ikke ideen bortkastet selv om den ikke utnyttes her.

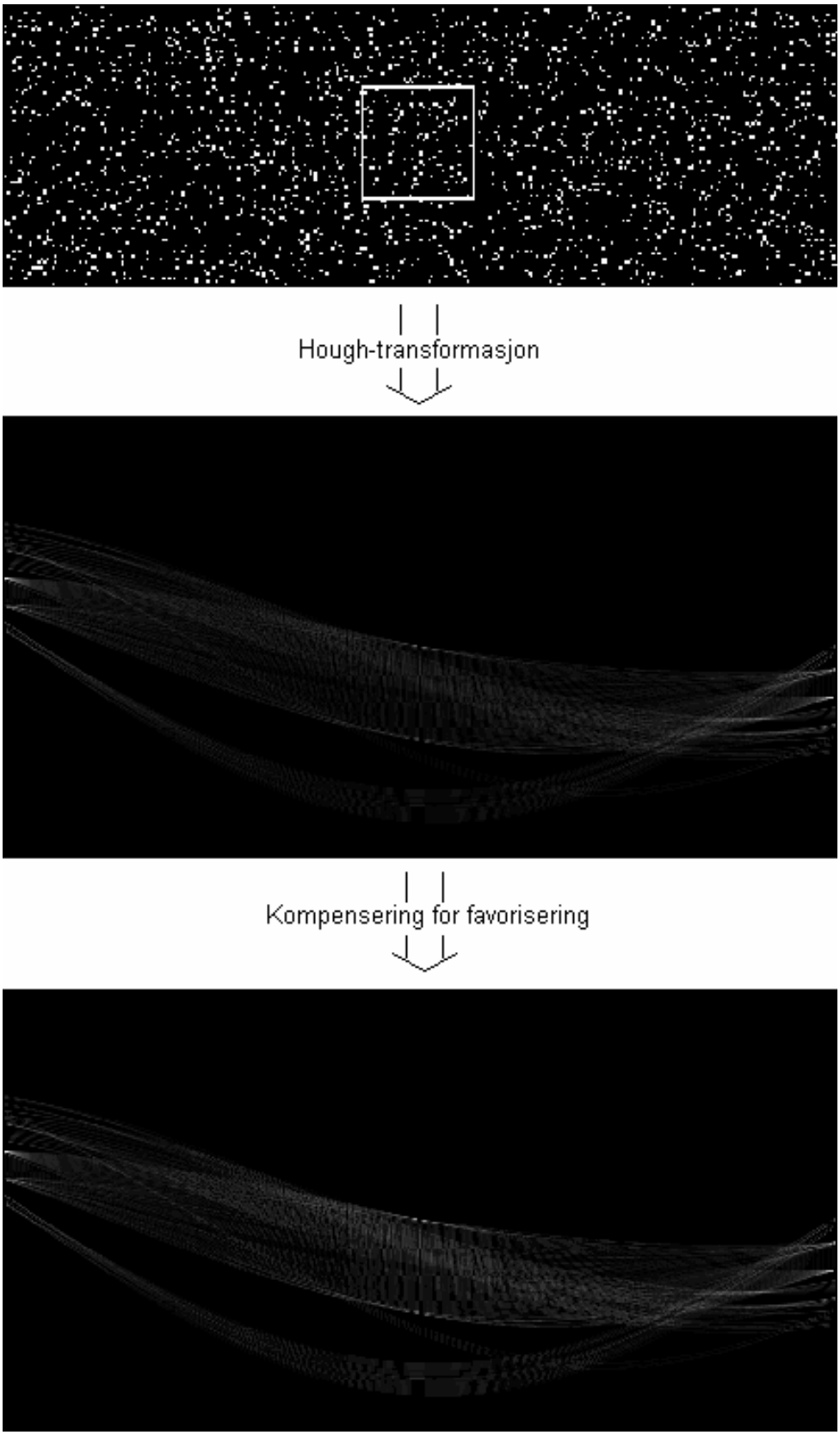
Hough-transformasjonen utføres her ved hjelp av MATLAB sin implementasjon i funksjonen *hough*. Parametrene er de samme som da Hough-transformasjonen ble utført ved forrige egenskapsberegning, siden det ikke er noen forskjell mellom kantene det er ønskelig å avdekke og størrelsen på kantbildet. For begrunnelse av valget av disse parametrene, se derfor forklaring i kapittel 4.5.2.

Hough-transformasjonen har en tendens til å favorisere kanter som ligger på langs i bildene. Siden kantene i Hough-transformasjonen får poeng etter hvor mange piksler som ligger langs den samme linjen vil kanter som ligger på langs i bildet plukke opp flere poeng fra støy enn kanter på tvers av bildet. Figur 49 viser et eksempel. Figuren viser et avlangt bilde med to like lange linjer, en vertikal og en horisontal. I resten av bildet er det tilfeldig fordelt støy og småkanter.



Figur 49: Avlangt bilde med to mørke linjer og støy (Svarte kanter og hvit bakgrunn)

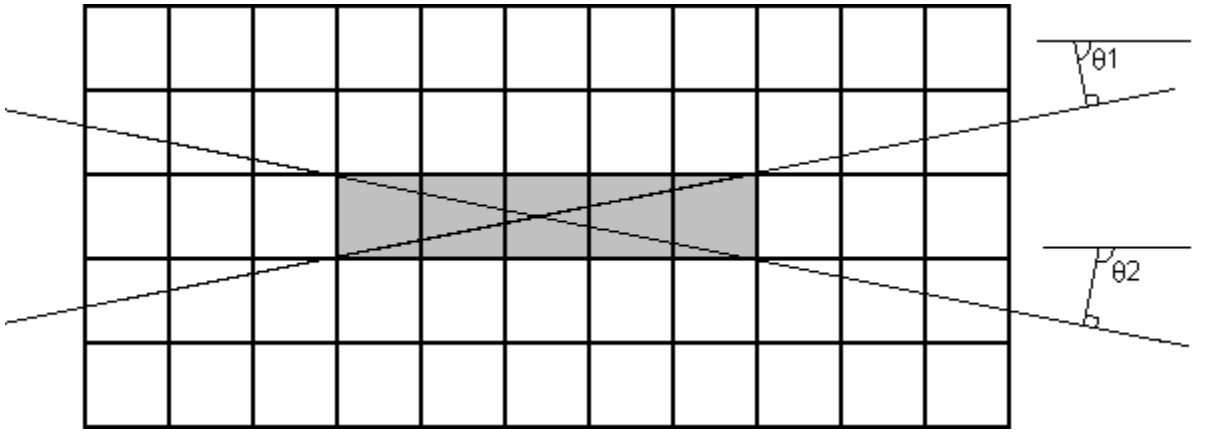
Hver av linjene får 6 poeng i Hough-transformasjonen som følge av pikslene de består av. I tillegg får den horisontale linjen 4 poeng som følge av støyen som ligger spredt utover resten av linjen. Siden bildet er lenger i bredden enn i høyden, blir det flere ekstrapoeng på de horisontale linjene enn de vertikale. For å kompensere for denne fordelen er det konstruert en funksjon som skalerer poengsummen i forhold til vinkelen linjen ligger i bildet, bildets høyde og bredde, samt hvor mye støy det er i bildet. Konstruksjonen av selve funksjonen er lagt ved i vedlegg B. Figur 50 viser hvordan funksjonen virker på et tegnet testbilde. Øverst i figuren vises kantbildet. Dette består av et kvadrat samt masse støy. Under dette vises parameterbildet fra Hough-transformasjonen. Hvert punkt i bildet tilsvarer en linje i bildet. Lysere punkter viser høyere poengsum for linja. Hver rad i bildet viser linjene med samme ρ, mens hver kolonne viser linjer med samme θ. Parameterbildet viser at linjene med lav eller høy θ, det vil si nært horisontale linjer har høyest poengsum, mens vertikale linjer har lave poengsummer. Siden de vertikale og de horisontale sidene på firkanten i kantbildet er like lange, burde de ha like mye poeng, noe det er kompensert for i parameterbildet under. Her kan det skimtes at poengene til de mer vertikale linjene er større, det vil si at verdiene til pikslene mot midten av bredden av parameterbildet er lysere. Dette nederste bildet er generert med kompenseringsfunksjonen.



Figur 50: Utjevning av favorisering av horisontale linjer

Kompenseringsfunksjonen er forkastet i algoritmen. En av grunnene til dette er at det svært sjeldent forekommer vertikale linjer i kantbildene. De vertikale kantene på bygninger og lignende blir som regel for korte til å gjøre seg bemerket, og gjerne for svake som følge av skygger eller lignende. I tillegg er det stor forskjell i hvor mye støy som forekommer ellers i bildet, som er en av parametrene operatøren må sette. Denne er vanskelig å anslå, men har stor innvirkning på det kompenserte parameterbildet. En siste årsak er at koden er unyttig så lenge rutene det jobbes på ikke er utpregede avlange.[21]

Deretter begynner uttrekningen av de lengste kantene fra parameterbildet. Til dette brukes MATLAB sine funksjoner *houghpeaks* og *houghlines*. Den første funksjonen finner de pikslene som har størst verdi i parameterbildet. Disse representerer kanter av like mange piksler som poengsummen. Siden det kan variere mye hvor mange kanter som bør plukkes ut fra hvert bilde er det ikke noe strengt krav til hvor mange kanter som skal plukkes ut. Hele 50 kanter kan plukkes ut, hvis så mange gode kanter finnes. I løpet av testingen på testdataene gitt i denne oppgaven ble det aldri oppdaget bilder med så mange kanter. Derimot er det viktigere å plukke ut aktuelle kanter. Siden de fleste uaktuelle rette kantene i kantbildene er forholdsvis små, er det satt krav til kantene om at de utplukkede kantene skal ha en viss lengde. Etter en gjennomgang av testdata med aktuelle menneskeskapte kanter har det vist seg at de aktuelle kantene har vært minst 20 % av den lengden som tidligere er oppgitt som lengden av det største objektet man kan tenke seg å finne. Siden oppløsningene i parameterbildet er så fin, er det også viktig å ikke plukke ut linjer som er representert med piksler i umiddelbar nærhet i parameterbildet. Én kant i kantbildet blir som regel representert med flere piksler i parameterbildet som en følge av den fine oppløsningen. Figur 51 viser et slikt tilfelle.

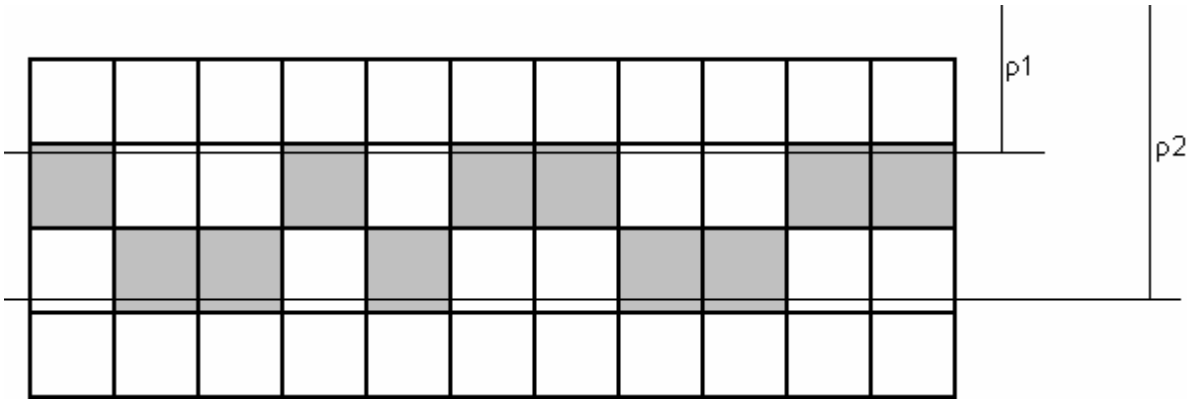


Figur 51: En kant, flere verdier av θ

En rett kant faller her sammen med flere linjer med forskjellige θ- og ρ-verdier. Vinklene θ1 og θ2 er vidt forskjellige, men teller den samme kanten når piksler langs linja skal telles. Tilsvarende blir det med ρ-verdier. I tillegg til at enkelte kanter kan registreres på flere ρ-verdier i parameterbildet, er det også ønskelig å ikke registrere linjer som ligger for nært hverandre til å kunne registreres som parallelle. For eksempel kan neppe to kanter som ikke har noe mellomrom

[21] Koden som utfører kompensasjonen finnes fortsatt i den vedlagte koden, men er kommentert ut. Dette er for å kunne vise hvordan den var implementert.

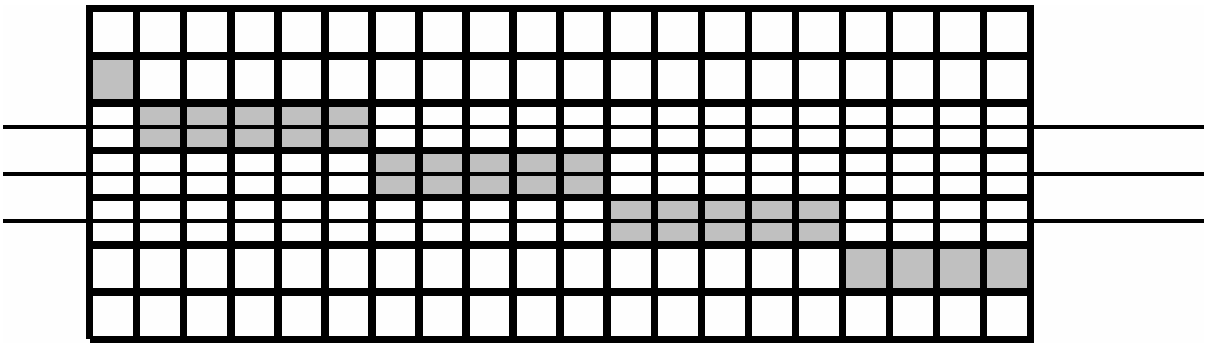
mellom seg regnes som parallelle kanter. Dette kan like gjerne være forårsaket av en ujevn kant som varierer mellom å ligge langs den to pikselrader i kantbildet, slik som illustrert i figur 52.



Figur 52: En kant, flere verdier av ρ

Av denne grunn var det viktig å ikke velge linjer i parameterdomenet som lå for nært hverandre i enten θ- eller ρ-retning. For å unngå dette ble det reservert et naboområde rundt hver piksel som representerte en linje som alt var plukket ut. I det reserverte området er det ikke mulighet til å plukke nye linjer. Det beste resultatet ble observert med et naboområde på 9x15 ruter. Dette tilsvarer linjer innenfor ± 0,8 graders θ og ± 3,5 piksels ρ.

Etter at linjene med flest poeng var plukket ut av parameterbildet, var den opprinnelige tanken å kun sammenligne θ-verdien til disse for å se hvilke som var parallelle. Dette ville gi de kantene i bildet som var parallelle. Dette viste seg å være utilstrekkelig. Særlig var det ett uønsket fenomen som slo til. Figur 53 viser én lang kant som består av flere horisontale biter. Disse bitene har en tendens til å bli trukket ut som egne kanter av *peaks*-funksjonen.



Figur 53: En skrå kant, tre parallelle linjer

Disse små kantstykkene er parallelle, men utgjør ingen parallellogrammer i bildet, siden de representerer den samme kanten. Dette var et fenomen som forekom ofte, og det var derfor viktig å unngå at de ble regnet som parallelle kanter. For å kunne avgjøre hvorvidt linjene plukket ut av parameterbildet faktisk kunne representere parallellogrammer måtte det derfor avgjøres hvor langs linjene kantene befant seg. Dette var det opprinnelig ønskelig å unngå, siden algoritmen dermed måtte løkke igjennom pikslene langs hver utplukkede linje i bildet, og dermed øke

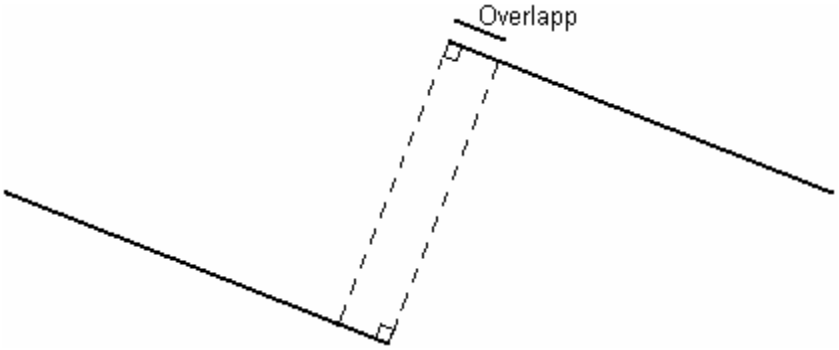
kjøretiden til algoritmen en del[22]. For å få en akseptabel funksjonalitet i denne egenskapsberegningen var det derimot nødvendig.

Metoden *houghlines* er en av MATLAB sine innebygde funksjoner og finner kantene med endekoordinater fra punktene uttrukket fra parameterbildet. Nå brukes kravet om minimumslengden til kantene. Kun kanter som overstiger denne lengden blir vurdert. Mange av kantene i bildene er avbrutt av manglende kantpiksler innimellom. Dette kan være kantpiksler som er plassert på nærliggende pikselrader, slik kanten i figur 52, eller det kan være trær eller lignende som stikker opp foran bygningen og gjør at den skarpe kanten langs mønet på en bygning blir brutt i noen piksler. Derfor er det ved identifisering av linjestykker tillatt brudd på fire sammenhengende piksler i kantene. Er det opphold på mer enn fire piksler, beregnes de to delene som hver sine kanter. Ved å tillate store brudd øker sannsynligheten for å anse løse bruddstykker som lange kanter, mens for små brudd vil sette for sterkt krav til kantene, som ofte ikke er like rette og sterke over det hele. Den valgte maksimumsgrensen på fire piksler for kantbrudd er en mellomting valgt etter gjennomgang av typiske kantbilder fra testdataene.

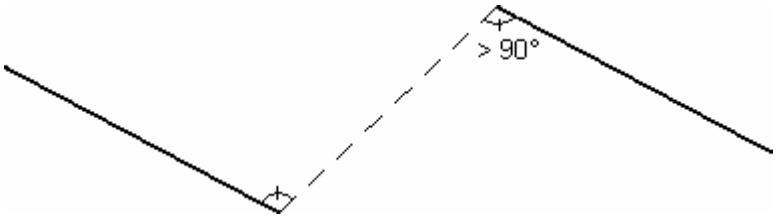
Etter at kantstykkene er identifisert utføres tester for hvorvidt kantpar kan representere trapeser i kantbildet. Siden det sjeldent lar seg gjøre å finne hele kanten langs bygningene, blir det ikke satt krav til at de parallelle kantstykkene skal være like lange. Dette fører til at kantene ikke fremstår som parallellogram i kantbildet, men som trapeser. For hver kant beregnes lengden av kanten som legges i en liste sammen med linjens θ-verdi. For hver ny kant som legges til lista testes den mot alle kanter i lista med samme θ-verdi, det vil si alle kanter som er parallelle med den nyeste. I tillegg utføres en test som sjekker om kantstykkene ikke inneholder for mange brudd. Tidligere ble det tillatt brudd på fire piksler eller mindre. Det finnes dog ingen grense for hvor mange slike brudd som kan finnes langs kantstykket. Dermed kan en kant teoretisk bestå av kun en kantpiksel i hver femte piksel. For å unngå at spredte småkanter skal settes sammen til lengre kanter sjekker denne metoden at minst en tredjedel av pikslene langs kanten er kantpiksler. Funksjonen som henter kantstykkene burde utført denne testen, men siden det her er valgt å bruke MATLAB sin funksjon til dette, er denne siste testen utført senere av en egen funksjon. For en eventuell effektivisering av koden ligger det her en mulighet til forbedring.

For å avgjøre hvorvidt de parallelle kantstykkene kan ansees som hver sin side av et trapes er det her valgt å sette krav til at linjestykkene skal overlappe hverandre i den retningen de strekker seg. Figur 54 viser to parallelle linjestykker som oppfyller kravet, mens figur 55 viser et par som ikke gjør det.

[22] Kjøretiden til denne delen er riktignok langt kortere enn Hough-transformasjonen, med den lave oppløsningen på de to parametrene der, men denne funksjonen var det akseptert som en nødvendighet for flere egenskaper og dermed lettere å godta.



Figur 54: Overlappende kantstykker



Figur 55: Ikke-overlappende kantstykker

Kantstykkene som ikke oppfyller kravet kan også utgjøre motsatte sider av et trapes, men det er ikke observert i løpet av testingen at de korteste endene av et parallellogram i bildet er plukket ut i kantbildet, uten at de lengste også er tatt med. Hvis det andre kantparet som utgjør parallellogrammet også er plukket ut, vil dette oppfylle kravet om overlapping, og parallellogrammet blir tatt med i beregningen. Det er derimot observert mange kantpar som er blitt luket ut av kravet og som heller ikke har representert noe motsatt kant av menneskeskapte objekter. Derfor har testen blitt betraktet som en god test.

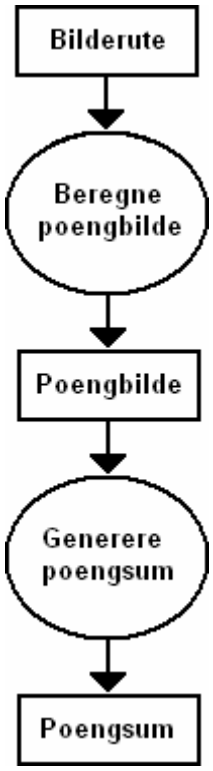
4.6.3 Kalkulasjon av poengsum for parallelle rette kanter

For de kantparene som ennå ikke er luket ut var det ennå noen som ikke tilhørte menneskeskapte objekter. De fleste av disse utmerket seg i forhold til de menneskeskapte parallellogrammene ved å være mye mindre ensfargede mellom kantstykkene. De fleste parallellogrammene som besto av bygningsflater som tak og vegger hadde en relativt uniform gråtone mellom kantene. Derfor ble det lagt vekt på disse egenskapene når det skulle genereres en poengsum. For hvert kantpar som kunne representere parallellogram ble det beregnet variansen og arealet til parallellogrammet kantene utgjorde. Parallellogram med varians mindre enn 12 regnes som forholdsvis uniforme, og dermed også sannsynligvis deler av menneskeskapte objekter. Dette skillet mellom uniforme og lite uniforme parallellogrammer er ikke en sikker metode. Bygningsflater kan som tidligere nevnt være dekket av busker og trær foran deler, noe som vil trekke variansen opp. I tillegg kan vinduer og dører på veggene skille seg ut med forskjellige gråtoner enn resten av veggen, og dermed bidra til å øke variansen. Derfor er grensen på 12 ganske høy, og godtar mye støy på flatene. Parallelle kanter ute i naturen gir ofte enda større varians.

Poengene til ruten gis ut ifra en funksjon som favoriserer parallellogram med store arealer og liten varians. Poengsummen er også skalert etter hvor mange trapes som er funnet. Testing har vist at det lønner seg å favoriseres få men store fremfor mange små.

4.7 Uniforme arealer ved rette linjer

Den siste egenskapsberegningen gir, som beskrevet i kapittel 3.4, poeng for kanter som har uniforme områder på minst en side. I *Main*-skriptet er den plassert etter de tre andre egenskapsberegningene, under overskriften *Uniform areas near edges*. Som figur 56 viser, består beregningene av to deler. Først beregnes det en poengsum for hver piksel i ruten. Deretter genereres det en felles poengsum for hele ruten på grunnlag av dette poengbildet. De to delene er beskrevet i hvert sitt underkapittel.



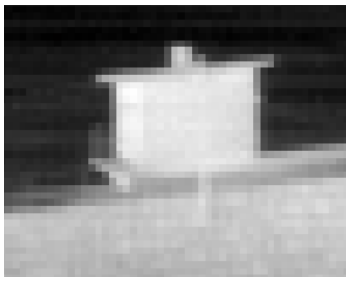
Figur 56: Dataflytdiagram for beregning av poengsum for uniforme arealer ved rette linjer

4.7.1 Poenggivning av hver piksel

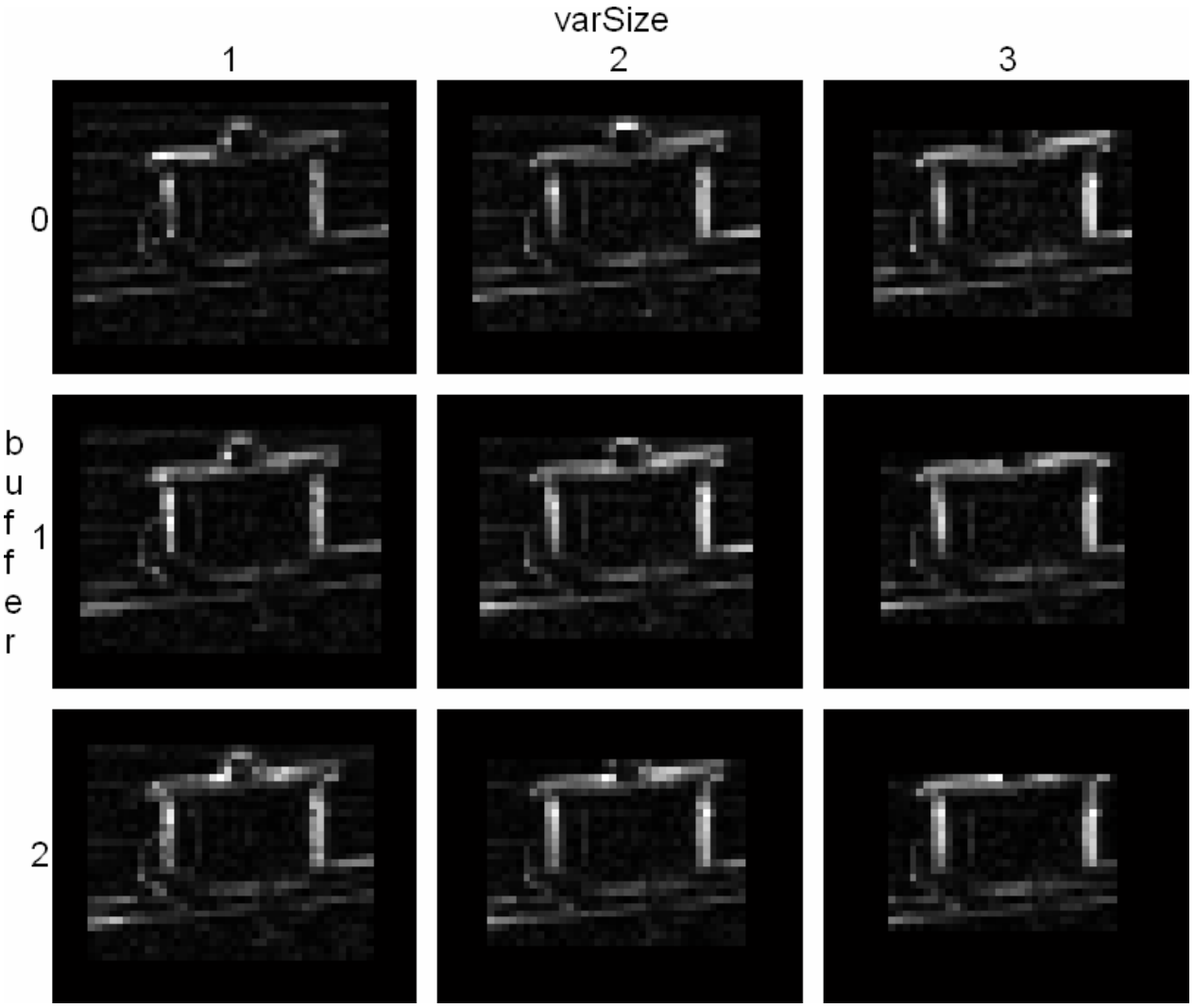
På bakgrunn av ruten fra bildet, størrelsen på arealet variansen skal beregnes i, og avstanden mellom kanten og dette arealet, beregnes det en poengsum for hver piksel i bildet. Poengsummen øker med styrken på kanten og avtar med variansen i området ved siden av kanten. Den nevnte størrelsen på arealet variansen skal beregnes i, og avstanden mellom kanten og dette arealet kalles henholdsvis *varSize* og *buffer* i koden.

Verdien på variablene *varSize* og *buffer* er her satt til henholdsvis 2 og 1 (hvilket er det samme oppsettet som vist i figur 34, bare med 5x5 piksler stort arealet i stedet for 3x3). Dette gir et brukbart resultat i de fleste bilder. For større verdier av hver av de to parametrene blir små objekter på bakken oversett. Dette kommer av at arealene fort strekker seg ut på motsatt side av objektene, slik at variansen også medberegner enkelte piksler fra bakgrunnen på andre siden av objektet. Oppsettet fungerer for objekt større enn sju piksler i bredden, hvilket her ansees som et minstekrav for å kunne klassifisere arealet som uniformt. Ved anvendelse av algoritmen på andre data, bør disse verdiene vurderes på nytt.

Figur 58 viser poengbildet for forskjellige verdier av de to parametrene, beregnet på bakgrunn av originalen i figur 57.



Figur 57: Originalbildet til testene i figur 58



Figur 58: Poengbilder for forskjellige verdier av parametre

Som det kan sees av pipen på hytta, blir mindre objekter borte ved større verdier for parametrene, mens mindre størrelse på området variansen regnes i gir mer utslag av støy rundt omkring i bildet.

I funksjonen foregår beregningen ved at det først beregnes et variansbilde, som har samme antall elementer som ruten. Hvert element i variansbildet gir variansen i ruten på den tilsvarende posisjonen. Med den tilsvarende posisjon menes arealet gitt av operatøren med sentrum i samme pikselposisjon i ruten som i variansbildet. Når det siden trengs variansen i et område, er det bare å slå opp i variansbildet på senterpikselen. Etter kalkulasjonen av variansbildet beregnes vertikale og horisontale kanter i hvert sitt kantbilde ved hjelp av derivasjon i de to retningene. Så beregnes poeng til pikslene i bildet ved å dividere kantstørrelsen i pikselen med variansen på den siden av kanten som har minst varians.[23] Dette gjøres for både vertikale og horisontale kanter, og pikselens poengsum blir den høyeste av de to nevnte.

4.7.2 Kalkulering av poengsum for uniforme arealer ved kanter

Siden det skal beregnes en felles poengsum for ruten, konverteres poengbildet til én poengsum. Som konverteringsmetode er det valgt å regne gjennomsnittet av poengene til de 10 % av piksel med høyest poengsum. Dette gir en god poengsum til ruter med få men gode ”uniformt område ved kant”-egenskaper, og dårlig poengsum til ruter med mange men dårlige slike egenskaper. Derfor regnes dette som en passende konvertering. Siden utplukkingen av de beste pikslene fra poengbildet medfører en sortering av alle pikslene er konverteringen nokså resurskrevende i forhold til resten av beregningen. Med tanke på dette kan en annen konvertering vurderes ved effektivisering av koden.

4.8 Klassifisering på bakgrunn av egenskapspoengene

Etter at hver rute i bildene har fått en poengsum for hver av egenskapsalgoritmene, må det avgjøres hvilke ruter som skal bli tatt vare på og hvilke som skal forkastes som uinteressante. Med poengsummer fra forskjellige algoritmer er det flere metoder for å komme til en slik konklusjon.

En metode er å sette en terskel for hver av egenskapene og karakterisere alle rutene med minst én fremtredende egenskap som interessant. Dette vil sikre at alle rutene som er bevart inneholder minst én egenskap som er forbundet med menneskeskapte objekter. Den vil derimot ikke ta vare på ruter med antydninger til flere gode egenskaper, uten å utmerke seg spesielt på én enkelt.

En annen mulighet som tar vare på slike tilfeller, er å legge sammen poengene for alle de forskjellige egenskapene og sette et minimumskrav til summen.

I tillegg til disse metodene finnes alle mellomting der enkelte egenskaper kan behandles hver for seg, mens andre må inntreffe samtidig for å gi ruten klassifiseringen interessant.

Hvilken metode som skal velges avhenger av hvordan egenskapene inntreffer. Hvis to egenskaper hver for seg opptrer i så vel uinteressante som interessante ruter, men kun samtidig i interessante

[23] For å unngå å dele på null, er alle variansene økt med én.

ruter, bør disse poengsummene sees i sammenheng under klassifiseringen. Skulle en egenskap derimot opptre helt uavhengig av de andre, bør denne behandles for seg selv. Valget av metode bør derfor avgjøres etter å ha sett hvilke egenskaper som opptrer hvor i bildene.

Testing av algoritmene på de gitte testdata viser at egenskapene i denne rapporten sjeldent overstiger hver sine poengterskler uten at rutene inneholder interessante objekter. Disse tersklene er vist under testingen av algoritmene i kapittel 5. Ved sammenligning av poengsummene for de forskjellige egenskapene til de samme rutene er det ingenting som tyder på at noen av disse bør kombineres. Dette er gjort og utførlig forklart i kapittel 5.7. Derfor velges det å bruke den valgte terskel for hver egenskap og klassifisere ruter med poengsum over terskelen i minst én egenskap, som interessante.

Kapittel 5: Testing

I dette kapitlet vil de forskjellige algoritmene bli testet. Testene skiller ikke mellom godkjent og underkjent funksjonalitet i algoritmene. De har til hensikt å vise hvordan algoritmene håndterer forskjellige testdata. Det vil så være opp til enhver å avgjøre hvorvidt dette er godt nok til forskjellige anvendelser av algoritmen.

Først testes algoritmen for oppdeling av bildet. Siden denne algoritmen skiller seg fra de andre algoritmene (den har ikke til formål å evaluere en rute etter dens egenskaper) vil også testingen være noe annerledes. Siden ruteinndelingen ikke behøver å være nøyaktige i forhold til største mulige observerte objekt, vil det ikke bli regnet ut om rutestørrelsene er korrekte. Det vil derimot bli sett på hvorvidt ruteinndelingen ser riktig ut i forhold til innholdet. Det vil si at menneskeskapte objekter omfattes fullstendig i minst én rute.

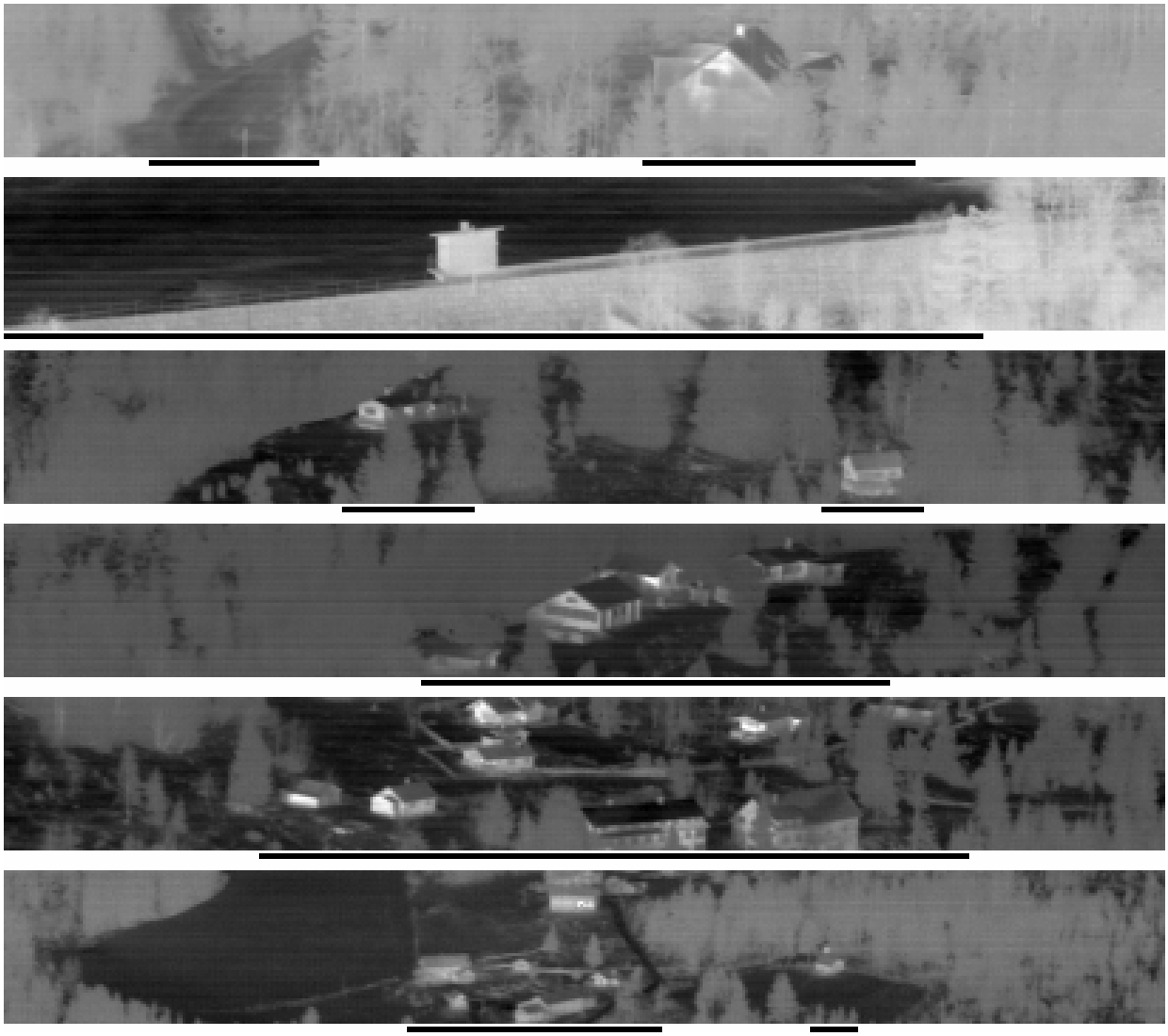
Deretter vil algoritmene for egenskapsidentifisering testes hver for seg. Testingen av hver algoritme er delt i tre deler, der de blir testet på bilder med forskjellige egenskaper. De tre delene er presentert i de neste avsnittene.

Først testes algoritmene på tegnede ideelle bilder som skal vise hvordan algoritmen gir poeng for den aktuelle egenskapen. Bildene skal vise objekter med de ønskede egenskapene og objekter uten disse. Deretter forventes det at algoritmene gir bedre poengsum til objektene med de ønskede egenskapene. Dette har til formål å vise at algoritmen fungerer som tenkt gjennom utviklingen. Siden de tegnede bildene er vesentlig forskjellig fra vanlige testbilder, er de parametrene i algoritmen som er ment oppgitt av en operatør satt noe annerledes. Dette gjøres for at algoritmen skal passe til bildene, både i format og i gråtoneverdier. Dette gir igjen poengsummer som ikke lar seg ukritisk sammenligne med poengsummer fra reelle testbilder.

Deretter vil algoritmene bli testet på egnede testbilder fra testdataene. Disse skal vise reelle objekter med de ønskede egenskapene. Algoritmene er testet på mange bilder, men siden det ikke har noen hensikt å vise alle testene her, er det plukket ut bilder hvor algoritmene viser en uventet eller uønsket oppførsel. Det vil også bli plukket ut reelle testbilder med forventet resultat, der dette vil gi meningsfullt sammenligningsgrunnlag. Siden det her blir plukket frem enkeltbilder der algoritmen feiler, kan leseren sitte igjen med et inntrykk av at algoritmen ikke er brukbar. Det er da viktig å huske på at bildene er plukket ut fra tusenvis av testbilder fordi de akkurat feilet i disse tilfellene. Målet med denne testingen er å vise svakheter og begrensninger ved algoritmene.

Siden de forrige testene har fokusert på å vise egenskaper ved algoritmene, og derfor håndplukket testbilder for å vise dette, er ikke algoritmenes generelle brukbarhet kommet godt frem. Det vil derfor i denne siste testdelen vises hvordan de forskjellige algoritmene fungerer på mer vanlige tilfeldige testdata. Testbildene i denne delen vil være ett uforandret sett med bilder fra testdataene, og dermed ikke tilpasset hver algoritme. Utplukkingen av disse bildene kunne ideelt sett vært trekt tilfeldig for å sikre et representativt utvalg blant alle testbildene. Dette ville med stor sannsynlighet ha resultert i kun bilder uten menneskeskapte objekter, siden disse utgjør det store flertall av bildene. Det er derimot bildene som inneholder slike objekter som er best egnet for å vise algoritmenes funksjonalitet. Derfor er det plukket ut tilfeldige bilder med en god blanding av menneskeskapte objekter og bakgrunn. De av bildene som inneholder

menneskeskapte objekter er vist i figur 59 og figur 60.[24] Under hvert bilde er det markert med tykke svarte streker hva som på forhånd er manuelt karakterisert som menneskeskapte objekter. Under testingene er det ønskelig at disse blir bevart av algoritmene. Det er ikke viktig at alle algoritmene plukker ut alle objektene, men når algoritmene slåes sammen bør disse objektene plukkes ut.

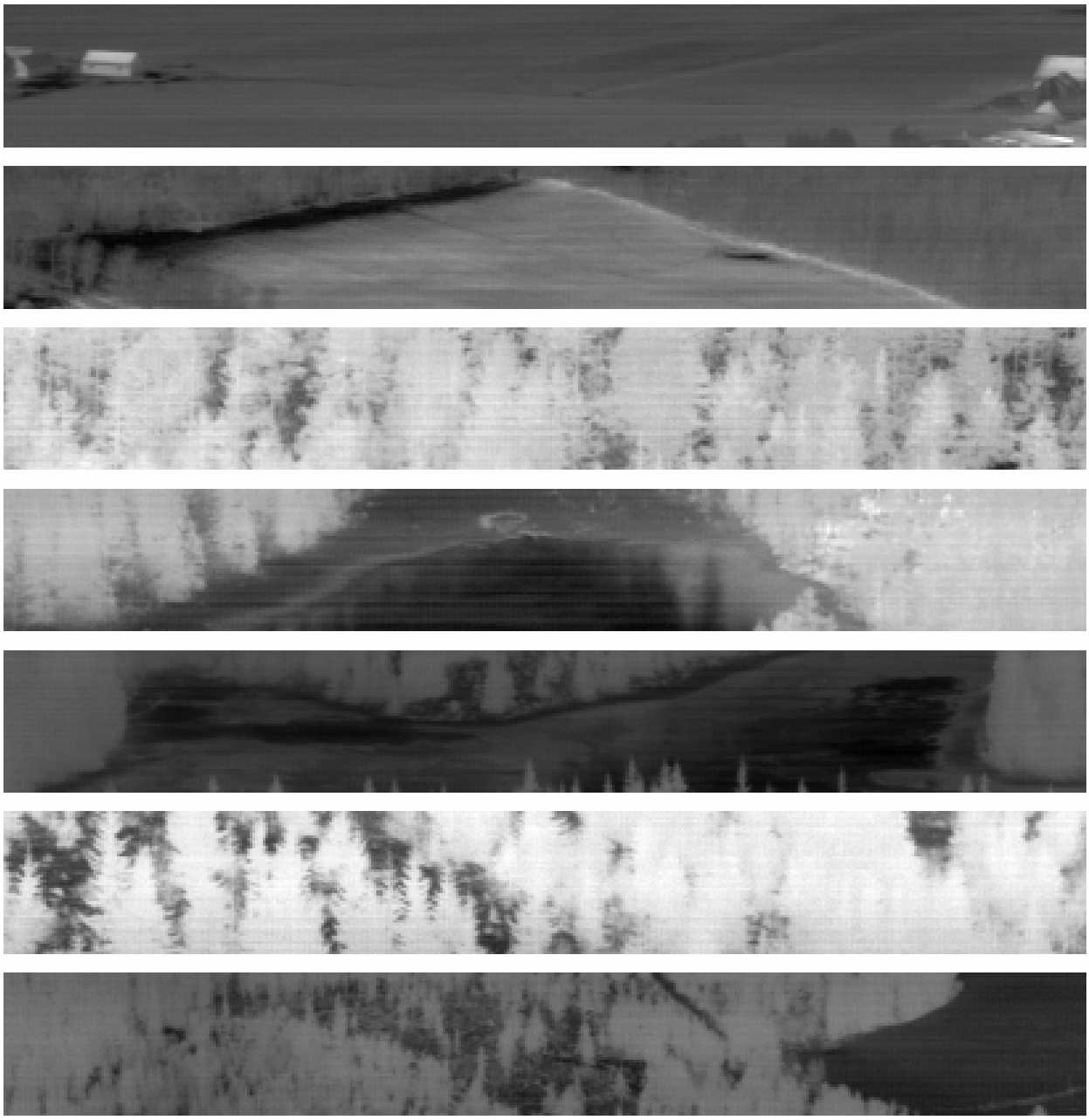


Figur 59: Testsett med menneskeskapte objekter (del 1)

[24] Grunnen til at testsettet er delt opp i deler er kun for bedre å kunne passe de inn på sidene.

Figur 60: Testsett med menneskeskapte objekter (del 2)

Resten av bildene inneholder typiske utklipp med naturlig bakgrunn. I disse bildene er det ikke ønskelig at noen områder bevares, foruten bygningene på hver side av bildet med jordet. Disse bildene er vist i figur 61.



Figur 61: Testsett uten menneskeskapte objekter

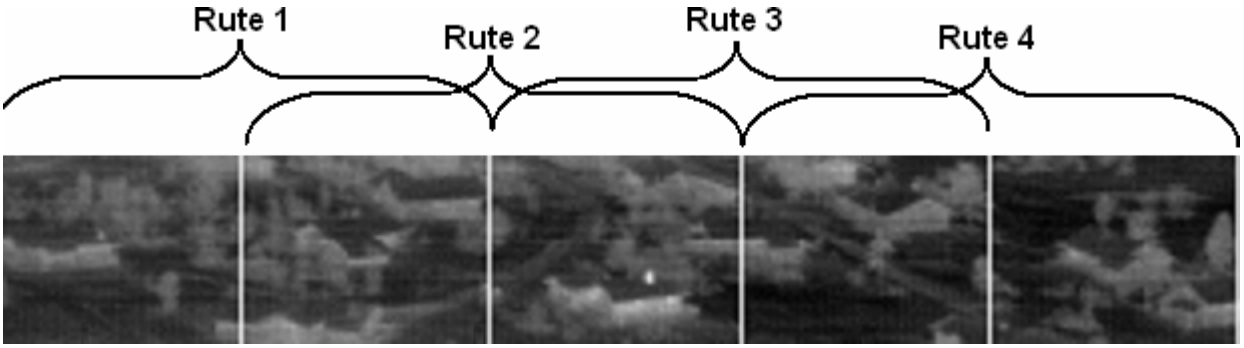
For å unngå å gjenta alle bildene i alle de fem delkapitlene, er de vist i vedlegg C sammen med poengsummer for hver av algoritmene. Det henvises derfor til dette vedlegget i den tredje delen av hver algoritmetest.

Etter testingen av algoritmene hver for seg vil de testes sammen, og det blir utviklet en passende vekting av egenskapspoengene for best å trekke ut interessante områder. Testingen blir foretatt på det samme settet av bilder som vist i figur 59, figur 60 og figur 61. Systemet vil da vise hvordan det fungerer som en enhet, og bør da bevare de menneskeskapte objektene som antydet i figur 59 og figur 60. Det blir lagt vekt på hvordan egenskapene fungerer i kombinasjon.

5.1 Test av oppdelingen av bildet

Som tidligere nevnt er ikke oppdelingen noen svært nøyaktig operasjon. Hovedmålet er at rutene skal være så små som mulig, men at alle menneskeskapte objekter i sin helhet skal falle innenfor en rute. Dette tar utgangspunkt i at objektstørrelsen ikke overstiger den maksimale grensen som her er anslått til 30 meter. For objekter større enn dette må hver del av objektet være innenfor en rute som inneholder store nok deler av objektet til at egenskapene som kjennetegner de menneskeskapte objektene fremgår i ruten. Dette er fordi hver del av objektet skal være inkludert i en rute som plukkes ut som interessant, og dermed bevares av algoritmen. Fokuset i testingen her har derfor vært å se om objektene faller innenfor minst én rute.

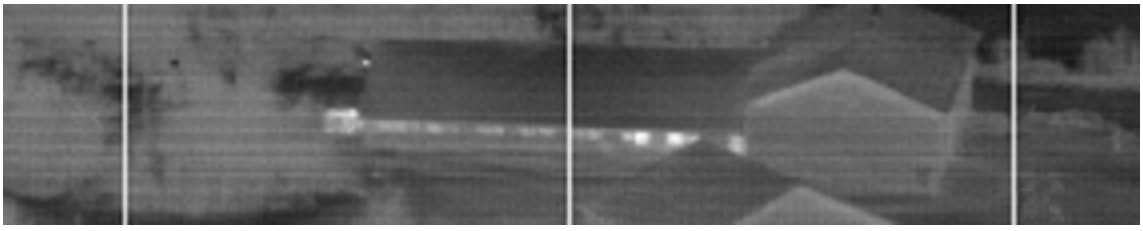
Figur 62 viser et utklipp fra et testbilde. Ruteinndelingen er tegnet inn i bildet ved hvite grenser. Siden rutene har 50 % overlapp er høyre kant av en rute ikke den førstkommende, men den andre kanten til høyre. Over bildeutklippet er dette antydet med klammer som viser rutebredden.



Figur 62: Bildeoppdeling av hyttefelt

Som figuren viser faller alle hyttene i sin helhet innenfor minst én rute. Dette er ikke noe problem med så små bygninger som her.

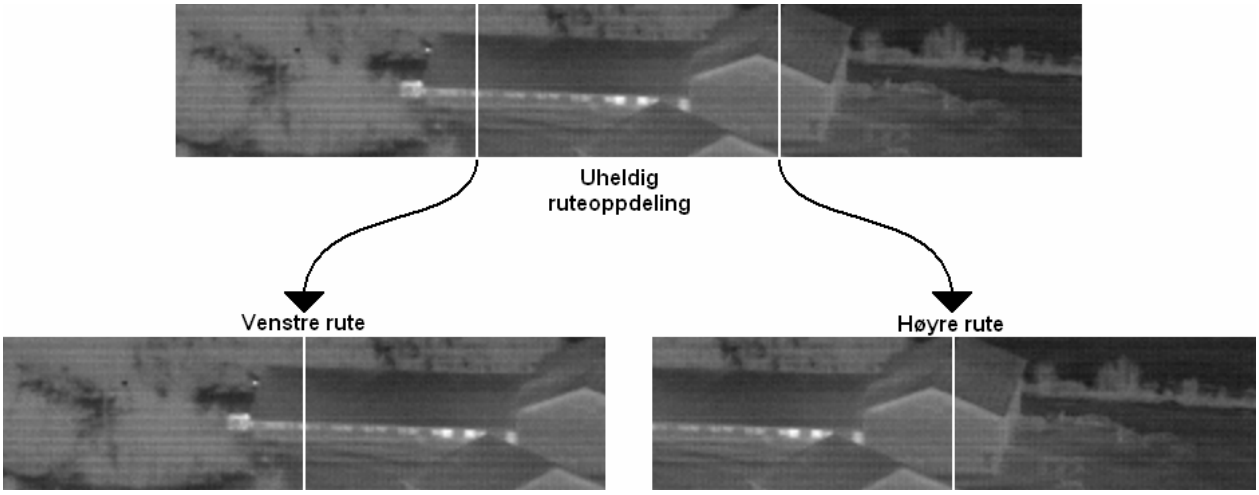
Figur 63 viser en større bygning. Siden bygningen strekker seg over mer enn en halv rute (det vil si avstanden mellom to hvite streker), kan det anes at denne bygningen er noe over 30 meter lang.



Figur 63: Bildeoppdeling av stor bygning

Allikevel passer bygningen fint inn i rutene. Selv om bygningene skulle være noe større enn estimert største størrelse, må de være plassert uheldige i forhold til ruteoppdelingen for at denne skal forårsake at objektet ikke blir fullstendig bevart. Figur 64 viser en maksimalt uheldig oppdeling av dette bildet med den utregnede rutebredden. Her er bygningen blitt delt opp slik at ingen ruter inneholder hele bygningen. Allikevel ser man av de to rutene at begge burde

inneholde nok egenskaper til at begge rutene blir klassifisert som interessante, og dermed hele bygningen blir bevart.

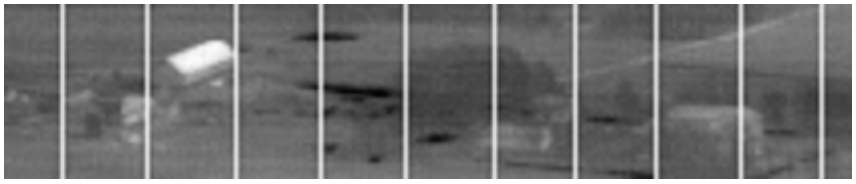


Figur 64: Uheldig bildeoppdeling av stor bygning

En ulempe ved den uheldige oppdelingen i figur 64 er at de to rutene vil inneholde mye bakgrunn på hver side av den store bygningen. Denne bakgrunnen vil også bli bevart, noe som gir litt ekstra unyttig bildedata etter anvendelsen av algoritmen.

Siden bildeoppdeling skal endre seg med avstanden til objektene i bildene, er det i figur 65 og figur 66 vist rutebredder i bilder med henholdsvis ca 0,5 og 2,5 kilometers avstand bygningene. Begge rutestørrelsene ser fornuftige ut i forhold til at rutene skal dekke ca. 60 meter i bredden på bakken (som tilsvarer ca 30 meter mellom hver av de hvite linjene).

Figur 65: Bildeoppdeling av bilde med store objekter



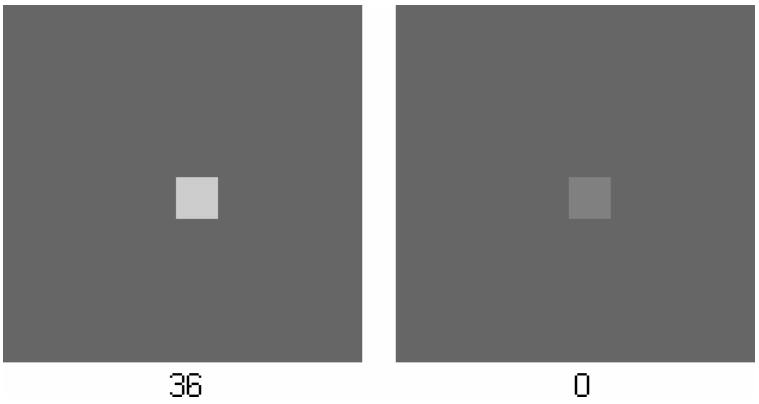
Figur 66: Bildeoppdeling av bilde med små objekter

5.2 Test av poenggivning for lyse områder definert med statisk terskel

Under følger tester av algoritmen for poenggivning av lyse områder, gitt den statiske terskelen. Det blir brukt testbilder fra de tre kategoriene tegnede bilder, egnede testbilder og andre testbilder, i hvert sitt underkapittel.

5.2.1 Tegnede bilder

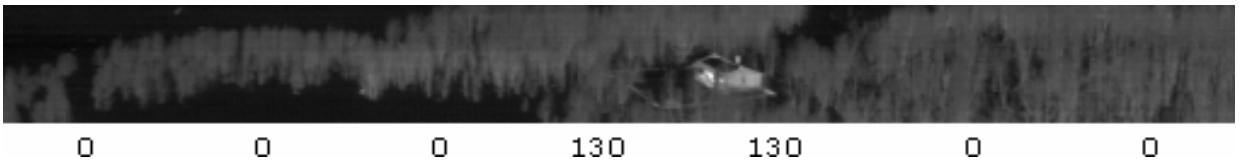
Figur 67 viser to bilder. Bildet til venstre inneholder en mørk bakgrunn med en lys firkant, mens bildet til høyre viser en ikke fullt så lys firkant. Den lyse firkanten har en verdi over terskelen, mens den ikke fullt så lyse firkanten til venstre har verdi under terskelen. Begge firkantene har en størrelse på 36 piksler. Under firkantene står poengsummene som bildene fikk av algoritmen med statisk terskling. Som forventet fikk hvert bilde like mange poeng som de hadde piksler med verdi over terskelen.



Figur 67: Tegnede testbilder og poengsummer for lyse områder definert med statisk terskel

5.2.2 Egnede testbilder

Figur 68 viser et hus omgitt av skog. Som poengsummene under viser, er disse en god indikator på hvor i bildet det finnes menneskeskapte objekter.

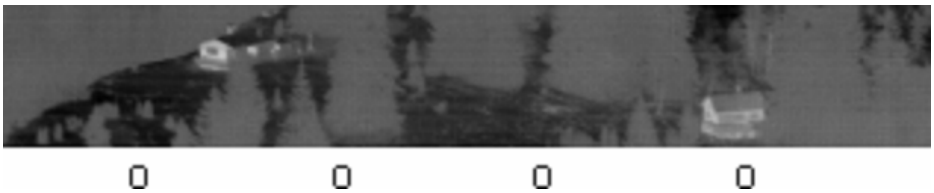


Figur 68: Vanlig testbilde med poengsummer for lyse områder definert med statisk terskel

I tillegg til at det svært ofte er lyse piksler på de menneskeskapte objektene, er det også sjeldent dette forkommer i bakgrunnen. Enkelte unntak finnes selvfølgelig. Figur 69 viser hvordan det oppvarmede fjellpartiet nevnt i kapittel 3.1.2 (se figur 24) overskrider grensen.

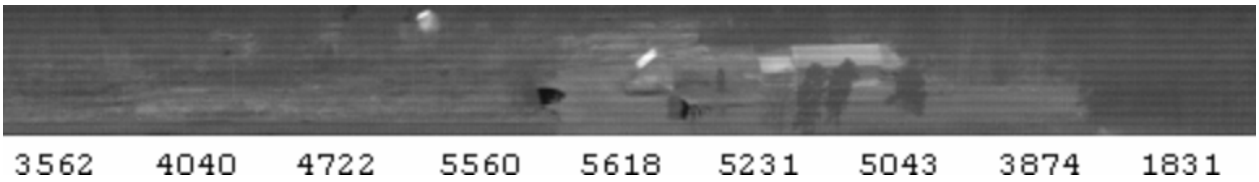
Figur 69: Testbilde av varme fjellpartier og poengsummer for lyse områder definert med statisk terskel

Figur 70 viser det motsatte tilfellet, der en bygning ikke inneholder noen lyse piksler. Dette forekommer som oftest under overskyete forhold.



Figur 70: Testbilde av hus uten lyse områder, og poengsummer for lyse områder definert med statisk terskel

Ett annet uheldig tilfelle er områder der oppvarmingen har vært så uvanlig stor at nesten alle objekter, menneskeskapte som naturlige, stråler mer enn terskelen. Figur 71 viser et slikt bilde, der nesten alle pikslene overstiger grensen. Siden bildet er normalisert til verdier mellom 0 og 1 for fremvisning, kan det ikke sees av bildet at det generelle nivået på pikselverdiene er uvanlig høyt. Poengsummene gir derimot en klar indikasjon.



Figur 71: Testbilde med generelt høye verdier, og poengsummer for lyse områder definert med statisk terskel

Slike tilfeller som i figur 71 forekommer svært sjeldent i testdataene og representerer ikke noe stort problem hvis de skulle bli inkludert i de bevarte dataene. Skulle det derimot gjøres opptak der de generelle lysverdiene varierer mye, kan dette problemet være avgjørende for algoritmens brukbarhet. En mulig forbedring vil være å benytte en dynamisk terskel.

5.2.3 Andre testbilder

Den øverste poenglinjen under bildene i vedlegg C viser hvordan denne algoritmen ga poeng til rutene i bildene i testsettet. Beskrivelsen under viser til disse bildene.

Som de to første delene av testsettet viser har de aller fleste bygningene varme punkter som gjør at de får poeng. Noen få bygninger lekker derimot ikke nok varme til å skille seg ut med denne metoden, slik som tidligere vist. Noen flere eksempler enn det som er trukket frem i kapittel 5.1.2, er vist i det tredje bildet ovenfra i 2. del av testsettet. Andre menneskeskapte objekter, som veier og demningen har ikke fått noen poeng. Dette er som forventet, siden disse ikke er typiske oppvarmede objekter.

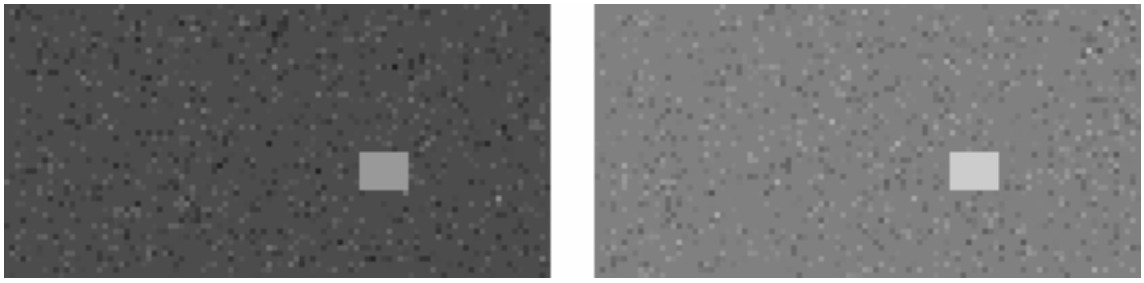
I den siste delen av testsettet ser man at det sjeldent forekommer varme punkter utenom menneskeskapte objekter. Kun fem punkter på det tredje øverste bildet i den tredje delen av testsettet kommer over den valgte grensen. Ellers gir ikke algoritmen noen poeng til naturlige områder i dette testsettet.

5.3 Test av poenggivning for lyse områder definert med dynamisk terskel

Under følger tester av algoritmen for poenggivning av lyse områder, gitt den dynamiske terskelen. Det blir brukt testbilder fra de tre kategoriene tegnede bilder, egnede testbilder og andre testbilder, i hvert sitt underkapittel.

5.3.1 Tegnede bilder

For å teste en dynamisk terskel trengs det en serie bilder, slik at terskelen kan bygge seg opp på statistiske egenskapene til flere bilder. Fordelen med en dynamisk terskel er som nevnt tidligere at den tilpasser seg det generelle lysnivået i bildene, og plukker ut objekter som utmerker seg i forhold til dette. For å se at den dynamiske terskelen har denne egenskapen har den blitt testet på en bildeserie bestående av en rekke med bildet til venstre i figur 72, etterfulgt av en rekke med bildet til høyre. Firkantene i nedre høyre kvartdel av bildene er like store, nærmere bestemt 63 piksler. Gråtoneverdien i firkanten i bildet til venstre er den samme som gjennomsnittsgråtonen i bakgrunnen i bildet til høyre. Firkanten i det høyre bildet skiller seg like mye fra bakgrunnen i gråtoneverdi som firkanten i bildet til venstre. Standardavviket er satt slik i forhold til støyen i bildene at disse firkantene skal skille seg ut som objekter over terskelen. Dermed er målet at den dynamiske terskelen skal skille ut firkanten i bildet til venstre når terskelen er tilpasset dette lysnivået. Deretter gis algoritmen plutselig bilder lik bildet til høyre. Den dynamiske terskelen skal da tilpasse seg det nye lysnivået og etter hvert skille ut den nye firkanten.



Figur 72: Tegnede testbilder for test av dynamisk terskel

Da terskelen hadde tilpasset seg nivået i bildet til høyre fikk dette en poengsum på 64. Grunnen til at denne summen var høyere enn antallet piksler i firkanten er at en av støypikslene også overstiger denne verdien. Figur 73 viser hvordan poengsummen utviklet seg etter at bildet ble byttet ut med bildet til høyre i figur 72. Den første poengsummen i figur 73 tilhører det siste bildet av typen til venstre i figur 72, mens resten av poengsummene tilhører bilder av typen til høyre i figuren.

Bildenr.	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7	n+8
Type	V	H	H	H	H	H	H	H	H
Poengsum	64	337	234	154	79	74	69	68	65

Bildenr.	n+9	n+10	n+11	n+12	n+13	n+14	n+15	n+16	n+17
Type	H	H	H	H	H	H	H	H	H
Poengsum	65	65	65	65	65	64	64	64	64

Figur 73: Poengsummer for test av dynamisk terskel

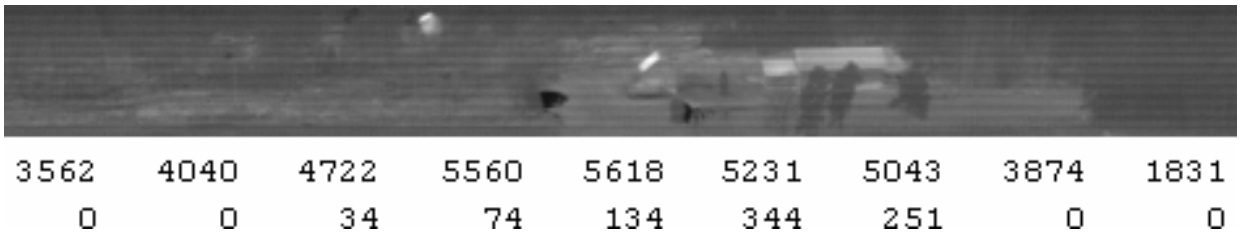
Som poengsummene viser, tilpasser terskelen seg lysnivået og skiller til slutt kun ut den lyse firkanten (i tillegg til én lys piksel i støyen). At terskelen etter bare 6 bilder kun skiller ut 6 piksler foruten den lyse firkanten, viser at terskelen har tilpasset seg nivået veldig fort. Dette er avhengig av valgte antall standardavvik og nivået på støyen i disse testene. De er her valgt for å passe dette eksempelet, slik at testen best mulig skal vise den dynamiske terskelens egenskaper. Derfor vil dette tatt flere bilder i virkelig testdata, men der ville heller ikke overgangen ha vært så markant som skillet mellom bildene i figur 72.

Før terskelen har justert seg vil den dynamiske terskelen plukke ut alle rutene i lik grad som en statisk terskel tilpasset det forrige lysnivået. Som for bildet rett etter overgangen og noen påfølgende bilder vil trolig alle rutene få poengsum over null. Dette gjør at alle rutene plukkes ut som interessante. Dette er ikke så farlig i og med at dette ikke hindrer menneskeskapte objekter fra å bli inkludert i de bevarte data. Som tidligere nevnt er det viktigst å inkludere alle menneskeskapte objekter, for deretter å gjerne så mye som mulig av de gjenværende områdene. Derimot vil det bli et problem der det generelle strålingsnivået synker i stedet for øker slik tilfellet er i denne testen. Da vil det i overgangen ikke bli plukket ut noen ruter og eventuelle menneskeskapte objekter går tapt fra dataene. Dette er ikke ønskelig, men vanskelig å unngå. En statisk terskel ville ikke fungert bedre, i og med at denne bare kunne vært tilpasset ett av de to strålingsnivåene. En dynamisk terskel har derimot evnen til å tilpasse seg etter hvert og fungere etter en liten overgangsperiode der terskelen justeres.

Denne testen viser at den dynamiske terskelen har den ønskede funksjonen som tiltenkt under utviklingen.

5.3.2 Egnede testbilder

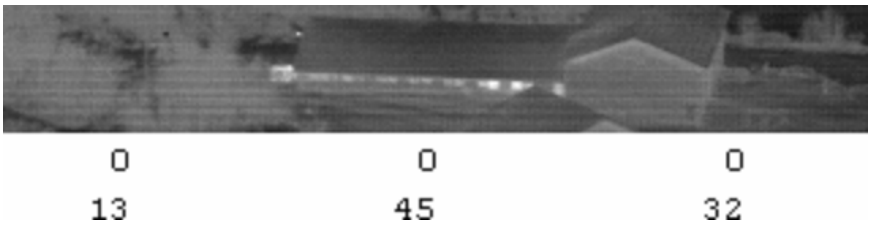
Siden den statiske terskelen feilet på bildeserien der visse områder hadde et altfor varmt gjennomsnittstemperatur (se figur 71) er det naturlig å teste den dynamiske terskelen på dette opptaket. Figur 74 viser det samme bildet som figur 71, med poengsummene fra den statiske terskelen øverst og poengsummer fra den dynamiske terskelen under.



Figur 74: Testbilde med generelt høye verdier, og poengsummer fra statisk (øverst) og dynamisk nederst) terskling

Som poengsummene viser har den dynamiske terskelen undertrykt den generelle endringen i varmeforholdene, og plukker her ut rutene med menneskeskapte objekter.

I tillegg til tilfellene der det generelle lysnivået er for høyt er det også interessant å se om den dynamiske terskelen tilpasser seg mørkere områder også. Testopptaket tatt i overskyet vær viste noe mangelfull identifisering av lyse områder. Figur 75 viser hvordan den dynamiske terskelen har tilpasset seg lave strålingsforhold og plukker ut menneskeskapte objekter der den statiske terskelen feiet.



Figur 75: Testbilde med generelt lave verdier, og poengsummer fra statisk (øverst) og dynamisk nederst) terskling

Det ble gjennom testdataene ikke tilfeller der den statiske terskelen fungerte bedre enn den dynamiske. Den dynamiske terskelen klarte allikevel ikke å plukke ut alle de ønskede lyse feltene. Ett eksempel er bildet i figur 70, der den dynamiske terskelen heller ikke gav noen poeng til noen av rutene.

5.3.3 Andre testbilder

Den andre poenglinjen under bildene i vedlegg C viser hvordan denne algoritmen ga poeng til rutene i bildene i testsettet. Beskrivelsen under viser til disse bildene.

De to første delene av testsettet viser at denne terskelen også gir poeng til nesten alle bygningene. Som for den statiske terskelen er andre menneskeskapte objekter oversett, noe som ikke er uventet siden det for det meste er bygninger som er varmet opp og dermed stråler mer varmestråling. Det er lett å følge sammenhengen mellom strekene som viser de manuelt identifiserte menneskeskapte objektene, og poengene større enn null. Dette er et tegn på at den tidligere nevnte grensen på minst ett poeng kan indikere menneskeskapte objekter i bildene. Sammenlignet med testen av statisk terskel på de samme bildene, går det frem at den dynamiske terskelen plukker ut de samme bygningene og andre bygninger, men ikke flere naturlige områder. Dette gjør den dynamiske terskelen best egnet, og den statiske terskelen overflødig så lenge den dynamiske er i bruk.

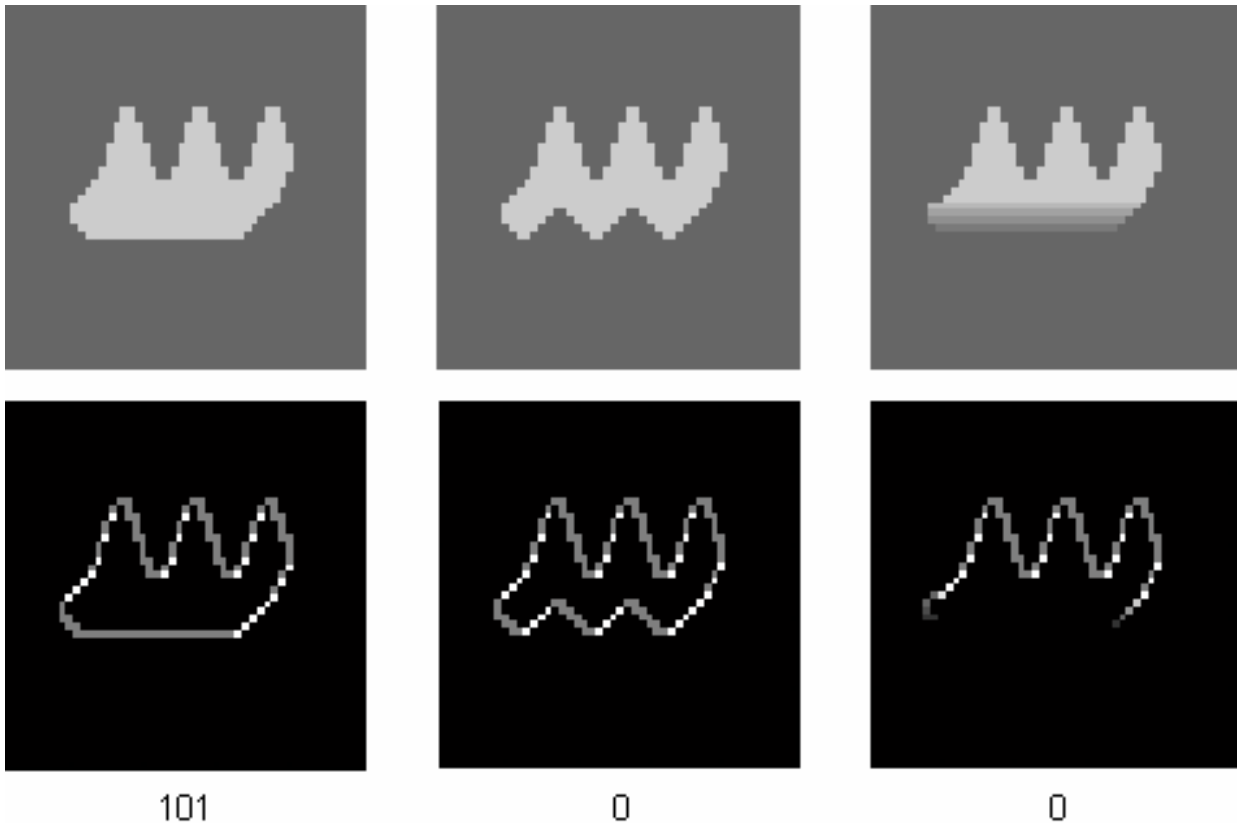
Den tredje delen viser ingen overraskelser. Kun rutene ytterst i det øverste bildet har poengsum over null, og det er akkurat disse rutene som inneholder menneskeskapte objekter.

5.4 Test av poenggivning for markante rette kanter

Under følger tester av algoritmen for poenggivning av markante rette kanter, med testbilder fra de tre kategoriene tegnede bilder, egnede testbilder og andre testbilder.

5.4.1 Tegnede bilder

Figur 76 viser tre tegninger som er testet i algoritmen. Under hvert bilde vises tilhørende kantbilde etter at svake kanter er fjernet. Under kantbildene igjen, står poengsummen bildene fikk.



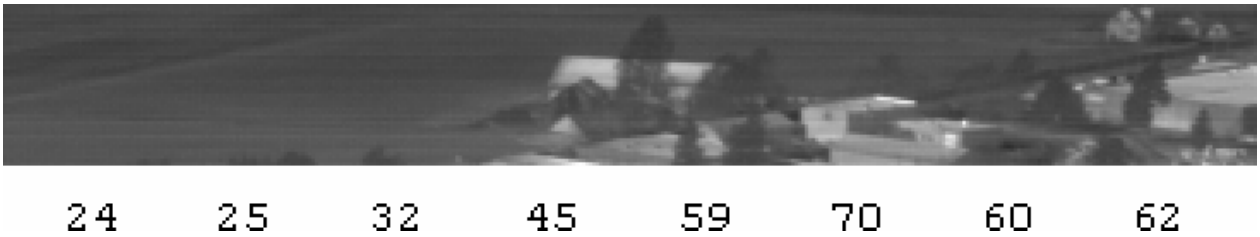
Figur 76: Tegnede testbilder med kantbilder og poengsummer for markante rette kanter

Som figuren viser får bildet med en markant rett underkant en poengsum over null. Bildet i midten, som ikke har noen rette kanter får ingen poeng. De korte skrå kantene i underkanten av objektet på bildet, er ikke lange nok til å bli medregnet. Senkes denne parameteren slik at disse kommer med i beregningen øker poengsummen til 85. Bildet helt til høyre har en rett underkant, men denne er ikke markant. Dermed får bildet heller ingen poeng. Som det tilsvarende kantbildet viser er underkanten ikke markant nok til å komme med.

Altså må objektene ha kanter som både er rette og markante for å få poeng.

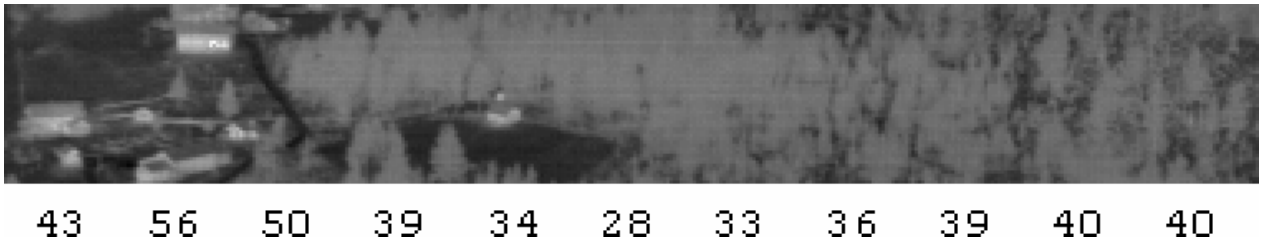
5.4.2 Egnede testbilder

Figur 77 viser et jorde til venstre og bygninger til høyre. Under bildet står poengsummen til ruten med senter rett over tallet.



Figur 77: Vanlig testbilde med poengsummer for markante rette kanter

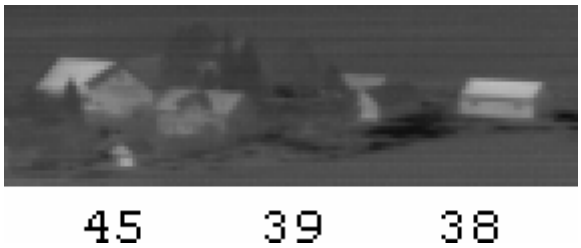
Som figuren antyder, og testing på flere bilder viser, holder egenskapens poengsum seg som regel mellom 10 og 30 for områder uten menneskeskapte markante rette kanter. Dette kommer av forskjellige tilfeldige kanter i bakgrunnen ellers. Når det dukker opp bygninger slik som i figur 77 øker poengsummen til over 40. Figur 78 viser et utklipp fra et annet bilde. Denne figuren viser at normal bakgrunn kan få poengsum opp til 40. Bygningene til venstre i bildet får derimot godt over 40. Etter gjennomgang av bildene i testdataene virker 47 som et passende skille mellom poengsummer til menneskeskapte objekter og bakgrunn. Denne grensen er foreslått på bakgrunn av alle testopptakene. De forskjellige opptakene har noe forskjellige egenskaper, så en tilpasset grense til hvert av opptakene ville trolig økt algoritmens evne til å plukke ut menneskeskapte objekter. Her er det derimot forsøkt å finne de beste verdiene for alle opptakene samlet, slik at verdiene har størst mulighet til å fungere tilfredsstillende på nye ukjente data.



Figur 78: Vanlig testbilde med poengsummer for markante rette kanter

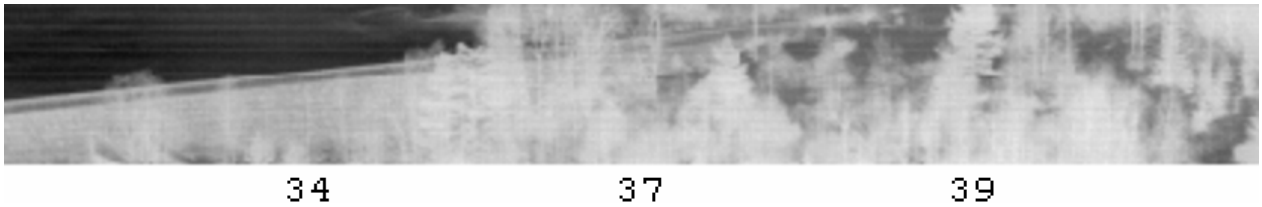
I forbindelse med figur 78 bør det nevnes at i tillegg til kanter langs bygninger oppdages også de lengre horisontale linjene utgjort av veiene på tvers av tunet. Disse kan skimtes som lyse streker mellom bygningene i bildet.

Algoritmen skiller ikke ut alle menneskeskapte objekter. Gården som ble brukt som eksempel på markante rette kanter i kapittel 3 (se figur 25), viser seg å være for liten på bildene for å bli plukket ut av denne algoritmen. Figur 79 viser poengsummene denne fikk. De små poengverdiene skyldes for korte rette kanter.



Figur 79: Figur 25 med poengsummer for markante rette kanter

Figur 80 viser en svakhet ved algoritmen. Med den markante kanten til demningen i venstre rute skulle man forvente fikk den største poengsummen av rutene. Som figuren viser får den delen av bildet noe mindre poeng enn skogsbakgrunnen til høyre i bildet.

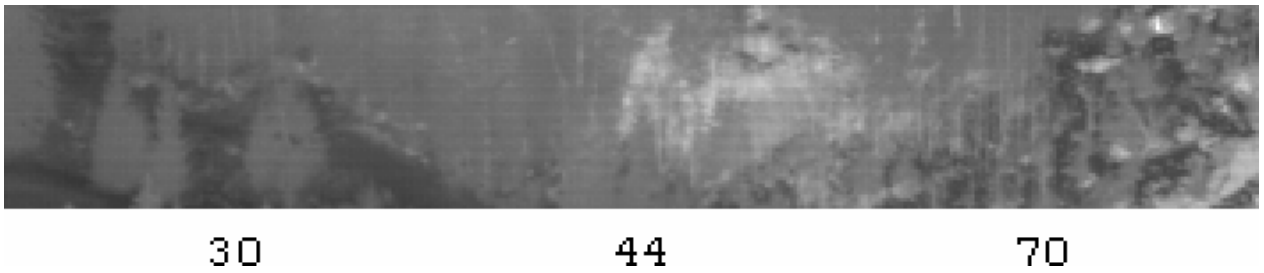


Figur 80: Testbilde med spraglete bakgrunn og poengsummer for markante rette kanter

Grunnen til at terskelen for markante rette kanter ikke kan settes under poengsummene til demningen er at bakgrunnen rundt får så høye poengsummer, slik som vist her. Dette kommer av at bakgrunnen her har veldig ujevn intensitet, og selv om det ikke er så mange rette kanter kommer det allikevel mange markante kantpiksler langs samme linjer. En forbedringsmulighet er å finne de lengste rette linjene slik det er gjort i beregningen av parallelle linjer. Siden dette er en relativt resurskrevende oppgave i forhold til denne algoritmen, bør dette gjøres kun en gang, for deretter å dele resultatet mellom flere egenskapsberegninger. Dermed kan spredte kantbidrag fra en spraglete bakgrunn skilles fra lange rette kanter, akkurat slik det er gjort i beregningen av parallelle linjer.

Det bør nevnes at algoritmen gav resten av rutene med deler av demning høyere poengsum enn alle de andre rutene med skog i samme bildet. Allikevel fikk ingen deler av demningen poengsum over 47, som er den foreslåtte grensen for menneskeskapte objekter. Grunnen til dette er at vinkelen kanten har i bildet passer innstillingen til Hough-transformasjonen dårlig. Til tross for den lave oppløsningen på θ- og ρ-verdien blir bare bruddstykker av kanten plukket ut.

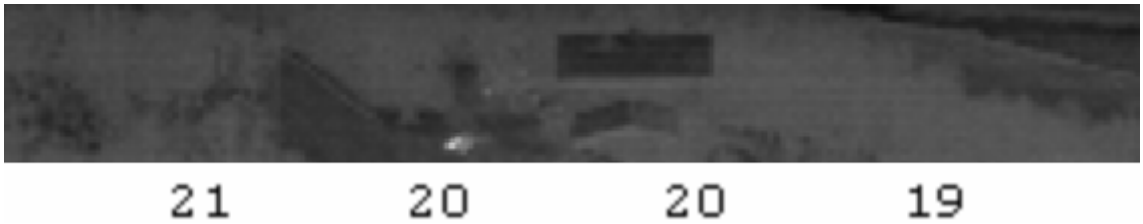
Figur 81 viser en bergknaus og enkelte steiner som er vesentlig varmere enn omgivelsene. Poenggivningen er overraskende høy for disse varme områdene, med opp til 70 poeng på det meste.



Figur 81: Testbilde med poengsummer for markante rette kanter

At steinene ikke inneholder noen tydelige rette kanter, men allikevel får så mange poeng kan tyde på at algoritmen for tungt baserer seg på at kantene skal være sterke og for lite på at de skal være rette. Hvor mye algoritmen skal basere seg på forskjellige egenskaper er avhengig av egenskapene i opptakene. Generelt kan det sies at siden tersklingen luker ut lyse punkter (som gjerne har sterke kanter mot mørkere omegn) er det ønskelig at andre egenskapsberegninger baserer seg på andre egenskaper, slik at flest mulig egenskaper blir utnyttet.

I den sammenheng er det interessant å se hvordan algoritmen håndterer opptak fra overskyete dager. Figur 82 viser at parametrene for hvor sterke kantene må være må justeres i forhold til de forrige opptakene for å registrere kantene rundt dette hustaket.



Figur 82: Testbilde fra overskyet dag og poengsummer for markante rette kanter

5.4.3 Andre testbilder

Den tredje poenglinjen under bildene i vedlegg C viser hvordan denne algoritmen ga poeng til rutene i bildene i testsettet. Beskrivelsen under viser til disse bildene.

De to første delene av testsettet viser at mange bygninger ikke poengsummer høyere enn 47, som foreslått som grense for menneskeskapte objekter. En av grunnene til at mange av kantene på bygningene ikke får høy nok poengsum er at bygningene enten ikke har stor nok varmestråling i forhold til bakgrunnen. Ett eksempel på dette kan finnes i det fjerde øverste bildet i andre del av testsettet. Rett over poengsummen 36 ligger et langt bygg på tvers av bildet. Selv om mønet former en lang rett kant med bakgrunnen, er ikke taket på bygningen tilstrekkelig varm til at kanten blir markant nok. En annen grunn er at noen kanter, som for eksempel i forbindelse med bygningene i det fjerde øverste bildet i den første delen av testsettet, delvis er dekket av andre bygninger og busker. Dette gjør at kantene ikke blir lange nok, og dermed fremstår ikke egenskapen tydelig i forbindelse med disse menneskeskapte objektene.

De overraskende lave poengsummene i forbindelse med demningen i det nest øverste bildet i første del av testsettet, er forklart i forrige delkapittel.

Den høye poengsummen til de to rutene lengst til høyre i det nederste bildet i den andre delen av testsettet, får sine poeng fra både bygningen til venstre og rette bruddstykker av den ellers svakt buede veien.

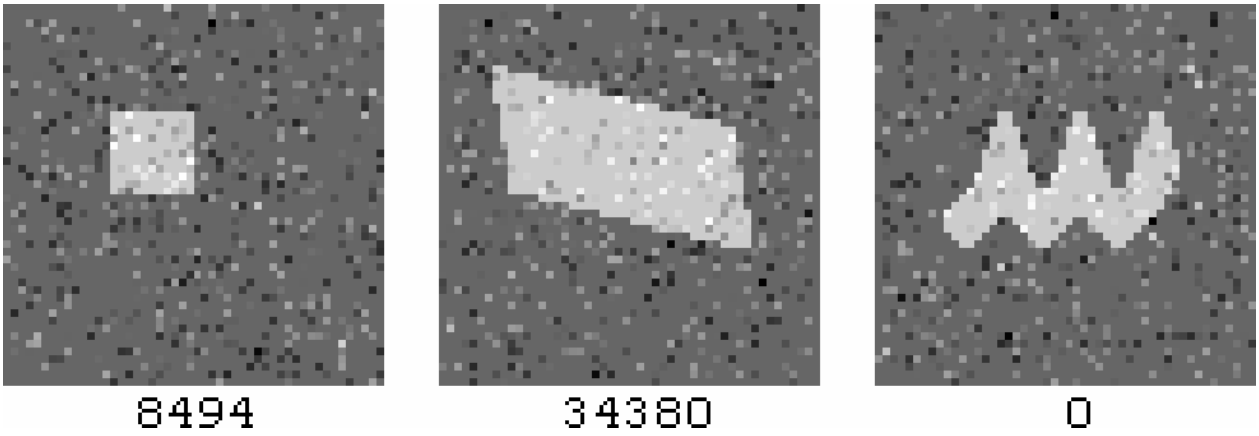
Det tredje testsettet viser at terskelen på 47 ser ut som en god terskel. Ingen av de naturlige områdene overstiger terskelen, selv om noen ligger tett oppunder. Dette gjelder særlig skog avbildet på nært hold, slik som i det tredje og sjette nest øverste bildene. Her er det trærnes lange snirklete kant mot skogbunnen som genererer svært mange kantpiksler, som igjen øker poengsummen. Mangelen på rette kanter gjør allikevel at poengsummene holdes under grensen på 47. Det potensielle problemet med den lange rette kanten langs jordet i det nest øverste bildet avverges ved at kanten ikke er markant nok. Dette er vanlig ved naturlige objekter av litt større størrelse.

5.5 Test av poenggivning for parallelle kanter

Under følger tester av algoritmen for poenggivning av parallelle rette kanter, med testbilder fra de tre kategoriene tegnede bilder, egnede testbilder og andre testbilder.

5.5.1 Tegnede bilder

Figur 83 viser tre tegnede testbilder med tilhørende poengsum fra algoritmen for poenggivning av parallelle linjer. Det har her ikke vært foretatt noen form for ruteinndeling. Hvert av bildene er påført litt støy for at ikke variansen skal bli svært lav og dermed gi poengsummer i milliardklassen.

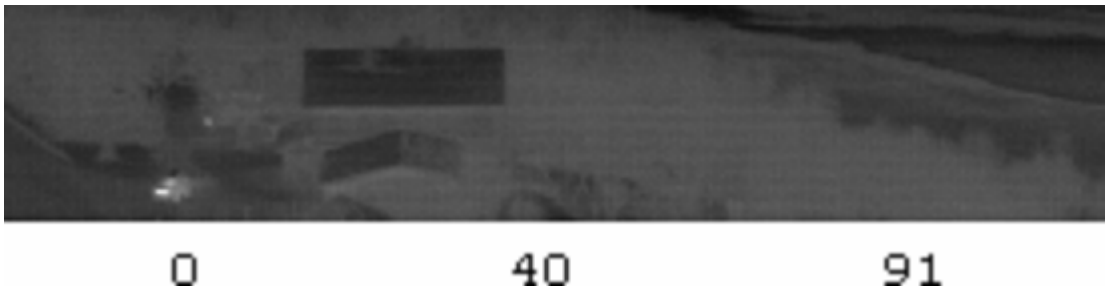


Figur 83: Tegnede testbilder med poengsummer for parallelle rette kanter

Bildet til venstre viser et kvadrat. Med lettsynlige parallelle linjer på begge kanter skulle man forvente en høy poengsum, noe den også har fått. Bildet i midten viser at algoritmen også finner parallelle linjer på skrå. Med større areal og omtrent like liten varians har dette rektangelet fått enda større poengsum enn kvadratet. Det siste bildet viser en uregelmessig form uten noen parallelle linjer. Bildet har dermed ikke fått noen poeng av algoritmen.

5.5.2 Egnede testbilder

Taket på figur 82 er et av de mest perfekte parallellogrammene i bildeseriene. Dette objektet utpreger seg ikke i noen av de tidligere algoritmene heller, så det ville vært ønskelig om det fikk en god poengsum av denne algoritmen. Figur 84 viser poengsummene til de aktuelle rutene.



Figur 84: Testbilde fra overskyet dag og poengsummer for parallelle kanter

Ruten i midten får 40 poeng, som kommer av det store taket. Kun de horisontale linjene av taket avslører parallellogrammet. De vertikale kantene blir for korte tatt i betraktning at den høyre kanten er delt på to pikselkolonner. Dette kan sees på kantbildet av ruten med 40 poeng, som er vist i figur 85.

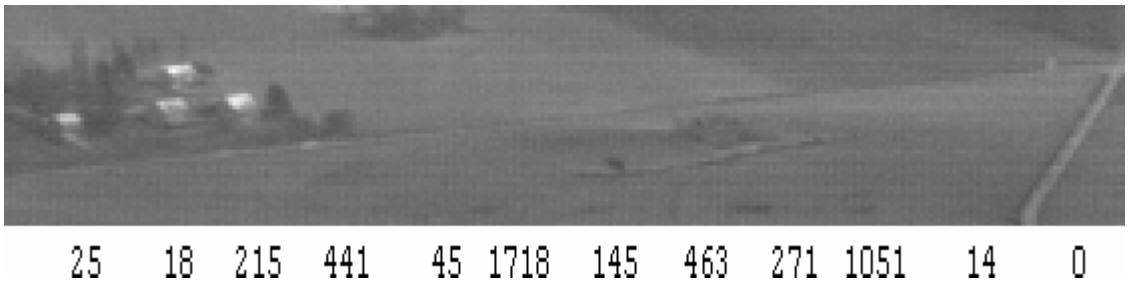
Figur 85: Kantbilde av midtre rute i figur 84

Også den øvre horisontale kanten deles mellom to pikselrader, men kantens lange lengde kombinert med det tillatte mellomrommet på fire piksler, gjør at kanten allikevel blir medregnet.

Figur 84 viser også en uønsket poenggivning på ruten til høyre. På tross av at ruten ikke inneholder noen menneskeskapte objekter får ruten hele 90 poeng. Dette kommer av en ikke så lett synlig vannkant øverst i bildet. Den rette kanten kan lettere sees øverst til høyre i kantbildet i figur 85. Under denne vannkanten ligger det i bildet en sand/stein strand frem mot vegetasjon enda lenger ned. Skillet mellom stranden og vegetasjonen kan også lettere sees i figur 85. Ettersom disse kantene slynger seg i noenlunde samme retning lenger til høyre i bildet enn det figur 84 og figur 85 viser, oppstår det flere lengder der kantene er både rette og parallelle. Siden

stranden i tillegg har en uniform stråling, gir dette til dels veldig gode poengsummer av algoritmen.

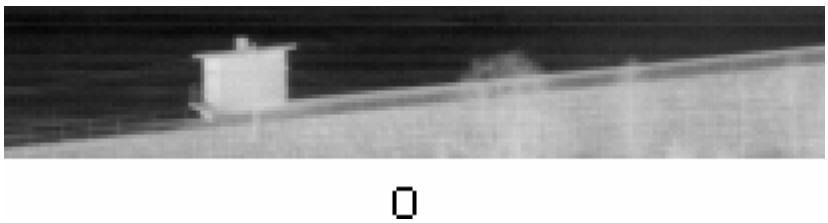
Ved bilder over en viss høyde blir bygningsflatene for små til at algoritmen klarer å finne parallellogrammer. Enkeltstående flater på bygningene vises da bare over et fåtalls piksler. I disse bildene har denne algoritmen en tendens til å finne større firkanter i naturen, slik som jordelapper, parallelle veier og lignende. Siden høyden er stor fremstår også naturlige bakgrunner som jorder og tett skog som uniforme, noe som gjør at algoritmen gir store poengsummer. Figur 86 viser et slikt eksempel.



Figur 86: Et bilde tatt fra stor høyde, med poengsummer for parallelle kanter

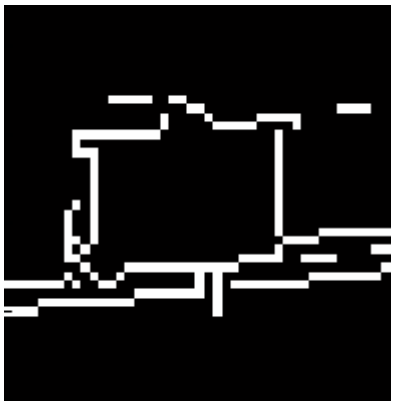
For å unngå disse tilfellene kan det være en fordel å unngå å bruke denne algoritmen når høyden stiger over en grense satt på forhånd. En annen mulighet er å utvikle en egen kantuttrekningsmetode som ikke trekker ut for svake kanter. Algoritmen må ha funnet kanter i ruten med 1051 poeng, på tross av at denne fremstår svært uniform. Grunnen til dette er at det, som nevnt i kapittel 4.6.1, brukes en av MATLAB sine metoder for automatisk fastsettelse av terskelen for kanter. Siden denne regnes ut på grunnlag av hvor mye variasjon det finnes i bildene, blir det ofte funnet de sterkeste kantene trukket ut, selv om de er så svake som i dette tilfellet.

Figur 87 viser igjen en dårlig egenskap ved algoritmen. Den firkantede lille bygningen på demningen burde bli gjenkjent som en bygningsflate, men den har ikke fått noen poeng.



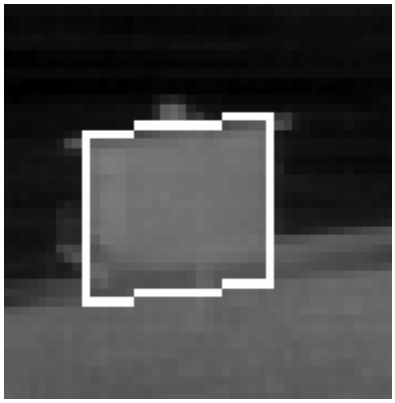
Figur 87: Bilde av bygning på demning og tilhørende poengsum for parallelle kanter

Grunnen til at denne flaten ikke fikk noen poeng er ikke at kantene ikke ble funnet. Kantbildet i figur 88 viser kanten rundt bygningen.



Figur 88: Kantbilde til bygningen på demningen

De rette og tydelige vertikale kantene langs sidene av bygningen ble funnet og betraktet som parallelle. Siden det er tillatt opphold på opptil fire piksler langs kantene ble disse kantene trukket ned til den nederste av de to nære horisontale kantene langs med demningen. I toppen ble de trukket til den øverste kanten på taket. Figur 89 viser det resulterende parallellogram. (Generelt sett pleier dette å være ett trapes, men siden de to oppdagede parallelle linjene her er akkurat like lange, blir det i dette tilfellet et parallellogram)



Figur 89: Trapes tegnet på bildet av bygningen på demningen

Som figuren viser strekker parallellogrammet seg ut over den uniforme veggen både i toppen og i bunnen. Her blir piksler tilhørende taket og demningen innlemmet i parallellogrammet. Siden disse har en noe annen verdi enn resten, blir den beregnede variansen til parallellogrammet for stort til å anta at firkanten tilhører en bygning.

Det kan i denne sammenheng settes spørsmålstegn ved om terskelen til variansen er for lav. Resultater fra flere hundre ruter med diverse naturlig innhold viser at altfor mye bakgrunn hadde blitt gitt poeng hvis terskelen økes. I tillegg kan det her argumenteres for at parallellogrammet i figur 89 faktisk inneholder en større varians en hva bygningsflater pleier, i og med at piksler utenfor flaten har dratt variansen opp. Her er det snarere de tillatte oppholdsrommene langs kantene som forårsaker at algoritmen feiler. Hvis denne grensen settes ned fra de fire som her er brukt, blir derimot ikke taket i figur 84 gitt poeng av algoritmen.

5.5.3 Andre testbilder

Den fjerde poenglinjen under bildene i vedlegg C viser hvordan denne algoritmen ga poeng til rutene i bildene i testsettet. Beskrivelsen under viser til disse bildene.

Som nevnt i forrige delkapittel har denne algoritmen mangler som ikke lot seg rette. Til tross for at den klarte å identifisere de mest fullkomne parallellogrammene, slik som i figur 83 og figur 84, er den utilstrekkelig for de fleste bygninger. Dette kommer godt frem i de to første delene av testsettet. I tillegg ser man tydelig at problemet med svært høye poengsummer til ruter med uniforme gråtoneverdier, slik som på jordeflatene i det andre og tredje bildet i den andre delen av testsettet, samt det første bildet i den tredje delen.

Siden disse rutene med uniforme gråtoneverdier fikk så til de grader høy poengsum, er det forsøkt å sette en øvre terskel for poengsummen i tillegg til en nedre. Dette var heller ingen mulighet til å skille ut menneskeskapte objekter med denne algoritmen, siden poengsummene for disse og for bakgrunnen overlappet i stor grad.

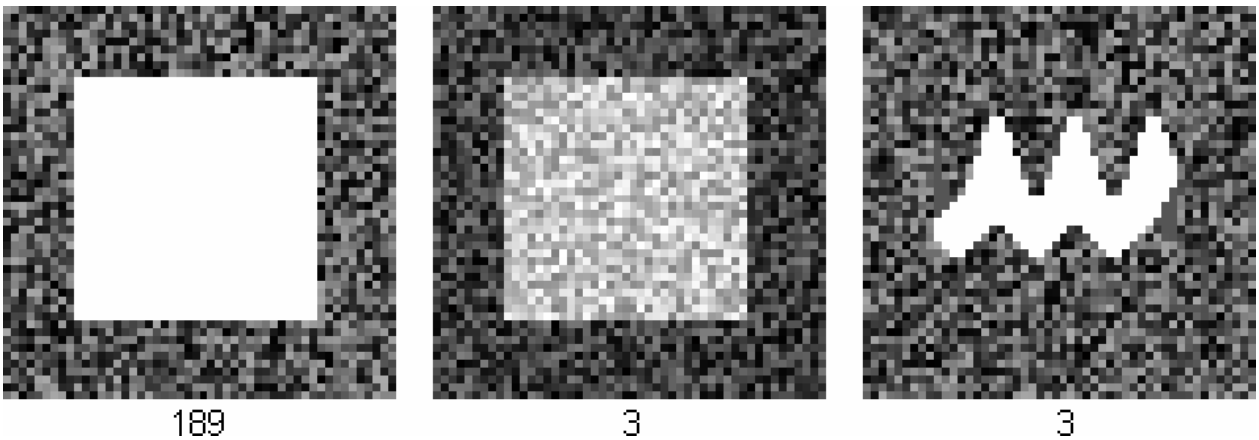
Med nærmest ingen sammenheng mellom poenggivning og lokasjon av menneskeskapte objekter er denne algoritmen funnet ubrukelig.

5.6 Test av poenggivning for uniforme arealer ved rette kanter

Under følger tester av algoritmen for poenggivning av uniforme arealer ved rette kanter, med testbilder fra de tre kategoriene tegnede bilder, egnede testbilder og andre testbilder.

5.6.1 Tegnede bilde

Figur 90 viser tre tegnede testbilder med tilhørende poengsum fra algoritmen for poenggivning av uniforme arealer ved rette kanter.



Figur 90: Tegnede testbilder med poengsummer for uniforme arealer ved rette kanter

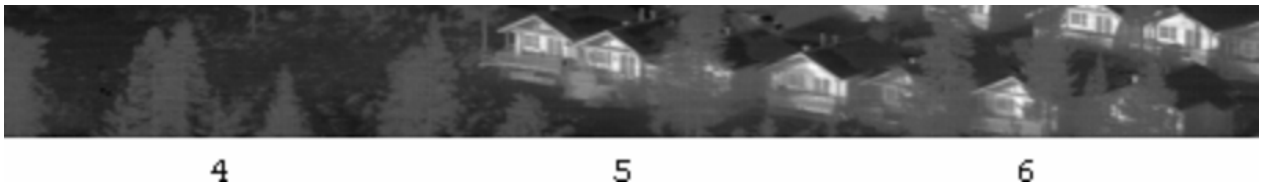
Bildet til venstre viser en firkant med uniforme gråtoneverdier på en spraglete bakgrunn. Firkanten har fått en poengsum oppunder 200. Bildet i midten viser den samme firkanten, men denne gangen med støy, slik at ikke lenger fremstår med uniforme gråtoner. Poengsummen har dermed falt til nesten null. Helt til venstre finnes en figur uten rette kanter. Figuren har uniforme

gråtoneverdier, men får allikevel en poengsum nær null. Dette viser at algoritmen favoriserer objekter med både rette kanter og uniforme gråtoneverdier.

5.6.2 Egnede testbilder

På reelle bilder varierer poengsummen for uniforme arealer ved rette kanter svært lite. Den holder seg mellom 0 og 10.

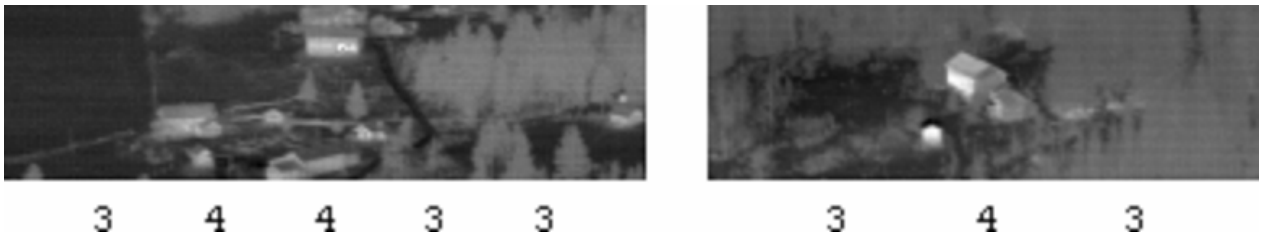
Figur 91 viser en rekke hytter til høyre og vanlig bakgrunn til venstre.



Figur 91: Bilde med hytter og tilhørende poengsummer for uniforme arealer ved rette kanter

Rutene med bygninger har fått fem og seks i poengsum, mens ruten med kun naturlig bakgrunn har fått fire. Dette er representativt for de aller fleste bildene, så en grense på mellom fire og fem passer fint til å skille ut interessante ruter med denne algoritmen.

Algoritmen viste lite overraskende poenggivning, men gav jevnt mer poeng til områder med menneskeskapte objekter enn ellers. Når opptakshøyden økte og bygningene ble mindre sank etter hvert poengsummen under fem. Noe annet var heller ikke ventet, siden egenskapen også forsvinner når objektene blir for fjerne. Figur 92 viser to utklipp med bygninger som har blitt for fjerne for å få poeng over fire.



Figur 92: To bilder med fjerne bygninger og poengsummer for uniforme arealer ved rette kanter

Som diskutert i kapittel 4.7.1 vil kombinasjonen av store arealer å beregne variansen i, samt en forskyvning av dette arealet i forhold til kanten, gjøre at små bygninger ikke får store poengsummer. Kombinasjonen av at rutene med bygninger i figur 92 har større poengsummer enn de andre rutene og at denne poengsummen er lik poengsummen for naturlig bakgrunn i figur 91, gjør det mulig at finere inndeling av poengsummene ville gjort det mulig å skille bygningene på figur 92 fra bakgrunnen i figur 91. Dette er undersøkt, men siden bygningene fikk lavere poengsum enn bakgrunnen, lar de seg ikke skille ved denne poengsummen.

5.6.3 Andre testbilder

Den nederste poenglinjen under bildene i vedlegg C viser hvordan denne algoritmen ga poeng til rutene i bildene i testsettet. Beskrivelsen under viser til disse bildene.

Testsettet viser at terskelen mellom 4 og 5 passer godt for å betegne interessante områder. Enkelte bygninger faller under terskelen, men disse er forholdsvis små og har dermed små kanter. Dette gjør at de ikke uventet får lave poengsummer. Ingen av områdene uten menneskeskapte objekter fikk poengsummer over terskelen. Dette tyder på en stabil og god algoritme.

5.7 Test av algoritmene sammen

I denne siste delen blir algoritmene satt sammen til én karakterisering av ruter som interessante eller uinteressante. Dette delkapitlet er delt i to deler. Først velges en kombinering av poengsummene fra de forskjellige algoritmene som klassifiseringsmetode. Deretter vises algoritmens effekt på testsettet.

Det hadde her vært ønskelig å kunne gi et konkret mål på reduksjonen av datamengden, men dette er altfor avhengig av innholdet i dataene. Er det mye menneskeskapte objekter, vil lite data kunne kuttes bort, selv om algoritmen fungerer optimalt. En prosentandel som angir hvor mye av de uinteressante områdene som er forkastet er vanskelig å beregne, i og med at det er ønskelig med enn viss omegn av bakgrunn rundt de bevarte menneskeskapte objektene for å sette disse i kontekst. En prosentandel som angir hvor mange av de menneskeskapte objektene som ikke ble bevart sier mer om algoritmen, selv om en liten prosentandel gjerne angir en algoritme som forkaster lite av den naturlige bakgrunnen også. Det er derfor heller gjengitt et relativt stort antall representative bilder som viser forskjellige situasjoner håndteres (det 19 bilder store testsettet). I tillegg gis det mer innsikt ved en liten forklaring, samt poengsummene til hvert bilde.

5.7.1 Valg av klassifiseringsmetode

Testene har hittil vist at hver algoritme plukker ut forskjellige menneskeskapte objekter. For eksempel gjenkjenner algoritmen for uniforme arealer ved rette kanter bygninger med lange kanter, mens tersklingen finner bygninger med mye stråling. Ingen av testene, foruten den foreslått forkastede algoritmen for parallelle rette kanter, gir mange uinteressante områder poeng over den foreslåtte terskelen til hver algoritme. Dette kommer av at egenskapene er av en art som forekommer blant en del typer menneskeskapte objekter, men som forekommer sjeldent i naturlige områder. Derfor passer det å kreve minst én av flere slike forskjellige egenskaper. Skulle disse vært kombinert ville få, om noen områder blitt bevart.

Det kan også argumenteres for at noen av algoritmene allerede har kombinert egenskaper, slik som markante og rette kanter. Denne inneholder en vektet fordeling mellom to egenskaper som sammen for det meste kun forekommer blant menneskeskapte objekter.

Derfor er det her valgt å klassifisere alle ruter med poengsum over den foreslåtte terskelen til minst én av algoritmene som interessante områder. Disse tersklene var:

- Lyse områder definert ved statisk terskel:
  Ikke brukt, siden dynamisk terskel identifiserte samme egenskap bedre.

- Lyse områder definert ved dynamisk terskel:
  Inkludere ruter med poengsum større enn 0.

- Markante rette kanter:
  Inkludere ruter med poengsum større enn 46.

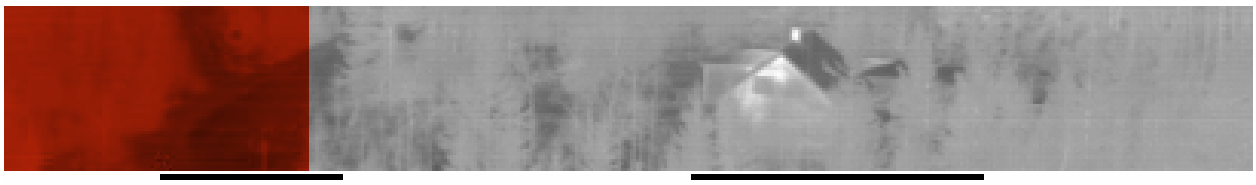
- Parallelle linjer:
  Ikke brukt, siden algoritmen ikke skilte mellom interessante og uinteressante områder.

- Uniforme arealer ved rette kanter:
  Inkludere ruter med poengsum større enn 4.

5.7.2 Test av hele algoritmen på testsettet

Bildene i testsettet blir her vist igjen. I tillegg til markeringene av de manuelt klassifiserte interessante områdene (strek under bildet), er de automatisk klassifiserte uinteressante områdene markert med rødt. Dette er gjort for lett å sammenligne de to klassifiseringene.

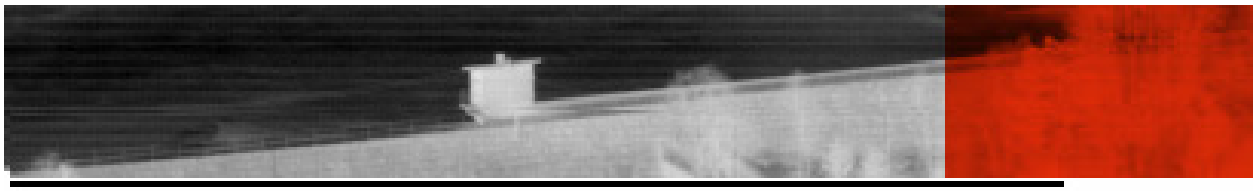
Etter hver del er hvert bilde i delen kort kommentert hvor godt den automatiske klassifiseringen er i forhold til den manuelle. I tillegg forklares det kort hvorfor den automatiske klassifiseringen har kommet til det viste resultatet. I denne sammenheng vil det kunne være interessant for leseren å sammenligne med poengsummene til de forskjellige algoritmene for egenskapsidentifisering, som er vist for hvert bilde i vedlegg C. I bildene i dette vedlegget vil det også ruteinndelingen av bildene gå frem (en rute sentrert over hver poengkolonne), noe som er nyttig å vite ved sammenligning av de to klassifiseringene. Ruteinndelingene er ikke markert i bildene i dette delkapitlet for å unngå å presse for mye informasjon inn i bildene.



Figur 93: 1. bilde i 1. del av testsettet med forkastede områder markert røde

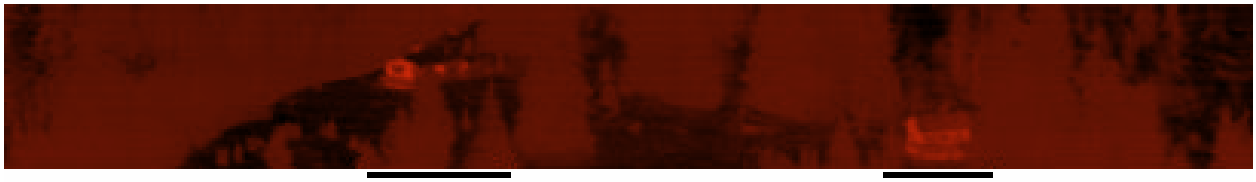
Figur 93 viser det første bildet i den første delen av testsettet. De lyse feltene på huset gjør at det blir bevart. Siden rutene er ganske store i forhold til den delen av huset som synes på bildet, blir det inkludert ganske mye data rundt huset.

Veien på venstre side er ikke bevart. Dette kan ikke sies å være en feilvurdering av noen av identifiseringsalgoritmen, siden veien ikke innehar noen av de søkte egenskapene. Det vil ikke være mulig å finne egenskaper som dekker absolutt alle typer menneskeskapte objekter, så noen objekter vil alltid kunne passere. Veien fremstår på bildet omtrent som et typisk mørkt felt mellom trærne i skog (se for eksempel det siste bildet i hele testsettet), og vil være vanskelig å skille fra sådanne uansett valgt egenskap.



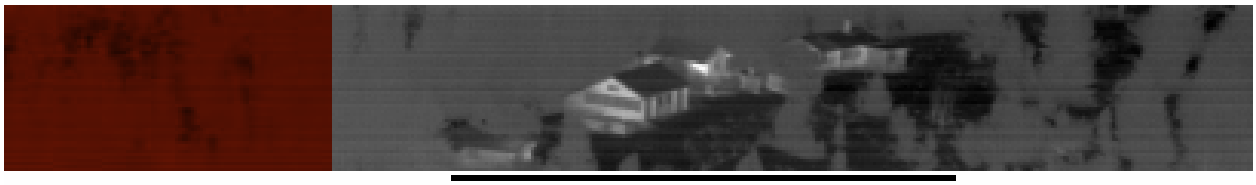
Figur 94: 2. bilde i 1. del av testsettet med forkastede områder markert røde

Figur 94 viser det andre bildet i den første delen av testsettet. Her er den manuelle og den automatiske klassifiseringen nesten identisk. Her må det nevnes at det er de vertikale kantene på hver side av det lille huset på demningen som forårsaker deler av den høye poenggivningen for uniforme arealene ved rette kanter. Siden denne ligger midt på demningen blir mesteparten av demningen inkludert i rutene som bevares. For å finne ut om resten av demningen bidro med mange poeng, ble algoritmen også testet på det samme bildet, men med det lille huset redigert bort. Kun den ene av de to 5-poengene falt ned til 4, mens den andre besto. Dette viser at demningen alene også stedvis kan bli plukket ut.



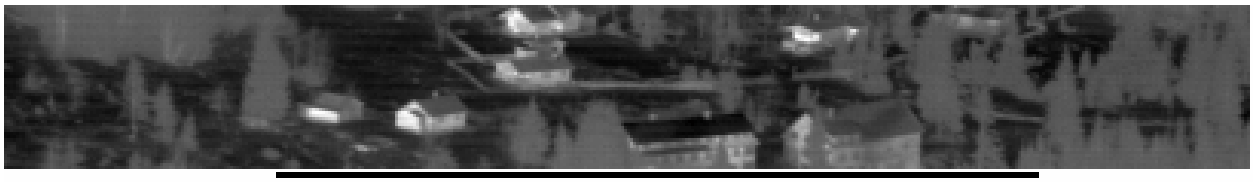
Figur 95: 3. bilde i 1. del av testsettet med forkastede områder markert røde

Figur 95 viser det tredje bildet i den første delen av testsettet. Dette bildet kan være det bildet i testsettet som håndteres dårligst, og viser svakheter ved algoritmen. Her er ingen av husene blitt bevart. Ingen av husene er varmet nok opp av solen til å stråle nok varme til å overstige den dynamiske terskelen. Siden dette gjelder begge husene ligger de antagelig i skygge. Ingen av husene lekker nok varme til å overstige terskelen heller. Angående markante rette kanter og uniforme arealer ved rette kanter, så er begge bygningene så små på bildene at slike egenskaper ikke lett blir oppdaget. Den skarpe kanten mellom taket og endeveggen på bygningen til venstre er for kort til å registreres, mens kanten mellom mønet og veggen på bygningen til høyre er for svak til å bli vurdert som kant. Blandingen av lite varme og svært små, gjør at bygningene ikke blir tatt med i algoritmen.



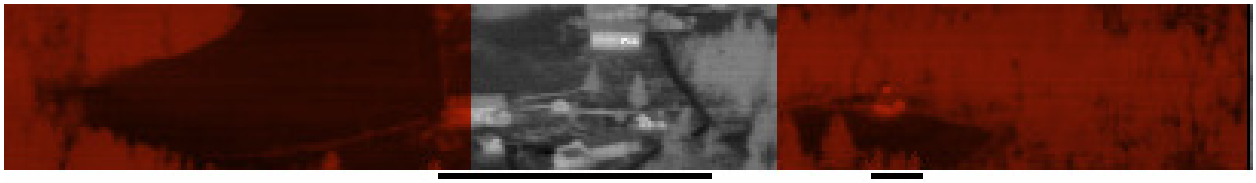
Figur 96: 4. bilde i 1. del av testsettet med forkastede områder markert røde

Figur 96 viser det 4. bildet i den første delen av testsettet. Her er alle områdene manuelt karakterisert som interessante bevart av algoritmen. Rutestørrelsen gjør som ventet at noe mer av arealet rundt bygningene og det manuelt merkede området, er tatt med. Alle de tre egenskapene er funnet i området.



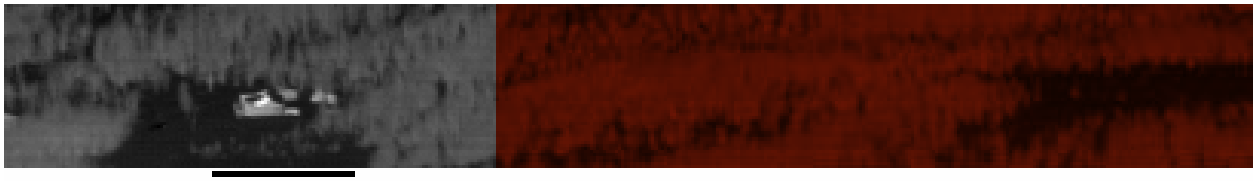
Figur 97: 5. bilde i 1. del av testsettet med forkastede områder markert røde

Figur 97 viser det 5. bildet i den første delen av testsettet. Her er igjen alle områdene manuelt karakterisert som interessante bevart av algoritmen, og igjen er noe mer areal rundt bygningene inkludert som følge av rutestørrelsen. I tillegg til de varme punktene på mange bygninger er det også registrert markante kanter mellom tak og vegg på mange.



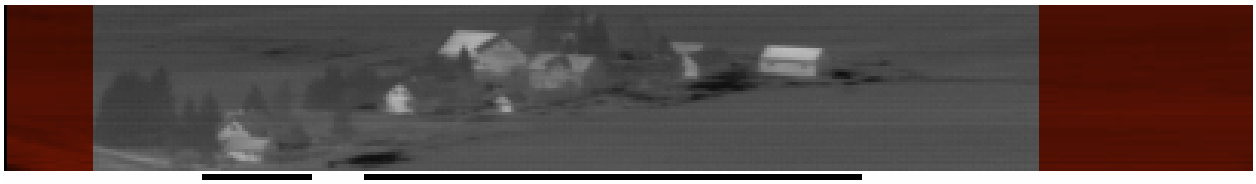
Figur 98: 6. bilde i 1. del av testsettet med forkastede områder markert røde

Figur 98 viser det 6. bildet i den første delen av testsettet. Her er det den øvre bygningen som utmerker seg som varm. Noen kanter langs samme bygning og noen veier på tvers av tunet innen samme ruter gjør at også poengsummen for markante rette kanter stiger over terskelen sin. De andre bygningene er for små til å bidra med lange nok kanter, og har ellers samme intensitet som skogen rundt, og utmerker seg heller ikke på den måten. Disse bygningene ligner generelt svært mye på skogen rundt, og er ikke overraskende ikke oppdaget av algoritmen.



Figur 99: 1. bilde i 2. del av testsettet med forkastede områder markert røde

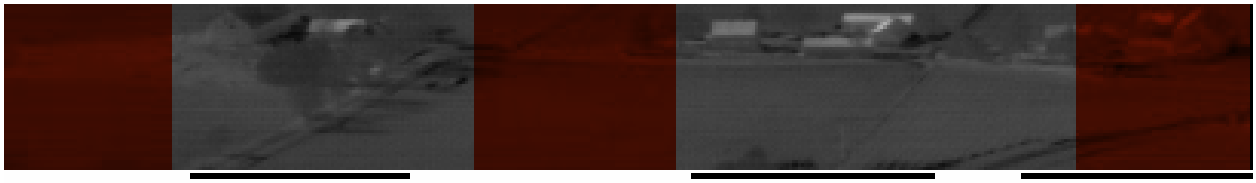
Figur 99 viser det første bildet i den andre delen av testsettet. Her er igjen alle områdene manuelt karakterisert som interessante bevart av algoritmen, og igjen er noe mer areal rundt bygningene inkludert som følge av rutestørrelsen. I tillegg til de varme punktene på mange bygninger er det også registrert uniforme arealer ved rette kanter. Dette kommer av både den rette kanten på undersiden av huset (der det faktisk er bakken som utgjør den mest uniforme siden av kanten!) og noen små kanter langs skogskanten.



Figur 100: 2. bilde i 2. del av testsettet med forkastede områder markert røde

Figur 100 viser det andre bildet i den andre delen av testsettet. Igjen alle områdene manuelt karakterisert som interessante bevart av algoritmen. Blant bygningsmassen er det registrert varme punkter og stedvis tilstrekkelig uniforme arealer ved rette kanter.

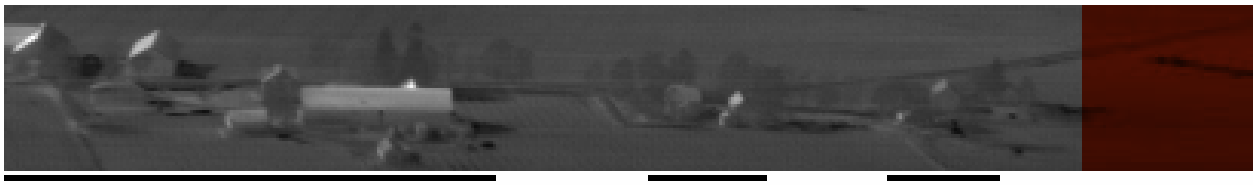
Helt til venstre i bildet viser den dynamiske terskelen seg bedre egnet enn den statiske, gitt at det er valgt å sette poenggrensen ved 0 poeng. Den ytterste venstre biten av bildet viser ingen interessante områder, og heller ingen som fremstår som særdeles lyse, og burde derfor ikke inkluderes. Dette gjør allikevel den statiske tersklingen, i motsetning til den dynamiske.



Figur 101: 3. bilde i 2. del av testsettet med forkastede områder markert røde

Figur 101 viser det tredje bildet i den andre delen av testsettet. Her er gården øverst til høyre ikke blitt bevart. Av bildet kan man se at denne gården er relativt vanskelig se i forhold til gården ved siden av. Dette kommer av liten varmestråling og gråtoneverdier nesten felles lik bakgrunnen. Dermed blir det heller ingen sterke kanter rundt bygningene. Siden bygningene nesten går i ett med bakgrunnen, er den vanskelig å finne, både manuelt og automatisk.

Gården til venstre er plukket ut av den dynamiske tersklingen. Nå er denne lavere enn den statiske, og plukker med seg piksler som den statiske overser. Sett i sammenheng med forrige bilde, der den statiske var for lav, mens den dynamiske var passe, er den dynamiske terskelen bedre egnet enn den statiske.

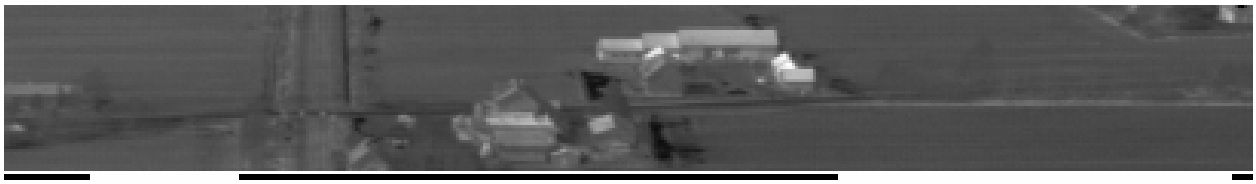


Figur 102: 4. bilde i 2. del av testsettet med forkastede områder markert røde

Figur 102 viser det 4. bildet i den andre delen av testsettet. Her er alle områdene manuelt karakterisert som interessante bevart av algoritmen. Den manuelle karakteriseringen har utelatt noen områder mellom de interessante gårdene. Disse er bevart fordi rutene er større enn halvparten av mellomrommene, og overlapper hverandre 50 %. Dermed blir hele området bevart

selv om én rute ikke skulle bli karakterisert som interessant av algoritmen. Dette er som tidligere forklart nødvendig fordi deler av et menneskeskapt objekt kan stikke inn i ruten uten å bli gjenkjent. Poengsummene til ruten som faller mellom to av områdene manuelt karakterisert som interessante (rute nr. 6 fra venstre, se ruter og poengsummer for bildet i vedlegg C) er under terskelen, og ville dermed ikke bli bevart var det ikke for de interessante naborutene.

Gården lengst til høyre er kun så vidt mulig å se, slik som gården lengst til høyre på forrige bilde. Denne inneholder derimot et par piksler lye nok til at ruten blir bevart. Dette viser igjen hvor passende det er ikke å kreve mange lyse piksler for å gjenkjenne interessante ruter.



Figur 103: 5. bilde i 2. del av testsettet med forkastede områder markert røde

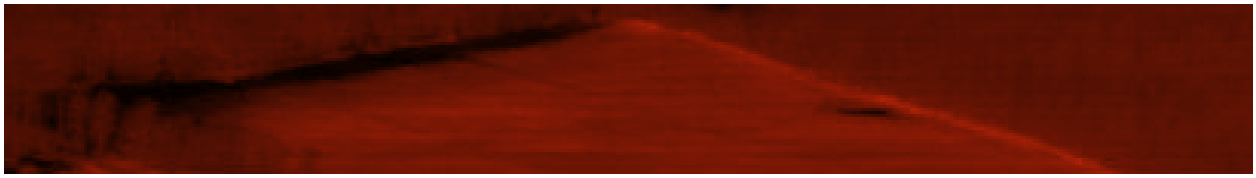
Figur 103 viser det 5. bildet i den andre delen av testsettet. Igjen er alle områdene manuelt karakterisert som interessante bevart av algoritmen. Det lange feltet til venstre som er manuelt karakterisert som uinteressant er inkludert av to grunner. For det første så er veien på tvers oppdaget som en kant med uniformt areal på siden, noe som må tilsvare jordet. Dette er i og for seg ikke galt, siden veien må kunne karakteriseres som menneskeskapt. Foruten dette ville området allikevel vært inkludert som følge av at det kun er én uinteressant rute mellom to interessante (med tanke på lyse områder), slik som diskutert under forrige bilde.

Figur 104: 6. bilde i 2. del av testsettet med forkastede områder markert røde

Figur 104 viser det 6. bildet i den andre delen av testsettet. Også her er det fullt samsvar mellom den manuelle og automatiske klassifiseringen av bildet. Både lyse felter på bygningene og rette kanter ved bygg og vei avslører dette området som interessant.

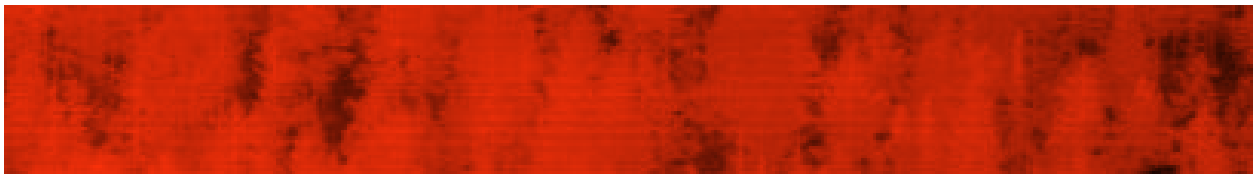
Figur 105: 1. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 105 viser det første bildet i den tredje delen av testsettet, som kun inneholder uinteressante områder av forskjellige slag, foruten akkurat dette bildet som inneholder noen bygninger på hver side. Som ønsket er det kun bygningene som er bevart, mens jordet i midten er forkastet.



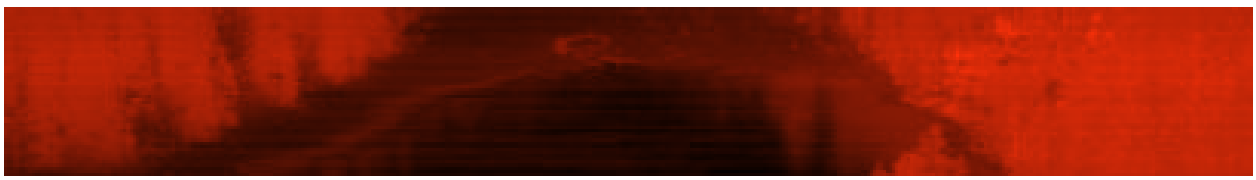
Figur 106: 2. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 106 viser det andre bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.



Figur 107: 3. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 107 viser det tredje bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.

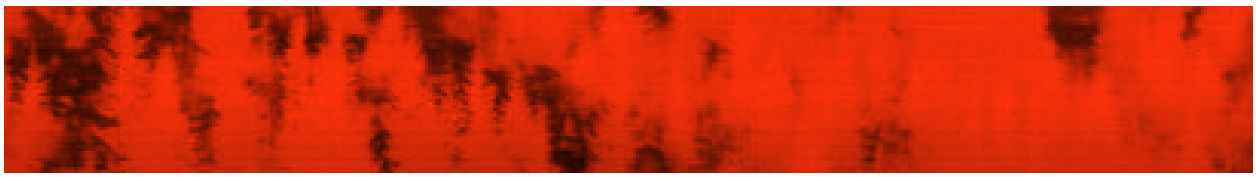


Figur 108: 4. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 108 viser det 4. bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.

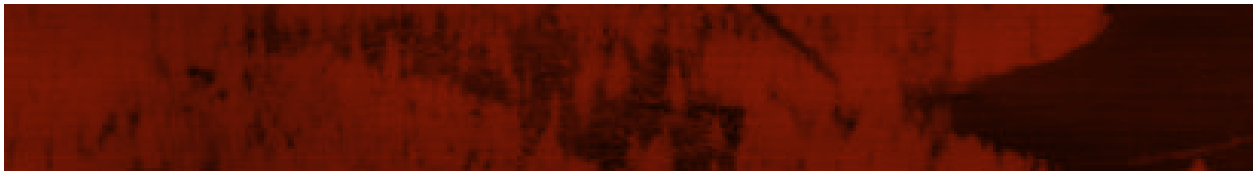
Figur 109: 5. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 109 viser det 5. bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.



Figur 110: 6. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 110 viser det 6. bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.



Figur 111: 7. bilde i 3. del av testsettet med forkastede områder markert røde

Figur 111 viser det 7. og siste bildet i den tredje delen av testsettet. Bildet inneholder ingen interessante områder, og hele bildet er som ønsket forkastet.

Som blant annet de siste 7 bildene i testsettet viser, er algoritmen god til å forkaste forskjellige uinteressante områder med bakgrunn av forskjellig slag.

Kapittel 6: Videre arbeid

I dette kapitlet er det listet foreslag til oppgaver som ville utvidet undersøkelsene gjort i denne rapporten. Noen av forslagene er nevnt i rapporten der de er aktuelle.

- Nevralt nett
  Det ville være interessant å teste et nevralt nett som utførte den samme oppgaven. Effektiviteten og egenskapene observert under testing av dette nettverket vil ved sammenligning med resultater i testingen foretatt her, kunne belyse spørsmålet om hvilke av disse metodene som egner seg best.

- Bildedeling i to dimensjoner
  Bildene i denne rapporten er kun delt i bredden. For bedre å kunne utelukke flere områder vil det også være aktuelt å dele bildene i høyden etter samme prinsipp. Det må i så henseende tas hensyn til hvorvidt bildene er tatt med landskapet eller rett nedover fra dronen. Bilder som er tatt rett ovenfra gir likere objekter i forskjellige høyder i bildet (vertikal posisjon). I disse tilfellene passer det best å utvide bildeoppdelingen til to dimensjoner.

- Rekursiv bildeoppdeling
  Bildene kan også deles rekursivt, for eksempel stadig dele rutene i 9 nye ruter (3x3 med 50 % overlapp), til ruter mindre enn objektstørrelsen. Dette vil kunne øke potensialet av mengde data som kan forkastes, men det kommer nok til også å øke resurskravet til bildeinndeling. Det vil også være nødvendig å foreta en ny evaluering av klassifiseringsmetoden ved en slik inndeling, i og med at man ikke lenger kan være sikker på å dekke hele objekter i hver minste rute.

- Mer vekt på rette kanter i algoritmen for markante rette kanter
  Som testingen av algoritmen for markante rette kanter tydet på, ble det muligens lagt for mye vekt på at kantene skulle være markante og for lite på at de skulle være rette. Derfor ville det være interessant å forsøke å endre algoritmens fokus for å se om dette bedret dens evne til å plukke ut menneskeskapte objekter og kun det.

- Fortegn på kantene i algoritmen for parallelle kanter
  Hvis alle kantene som finnes får et fortegn etter hvilken vei kanten gikk (fra mørkt til lyst eller omvendt) ville det være mulig å kreve at kantene som betraktes som parallelle skal ha motsatt fortegn. Dette ville øke sannsynligheten for at disse tilhørte hver sin ende av samme objektet, siden objektet ofte er enten lysere eller mørkere enn bakgrunnen.

- Wavelets i stedet for varians i algoritmen for uniforme arealer ved rette kanter
  Som foreslått tidligere i rapporten, kan Wavelets være mer brukbart enn å beregne varians som grunnlag for denne algoritmen. Dette burde vært testet og sammenlignet med løsningen brukt i denne rapporten.

- Effektiv implementering og test av kjøretider
  Algoritmene i denne rapporten burde vært implementert med tanke på maksimal ytelse,

for deretter å teste kjøretiden. Det er essensielt at denne holdes så lavt at den kan kjøres mellom hver gang det tas et bilde i dronen, slik at systemet kan brukes i sanntid. Ved en slik implementasjon ville det også være interessant å se hvilke egenskapsuttrekkelser som er mest tidskrevende, og for dermed å se hvilke algoritmer som eventuelt må revurderes.

- Forskjellige egenskapsidentifiseringer til forskjellige bilder
  Det vil også være interessant å benytte andre algoritmer for å detektere andre egenskaper i opptak med andre formål. For eksempel ville det være mulig å lete etter bortkomne mennesker i øde landskap som vidde eller hav. Til dette kan man tenke seg algoritmer for å finne små varme flekker på IR-bilder. I stedet for å klippe og lagre bildet, ville det være aktuelt å sende en posisjon samt bildet til letemannskap. Noe lignende kunne man tenke seg for kartlegging av viltbestander, som for eksempel telling av rein på vidda.

Kapittel 7: Konklusjon

Algoritmen i oppgaven bevarte de aller fleste av de menneskeskapte objektene i testdataene, selv om store mengder av dataene ble forkastet som uinteressante.

Til å gjenkjenne menneskeskapte objekter kan det letes etter minst én av følgende egenskaper:

- Høy varmestråling
  Menneskeskapte objekter er som regel varmere enn omgivelsene, enten det skyldes oppvarming innenfra eller fra sola. Naturlige omgivelser annet enn fjell, slik som vegetasjon og jordbunn varmes ikke tilsvarende opp av sola.

- Rette kanter
  Siden mange menneskeskapte objekter inneholder rette kanter, slik som bygninger, danner disse rette kanter mot bakgrunnen eller mellom forskjellige deler av objektet.

- Uniforme arealer ved kanter
  Mange menneskeskapte objekter har uniforme flater, i motsetning til vegestasjon og annen naturlig bakgrunn.

Selv om det i oppgaven har vært sett på som viktigere å bevare alle menneskeskapte objekter enn å forkaste alle uinteressante områder, overser algoritmen enkelte menneskeskapte objekter. Disse har to felles egenskaper. Enten så stråler de for lite varmestråling, eller så er de for små til å kunne inneholde rette kanter lange nok til å utmerke seg. Dette kan skyldes at de enten delvis er skjult bak naturlige elementer som trær og busker, eller at de ligger for langt unna. Disse objektene er vanskelig å skille fra bakgrunnen fordi de kan fremstå som svært like naturlige elementer i bakgrunnen.

Kilder

Under er alle kildene og referansene fra oppgaven listet. Referansene er delt opp etter kildetypen og er nummerert etter den rekkefølgen de forekommer i rapporten.

Artikler

[a1] A. Skodras, C. Christopoulos, T. Ebrahimi:
The JPEG 2000 Still Image Compression Standard 2001

[a2] P. Cayouette, G. Labonté, A. Morin:
Probabilistic neural networks for infrared imaging target discrimination 2003

[a3] J. L. Solka, D .J. Marchette, B. C. Wallet, V. L. Irwin, G. W. Rogers:
Identification of Man-Made Regions in Unmanned Aerial Vehicle Imagery and Videos 1998

[a4] G. Marsiglia, L. Fortunato, A. Ondini, G. Balzarotti:
Template Matching Techniques for Automatic IR Target Recognition in real and simulated scenarios: tests and evaluations 2003

[a5] T. McInerney, D. Terzopoulos:
Topology Adaptable Snakes 1995

Bøker

[b1] K. Mehrotra, C. K. Mohan, S. Ranka:
Elements of Artificial Neural Networks, second printing
The MIT press 2000
ISBN: 0-262-13328-8

[b2] R. C. Gonzalez, R. E. Woods:
Digital Image Processing, Second edition
Prentice Hall 2002
ISBN: 0-20-118075-8

[b3] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling:
Numerical Recipes in C, The Art of Scientific Computing
Cambridge University Press 1988
ISBN: 0-521-35465-X

Webadresser

[w1] www.caplex.net 25.04.05

[w2] www.chiariglione.org/mpeg 25.04.05

[w3] www.fas.org/irp/program/core/dted.htm 25.04.05

[w3] www.ieee.org 25.04.05

[w4] www.ieee.org/portal/site/mainsite/
menuitem.818c0c39e85ef176fb2275875bac26c8/
index.jsp?&pName=corp_level1&path=web/
search&file=index.xml&xsl=generic.xsl 25.04.05

[w5] www.spie.org 25.04.05

[w6] spie.org/app/Publications 25.04.05

[w7] www.acm.org 25.04.05

[w8] campus.acm.org/public/search/search.cfm 25.04.05
[w9] www.google.com 25.04.05
[w10] www.amara.com/IEEEwave/IEEEwavelet.html 02.06.05
[w11] www.fas.org/irp/program/core/dted.htm 25.04.05

Personer

[p1] Arne Jan Rødland 16.02.05

Vedlegg A: Kildekode

Dette vedlegget gjengir kildekoden. De forskjellige metodene blir gjengitt i hvert sitt avsnitt med metodenavnet som overskrift.

Kommentarene i koden er skrevet på engelsk, i motsetning til resten av denne rapporten. Engelsk er valgt fordi det står bedre i stil til resten av koden, der alle reserverte ord og metodenavn i MATLAB-bibliotekene har engelske navn.

Noen steder er koden delt inn i bolker med overskrifter for å lette forståelsen og lesbarheten til koden. Disse overskriftene er markert ved seks %-tegn først på linjen. Andre vanlige kommentarer er markert med kun ett %-tegn.

Følgende metoder er inkludert i dette vedlegg:

acceptedLine

```
function accept = acceptedLine(edgeImg, line)
%ACCEPTEDLINE Checks if line has more than 2/3 of its pixels
%ACCEPT = ACCEPTEDLINE( EDGEIMG, LINE )
%               EDGEIMG => The image with the edges.
%               LINE    => The line to check (same structure as lines from
%                          houghline, a MATLAB method.
%               ACCEPT  => 1 if line has more than 66 % of its pixels set,
%                          0 else.

theta=line.theta*pi/180;
rho=line.rho;
linePoints=0;
emptySpaces=0;

if( mod(theta,pi)<0.25*pi || mod(theta,pi)>0.75*pi )
    for(y=line.point1(2):line.point2(2))
        x=round((rho-y*sin(theta))/cos(theta))+1;
        if(x>size(edgeImg,2) || x<1)
            % Rounding off may cuse this to happend.
            continue;
        end
        if(edgeImg(y,x)>0)
            linePoints=linePoints+1;
        else
            emptySpaces=emptySpaces+1;
        end
    end
else
    for(x=line.point1(1):line.point2(1))
        y=round((rho-x*cos(theta))/sin(theta))+1;
        if(y>size(edgeImg,1) || y<1)
            % Rounding off may cuse this to happend.
            continue;
        end
        if(edgeImg(y,x)>0)
            linePoints=linePoints+1;
        else
            emptySpaces=emptySpaces+1;
        end
    end
end
accept=0;
if(emptySpaces+linePoints~=0 && ...
        emptySpaces/(emptySpaces+linePoints) < 1/3)
    accept=1;
end
```

averageValueOfTop

```
function average = averageValueOfTop( image, p )
%AVERAGEVALUEOFTOP Calculates the average value of the top p of the values
%                  in image.
%AVERAGE = AVERAGEVALUEOFTOP( IMAGE, P )
%               IMAGE   => The image with the values.
```

```
%               P       => The prosentage of the top to calculate the
%                          average from. Given between 0 and 1.
%               AVERAGE => The resulting average.
if(p>1)
    error('Trying to extract more than 100% in method averageValueOfTop.');
end
nbOfTopPixels=ceil(prod(size(image))*p);
list=sort(image(:),'descend');
average=sum(list(1:nbOfTopPixels)) / nbOfTopPixels;
```

cutBelow

```
function out_img = cutBelow( img, t )
%CUTBELOW Sets the values in an image less than a threshold to zero.
%OUT_IMG = CUTBELOW( IMG, T )
%               IMG   => The image to cut the lowest values in.
%               T     => The threshold.
%               OUT_IMG => The resulting image.
[M,N]=size(img);
out_img=img;
for(i=1:M)
    for(j=1:N)
        if(img(i,j)<t)
            out_img(i,j)=0;
        end
    end
end
```

dist_to_pixel

```
function dist = dist_to_pixel( navdata, mapFile, imagepos )
%DIST_TO_PIXEL  Calculates the distance from cameraposition to array in
%               given pixel hits terrain. Uses DTED. Returns -1 for
%               distances too far away to be certain. This distance is set
%               in the TOO_FAR_DIST variable below. Mark: Phenomena as
%               earths curved surface is not taken into account.
%DIST = DIST_TO_PIXEL( NAVDATA, MAPFILE, IMGPOS )
%               NAVDATA => Navigation data on format mk3.
%               MAPFILE => Path to DTED in an .DT1 file.
%                          Fileextention included.
%               IMAGEPOS=> Position of pixel in image: [col, row]
%               DIST =>    Distance from missil to pixels position on
%                          ground (meters).

steplen = 10;
too_far_dist=50000; % Returns -1 if distance to pixel grows beyond
                    % to_far_dist.
[dtedm,dtedo,dtedi,dteda] = read_dted(mapFile);
%read_dted is a KDA-method, not provided in this report.


msl_geoangpos(1) = navdata(6,9); %UAV lat
msl_geoangpos(2) = navdata(7,9); %UAV lon
msl_geoangpos(3) = -navdata(3,9); %UAV height
msl_dtedpos = geo2dtedindex([msl_geoangpos(2);msl_geoangpos(1)],...
                            dtedo, dtedi);
%geo2dtedindex is a KDA-method, not provided in this report.
```

```
losang = im2ang(navdata, imagepos); %Azimuth & pitch
geopos = im2geo(navdata, imagepos); % [X Y Z]
%im2geo is a KDA-method, not provided in this report.

% Coordinatetransformation from geographical coordinates (x,y,z) to
% geographical angle coordinates (lat,lon,h).
geoangpos = geo2geoang(navdata, geopos);
dtedpos = geo2dtedindex([geoangpos(2);geoangpos(1)], dtedo, dtedi);
%geo2geoang and geo2dtedindex is a KDA-method, not provided in this report.

% Distance in meters between two geographical angle coordinates (lat,lon,h)
radius = geoang_xydist(msl_geoangpos, geoangpos);
%geoang_xydist is a KDA-method, not provided in this report.
numpoints = 1 + ceil(radius / steplen);

if numpoints < 2
    %msl_position and pixel_position projected onto zerolevel is smaller
    %than steplen. Given relative small steplen; UAV looks almost straight
    %downwards. Distance is approxematly hight of UAV-hight over terrain.
    dist=msl_geoangpos(3) - dtedm(msl_dtedpos(1),msl_dtedpos(2));
    %(Approximatly)
    return;
elseif radius>too_far_dist
    dist=-1;
    return;
end

rowlist = round([msl_dtedpos(1):...
    (dtedpos(1)-msl_dtedpos(1))/(numpoints - 1):dtedpos(1)]);
collist = round([msl_dtedpos(2):...
    (dtedpos(2)-msl_dtedpos(2))/(numpoints - 1):dtedpos(2)]);

k=0;
ang=0;
compangle = pi/2 + losang(2);
while(ang < compangle)
    if(k+1>numpoints)
       dist=-1;
       return;
    else
        k=k+1;
    end
    ang=atan(k*steplen/(msl_geoangpos(3) - dtedm(rowlist(k),collist(k))));
end
dist=(msl_geoangpos(3) - dtedm(rowlist(k),collist(k)))/sin(-losang(2));
```

drawLine

```
function out_img = drawLine( img, point1X, point1Y, point2X, point2Y,...
    value )
%DRAWLINE Draws a line in an image based on endpoints of line.
%OUT_IMG = DRAWLINE( IMAGE, P1X, P1Y, P2X, P2Y, VALUE )
%               IMAGE   => The image to draw the line into.
%               P1X     => X-coordinate of first point.
%               P1Y     => Y-coordinate of first point.
```

```
%               P2X     => X-coordinate of second point.
%               P2Y     => Y-coordinate of second point.
%               VAULE   => Value of the pixels in the line.
%               OUT_IMG => The resulting image.

out_img = img;
deltaX = abs(point2X-point1X);
deltaY = abs(point2Y-point1Y);

if(deltaX==0)
    for(y=min(point2Y,point1Y):...
            max(point2Y,point1Y))
        out_img(point1X,y)=value;
    end
    return
elseif(deltaY==0)
    for(x=min(point2X,point1X):...
            max(point2X,point1X))
        out_img(x,point1Y)=value;
    end
    return
end

% Y= a*X + b
a = (point2Y-point1Y)/(point2X-point1X);
b = point1Y - a*point1X;
if(deltaX<deltaY)
    for(y=min(point2Y,point1Y):...
            max(point2Y,point1Y))
        out_img(round((y-b)/a),y)=value;
    end
    return
else
    for(x=min(point2X,point1X):...
            max(point2X,point1X))
        out_img(x,round(a*x+b))=value;
    end
end
```

getAngle

```
function ang = getAngle( x1,y1,x2,y2,x3,y3 )
% GETANGLE      Returns the smallest angle formed by the lines
%               [x1 y1] to [x2 y2] and [x3 y3] to [x2 y2].
% ANG = GETANGLE( X1,Y1, X2,Y2, X3,Y3 )
%               Xn,Yn => Cartesian coordinates.
%               ANG   => Resulting angle in degrees.

v1=[x1-x2,y1-y2];
l1=sqrt(v1(1)^2 + v1(2)^2);
v2=[x3-x2,y3-y2];
l2=sqrt(v2(1)^2 + v2(2)^2);
scalarprod = v1(1)*v2(1) + v1(2)*v2(2);
if(l1*l2==0)
    % Two of the points are equal
    ang=0;
```

```
    return;
end
ang = acos( scalarprod/(l1*l2) ) *180/pi;
```

getMapFile

```
function mapFile = getMapFile( filepath )
%GETMAPFILE Returns the path to the map containing the flight of the
%           recording in the given file.
%MAPFILE = GETMAPFILE( FILEPATH )
%               FILEPATH => Path to the file with the recording.
%               MAPFILE  => The path to the mapfile.

if( size(strfind(filepath,'log_11122003_01'),1)~=0 )
    mapFile='C:\Tor\Data\Kartdata\DTED\E010\N59.DT1';
    return;
elseif( size(strfind(filepath,'log_18032004_02'),1)~=0 )
    if( size(strfind(filepath,'run001'),1)~=0 )
        mapFile='C:\Tor\Data\Kartdata\DTED\E009\N59.DT1';
    elseif( size(strfind(filepath,'run002'),1)~=0 )
        mapFile='C:\Tor\Data\Kartdata\DTED\E008\N60.DT1';
    else
        mapFile=-1;
    end
    return;
elseif( size(strfind(filepath,'log_19032004_01'),1)~=0 )
    mapFile='C:\Tor\Data\Kartdata\DTED\E010\N59.DT1';
    return;
elseif( size(strfind(filepath,'log_20122004_01\'),1)~=0 )
    if( size(strfind(filepath,'run010'),1)~=0 )
        mapFile='C:\Tor\Data\Kartdata\DTED\E007\N59.DT1';
    elseif( size(strfind(filepath,'run004'),1)~=0 )
        mapFile='C:\Tor\Data\Kartdata\DTED\E009\N59.DT1';
        % This run goes over in E008
    end
    return;
end
mapFile=-1;
return;
```

getMeridian_FalseNorthing

```
function [ meridian, fn ] = getMeridian_FalseNorthing( filepath )
%GETMERIDIAN_FALSENORTHING Returns the meridian and false northing of the
%           recording in the given file.
%[ MERIDIAN, FN ] = GETMERIDIAN_FALSENORTHING( FILEPATH )
%               FILEPATH  => Path to the file with the recording.
%               MERIDIAN  => Meridian of the flightfile.
%               FN        => False northing of the flightfile.

if( strfind(filepath,'log_11122003_01')~=0 )
    meridian=10.9083333;
    fn=6577170;
    return;
elseif( strfind(filepath,'log_18032004_02')~=0 )
    meridian=10.7717;
    fn=6583109;
```

```
    return;
elseif( strfind(filepath,'log_19032004_01')~=0 )
    meridian=10.9083333;
    fn=6577170;
    return;
elseif( strfind(filepath,'log_20122004_01')~=0 )
    meridian=9.5814;
    fn=6613176;
    return;
end
meridian=-1;
fn=-1;
return;
```

getNeighbours

```
function neighbours = getNeighbours( in_img, x, y )
%GETNEIGHBOURS returns a (nbOfNeighbours+1)*2 matrise. First column
%              contains x and y coordinates of centerpixel. Rest columns
%              contain x and y coordinates of any present neighbours (that
%              is coordinates of the 8 sourrounding pixels which does not
%              have value zero).
%NEIGHBOURS = GETNEIGHBOURS( IMAGE, X, Y )
%               IMAGE   => The image to look for neighbours in.
%               X       => X coordinate of current pixel.
%               Y       => Y coordinate of current pixel.
%               NEIGHBOURS => A (nbOfNeighbours+1)*2 matrise. First column
%                          contains x and y coordinates of centerpixel.
%                          Rest columns contain x and y coordinates of any
%                          present neighbours (that is coordinates of the 8
%                          sourrounding pixels which does not have value
%                          zero).

[M,N]=size(in_img);
neighbours=[x,y]';
for p=-1:1
    for q=-1:1
        if(~(p==0 && q==0) && x+p>0 && x+p<=M && y+q>0 && y+q<=N ...
            && in_img(x+p,y+q)~=0)
            neighbours=[neighbours';[x+p,y+q]]';
        end
    end
end
```

getVarianceInParallellogram

```
function [var area] = getVarianceInParallellogram( currentWindow, ...
                                                   line1, line2 )
%GETVARIANCEINPARALLELLOGRAM Calculates the variance of the pixels in the
%                            parallellogram formed by the parallell lines
%                            'line1' and 'line2'.
%POINTS = PARALLELLEXTRACTION( TEMPIMG, LINE1, LINE2 )
%               TEMPIMG => The image to calculate the score from.
%               LINE1   => Line forming the border of the parallellogram
%                          together with line2, which is assumed to be
%                          parallell.
%               LINE2   => Line forming the border of the parallellogram
```

```
%                           together with line1, which is assumed to be
%                           parallell.
%               VAR     => The calculated variance.

Xoffset = min(min(line1.point1(1), line1.point2(1)),...
              min(line2.point1(1), line2.point2(1))) -1;
Yoffset = min(min(line1.point1(2), line1.point2(2)),...
              min(line2.point1(2), line2.point2(2))) -1;
maxX = max(line1.point2(1), line2.point2(1));
maxY = max(max(line1.point1(2), line1.point2(2)),...
              max(line2.point1(2), line2.point2(2)));

borderImg=zeros(maxX-Xoffset,maxY-Yoffset);
borderImg=drawLine(borderImg, line1.point1(1)-Xoffset,...
                               line1.point1(2)-Yoffset,...
                               line1.point2(1)-Xoffset,...
                               line1.point2(2)-Yoffset, 1);
borderImg=drawLine(borderImg, line2.point1(1)-Xoffset,...
                               line2.point1(2)-Yoffset,...
                               line2.point2(1)-Xoffset,...
                               line2.point2(2)-Yoffset, 1);
borderImg=drawLine(borderImg, line1.point1(1)-Xoffset,...
                               line1.point1(2)-Yoffset,...
                               line2.point1(1)-Xoffset,...
                               line2.point1(2)-Yoffset, 1);
borderImg=drawLine(borderImg, line1.point2(1)-Xoffset,...
                               line1.point2(2)-Yoffset,...
                               line2.point2(1)-Xoffset,...
                               line2.point2(2)-Yoffset, 1);

% %For showing the borderimage in the original image:
% parallellogramInCurrentWindow=currentWindow;
% for(k=1:size(borderImg,1))
%     for(l=1:size(borderImg,2))
%         if(borderImg(k,l)>0)
%             parallellogramInCurrentWindow(l+Yoffset, k+Xoffset) = 5000;
%         end
%     end
% end
% showImg(normalize(parallellogramInCurrentWindow));

for(k=1:size(borderImg,1))
    state=1;
    if( exist('buffer', 'var') )
        clear buffer;
    end
    for(l=1:size(borderImg,2))
        switch state
            case 1
                if(borderImg(k,l)==1)
                    state=2;
                end
            case 2
                if(borderImg(k,l)==0)
                    state = 3;
                    if( exist('buffer', 'var') )
```

```
                        buffer = [buffer currentWindow(l+Yoffset, ...
                                                       k+Xoffset)];
                    else
                        buffer = currentWindow(l+Yoffset, k+Xoffset);
                    end
                end
            case 3
                if(borderImg(k,l)==0)
                    if( exist('buffer', 'var') )
                        buffer = [buffer currentWindow(l+Yoffset, ...
                                                       k+Xoffset)];
                    else
                        buffer = currentWindow(l+Yoffset, k+Xoffset);
                    end
                else
                    if( exist('varpxls', 'var') )
                        varpxls = [varpxls buffer];
                    else
                        varpxls = buffer;
                    end
                end
        end
    end
end

if( ~exist('varpxls', 'var') )
    var=-1;
    area=0;
    return;
end
area = size(varpxls,2);
var = std(varpxls);
```

jumpSteps

```
function jumpsteps = jumpSteps( filepath, imgNb )
%JUMPSTEPS gets the number of images to descard given a recording and a
%          imagenumber. Due to regularly calibration of sensor, images
%          regularly falls away and should be descarded.
%JUMPSTEPS = JUMPSTEPS( FILEPATH, IMGNB )
%               FILEPATH  => The path to the file with the recording.
%               IMGNB     => The number of the current image in the
%                            recording.
%               JUMPSTEPS => Number of pictures to descard.

jumpsteps=0;
if( size(strfind(filepath,'log_11122003_01'),1)~=0 )
    if(mod(imgNb-2,10)==0)
        jumpsteps=1;
        return;
    end
elseif( size(strfind(filepath,'log_18032004_02'),1)~=0 )
    if( size(strfind(filepath,'run001'),1)~=0 )
        if(mod(imgNb-3,20)==0)
            jumpsteps=3;
            return;
        end
```

```
    elseif( size(strfind(filepath,'run002'),1)~=0 )
        if(mod(imgNb-3,20)==0)
            jumpsteps=3;
            return;
        end
    end
    return;
elseif( size(strfind(filepath,'log_19032004_01'),1)~=0 )
    if( size(strfind(filepath,'run003'),1)~=0 || ...
        size(strfind(filepath,'run002'),1)~=0)
        if(mod(imgNb-3,20)==0)
            jumpsteps=3;
            return;
        end
    end
    return;
elseif( size(strfind(filepath,'log_20122004_01\'),1)~=0 )
    if(mod(imgNb-3,20)==0)
        jumpsteps=3;
        return;
    end
end
return;
```

magnHough

```
function [magnHoughImg T R] = magnHough( edgeImg, thetares, rhores )
%MAGNHOUGH Houghtransform that accumulate the magnitude of all the pixels
%               that forms the line, instead of the usual one point more
%               for each pixels. See documentation on MATLABs 'hough'
%               method for more information on normal houghtransform.
%[MAGNIMG T R] = MAGNHOUGH( EDGEIMG, THETARES, RHORES )
%               EDGEIMG => The image with the edges to calculate the
%                          houghtransform form.
%               THETARES=> The resolution of the theta.
%               RHORES  => The resolution of the rho.
%               MAGNIMG => The resulting hough image.
%               T       => A convertionvector from index in magnimg to the
%                          actual angle.
%               R       => A convertionvector from index in magnimg to the
%                          actual distance.

[M N]=size(edgeImg);
nbOfThetas = floor(180/thetares);
nbOfRhos = floor(2*sqrt(M^2+N^2)/rhores);
magnHoughImg = zeros(nbOfRhos, nbOfThetas);

T=zeros(1,nbOfThetas);
for(t=1:nbOfThetas)
    T(t)= -90 + t*thetares;
end

R=zeros(1,nbOfRhos);
for(r=1:nbOfRhos)
    R(r)= -floor(sqrt(M^2+N^2)) + r*rhores;
end
clear t r
```

```
deg2rad=pi/180;
for(y=1:M)
    for(x=1:N)
        if(edgeImg(y,x)~=0)
            for(t=1:nbOfThetas)
                theta=T(t);
                rho=x*cos(theta*deg2rad)+y*sin(theta*deg2rad);
                r=floor((rho-R(1)+rhores)/rhores);
                if(r<nbOfRhos)
                    magnHoughImg(r,t)=magnHoughImg(r,t)+edgeImg(y,x);
                end
            end
        end
    end
end
```

Main

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Main.m is the main script of the program.  %%
%% Tor Nordseth, Spring 2005                  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%% General inputparameters

% navFile gives the path to the serie of pictures
navFile = 'C:\Tor\Data\nsmir_land\log_19032004_01\run003';
% mapFile gives the path to the mapdata
mapFile = getMapFile(navFile);
% meridian gives the offset of the meridian in the navigational data
% fn gives the offset of the northing in the navigational data
[meridian,fn]=getMeridian_FalseNorthing(navFile);
% Format indicates the format of the testdata: Either class 0 or 1
Format=1;
% Which picture in the sequence to start at, and which to stop at
startNb=221;
stopNb=221;

% The field of view per pixel is dependent on the camera used. The value of
% this parameter will not be displayed here due to confidentiality.
%FOVAngleOfPixel=XXX; % Angle in radians

% Following four parameters are for use in the calculation of the
% windowsize
distMatrixDim=[1,4]; %How many points to calculate distance to in the image
steplen=1; %Stepsize (in pixels) when stepping for appropriate LOS-angle
stdLimit=0.2; %Limit for accepted differance in coverage of a pixel
maxSizeOfObject=30; % Looks for objects less than this size in width
                    % (meters)

% Threshold to be used in static thresholding
%staticImgThreshold=XXX; % Value in the same specter as the raw
                         % pixelvalues.
```

```
% Dynamic threshold parameters
nbOfStd=5;

% Minimum difference in pixelvalue to be accepted as edge. (used in
% sharpEdgeExtraction)
%lowerEdgeThreshold=XXX; % Value in the same specter as the raw
                         % pixelvalues.
%edgeScoreThreshold=XXX; % Value dependent on the specter of the raw
                         % pixelvalues.

% Parameters in calculation of uniform areas near edges:
varSize=2;
buffer=1;

%Reset any remaining logs
clear thresholdLog dynamicThreshold i list;


%%%%%% Looping through the images

imgNb=startNb-1;
while(imgNb+1<=stopNb)
    imgNb=imgNb+1;

    % chech for need of jumping over some images
    jumpsteps = jumpSteps(navFile, imgNb);
    if(jumpsteps~=0)
        imgNb=imgNb+jumpsteps;
        continue;
    end

    % reading the image
    try
        img=readir(navFile,imgNb);
        %readir is a KDA-method, not given in this report.
    catch
        errmsg = lasterr;
        if(strfind(errmsg, 'Cannot read ir image'))
            disp(['Missing ir image: ',int2str(imgNb)]);
        else
            disp(['Unknown error encountered: ',errmsg]);
        end
        continue;
    end

    normBinimg=normalizeBin(img); %For imshow usage later
    [N M]=size(img);
    M=M/(1+Format);

    %Converting navigation data and adjusting for false meridian and
    %northing.
    [atr, navdata] = readatr(navFile,imgNb);
    %readir is a KDA-method, not given in this report.
    fn = ones(1,size(navdata,2)).*fn;
    navdata(1,:) = navdata(1,:)+fn;
```

```
navdata = atr2mk3(atr,navdata, meridian); clear atr;
%atr2mk3 is a KDA-method, not given in this report.


%%%%%% Maintaining dynamic threshold
nbOfSample=round(N*M*0.01);
for(i=1:nbOfSample)
    list(i)=img(ceil(N*rand()),(Format*M/2)+ceil(M*rand()));
end

x = mean(list)+nbOfStd*std(list);
if(~exist('dynamicThreshold'))
    dynamicThreshold = x;
end
dynamicThreshold = 0.9*dynamicThreshold + 0.1*x;

% Logging threshold evolution
if(exist('thresholdLog'))
    thresholdLog(size(thresholdLog,1)+1,1)=imgNb;
    thresholdLog(size(thresholdLog,1),2)=dynamicThreshold;
else
    thresholdLog(1,1)=imgNb;
    thresholdLog(1,2)=dynamicThreshold;
end


%%%%%% Calculation of size of windows

% Distributing point equally around the image
imgpos=sprayAcrossImage(distMatrixDim, img, Format);

% Calculating the distance to selected pixels across the image
pxl_cov=zeros(1,size(imgpos,2));
for(k=1:size(imgpos,1))
    for(l=1:size(imgpos,2))
        dist = dist_to_pixel( navdata, mapFile,...
            [imgpos(k,l,2), imgpos(k,l,1)] ) * FOVAngleOfPixel;
        if(dist==-1)
            disp(['Looking above horizon: image ',...
                int2str(imgNb)]);
        end
        pxl_cov(k,l)= dist;
    end
end
% pxl_cov now contains the coverage of the pixels (in meters) in
% corresponding positions in the image as the given imagepositions.

% Some checking on the usability of fixed sized window over whole image
meanWidth=mean(pxl_cov(:));
maxDiff=max(max(pxl_cov(:))-meanWidth, meanWidth-min(pxl_cov(:)));
if(maxDiff/meanWidth > stdLimit)
    disp(['A fixed size window is not appropriate: image ',...
        int2str(imgNb)]);
end
clear maxDiff
```

```
% covrage of a pixel * max size of object = size of sliding window
windowWidth = ceil(maxSizeOfObject / meanWidth);
nbOfWindows = floor((size(img,2)/(Format+1))/windowWidth);
Ww = ones(1,nbOfWindows) .* ...
    floor((size(img,2)/(Format+1))/nbOfWindows);
rest=mod((size(img,2)/(Format+1)),nbOfWindows);
for(g=1:rest)
    Ww(g)=Ww(g)+1;
end

clear windows;
windows(1,1)=0;
for(g=2:size(Ww,2)+1)
    windows(1,g)=windows(1,g-1)+Ww(g-1);
end
windows(1,1)=1;
clear meanWidth rest Ww windowWidth g

%Each window is made of two smaller neighbouring windows,
%which overall gives one less window
nbOfWindows=nbOfWindows-1;


%%%%%% Looping through the windows

%Creating a matrix with the scores from each feature and sum of these,
%for each window
points=zeros(5,nbOfWindows);

% Looping over the windows for each image, to estimate magnitude of
% features of man-made-objects
for(i=1:nbOfWindows)
    currentWindow=img(:,  Format*size(img,2)/4+windows(i):...
                    Format*size(img,2)/4+windows(i+2));


    %%%%%% Thresholding
    % Static threshold
    tre=threshold(currentWindow, staticImgThreshold);
    thresholdingPoints=sum(tre(:));
    points(1,i)=round(thresholdingPoints);
    % Dynamic threshold
    tre=threshold(currentWindow, dynamicThreshold);
    thresholdingPoints=sum(tre(:));
    points(2,i)=round(thresholdingPoints);


    %%%%%% Sharp edges
    points(3,i) = sharpEdgeExtraction(currentWindow,...
        lowerEdgeThreshold, edgeScoreThreshold);


    %%%%%% Parallell edges
```

```
        points(4,i) = parallellExtraction(currentWindow, 'sobel');


        %%%%%% Uniform areas near edges
        pixelpoints = varianceByEdge(currentWindow, varSize, buffer);
        points(5,i) = round(averageValueOfTop(pixelpoints,0.1));
        clear pixelpoints

    end
    clear currentWindow tempnorm tre thresholdingPoints imgThreshold ...
        thresholdingPoints edgeThreshold edgemagnitude

    %%%%% Keeping interesting areas
    % Clearing areas in image with little sign of man-made-objects
    img2=zeros(64,1024);
    for(k=1:nbOfWindows)
       %score = dot(points(:,k) , [thresholdParameter parallellParameter]);
        if(         points(2,k)>= 1 ...
                ||  points(3,k)>=47 ...
                ||  points(5,k)>= 5 )
            img2(:,windows(k):windows(k+2)) = ...
                normBinimg(:,Format*size(img,2)/4+windows(k):...
                             Format*size(img,2)/4+windows(k+2));
        end
    end

    % Displaying points
    disp(['Img ' int2str(imgNb) ' got points:']);
    points

    %Showing resulting image on screen
    figure;
    subplot(2,1,1);
    imshow(normBinimg(:,Format*size(img,2)/4:...
                        3*Format*size(img,2)/4));
    title('original');
    subplot(2,1,2);
    imshow(img2);
    clear img2

end % End of loop on images
```

normalizeBin

```
function out_img = normalizeBin( in_img )
%NORMALIZEBIN Converts the imagevalues to values between 0 and 1.
%OUT_IMG = NORMALIZEBIN( IMAGE )
%               IMAGE   => The image to normalize.
%               OUT_IMG => The resulting image.

out_img = in_img - min(min(in_img));
m=max(max(out_img));
if(m~=0)
    out_img = out_img /m;
end
```

parallellExtraction

```
function points = parallellExtraction( currentWindow, edgeMethod)
%PARALLELLEXTRACTION Calculates a score from the amount of parallell lines
%               in the image.
%POINTS = PARALLELLEXTRACTION( IMG, EDGEMETHOD )
%               IMG     => The image to calculate the score from.
%               EDGEMETHOD => The method to used to extraxt the edges in
%                           the image. See documentation of MATLAB method
%                           'edge' for more information.
%               POINTS  => The amount of points the image gets.

points=0;

% Extracting the edges
edgeImg=edge(currentWindow, edgeMethod);

% Removing unsignificant edges (descarded as noise).
edgeImg=remTriples(edgeImg);

% Doing the Hough.
[H,T,R] = hough(edgeImg, 'Thetaresolution',0.2,'Rhoresolution',0.5);
Hcut=H;

%%%%%% A discarded attemt to improve Hcut %%%%%%%%
%
%    %The points are scaled on the basis of how many pixels they have run
%    %across in the original image.
%     [n,m] = size(edgeImg);
%     k1=(m*0.2-n*0.2)/2;
%     k2=2*pi/180;
%     for(q=1:size(T,2))
%         Hcut(:,q) = round(Hcut(:,q) ./ ...
%               ((ones(size(H,1),1).*n + k1.*(1 - cos(k2*T(q))))/100));
%     end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% This (below) might look like a greate complexity, but is konstant with
% regards to the problemsize (size of image and number of images).
% Worst case runtime is less than maxNbOfPeaks^2.
maxNbOfPeaks=50;
allowGapInLine=4;
neighbourhood=[9,15];
% By empiric testing: Lines should be at least 20% of objectsize
% (set earlier).
minLengthOfLines = round( size(currentWindow,2) / 2 * 0.15 );
peaks = houghpeaks(Hcut,maxNbOfPeaks,...
           'Threshold',minLengthOfLines,...
           'NHoodSize',neighbourhood);
if(size(peaks,1)==0)
    return
end
lines = houghlines(edgeImg, T, R, peaks,...
                   'FillGap',allowGapInLine,...
```

```
                    'MinLength',minLengthOfLines);

nbOfAcceptedParallellograms=0;
if(size(lines,2)>1)
    linesAngleAndLength = zeros(size(lines,2),2);
    for(k=1:size(lines,2))
        linesAngleAndLength(k,1) = round(lines(k).theta);
        linesAngleAndLength(k,2) = sqrt(round(...
            (lines(k).point1(1)-lines(k).point2(1))^2 +...
            (lines(k).point1(2)-lines(k).point2(2))^2 ));
        if(k==1)
            continue;
        end
        for(j=k-1:-1:1)
            if( (linesAngleAndLength(k,1) >= linesAngleAndLength(j,1)-1 ...
              && linesAngleAndLength(k,1) <= linesAngleAndLength(j,1)+1)...
                    && acceptedLine(edgeImg, lines(j))...
                    && acceptedLine(edgeImg, lines(k)))
                % To prevent parallell lines on totaly different places in
                % the image to act as opposite sides of a parallellogram,
                % they are checed to see if they overlap in their common
                % direction.
                longest = 0;
                shortest = 0;
                if(linesAngleAndLength(k,2) < linesAngleAndLength(j,2))
                    longest = j;
                    shortest = k;
                else
                    longest = k;
                    shortest = j;
                end
                ang1 = getAngle(...
                     lines(longest).point1(1),lines(longest).point1(2),...
                     lines(longest).point2(1),lines(longest).point2(2),...
                     lines(shortest).point2(1),lines(shortest).point2(2));
                ang2 = getAngle(...
                     lines(longest).point2(1),lines(longest).point2(2),...
                     lines(longest).point1(1),lines(longest).point1(2),...
                     lines(shortest).point1(1),lines(shortest).point1(2));
                ang = min(ang1,ang2);
                % Distance calculation:
                dx=abs(lines(longest).point2(1)-lines(shortest).point2(1));
                dy=abs(lines(longest).point2(2)-lines(shortest).point2(2));
                dist=round(sin(ang1*pi/180)*sqrt(dx*dx+dy*dy));
                clear ang1 ang2 dx dy;
                % "dist=abs(lines(k).rho - lines(j).rho)" is no good
                % because some rho may be positiv or negative when theta is
                % close to 90.
                ang = tan(ang*pi/180);
                if( ang~=0 && dist>0 &&...
                     dist/ang <= linesAngleAndLength(longest,2) )
                    % Mark: Possibly more than 2 parallell lines.
                    [var area] = getVarianceInParallellogram(...
                        currentWindow, lines(shortest), lines(longest));
                    if(area==0)
                        continue
                    end
```

```
                    if(var==0)
                        var=1; %Cannot devide by zero.
                    end
%                     disp(['L=' int2str(longest)...
%                         ', S=' int2str(shortest)...
%                         ', var=' int2str(round(var))...
%                         ',area=' int2str(area)]);
                    % Variance over 12 signal an area containing large dark
                    % and bright areas.
                    if(var<12)
                        nbOfAcceptedParallellograms = ...
                            nbOfAcceptedParallellograms+1;
                        points = points + ...
                            round(10000*area/...
                            (prod(size(currentWindow)) * var) );
                    end
                end
            end
        end
    end
end
```

remTriples

```
function out_img = remTriples( in_img )
%REMTRIPLES Removes pixelclusters with less than four pixels.
%OUT_IMG = REMTRIPLES( IMAGE )
%               IMAGE   => The image to remove clusters from.
%               OUT_IMG => Image without the clusters.

out_img=in_img;
[M,N]=size(in_img);
for(i=1:M)
    for(j=1:N)
        if(out_img(i,j)>=1)
            n1=getNeighbours(in_img,i,j);
            switch size(n1,2)
                case {1} % No neighbours
                    %remove singel
                    out_img(i,j)=0;
                case {2} % One neighbour
                    if(size(getNeighbours(in_img,n1(1,2),n1(2,2)),2)==2)
                        %remove double
                        out_img(i,j)=0;
                        out_img(n1(1,2),n1(2,2))=0;
                    end
                    % This could still be part of a trippel, but it will be
                    % removed when another pixel in trippel hits case 3.
                case {3} % Two neighbours
                    n2=getNeighbours(in_img,n1(1,2),n1(2,2));
                    n3=getNeighbours(in_img,n1(1,3),n1(2,3));
                    if( (size(n2,2)==2 && size(n3,2)==2) || ...
                         ... %n1 has two neighbours, which both have only
                         ... %one neighbour each (must be n1)
                            (size(n2,2)==3 && size(n3,2)==3 && ...
```

```
                            ... %n1 and n2 both has two neighbours, means
                            ... %n3 has to have 2 neigbours beeing n1 and
                            ... %n2 for this to be a trippel
                                ((n3(1,1)==n2(1,2) && n3(2,1)==n2(2,2))...
                                                   || ...
                                 (n3(1,1)==n2(1,3) && n3(2,1)==n2(2,3)) ...
                                ) ...
                            ) ...
                        )
                         %remove triplet
                         out_img(i,j)=0;
                         out_img(n1(1,2),n1(2,2))=0;
                         out_img(n1(1,3),n1(2,3))=0;
                    end
                end
            end
        end
end
```

sharpEdgeExtraction

```
function points = sharpEdgeExtraction( currentWindow,...
    lowerEdgeThreshold, edgeScoreThreshold )
%SHARPEDGEEXTRACTION Calculates a score based on how sharp and long the
%                straight edges in the image are. The longer and sharper,
%                the better score.
%POINTS = SHARPEDGEEXTRACTION( CURRENTWINDOW, LOWEREDGETHRESHOLD,
%                              EDGESCORETHRESHOLD)
%                CURRENTWINDOW => The image to extract the edges from.
%                LOWEREDGETHRESHOLD => The threshold to descard edges lower
%                              than.
%                EDGESCORETHRESHOLD => The threshold to descard houghpixels
%                              lower than.
%                POINTS  => The score the image gets, based on how long and
%                              sharp edges it contains.

edgemagnitude = verticalDerivative(currentWindow, 1) + ...
                verticalDerivative(currentWindow', 1)';
edgemagnitude = remTriples(cutBelow(edgemagnitude,lowerEdgeThreshold));
[magnHoughImg T R] = magnHough(edgemagnitude, 0.2, 0.5);
magnHoughImg=cutBelow(magnHoughImg,edgeScoreThreshold);
points = round(sum(magnHoughImg(:)) / (300*prod(size(currentWindow))));
```

sprayAcrossImage

```
function imgpos = sprayAcrossImage( distMatrixDim, img, Format )
%SPRAYACROSSIMAGE Finds imagepositions distributed around the image (in a
%                 grid system) with the number of positions in the hight
%                 and width as given in DISTMATRIXDIM.
%IMGPOS = SPRAYACROSSIMAGE( DISTMATRIXDIM, IMG, FORMAT )
%                DISTMATRIXDIM => Number of rows and columns of grid.
%                IMG     => The image to spread across.
%                FORMAT  => The type of format of the image (0 or 1).
%                IMGPOS  => The resulting imagepositions.

imgpos=zeros(distMatrixDim);
offset = Format*size(img,2)/4;
```

```
if(Format)
    img2=img;
    img=img2(:,offset:3*offset);
end
for(i=1:distMatrixDim(1))
    imgpos(i,:,1)=floor(i*size(img,1)/(1+distMatrixDim(1)));
end
for(i=1:distMatrixDim(2))
    imgpos(:,i,2)=floor(i*size(img,2)/(1+distMatrixDim(2)));
end
if(Format)
    img=img2;
    imgpos(:,:,2)=imgpos(:,:,2) + offset;
end
```

threshold

```
function thresImg = threshold( img, threshold )
%THRESHOLD Sets pixels less than threshold to 0, and the others to 1.
%THRESIMG = THRESHOLD( IMAGE, T )
%               IMAGE   => The image to threshold.
%               T       => The threshold.
%               THRESIMG => The resulting image.
[M,N]=size(img);
thresImg=zeros(M,N);
for(i=1:M)
    for(j=1:N)
        if threshold>img(i,j)
            thresImg(i,j)=0;
        else
            thresImg(i,j)=1;
        end
    end
end
```

varianceByEdge

```
function pointpicture = varianceByEdge( currentWindow, varSize, buffer)
%VARIANCEBYEDGE Calculates a score for each pixel in a picture, based on
%               the magnitude of the edge in the pixel combined with the
%               variance on one side of the edge. High edgemagnitude
%               combined with low variance gives the highest score. The
%               method is ment to give good scores to pixels on the edge of
%               uniform areas with straight edges.
%POINTPICTURE = VARIANCEBYEDGE( CURRENTWINDOW, VARSIZE, BUFFER)
%               CURRENTWINDOW => Image with pixels to calculate scores
%                           from.
%               VARSIZE => Number of pixels in radius on area to calculate
%                           variance in (area is square). 1 gives 3x3 area,
%                           2 gives 5x5 etc.
%               BUFFER =>  Number of pixels between edgepixel and
%                           variancearea.
%               POINTPICTURE => The same picture as currentWindow, but
%                           greyvalues swopped with points/score of the
%                           pixel.

[M,N] = size(currentWindow);
```

```
pointpicture = zeros(M,N);
if(M<2*varSize || N<2*varSize)
    return;
end

% Calculating a variance image
var = zeros(M,N);
for(k=varSize+1 : M-varSize)
    for(l=varSize+1 : N-varSize)
        vec = currentWindow(k-varSize:k+varSize,l-varSize:l+varSize);
        var(k,l) = std(vec(:));
    end
end
var = var+ones(size(var)); % We do not want to devide by zero.

vertEdge = verticalDerivative(currentWindow, 1);
horiEdge = verticalDerivative(currentWindow', 1)';
for(k=varSize*2+2+buffer : M-varSize*2-2-buffer)
    for(l=varSize*2+2+buffer : N-varSize*2-2-buffer)
        p1 = vertEdge(k,l) /...
            min(var(k,l+varSize+1+buffer),var(k,l-varSize-1-buffer));
        p2 = horiEdge(k,l) /...
            min(var(k+varSize+1+buffer,l),var(k-varSize-1-buffer,l));
        pointpicture(k,l) = max(p1,p2);
    end
end
```

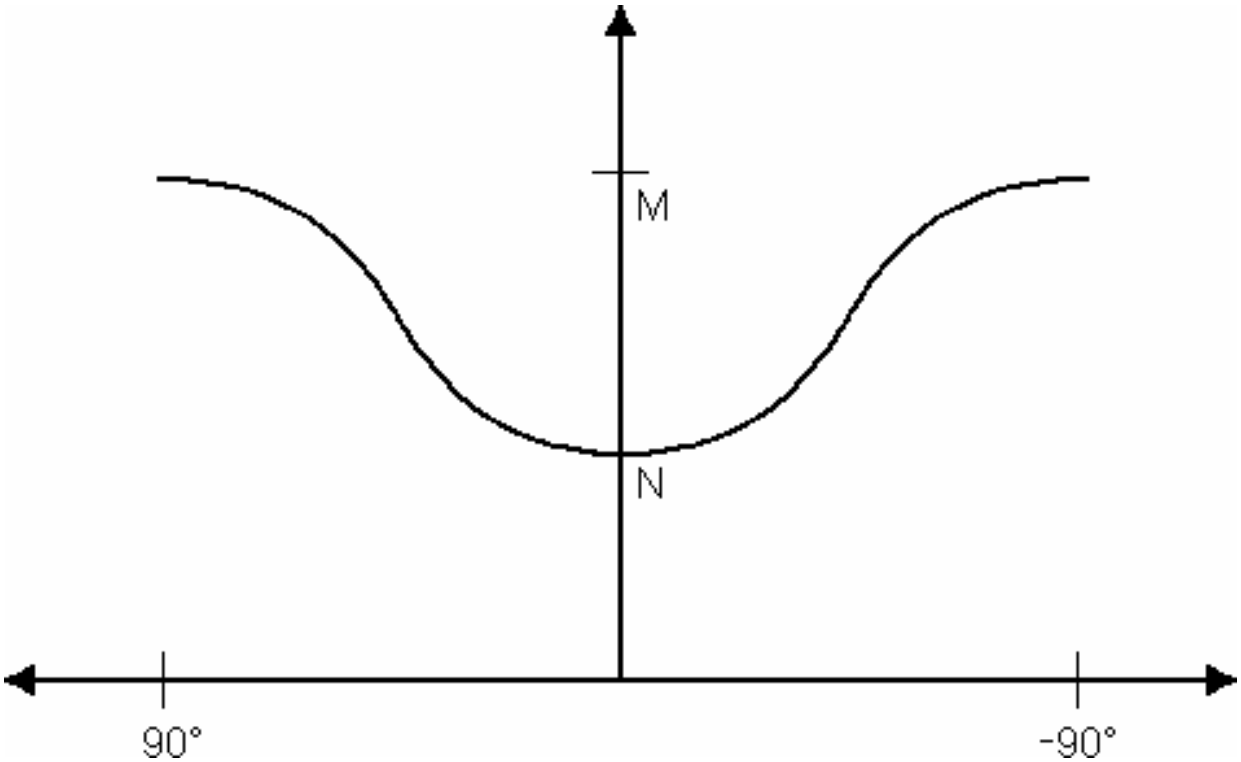
verticalDerivative

```
function [ out_img ] = verticalDerivative( in_img, absDiff )
%VERTICALDERIVATIVE Takes the 1. order derivative of an image in the
%                   vertical direction. (Pixelwise differanse in vertical
%                   direction)
%OUT_IMG = VERTICALDERIVATIVE( IMAGE, ABSDIFF )
%               IMAGE   => The image derive.
%               ABSDIFF => 1 if result should contain absolutevalue of the
%                          derivative, 0 else.
%               OUT_IMG => The resulting image.

[M,N]=size(in_img);
tmp = in_img(1:M,2:N); %Loosing left row
tmp = [tmp, in_img(1:M,N)]; %Duplicating right row
if(absDiff)
    out_img = abs(in_img - tmp);
else
    out_img = in_img - tmp;
end
```

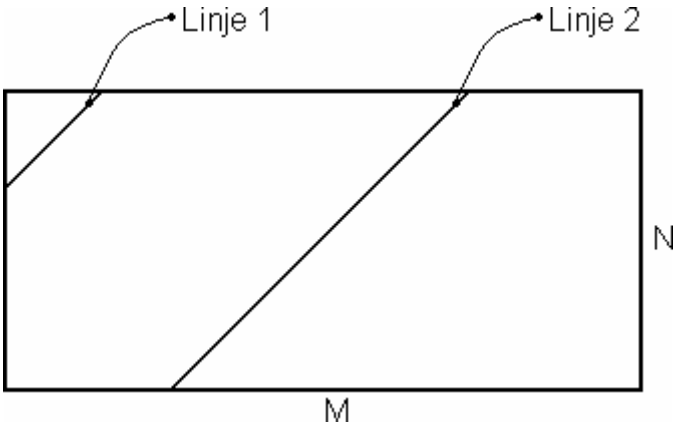
Vedlegg B: Utledning av kompenseringsfunksjon

I dette vedlegget forklares utviklingen av funksjonen for kompensering av poenggivningen i Hough-transformasjonen. Siden Hough-transformasjonen, i bilder med uniformt fordelt støy, favoriserer kanter som ligger i bildets lengderetning (som forklart i rapportens kapittel 4.6.2), ønskes det en funksjon som minker poengsummene til disse i forhold til de vertikale kantene. Forholdet mellom grunnlaget for poenggivningen til de to ytterpunktene er som vist i figur B2; Bildets bredde for linjer med θ-verdi lik 90° eller -90°, og bildets høyde for linjer med θ-verdi lik 0° (For illustrasjon av en linjes θ-verdi, se figur 41 i rapportens kapittel 4.5.2). M og N angir lengden og bredden av bildet.



Figur B1: Graf som viser grunnlaget for poengberegningen

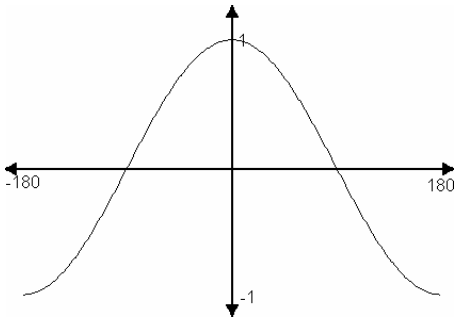
Grunnlaget for kanter med θ-verdi mellom 90° og 0° varierer. Figur B2 viser to linjer med 45°, men som har vidt forskjellige lengder.



Figur B2: To linjer med lik θ-verdi men forskjellig lengde

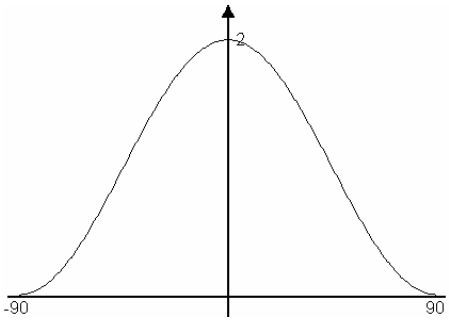
Dermed blir det ingen eksakt kompensering uten å ta ρ-verdien i betraktning i tillegg. Dette er sett på som unødvendig, siden kompensasjonen allikevel ikke vil være eksakt uten å vite hvor den spredte støyen ligger. Det ansees som tilstrekkelig å bruke en middels verdi for kanter med disse vinklene som θ-verdi, som å la formen til en cosinusbølge angi lengden, slik det er antydet i figur B1. Dette medfører dessverre en undertrykking av skrå linjer i hjørnet av bildene, slik som linje 1 i figur B2, siden denne har et poenggrunnlag mindre enn det anvendte snittet.

Med utgangspunkt i cosinusbølgen ble følgende steg foretatt for å konstruere funksjonen vist i figur B1:



$\cos(\theta)$

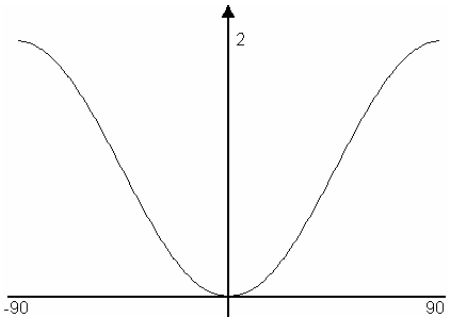
1: Dobling av frekvensen, samt heving av funksjonen til positiv side.



$1+\cos(2\cdot\theta)$

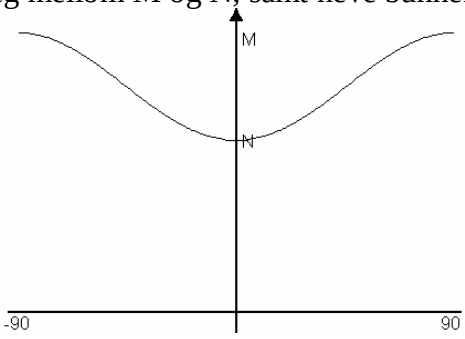
2: Snu funksjonen horisontalt om midten

$$2-(1+\cos(2\cdot\theta))$$



3: Skalere funksjonen så den strekker seg mellom M og N, samt heve bunnen til N

$$N+\frac{M-N}{2}\cdot(2-(1+\cos(2\cdot\theta)))$$



Ved forenkling av uttrykket får man:

$$N+\frac{M-N}{2}\cdot(\cos(2\cdot\theta)-1)$$

Kantene ikke får like mye poeng ekstra som det er forskjell i antall piksler mellom M og N. Kun den andelen av disse som det antas å være støybefengte bidrar til poengsummen. Derfor er det satt inn en parameter, *p*, som skalerer verdiene av M og N til den andelen som antas å inneholde støy. Dermed får man funksjonen:

$$p\cdot N+\frac{p\cdot M-p\cdot N}{2}\cdot(\cos(2\cdot\theta)-1) \quad = \quad p\cdot\left[N+\frac{M-N}{2}\cdot(\cos(2\cdot\theta)-1)\right]$$

I tillegg er det satt inn parametere for å skalere absoluttverdien av funksjonen (det vil si at den er delt på 100), slik at den skal passe med metoden den er brukt til å endre poengsummene.

$$\frac{p}{100}\cdot\left[N+\frac{M-N}{2}\cdot(\cos(2\cdot\theta)-1)\right]$$

Det er denne funksjonen som er brukt for kompenseringen.

Vedlegg C: Testsettet med poengsummer

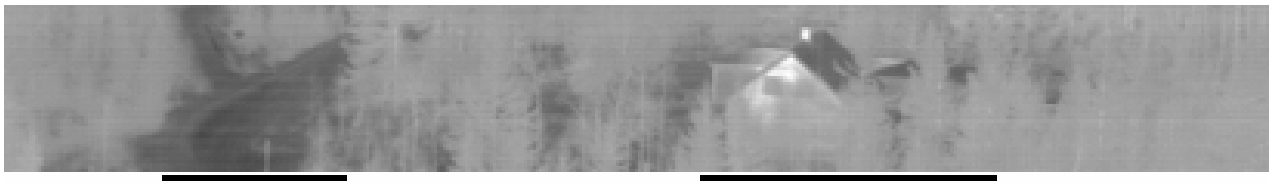
Bildene i testsettet blir her presentert med poengsummene for hver av de fem algoritmene for identifisering av egenskaper. Poengene er listet under ruten i samme rekkefølge som de er blitt presentert her i rapporten:

1) Lyse områder definert ved statisk terskel
2) Lyse områder definert ved dynamisk terskel
3) Markante rette kanter
4) Parallelle linjer
5) Uniforme arealer ved rette kanter

Manuelt identifiserte interessante områder er merket med en strek under bildet.

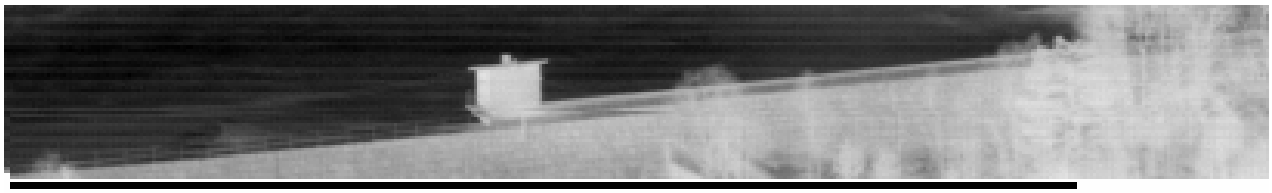
1. del av testsettet:

Bilde nr. 1:



0	5	5
0	3	3
38	45	39
0	0	0
3	3	3

Bilde nr. 2:

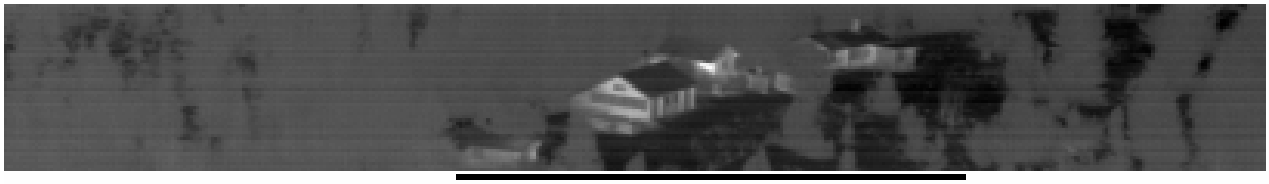


0	0	0
0	0	0
34	35	34
0	0	0
5	5	4

Bilde nr. 3:

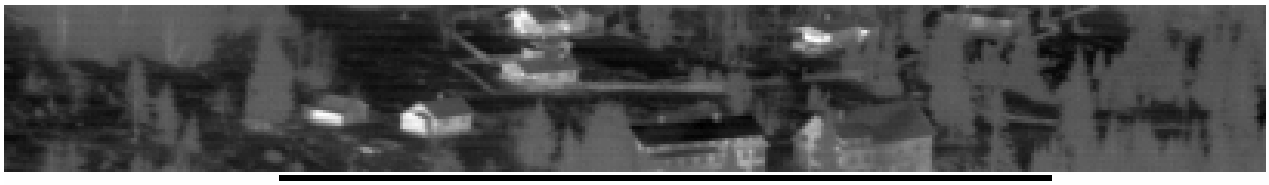
0	0	0	0	0	0
0	0	0	0	0	0
30	35	35	33	34	40
0	0	0	0	0	0
3	4	4	3	3	3

Bilde nr. 4:



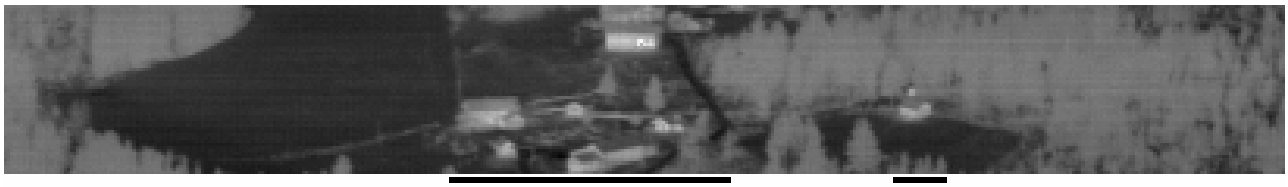
0	11	11	
0	11	11	
32	45	49	
0	0	0	
3	5	4	

Bilde nr. 5:



55	81	42	20	4
13	19	10	4	0
47	62	69	64	49
0	0	0	0	0
4	4	4	4	4

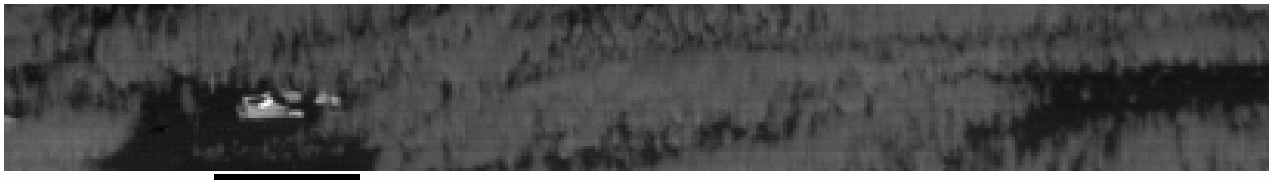
Bilde nr. 6:



0	0	0	0	0	8	8	0	0	0	0	0
0	0	0	0	0	8	8	0	0	0	0	0
26	25	22	27	43	56	50	39	34	28	33	36
96	0	0	12	12	9	0	0	0	122	120	0
4	4	3	3	4	4	3	3	3	3	3	3

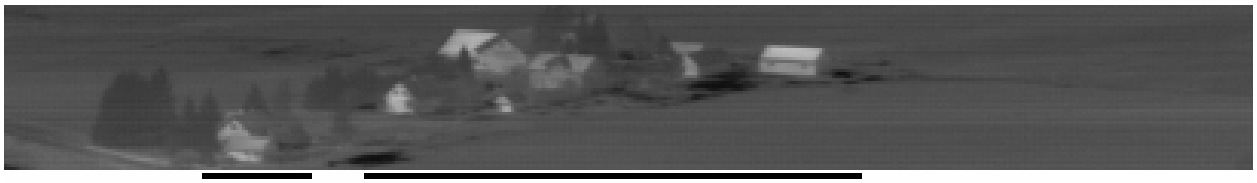
2. del av testsettet:

Bilde nr. 1:



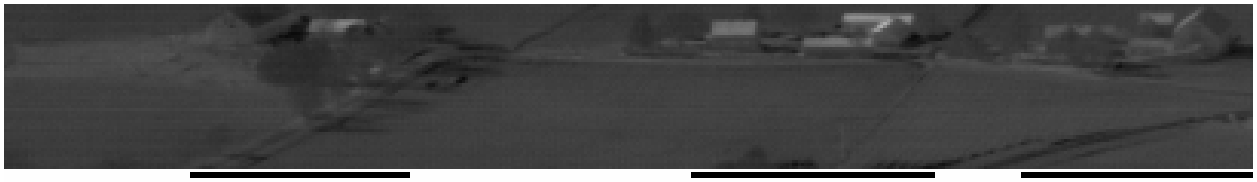
1	70	70	0	0	0	0	0	0
0	22	22	0	0	0	0	0	0
42	47	47	42	43	39	41	42	39
0	0	0	0	0	0	0	0	0
3	5	4	3	3	3	3	3	4

Bilde nr. 2:



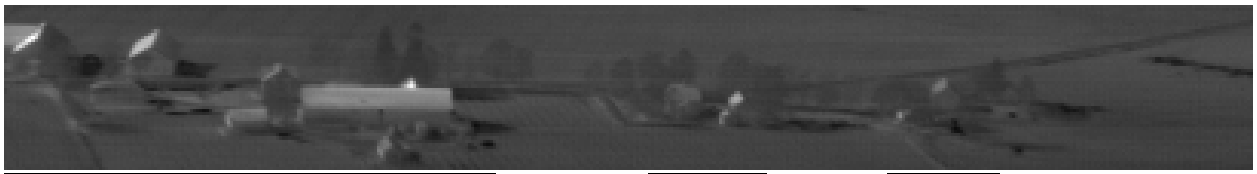
15	67	194	137	45	110	74	0	0
0	2	47	45	17	53	39	0	0
28	34	45	45	39	38	28	21	21
0	0	8	0	3	14	11	277	581
3	3	4	5	4	5	4	3	3

Bilde nr. 3:



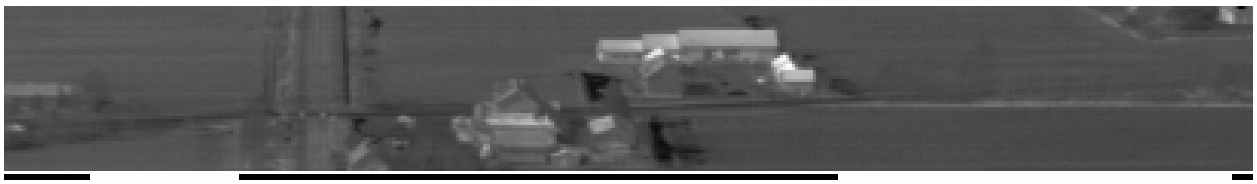
0	0	0	0	0	0	0	5	5	0	0	0
0	0	3	3	0	0	0	28	32	4	0	0
17	22	29	28	21	19	24	35	37	31	29	27
137	43	36	32	9	15	454	23	10	21	33	51
3	3	3	3	3	3	3	4	4	4	4	4

Bilde nr. 4:



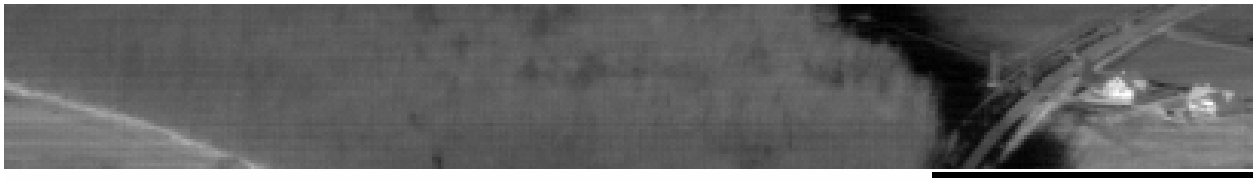
46	23	6	21	15	0	6	6	0	0	0	0
111	33	129	228	105	0	9	9	2	2	0	0
39	36	33	36	30	22	24	25	26	26	22	20
2	6	21	32	57	13	0	9	10	0	25	5
5	5	4	5	4	3	4	4	4	4	4	4

Bilde nr. 5:



6	2	22	186	427	439	175	0	22
5	0	15	164	401	428	173	0	20
27	33	43	54	67	60	37	24	30
14	13	16	17	3	0	13	13	9
4	3	3	4	7	7	5	5	5

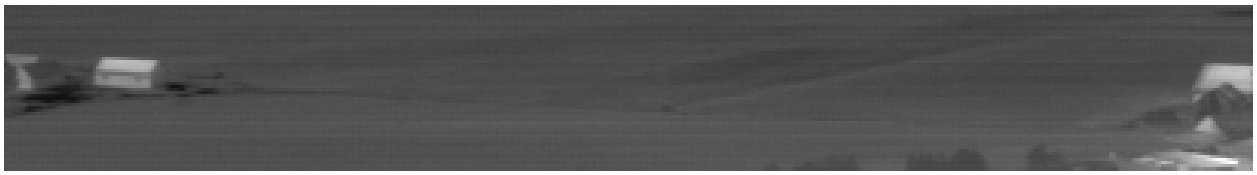
Bilde nr. 6:



0	0	0	0	0	0	0	0	0	54	97
0	0	0	0	0	0	0	0	0	28	35
25	21	18	17	19	18	20	27	41	53	50
10	10	0	2	3	0	0	22	0	3	4
3	3	2	2	2	2	2	3	3	3	4

3. del av testsettet:

Bilde nr. 1:

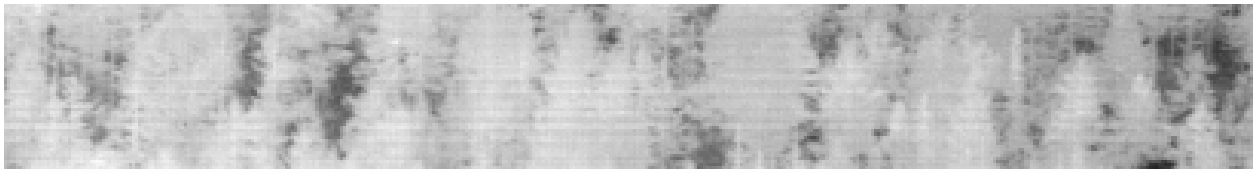


110	74	0	0	0	0	0	92	137
53	39	0	0	0	0	0	23	213
38	28	21	21	22	24	25	32	45
14	11	277	581	484	76	3	0	0
5	4	3	3	3	3	3	3	5

Bilde nr. 2:

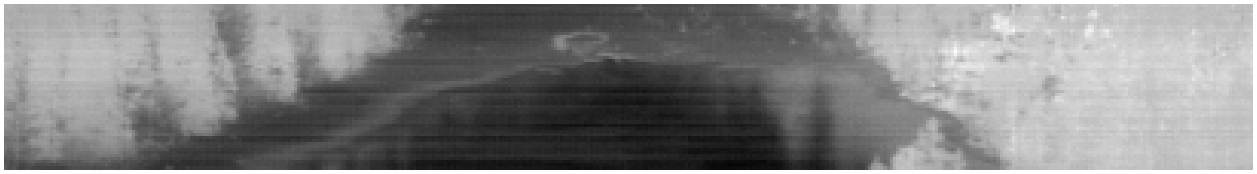
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
32	30	29	29	27	25	25	26	25	21	18
0	17	29	30	27	15	15	12	10	10	0
3	3	3	3	3	3	3	3	3	3	2

Bilde nr. 3:



0	0	5	5
0	0	0	0
27	38	45	39
0	0	0	0
3	3	3	3

Bilde nr. 4:

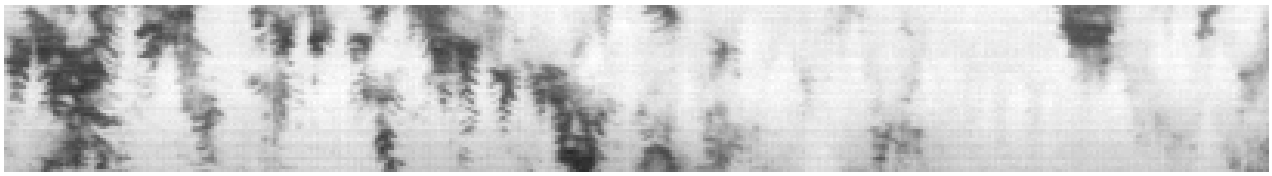


0	0	0
0	0	0
29	28	28
0	0	0
3	3	3

Bilde nr. 5:

0	0	0
0	0	0
31	38	35
0	0	0
3	3	3

Bilde nr. 6:



0	0
0	0
41	29
0	0
4	3

Bilde nr. 7:

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
22	27	29	33	38	43	38	29	26	25	22
34	7	24	24	0	0	0	0	96	0	0
3	3	3	3	3	3	3	3	4	4	3