Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Kybernetikk/Signalbehandling	Vårsemesteret, 2012 ~~Åpen~~ / Konfidensiell
Forfatter: Ole Ronny Andersen	..... (signatur forfatter)
Fagansvarlig: Kjersti Engan Veileder(e): Kjersti Engan og Tom Ryen	
Tittel på masteroppgaven: Lokalisering av typeskilt i annlegsbilder Engelsk tittel:	
Studiepoeng: 30	
Emneord: Automatisk lokalisering Hough Tranform	Sidetall: 59 + vedlegg/annet: 2 Stavanger, ... dato/år

Universitetet i Stavanger

MASTEROPPGAVE

Teknisk-naturvitenskaplig fakultet

Institutt for data- og elektroteknikk

Av

Ole Ronny Andersen

Lokalisering av typeskilt i anleggsbilder

Veiledere: Kjersti Engan og Tom Ryen

Forord

Dette er den avsluttende oppgaven for masterstudiet til Ole Ronny Andersen ved Universitetet i Stavanger. Masteroppgaven tilsvarer 30 studiepoeng, og er gjennomført ved Universitetet i Stavanger i løpet av våren 2012 i sammarbeid med Verico AS.

Prosjektet bygger på videre på et forprosjekt som ble påbegynt høsten 2011, sammen med Thomas Ivesdal-Tronstad.

Prosjektet har vært lærerikt, interresant og utfordrende. Valg av oppgave ble gjort ut fra interesse for de praktiske utfordringene med oppgaven, og generell interesse for bildebehandling.

Jeg vil takke veilederne Kjersti Engan og Tom Ryen ved Universitetet i Stavanger, for gode tilbakemeldinger og rettledninger underveis. Jeg vil også takke medstudentene mine Thomas Ivesdal-Tronstad og Jan Heldal og for deres kunnskapsdeling. Og til slutt vil jeg takke Verico AS som har vært veldig imøtekommende og hjelpsomme.

Sammendrag

Denne rapporten tar for seg en algoritme som automatisk lokaliserer type-skilt i anleggsbilder. Den store variasjonen i struktur, størrelse, form, farger, bakgrunner og vinkel bilder er tatt fra har gjort oppgaven utfordrende. Rapporten vil først gå gjennom noe grunnleggende nødvendig teori som ligger til grunn i den utviklede algoritmen. Videre vil algoritmen som er blitt utviklet presenteres og bli gjennomgått med noen foklaringer. Algoritmen vil så bli testet på tre forskjellige testsett, de resultatene som algoritmen ender opp med vil det bli målt ytelsen til. Til slutt er det en konklusjon og forslag til videre arbeid.

Innhold

Liste Over Figurer

Liste Over Tabeller

1 Introduksjon

Hensikten med arbeidet i denne rapporten har vært å produsere en algoritme for automatisk lokalisering av type-skilt i digitale bilder. Grunnen for at dette skal automatiseres er på grunn av at Verico AS har en voksende database over anleggsbilder som skal klassifiseres, og en manuell klassifisering av en stor og voksende mengde bilder krever mye tid. Hensikten med lokaliseringen er for at en senere automatisk tekstgjenkjenning på type-skiltene skal enklere kunne bli utført. Denne prossesen med tekstgjenkjenningen er ikke en del av denne oppgaven.

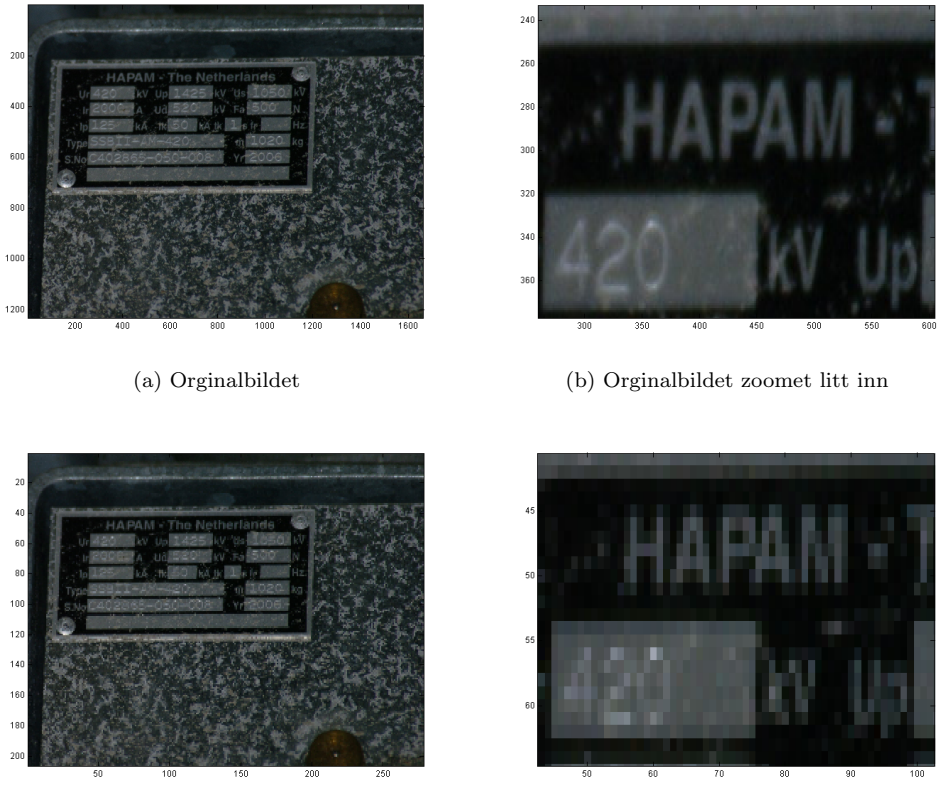
Jeg vil videre anbefale å ha pdf-versjonen tilgjengelig når rapporten blir gjennomgått. Dette er på grunn av at det er en del figurer som er litt små i størrelse, men som kan forstørres opp en del.

2 Teori

Teori kapittelet presenterer grunnleggende metoder som blir brukt i den endelige algoritmen som blir presentert i kapittel 4.

2.1 Nedsampling

Nedsampling kan brukes i bildebehandling hvis enn har veldig store bilder, det vil si bildet har stort antall piksler eller bildeelementer. Jo flere piksler jo mer må prosesseres. En nedsampling på 2 vil si at vi kaster hvert andre piksel i hver kolonne og rekke fra bildet. Resultatet av en nedsampling blir illustrert i et eksempel nedfor i figur 1. Eksempelet viser en nedsamplingfaktor på 6, det vil si at vi kun beholder hvert 6 piksel i kolonnene og rekkene til bildet.



(a) Orginalbildet

(b) Orginalbildet zoomet litt inn

(c) Resultat etter nedsampling på 6

(d) Nedsamplet bildet zoomet litt inn

Figure 1: Nedsampling

I figur 1 kan det observeres at nedsampling har en lignende virkning som en svak glatting har. Denne svake glattingen vil reduserer høye frekvenser i bildet men den viktigste informasjonen i bildet blir bevart. I dette prosjektet har det blitt brukt nedsampling på 4. Men ved nedsampling på 6 eller mer kunne det forekomme at viktig informasjon ble mistet. Nedsamplingen har ikke hatt stor betydning på selve sluttresultatet utenom at prosesseringstiden har gått betraktelig ned.

[2] [3]

2.2 Wiener filtrering

Et wiener-filter er et 2-Dimensjonalt adaptivt filter som brukes til å fjerne støy med en bestem varians. Hvis det ikke oppgis en varians som skal fjernes beregner filteret selv variansen ut i fra nabopikslene til pikselen som skal prosesseres. I MATLAB heter denne funksjonen **wiener2**.[4] [5]

Middelverdi:

$$\mu = \frac{1}{M \cdot N} \sum_{m,n \in W} f(m,n) \tag{1}$$

- $f(m,n)$ er gråskala verdi til piksel (m,n) i et gråskalabilde.
- W er det lokale vinduet som har størrelsen MxN.

Varians:

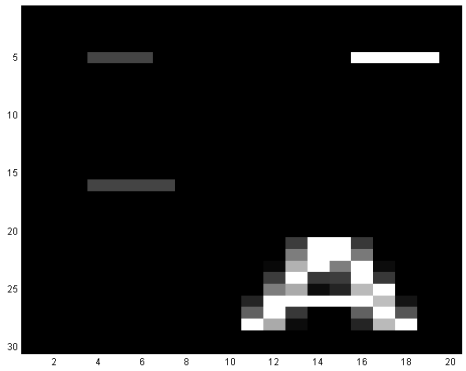
$$\sigma^2 = \frac{1}{M \cdot N} \sum_{m,n \in W} f^2(m,n) - \mu^2 \tag{2}$$

Verdi etter filtrering:

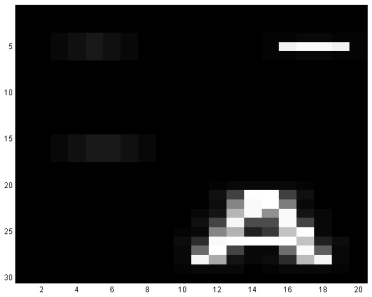
$$g(m,n) = \mu + \frac{\sigma^2 - v^2}{\sigma^2}[f(m,n) - \mu] \tag{3}$$

- $g(m,n)$ er verdi ut etter filtrering
- v^2 er variansen til støyen. Hvis ikke vi har satt noen verdi på v^2 vil den settes til gjennomsnittet av alle lokale varianser til bildet.

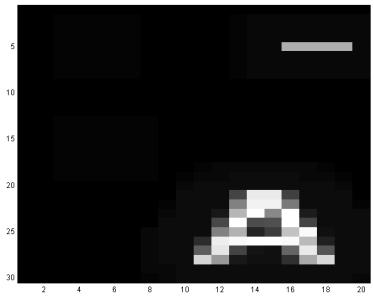
Hvis v^2 settes lik gjennomsnittet av alle lokale varianser i bildet:
Hvis vi har for eksempel har et bilde hvor det er liten varians i store deler av bildet og stor varians i en mindre del av bildet vil gjennomsnittet til alle de lokale variansene bli lav. Det vil igjen føre til at de områdene med lav varians blir glattet mer enn området med høy varians. Dette blir illustrert i figur 2 hvor Wiener-filteret også blir sammenlignet med filtrene Median-filter og Average-filter. Wiener-filteret beregner v^2 ut fra alle lokale varianser selv.



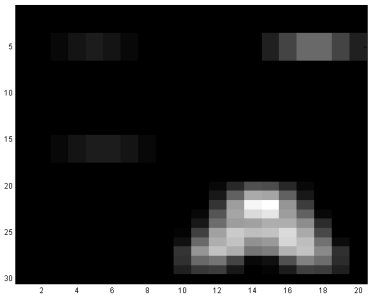
(a) testbilde



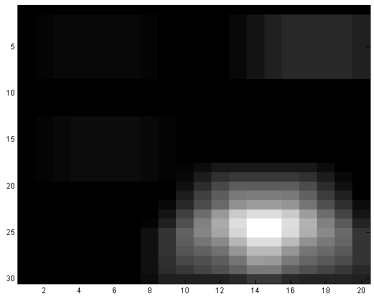
(b) testbilde 'Wiener-filtert' med 3x3-maske



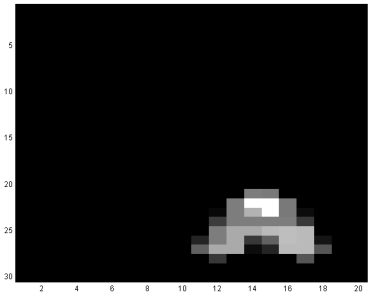
(c) testbilde 'Wiener-filtert' med 7x7-maske



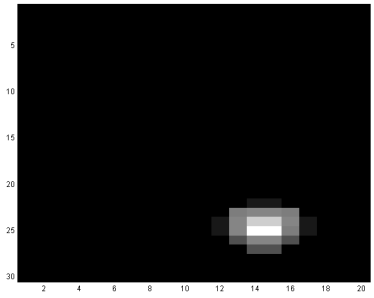
(d) testbilde 'Average-filtert' med 3x3-maske



(e) testbilde 'Average-filtert' med 7x7-maske



(f) testbilde 'Median-filtert' med 3x3-maske



(g) testbilde 'Median-filtert' med 7x7-maske

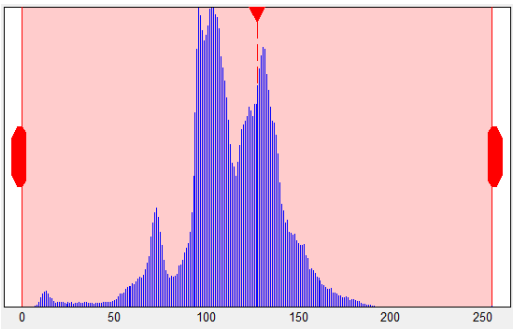
Figure 2: Wiener-filter sammenlignet med Average-filter og Median-filter

2.3 Kontrastforbedring med imadjust.m

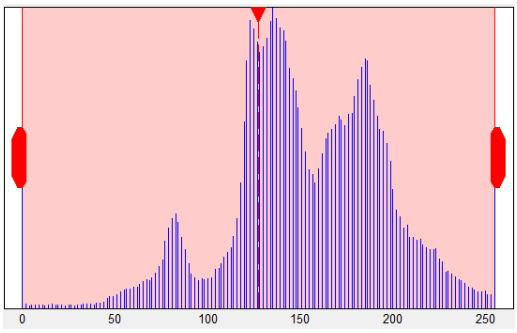
En teknikk som kan brukes til å forbedre kontrasten i et gråskala bilde er å lage en ny 'mapping' på fordelingen til intensitets-verdiene til bildet som prosesseres slik at at minimum og maximum intensitets-verdier på bildet blir satt til en ny minimum og maximum. I Matlab kalles denne funksjonen **imadjust**[1] [6]. Ved *default* setter **imadjust** intensitetverdiene slik at 1% av pikslene i bildet settes til verdien '0' og en annen 1% av pikslene settes til verdien '255'. Det er litt lettere å illustrere hva som skjer ved å se på et enkelt eksempel som viser piksel intensitetsverdiene i et histogram,se figur 3.

(a) Rød-matrisen før prosessering

(b) Rød-matrisen etter prosessering med **imadjust**



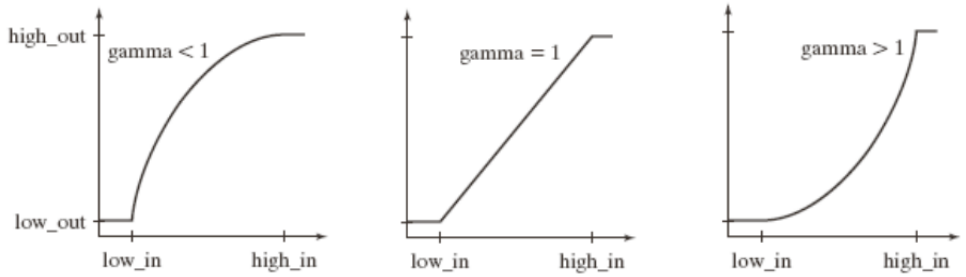
(c) Histogram over intensitets-verdier til Rød-matrisen før prosessering



(d) Histogram over intensitets-verdier til Rød-matrisen etter prosessering med **imadjust**

Figure 3: Imadjust med default parametre

Ved å studere histogrammet i figur 3 d) som er blitt prosessert med **imadjust** funksjonen i MATLAB med default parametre, kommer det frem at 1% av pikslene fått intensitet verdien '0'. Det er også 1% av pikslene til bildet som har fått intesitetsverdien '255'. Vi ser også at dette tydelig har økt kontrasten ved å studere bildene i figur 3 a) og b). Figur 3 d) viser også at resten av verdiene har blitt strukket likt utover histogrammet. Det er fordi ved default gjøres det en linær 'mapping' av intensitets-verdiene. Det trengs ikke å bli utført linær kontrast strekking men det kan også brukes logaritmisk kontrast strekking, dette er illustrert i figur 4



(a) Logaritmisk vektet mot høye intensitets-verdier

(b) Linær (default)

(c) Logaritmisk vektet mot lave intensitets-verdier

Figure 4: De ulike kontrast strekkings metodene til funksjonen **imadjust** [1]

2.4 Kantdeteksjon

Kantdeteksjon er den mest kjente metoden for å detektere endring eller forskjell i intensitets-verdier i et bilde. Dette kan gjøres ved å beregne gradienten som består av den deriverte i horisontale- og vertikale-retning. Jo større endring i intensitet jo større verdi får vi etter kantdeteksjon. Det finnes forskjellige metoder for å tilnærme de deriverte. I denne oppgaven har brukes det en funksjon basert på Sobels tilnærming oppdaget i forprosjekt arbeidet[2], som heter **colorgrad.m** hentet fra [1].

2.4.1 Sobel

Sobel tilnærming til de deriverte går ut på å konvolvere en filter-maske vertikalt og horisontalt med bildet som vi ønsker å finne kanter i. Vanligvis er størrelsen til filter-masken 3×3 det er standardvalget til MATLAB også hvis ikke noe annet er spesifisert. I figur 5 vises den horisontale og vertikale Sobel masken.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(a) Horisontal filter-maske

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

(b) Vertikal filter-maske

Figure 5: Eksempel på Sobel filter-masker

Lengde og vinkel på Sobel gradient:
Vi har et bilde som er et 2D-signal og derfor blir gradienten definert som en vektor $\nabla \mathbf{f}$:

$$\nabla \mathbf{f} = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{4}$$

hvor,

$$g_x = \frac{\partial f}{\partial x} \quad og \quad g_y = \frac{\partial f}{\partial y} \tag{5}$$

Størrelsesordenen eller lengden til gradienten er definert som lengden av vektoren:

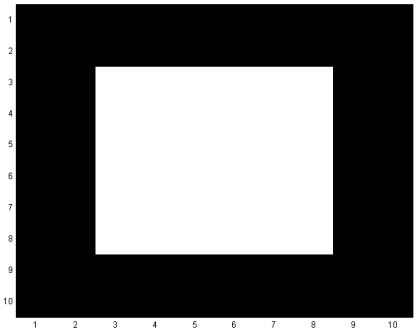
$$\nabla f = [g_x^2 + g_y^2]^{\frac{1}{2}} = \sqrt{g_x^2 + g_y^2} \tag{6}$$

Vinkelen til gradienten er definert vinkelen mellom g_x og g_y:

$$\alpha(x, y) = \tan^{-1}\left(\frac{g_y}{g_x}\right) \tag{7}$$

Eksempel på kantdeteksjon med Sobel-maske som har størrelsen 3×3:

I figur 6 er det bilde av en firkant som vi skal utføre en kantdeteksjon med Sobel-metoden. Til høyre i figuren er matrisen med piksel-verdiene til bildet.



(a) Testbilde

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

(b) Pikselverdier til testbilde

Figure 6: Bildet som skal gjøres Sobel-kantdeteksjon på

g_x finnes ved å konvolvere 'Testbildet' med masken i figur 7:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	-1	-1	0	0	0	0	1	1	0
0	-3	-3	0	0	0	0	3	3	0
0	-4	-4	0	0	0	0	4	4	0
0	-4	-4	0	0	0	0	4	4	0
0	-4	-4	0	0	0	0	4	4	0
0	-4	-4	0	0	0	0	4	4	0
0	-3	-3	0	0	0	0	3	3	0
0	-1	-1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0

Figure 7: Sobel-maske til venstre og resultat g_x til høyre

g_y finnes ved å konvolvere 'Testbildet' med masken i figur 8:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	0
0	-1	-3	-4	-4	-4	-4	-3	-1	0
0	-1	-3	-4	-4	-4	-4	-3	-1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	3	4	4	4	4	3	1	0
0	1	3	4	4	4	4	3	1	0
0	0	0	0	0	0	0	0	0	0

Figure 8: Sobel-maske til venstre og resultat g_y til høyre

g_x^2 og g_y^2:

0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	1	0
0	3	3	0	0	0	0	3	3	0
0	4	4	0	0	0	0	4	4	0
0	4	4	0	0	0	0	4	4	0
0	4	4	0	0	0	0	4	4	0
0	4	4	0	0	0	0	4	4	0
0	3	3	0	0	0	0	3	3	0
0	1	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0

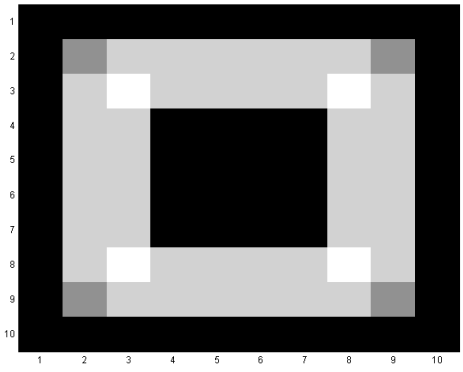
(a) gx^2

0	0	0	0	0	0	0	0	0	0
0	1	3	4	4	4	4	3	1	0
0	1	3	4	4	4	4	3	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	3	4	4	4	4	3	1	0
0	1	3	4	4	4	4	3	1	0
0	0	0	0	0	0	0	0	0	0

(b) gy^2

Figure 9: Resultat av g_x^2 og g_y^2

Sluttresultat til gradienten $\nabla f = \sqrt{g_x^2 + g_y^2}$ ved Sobel-metoden:



(a) Sluttresultat vist i form av bilde

0	0	0	0	0	0	0	0	0	0
0	$\sqrt{2}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{2}$	0
0	$\sqrt{4}$	$\sqrt{6}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{6}$	$\sqrt{4}$	0
0	$\sqrt{4}$	$\sqrt{4}$	0	0	0	0	$\sqrt{4}$	$\sqrt{4}$	0
0	$\sqrt{4}$	$\sqrt{4}$	0	0	0	0	$\sqrt{4}$	$\sqrt{4}$	0
0	$\sqrt{4}$	$\sqrt{4}$	0	0	0	0	$\sqrt{4}$	$\sqrt{4}$	0
0	$\sqrt{4}$	$\sqrt{4}$	0	0	0	0	$\sqrt{4}$	$\sqrt{4}$	0
0	$\sqrt{4}$	$\sqrt{6}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{6}$	$\sqrt{4}$	0
0	$\sqrt{2}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{4}$	$\sqrt{2}$	0
0	0	0	0	0	0	0	0	0	0

(b) Sluttresultat vist i form av matrise-verdier

Figure 10: Resultat av kantdeteksjonen ved bruk av Sobel

I figur 10 ser vi en tydelig kantdeteksjon av firkantobjektet i 'Testbildet' fra figur 6.

2.4.2 Colorgrad

Den vanlige Sobel-metoden fra kapittel 2.4.1 finner bare gradient til gråskala- og binære-bilder, og krever dermed at vi må gjøre om et farge-bilde om til gråskala-bilde for å kunne prosessere bildet. I dette prosjektet har vi veldig stor variasjon i bildene som skal prosesseres dermed viktig av vi beholder så mye informasjon av betydning vi kan. Dermed kommer funksjonen **colorgrad.m** som er hentet fra [1] godt med. Denne funksjonen bygger videre på Sobel-metoden og bruker Sobel-maskene.

Colorgrad.m går ut på at gradienten blir beregnet for hver av de 3 matrisene(Rød Grønn Blå) som et RGB-fargebilde består av. De tre gradient-resultatene for hver farge-matrise blir så slått sammen til et resultat.

Utvider gradient konseptet med 2 vektorfunksjoner som finner endringer i x- og y-retning for de tre fargeplanene og legger endringene sammen:

$$\boldsymbol{u} = \frac{\partial R}{\partial x}\boldsymbol{r} + \frac{\partial G}{\partial x}\boldsymbol{g} + \frac{\partial B}{\partial x}\boldsymbol{b} \tag{8}$$

og

$$\boldsymbol{v} = \frac{\partial R}{\partial y}\boldsymbol{r} + \frac{\partial G}{\partial y}\boldsymbol{g} + \frac{\partial B}{\partial y}\boldsymbol{b} \tag{9}$$

Hvor $\boldsymbol{r}$, $\boldsymbol{g}$ og $\boldsymbol{b}$ er enhetsvektorene til hendholdsvis Rød(1,0,0), Grønn(0,1,0) og Blå(0,0,1) i RGB-rommet.

Finner lengden til til vektorene:

$$g_{xx} = \boldsymbol{u} \bullet \boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{u} = \left|\frac{\partial R}{\partial x}\right|^2 + \left|\frac{\partial G}{\partial x}\right|^2 + \left|\frac{\partial B}{\partial x}\right|^2 \tag{10}$$

$$g_{yy} = \boldsymbol{v} \bullet \boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{v} = \left|\frac{\partial R}{\partial y}\right|^2 + \left|\frac{\partial G}{\partial y}\right|^2 + \left|\frac{\partial B}{\partial y}\right|^2 \tag{11}$$

$$g_{xy} = \boldsymbol{u} \bullet \boldsymbol{v} = \boldsymbol{u}^T\boldsymbol{v} = \frac{\partial R}{\partial x}\frac{\partial R}{\partial y} + \frac{\partial G}{\partial x}\frac{\partial G}{\partial y} + \frac{\partial B}{\partial x}\frac{\partial B}{\partial y} \tag{12}$$

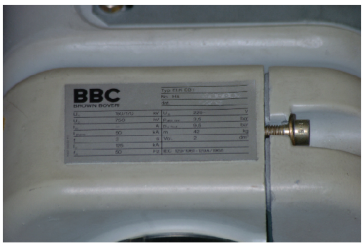
Kan også her finne vinkel og lengde til gradient som vi kunne i Sobel-metoden.

Vinkelendring er definert som:

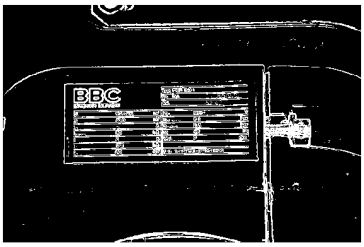
$$\theta(x,y) = \frac{1}{2}\tan^{-1}\left[\frac{2g_{xy}}{g_{xx} - g_{yy}}\right] \tag{13}$$

Endring i lengden til gradienten er definert som:

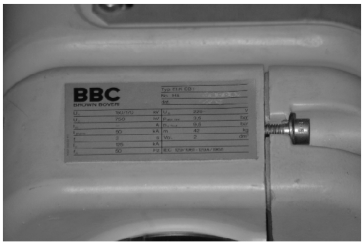
$$F_\theta(x,y) = \left\{\frac{1}{2}\left[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})\cos 2\theta(x,y) + 2g_{xy}\sin 2\theta(x,y)\right]\right\}^{\frac{1}{2}} \tag{14}$$



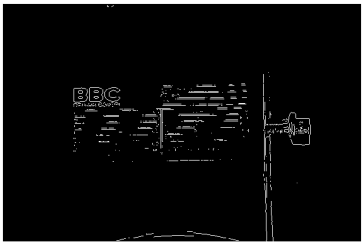
(a) Testbildet (RGB-format)



(b) Resultat av **Colorgrad**, med Terskling=0.08



(c) Testbildet (Gråskala-format)



(d) Resultat av **Sobel**, med Terskling=0.08

Figure 11: Resultat av kantdeteksjonen ved bruk av Sobel og Colorgrad

Ved å studere figur 11 vises det en tydelig forskjell på hvor mye informasjon vi har igjen etter å ha brukt lik terskling på kantdeteksjonen gjort av Colorgrad og Sobel.

2.5 Morfologisk operasjon

I bildebehandling brukes morfologi som et verktøy for å hente ut bildekomponenter som kan være nyttige for å representere eller beskrive formen til et område. I de mofologiske funksjonene som blir gjennomgått her brukes et strukturelement. Et strukturelement er en binær matrise, og kan ha forskjellige former alt etter hvilken hensikt og effekt som er ønskelig på bildet som prosesseres. I denne oppgaven vil det kun bli brukt morfologiske operasjoner for binær bilder, selv om det finnes morfologiske operasjoner for gråskala-bilder [1].

2.5.1 Dilasjon

Dilasjon er en morfologisk operasjon som utvider et objekt eller området i et bilde. Størrelse og formen på utvidelsen er avhengig av formen og størrelsen på strukturelementet. Dilasjon er definert som:

$$A \oplus B = \{z|(\hat{B})_z \cap A \neq \varnothing\}$$

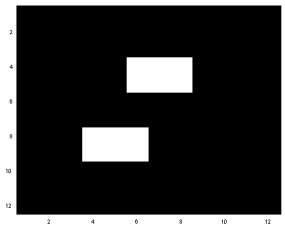
hvor,

- '$\oplus$' er symbolet for dilasjon.
- '$\hat{B}$' er strukturelementet B reflektert.
- 'A' er bildet som skal prosesseres.

I figur 12 er en illustrasjon som viser hvordan en morfologisk dilasjon operasjon fungerer. Strukturelementet er vertikalt og har sitt origo i det midterste elementet markert med blå farge. Strukturelementet vil bli flyttet gjennom hele bildet i figur 12 b) og der som origo til strukturelementet overlapper en 1-er i bildet, settes alle piksler som blir dekket av hele strukturelementet til enere. De røde enerne i figur 12 e) er områder som har blitt satt til ener verdier ved dilasjon.

1
1
1

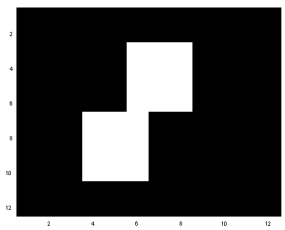
(a) Strukturelementet



(b) Orginalbildet

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

(c) Pikselverdiene til Orginalbildet



(d) Bildet etter dilasjon

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

(e) Pikselverdiene etter dilasjon

Figure 12: Illustrasjon av morfologisk dilasjon

2.5.2 Erosjon

Erosjon er en operasjon som forminsker eller krymper et området eller objekt. Dette gjøres også med et strukturelement. Strukturelementet avgjør formen og hvor kraftig forminskningen blir. Erosjon er definert som:

$$A \ominus B = \{z|(B)_z \subseteq A\}$$

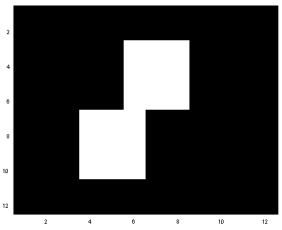
hvor,

- '$\ominus$' er symbolet for erosjon.
- 'B' er strukturelementet.
- 'A' er bildet som skal prosesseres.

I figur 13 er en illustrasjon som viser hvordan en morfologisk erosjon operasjon fungerer. Strukturelementet er her det samme som ble brukt ved dilasjon. Strukturelementet vil bli flyttet gjennom hele bildet i figur 13 b) for å se hvor og hvor hele strukturelementet passer inn i objektet. Objektet er her de områdene som har pikselverdi lik 1. Alle de stedene hvor strukturelementet passet inn i objektet settes pikselverdien som overlappes av strukturelementets origo til 1. Resultatet av erosjon operasjonen er vist i figur 13 c) og d).

1
1
1

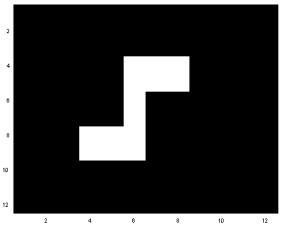
(a) Strukturelementet



(b) Bilde før erosjon

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

(c) Pikselverdiene før erosjon



(d) Bildet etter erosjon

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

(e) Pikselverdiene etter erosjon

Figure 13: Illustrasjon av morfologisk erosjon

2.5.3 Morfologisk lukking

Morfologisk lukking er en kombinasjon av dilasjon og erosjon. Morfologisk lukking er definert som først en dilasjon av 'A' med strukturelementet 'B', etterfulgt av erosjon med det samme strukturelementet av resultatet fra dilasjonen. Det finnes andre kombinasjoner mellom dilasjon og erosjon, men de brukes ikke i denne oppgaven.

Morfologisk lukking er definert som:

$$A \bullet B = (A \oplus B) \ominus B$$

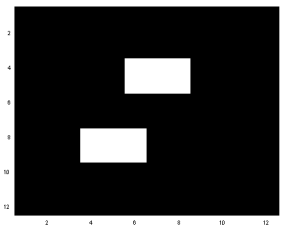
hvor,

- '•' er symbolet for morfologisk lukking.
- 'B' er strukturelementet.
- 'A' er bildet som skal prosesseres.

Ved å sammenligne figur 14 som viser en illustrasjon av morfologisk lukking med figurene 12 og 13, kan det observeres at vi får samme resultat etter en lukking som ved en dilasjon etterfulgt av erosjon. Sluttresultatet i figur 14 d) viser samme resultat som i resultatet i figur 13 d) fordi den viser ersosjon av resultatet fra dilasjon illustrasjonen i figur 12 d) med samme strukturelement.

1
1
1

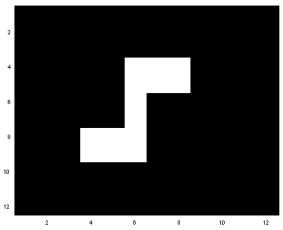
(a) Strukturelementet



(b) Bilde før morfologisk lukking

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

(c) Pikselverdiene før lukking



(d) Bildet etter lukking

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

(e) Pikselverdiene etter lukking

Figure 14: Illustrasjon av morfologisk lukking

2.6 Linje deteksjon ved bruk av Hough Transform (HT)

Hough Transform er en teknikk som kan brukes til å trekke ut karakteristiske trekk i bildebehandling på en effektiv måte. Typisk brukes Hough Transform til å trekke ut rette linjer fra en kantdeteksjon. I praksis vil vi ha forskjellig støy i en kantdeteksjon som gjør at vi ikke har perfekte kanter. Dette kan føre til at vi har diskontinuerlige kanter, og for å prøve å knytte disse kantsegmentene sammen til meningsfulle kanter brukes ofte Hough Transform etter en kantdeteksjon. [1][7]

Vanligvis beskriver vi rette linjer med ligningen:

$$y = ax + b \tag{15}$$

Ved bruk av ligning 15 for å beskrive rette linjer oppstår det problemer fordi stigningstallet **'a'** går mot uendelig når linjer nærmer seg vertikal-retning. Hough Transformasjon beskriver en linje ved bruk av normal representasjon av en linje, som unngår dette problemet. Ligningen som bruker normal representasjon av linje er vist i ligning 16.

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \tag{16}$$

hvor,

- ρ = lengden til den blå stiplede linjen i figur 15 b). Og har rekkvidden: $-D \leqslant \rho \leqslant D$.
- D = diagonalen til bildet i figur 15 b).
- θ = er vinkelen mellom den blå stiplede linjen og den horisontale-aksen i figur 15 b). Og har rekkvidden: $-90 \leqslant \theta < 90$
- x = horisontale-aksen i figur 15 b).
- y = vertikale-aksen i figur 15 b).

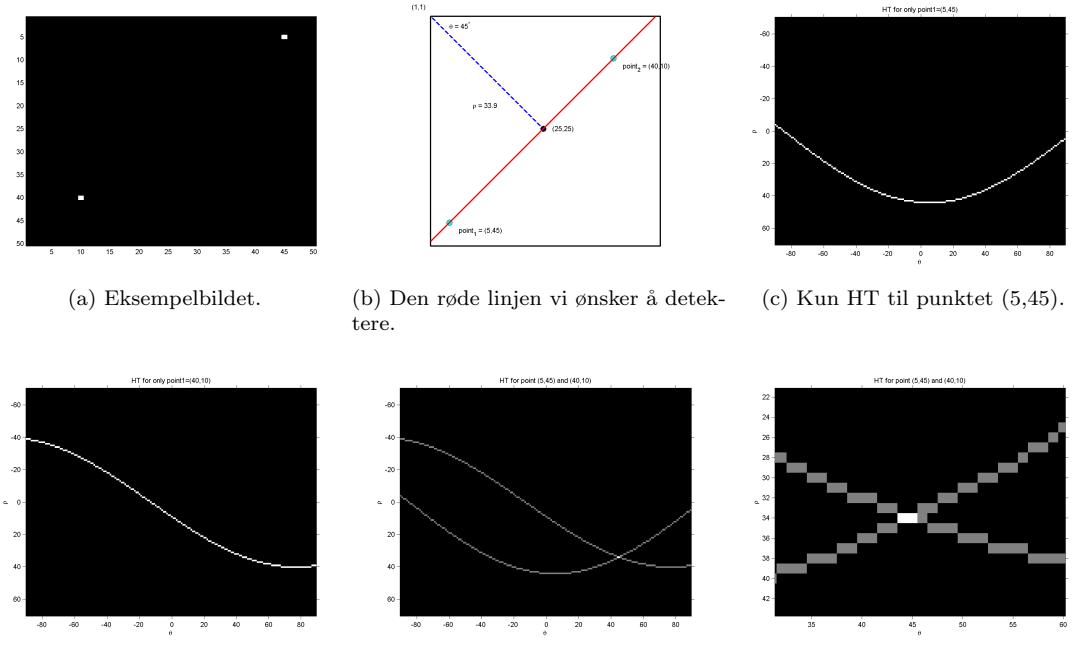
Ved å studere ligning 17, kan vi se sammenhengen mellom ligning 15 og 16:

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \Rightarrow y = \left(-\frac{\cos\theta}{\sin\theta} \right) x + \left(\frac{\rho}{\sin\theta} \right) \tag{17}$$

Eksempel på Hough Transform:

I figur 15 vises et enkelt eksempel på representasjon av en linje som går gjennom to punkt ved bruk at Hough Transform. De to hvite punktene i figur 15 a) representerer 2 piksler i en potensiell linje. Vi ønsker å finne den røde rette linjen ved hjelp av Hough Transform. Den stiplede blå linjen er vinkelrett på den røde linjen som vi vil finne og brukes til å beskrive den rette røde linjen. Koordinatene til de to pikslene er for punkt1 = (5,45) og for punkt2 = (40,10).

Ved å ta Hough Transform for kun et av punktene hver for seg vil vi få resultatet i figur 15 c) og d) som viser en sinusformet bølge. Den totale Hough Transformen til begge punktene er det samme som å kombinerer resultatetene fra figur15 c) og d), og da får vi en topp eller en maksimum i Hough Transformen som kommer frem i figur 15 e) og f). Der hvor de to sinusformede bølgene krysser hverandre kalles en topp, og punktet til toppen har vi en potensiell rett linje som inkluderer begge pikslene. Ellers i figur 15 e) har vi ingen kontakt mellom de to sinus-bølgene og det betyr at vi ikke har en rett linje som inkluderer begge punktene ellers i Hough Transformen. Ved å lese av verdiene til den horisontale- og vertikale-aksen i topp-punktet er $\rho \approx 34$ og $\theta \approx 45$. Disse verdiene var de vi ønsket å finne ifølge 15 b) hvor $\rho \approx 33.9$ og $\theta \approx 45$. [1] [8]



(a) Eksempelbildet.

(b) Den røde linjen vi ønsker å detektere.

(c) Kun HT til punktet (5,45).

(d) Kun HT til punktet (40,10).

(e) Totale HT.

(f) Totale HT ,forstørret inn på maksimums området.

Figure 15: Illustrasjon av Hough Transform(HT).

Hough Transform for gråskala bilder:

I Matlab er det en innebygget Hough Transform for svart-hvitt bilder. Men ved bruk av colorgrad.m mottar vi en kantdeteksjon som er gråskala format. Fra forprosjektet ble det observert at å gå fra gråskala til svart-hvitt bilde for å gjøre en Hough Transform på svart-hvitt bilde, kunne det tapes viktig informasjon og støy kunne også bli forsterket [2]. Videre ble det oppdaget en Hough Transform for gråskala bilder. Dermed unngås tersklingen i overgang til svart-hvitt bilder og vi kan bevare viktig informasjon.

Fra MATLAB's fildelingside [9] ble den gråskala Hough Transformasjon funksjonen funnet.

Filen heter Hough_Grd.m og har to parametre som kan justeres på. Ved å justere på disse parametrene kan det resultere i at flere linjer blir detektert. De to parametrene er:

- Grdthres er terskling for intensiteten til en piksel som tilhører en linje
- Detsent er sensitiviteten til linjedeteksjonen

3 Andre om lokalisering av nummerskilt

Lokalisering av potensielle type-skilt i bilder er komplisert på grunn av store variasjoner i struktur, størrelse, form, farge,bakgrunn og selve plasseringen av type-skiltet i bildet. Vinkelen som bildene er tatt fra kan også skape problemer. Ulik lys fordeling som overeksponering eller kraftige skygger og refleksjoner kan skape problemer. I denne oppgaven skal det lokaliseres type-skilt i anleggsbilder, men de mest vanlige problemstillingene som har mange fellestrekk med problemstillingen i denne oppgaven er prosjekter basert på automatisk lokalisering og gjenkjennelse av nummerskilt til biler (ALPR - Automatic License Plate Recognition).

ALPR-prosjekter er ofte delt opp i 3 deler:

- Lokalisering av nummerskilt
- Segmentering av data i skiltet
- Gjenkjenning av segmentert data

I denne oppgaven er det den første delen som går ut på å lokalisere nummerskiltet, som har vært av interesse. Denne delen går ut på å finne potensielle områder som inneholder nummerskiltet. Denne lokaliseringsdelen beskrives som den mest utfordrende oppgaven [10] [11].

Det er vanlig at det brukes en kantdeteksjon som en del av lokaliserings algoritmen for å finne standardiserte nummerskilt [10] [11] [12] [13] [14].

Etter kantdeteksjon er det noen disse tidligere arbeidene som lokaliserer områder med mange vertikale kanter som kandidat områder [10] [11] [12] [15]. I prosjekt [14] ble det videre etter kantdeteksjon brukt forskjellige morfologiske metoder i kombinasjon med Hough Transform slik at de sitter igjen med nummerplate kandidater. I prosjekt [13] ble det videre etter kantdeteksjon brukt en kombinasjon av Hough Transform og kontur (imcontour.m i matlab) for å finne lukkede grenser til et objekt. Ellers er normalt å utføre en Hough Transform på kantdeteksjon-resultatet for å finne vertikale- og horisontale grensene til et nummerskilt [14].

4 Algoritmeutvikling

I dette kapittelet vil algoritmen som er blitt utarbeidet våren 2012 bli gjennomgått. Arbeidet ble utført Algoritmen bruker også elementer fra forprosjektet [2] som ble utført høsten 2011. Programvare som er blitt brukt i arbeidet med å utvikle algoritmen er MATLAB. En forenklet oversikt over algoritmen blir vist i figur 16. Ved å studere blokkdiagrammet kan det observeres at blokkene har forskjellige nummer oppe helt til venstre i hver enkelt blokk. Numrene i blokkene vil bli brukt i beskrivelsen av algoritmen for å gi en bedre oversikt og beskrivelse. I figur 17 a) er et eksempel bilde som skal brukes gjennom forklaringen av den automatiske algoritmen i dette kapittelet.

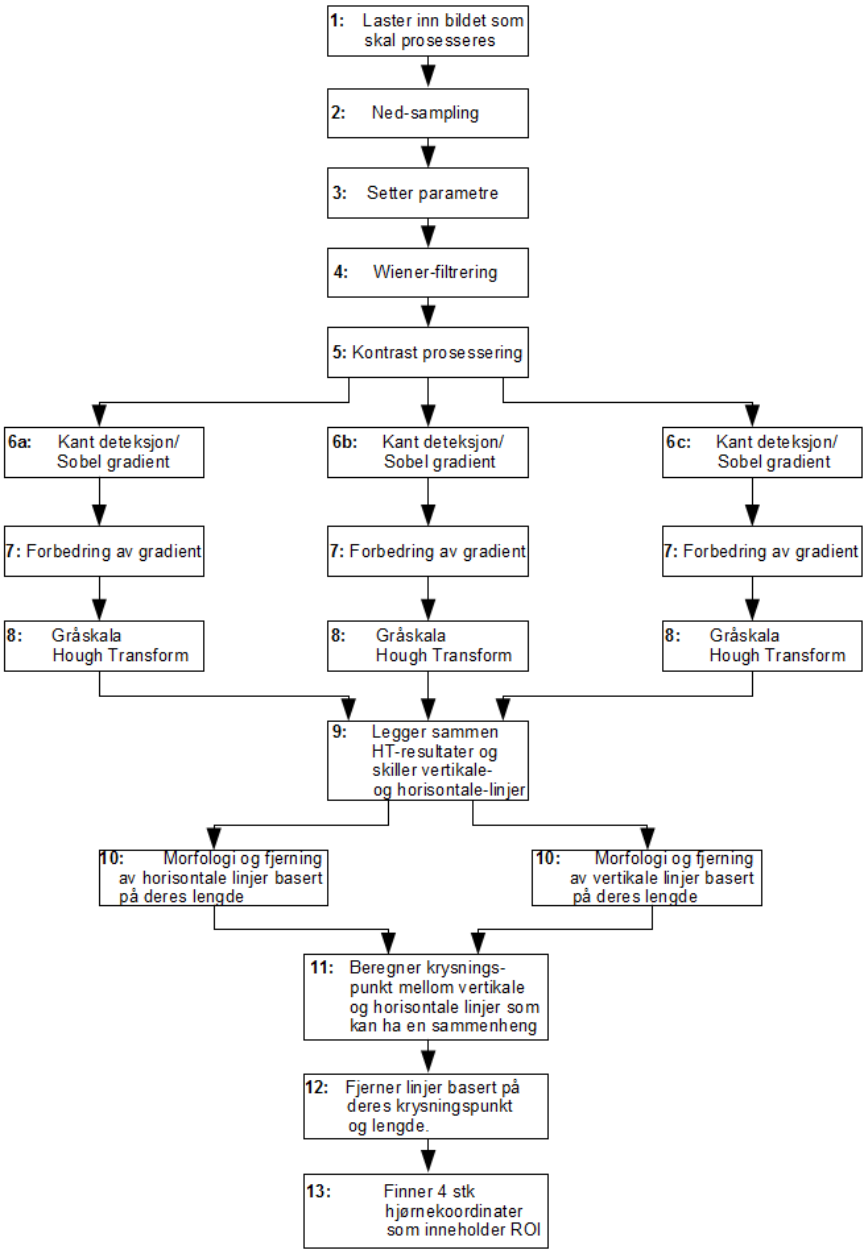


Figure 16: Enkelt blokkdiagram med piler som representerer flyten til algoritmen.

4.1 Forbehandling

Blokknummer: 2, 4 og 5.
Etter at bilde som skal prosesseres er lastet inn, blir det gjort en forbehandling. Forbehandlingen består av blokkene med nummer 2, 4 og 5 i figur 16:

- Nedsampling
- Wiener-filtrering
- Kontrast prosessering

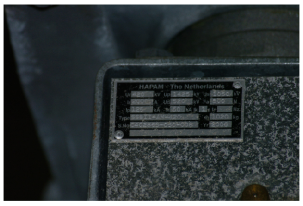
I blokk nummer 3 settes parametre basert på størrelsen til bildet som er blitt nedsamplet. Dette er ikke en del av forbehandlingen, men er bare satt inn i blokkdiagrammet i den rekkefølgen her for det er slik rekkefølgen er i m-filen for algoritmen.

Blokknummer 2, Nedsampling
Nedsampling ble presentert i forprosjektet [2], som en del av forbehandlingen i det videre arbeidet til masteroppgaven. Nedsamplingsfaktor ble ikke bestemt i forprosjektet. I arbeidet med masteroppgaven ble det derfor testet ut forskjellige nedsamplingsfaktorer på forskjellige bilder. En nedsamplingsfaktor som gav tilfredsstillende resultater med tanke på tid og bevaring av viktigste informasjon var ved bruk av nedsamplings faktor på fire.

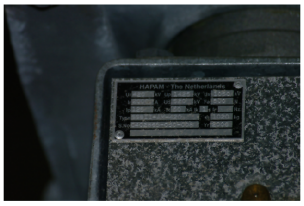
Blokknummer 4, Wiener-filtrering
Wiener-filtreret blir brukt til å gjøre en adaptiv glatting i forbehandlingen for å fjerne støy. Dette er for at videre kantdeteksjon og Hough Transform er begge sensitive for støy [14] [16]. De fleste bildene har et større område med lav varians, og et mindre område hvor skiltet befinner seg som vanligvis har en større varians enn bakgrunnen som passer bra med egenskapene til Wiener-filteret. Wiener-filtreringen vil føre til liten glatting i områder med høy varians og en større glatting i områder med lav varians, forklart mer detaljert i kapittel 2.2. I figur 17 b) vises resultatet etter en wiener filtrering. Resultatet vil oppfattes som om det har blitt veldig svakt glattet.

Blokknummer 5, Kontrast prosessering
Kontrast prosesseringen var først ikke en del av forbehandlingen. Grunnen til at den ble med etterhvert, var at til å begynne med bestod treningssettet av 6 forskjellige bilder. Men etter å ha økt treningssettet med 10 bilder til oppstod problemet at ikke alle kantene til skiltene blir detektert. Det var forskjellige grunner til dette for eksempel: skygger over kantene til skiltet, maling over kanter eller grums som forkludret kantene til skiltet, bakgrunn har veldig lik farge og intensitet som selve skiltet og at noen områder av bilder var overeksponering samtidig som andre områder var mørke.

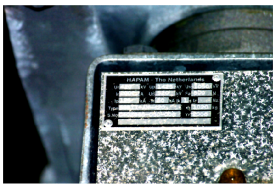
Etter å ha prøvd kontrast forbedrings funksjonene til Matlab histeq.m, adapthisteq.m og imadjust.m opp mot hverandre, virket det som imadjust.m funksjonen var den mest robuste i forhold til de bildene som skulle prosesseres i denne oppgaven. Denne funksjonen er egentlig beregnet for gråskala bilder, men er her blitt brukt på de tre gråskala matrisene som RGB-bildet består av. Dette kan medføre til at farger i bildet blir endret, men det har ikke hatt betydning for resultatet ettersom farge-informasjonen ikke har blitt brukt til annet enn kantdeteksjon i denne oppgaven. I figur 17 c), d) og e) er de tre forskjellige resultatene av kontrast prosesseringen, og blir produsert av funksjonen myKontrast.m. De tre bildene som ble produsert av myKontrast.m blir det videre gjort kantdeteksjon på. Derfor deles blokkdiagrammet i figur 16 etter blokk nummer 5, og går videre til blokk nummer 6a, 6b og 6c som mottar hvert sitt forskjellig kontrast prosesserte bilde som vises i figur 17 c), d) og e).



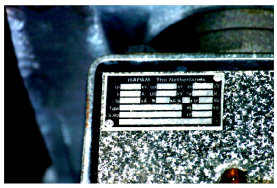
(a) Orginalbildet nedsamplet med 4



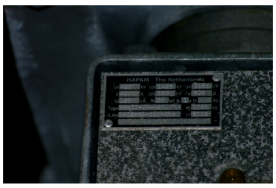
(b) Etter wiener-filtrering



(c) Konstrast prosessering 1, behandles videre av blokknummer 6a i figur 16.



(d) Konstrast prosessering 2, behandles videre av blokknummer 6b i figur 16



(e) Konstrast prosessering 3, behandles videre av blokknummer 6c i figur 16

Figure 17: Illustrasjon som viser de forskjellige resultatene i forbehandlingen

myKontrast.m
myKontrast.m bruker den innebygde matlab funksjonen imadjust.m forklart i kapittel 2.3. Imadjust.m bruker en funksjon som heter strechlim.m. Ved å gi strechlim.m et gråskala gråskala bilde vil funksjonen returnere en vektor som inneholder øvre- og nedre-grense som kan brukes til kontraststrekking i imadjust.m funksjonen. Stretchlim.m kan også brukes på RGB-bilder men da gir den ut tre stk vektorer, en vektor for hver fargematrise som også inneholder en øvre- og nedre-grense for kontraststrekking. Ved å ta bruke stretchlim.m på R-matrisen som har histogrammet vist i figur 18 vil den gi nedre grense for at 1% av pikslene skal skal bli satt til intensitetsverdi lik 0, og en øvre grense for at 1% av pikslene skal bli satt til intensitetsverdi lik 255. Grense-verdiene mottat av stretchlim er verdier på skalaen mellom 0 til 255. All kontraststrekking utført i myKontrast.m er linært strukket, som er illustrert i figur 4 b).

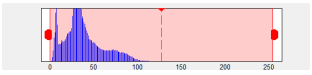
Grense nr 1 som myKontrast bruker til histogram strekking:
Finnes ved å ta verdiene gitt av stretchlim.m vist i figur 18 b) ,og legge til en verdi på 5% av 255 på den nedre grensen og trekke fra samme verdi på den øvre grensen gitt av stretchlim.m produseres grensene i figur 18 c). Grensene er markert med rød.

Grense nr 2 som myKontrast bruker til histogram strekking:
Finnes ved å ta verdiene til grense nr 1 vist i figur 18 c) og regne ut differensen mellom de to grenseverdiene, for så å legge en fjerdedel av differensen til nedre grense-verdi og trekke fra samme verdien fra øvre grense-verdi produseres grensene i figur 18 d). Grensene er markert med rød.

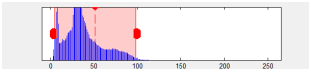
Grense nr 3 som myKontrast bruker til histogramstrekking:
Finnes ved å ta de 5% første verdiene til histogrammet og sette de til intensitetsverdi lik 0. Og ta 5% av verdiene i slutten av histogrammet til høyre og setter de til intensitetsverdi lik 255. Denne svake strekkingen gjøres fordi ofte er kontrasten i orginalbildet veldig bra i utgangspunktet.Grensene som grense nr 3 bruker til strekking er vist i figur 18 d). Grensene er markert med rød.

I figur 19 b), c) og d) vises resultat etter strekking med imadjust.m funksjonen med grensene 1,2 og 3 produsert av myKontrast.m

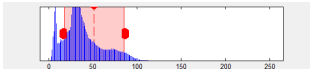
Det blir beregnet tre stk grenser på samme måte for hvert av de tre fargeplanene (RGB). På grunn av at dette kan det oppstå ulike grenser i kontraststrekking for de ulike fargeplanene, som kan føre til at fargene i bildet blir endret.



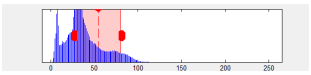
(a) Histogram over intensitetsverdiene til R-matrisen til orginalbildet.



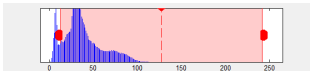
(b) Øvre og nedre grense gitt av stretchlim.m



(c) Grense nr. 1, som myKontrast.m beregner



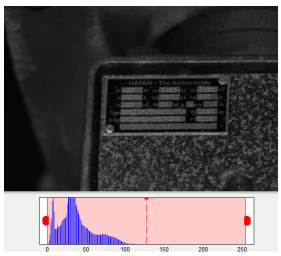
(d) Grense nr. 2, som myKontrast.m beregner



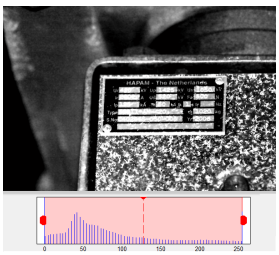
(e) Grense nr. 3, som myKontrast.m beregner

Figure 18: Illustrasjon over grense-verdiene som myKontrast.m beregner for kontrast strekking. Verdier utenfor grensene blir satt til 0 eller 255.

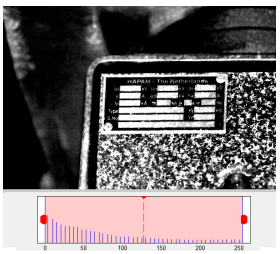
For enkelhets skyld er det i figur 19 a) vist R-matrisen til orginalbildet og tilhørende historam. Og i 19 b), c) og d) er resultatbildene og deres intenisitets histogram etter kontraststrekking. Figur 19 b) har brukt grense nr 1 til kontrast strekking. Figur 19 c) har brukt grense nr 2 til kontrast strekking. Figur 19 d) har brukt grense nr 3 til kontrast strekking. Den samme prosedyren gjort for R-matrisen blir gjort for alle de tre matrisene som fargebildet består av og satt sammen igjen til resultatet vist i figur 17 b).



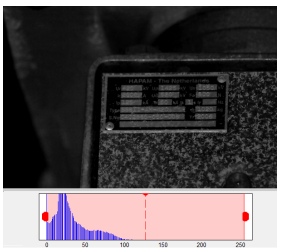
(a) R-matrisen til orginalbildet og intensitest histogrammet.



(b) Konstrast prosessert 1.



(c) Konstrast prosessert 2.



(d) Konstrast prosessert 3.

Figure 19: Illustrasjon som viser de tre forskjellige resultatene av kontrast prosessering gjort *kun* på R-matrisen til orginalbildet. Disse endringen blir utført av m-filen: myKontrast.m

Etter å ha strekket de tre fargeplanene med tre forskjellige grenser og satt de sammen til tre bilder igjen har vi de tre resultat bildene vist i figur 17 c), d) og e).

4.2 Parametersetting

Blokknummer: 3.
Parametersetting kommer rett etter nedsamplingen fordi parameterene settes ut i fra størrelsen på det nedsamplede bildet. Parametersetting er blokk nummer 3 i blokkdiagrammet i figur 16.

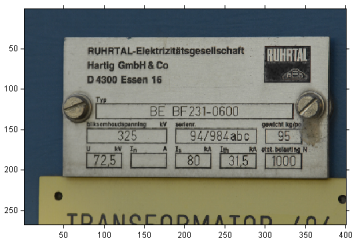
Parametersett1 og Parametersett2:
Hvert bilde består av m x n piksler, hvor m representerer den vertikale- og n den horisontale-størrelsen til bildet. Har vi m>n da har vi eksempel 1 i figur **??** a) og da velges parametersett1, ellers hvis vi har m<n da har vi eksempel 2 i figur **??** b) og da velges parametersett2.
Videre når vi har bestemt hvilket parametersett som vil bli brukt vil parametrene i dette parametersettet bli beregnet basert på antall piksler m og n som bildet består av. Dette kommer av at alle bildene som proseseres ikke har likt antall piksler. Formel for beregning av parametrene er blitt til under trening av treningssettet. Selve formelen for paramtrene kan sees ved å studere m-filen setParam.m i vedlegg A.2.

Det er totalt 21 parametre som settes og de settes av m-filen setParam.m. Hver enkelt parameter vil blir forklart nærmere i det blokknummeret hvor funksjonene som bruker de ulike parameterene blir tatt i bruk.

Funksjonene som bruker parametrene som blir satt av setParam.m listet her:

- **myClearHLines.m:** dH1, dH2, KH1, KH2, minH1 og minH2.
- **myClearVLines.m:** dV1, dV2, KV1, KV2, minV1 og minV2.
- **morph.m:** BH1, BH2, BV1 og BV2
- **findSingleCorner.m:** xDist og yDist
- **VHLinjerTot.m:** Vtol og Htol.
- **grenserMinMax.m:** Ratio.

(a) Eksempel 1 (m>n), bruker Parametersett1.



(b) Eksempel 2 (m<n), bruker Parametersett2.

Figure 20: Illustrasjon over hva som avgjør valget av parametsersett1 og parametersett2 i setParam.m funksjonen. Stående bilder har parametersett1 og liggende bilder har parametersett1.

4.3 Kantdeteksjon og forbedring av gradient

Blokknummer: 6a, 6b, 6c og 7.
På de tre kontrast forbedrede fargebildene produsert i forbehandlingen blir det gjort en kantdeteksjon og videre en kontrastprosessering på kantdeteksjonen. Disse operasjonene gjøres i blokk nummer 6a, 6b, 6c og 7 i blokkdiagrammet fra figur 16. Til dette er det blitt laget m-filen myVG.m som bruker funksjonene colorgrad.m fra kapittel 2.4.2 og imadjust.m fra kapittel 2.3. Funksjonen myVG.m utfører først kantdeteksjon for hvert av de tre bildene og bruker så en linær kontraststrekking hvor 1% av de minimum-verdiene i kantdeteksjonen settes til verdi '0' og 1% av maximum-verdiene til '255'. Etter kontraststrekkingen blir det videre utført en gammajustering på kantdetksjonen vektet mot høyre del av intensitetshistogrammet illustrert i grafen til figur 4 a). Dette er for at de kantene som ikke har fullt så stor intensitetsverdi som de med størst intensitetsverdi skal bli mer utlignet, slik at kantene lettere kan bli detektert av Hough Transformen i neste steg i blokkdiagrammet fra figur 16. De tre kantdeteksjonene kan sees hvis en ser bort fra røde og grønne linjene i figur 21 a), b) og c). De røde og grønne linjene er linjer detektert av Hough Transformen og blitt plottet på resultatet til kantdeteksjonen. Både de kantdeteksjonene som ikke har fått kontrast prosessering, og de som har blitt kontrastprosessert blir det videre gjort Hough Transformasjon på.

4.4 Gråskala Hough Transform

Blokknummer: 8 og 9.
Etter blokknummer 7 i blokkdiagrammet fra figur 16 har vi tre stk kantdeteksjoner. Disse tre kantdeteksjon bildene blir det gjort Hough Transformasjon på for å finne rette linjer i bildet og linke sammen segmenter 2.6. Dette gjøres fordi det antas at type-skiltet skal finnes er rektangulært og består av rette linjer. Det brukes en gråskala Hough Transform som ble funnet på MATLAB's internett side for deling av filer [9]. Valget av gråskala Hough Transformen er basert på tidligere arbeid i forprosjektet [2] hvor det ble testet Hough Transform for gråskala bilder opp mot Hough Transform for svart-hvitt bilder. Hvor det virket som at den gråskala Hough Transformen var mer pålitelig i sin deteksjon av rette linjer.

Det er ikke alltid at en Hough Transform klarer å detektere alle de rette linjene som er i et bilde, og ettersom at denne algoritmen krever at alle sidene til skiltet skal være detektert har det blitt laget m-filen VHLinjerTot.m. Denne filen ble laget under arbeidet av masteroppgaven, og er laget for at Hough Transformen lettere skal klare å detektere flere rette linjer. Funksjonen deler bildet opp i mindre bildeblokker og utfører Hough Transform på hver bildeblokk noe som øker deteksjonen av linjer. Det blir også gjort flere Hough Transformer på hver bildeblokk med forskjellige parametre på grunn av at ikke alle linjer blir detektert med de samme parametrene.

VHLinjerTot.m:
Denne filen mottar de tre kantdeksjonene og utfører Hough Transform på dem. Ved å dele opp bildet av kantdeteksjonen i mindre blokker og utfører Hough Transformasjon på hver blokk vil som sagt flere linjesegment blir detektert. Alle linjesegment som totalt blir funnet på de tre kantdeteksjon-bildene blir til slutt lagt sammen. Ved å studere figur 21 a), b) og c) ser vi linjesegmentene funnet for hver av de tre kantdeteksjonene. Dette tilsvarer de tre blokkene med blokknummer 8 i blokkdiagrammet fra figur 16. Grunnen til at linjene er merket med forskjellige farger er for at VHLinjerTot.m filen også utfører en sortering av horisontale og vertikale linjer etter at de er detektert.

VHLinjerTot.m funksjonen mottar også de to parametrene Vtol og Htol satt i funksjonen setParam.m fra kapittel 4.2. Vtol og Htol er lik for begge parametersettene.

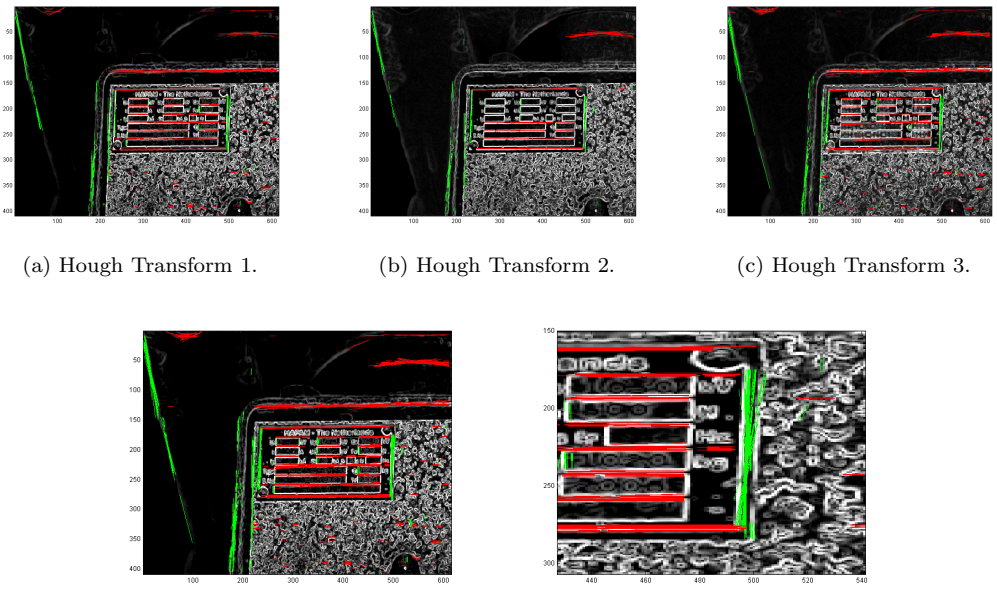
Vtol = Vertikal vinkel toleranse, og er fast satt til 22 grader. Dette vil si at de vertikale

linjesegmentene kan variere 22 grader fra den vertikale-aksen som er 90 grader. Altså må vertikale linjersegment som skal bli beholdt være innenfor området 70 til 110 grader.

Htol = Horisontale vinkel toleranse, og er fast satt til 18 grader. Dette vil si at de horisontale linjesegmentene kan variere 18 grader fra den horisontale-aksen som er 0 grader. Altså må horisontale linjersegment som skal bli beholdt være innenfor området fra -18 til 18 grader.

De linjesegmentene vist i figur 21 er linjesegmentene som er igjen etter at linjesegment med vinkel utenfor grensene satt av parametrene Vtol og Htol er fjernet.

For å detektere mest mulig linjer med den gråskala Hough Transformasjonen funksjonen Hough_Grd.m som har parameterene Grdtresh og Detsent 2.6, har det blitt funnet 7 faste parametersett. Med de 7 parametersettene har det blitt oppnådd gode resultater for deteksjon av linjer med Hough Transformasjonen. Videre legges linjesegmentene som er blitt detektert med de forskjellige parametersettene sammen. Disse 7 parametersettene blir ikke satt av setParam.m funksjonen, men er kommet frem til ved testing på treningsettet. Dermed når det blir gjort *en* Hough Transformasjon i denne oppgaven, så menes det at det er blitt gjort med de 7 forskjellige faste paramatrene. Det er laget m-filen HT2012.m som utfører Houg_Grd.m med de 7 paramtetrene og legger sammen resultatene fra de ulike Hough Transformene. Alle bilder/bildeblokker i denne oppgaven som det blir gjort en Hough Transformasjon på for å detektere linjer bruker filen HT2012.m. De ulike parametrene kan medføre at flere uønskede linjer blir detektert av Hough Transformasjonen, men det har totalt gitt flere ønskede linjer som er nødvendige videre for algoritmen.



(a) Hough Transform 1.

(b) Hough Transform 2.

(c) Hough Transform 3.

(d) Alle de tre Hough Transformene lagt sammen.

(e) Forstørret inn på noen av linjedeteksjonene gjort av Hough Transformen

Figure 21: Illustrasjon som viser de tre forskjellige Hough Transform resultatene, og nederst til venstre det totale resultatet ved å legge sammen de tre øverste resultatene.

4.5 Morfologi og fjerning av linjesegment basert på deres lengde

Blokknummer: 10.
Denne blokken mottar de røde og grønne linjesegmentene vist i figur 21 d) fra blokknummer 9. Ved å studere figur 21 e) som er forstørret inn på et område av linjesegmentene kan det observeres at det er flere linjer som representerer samme linjen som er et resultat etter å lagt sammen flere forskjellige Hough Transformasjoner. Ideelt er det ønskelig at alle linjesegmentene som representerer samme linje skal legges sammen til et linjesegment. Og ofte er det linjesegment som er deler av samme linje som ikke er i kontakt med hverandre.
Et forsøk på å legge sammen linjesegment som representerer samme linje gjøres ved å lage et binært bilde av linjesegmentene, hvor linjesegmentene blir representert med logisk 1-ere. Videre er det da lett å utføre morfologisk lukking på binærbildet. Etter lukkingen har vi ikke helt 'perfekte' linjer i binærbildet og det trengs derfor en funksjon for å hente ut linjesegment fra 1-er gruppene dannet i binærbildet basert på formen til 1-er gruppene som er dannet. Alt dette blir gjort i funksjonen morph.m.

morph.m:
I denne filen blir linjesegmentene gjort om til et binær bilde. Videre blir det utført en morfologisk lukking som er blitt beskrevet i kapittel 2.5.3. For de horisontale linjesegmentene blir det brukt et horisontalt strukturelement kalt BH1 og BH2 i filen setParam.m. For de vertikale linjesegmentene blir det brukt et vertikalt strukturelement kalt BV1 og BV2 i filen setParam.m. Lengdene på strukturelementene blir satt i setParam.m og er basert på bildestørrelsen på bildet som prosesseres, og om bildet er stående eller liggende. Strukturelementet BH1 og BV1 blir brukt i først, mens strukturelementene BH2 og BV2 blir brukt senere i blokknummer 10. De fire strukturelementene som blir brukt er vist i figur 25. Denne morfologiske lukkingen resulterer i at linjesegment som er nær hverandre blir koblet sammen alt ettersom hvor stort strukturelementet er. Denne koblingen av ulike elemnter vi får ved morfologisk lukking er en ønsket effekt. Ved å studere figur 22 a) og b) kan det sees at noen linjesegmenter som er nær hverandre har blitt lagt sammen.
Dette er den effekten som ble illustrert i eksempelet om morfologisk lukking i kapittel 2.5.3. For denne oppgaven brukes kun binær morfologi og for å kunne utføre morfologien må vi ha linjene omgjort til et binær bilde. Dermed har det blitt laget en funksjon som heter lines2bw.m som lager et binær bilde hvor linjesegmentene blir representert som logisk 1, dette blir illustrert i figur 22 a). Funksjonene er basert på at de horisontale og vertikale linjesegmentene separert gjenneom denne prosessen. Etter morfologisk lukkingen har vi resultatet for de horisontale linjesegmentene vist i figur 22 b). Resultatet fra lukkingen vil vi ha gjort om til linjesegmenter igjen, og det utføres av funksjonen bw2lines.m som kalles i morph.m. Ved å studere figur 22 c) ser vi det endelige resultatet etter funksjonen morph.m av de vertikal og horisontale linjesegmentene.

(a) De horisontale linjesegmentene gjort om til et binær bilde.

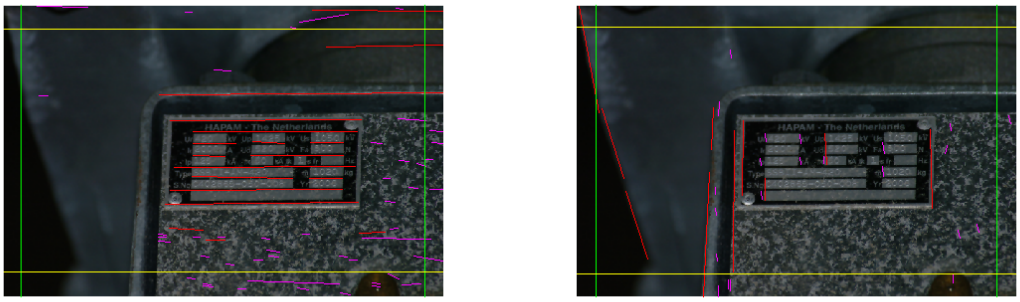
(b) De horisontale linjesegmentene etter morfologisk lukking med parameteren BH1.

(c) Resultat etter morph.m med parametrene BH1 og BV1.

Figure 22: Illustrasjon av noen resultater fra de forskjellige stegene i funksjonen morph.m.

myClearHlines.m og myClearVlines:
Videre er det ønskelig å fjerne uinteressante linjsegment detektert av Hough Transformen. Ved å studere resultatet etter morph.m 22 c) kan det observeres at de minste linjesegmentene ofte er upålitelige og har somregel ikke noen sammenheng med andre linjesegment. For å kunne utføre en kraftigere morfologisk lukking for å koble linjesegmenter sammen må de minste og ofte upålitelige linjesegmentene fjernes.
I begynnelsen av utviklingen av algoritmen ble det brukt MATLAB's innebygde funksjon imclearborder.m som fjerner linjesegment i kontakt med rammen til bildet som prosesseres. Etterhvert ble det observert at linjesegment som var i kontakt med rammen kunne være deler av kanter til skiltet som skulle detekteres. For skilt som skal lokaliseres som befinner seg i nærheten av midten, måtte potensielle linjesegment til skiltet som er i kontakt med rammen ha en lengde på nesten halvparten av bildestørrelsen. Det ble også observert at noen skilt er plassert nær rammen til bildet, men de gangene selve skiltet er nær kanten så virker det som om det er på grunn av at skiltet er så stor i forhold selve til bildet. Dette kan observeres ved å se på gjennom alle bildene som blir testet i denne oppgaven. Videre ble det laget to m-filer som heter myClearHlines.m og myClearVlines.m som er basert på de observasjonene beskrevet.

myClearHlines.m og myClearVlines.m brukes to ganger i algoritmen. Første gangen er her i blokknummer 10 i figur 16, etter at den første morfologiske lukkingen.
Formålet med funksjonene er som nevnt at de skal prøve å fjerne forskjellige uinterssante linjesegment. For lettere å kunne forklare funksjonene vil figur 23 a) bli brukt. Ved å studere figur 23 a) kan det observeres to gule horisontale linjer og to grønne vertikale linjer. Dette er grenser satt av parametrene 'dH1' og 'dH2' satt i funksjonen setParam.m hvor de grønne vertikale grensene er 'dH1' og de gule horisontale grensene er 'dH2'. Alle horisontale linjesegment som har pikselverdier innenfor disse grensene og som samtidig har en kortere euklidisk lengde enn 'KH1' parameteren blir fjernet. Ellers for *hele* bildet brukes parameteren 'minH1', som er en euklidisk lengde som horisontale linjesegmentene må være lengre enn for ikke å bli fjernet. De linjesegmentene med rød farge i figur 23 a) er linjesegmentene som er igjen etter funksjonen myClearHlines.m. De linjsegmentene med fargen magenta er linjesegment som blir fjernet av myClearHlines.m. Det samme blir gjort i myClearVlines.m bare at den bruker parametrene 'dV1', 'dV2', 'KV1' og 'minV1', og resultatet ved bruk av den på de vertikale linjesegmentene vises i figur 23 b).



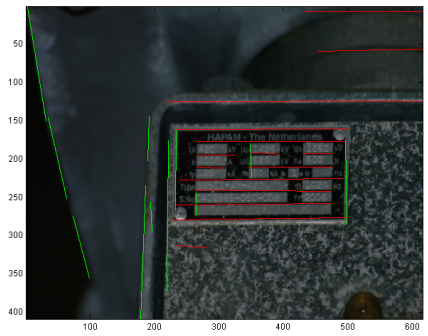
(a) myClearHlines.m med grenser plottet

(b) myClearVlines.m med grenser plottet

Figure 23: Uinteressante linjesegment her markert med fargen magenta blir fjernet.

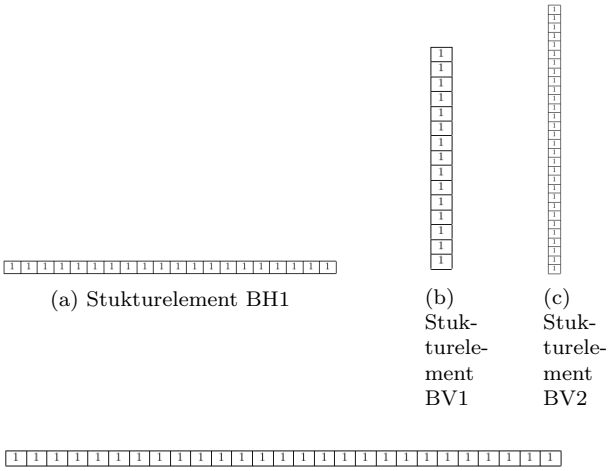
Etter at myClearHlines.m og myClearVlines.m har blitt brukt er de linjesegmentene som er igjen mer pålitelige og vi kan dermed gjøre en ny morphologisk lukking på linjesegmentene med funksjonen morph.m, men med de kraftigere parametre 'BH2' og 'BV2'. Ved å studere figur 24

ser vi resultatet etter morfologisk lukking med parametrene BH2 og BV2. Dette er det endelige resultatet produsert i blokknummer 10 som sendes videre til blokknummer 11 i figur 16.



(a) Resultat etter morph.m med parametrene BH2 og BV2

Figure 24: Resultat som sendes videre til blokknummer 11, etter prosessering av blokknummer 10.



(a) Stukturelement BH1

(b) Stukturelement BV1

(c) Stukturelement BV2

(d) Stukturelement BH2

Figure 25: Strukturelementene som brukes av morph.m, og som er blitt satt av setParam.m basert på bildestørrelsen som prosessers.

4.6 Beregning av krysningspunkt

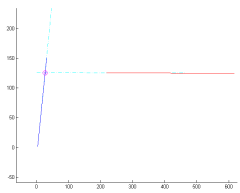
Blokknummer: 11.
I blokknummer 11 i blokkdiagrammet fra figur 16 blir det først beregnet krysningspunkt mellom vertikale- og horisontale linjesegment som kan ha en sammenheng. Det blir ikke beregnet krysningspunkt mellom forskjellige vertikale linjesegment selv om de også kan krysse hverandre. Alt som blir utført i blokknummer 11 er det funksjonen linjeGrense.m som utfører. Denne funksjonen starter med å beregner alle mulige krysningspunkt mellom vertikale- og horisontale linjesegment.

Videre fjernes krysningspunkt som er blitt beregnet ut fra linjesegmenter som ikke kan ha sammenheng.

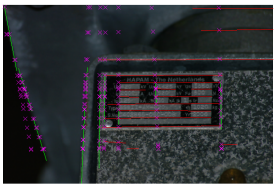
linjeGrense.m
funksjoner som brues av linjeGrense.m:

- linecrossings.m, finner krysningspunktet hvor to vektorer krysser hverandre. Linjesegmentene trenger ikke å krysse hverandre men de blir forlenget slik at de treffer hverandre.
- linesegs2lines.m, lager et linje objekt av linjesegmenter som er på formen $[x_1, y_1, x_2, y_2]$, hvor x_1 og y_1 er koordinatene til start punktet til et linjesegment, og x_2 og y_2 er koordinatene til sluttpunktet.
- newlines.m, lager et linjeobjekt ut i fra en vektor og et punkt som vektoren går gjennom.
- vectangle.m, beregner vinkelen mellom to vektorer.
- vectlength.m, beregner lengden til en vektor.
- linjeGrenseV.m, beregner en minimum- og maximum-grense for hvert vertikalt linjesegment som hvert krysningspunkt produsert av dette linjesegmentet må være innenfor. De krysningspunktene utenfor grensene blir fjernet.
- linjeGrenseH.m, beregner en minimum- og maximum-grense for hvert horisontalt linjesegment som krysningspunkt produsert av dette linjesegmenet må være innenfor. De krysningspunktene utenfor grensene blir fjernet.
- DistBetween2Segment.m, beregner minste mulige distanse mellom to linjesegment.

Ved å sturere figur 26 a) ser vi en enkel illustrasjon over resultatet av beregning av kryssningspunktet mellom to linjesegment. Den blå linjen er det vertikale linjesegmentet og den røde linjen er det horisontale linjesegmentet. De cyan-fargede stiplede linjene er forlengelsen av linjene, og det resulterende krysningspunktet er markert med fargen magenta.
Videre blir resultatet av å beregne krysningspunktene mellom alle de vertikale- og horisontale-linjesegmentene får vi resultatet vist i figur 26 b).



(a) Eksempel på beregning av krysningspunkt mellom et vertikalt- og et horisontalt-linjesegment.

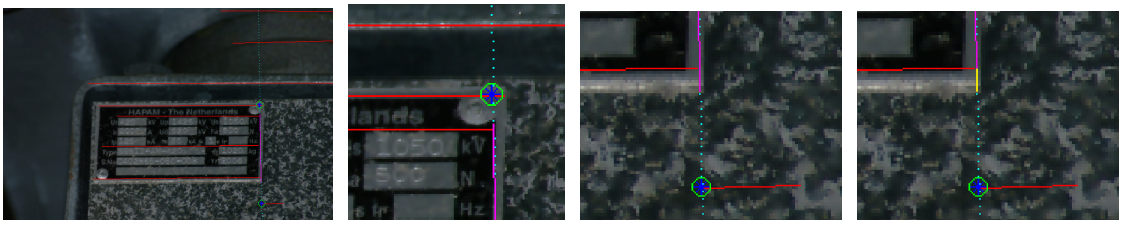


(b) Alle krysningspunkt beregnet mellom alle vertikale- og horisontale-linjesegment. Totalt 170 krysningspunkt.

Figure 26: Illustrasjon over krysningspunktet som blir funnet mellom vertikale- og et horisontale-linjesegment.

Videre i linjeGrense.m fjernes krysningspunkt som er usannsynlinge. De krysningspunktene som er usannsynlige er et resultat av det bare ble sett på to linjesegment seperat om gangen ved

beregningene. Denne fjerningen av usannsynlige krysningspunkt blir forklart sammen med figur 27, hvor det skal finnes grenser til de mulige krysningspunktene som det vertikale linjesegmentet med fargen magenta kan produsere med horisontale linjesegment. Den stiplede cyan fargede linjen viser forlengelsen av det vertikale linjesegmentet som det skal finnes grensene til. De røde horisontale linjesegmentene som er vist er *kun* de linjesegmentene som har en distanse mindre enn 5 piksler fra det vertikale linjesegmentet. Distansen på 5 piksler er en fast parameter satt i funksjonen. Det er bare disse horisontale linjesegmentene som blir med i beregningen av minimum- og maksimum grense for det vertikale linjesegmentet med fargen magenta. Grensene som har blitt beregnet av funksjonen i dette eksempelet er vist i med grønne sirkler i figur 27 a). Det er også en fast parameter satt i funksjonen kalt 'overlapp'. Den er et tall på hvor mangen piksler det magenta vertikale linjesegmentet 'er fysiskt videre forbi' et horisontalt rødt linjesegment før forlengelsen overtar og treffer et rødt horisontalt linjesegment. Er 'overlappen' mindre enn 4 piksler trekkes grensen tilbake til siste passerte horisontal rød linje. Er 'overlapp' over 4 piskler som den er her i begge endene av det vertikale linjesegmentet blir grensene satt av første røde linjesegment som den stiplede forlengelsen møter. Dette er for at når vi fortsetter med den stiplede forlengelsen øker usikkerheten, da den ikke er et fysiskt linjesegment. Den neste horisontale røde linjen som som den stiplede forlengning møter vil 'sperre' og sette en grense. Dette vises i figur 27 c) og d), og ved å sammenligne figur c) og d) er 'overlapp-målet' til det magenta linjesegmentet markert med gul i figur 27 d). Dette prinsippet blir brukt for alle vertikale og horisontale linjesegment. Og resultatet av dette blir betraktelig mindre antall krysningspunkt som blir vist i figur 28. Dette resultatet blir sendt videre til blokknummer 12 i vist figur 16.



(a) Alle de røde linjesegmentene har avstand mindre enn fire piksler til det vertikale linjesegmentet.

(b) Maksimum-grensen forstørret.

(c) Minimum-grensen forstørret.

(d) 'Overlapp' markert med fargen gul.

Figure 27: Minimum- og maksimum grensene funnet til et av de vertikale linjesegmentet.

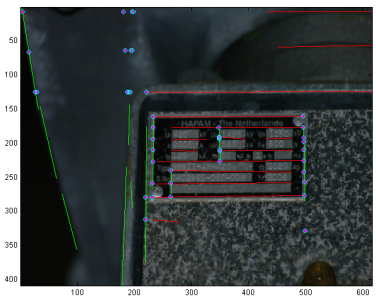
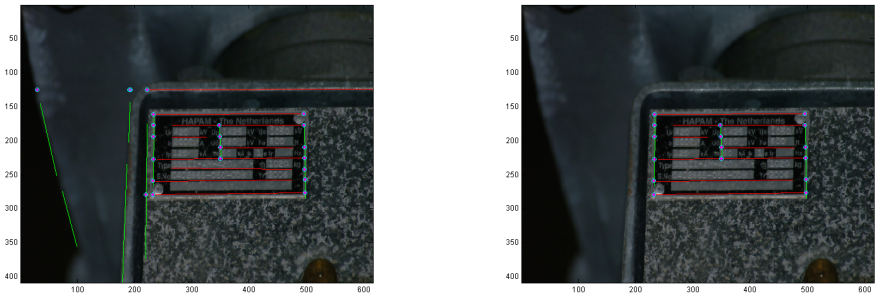


Figure 28: Illustrasjon over de krysningspunktet som er igjen etter reduksjon. Totalt 43 krysningspunkt.

4.7 Fjerning av linjesegment basert på deres krysningspunkt og lengde

Blokknummer: 12.
Her startes det med først å fjerne linjesegment med funksjonene myClearHlines.m og myClearVlines.m igjen, men med litt kraftigere parametre satt av setParam.m funksjonen. Etter eventuell fjerning av linjesegment fjernes de krysningspunktene som er beregnet av eventuelle linjesegment som fjernes. Grunnen til at disse kraftigere paramtrene ikke ble brukt første gang av funksjonene myClearHlines.m og myClearVlines.m var for at de linjesegmentene med en lengde opp til 25 piksler detektert av Hough Transformen er veldig upålitelige. Derfor måtte de fjernes før det ble utført et kraftige morfologisk lukking. Og dessuten kan linjesegmentene være til 'hjelp' for funksjonene som blir utført i blokknummer 10 i blokkdiagrammet fra figur 16, hvor utregning av minimum- og maksimum-grensene til hvert linjesegment beregnes som er en veldig viktig del av algoritmen. Derfor blir det ikke utført en kraftigere fjerning av linjer før her i blokknummer 12. Alt dette blir gjort av filen cleanCornerLines.m, som bruker funksjonene myClearHlines.m og myClearVlines.m. Resultatet etter cleanCornerLines.m vises i figur 29 a).



(a) Resultat etter cleanCornerLines.m

(b) Resultat etter findSingleCorner.m

Figure 29: Illustrasjon av resultater produsert underveis og det endelige resultatet produsert i blokknummer 12 fra blokkdiagrammet.

Videre etterfølges cleanCornerLines.m med funksjonen findSingleCorner.m som først ble utviklet for å fjerne linjesegment som kun hadde blitt med i beregningen av et krysningspunkt. Men etterhvert ble funksjonen utviklet til å fjerne linjesegment som hadde flere krysningspunkt, men kun på en side av linjesegmentet. I og med at vi skal finne et rektanulært skilt kan ikke alle krysningspunktene befinne seg på bare en side av linjesegmentet. Etter utviklingen av funksjonen kan funksjonsnavnet være litt missvisende.
Et annet krav var en minimumavstand mellom krysningspunktene produsert på linjesegmentet inkludert 'forlengelsen' som ble brukt ved utregning av krysningspunkt. Derfor er det satt noen faste parametre på denne minimumavstanden i setParam.m som kalles 'xdist' og 'ydist'. Hvor 'xdist' er en minimum avstand som kreves mellom krysningspunktene på en horisontal linje. Og 'ydist' er en minimum avstand som kreves mellom krysningspunktene på en vertikal linje. Disse parametrene blir dermed en minimum lengde som det tenkes at et type-skilt kan ha i vertikal og horisontal lengde. Ved å studere figur 29 b) kan det observeres resultatet etter å ha kjørt funksjonen findSingleCorner.m. Videre sendes resultatet fra findSingleCorner.m til funksjonen grenserMinMax.m. Funksjonen har forløpig en fast parameter 'Ratio' som er satt til 0.8. Funksjonen fjerner vertikale linjesegment som befinner seg mer enn parameteren 'Ratio' utenfor øverste og nederste horisontale linjesegment. Det vil si at linjesegmentets piksler må være 80% utenfor grensene for å bli fjernet. Den fjerner også horisontale linjesegment som befinner seg mer enn parameteren 'Ratio' utenfor det vertikale linjesegmentet til høyre og venstre. I dette eksempelet vil dermed ikke denne funksjonen fjerne noen av de resterende linjesegmentene som var igjen etter funksjo-

nen findSingleCorner.m. Dermed sendes resultatet vist i figur 29 b) videre til blokknummer 13 i blokkdiagrammet fra figur 16, hvor det skal bli funnet fire potensielle hjørnekordinater til type-skiltet.

4.8 Finne fire potensielle hjørnekoordinater som inneholder type-skiltet.

Blokknummer: 13.

Her skal det finnes fire potensielle hjørnekordinater som inneholder tupe-skiltet. Algoritmen skal resultere i fire potensielle hjørnekoordinater, og den gir også ut den stiplede cyan linjen som er plottet mellom de fire hjørenkoordinatene i figur 30. Resultatet som er vist i figur 30 er beregnet av funksjonen finnROI.m.

finnROI.m:
Det første denne funksjonen gjør er å finne det horisontale linjesegmentet som befinner seg øverste, og så det horisontale linjesegmentet som befinner seg nederst i bildet. Og så finnes de vertikale linjesegmentene som befinner seg lengst til høyre og venstre. Dette gjøres på de linjesegmentene som var igjen etter blokknummer 12 i blokkdiagrammet fra figur 16, som er vist i figur 29 b). Ut i fra de ytterste linjesegmentene funnet blir det søkt om det befinner seg et lengre linjesegmenet ved å gå 20 piksler innover mot midten i det området som de ytterste linjesegmentene funnet danner. Dette gjelder for alle bilder som prosesseres. Hvis det finnes lengre linjesegment vil disse velges som den ytterste grensen til skiltet, fordi de lengste linjene har virket som de mest pålitelige linjene. Vi har nå de fire ytterste linjesegmentene og beregner de fire potensielle hjørnekoordinatene til type-skiltet. De koordinatene som er blitt funnet blir sjekket at de er innenfor bildestørrelsen. Er de krysningspunktene innenfor bilde-rammen blir dette de fire resulterende hjørnekoordinatene, som det ble i dette eksempelet.

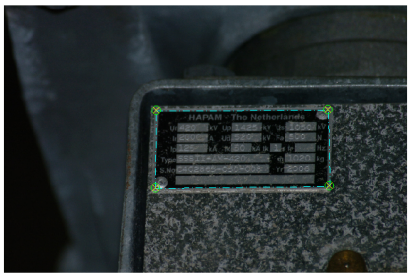


Figure 30: De fire potensielle hjørnekoordinatene produsert automatisk av algoritmen

Hvis det viser seg at noen av hjørnekoordinatene funnet har koordinater som befinner seg utenfor rammen til bildet, vil det bli prøvd å finne et linjesegment som erstatter det linjesegmentet som fører til at et koordinatet er utenfor rammen. Dette gjøres ved å søke etter et nytt linjesegment lenger innover mot midten i området som var omringet av de ytterste linjesegmentene. [2]

5 Test av algoritmen

For teste algoritmen har det blitt laget tre stk testsett med 30 forskjellige bilder i hvert sett som er plukket ut fra bilder tildelt av Verico AS. Dette gir totalt 90 ulike bilder som algoritmen testes på. For å kjøre algoritmen automatisk på alle tre settene med bilder er det laget funksjonen: testsett.m. Denne funksjonen lagrer de fire koordinatene funnet av algoritmen i en tabell. Til slutt produserer testsett.m figurer av alle bildene med automatisk lokaliserte hjørnekoordinater og manuelt avleste hjørnekoordinater. Det blir også plottet stiplede linjer mellom hjørnekoordinatene som gjør det lettere å sammenligne resultatet med fasiten.

De hjørnekoordinatene som er produsert automatisk av algoritmen blir plottet med et blått kryss og en liten gul sirkel for å lettere kunne få øye på hjørnekoordinatene i forskjellige bilder. Det blir også plottet en stiplet linje med fargen cyan mellom de automatisk produserte hjørnekoordinatene.

De hjørnekoordinatene som er manuelt avlest som er fasiten blir plottet med et rødt kryss og en liten cyan farget sirkel. Det blir også plottet en stiplet linje med fargen magenta mellom de manuelt avleste hjørnekoordinatene.

I figur 31 er det vist et eksempel fra et av testsettene med resultat fra algoritmen plottet sammen med fasit.

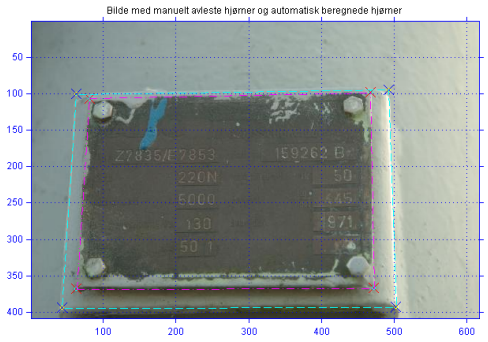


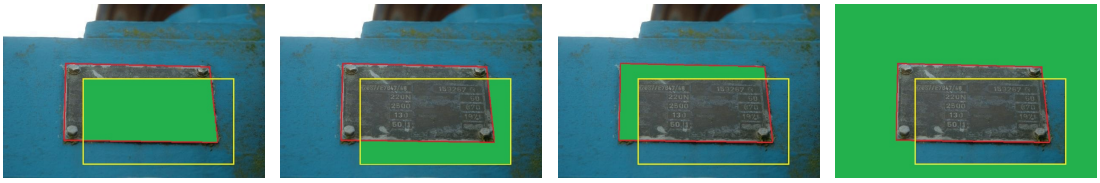
Figure 31: Området automatisk detektert av algoritmen er markert med en stiplet linje med fargen cyan. Skiltet funnet manuelt er merket med stiplet linje med fargen magenta.

Det vil også bli beregnet en rekke verdier for testen. Disse verdiene blir forklart i kapittel 6 sammen med vurderingen av resultatene.

6 Vurdering av resultater.

For å vurdere resultatene til algoritmen blir det beregnet en del verdier til hvert bilde som er blitt utført klassifisering på. Hver piksel i bildet som prossesers kan ha fire utfall ved en klassifisering som blir illustrert med grønn farge i de fire bildene vist i figur 32. Det som i virkeligheten er rammen til skiltet er markert med den røde rammen. Den gule rammen er et eksempel laget for å illustrere en klassifisering gjort av algoritmen, og for å visualisere de fire verdiene TN, FN, FP og TP. Alle verdiene som blir beregnet i dette delkapittelet blir beregnet blir beregnet av funksjonen finnSTAT.m. [17] [18][19] [20]

- TP - Sanne positive. Dette er antall piksler som er klassifisert som skilt av algoritmen, og som i virkeligheten er det.
- TN - Sanne negative. Dette er antall piksler som er klassifisert som bakgrunn av algoritmen, og som i virkeligheten er bakgrunn.
- FN - Falske negative. Dette er antall piksler som er klassifisert som bakgrunn men som i virkeligheten er en del av skiltet.
- FP - Falske positive. Dette er antall piksler som er klassifisert som skilt, men som i virkeligheten er bakgrunn.



(a) TP, sanne positive (b) FP, falske positive (c) FN, falske negative (d) TN, sanne negative

Figure 32: De fire utfallene en klassifisering kan ha markert med grønn.

Videre ut i fra verdiene TN, FN, FP og TP beregnes seks verdier som kan brukes til å måle hvor god klassifiseringene som er blitt gjort er. Disse verdiene er:

- PPV - Positiv prediktiv verdi, er et tall på antall sanne positive det er i forhold til antallet sanne positve og falske positive. Derfor er det ønskelig at dette forholdet er så nær 1 som mulig. $PPV = \frac{TP}{TP+FP}$
- NPV - Negativ prediktiv verdi, er et tall på antall sanne negative i forhold til antallet sanne negative og falske negative. Derfor er det ønskelig at dette forholdet er så nær 1 som mulig.$NPV = \frac{TN}{FN+TN}$
- FDR - Falsk oppdagelse rate, er et tall på antall falske negative det er i forhold til antallet sanne positive og falske positive. Derfor er det ønskelig at dette forholdet er så nær 0 som mulig.$FDR = \frac{FP}{FP+TP}$
- Sensitivitet - Er et tall på hvor mangen piksler som er blitt korrekt klassifisert til skilt som i virkeligheten er skilt. Derfor er det ønskelig at dette forholdet er så nær 1 som mulig. Sensitivitet= $\frac{TP}{TP+FN}$
- Spesifisitet - Er et tall på hvor mangen piksler som er blitt korrekt klassifisert til bakgrunn i forhold til totalt antall piksler som i virkeligheten er bakgrunn. Derfor er det ønskelig at dette forholdet er så nær 1 som mulig. Spesifisitet= $\frac{TN}{FP+TN}$
- ACC - er et tall på hvor mangen av alle pikslene som er blitt korrekt klassifisert i forhold til totalt antall piksler. ACC må ikke sees på alene, men i sammenhenge med spesifisitet- og sensitivitet-verdiene, fordi ACC verdien kan være god uten at noe av skiltet er detektert i det hele tatt. Det er ønskelig at ACC verdien er så nær 1 som mulig. $ACC = \frac{(TN+TP)}{(TP+FN+TN+FP)}$.

Det vil nå bli presentert resultater for de tre testsettene basert på verdiene som er blitt forklart. Resultatene er fordelt over to tabeller, og hvert testsett har to tilhørende tabeller. Tabellene med resultatene blir etterfulgt av bildene for det gjeldende testsettet. I disse bildene er resultatet og fasit plottet på bildet slik det ble forklart i kapittel 5. Dette er gjort for at det skal bli lettere for andre å få oversikt over resultatene og sammenligne. Til slutt vil det bli vist tabell 7 med gjennomsnittsverdiene beregnet for alle de tre testene.

6.1 Testsett 1:

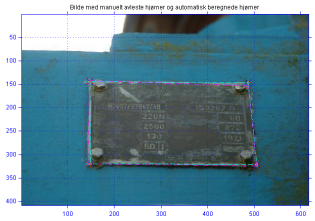
For testsett 1 kan det i tabell 2 observeres at resultater på både spesifisitet og sensitivitet for de fleste bildene har en verdi på over 0.85. Dette betyr at det er god deteksjon på både skilt og bakgrunn. Men for bilde nummer 2, 6, 13, 18, 19, 20 fra figur 33 og figur 34 er det noe mindre gode verdier. Det er forskjellige grunner til dette. Foreksempel for bilde nr 2 og 6 er det en firkant rundt selve skiltet og algoritmen klarer ikke skille mellom ytterste firkant og selve skiltet og detekterer dermed den største firkanten dannet. De bildene med dårligst verdier på sensitivitet og spesifisitet er bilde nr 18, 19 og 20 i figur 34 f) g) og h), som alle er veldig mørke i store deler av bildet, og i tillegg har de en del rette linjer som gjør det vanskelig for denne algoritmen å finne de rette linjene. Skiltene er også ganske små i forholdt til selve bildet. For de fleste bildene med gode verdier på både sensitivitet og spesifisitet kan det sies at det ikke er altformye rette linjer bakgrunnen som ikke er en del av skiltet. Videre fra 2 kan se at den minste verdien til sensitiviteten er 0.86, noe som betyr at det er de fleste pikslene som er skilt blir klassifisert som skilt. Dette lover godt for algoritmen og betyr at den har et godt potensiale for videre utvikling. Hadde den heller hatt en tendens til å klassifisere skiltet til bakgrunn ville det vært vanskeligere. Fra tabell 2 kommer det også frem at gjennomsnittlig spesifisitet har verdi på 0.9598 og gjennomsnittlig sensitivitet er 0.9134 og ønskelig er nærmest mulig 1, som er bra for disse bildene. NPV har den beste gjennomsnitsverdien på 0.9796 som vil si nesten 98% av verdier som er klassifisert til bakgrunn i virkeligheten er det. Verdien til PPV er 0.8496 som vil si at 84% av de verdiene som blir klassifiserte til å være type-skiltet faktisk er det.

Bildenr:	**TN**	**TP**	**FN**	**FP**
1	187937	61236	1190	965
2	118204	106842	0	26282
3	165169	65994	8793	44
4	196036	49047	208	6037
5	132854	42455	4611	0
6	174037	12319	96	64876
7	135490	113039	2238	561
8	166574	83759	805	190
9	131632	43587	4701	0
10	118048	127520	4545	1215
11	150322	98222	149	2635
12	186301	50017	3601	81
13	101368	284599	8279	35090
14	225245	178978	23909	1204
15	219592	29414	2322	0
16	213167	23003	3462	368
17	149407	274299	1861	3769
18	204370	14141	0	210825
19	190937	6610	0	231789
20	136867	41994	6369	66098
21	242489	182268	322	4257
22	243375	6780	368	805
23	110854	119622	298	9226
24	81099	146701	7139	5061
25	158658	65084	2004	25582
26	69997	164580	0	5423
27	109236	128613	1653	498
28	207787	29973	405	1835
29	314214	105368	8749	1005
30	106138	130459	982	2421

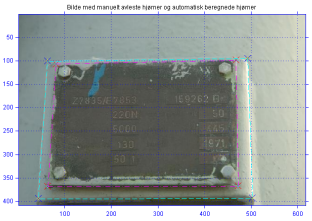
Table 1: Data 1 for Testsett 1

Bildenr:	**PPV**	**NPV**	**FDR**	**Sensitivitet**	**Spesifisitet**	**ACC**
1	0.98449	0.99371	0.015514	0.98094	0.99489	0.99143
2	0.80258	1	0.19742	1	0.8181	0.89543
3	0.99933	0.94945	0.00066628	0.88243	0.99973	0.96318
4	0.8904	0.99894	0.1096	0.99578	0.97012	0.97515
5	1	0.96646	0	0.90203	1	0.97437
6	0.15958	0.99945	0.84042	0.99227	0.72845	0.74149
7	0.99506	0.98375	0.0049384	0.98059	0.99588	0.98886
8	0.99774	0.99519	0.0022633	0.99048	0.99886	0.99604
9	1	0.96552	0	0.90265	1	0.97387
10	0.99056	0.96293	0.009438	0.96559	0.98981	0.97708
11	0.97387	0.99901	0.026126	0.99849	0.98277	0.98892
12	0.99838	0.98104	0.0016168	0.93284	0.99957	0.98466
13	0.89024	0.92449	0.10976	0.97173	0.74285	0.89899
14	0.99332	0.90404	0.0066821	0.88216	0.99468	0.94151
15	1	0.98954	0	0.92683	1	0.99076
16	0.98425	0.98402	0.015746	0.86919	0.99828	0.98404
17	0.98645	0.9877	0.013554	0.99326	0.97539	0.98689
18	0.062858	1	0.93714	1	0.49223	0.50895
19	0.027727	1	0.97227	1	0.45168	0.46012
20	0.3885	0.95553	0.6115	0.86831	0.67434	0.71166
21	0.97718	0.99867	0.022823	0.99824	0.98275	0.98933
22	0.89387	0.99849	0.10613	0.94852	0.9967	0.99533
23	0.9284	0.99732	0.071604	0.99752	0.92317	0.96032
24	0.96665	0.91909	0.033348	0.95359	0.94126	0.94917
25	0.71784	0.98753	0.28216	0.97013	0.86115	0.89024
26	0.9681	1	0.031899	1	0.9281	0.9774
27	0.99614	0.98509	0.0038571	0.98731	0.99546	0.99104
28	0.94231	0.99805	0.05769	0.98667	0.99125	0.99067
29	0.99055	0.97291	0.0094479	0.92333	0.99681	0.97728
30	0.98178	0.99083	0.018219	0.99253	0.9777	0.98582
Gj.snitt:	0.8496	0.9796	0.1504	0.9598	0.9134	0.9213

Table 2: Data 2 for Testsett 1

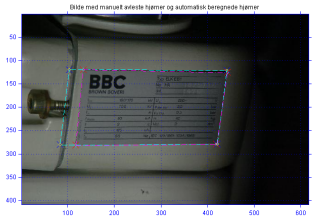


(a) Bilde nr 1

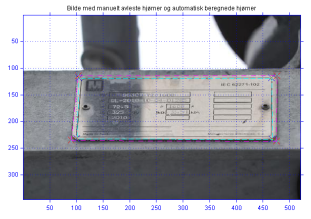


(b) Bilde nr 2

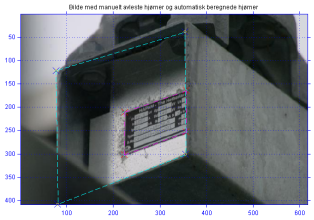
(c) Bilde nr 3



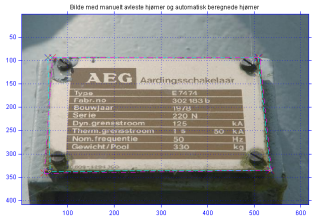
(d) Bilde nr 4



(e) Bilde nr 5

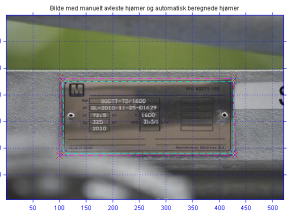


(f) Bilde nr 6

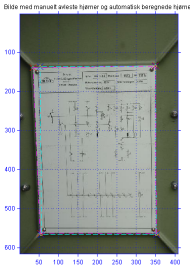


(g) Bilde nr 7

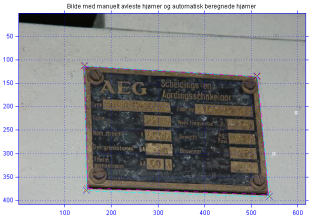
(h) Bilde nr 8



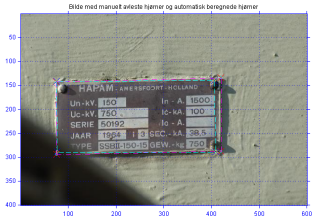
(i) Bilde nr 9



(j) Bilde nr 10

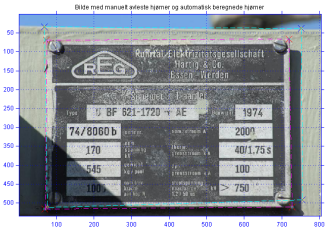


(k) Bilde nr 11

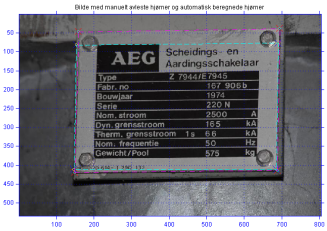


(l) Bilde nr 12

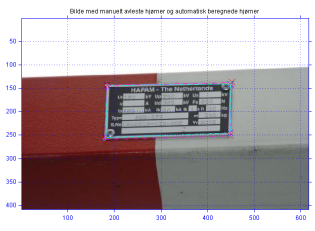
Figure 33: Bilde nr 1 til og med 12 fra Testsett 1.



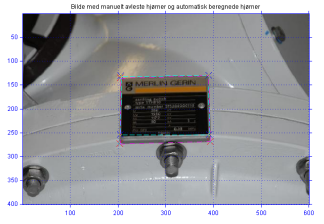
(a) Bilde nr 13



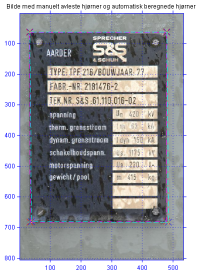
(b) Bilde nr 14



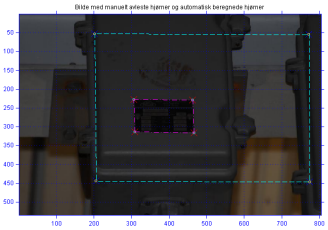
(c) Bilde nr 15



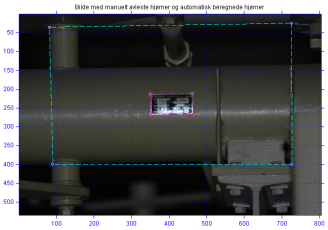
(d) Bilde nr 16



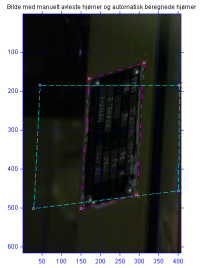
(e) Bilde nr 17



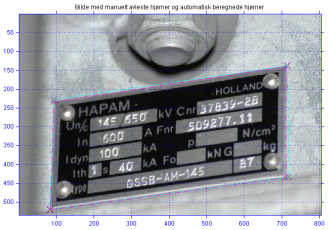
(f) Bilde nr 18



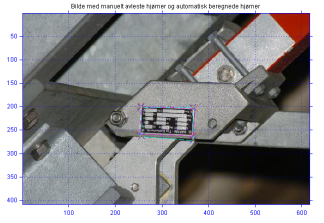
(g) Bilde nr 19



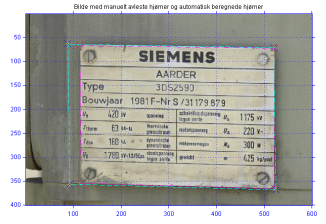
(h) Bilde nr 20



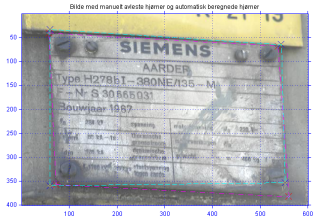
(i) Bilde nr 21



(j) Bilde nr 22

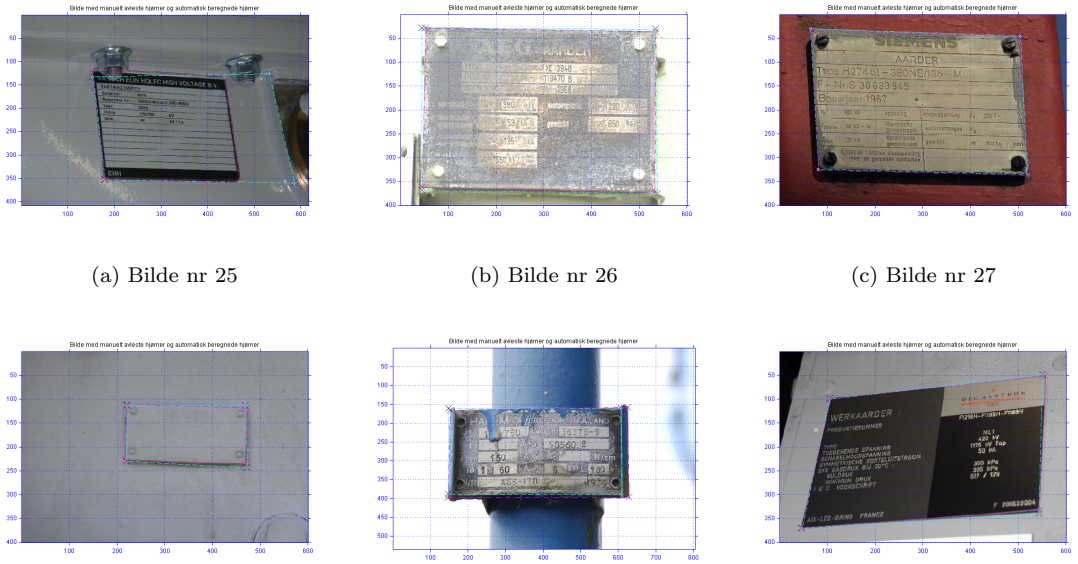


(k) Bilde nr 23



(l) Bilde nr 24

Figure 34: Bilde nr 13 til og med 24 fra Testsett 1.



(a) Bilde nr 25 (b) Bilde nr 26 (c) Bilde nr 27

(d) Bilde nr 28 (e) Bilde nr 29 (f) Bilde nr 30

Figure 35: Bilde nr 25 til og med 30 fra Testsett 1.

6.2 Testsett 2:

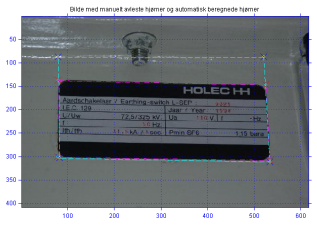
For Testsett 2 kan vi ved å studere resultater i tabell 4 at alle verdier med unntak av spesifisitet har litt dårligere gjennomsnittsverdier i sammenligning med gjennomsnittsverdiene til Testsett 1 fra tabell 2. Noe av grunnen er nok at bilde nr 24 i Testsett 2 som er vist i figur l) ikke klarer å klassifiserer noe av det som i virkeligheten er skilt som skilt, dette gir en TP verdi på 0 som vi ser i tabell 3. Dette medfører videre til at sensitivitetsverdien blir 0 for dette bildet. Hovedgrunnen til denne grove feildeteksjonen er på grunn av parameteren 'minV2' som settes i setParam.m fra kapittel 4.2 har en fast verdi på 40. Det vil si at en minimum vertikal linje må være minimum 40 piksler for ikke å bli fjernet. Den detekterte vertikale venstre kanten til skiltet i figur l) var akkurat under 40 piksler og ble dermed fjernet.
Det kan også være interessant å observere forskjellen på lokaliseringen av skilt nr 20 og nr 21 i figur h) og i). I figur i) er det en skygge som får algoritmen til å tro at den ene detekterte kanten i skiltet hører sammen med den horisontale skyggen som går gjennom bildet og øvre del av skiltet. Dette gir to helt forskjellige resultater på noen bilder som er ganske like.
Noen av resultater er fremdeles ganske gode som den gjennomsnittlige spesifisitet og NPV. Gjennomsnittlige spesifisiteten er på 0.9345 som vil si 93.45% av piksler som er klassifisert til bakgrunn, er i virkeligheten bakgrunn.
NPV verdien er på 0.9380 som vil si at 93.8 % av verdier alle som blir klassifisert til bakgrunn er også i virkeligheten bakgrunn.

Bildenr:	**TN**	**TP**	**FN**	**FP**
1	151610	73898	509	25311
2	113482	121827	3001	1690
3	250246	175067	2527	1496
4	116035	112610	7425	15258
5	148884	16457	0	74659
6	183012	7594	0	60722
7	75553	169582	6193	0
8	73405	91106	75020	469
9	223598	4850	8	22872
10	152220	81346	903	5531
11	96349	140783	2084	784
12	196436	115332	114953	2615
13	76032	140443	22576	949
14	146456	103022	395	1455
15	209970	193296	5004	21066
16	181751	45194	1183	11872
17	222316	11682	615	5387
18	190184	29002	582	20232
19	200762	45386	5132	48
20	217312	30176	2964	876
21	151248	23971	11113	64996
22	147037	99871	2455	1965
23	210224	212532	1160	5420
24	232885	0	4384	14059
25	195525	5224	50579	0
26	412152	7795	9366	23
27	187825	11677	266	51560
28	136117	110703	418	4090
29	192199	56718	986	1425
30	79130	160343	11840	15

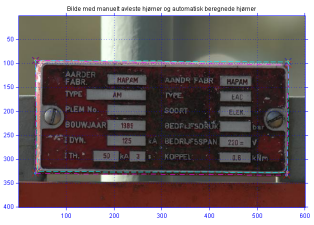
Table 3: Data 1 for Testsett 2

Bildenr:	**PPV**	**NPV**	**FDR**	**Sensitivitet**	**Spesifisitet**	**ACC**
1	0.74487	0.99665	0.25513	0.99316	0.85694	0.89727
2	0.98632	0.97424	0.013682	0.97596	0.98533	0.98045
3	0.99153	0.99	0.0084729	0.98577	0.99406	0.99063
4	0.88067	0.93986	0.11933	0.93814	0.88379	0.90975
5	0.18062	1	0.81938	1	0.66602	0.68892
6	0.11116	1	0.88884	1	0.75087	0.7584
7	1	0.92424	0	0.96477	1	0.97536
8	0.99488	0.49456	0.0051215	0.54842	0.99365	0.68546
9	0.17495	0.99996	0.82505	0.99835	0.9072	0.90896
10	0.93634	0.9941	0.063665	0.98902	0.96494	0.97319
11	0.99446	0.97883	0.005538	0.98541	0.99193	0.98805
12	0.97783	0.63084	0.022171	0.50082	0.98686	0.72616
13	0.99329	0.77105	0.0067118	0.86151	0.98767	0.90198
14	0.98607	0.99731	0.013927	0.99618	0.99016	0.99264
15	0.90173	0.97672	0.098273	0.97477	0.90882	0.93928
16	0.79196	0.99353	0.20804	0.97449	0.93868	0.9456
17	0.6844	0.99724	0.3156	0.94999	0.97634	0.97499
18	0.58906	0.99695	0.41094	0.98033	0.90385	0.91327
19	0.99894	0.97507	0.0010565	0.89841	0.99976	0.97939
20	0.97179	0.98654	0.028211	0.91056	0.99599	0.98472
21	0.26944	0.93155	0.73056	0.68325	0.69943	0.69717
22	0.9807	0.98358	0.019296	0.97601	0.98681	0.98241
23	0.97513	0.99451	0.024868	0.99457	0.97487	0.98467
24	0	0.98152	1	0	0.94307	0.92662
25	1	0.79448	0	0.093615	1	0.79875
26	0.99706	0.97778	0.0029419	0.45423	0.99994	0.97813
27	0.18465	0.99859	0.81535	0.97773	0.78461	0.79379
28	0.96437	0.99694	0.035629	0.99624	0.97083	0.98206
29	0.97549	0.9949	0.024509	0.98291	0.99264	0.99041
30	0.99991	0.86985	9.3541e-005	0.93124	0.99981	0.95283
Gj.snitt:	0.7746	0.9380	0.2254	0.8505	0.9345	0.9067

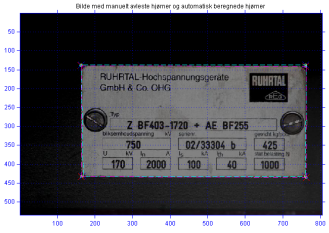
Table 4: Data 2 for Testsett 2



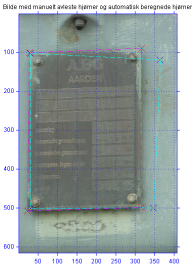
(a) Bilde nr 1



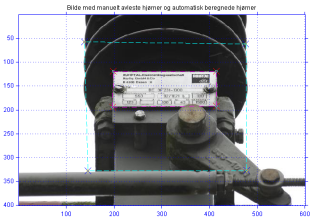
(b) Bilde nr 2



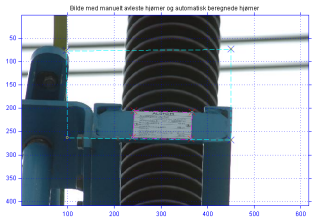
(c) Bilde nr 3



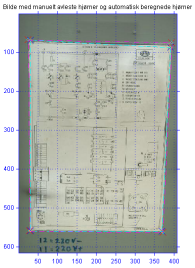
(d) Bilde nr 4



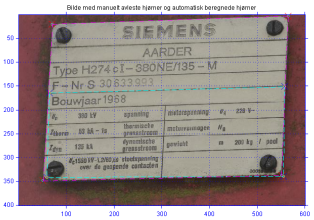
(e) Bilde nr 5



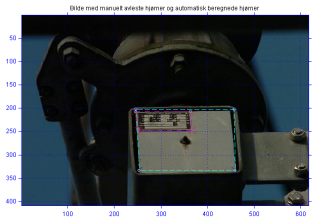
(f) Bilde nr 6



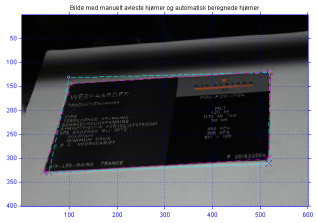
(g) Bilde nr 7



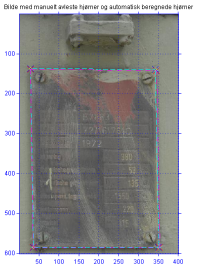
(h) Bilde nr 8



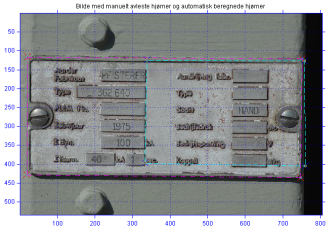
(i) Bilde nr 9



(j) Bilde nr 10

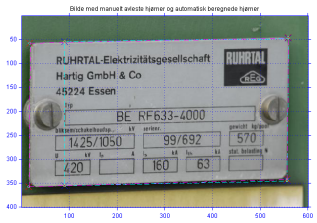


(k) Bilde nr 11

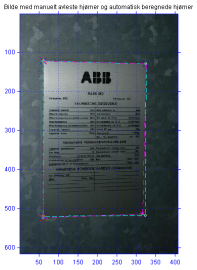


(l) Bilde nr 12

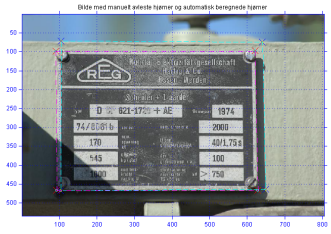
Figure 36: Bilde nr 1 til og med 12 fra Testsett 2.



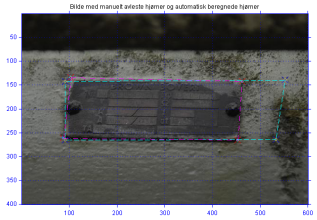
(a) Bilde nr 13



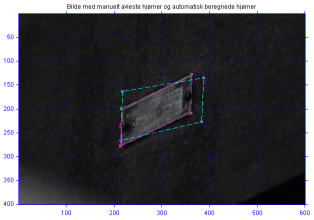
(b) Bilde nr 14



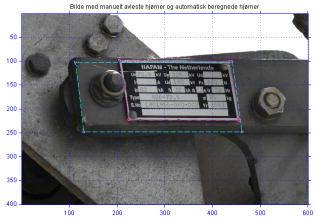
(c) Bilde nr 15



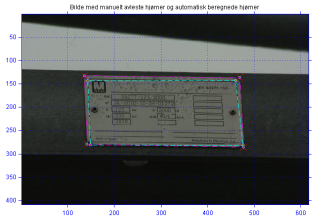
(d) Bilde nr 16



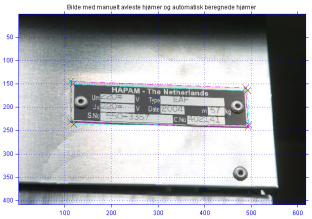
(e) Bilde nr 17



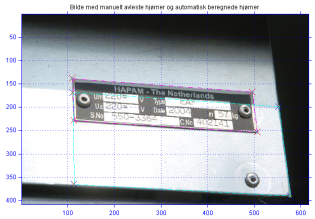
(f) Bilde nr 18



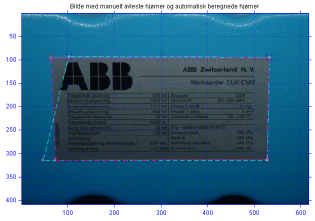
(g) Bilde nr 19



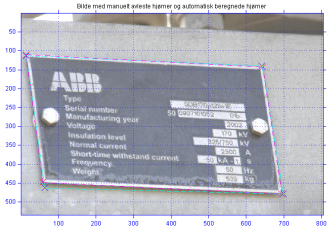
(h) Bilde nr 20



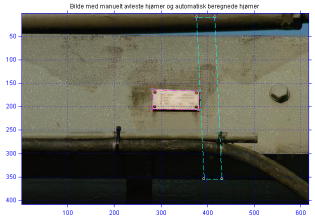
(i) Bilde nr 21



(j) Bilde nr 22

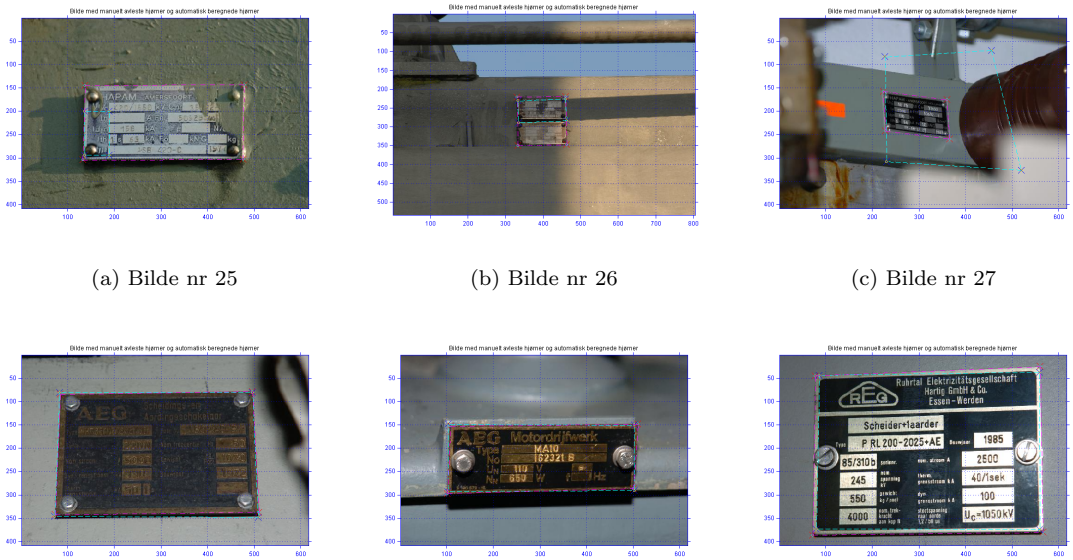


(k) Bilde nr 23



(l) Bilde nr 24

Figure 37: Bilde nr 13 til og med 24 fra Testsett 2.



(a) Bilde nr 25 (b) Bilde nr 26 (c) Bilde nr 27

(d) Bilde nr 28 (e) Bilde nr 29 (f) Bilde nr 30

Figure 38: Bilde nr 25 til og med 30 fra Testsett 2.

Bildenr:	TN	TP	FN	FP
1	240950	9307	1049	22
2	227060	12468	257	215
3	110875	7053	0	122072
4	81281	115253	1453	53341
5	106683	118505	14101	711
6	212157	184657	299	32223
7	108674	129431	520	1375
8	147344	56342	0	47642
9	356654	46744	355	25583
10	203219	45976	696	1437
11	89001	150015	11723	589
12	76676	139221	22020	2083
13	218306	20171	1523	0
14	71902	159228	2052	6818
15	236959	3928	6685	3756
16	147168	102897	400	863
17	245672	166160	17411	93
18	160130	31314	0	48556
19	205341	6410	0	39577
20	94417	135966	9294	323
21	216028	8087	32	15853
22	164110	38332	37558	0
23	94026	119900	28545	8857
24	96334	108891	480	45623
25	238306	5353	7531	138
26	198206	57634	0	173496
27	226169	118443	2865	81859
28	239248	8586	1127	2367
29	399145	13386	91	16714
30	154500	22011	878	73939

Table 5: Data 1 for Testsett 3

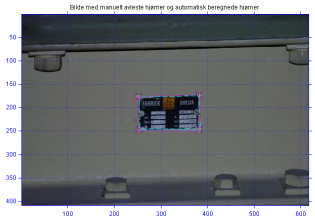
6.3 Testsett 3:

For Testsett 3 kan vi ved å studere tabell 6 observere at gjennomsnittsverdien til spesifisitet er 0.8829 og at sensitivitet har gjennomsnittsverdien 0.9061.
Men det er et par unntak som bilde nr 15, 22 og 25 som er vist i figur 40 c), j) og figur 40 a) som har noe dårlige verdier.

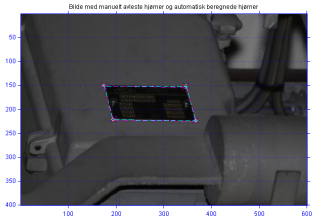
Et bilde som overasker er bilde nr 5 vist i figur 40 e). Her er ikke den venstre vertikale kanten til type-skiltet med. Noe som algoritmen egentlig krever, men på grunn av mangen små vertikale linjer fra data på type-skiltet og skruen, danner disse til sammens en vertikal linje som utgjør denne deteksjonen som er litt heldig. Deteksjonen på bilde nr 5 resulterer faktisk i ganske gode verdier som er vist i tabell 6.

Bildenr:	**PPV**	**NPV**	**FDR**	**Sensitivitet**	**Spesifisitet**	**ACC**
1	0.99764	0.99567	0.0023582	0.89871	0.99991	0.99574
2	0.98305	0.99887	0.016952	0.9798	0.99905	0.99803
3	0.054621	1	0.94538	1	0.47597	0.49137
4	0.68361	0.98244	0.31639	0.98755	0.60377	0.78198
5	0.99404	0.88325	0.005964	0.89366	0.99338	0.93828
6	0.85142	0.99859	0.14858	0.99838	0.86814	0.92425
7	0.98949	0.99524	0.010512	0.996	0.98751	0.9921
8	0.54183	1	0.45817	1	0.75566	0.81044
9	0.64629	0.99901	0.35371	0.99246	0.93307	0.93959
10	0.96969	0.99659	0.030308	0.98509	0.99298	0.99151
11	0.99609	0.88361	0.0039109	0.92752	0.99343	0.95101
12	0.98526	0.77689	0.014741	0.86343	0.97355	0.89957
13	1	0.99307	0	0.9298	1	0.99365
14	0.95894	0.97225	0.041061	0.98728	0.91339	0.96304
15	0.51119	0.97256	0.48881	0.37011	0.9844	0.95846
16	0.9917	0.9973	0.0083	0.9961	0.9942	0.9950
17	0.99944	0.93382	0.00055939	0.90515	0.99962	0.95923
18	0.39206	1	0.60794	1	0.76733	0.79768
19	0.13939	1	0.86061	1	0.83841	0.84253
20	0.99763	0.91039	0.00237	0.93602	0.99659	0.95993
21	0.3378	0.99985	0.6622	0.99606	0.93163	0.93381
22	1	0.81376	0	0.5051	1	0.84351
23	0.93121	0.76711	0.068788	0.80771	0.91391	0.85118
24	0.70473	0.99504	0.29527	0.99561	0.67861	0.81656
25	0.97487	0.96937	0.025132	0.41548	0.99942	0.96949
26	0.24936	1	0.75064	1	0.53324	0.5959
27	0.59132	0.98749	0.40868	0.97638	0.73425	0.80266
28	0.78389	0.99531	0.21611	0.88397	0.9902	0.9861
29	0.44472	0.99977	0.55528	0.99325	0.95981	0.96086
30	0.2294	0.99435	0.7706	0.96164	0.67633	0.70231
Gj.snitt	0.7310	0.9604	0.2690	0.9061	0.8829	0.8882

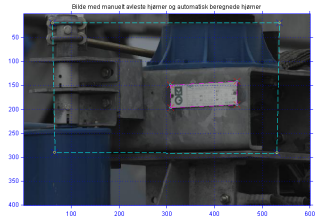
Table 6: Data 2 for Testsett 3



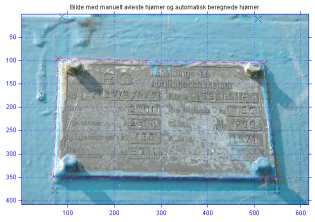
(a) Bilde nr 1



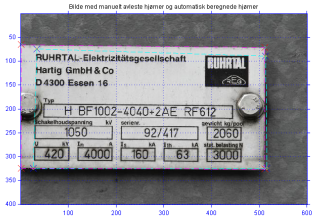
(b) Bilde nr 2



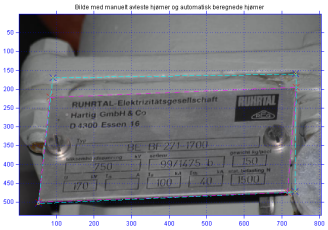
(c) Bilde nr 3



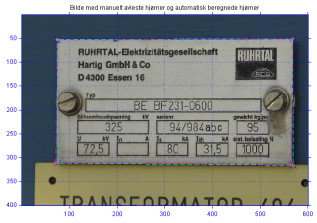
(d) Bilde nr 4



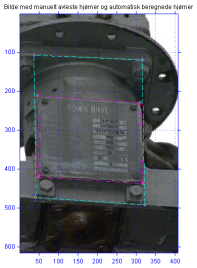
(e) Bilde nr 5



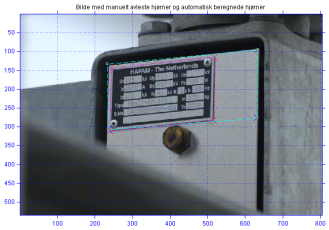
(f) Bilde nr 6



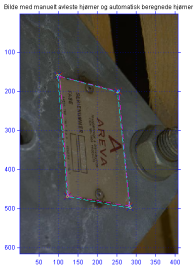
(g) Bilde nr 7



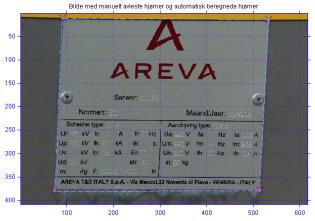
(h) Bilde nr 8



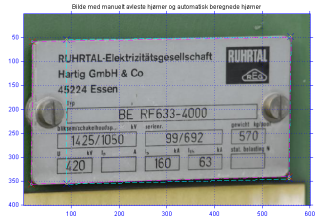
(i) Bilde nr 9



(j) Bilde nr 10

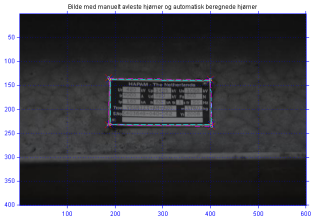


(k) Bilde nr 11

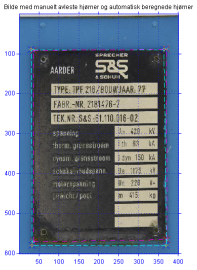


(l) Bilde nr 12

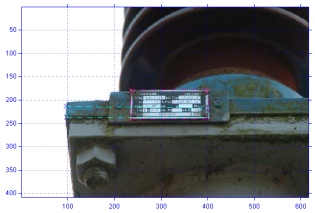
Figure 39: Bilde nr 1 til og med 12 fra Testsett 3.



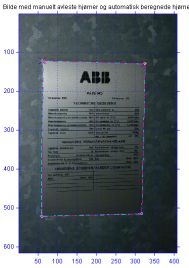
(a) Bilde nr 13



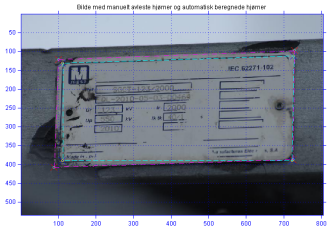
(b) Bilde nr 14



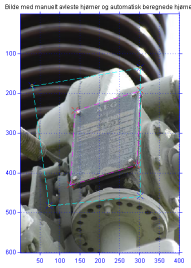
(c) Bilde nr 15



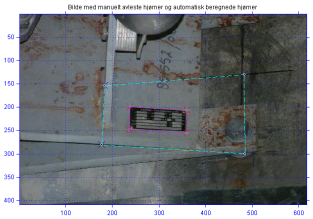
(d) Bilde nr 16



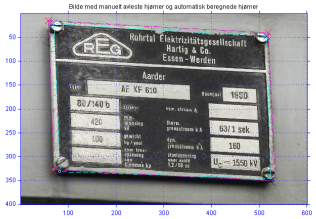
(e) Bilde nr 17



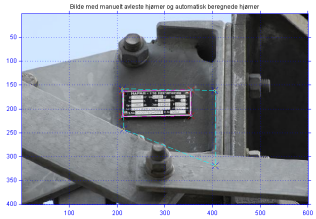
(f) Bilde nr 18



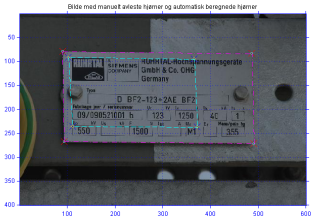
(g) Bilde nr 19



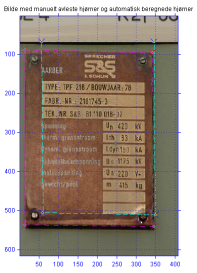
(h) Bilde nr 20



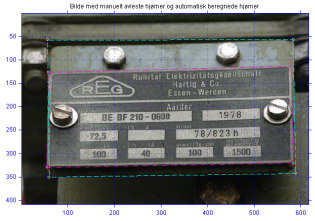
(i) Bilde nr 21



(j) Bilde nr 22

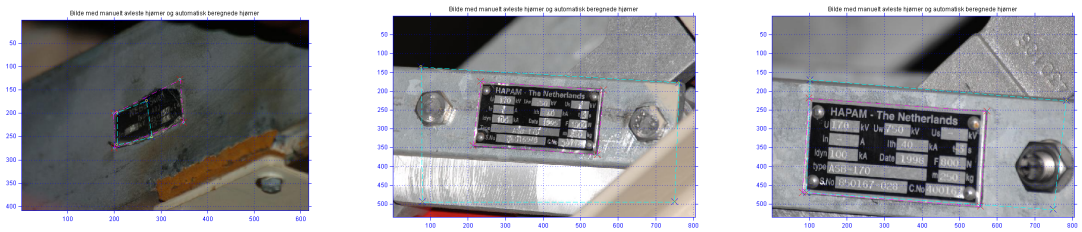


(k) Bilde nr 23

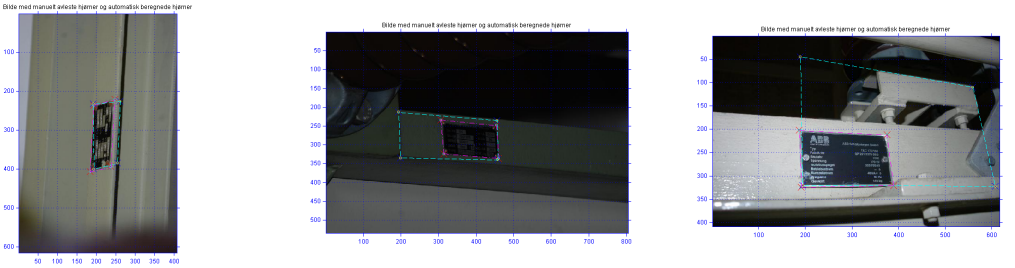


(l) Bilde nr 24

Figure 40: Bilde nr 13 til og med 24 fra Testsett 3.



(a) Bilde nr 25 (b) Bilde nr 26 (c) Bilde nr 27



(d) Bilde nr 28 (e) Bilde nr 29 (f) Bilde nr 30

Figure 41: Bilde nr 25 til og med 30 fra Testsett 3.

Testsett nr:	PPV	NPV	FDR	Sensitivitet	Spesifisitet	ACC
1:	0.8496	0.9796	0.1504	0.9598	0.9134	0.9213
2:	0.7746	0.9380	0.2254	0.8505	0.9345	0.9067
3:	0.7310	0.9604	0.2690	0.9061	0.8829	0.8882

Table 7: Sammenligning av de tre testsettene.

Til slutt ved å studere tabell 7 kommer det frem at resultatene for de 3 testsettene er ganske like. Det er nok på grunn av at det er ganske lik fordeling av samme typen bilder i de tre settene. Både gjennomsnittlig spesitivitet og sensitivitet har relativt høye verdier for alle tre settene.
PPV som er en verdi på hvor mange av alle klassifiseringene som er gjort til skilt som i virkeligheten er skilt. PPV verdien er litt liten for alle tre testsettene og bør bli større. NPV verdien som er en verdi på hvor mangen av alle klassifiseringene som er gjort til bakgrunn som i virkeligheten er bakgrunn. NPV verdien er ganske god gjennom alle tre testsettene og viser at det er potensiale til videre utvikling av algoritmen som kan peile oss enda nærmere målet som er enda bedre resultater. Det kan observeres ved å studere alle figurene med resultatene plottot på til de tre testsettene at det virker som at bildene som inneholder de minste type-skiltene med mangen rette kanter i bakgrunnen, som det er algoritmen har vanskeligst for å lokalisere.

Sensitivitets- og spesifisitets- verdiene er også relativt gode men bør bli bedre. Spesielt sensitivitet-verdien bør bedres.
Det kan også legges merke til at i alle klassifiseringen som blir gjort som skal være type-skilt er det alltid deler av denne deteksjonen som også er en del av et type-skilt i virkeligheten, med unntak av bilde nr 24 i figur 37 l). Altså at i alle klassifiseringene fikk noen sanne positive utenom det ene unntaket som nevnt.

7 Konklusjon og diskusjon

Det er ønskelig å få automatisk lokalisert type-skilt i digitale bilder. Det er blitt utviklet en automatisk algoritme som kan gi både gode og dårlige resultater. Det er blitt brukt Hough Transform til å finne rette linjer og så ut fra de rette linjene finne et type-skilt og plukke ut fire potensielle hjørnekoordinatet til type-skiltet.

Hough Transformen er ganske robust, men det er veldig ofte at den ikke klarer å detektere hele kanten men bare deler av den, selv når kanten kan være nokså tydelig.

På grunn av den store variasjonen i selve størrelsen til typeskilet i kombinasjon med plasseringen til hvor type-skiltet kan befinne seg i bildet har gjort det veldig vanskelig å lage metoder for å sette ulike parametre.

Etter å ha testet algoritmen med 3 testsett på tilsammens 90 bilder kom det frem at spesifisitet- og NPV-verdiene er stabilt ganske gode. Dette betyr at algoritmen værtfall er ganske god til å definere bakgrunn slik at det kan videre søkes i resten av bildet. Ellers har testen gitt vekslende gode og dårlige resultater. Men det virker som algoritmen har godt potensial til videre utvikling.

8 Videre arbeid

- Forbedre deteksjon av rette linjer. Ofte blir bare deler av linjer detekter selv om det i kantdeteksjonen er relativt god kant.
- Kontrast prosseringen kan forbedres.
- Videre sortering av linjesegment ved å bruke fargeinformasjon, varians og antall kanter.
- Kombinere Hough Transform med kontur funksjonen som er innebygget i matlab kalt imcontour.m.
- Forbedre metodene for å beregne forskjellige parametre som brukes
- Lage en robust metode for å finne områder som inneholder tekst slik at informasjon om området kan lettere hentes ut.

Referanser

[1] Rafael C. Gonzalez, Richard E. Woods, and Steven L Eddins. Digital Image Processing using MATLAB. Gatesmark Publishing, second edition edition, 2009.

[2] Ole Ronny Andersen and Thomas Ivesdal-Tronstad. Automatic type plate classification. Forprosjekt masteroppgave, 2011. [University of Stavanger].

[3] Mathworks. Matlab documentation. http://www.mathworks.se/help/toolbox/images/ref/imresize.html, r2012a. [Online; sist testet: 21-Mai-2012].

[4] Mathworks. Matlab documentation. http://www.mathworks.se/help/toolbox/images/ref/wiener2.html, r2012a. [Online; sist testet: 21-Mai-2012].

[5] Li Liqing WangXinHou and HuangXiuBao. Application of wiener filter to automatic inspection of weaving density for woven fabrics. http://metronu.ulb.ac.be/imacs/papers/T1-I-45-1005.pdf, 2002. [Online; sist testet: 21-Mai-2012].

[6] Mathworks. Matlab documentation. http://www.mathworks.se/help/toolbox/images/ref/imadjust.html, r2012a. [Online; sist testet: 21-Mai-2012].

[7] Mathworks. Matlab documentation, hough transform. http://www.mathworks.se/help/toolbox/images/ref/hough.html, r2012a. [Online; sist testet: 31-Mai-2012].

[8] Steve L. Eddins. *Hough transform coordinate system*. http://blogs.mathworks.com/steve/2006/10/19/hough-transform-coordinate-system/, 2006. [Online; sist testet: 21-Mai-2012].

[9] Tao Peng. Mathworks, hough transform. http://www.mathworks.com/matlabcentral/fileexchange/9226-detect-lines-in-grayscale-image-using-hough-transform, 2005. [Online; sist testet: 31-Mai-2012].

[10] Satadal Saha, Subhadip Basu, and Mita Nasipuri. Automatic localization and recognition of license plate characters for indian vehicles. http://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=32&ved=0CFIQFjABOB4&url=http%3A%2F%2Fojs.excelingtech.co.uk%2Findex.php%2FIJCSET%2Farticle%2Fdownload%2F91%2F58&ei=kbbIT9LMH-j_4QTCuOwK&usg=AFQjCNHGMojVOEZkbRAho5cDueyrI0Ufkw, 2011. [Online; sist testet: 1-Juni-2012].

[11] Hamid Mahini, Shohreh Kasaei, Faezeh Dorri, and Fatemeh Dorri. An efficient features-based license plate localization method. http://ipl.ce.sharif.edu/Papers/Icpr06.pdf, 2006. [Online; sist testet: 1-Juni-2012].

[12] Muhammad H Dashtban, Zahra Dashtban, and Hassan Bevrani. A novel approach for vehicle license plate localization and recognition. http://www.ijcaonline.org/volume26/number11/pxc3874382.pdf, 2011. [Online; sist testet: 1-Juni-2012].

[13] Tran Duc Duan, Duong Anh Duc, and Tran Le Hong Du. Combining hough transform and countour algorithm for detecting vehicles' license-plates. http://vision.cs.uiuc.edu/~ddtran2/pubs/tlhdu_isimp04.pdf, 2004. [Online; sist testet: 1-Juni-2012].

[14] Veleppa Canapathy and Wen Lik Dennis Lui. A malaysian vehicle license plate localization and recognition system. http://www.freewebs.com/dennislui/JSCI%20Paper.pdf, 2006. [Online; sist testet: 1-Juni-2012].

[15] Balázs Enyedi, Lajos Konyha, and Kálmán Fazekas. Real time number plate localization algorithms. http://iris.elf.stuba.sk/JEEEC/data/pdf/2_106-2.pdf, 2006. [Online; sist testet: 1-Juni-2012].

[16] Steve L. Eddins. *Showing image pixels associated with a hough transform bin.* http://blogs.mathworks.com/steve/2006/09/01/showing-image-pixels-associated-with-a-hough-transform-bin/, 2006. [Online; sist testet: 21-Mai-2012].

[17] Wikipedia. Positive predictive value. http://en.wikipedia.org/wiki/Positive_predictive_value, 2012. [Online; sist testet: 21-Mai-2012].

[18] Tom Fawcett. An introduction to roc analysis. http://vicos.fri.uni-lj.si/data/vprsistemi/ROCintro.pdf, 2002. [Online; sist testet: 21-Mai-2012].

[19] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern classification. A Wiley-Interscience Publication, second edition edition, 2001.

[20] Wikipedia. Receiver operating characteristic. http://en.wikipedia.org/wiki/Receiver_operating_characteristic, 2012. [Online; sist testet: 21-Mai-2012].

A Matlab-funksjoner brukt i oppgaven

A.1 Liste over Matlab-funksjoner som er blitt brukt:

- testsett.m: Dette er et skript som brukes til å kjøre algoritmen på de 3 forskjellige testsettene, beregne verdier på testresultater og printe figurer.
- finnSTAT.m: Denne filen beregner TN,TP,FN,FP,ACC,PPV,NPV, FDR, Sensitivitet, Spesifisitet.
- grayHT2012.m: Dette er algoritmen som kan kjøres på en bilde om gangen.
- setParam.m: Denne filen setter parametre som blir brukt i grayHT2012.m og andre funksjoner listet i A.2.
- bw2Lines.m: Denne funksjonen henter ut linjesegment fra et binær bilde
- cleanCornerLines.m: Denne funksjonen fjerner linjsegment og krysningspunkt som de har vært med i beregningen av.
- findSingleCorner.m: Fjerner krysningspunkt mellom linjer som antaglivis ikke har noen sammenheng mellom hverandre.
- finnHLines.m: Deler opp et bilde i 2 horisontalte bildeblokker og utfører Hough Transform og plukker ut horisontale linjesegment på bildeblokkene.
- finnHLines12.m: Deler opp et bilde i 12 horisontale bildeblokker og utfører Hough Transform og plukker ut horisontale linjesegment på bildeblokkene.
- finnROI.m: Finner fire hjørnekoordinater ut i fra linjesegment.
- finnVLines.m: Deler opp et bilde i 18 vertikale bildeblokker og utfører Hough Transform og plukker ut vertikale linjesegment.
- finnVLines2.m: Deler opp et bilde i 2 vertikale bildeblokker og utfører Hough Transform og plukker ut vertikale linjesegment.
- lines2bwim.m: gjør om linjesegment til et binærbilde.
- morph.m: Kan utføre vertikal og horisontal morfologisk lukking på linjesegment.
- myClearHlines.m: Blir forklart i 4.5
- myClearVlines.m Blir forklart i 4.5
- myKontrast.m: denne funksjonen er adaptiv og prøver å øke kontrasten til input bildet. Og lager 3 forskjellige output bilder med 3 forskjellige kontraster.
- myVG.m: denne funksjonen beregner kantdetesjonen til de tre inputbildene med funksjonen colorgrad.m. Videre blir det gjort litt kontrast prossesering i forsøk på å forbedre gradientene med funksjonene imadjust og en gammajustering.
- VHLinjerTot.m beregner gråskala Hough Transform ved å dele opp bilder i mindre deler for å kunne gjøre en kraftigere detektering av rette linjer. Til slutt sorteres vertikale og horisontale linjesegment og linjesegment legges sammen.
- grenserMinMax.m: Denne funksjonen fjerner Vertikale linjer som ligger mer enn 'Ratio' utenfor øverste horisontale-grensen og vertikale linjer mer enn 'Ratio' utenfor den nederste horisontale-grensen fjernes. Det samme prinsippet utføres med Horisontale linjer utenfor de ytterste vertikale grensene.

- linecrossings.m: finner krysningspunktet hvor to vektorer krysser hverandre. Linjesegmentene trenger ikke å krysse hverandre men de blir forlenget slik at de treffer hverandre.
- linesegs2lines.m: lager et linje objekt av linjesegmenter som er på formen $[x_1, y_1, x_2, y_2]$, hvor x_1 og y_1 er koordinatene til start punktet til et linjesegment, og x_2 og y_2 er koordinatene til sluttpunktet.
- newlines.m: lager et linjeobjekt ut i fra en vektor og et punkt som vektoren går gjennom.
- vectangle.m: beregner vinkelen mellom to vektorer.
- vectlength.m: beregner lengden til en vektor.
- linjeGrenseV.m: beregner en minimum- og maximum-grense for hvert vertikalt linjesegment som hvert krysningspunkt produsert av dette linjesegmentet må være innenfor. De krysningspunktene utenfor grensene blir fjernet.
- linjeGrenseH.m: beregner en minimum- og maximum-grense for hvert horisontalt linjesegment som krysningspunkt produsert av dette linjesegmenet må være innenfor. De krysningspunktene utenfor grensene blir fjernet.
- DistBetween2Segment.m: beregner minste mulige distanse mellom to linjesegment. Hentet fra matlabcentral
- colorgrad.m: beregner gradient for fargebilder hentet fra [1]
- Hough_Grd.m: Beregner gråskala Hough Transform. Hentet fra matlabcentral [9]
- DrawLines_2Ends.m: Brukes for å plotte linjesegmentene funnet av Hough_Grd.m. Hentet fra matlabcentral [9]
- lines2inf.m: Forlenger linjesegment til de når bilderammen.

A.2 setParam.m

```
function [dH1,dH2,dV1,dV2,BH1,BH2,BV1,BV2,...
          VTH,VTV,minH1,minH2,minV1,...
          minV2,KH1,KH2,KV1,KV2,xDist,...
          yDist,Ratio,Vtol,Htol]=setParam(m,n)
%% setter forskjellige parametre for filen grayHT2012.m:
if m>n
%% PARAMETERSETT 1:

%Avstand til ramme parametre:
%(myClearHLines.m og myClearVLines.m)
dH1=m/25;%35; %hvor nær en H-linje må være H-rammen for å bli fjernet
dH2=n/12;%20; %hvor nær en H-linje må være V-rammen for å bli fjernet
dV1=m/25;%35; %hvor nær en V-linje må være H-rammen for å bli fjernet
dV2=n/20;%30; %hvor nær en V-linje må være V-rammen for å bli fjernet

%minimum lengde på linjesegment som er nær bilde-rammen:
%(myClearHLines.m og myClearVLines.m)
KH1 = floor(0.25*n);
KH2 = floor(0.4*n);
KV1 = floor(0.25*m);
KV2 = floor(0.4*m);

%minimumlengde på horisontale og vertikale linjesegment:
```

```
%(myClearHLines.m og myClearVLines.m)
minH1 = 20; %minimum lengde på horisontale linjesegment 1.sortering
minV1 = 25; %minimum lengde på vertikale linjesegment 1.sortering
minH2 = 40; %minimum lengde på horisontale linjesegment 2.sortering
minV2 = 50; %minimum lengde på vertikale linjesegment 2.sortering

%morphologi paramtetre:
%(morph.m)
imFillH1 = floor(m/40);%avstand mellom linjer som skal legges sammen
imFillV1 = floor(n/20);
imFillH2 = floor(m/20);%avstand mellom linjer som skal legges sammen
imFillV2 = floor(n/15);
BH1=ones(1,imFillH1); %for morphologi av horisontale linjesegment
BH2=ones(1,imFillH2); %for morphologi av horisontale linjesegment
BV1=ones(imFillV1,1); %for morphologi av vertikale linjesegment
BV2=ones(imFillV2,1); %for morphologi av vertikale linjesegment

% Vinkeltoleranse for:
%(vinkelsymmetriH.m og vinkelsymmetriV.m)
VTH = 5; %horisontal vinkeltoleranse
VTV = 4; %vertikal vinkeltoleranse

% Minimum avstand mellom hjørner på til et linjesegment
% (findSingleCorner.m)
xDist = 30;
yDist = 40;

% hvor mangen prosent en linje kan ligge utenfor vertikal og horisontal
% grensene i m-filen 'grenserMinMax.m':
Ratio = 0.8;

% Vinkeltoleranse til HoughLinjer:
% (VHLinjerTot.m)
% er lik for begge parametersett.
Vtol = 22;
Htol = 18;

else
%% PARAMETERSETT 2:

%Avstand til ramme parametre:
% (myClearHLines.m og myClearVLines.m)
dH1=n/25;%35; %hvor nær en H-linje må være V-rammen for å bli fjernet
dH2=m/12;%20; %hvor nær en H-linje må være H-rammen for å bli fjernet
dV2=n/20;%30; %hvor nær en V-linje må være H-rammen for å bli fjernet
dV1=m/15;%25; %hvor nær en V-linje må være V-rammen for å bli fjernet

%minimum lengde på linjesegment som er nær bilde-rammen:
% (myClearHLines.m og myClearVLines.m)
KH1 = floor(0.25*n);
KH2 = floor(0.4*n);
KV1 = floor(0.25*m);
KV2 = floor(0.4*m);
```

```
77
78  %minimumlengde på horisontale og vertikale linjesegment:
79  % (myClearHLines.m og myClearVLines.m)
80  minH1 = 25; %minimum lengde på horisontale linjesegment ved 1.sortering
81  minV1 = 20; %minimum lengde på vertikale linjesegment ved 1.sortering
82  minH2 = 50; %minimum lengde på horisontale linjesegment 2.sortering
83  minV2 = 40; %minimum lengde på vertikale linjesegment 2.sortering
84
85  %morphologi paramtetre:
86  % (morph.m)
87  imFillH1 = floor(m/20);%avstand mellom linjer som skal legges sammen
88  imFillV1 = floor(n/40);
89  imFillH2 = floor(m/15);%avstand mellom linjer som skal legges sammen
90  imFillV2 = floor(n/20);
91  BH1=ones(1,imFillH1); %for morphologi av horisontale linjesegment
92  BH2=ones(1,imFillH2); %for morphologi av horisontale linjesegment
93  BV1=ones(imFillV1,1); %for morphologi av vertikale linjesegment
94  BV2=ones(imFillV2,1); %for morphologi av vertikale linjesegment
95
96  % Vinkeltoleranse for:
97  %(vinkelsymmetriH.m og vinkelsymmetriV.m)
98  VTH = 4; %horisontal vinkeltoleranse
99  VTV = 5; %vertikal vinkeltoleranse
100
101 % Minimum avstand mellom hjørner som blir produsert av et linjesegment
102 % (findSingleCorner.m)
103 xDist = 40;
104 yDist = 30;
105
106 % hvor mangen prosent en linje kan ligge utenfor vertikal og horisontal
107 % (grenserMinMax.m):
108 Ratio = 0.8;
109
110 %Vinkeltoleranse til HoughLinjer:
111 % (VHLinjerTot.m)
112 Vtol = 22;
113 Htol = 18;
114
115 end
```