Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: **Informasjonsteknologi – Master i teknologi/siv.ing. – 5 år**	Vårsemesteret, 2011 **Konfidensiell**
Forfatter: **Jostein Øygarden**	…… (signatur forfatter)
Fagansvarlig: **Tom Ryen (UiS)** Veileder(e): **Harald Ommang (Verico AS)**	
Tittel på masteroppgaven: **Deteksjon og dekoding av strekkoder i digitale bilder.** Engelsk tittel: **Detection and decoding of barcodes in digital images.**	
Studiepoeng: **30**	
Emneord: **Strekkoder, detektering, dekoding, UPC-E, Java, Matlab**	Sidetall: **64** + vedlegg/annet: **CD-ROM** **Stavanger, 3/6-2011**

Masteroppgave Informasjonsteknologi
Datateknikk

DETEKSJON OG DEKODING AV STREKKODER I DIGITALE BILDER

Av:
Jostein Øygarden

Veileder:
Tom Ryen

Veileder:
Harald Ommang

03.06.2011

Sammendrag

Denne masteroppgaven, gitt av Verico AS, presenterer løsninger på både deteksjon og dekoding av strekkoder i digitale bilder. I forbindelse med Vericos verktøy for lagring og systematisering av utstyrsdata, prosesserer de hvert år flere tusen fotografier av strekkodemerkede enheter. Av økonomiske og praktiske årsaker blir fotografiene prosessert i Vericos egne kontorlokaler etter hver befaring. Det mest tidkrevende steget i prosesseringsarbeidet går ut på å manuelt skrive inn strekkoden avlest på hvert bilde. Det er dette steget vi automatiserer i denne oppgaven.

Løsningen av oppgaven ligger i hovedsak i tre algoritmer. De to første algoritmene er implementert i Matlab, og detekterer henholdsvis én og flere strekkoder per bilde. Den siste algoritmen dekoder strekkoder i digitale bilder og er implementert i Java. Algoritmen for detektering av én strekkode per bilde utfører en rekke morfologiske operasjoner for å detektere strekkoden. For å detektere flere strekkoder i samme bilde blir bildets strukturer og kontraster brukt for å vurdere hvor strekkodene er. Den siste algoritmen bruker gråtoneverdier i bildet for å dekode strekkoder funnet av de to første algoritmene.

Algoritmene for detektering og dekoding av én strekkode per bilde er testet på 1001 bilder fra et av Vericos prosjekter. Her detekterer og dekoder algoritmene 98,4% av bildene korrekt. Dette prosjektet har per i dag over ti tusen bilder. Ved å bruke algoritmene utviklet i denne oppgaven ville Verico kommet unna med å manuelt lese av strekkoder i omtrent 160 av de totale 10 000 bildene, noe som hadde spart dem for mye arbeid. Algoritmen for deteksjon og dekoding av flere strekkoder per bilde treffer også godt i eksempelbildene gitt av Verico.

Innhold

Figurer

Tabeller

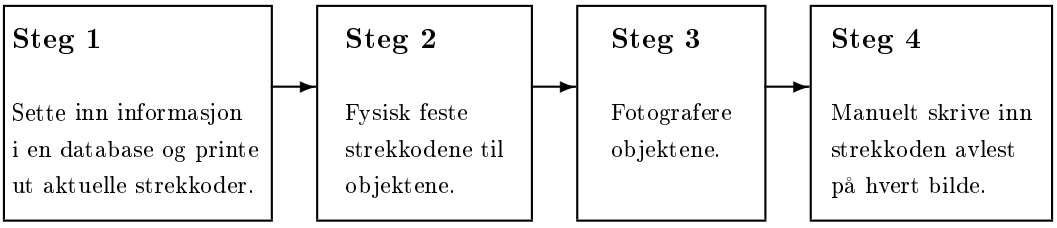
Pseudokoder

1 Introduksjon

Denne masteroppgaven bygger videre på et prosjektarbeid gitt av Verico AS [1] høsten 2010. Dette prosjektarbeidet er også tatt med i denne rapporten, for å gi en best mulig forståelse i problemstilling og metodevalg. Målet med oppgaven er å effektivisere prosesser i oppdrag Verico utfører for sine kunder.

1.1 Motivasjon

Hvert år behandler Verico tusenvis av strekkodemerkede bilder i forbindelse med oppdrag de utfører for sine kunder. Et av produktene Verico tilbyr går ut på å generere et digitalt register over kunders arkiverte dokumenter og eiendeler (arkivrom). Verico har også levert digitale registre over innhold på strømleverandørers nettstasjoner. Felles for disse oppdragene er at klistermerker med strekkoder blir festet på aktuelle objekter, før de fotograferes. Bildene blir til slutt prosessert ved hjelp av Vericos verktøy for å generere de digitale oversiktene. Poenget med strekkodene er at hver enhet skal kunne få et unikt nummer, som er en forutsetning for korrekt identifisering av alle enhetene.



Figur 1.1: Bilde av prosessen før denne oppgaven.

Siden kun bilder blir tatt hos kunden, mens resten av arbeidet kan bli gjort hvor som helst, er denne måten å utføre oppdragene på økonomisk lønnsom. I tillegg åpner denne fremgangsmåten for hurtige befaringer hos kunden, siden selve prosesseringen blir gjort i etterkant. Fremgangsmåten gjør det også mulig for kunden å ta de nødvendige fotografiene selv, med hvilket som helst kamera, for så å sende dem inn til Verico som genererer det digitale registeret.

1.2 Problemstilling

Prosessen for registrering og identifisering av enheter er illustrert i figur 1.1. I steg 1 blir nødvendig informasjon angående oppdraget lagret i en database. For å identifisere korrekt objekt til informasjonen blir strekkoder generert, og printet ut på klistermerker. Disse klistermerkene blir i steg 2 fysisk limt på korrekt objekt, før objektene blir fotograferte i steg 3. Steg 4 går ut på å manuelt lese av og skrive inn strekkodene avlest på fotografiene for å koble fotografiene sammen med informasjonen i databasen. Dette tar lang tid når det er snakk om flere tusen bilder. Ved å automatisere deler av, eller hele dette steget, vil vi redusere denne arbeidsmengden betraktelig. Dette ønsker vi å oppnå ved å implementere automatisk deteksjon og dekoding av strekkodene.

I tillegg vil vi her gå et steg videre. I stedet for å kun dekode en strekkode per bilde, ønsker vi å dekode flest mulig strekkoder i bilder hvor dette er mulig. Dette vil åpne for flere muligheter i sluttproduktet kunden bestiller. Ta utgangspunkt i at et av objektene som skal være med i en digital oversikt er en bokhylle[1]. Denne bokhyllen vil da bli strekkodemerket og fotografert. Dette bildet vil sannsynligvis også vise strekkodemerkede permer på hyllen. Ved dekoding av disse strekkodene åpnes det for

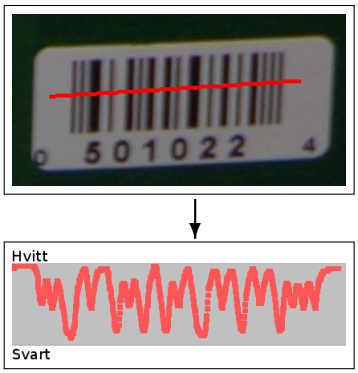
[1] I et arkiv med permer er det viktig å også identifisere bokhyllene, slik at kunden vet hvor permen fysisk befinner seg.

implementering av en visuell søkefunksjon i sluttproduktet. Dersom brukeren av produktet søker etter en av disse permene kan dette bildet vise nøyaktig hvor i hyllen permen står.

1.3 Sammendrag av løsninger

Løsningen av problemstillingen er delt i to hoveddeler. Del 1 går ut på å automatisk detektere og dekode én strekkode per bilde, mens del 2 fokuserer på å detektere og dekode flere strekkoder per bilde.

Oversiktsbilde av løsning på problemstilling del 1:

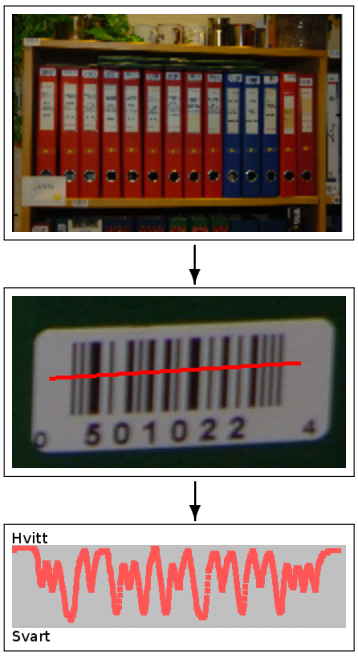


Deteksjon av en strekkode per bilde.
Her benyttes morfologisk bildebehandling. Mer om dette i kapittel 4. Resultater i kapittel 5.

Dekoding av strekkoden.
Strekkoden blir dekodet ved hjelp av gråtoneanalyse av bildet. Mer om dette i kapittel 3.

Figur 1.2: Oversiktsbilde av løsning på problemstilling del 1.

Oversiktsbilde av løsning på problemstilling del 2:



Definere strekkodenes størrelse.
Dette blir gjort ved at brukeren klikker to ganger på bildet som skal prosesseres (første museklikk i starten av en strekkode, andre museklikk i slutten).

Deteksjon av flere strekkoder per bilde.
Strukturer og kontraster i bildet blir brukt for å detektere strekkodene. Mer om dette i kapittel 6. Resultater i kapittel 7.

Dekoding av strekkodene.
Strekkodene blir dekodet ved hjelp av gråtoneanalyse av bildet. Mer om dette i kapittel 3.

Figur 1.3: Oversiktsbilde av løsning på problemstilling del 2.

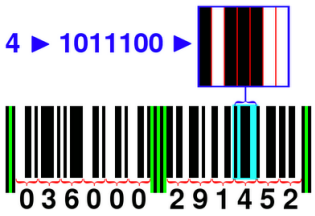
2 Om strekkoder

Før vi begynner å detektere og dekode strekkoder er det nødvendig å bestemme seg for en strekkodetype, og finne ut hvordan denne typen strekkoder er bygget opp.

2.1 Valg av strekkodetype

Kriterier:

- For å gjøre dekodingen av strekkodene mer robust, bør strekkodene ha både start- og sluttkoder (i figuren under er start-, midt-, og sluttkodene grønne). Uten startkode og sluttkode vil det kunne være vanskelig å finne ut hvor strekkoden starter og slutter.



Figur 2.1: UPC strekkode med start-, midt- og sluttkode.

- Å velge en utbredt standard vil være hensiktsmessig med tanke på å unngå eventuelle kompatibilitetsproblemer som kan dukke opp senere. For eksempel om noen i fremtiden ønsker å lese av strekkodene ved hjelp av en standard strekkodeleser.
- Strekkoden må også ta så liten plass som mulig, slik at det er fysisk plass til den på objektene som skal merkes, men fortsatt ha nok siffer til å kunne identifisere tilstrekkelig antall objekter.

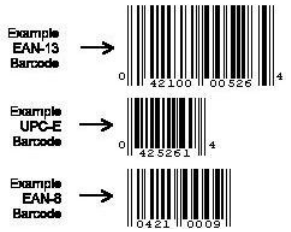
Alternativene:

Det finnes mange standarder for endimensjonale strekkoder [2]. De to viktigste er EAN [3] og UPC [4].

- **EAN (European Article Numbering)**
  EAN er verdens mest brukte strekkodesystem. De fleste strekkodelesere kan dermed lese EAN-strekkoder. Den fysisk minste typen som bærer EAN-navnet er EAN-8 [5]. Denne ble utviklet nettopp på grunn av at EAN-13 i noen tilfeller ble for stor (f.eks. merking av sigarettpakker og penner). EAN-8 har en startkode, en midtkode og en sluttkode. Selv om EAN-8 er en forenkling av EAN-13, er det ikke mulig å direkte oversette en EAN-8-strekkode til en EAN-13-strekkode.
- **UPC (Universal Product Code)**
  UPC er et system tilsvarende EAN. UPC er den orginale strekkoden, og fortsatt mest brukt i USA og Canada. De fleste strekkodelesere kan derfor lese av UPC-strekkoder. Innenfor UPC er den minste standarden UPC-E [6]. Denne er en forenkling av UPC-A [7] (tilsvarende EAN-13). UPC-E har en startkode, en midtkode og en sluttkode. UPC-E-strekkoder kan oversettes direkte til UPC-A-strekkoder.

Konklusjon:

Både EAN-8 og UPC-E tilfredstiller kriteriene vi har. Siden UPC-E-strekkoder tar fysisk mindre plass enn EAN-8, se figur 2.2, faller valget av strekkodetype i denne oppgaven på UPC-E. Det er mulig å identifisere 1 million enheter ved bruk av UPC-E-strekkoder, noe som er tilstrekkelig i alle kundeoppdragene Verico jobber med.



Figur 2.2: Eksempel på EAN-13, UPC-E og EAN-8-strekkode.

2.2 Oppbygging av en UPC-E-strekkode

En UPC-E-strekkode er bygget opp av følgende, lest fra venstre:

- En startkode (alltid ”101“). Svarte striper har verdi ”1“ og hvite striper har verdi ”0“.
- 42 bit som representerer strekkodens 6 tall. Systemnummeret (alltid “0” eller “1”) og kontrollsifferet (mellom “0” og “9”) bestemmer hvordan disse sifferene er kodet (se tabellene under).

Kontrollsiffer	Systemnummer 0	Systemnummer 1
0	EEEOOO	OOOEEE
1	EEOEOO	OOEOEE
2	EEOOEO	OOEEOE
3	EEOOOE	OOEEEO
4	EOEEOO	OEOOEE
5	EOOEEO	OEEOOE
6	EOOOEE	OEEEOO
7	EOEOEO	OEOEOE
8	EOEOOE	OEOEEO
9	EOOEOE	OEEOEO

Tabell 2.1: Paritetstabell (“left-hand even” og ”left-hand odd“).

Tall	Venstremønster (O)	Høyremønster (E)
0	0001101	0100111
1	0011001	0110011
2	0010011	0011011
3	0111101	0100001
4	0100011	0011101
5	0110001	0111001
6	0101111	0000101
7	0111011	0010001
8	0110111	0001001
9	0001011	0010111

Tabell 2.2: Koding av tall i UPC-E-strekkoder.

- Midtkode (alltid ”01010“).
- Sluttkode (alltid ”1“).

Strukturen til en UPC-E-strekkode er helt lik venstre halvdel av en UPC-A-strekkode. Forskjellen mellom UPC-E og UPC-A er at UPC-A har en høyre halvdel etter midtkoden, for så å avslutte med sluttkoden ”101“.

2.3 Eksempel på dekoding av en UPC-E-strekkode

Bildet under viser en UPC-E-strekkode som er laget og printet ut ved hjelp av en strekkodeprinter og medfølgende programvare av Canon. Bildet er tatt med et helt ordinært digitalkamera.

Figur 2.3: Hjemmelaget UPC-E strekkode.

Tabell 2.1 gir oss mønsteret "EOEOOE" (systemnummer=0, kontrollsiffer=8).
Strekkodens oppbygging blir da:

1. Startkode (alltid lik): 101.
2. Første tall (6). Left-hand Even parity: 0000101
3. Andre tall (5). Left-hand Odd parity: 0110001
4. Tredje tall (4). Left-hand Even parity: 0011101
5. Fjerde tall (1). Left-hand Odd parity: 0011001
6. Femte tall (2). Left-hand Odd parity: 0010011
7. Sjette tall (3). Left-hand Even parity: 0100001
8. Midtkode (alltid lik): 01010
9. Sluttkode (alltid lik): 1

Satt sammen blir disse 51 bitene:
101 0000101 0110001 0011101 0011001 0010011 0100001 01010 1
Dette stemmer overens med de horisontale strekene i figur 2.3. Husk at "1"=svart og "0"=hvit.

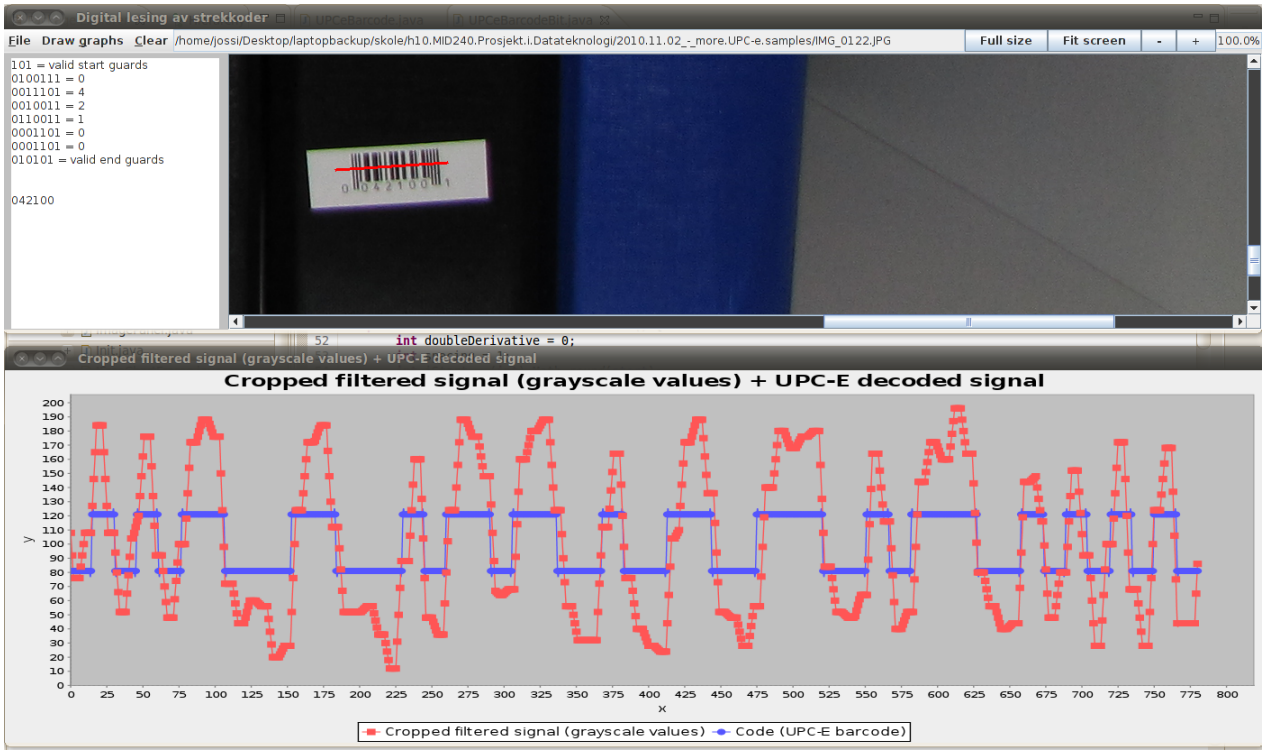
3 Dekoding av UPC-E-strekkoder

Dette kapittelet går gjennom et program for automatisk dekoding av UPC-E-strekkoder. Programmet er skrevet i tredjegenerasjons programmeringsspråket Java [8]. I tillegg til standardbibliotekene som følger med Java, er biblioteket JFreeChart [9] brukt for å tegne grafer. En tråd på JFree's forum [10] er brukt som inspirasjon for å bruke dette biblioteket. Det er to versjoner av dette programmet. Den ene er GUI-basert[2] og lar brukeren selv markere hvor strekkoden er ved bruk av mus. Den andre er kommandolinjeorientert der koordinater for strekkodens plassering gis som parametre.

3.1 GUI-programmet

Det brukeren av dekodingsprogrammet må gjøre for å dekode en UPC-E-strekkode:

1. Starte programmet. Figur 3.1 er et eksempel på hva brukeren ser.
2. Klikke på "File" → Finne et bilde lagret på harddisken → Åpne bildet ved å klikke på "Open".
3. Klikke og dra musen over en strekkode på bildet. Det blir under denne prosessen tegnet en rød linje som starter der bruker klikket inn museknappen, og ender der brukeren slipper museknappen. Linjen må begynne på venstre side av strekkoden, og slutte på høyre side av strekkoden for at programmet skal greie å dekode strekkoden. Programmet vil dekode strekkoden korrekt i situasjoner hvor streken starter og/eller slutter enten på det hvite klistermerket, eller på permen klistermerket er festet på. Dersom strekkoden i bildet er utydelig, eller ikke en UPC-E-strekkode i det hele tatt, vil informasjonsrammen til venstre i figuren vise nøyaktig hva som gikk galt. Brukeren kan også se hvordan strekkoden blir tolket grafisk ved å klikke på en av knappene under menyen "Draw Graphs".

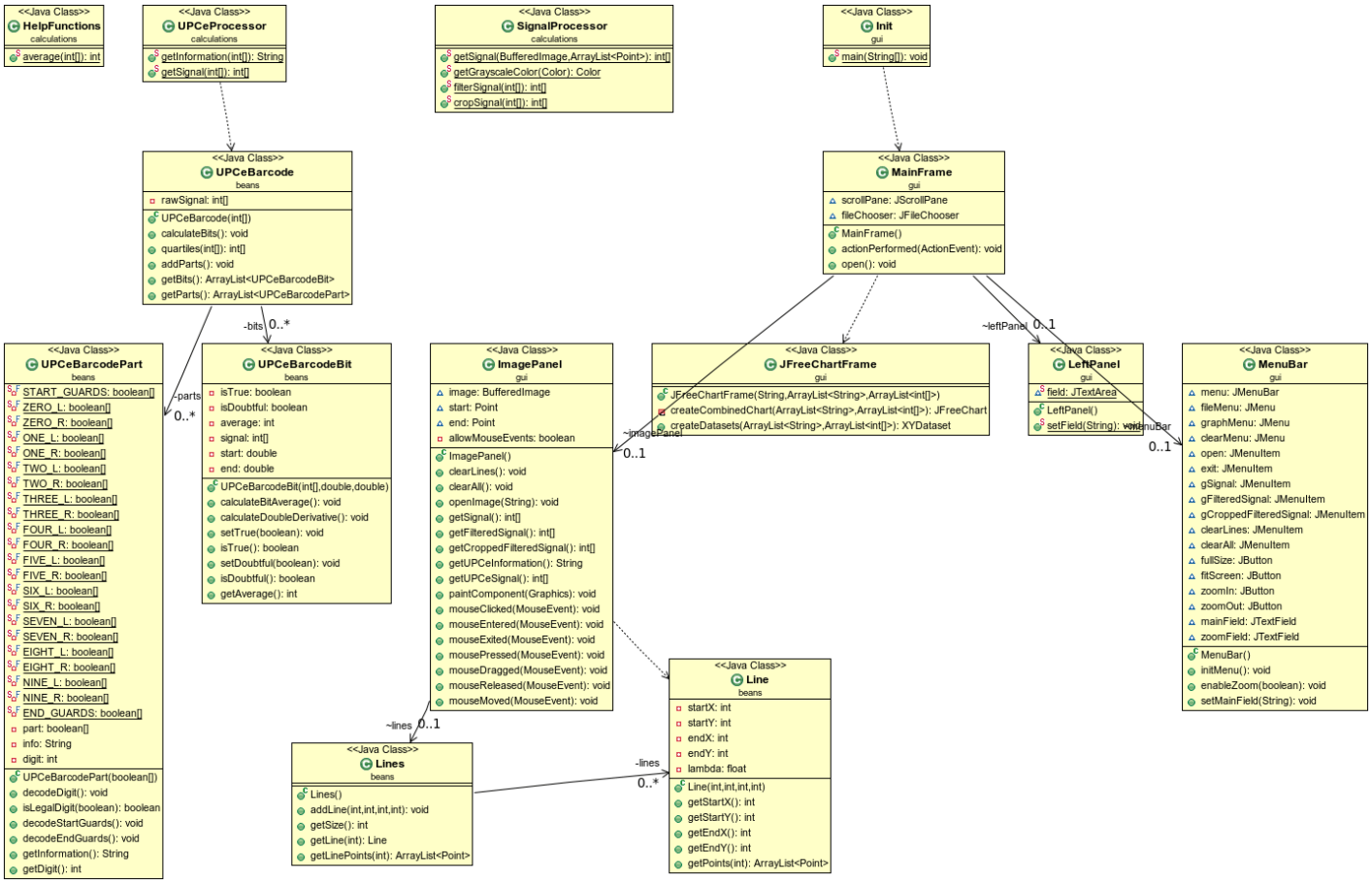


Figur 3.1: Dekodingsprogram - Programmet i bruk.

[2]GUI - Graphical User Interface - Grafisk brukergrensesnitt

3.2 Klassediagram

Dette diagrammet er tegnet i Eclipse [11], ved hjelp av pluginen "ObjectAid UML Explorer" [12].



Figur 3.2: Dekodingsprogram - Klassediagram

- Dekodingsprogrammets grensesnitt ("MainFrame") er delt opp i tre rammer (figur 3.1):
  - Menylinjen ("MenuBar").
  - Den hvite informasjonsrammen til venstre ("LeftPanel").
  - Rammen som viser bildet med strekkoden ("ImagePanel").
- "JFreeChartFrame" tegner grafen som vises nederst i figur 3.1.
- Strekkoden blir dekodet av klassen "UPCeProcessor".

3.3 Algoritmen for dekoding

I det brukeren slipper museknappen etter å ha tegnet den røde linjen i programvinduet (punkt 3 i kapittel 3.1), blir algoritmen beskrevet under utført i bakgrunnen.

3.3.1 Algoritmens hovedpunkter

1. 1000 koordinater langs linjen tegnet i programvinduet blir beregnet (kapittel 3.3.2 - punkt a).
2. Fargeverdien til hver av disse 1000 koordinatene blir hentet ut, og omformet til gråtoneverdier. Om linjen for eksempel er kun 50 piksler bred vil tabellen med gråtoneverdiene inneholde 20 eksemplarer av hver piksel ($20 \cdot 50 = 1000$).
3. Signalet (de 1000 gråtoneverdiene) blir så lavpassfiltrert (kapittel 3.3.2 - punkt b).
4. Signalets gjennomsnitt blir regnet ut (kapittel 3.3.2 - punkt c).
5. Signalet blir klippet ved startkoden og sluttkoden (kapittel 3.3.2 - punkt d).
6. Etter at signalet er filtrert og klippet blir gjennomsnittet av signalets verdier regnet ut på nytt (kapittel 3.3.2 - punkt c).
7. Signalets gråsone blir definert (kapittel 3.3.2 - punkt e).
8. Signalet blir delt opp i 51 perioder (på grunn av at UPC-E strekkoder er bygget opp av 51 bit). Følgende skjer med hver av de 51 periodene:
   (a) Periodens gjennomsnitt blir regnet ut (kapittel 3.3.2 - punkt f).
   (b) Om periodens gjennomsnitt (kapittel 3.3.2 - punkt f) er mindre enn signalets gjennomsnitt (kapittel 3.3.2 - punkt c) vil denne perioden få bitverdi "1". Er periodens gjennomsnitt større enn signalets gjennomsnitt vil bitverdien bli satt til "0".
   (c) Dersom periodens gjennomsnitt (kapittel 3.3.2 - punkt f) er innenfor signalets gråsone (kapittel 3.3.2 - punkt e) blir perioden merket som usikker.
9. Alle periodene som er blitt merket som usikre blir så sjekket og eventuelt korrigert (kapittel 3.3.2 - punkt g).
10. Nå har alle periodene fått hver sin verdi (enten "0" eller "1"). De 51 periodene blir satt sammen til et signal igjen (51 bit), og dekodet til 6 siffer (kapittel 3.3.2 - punkt h).

3.3.2 Detaljer i hvert hovedpunkt

a) Finne koordinater langs linjen tegnet på skjerm

Pseudokode 3.1 Dekodingsprogram - Finne koordinater langs linjen tegnet på skjerm.

for $n = 1$ to 1000 **do**
  $\lambda = \frac{n}{1000}$
  $x_n = (1 - \lambda)x_{start} + \lambda x_{end}$
  $y_n = (1 - \lambda)y_{start} + \lambda y_{end}$
end for

Her er (x_{start}, y_{start}) og (x_{end}, y_{end}) koordinatene til endepunktene.

b) Lavpassfiltrering av signal

Signalet (1000 gråtoneverdier [kapittel 3.3.1 - punkt 2]) kan inneholde noe støy, som vil gi utslag i avlesningen om det ikke blir filtrert. Et "Moving Average Filter" [13] er brukt for å jevne ut signalet, og dermed redusere denne støyen. Se resultat av filtreringen i figur 3.3. Her er Z_n innsampel og W_n utsampel.

Pseudokode 3.2 Dekodingsprogram - Filtrering av signalet.

for $n = 1$ to 3 **do**
  $W_n = \frac{1}{n} \sum_{i=0}^{n-1} Z_{n-i}$
end for
for $n = 4$ to 1000 **do**
  $W_n = \frac{1}{4} \sum_{i=0}^{3} Z_{n-i}$
end for

c) Utregning av signalets gjennomsnitt

Følgende likning blir brukt for å regne ut signalets gjennomsnitt: $\boxed{\overline{W} = \frac{1}{1000} \sum_{i=1}^{1000} W_n}$

d) Klipping av signal

Klipping av signalet (signalet = 1000 gråtoneverdier [kapittel 3.3.1 - punkt 2]) gjør at den røde linjen over strekkoden ikke trenger å treffe nøyaktig på start- og sluttkoden.

Først blir en grense nødvendig for klippingen regnet ut. Denne grensen er midt mellom signalets øvre kvartil[3] og median. Deretter blir signalet klippet på begge sider slik at første og siste verdi er større enn denne grensen. Da er begge sider av signalet som er utenfor merkelappen klippet bort. Til slutt blir signalet klippet slik at første og siste verdi er mindre enn signalgjennomsnittet (kapittel 3.3.2 - punkt c). Nå er også alt det hvite på merkelappen før startkoden og etter sluttkoden klippet bort.

Pseudokode 3.3 Dekodingsprogram - Klipping av signal.

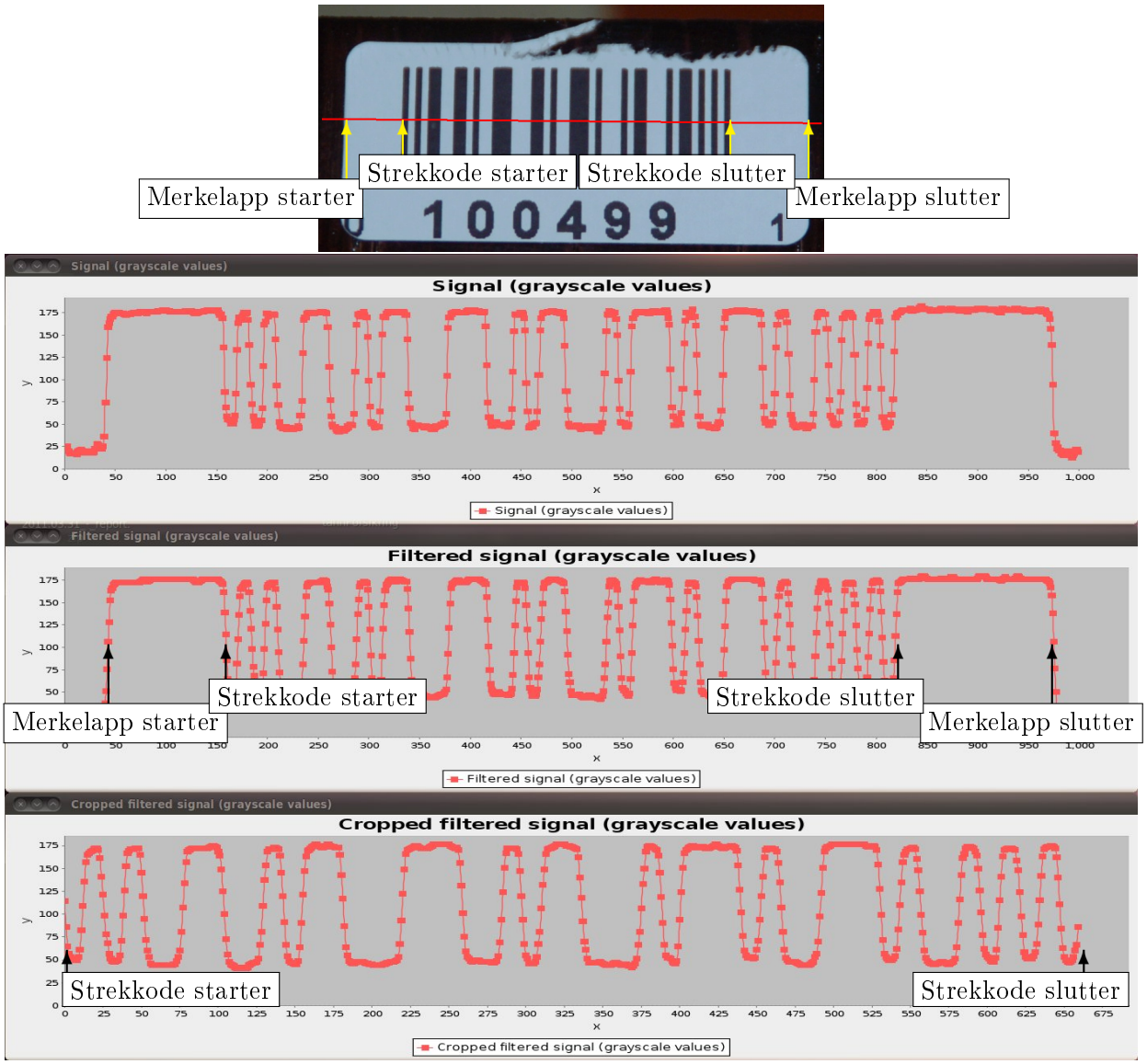
Require: *signalAverage* already calculated

Require: Signal values sorted in alphabetic order
$n = \#pointsInSignal$
$median = signalValue_{\frac{n+1}{2}}$
$upperQuartile = signalValue_{3.\frac{n+1}{4}}$
$signalLimit = \frac{upperQuartile+median}{2}$

Require: Signal values in original order
$start = 0$
$end = \#pointsInSignal$
while $signal[start] < signalLimit$ **do**
  $start = start + 1$ //Remove everything left for the Barcode label
end while
while $signal[start] > signalAverage$ **do**
  $start = start + 1$ //Remove everything left for the first black line
end while
while $signal[end] < signalLimit$ **do**
  $end = end - 1$ //Remove everything right for the Barcode label
end while
while $signal[end] > signalAverage$ **do**
  $end = end - 1$ //Remove everything right for the last black line
end while

Alt før "start" og etter "end" blir klippet bort. Dette gjør at signalets størrelse etter klipping vil være mindre enn 1000, som eksempelet i figur 3.3 på neste side viser.

[3] Øvre kvartil på et datasett er grensen der 25% av innholdet i det sorterte datasettet er over.

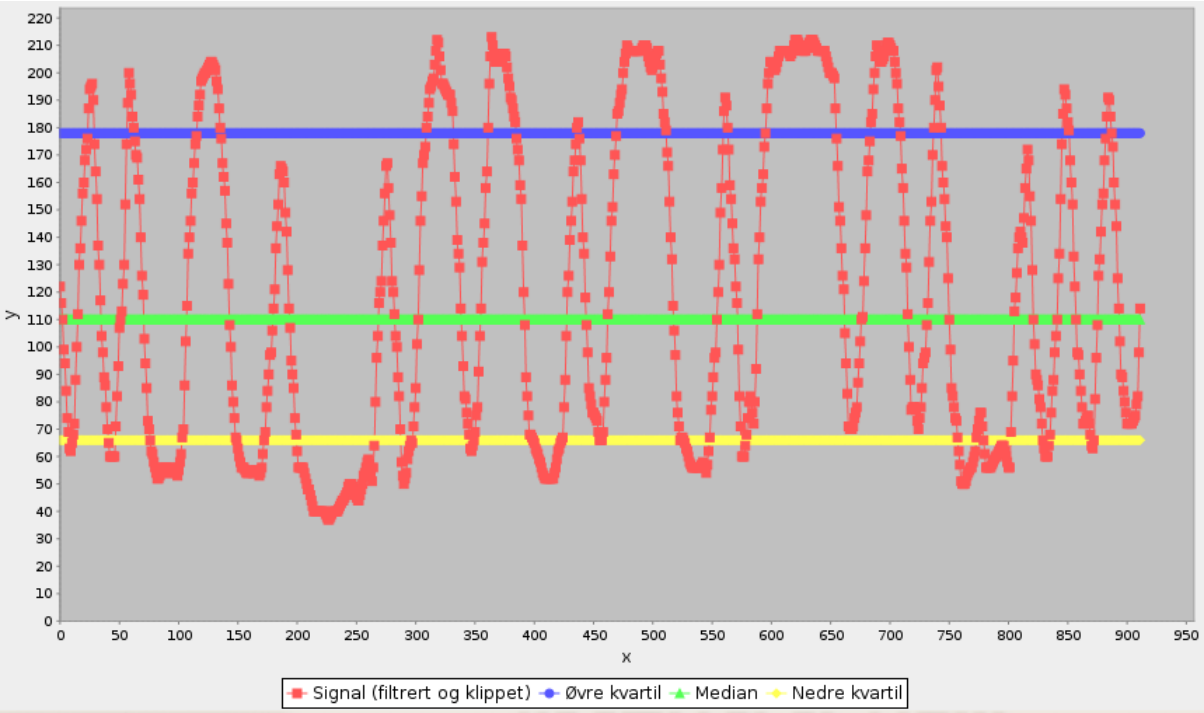


Figur 3.3: Dekodingsprogram - Behandling av signalet til en UPC-E strekkode.

Den øverste av de tre grafene i figur 3.3 viser gråtoneverdier langs den røde linjen over strekkoden. Som vi ser gir sort på bildet lave gråtoneverdier, mens hvit gir høye gråtoneverdier. Grafen i midten viser resultatet etter at gråtoneverdiene er blitt lavpassfiltrert. I den siste grafen i figuren er signalet blitt klippet, slik at det kun inneholder filtrerte gråtoneverdier fra selve innholdet i strekkoden.

e) Definering av signalets gråsone

Gråsonen er et belte som strekker seg fra en egendefinert øvre og nedre grense. Signalets øvre og nedre kvartil blir brukt for å beregne disse grensene. Øvre kvartil er grensen der 25% av datasettet er over, mens nedre kvartil er grensen hvor 25% av datasettet er under.



Figur 3.4: Dekodingsprogram - Øvre kvartil, median og nedre kvartil til et filtrert og klippet signal.

Gråsonens øvre grense er midt mellom øvre kvartil og median, mens nedre grense er midt mellom median og nedre kvartil.

Pseudokode 3.4 Dekodingsprogram - Beregning av signalets gråsone.

Require: Signal values sorted in alphabetic order

$n = \#pointsInSignal$

$lowerQuartile = signalValue_{\frac{n+1}{4}}$

$median = signalValue_{\frac{n+1}{2}}$

$upperQuartile = signalValue_{3.\frac{n+1}{4}}$

$MAX = \frac{upperQuartile+median}{2}$

$MIN = \frac{lowerQuartile+median}{2}$

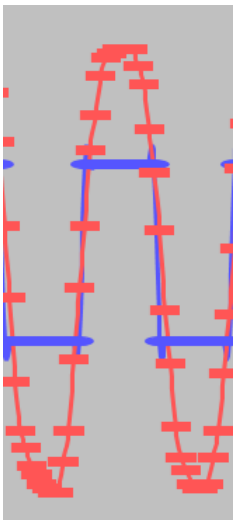
f) Utregning av periodens gjennomsnitt

Eksempel: Hele signalet (gråtoneverdier langs linjen tegnet over strekkoden av brukeren) har totalt 748 punkter. Siden det er en UPC-E-strekkode skal dette da deles opp i 51 perioder (kapittel 2.2). Pseudokoden under viser hvordan gjennomsnittet til den første av periodene regnes ut.

Pseudokode 3.5 Dekodingsprogram - Utregning av periodens gjennomsnitt.

$n = \frac{748}{51} \approx 14.6667 \approx 15$
$spacing = \frac{n}{5} = \frac{15}{5} = 3$
$periodAverage = \frac{1}{n-(2 \cdot spacing)} \sum_{i=1+spacing}^{n-spacing} signalValue_i$

Grunnen til at 1/5 av verdiene i både starten og slutten av perioden ikke tas med når gjennomsnittet blir kalkulert, er at disse verdiene indikerer overgang til forrige/neste periode. Siden signalet også er blitt lavpassfiltrert tidligere (kapittel 3.3.2 - punkt b), er disse verdiene også blitt ytterligere påvirket av naboperiodene.

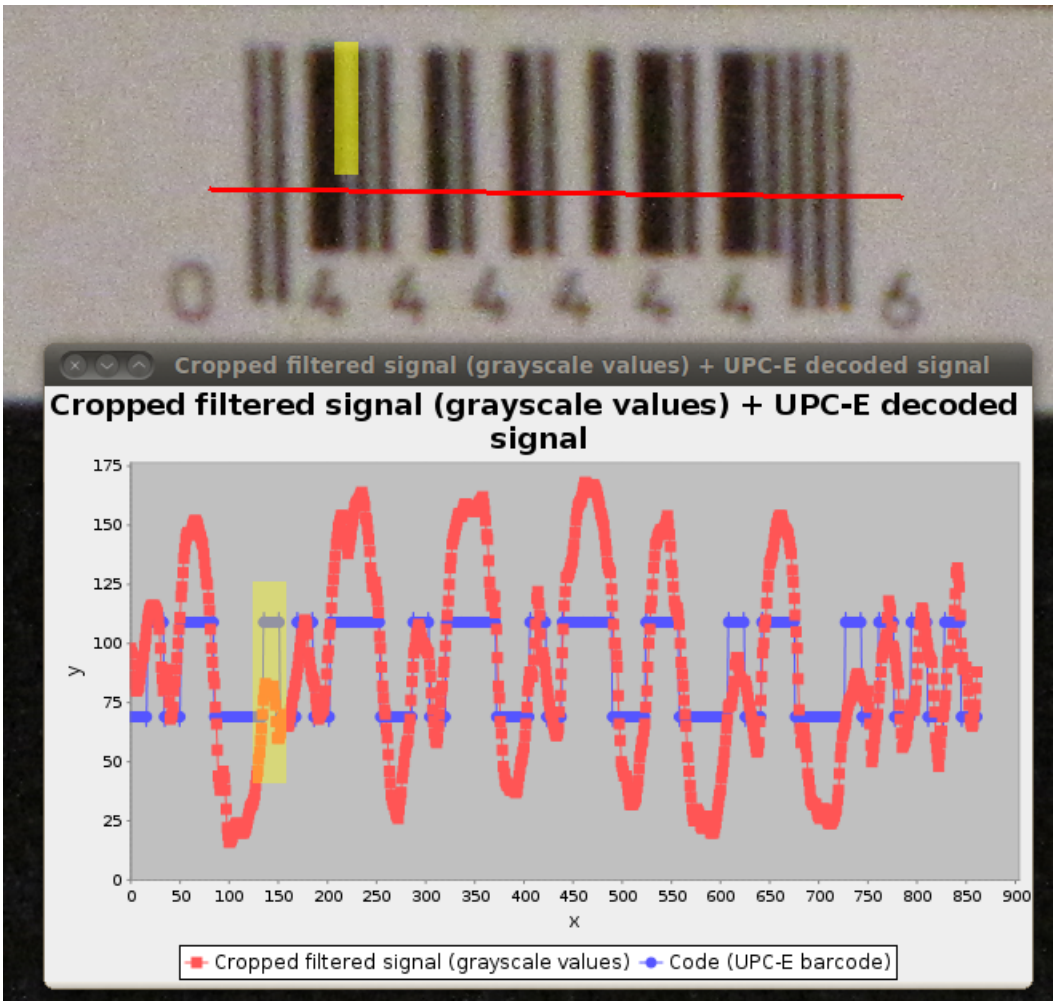


Figur 3.5: Dekodingsprogram - Utregning av periodens gjennomsnitt.

Figur 3.5 viser tre perioder i et UPC-E-signal. Rødt viser de filtrerte gråtoneverdiene, mens blått viser hvor periodene starter og slutter. I dette bildet har hver periode 15 gråtoneverdier, noe som vil si at periodenes gjennomsnitt blir regnet ut fra de 9 midterste verdiene i hver periode (se pseudokode 3.5). Vi kan se at de 3 ytterste gråtoneverdiene for hver periode indikerer overgang til forrige/neste periode. Det er også tydelig at de 9 midterste verdiene i hver periode er nok for å vurdere om streken er hvit eller svart.

g) Korrigering av usikre perioder

En strekkode med en tynn hvit horisontal strek mellom to svarte kommer ofte ikke tydelig frem i et filtrert signal. Bildet under viser en slik situasjon.



Figur 3.6: Dekodingsprogram - UPC-E strekkode med en usikker periode.

Før prosessen beskrevet under ble utført, var den hvite perioden merket gult i figur 3.6 tolket som en usikker svart periode (altså bit-verdi "1")[4]. Grunnen til at den ble merket usikker er at periodens gjennomsnitt (kapittel 3.3.2 - punkt f) er innenfor signalets gråsone (kapittel 3.3.2 - punkt e).

En usikker periode som har samme bit-verdi som naboen(e) blir vurdert ved hjelp av informasjon om den dobbelderiverte, i stedet for gjennomsnittsverdi, som blir brukt på alle andre tilfeller. Den dobbelderiverte for hvert punkt i perioden blir regnet ut (metoden for utregningen er hentet fra [14]). Er summen av disse verdiene positiv vil det si at grafen er konveks, den riktige bitverdien i et UPC-E-signal er da "1". Dersom summen av verdiene er negativ er grafen konkav, og bitverdien blir satt til "0".

[4]Husk: Mørke partier i bildet har lave gråtoneverdier. Svart i strekkoden skal ha bitverdien "1".

Pseudokode 3.6 Dekodingsprogram - Korrigering av usikre perioder.

for $i = 1$ **to** 51 (Antall perioder i en UPC-E-strekkode) **do**
  if $periode_i$ er merket som usikker **then**
    if $periode_i$ har samme bitverdi som naboen(e) **then**
      $sum = 0$ (Sum av de dobbelderiverte)
      for $j = periodStart + 1$ **to** $periodEnd - 1$ **do**
        $sum = sum + signalValue_{j+1} - 2 \cdot signalValue_j + signalValue_{j-1}$
      end for
      if $sum \geq 0$ **then**
        bitverdien til $periode_i$ = "1"
      else
        bitverdien til $periode_i$ = "0"
      end if
    end if
  end if
end for

Pseudokode 3.6 viser hvordan de usikre periodene blir korrigert. For pseudokoden gjelder:

- Svarte perioder skal ha bitverdi "1", hvite skal ha bitverdi "0".
- Signalet består av gråtoneverdier langs linjen tegnet over strekkoden av brukeren, se figur 3.6.
- "periodStart" og "periodEnd" forteller hvor i signalet perioden starter og slutter.
- $signalValue_j$ = Gråtoneverdi på plass "j" i signalet.

h) Dekoding av signal

Signalet som skal dekodes består av 51 perioder, altså 51 bit (kapittel 2.2).

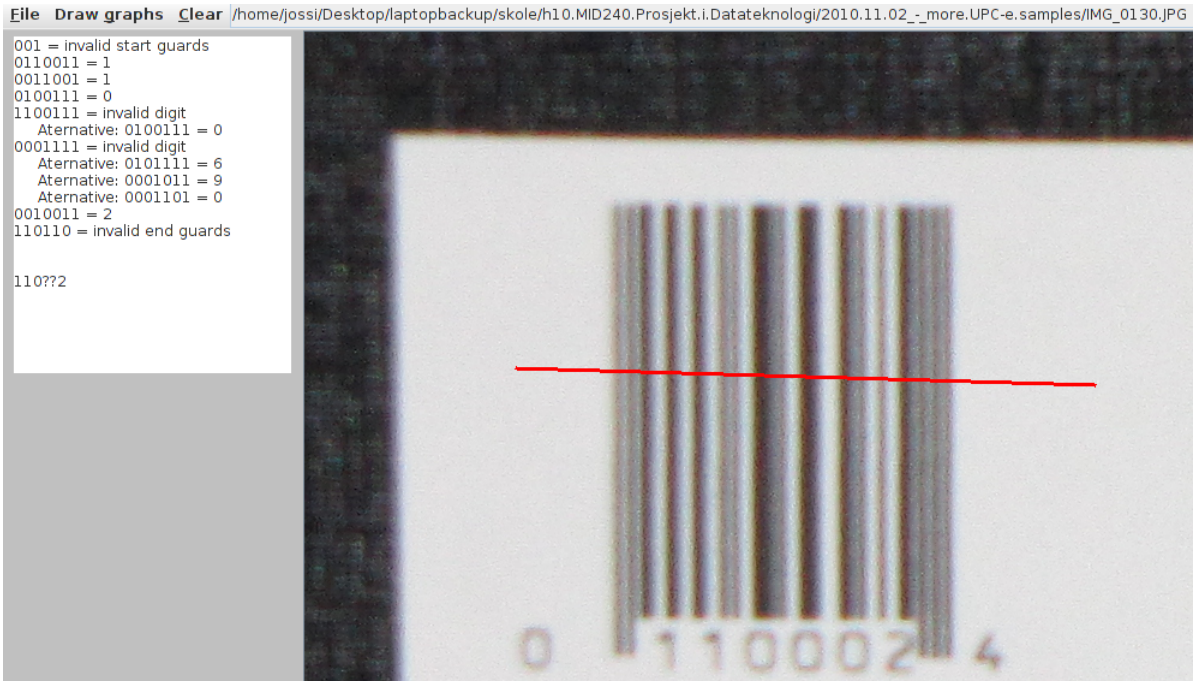
Er de første 3 bitene lik ”101” betyr det at startkoden er riktig avlest.

De neste 42 bitene blir delt opp i seks deler (7 bit per del). Hver av delene blir sammenlignet med de 20 kombinasjonene en UPC-E-strekkode kan ha (tabell 2.2), for å finne ut hvilke tall de representerer.

Dersom bildet er utydelig kan det oppstå situasjoner hvor den avleste koden ikke stemmer med noen av de 20 gyldige kombinasjonene. Programmet vil da prøve automatisk feilretting.

Totalt er det 128 mulige kombinasjoner ($2^7 = 128$). Det vil altså si at vi har et 7-dimensjonalt binært rom, der kun 20 av 128 kombinasjoner er gyldige.
Hvis programmet leser av en kombinasjon som ikke er gyldig, vil algoritmen automatisk vise de lovlige kombinasjonene som ligger nærmest i det 7-dimensjonale rommet. Dette blir gjort ved å bytte på en og en av de 7 bitene, for å sjekke om dette kan gi en gyldig kombinasjon. Figur 3.7 viser dette i praksis.

Er de siste 6 bitene lik ”010101“ betyr det at midt- og sluttkoden er riktig avlest.

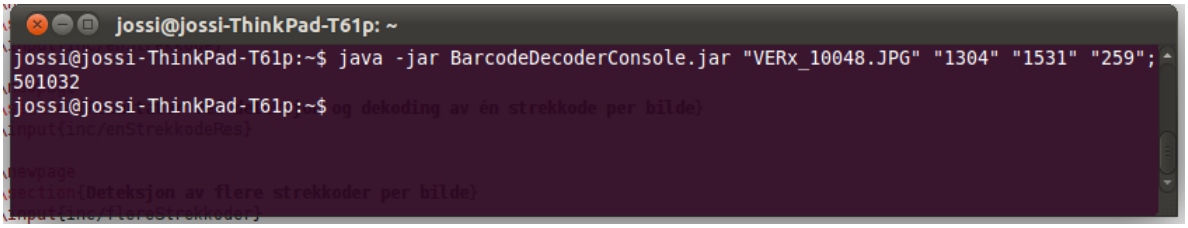


Figur 3.7: Dekodingsprogram - Dekoding av utydelig fotografi.

I figur 3.7 viser den hvite rammen til venstre resultatet av å dekode strekkoden under den røde streken. Informasjonen i rammen forteller oss at tall nummer en, to, tre og seks i strekkoden ble dekodet korrekt. Tall nummer fire ble ikke dekodet korrekt, men her får brukeren opp et forslag til hva tallet kan være. Dette forslaget stemmer i dette eksempelet. Tall nummer fem blir heller ikke dekodet korrekt, men her finner vi riktig tall blant de tre forslagene brukeren blir gitt. Helt nederst i rammen finner vi resultatet av dekodingen (”110??2“).

3.4 Konsollprogram

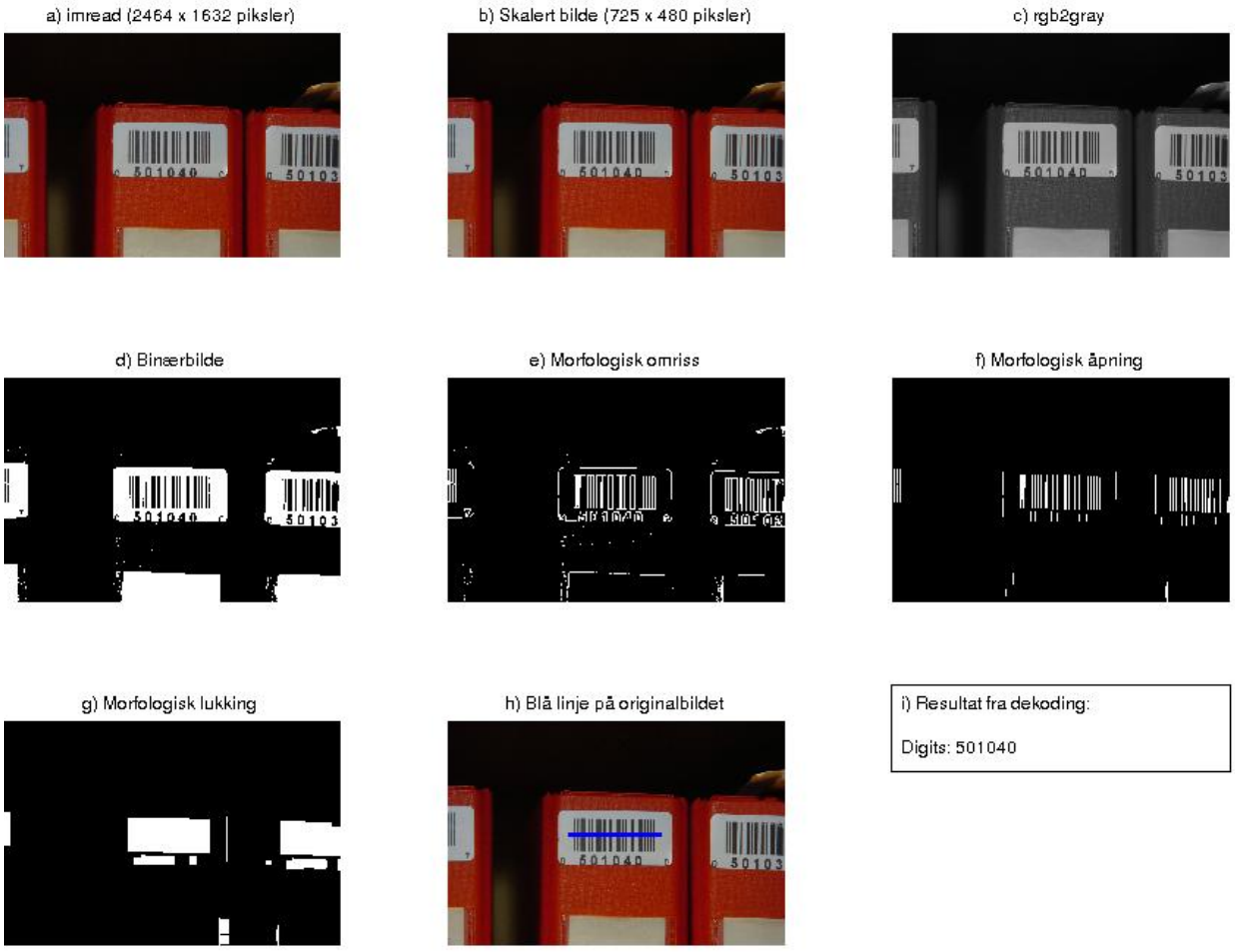
I tillegg til GUI-programmet i kapittel 3.1, er også en kommandolinjebasert versjon av dette programmet utviklet. Ved å kjøre denne med de fire argumentene $filnavn$, x_1, x_2, y, blir strekkoden mellom (x_1,y) og (x_2,y) på bilde $filnavn$ dekodet og returnert som tekst. Denne versjonen av programmet blir brukt i løsningen til begge problemstillingene (se sammendrag av løsninger kapittel 1.3). Grunnen til dette er at vi i begge problemstillingene detekterer strekkodene ved hjelp av Matlab [15], og ikke ved å bruke musen som GUI-programmet fra kapittel 3.1 krever. En kjørbar JAR-fil av konsollprogrammet gjør at Matlab enkelt kan kjøre et systemkall til dette programmet for å utføre dekodingen.



Figur 3.8: Dekodingsprogram - Konsollprogrammet i bruk.

4 Deteksjon av én strekkode per bilde

Dette kapittelet presenterer en algoritme for automatisk detektering av én strekkode per bilde. For å vise en komplett løsning på del 1 av problemstillingen (figur 1.2) tar algoritmen i bruk konsollprogrammet som dekoder UPC-E-strekkoder (kapittel 3.4). Algoritmen er programmert i fjerdegenerasjons programmeringsspråket Matlab [15]. I tillegg til grunnpakken i Matlab er tilleggspakken "Image Processing Toolbox™" [16] brukt.



Figur 4.1: Deteksjon av én strekkode per bilde - Eksempel.

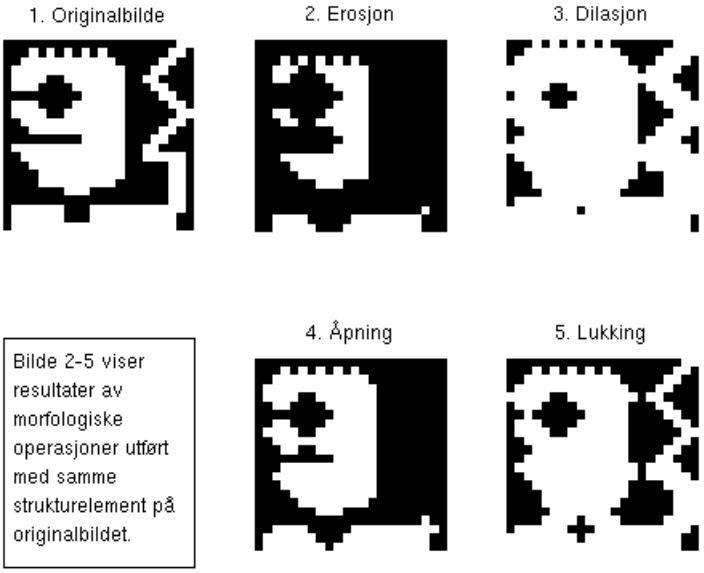
Figur 4.1 viser et eksempel på hva som skjer når Matlab-scriptet prosesserer et bilde. Hele algoritmen er delt opp i ni deler som alle er illustrert med hver sitt miniatyrbilde. Disse ni delene gjør alt fra innlesing av bildet fra fil, til detektering og dekoding av strekkoden i bildet.

4.1 Morfologisk bildebehandling

Algoritmen for å detektere én strekkode per bilde tar i bruk flere morfologiske operasjoner. Under følger en kort innføring i morfologiske operasjoner aktuelle for denne oppgaven. Boken "Hands-on Morphological Image Processing" [17] er kilden til det faglige innholdet i dette kapittelet.

Generelt

Morfologiske operasjoner blir brukt for å trekke ut bestemte strukturer og egenskaper fra bilder. Dette blir gjort ved å analysere bildene med et strukturelement. I denne oppgaven er morfologiske operasjoner kun utført på binære bilder[5]. En morfologisk operasjon på et binært bilde danner et nytt binært bilde. Dersom den morfologiske operasjonen gir et positivt resultat på en piksel i originalbildet, vil denne pikselen ha binærverdi "1" i resultatbildet.



Figur 4.2: Morfologiske operasjoner.

Figur 4.2 viser resultatene av å utføre de morfologiske operasjonene erosjon, dilasjon, åpning og lukking på et binært bilde.

[5]Et binært bilde er et digitalt bilde som bare har to mulige verdier for hver piksel.

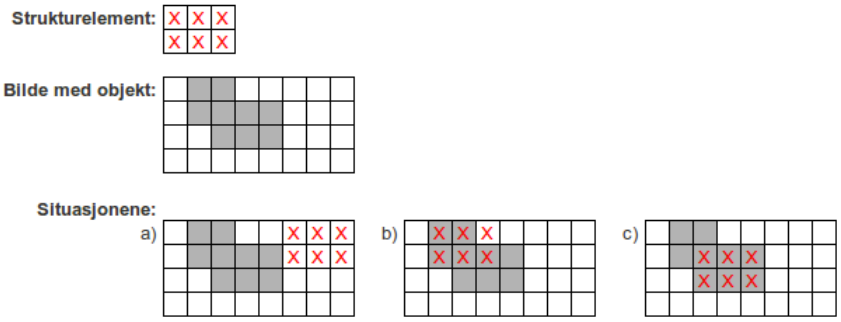
Strukturelement

Et strukturelements form og størrelse er definert av en matrise av piksler. Under morfologiske operasjoner blir strukturelementet ført over et bilde. I hver av posisjonene vil en av de tre følgende situasjonene forekomme:

a) Strukturelementet overlapper ikke objektet.

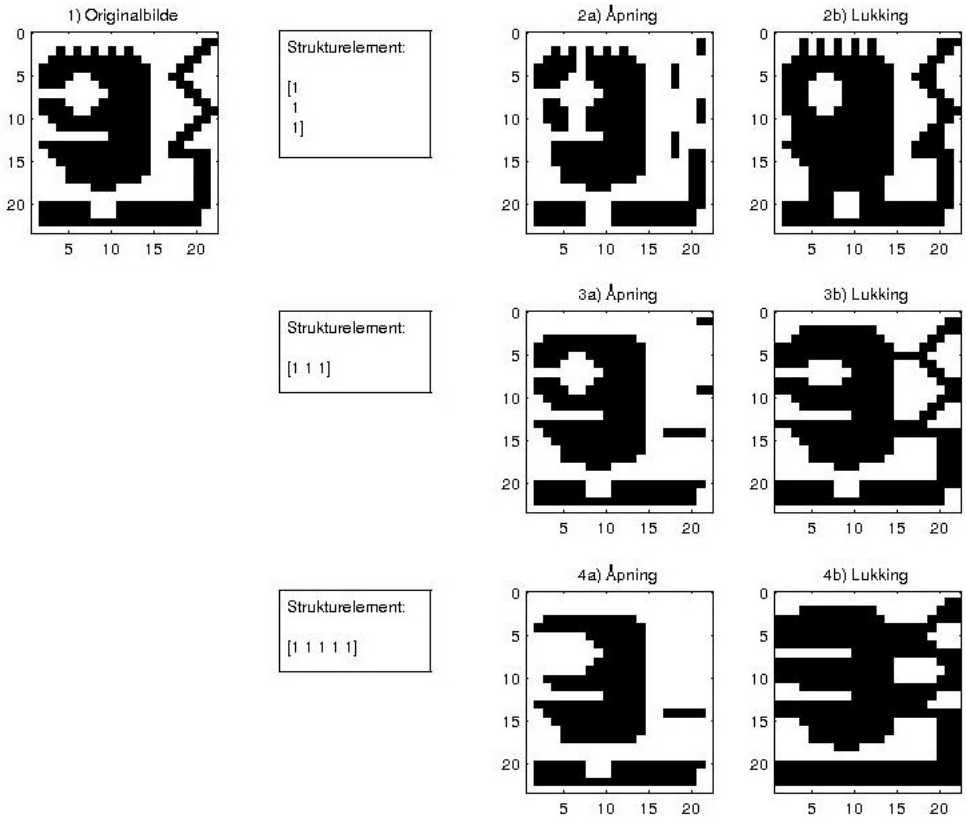
b) Strukturelementet overlapper delvis objektet.

c) Strukturelementet ligger inne i objektet.



Figur 4.3: Morfologiske strukturelementer.

Hvert strukturelement har en piksel som er definert som origo. Et strukturelement formet som et kvadrat, der dimensjonen er oddetall, vil ha origo midt i sentrum.



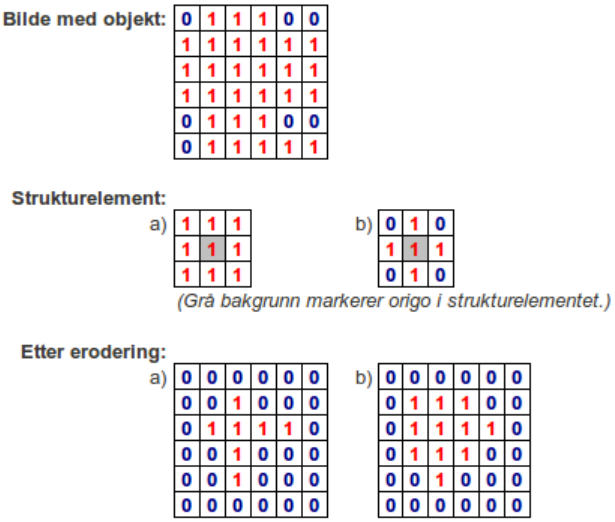
Figur 4.4: Morfologiske operasjoner med diverse strukturelementer.

Som figur 4.4 viser er det strukturelementets form og størrelse som avgjør hva de forskjellige morfologiske operasjonene gjør med originalbildet.

Morfologisk erosjon

Erosjon og dilasjon er de grunnleggende morfologiske operatorene. Alle andre morfologiske operasjoner kan uttrykkes ved hjelp av disse to. Erodering av et binært bilde vil krympe objektene i bildet.

Dersom strukturelementet passer i posisjon (x,y) i bildet vil bit på plasseringen (x,y) få verdi "1" i resultatbildet. Et strukturelement som har origo plassert i (x,y) passer i denne posisjonen dersom alle av strukturelementets pikselverdier $\neq 0$ svarer til pikselverdi $\neq 0$ i bildet.

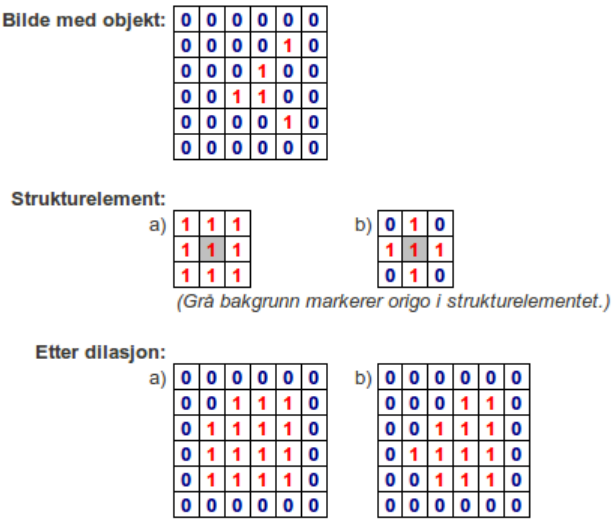


Figur 4.5: Morfologisk erosjon.

Morfologisk dilasjon

Erosjon og dilasjon er de grunnleggende morfologiske operatorene. Alle andre morfologiske operasjoner kan uttrykkes ved hjelp av disse to. Dilasjon av et binært bilde vil utvide objektene i bildet.

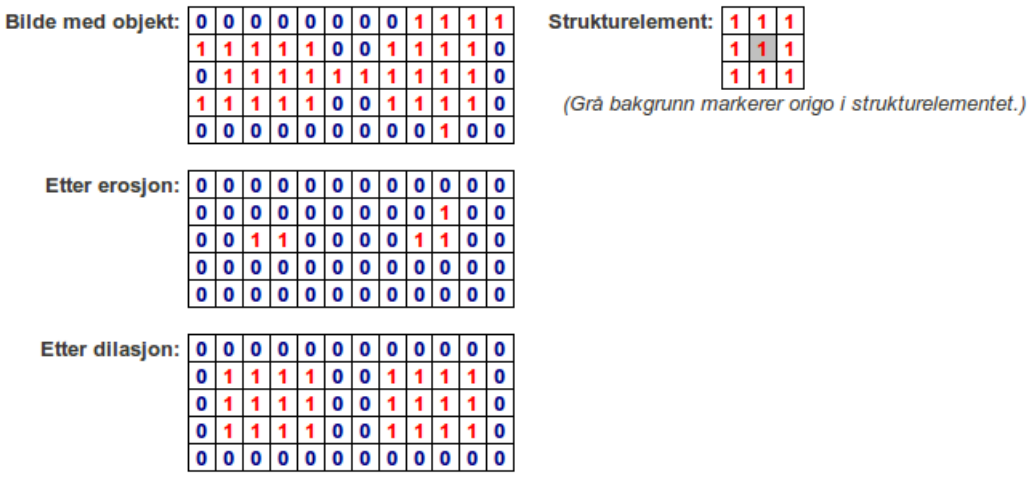
Dersom strukturelementet passer i posisjon (x,y) i bildet vil bit på plasseringen (x,y) få verdi "1" i resultatbildet. Et strukturelement som har origo plassert i (x,y) passer i denne posisjonen dersom minst et av strukturelementets pikselverdier $\neq 0$ svarer til pikselverdi $\neq 0$ i bildet.



Figur 4.6: Morfologisk dilasjon.

Morfologisk åpning

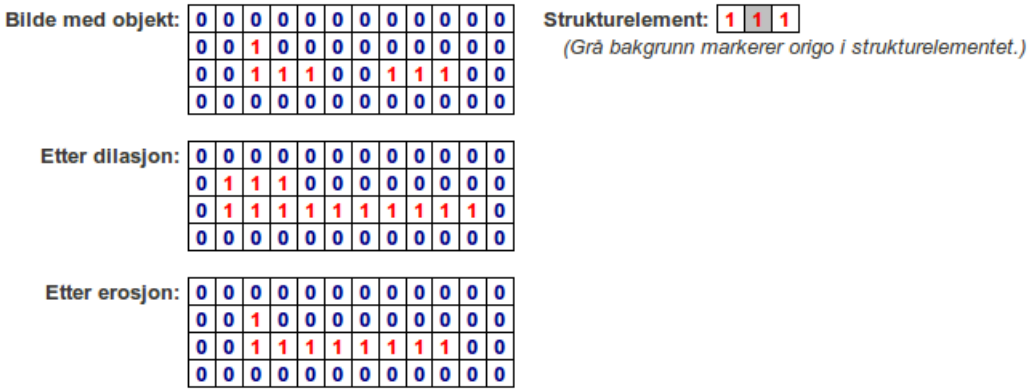
Morfologisk åpning av et bilde vil først erodere bildet, for så å dilatere resultatet. Erosjonen krymper alle strukturer i bildet, i tillegg til å fjerne strukturene som ikke er store nok til å kunne inneholde strukturelementet. Dilasjonen utvider så strukturen(e) som overlevde erosjonen.



Figur 4.7: Morfologisk åpning.

Morfologisk lukking

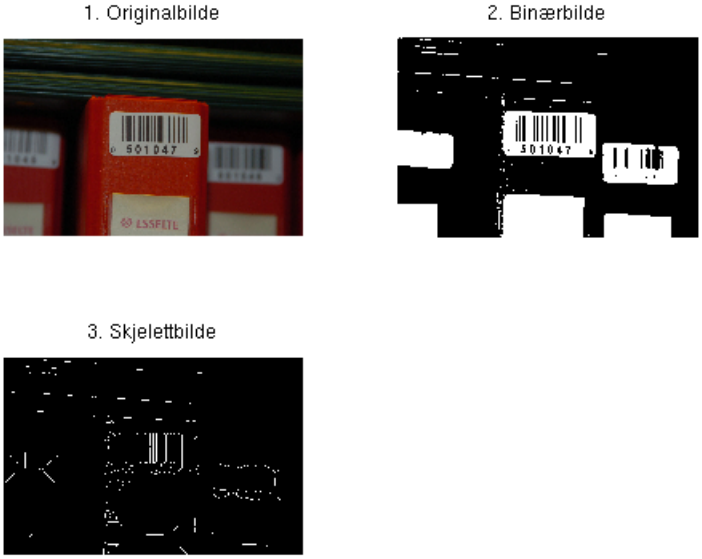
Morfologisk lukking av et bilde vil først dilatere bildet, for så å erodere resultatet. Dilasjonen vil utvide strukturen(e) i bildet, i tillegg til å fylle igjen hull. Ved å erodere resultatet vil strukturen(e) få omtrent samme størrelse igjen, men uten eventuelle hull som var før dilasjonen ble utført. Situasjoner der strukturer ble smeltet sammen etter dilasjonen vil ikke bli adskilt av erosjonen.



Figur 4.8: Morfologisk lukking.

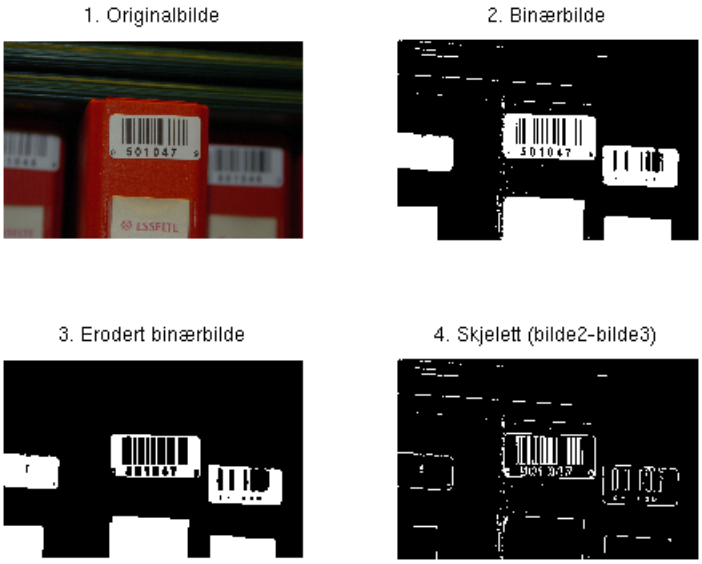
Morfologiske skjelett

Ved hjelp av skjelettering blir regioner i et binært bilde redusert, slik at kun skjelettet av de opprinnelige regionene står igjen (se figur 4.9).



Figur 4.9: Morfologisk skjelett.

I vår algoritme er det ønskelig å bevare strekkodens struktur og fjerne urelevant støy. For å gjøre dette er det tatt utgangspunkt i en metode presentert i "Barcode Localization Using a Bottom Hat Filter" [18]. Ved å subtrahere erodert bilde fra originalbilde vil omrisset av regionene på det originale bildet stå igjen (se figur 4.10).



Figur 4.10: Morfologisk omriss.

4.2 Algoritmen for detektering av én strekkode per bilde

Dette delkapittelet forklarer hva som skjer når Matlab-scriptet blir kjørt på et bilde.

4.2.1 Algoritmens hovedpunkter

a) Les inn bildet i Matlab.

b) Bildet blir skalert slik at den minste aksen er akkurat 480 piksler.

c) RGB-bildet blir konvertert til et gråtonebilde.

d) Gråtonebildet blir så omgjort til et binærbilde.

e) Et morfologisk omriss av binærbildet blir generert.

f) Morfologisk åpning av bildet for å bevare loddrette streker.

g) Morfologisk lukking vil sette sammen de loddrette strekene.

h) En blå strek blir tegnet over strukturen i bildet som mest sannsynlig er en strekkode.

i) Konsollprogram fra kapittel 3.4 dekoder strekkoden.

Figur 4.11: Deteksjon av én strekkode per bilde - Oversikt.

4.2.2 Detaljer i hvert hovedpunkt

Dette delkapittelet går gjennom hele Matlab-koden. Et Matlab-tillegg [19] til LaTeX er brukt for å inkludere Matlab-kode i dette dokumentet.

a) Lese inn bildet fra fil

```
1      iOriginal  = imread(path);
```

b) Skalering av bilde

```
1      if (size(iOriginal,2) > size(iOriginal,1))
2          scale = 480/size(iOriginal,1);
3      else
4          scale = 480/size(iOriginal,2);
5      end
6      iResized = imresize(iOriginal, scale);
```

I figur 4.11 blir originalbildet på 2464x1632 piksler skalert ned til 725x480 piksler. Selve deteksjonen i algoritmen blir videre utført på den skalerte versjonen, i stedet for det store originalbildet. Denne endringen utgjør ingen merkbar forskjell i algoritmens nøyaktighet. Algoritmens prosesseringstid per bilde blir derimot redusert fra ∼2.3 til ∼1.8 sekunder[6].

c) Konvertere RGB-bilde til gråtonebilde

```
1      iGrayscale = rgb2gray(iResized);
```

d) Fra gråtonebilde til binærbilde

```
1      level = graythresh(iGrayscale);
2      iGraythresh = im2bw(iGrayscale, level);
```

Først blir en grenseverdi for terskling mellom svart og hvitt i bildet beregnet. Denne verdien blir så brukt for å omgjøre bildet til et binærbilde.

e) Morfologisk omriss

```
1      se = strel('square',5);
2      iEroded = imerode(iGraythresh,se);
3      iSkeleton = imsubtract(iGraythresh, iEroded);
```

Først blir binærbildet erodert med et 5 piksler bredt kvadrat som strukturelement. Deretter blir dette eroderte bildet trukket fra binærbildet for å danne det morfologiske omrisset. Denne operasjonen vil bevare kanter i bildet. Figur 4.10 viser et eksempel på morfologiske omriss.

f) Morfologisk åpning

```
1      se = strel('line', 17, 90);
2      iOpen = imopen(iSkeleton, se);
```

[6]Spesifikasjoner på testmaskin: Matlab versjon = 7.11.0 (R2010b), OS = Ubuntu 11.04 32bit, CPU = Intel Core 2 Duo T9500 @ 2.60GHz, Minne = 3.0 GiB

Morfologisk åpning på bildet blir utført med en vertikal strek som er 17 piksler høy og 1 piksel bred som strukturelement. Dette vil bevare strekkodens vertikale streker, og alle andre strukturer i bildet som er store til å romme strukturelementet. De strukturene strukturelementet ikke passer inni vil forsvinne.

g) Morfologisk lukking

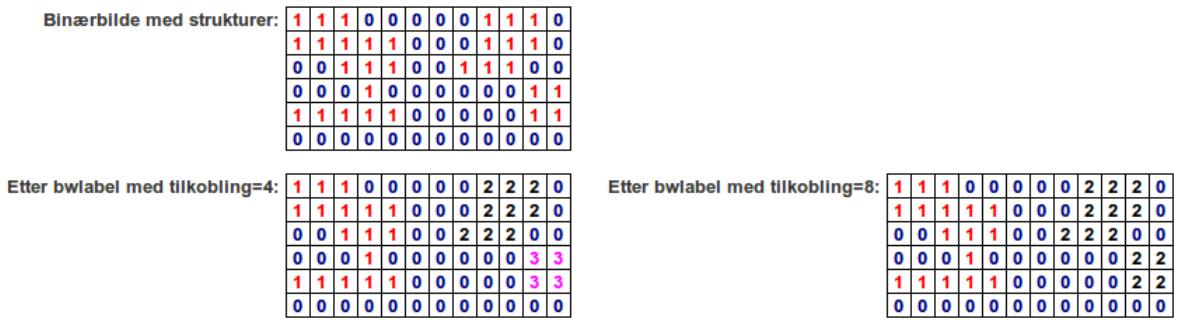
```
1       se = strel('rectangle',[1,20]);
2       iClose = imclose(iOpen,se);
```

Morfologisk lukking på bildet blir utført med et rektangel som er 1 piksel høy og 20 piksler bred som strukturelement. Dette blir gjort for å koble sammen de vertikale strekene i strekkoden. Som figur 4.11 viser blir det her dannet en stor struktur over strekkoden.

h) Finne strukturen som mest sannsynlig er strekkoden

```
1       % Show region with the biggest area where (width/height)<2 (if exist)
2       [labeled,numObjects] = bwlabel(iClose,4); % Label components
3
4       if (numObjects>0)
5           properties = {'Area', 'BoundingBox', 'Extrema', 'Centroid'};
6           linedata = regionprops(labeled, properties);
7           [sortedAreas sortedAreasIndex] = sort([linedata.Area], 'descend');
8
9           for j = 1:size(sortedAreas,2),
10              idx = sortedAreasIndex(j);
11              if( ((linedata(idx).BoundingBox(4)) / ...
                    (linedata(idx).BoundingBox(3))) < 2 )
12                  break;
13              end
14          end
```

De forskjellige ikke-sammenhengende strukturene i binærbildet blir først nummerert i linje 2. I metodekallet til `bwlabel` er argumentet for tilkobling satt til 4. Det vil si at de forskjellige bitene i bildet henger sammen dersom de enten står sammen vertikalt eller horisontalt. Ved å sette dette argumentet til 8, vil også piksler som står diagonalt tolket som samme struktur (se figur 4.12).



Figur 4.12: Deteksjon av én strekkode per bilde - Nummerering av strukturer i binærbilder.

Dersom bildet inneholder minimum en struktur (linje 4), blir diverse egenskaper (linje 5) for disse strukturene kalkulert (linje 6). Linje 7 sorterer så strukturene etter areal (største areal først i lista, minste areal sist). Videre går for-løkken fra linje 9 til 14 gjennom strukturene (her blir strukturen med størst areal tatt først). Dersom strukturens bredde dividert på høyde er mindre enn 2, blir for-løkken stoppet. Dette er strukturen som mest sannsynlig viser den største strekkoden i bildet.

i) Dekode strekkoden

```
leftX = max( linedata(idx).Extrema(8,1), linedata(idx).Extrema(7,1) ...
    ); % max(left-topX, left-bottomX)
leftX = leftX-10;
rightX = max( linedata(idx).Extrema(3,1), linedata(idx).Extrema(4,1) ...
    ); % max(right-topX, right-bottomX)
rightX = rightX+10;
Y = linedata(idx).Centroid(2);
leftX = leftX/scale;
rightX = rightX/scale;
Y = Y/scale;
```

Først blir koordinatene til strukturen funnet i punkt h) hentet ut.

```
[status,result] = system(['java -jar BarcodeDecoderConsole.jar "' ...
    folder '/' file '" "' num2str(round(leftX)) '" "' ...
    num2str(round(rightX)) '" "' num2str(round(Y)) '"']);
```

Til slutt kjører Matlab et systemkall til konsollprogrammet som dekoder strekkoder (kapittel 3.4). I dette systemkallet blir både lokasjon av bildet (original størrelse, ikke skalert versjon) og koordinatene til hvor i bildet strekkoden er lagt ved. Resultatet fra konsollprogrammet blir lagt inn i variabelen `result` og senere skrevet på bildet.

5 Resultater ved deteksjon og dekoding av én strekkode per bilde

Algoritmen presentert i kapittel 4 er testet både på bilder fra et av Vericos pågående prosjekt, i tillegg til noen utvalgte bilder fra tidligere prosjekter.

5.1 Prosjekt: Befaring av transformatorstasjoner og nettstasjoner

Målet med dette prosjektet er å lage et digitalt register over innholdet på en strømleverandørs transformatorstasjoner og nettstasjoner. Dette blir gjort ved at en av Vericos ansatte reiser på befaring på de forskjellige stasjonene. Her merkes diverse objekter med strekkoder før de fotograferes. Deretter blir den digitale oversikten generert ved å dekode strekkodene i de forskjellige bildene manuelt. Selve prosjektet har i dag over ti tusen bilder.

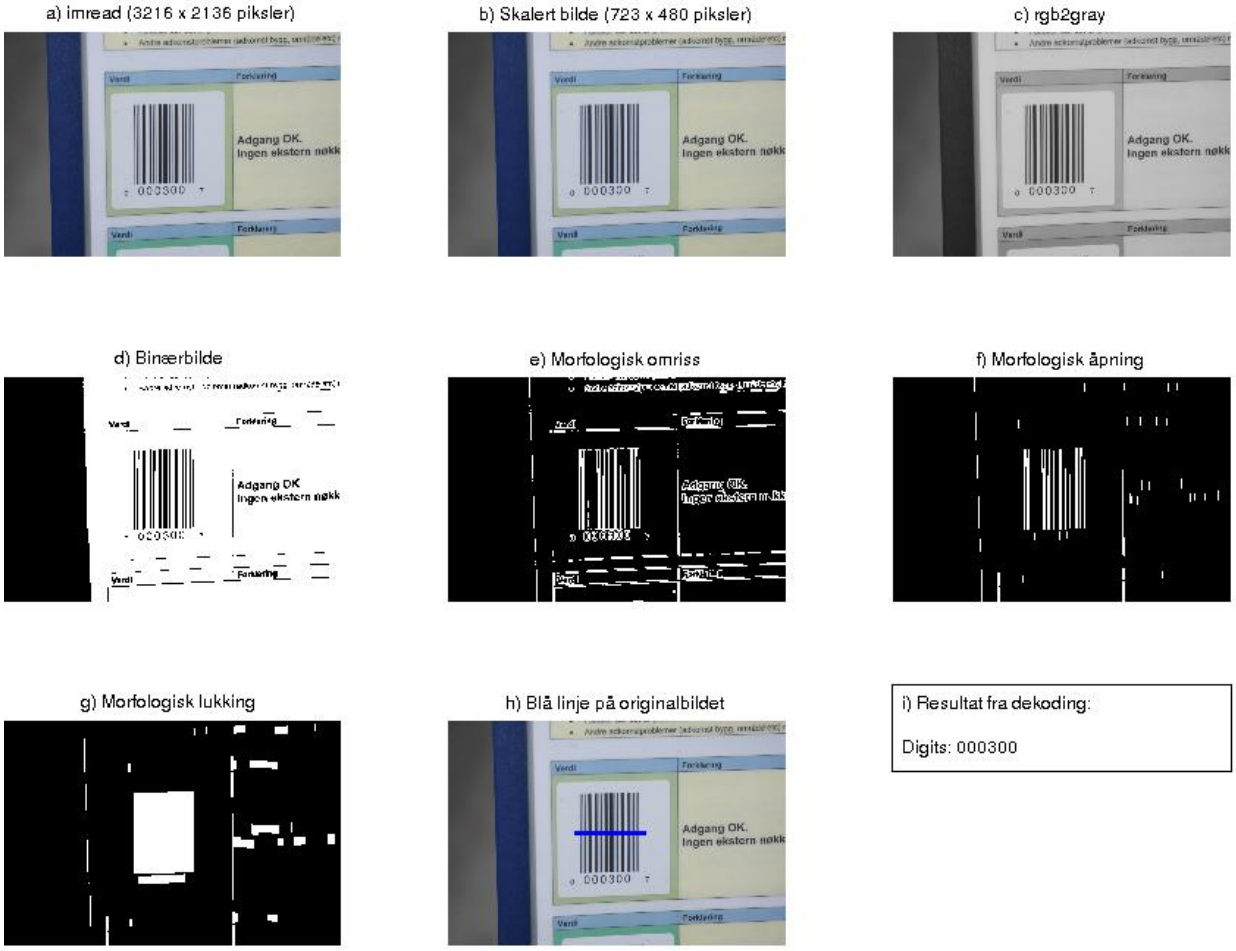
Resultater etter å ha kjørt et utvalg av prosjektets fotografier gjennom vår algoritme:

- Antall bilder: 1001
- Antall korrekte dekodinger: 985
- Antall mislykkede dekodinger: 16
- Treffprosent $\approx 98.4\%$
- Total prosesseringstid[7]: 1797 sekunder
- Prosesseringstid per bilde ≈ 1.8 sekunder

[7] Spesifikasjoner på testmaskin: Matlab versjon = 7.11.0 (R2010b), OS = Ubuntu 11.04 32bit, CPU = Intel Core 2 Duo T9500 @ 2.60GHz, Minne = 3.0 GiB

5.1.1 Eksempel på korrekt dekoding

Figur 5.1 viser resultatet til ett av de 985 bildene som er korrekt dekodet.



Figur 5.1: Deteksjon av én strekkode per bilde - Eksempel på korrekt dekoding.

Miniatyrbildene i figuren over viser at alle ni delene av algoritmen går etter planen. Strekkoden blir først detektert, før dekodingen gir resultatet "00300". Dette stemmer med strekkoden i fotografiet.

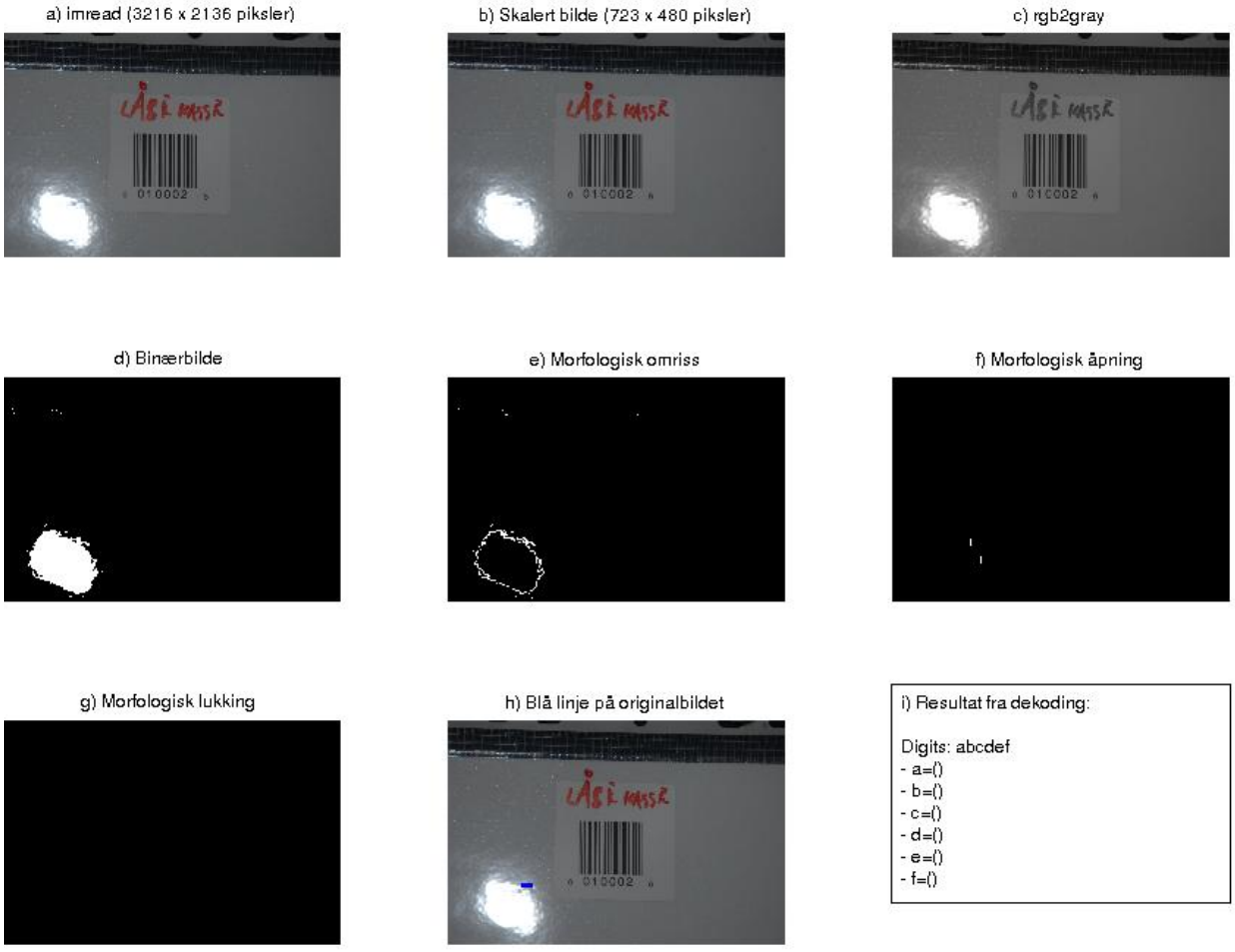
5.1.2 Mislykkede dekodinger

Av 1001 kjøringer var 16 tilfeller feilkodet. Tabell 5.1 oppsummerer årsaker til hvorfor disse ble dekodet feil.

Årsak	Antall bilder
Refleksjon av blits i bildet	9
Fotografert på skrått	4
Uskarpt bilde	2
Rusk på strekkoden	1

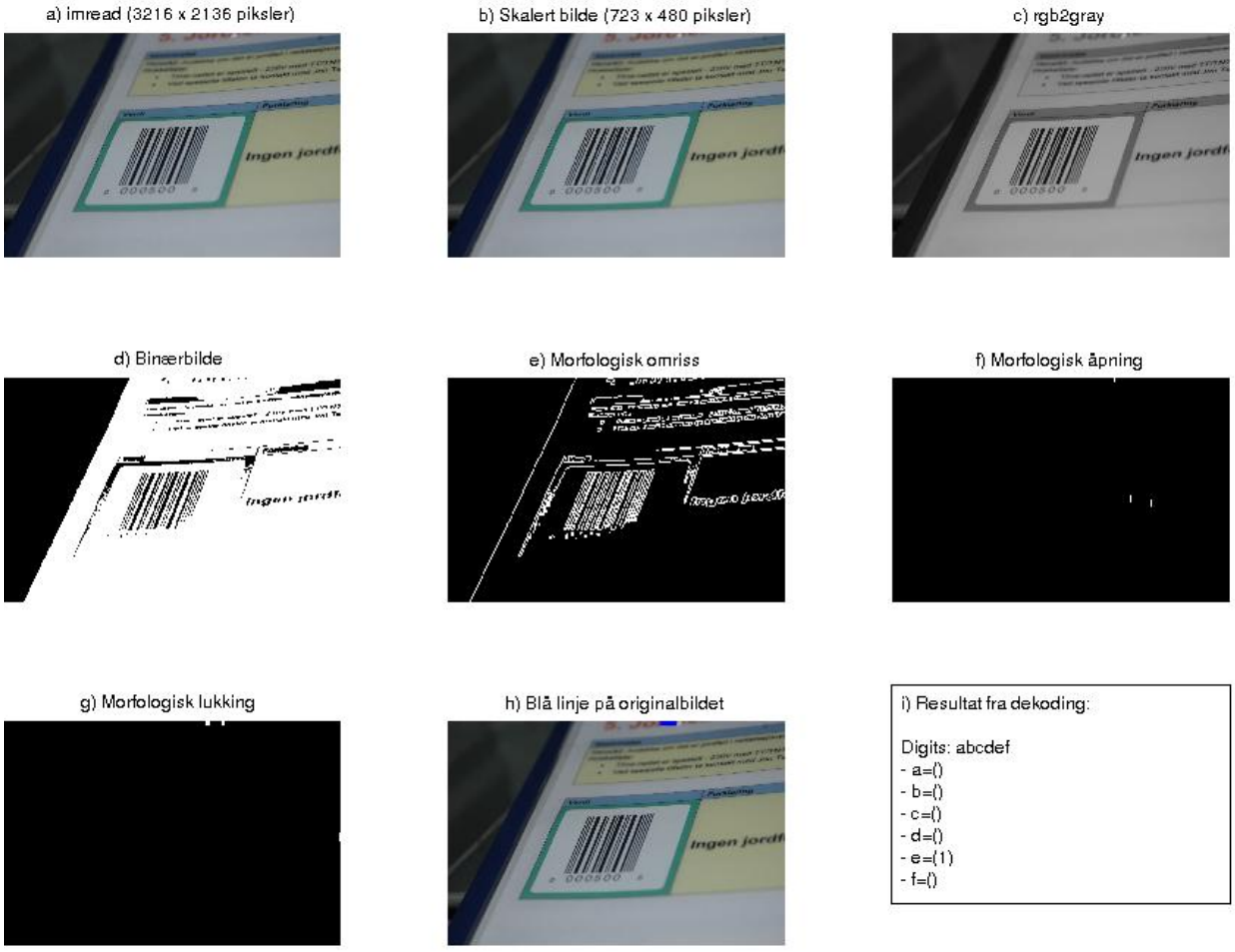
Tabell 5.1: Årsaker til mislykkede dekodinger i prosjektet (befaring av transformatorstasjoner og nettstasjoner).

Refleksjon av blits i bildet (figur 5.2): Etter at gråtonebildet er generert blir en terskelverdi for svart og hvitt regnet ut. Et binærbilde blir så generert ved hjelp av denne terskelverdien. I bilder lik det under er terskelverdien så lys at de hvite verdiene på strekkodens klistermerke blir tolket som svarte.



Figur 5.2: Deteksjon av én strekkode per bilde - Eksempel på mislykket dekoding, refleksjon av blits.

Fotografert på skrått (figur 5.3): I steg f) i figuren blir et strukturelement som er 1 piksel bredt og 17 piksler høyt brukt til morfologisk åpning av bildet. Strekkoden vil i dette tilfellet forsvinne på grunn av at de loddrette strekene på klistermerket ikke står vertikalt på fotografiet.



Figur 5.3: Deteksjon av én strekkode per bilde - Eksempel på mislykket dekoding, fotografert på skrått.

Uskarpt bilde (figur 5.4): I begge tilfellene hvor uklart bilde var årsaken for mislykket dekoding var det selve dekodingen (kapittel 3), og ikke detekteringen som feilet. Grunnen til dette er at strekkodene ikke er i fokus på fotografiene.

Figur 5.4: Deteksjon av én strekkode per bilde - Eksempel på mislykket dekoding, uskarpt bilde.

Rusk på strekkoden (figur 5.5): Den lille hvite flekken inni den røde sirkelen på bildet under var i dette tilfellet nok til at dekodingsprogrammet ikke fikk dekodet strekkoden.

Figur 5.5: Deteksjon av én strekkode per bilde - Eksempel på mislykket dekoding, rusk på strekkoden.

5.1.3 Hvordan forbedre treffprosenten

For å forbedre treffprosenten presenterer vi her to mulige utvidelser av algoritmen.

Retningsbestemt bildeåpning:
Algoritmen utfører en morfologisk åpning av bildet med et strukturelement som er 1 piksel bredt og 17 piksler høyt (figur 4.11 - steg f)). Dette gjør at strekkoder som ikke står vinkelrett på fotografiet vil forsvinne (figur 5.3). Ved å ta i bruk en metode presentert i [18] og [20] vil disse feilene bli rettet. Metoden går ut på å utføre morfologisk åpning av bildet 16 ganger. For hver av disse gangene blir strukturelementet rotert 22,5 grader (360/16). Resultatet med flest 1-ere er resultatet hvor strekene i strekkoden er bevart, dette blir brukt videre i algoritmen. Denne utvidelsen vil i dette prosjektet luke bort 4/1001 feil. Prosesseringstiden vil bli utvidet fra 1797 sekunder til:

$time = 1797sec$
$eachOpening = 0.2sec$
$extraOpenings = 15$
$pictures = 1001$
$= time + (eachOpening * extraOpenings * pictures) = 4800sec$

I tillegg må det legges til ekstra tid for å sammenlikne de 16 forskjellige resultatene per bilde.

Dekode i to ulike høyder:
Figur 5.5 viser hva litt rusk på strekkoden kan gjøre. Dersom algoritmen hadde prøvd å dekode strekkoden med to ulike høyder, som vist i figur 5.6, hadde vi i dette prosjektet fjernet 1/1001 feil.

Figur 5.6: Deteksjon av én strekkode per bilde - Dekode i to ulike høyder.

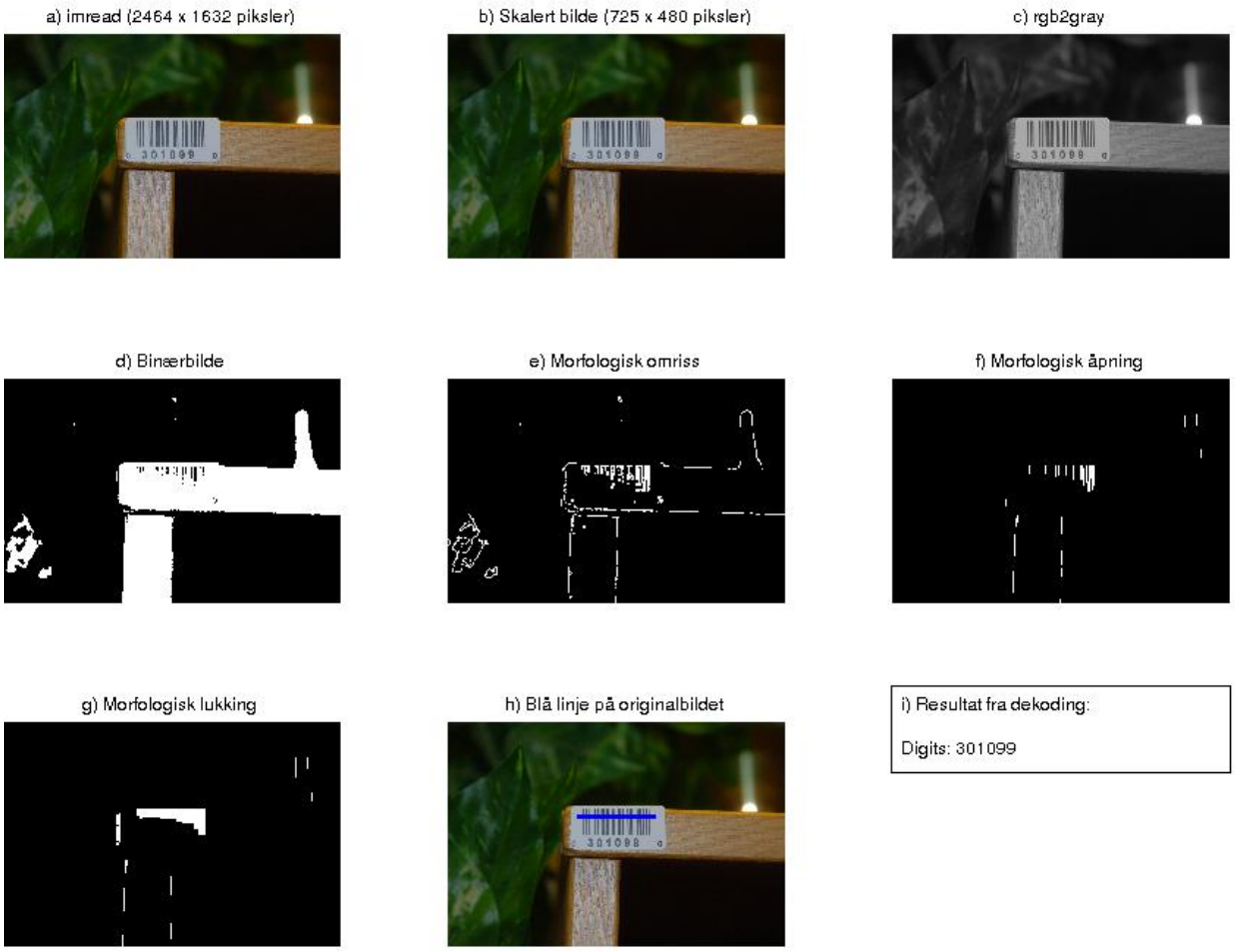
Prosesseringstiden vil bli utvidet fra 1797.1265 sekunder til:

$time = 1797.1265sec$
$eachDecoding = 0.65sec$
$pictures = 1001$
$= time + (eachDecoding * pictures) = 2448sec$

I tillegg må det legges til ekstra tid for å sammenlikne hver av bildenes to resultatene.

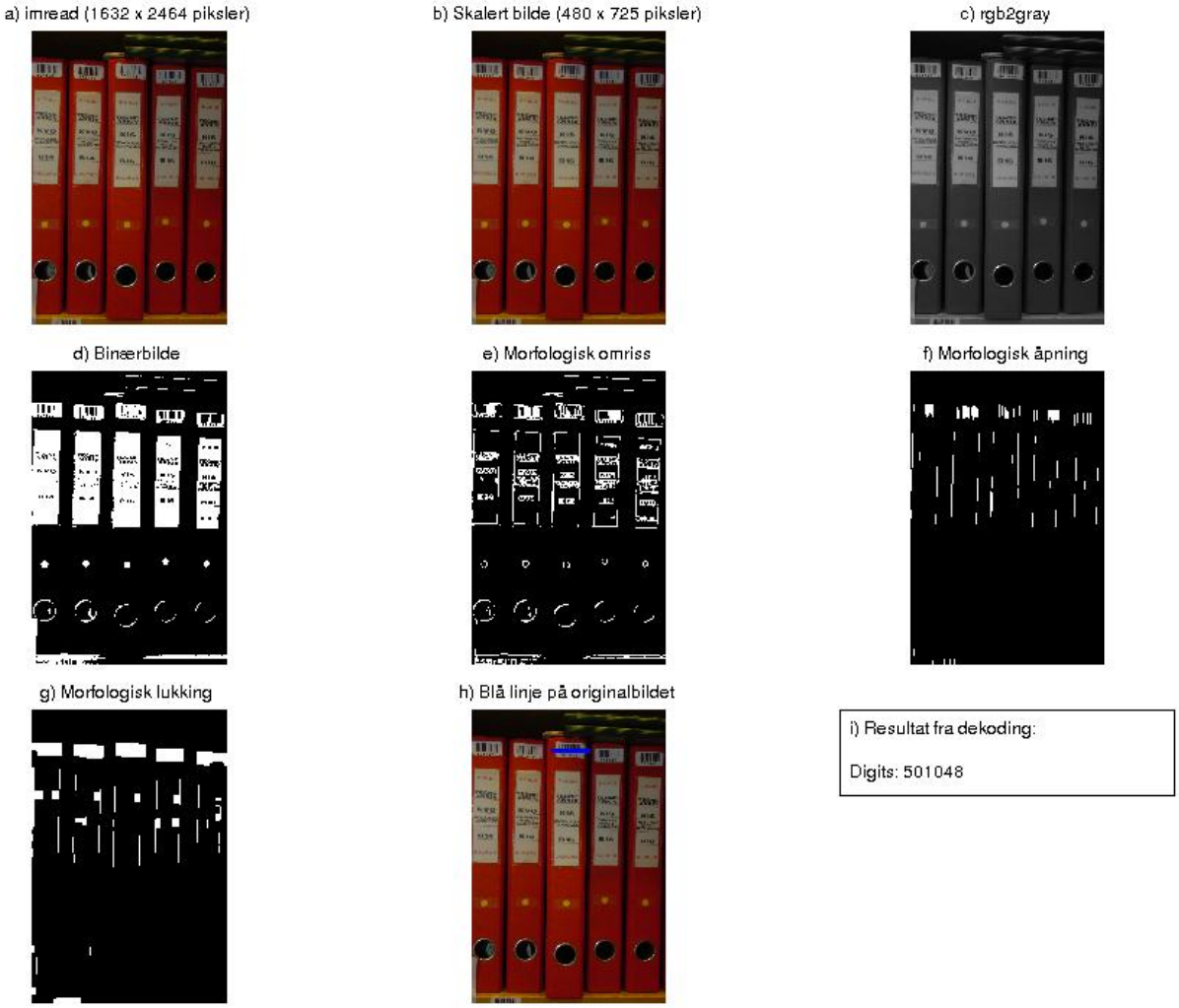
5.2 Eksempler som viser algoritmens robusthet

Man kunne være fristet til å basere deteksjonen av strekkoden på dens rektangulære form. Figur 5.7 viser tydelig hvorfor dette ikke er lurt. I dette bildet er store deler av strekkoden for lys i forhold til bildets terskelverdi mellom svart og hvit. Dette gjør at mye av strekkoden forsvinner i det bildet blir gjort binært (steg d)).



Figur 5.7: Deteksjon av én strekkode per bilde - Korrekt dekoding, refleksjon av blits.

Figur 5.8 viser at algoritmen også virker i tilfeller hvor strekkoden ikke tar opp like stor plass av fotografiet. Dette virker fordi logikken som i steg h) finner strekkoden ikke bruker noen grenser for å finne rett region, men størrelse på regionen i forhold til de andre regionene på bildet.



Figur 5.8: Deteksjon av én strekkode per bilde - Korrekt dekoding, robust.

6 Deteksjon av flere strekkoder per bilde

Dette kapittelet presenterer en algoritme som løser del 2 av problemstillingen (se figur 1.3). Algoritmen er, i likhet med algoritmen for dekoding av én strekkode per bilde (kapittel 4), laget i utviklingsmiljøet Matlab [15] med tilleggspakken "Image Processing Toolbox™" [16]. Figur 6.1 viser et eksempel på hva som skjer når Matlab-scriptet prosesserer et bilde.

Figur 6.1: Deteksjon av flere strekkoder per bilde - Eksempel.

6.1 Algoritmens hovedpunkter

a) Lese inn bildet i Matlab og definere strekkodenes fysiske størrelse.

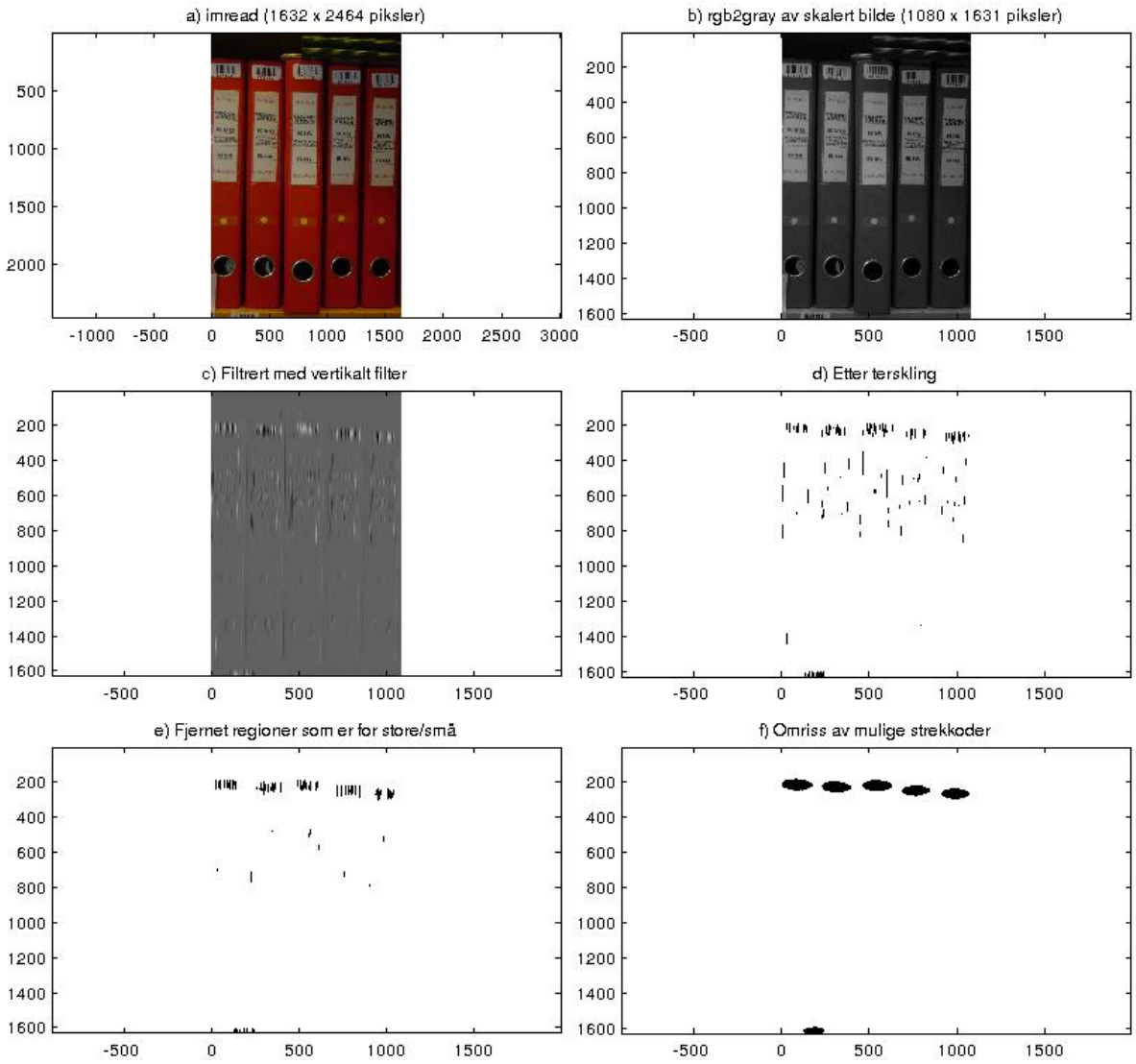
b) RGB-bildet blir skalert slik at den minste aksen er akkurat 1080 piksler og deretter konvertert til et gråtonebilde.

c) Bildet blir filtrert med et vertikalt filter.

d) Bildet blir omgjort til binærbilde.

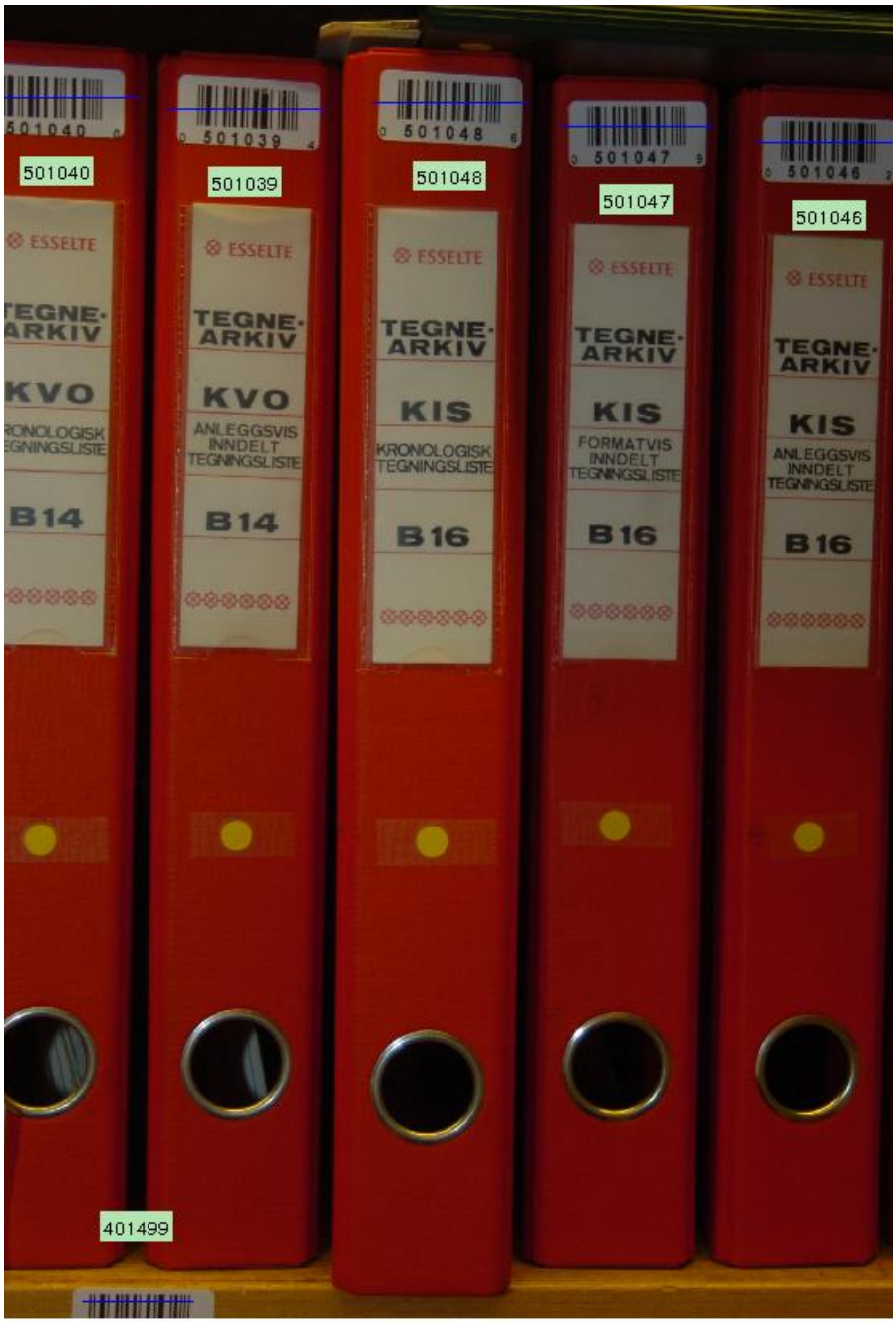
e) Regioner i bildet som er for store eller for små blir fjernet.

f) De gjenværende regionene blir koblet sammen for å danne omriss av mulige strekkoder.



Figur 6.2: Deteksjon av flere strekkoder per bilde - Oversikt (del 1 av 2).

g) Hver av de gjenværende regionene blir dekodet dersom de er store nok. Resultatet blir skrevet på originalbildet.



Figur 6.3: Deteksjon av flere strekkoder per bilde - Oversikt (del 2 av 2).

6.2 Detaljer i hvert hovedpunkt

M-code [19] er brukt for å inkludere Matlab-koden under.

a) Lese inn bildet fra fil

```
1          iOriginal  = imread(path);
```

Leser inn bildet fra filsystem.

```
1          [barcodeSizeX, barcodeSizeY] = ginput(2);
2          title('Click on left, and then right side of a barcode.');
3          barcodeWidth = sqrt(((barcodeSizeX(2)-barcodeSizeX(1))^2 + ...
               (barcodeSizeY(2)-barcodeSizeY(1))^2));
```

I utgangspunktet kan strekkodene være så store at de tar opp hele fotografiet, eller være så små at de såvidt er synlige. Algoritmen er bygd opp slik at brukeren må klikke to ganger på bildet før den går videre. Først på den ene siden av en av bildets strekkoder, deretter på den andre siden. Dette gir algoritmen nødvendig informasjon om hvor store strekkodene i bildet er. Treffprosenten vil øke mye når algoritmen allerede vet omtrent størrelse på strekkodene den skal detektere.

b) Skalering og konvertering til gråtonebilde

```
1          if (size(iOriginal,2) > size(iOriginal,1))
2              scale = 1080/size(iOriginal, 1);
3          else
4              scale = 1080/size(iOriginal, 2);
5          end
6          iResized = imresize(iOriginal, scale);
```

I eksempelet i figur 6.2 blir originalbildet på 1632x2464 piksler skalert ned til 1080x1631 piksler. Selve deteksjonen i algoritmen blir videre utført på den skalerte versjonen, i stedet for det store originalbildet. Denne endringen utgjør ingen merkbar forskjell i algoritmens nøyaktighet. Algoritmens prosesseringstid blir derimot forbedret med $\sim$1 sekund per bilde.

```
1          iGrayscale = rgb2gray(iResized);
```

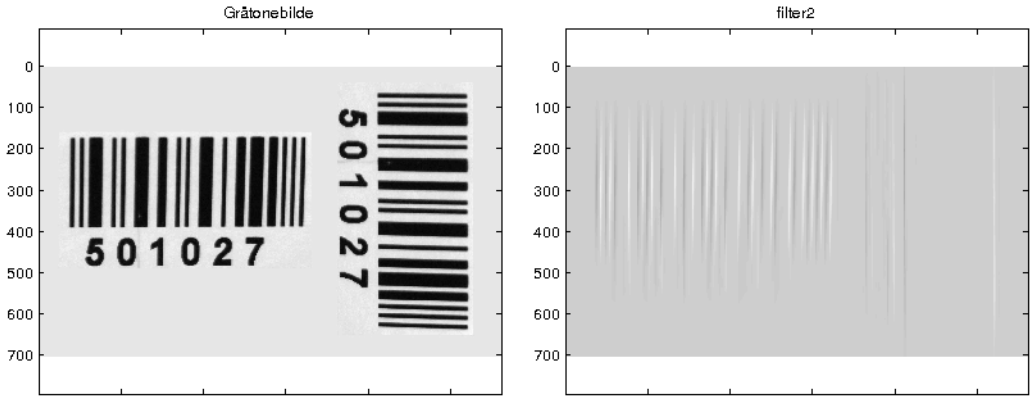
Koden over konverterer RGB-bildet til gråtonebilde.

c) Vertikal filtrering

```
1          bvert = [ones(round(barcodeWidth/3), 1),-ones(round(barcodeWidth/3), 1)];
2          iFilter2 = filter2(bvert, iGrayscale);
```

Masken `bvert` representerer et todimensjonalt FIR-filter[8] [21]. `filter2`-kommandoen roterer først filtermatrisen `bvert` 180 grader for å danne en konvolusjonskjerne. Deretter kaller den `conv2`, en todimensjonal konvolusjonsfunksjon, som utfører filtreringen. I formelen for den todimensjonale konvolusjonen under er F bildematrisen og G konvolusjonskjernen.

$$H[x][y] = \sum_{j=0}^{height-1} \sum_{i=0}^{width-1} F[x+i][y+j]G[i][j]$$



Figur 6.4: Deteksjon av flere strekkoder per bilde - Lineær filtrering.

Figur 6.4 viser at de vertikale strekene i strekkoden blir bevart ved å utføre en vertikal filtrering, mens de horisontale forsvinner.

Et lite eksempel: I utskriften under er Y resultatet av å filtrere matrise X med maske B:

```
1  X =   0     0   200     0
2        0     0   200     0
3        0     0   200     0
4        0     0   200     0
5       50    50    50    50
6        0     0   200     0
7
8  B =   1    -1
9        1    -1
10       1    -1
11
12 >> Y = filter2(B,X)
13 Y =   0  -400   400     0
14       0  -600   600     0
15       0  -600   600     0
16       0  -400   400    50
17       0  -400   400    50
18       0  -200   200    50
```

Utskriften over viser oss de samme tendensene som figur 6.4. Verdiene som viser vertikale strukturer blir bevart, mens horisontale strukturer blir visket ut.

[8] FIR - Finite Inpulse Response

d) Fra gråtonebilde til binærbilde

```
thresholdPercent = 2;
```

```
        iFilter2Sorted = sort(abs(iFilter2(:)), 'descend');
        threshold = ...
            iFilter2Sorted(ceil(thresholdPercent*numel(iFilter2Sorted)/100));
        iBinary = (abs(iFilter2) > threshold);
```

Først blir en grenseverdi for terskling mellom svart og hvitt i bildet beregnet. Denne verdien blir så brukt for å omgjøre bildet til et binærbilde. Alle gråtoneverdiene i bildet som er mørkere enn terskelverdien blir satt til svart, mens de lysere blir satt til hvitt. Terskelverdien er satt så lavt at kun 2 prosent av pikslene i bildet blir svarte etter terskling, mens de resterende 98 prosentene blir hvite.

e) Fjerne regioner i bildet som er for store eller for små

```
properties = {'Area', 'Extrema', 'MinorAxisLength', 'MajorAxisLength'};
```

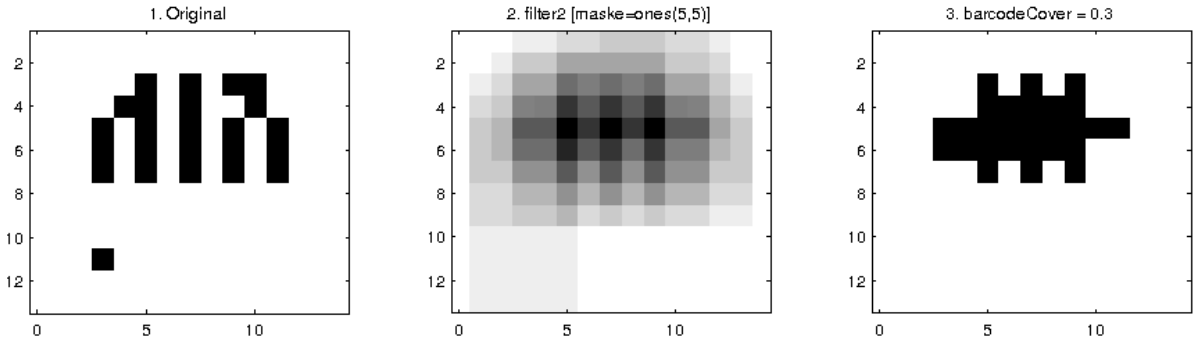
```
        barcodeLimMin = struct('Area', round(barcodeWidth/6));
        barcodeLimMax = struct('MajorAxisLength', round(barcodeWidth+10), ...
            'MinorAxisLength', round(barcodeWidth-10));
        iCleanedBinary = iBinary;
        cc = bwconncomp(iCleanedBinary, 4);
        stat  = regionprops(cc, properties);
        for j=1:cc.NumObjects
            if stat(j).Area < barcodeLimMin.Area
                iCleanedBinary(cc.PixelIdxList{j}) = 0;
            end
            if stat(j).MajorAxisLength > barcodeLimMax.MajorAxisLength
                iCleanedBinary(cc.PixelIdxList{j}) = 0;
            end
            if stat(j).MinorAxisLength > barcodeLimMax.MinorAxisLength
                iCleanedBinary(cc.PixelIdxList{j}) = 0;
            end
        end
```

Først blir noen grenseverdier for bildet kalkulert ut ifra hva algoritmen fikk inn av informasjon om strekkodestørrelsene i punkt a). Disse grensene definerer minste areal en region kan ha og hvor lang regionen maksimalt kan være i høyde- og bredderetning. Regioner i bildet som ikke er innenfor disse grensene blir fjernet.

f) Danne omriss av mulige strekkoder

```
1  barcodeCover = 0.17;
```

```
1          barcodeArea = ones(round(barcodeWidth/2.5), round(barcodeWidth));
2          iBarcode = ...
               filter2(barcodeArea/sum(barcodeArea(:)),double(iCleanedBinary));
3          iBarcode = (iBarcode > barcodeCover);
```



Figur 6.5: Deteksjon av flere strekkoder per bilde - Omriss av regioner.

Figur 6.5 illustrerer hva kodelinjene over figuren gjør med et binærbilde. Først blir hele bildet filtrert av et todimensjonalt FIR-filter. En matrise bestående av kun 1-ere blir brukt som maske i denne filtreringen. Matrisens størrelse er bestemt ut ifra hva algoritmen fikk av informasjon om strekkodenes størrelse fra brukeren i punkt a). Til slutt blir resultatet av filtreringen omgjort til binærbilde ved hjelp av terskelverdien `barcodeCover`.

g) Dekoding av regioner

```
1          cc = bwconncomp(iBarcode, 4);
2          stat  = regionprops(cc, properties);
3          hold on;
4          for j=1:cc.NumObjects
5              leftX = max( stat(j).Extrema(8,1), stat(j).Extrema(7,1) ); % ...
                   max(left-topX, left-bottomX)
6              leftX = leftX-5;
7              rightX = max( stat(j).Extrema(3,1), stat(j).Extrema(4,1) ); % ...
                   max(right-topX, right-bottomX)
8              rightX = rightX+5;
9              topY = max( stat(j).Extrema(1,2), stat(j).Extrema(2,2) );
10             bottomY = max( stat(j).Extrema(5,2), stat(j).Extrema(6,2) );
11             leftX = leftX/scale;
12             rightX = rightX/scale;
13             topY = topY/scale;
14             bottomY = bottomY/scale;
15
16             topY = max(ceil(topY),1);
17             bottomY = max(ceil(bottomY),1);
18             Y = topY + (bottomY-topY)/2;
19
20             if ((rightX-leftX) > 0.7*(size(barcodeArea,2)/scale)) && ...
                   ((bottomY-topY) > 0.7*(size(barcodeArea,1)/scale))
21                 diff = ((size(barcodeArea,2)+10)/scale) - (rightX-leftX);
22                 if diff>0
23                     leftX = leftX-(diff/2);
```

```
24                      rightX = rightX+(diff/2);
25                  end
26                  leftX = max(ceil(leftX),1);
27                  rightX = min(floor(rightX),size(iOriginal,2)-1);
28                  plot([leftX,rightX], [Y,Y],'-','LineWidth',1,...
29                          'MarkerEdgeColor','r',...
30                          'MarkerFaceColor','g',...
31                          'MarkerSize',5)
32                  [status,result] = system(['java -jar ...
                        BarcodeDecoderConsole.jar "' folder '/' file '" "' ...
                        num2str(leftX) '" "' num2str(rightX) '" "' ...
                        num2str(round(Y)) '"']);
33                  if Y>(size(iOriginal,1)/2)
34                      text(leftX+((rightX-leftX)/2),...
35                          Y-((1.5*size(barcodeArea,1))/scale),...
36                          result,...
37                          'BackgroundColor',[.7 .9 .7],...
38                          'VerticalAlignment','bottom',...
39                          'HorizontalAlignment','center');
40                  else
41                      text(leftX+((rightX-leftX)/2),...
42                          Y+((1.5*size(barcodeArea,1))/scale), ...
43                          result, ...
44                          'BackgroundColor',[.7 .9 .7], ...
45                          'VerticalAlignment','top', ...
46                          'HorizontalAlignment','center');
47                  end
48              end
49          end
50          hold off;
```

Regionene blir kun dekodet ved hjelp av konsollprogrammet fra kapittel 3.4 dersom de er store nok (linje 20). Dekodingen blir gjort på originalbildet, og ikke den skalerte versjonen. Resultatet blir til slutt skrevet på bildet (linje 33 til 47).

7 Resultater ved deteksjon og dekoding av flere strekkoder per bilde

Algoritmen fra kapittel 6 er testet på noen utvalgte bilder gitt av Verico. Resultatene fra disse bildene er som følger:

- Antall bilder: 18
- Antall strekkoder: 112
  - Antall loddrette strekkoder: 4 (se figur 7.1)
  - Antall vannrette strekkoder: 108 (se figur 7.1)
- Antall av strekkodene algoritmen detektet: 94
  - Korrekte dekodinger: 76/94
  - Mislykkede dekodinger: 18/94
- Antall feil detekteringer: 11
- Prosesseringstid per bilde[9] = 2 + (0.4 · antallDekodinger) sekunder

Vannrette og loddrette strekkoder

Figur 7.1: Deteksjon av flere strekkoder per bilde - Et bilde med en vannrett og en loddrett strekkode.

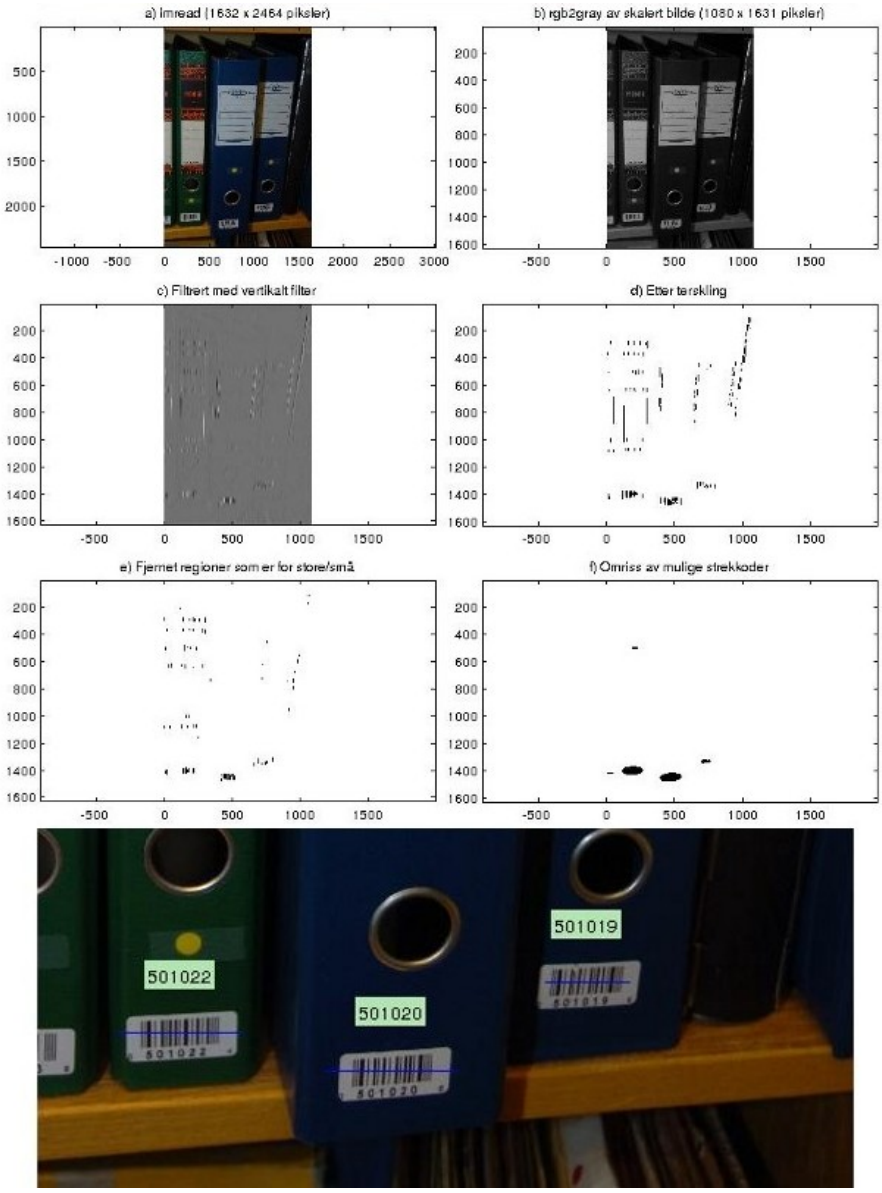
Under algoritmens vertikale filtrering i punkt c) forsvinner bildets vertikale streker, noe vi tydelig ser i figur 6.4. Dette er årsaken til at kun vannrette strekkoder blir detektert og dekodet.

[9] Spesifikasjoner på testmaskin: Matlab versjon = 7.11.0 (R2010b), OS = Ubuntu 11.04 32bit, CPU = Intel Core 2 Duo T9500 @ 2.60GHz, Minne = 3.0 GiB

Eksempler på korrekt dekoding

Figur 7.2: Deteksjon av flere strekkoder per bilde - Eksempel på korrekt dekoding.

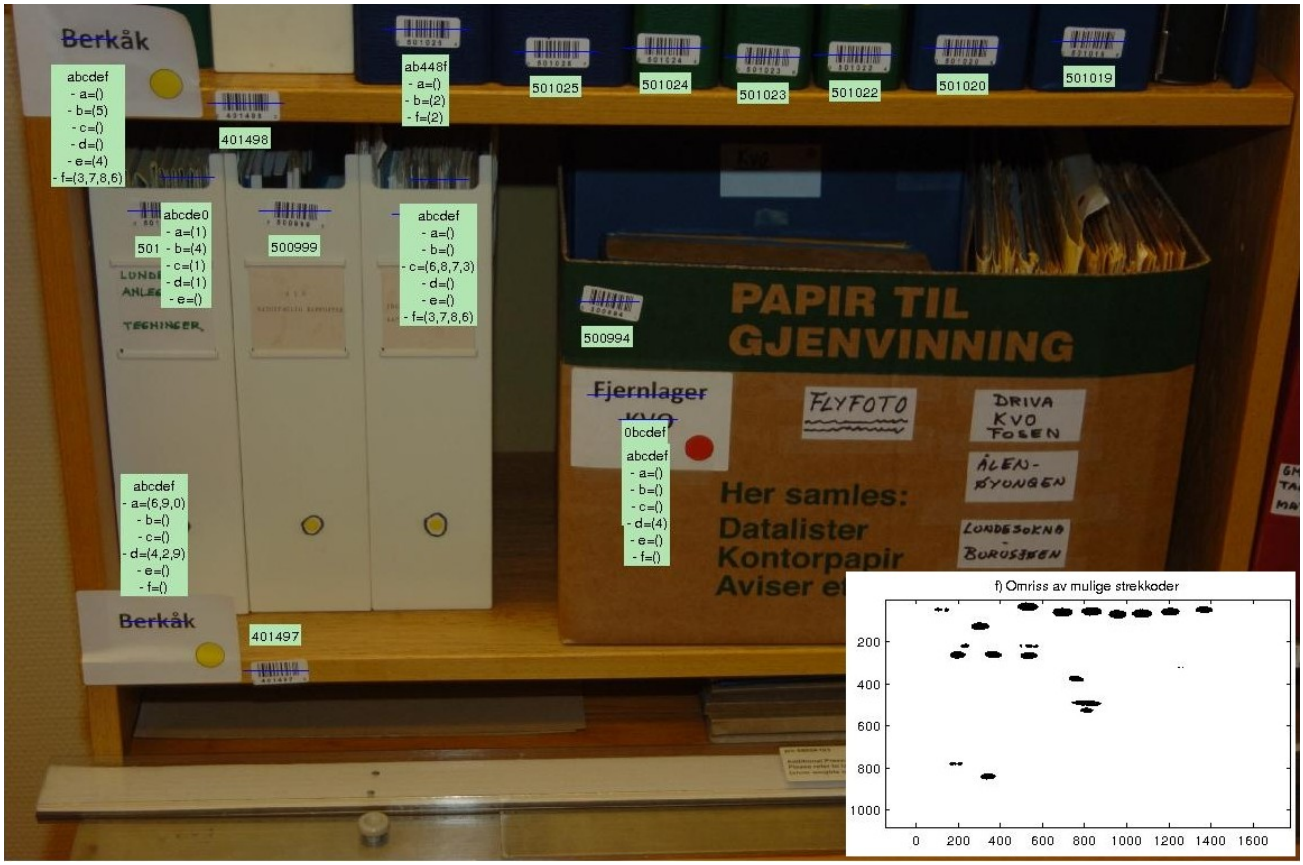
Bildet i figur 7.2 inneholder 14 vannrette og 2 loddrette strekkoder. Algoritmen detekterer og dekoder i dette eksempelet 15 områder. Av disse er 14 av dem strekkoder, hvor alle blir dekodet korrekt.



Figur 7.3: Deteksjon av flere strekkoder per bilde - Bilde fotografert på skrått.

Bildet over viser at algoritmen detekterer og dekoder samtlige av fotografiets tre strekkoder, selv om strekkodene ikke står helt vinkelrett på bildet.

Algoritmens nøyaktighet - Et eksempel



Figur 7.4: Deteksjon av flere strekkoder per bilde - Algoritmens nøyaktighet.

Som binærbildet nederst til høyre i figur 7.4 viser, har algoritmen tegnet omriss av totalt 23 regioner. I neste steg (kapittel 6.2 - punkt g) blir 19 av disse 23 regionene dekodet (13 av disse 19 er strekkoder). Det er fullt mulig å gjøre algoritmen enda mer presis, og dermed slippe å dekode så mange regioner som ikke er strekkoder. Problemet med dette er at vi da også risikerer å miste noen av regionene som er strekkoder i bildet. I denne oppgaven er det viktigste å finne flest mulig av strekkodene, og da godta at noen av regionene som er dekodet kanskje ikke inneholder strekkoder.

Uklare bilder

Figur 7.5: Deteksjon av flere strekkoder per bilde - Uklart bilde.

Bildet over viser at algoritmen greier å detektere og dekode mange strekkoder, selv når fotografiet er så uklart som dette.

Skygge på bilder

Figur 7.6: Deteksjon av flere strekkoder per bilde - Skygge på bilde.

I figur 7.6 detekterer algoritmen alle strekkodene korrekt. Men siden den ene strekkoden er i delvis skygge blir terskelverdien mellom svart og hvitt satt feil, og strekkoden blir derfor ikke dekodet korrekt.

8 Konklusjon

Tre helt nye algoritmer for deteksjon og dekoding av UPC-E-strekkoder i digitale bilder er utviklet. Satt sammen presenterer disse tre algoritmene to fullstendige løsninger på to problemstillinger. Den ene detekterer og dekoder én strekkode per bilde, mens den andre detekterer og dekoder flere strekkoder per bilde.

Dekodingsprogram

Felles for begge delene av løsningen er at de benytter seg av dekodingsprogrammet fra kapittel 3. Dette programmet dekoder UPC-E-strekkoder uten problem så lenge strekkodene er hele og i fokus på fotografiene. I situasjoner hvor strekkodene er uklare, eller av andre grunner ikke lar seg dekode, finner algoritmen mulige treff på sifrene som ikke lar seg dekode. Bildene Verico jobber med er som regel i god nok kvalitet til at algoritmen utviklet i dette prosjektet greier å dekode strekkodene. De har derfor allerede implementert dekodingsprogrammet i sine verktøy. For Verico er det viktig at strekkodene enten blir dekodet korrekt, eller ikke dekodet i det hele tatt. Dersom en strekkode blir dekodet feil er det bare å utføre en tradisjonell manuell avlesning av denne strekkoden. Om strekkoden derimot er blitt dekodet til feil siffer, uten at de får beskjed om dette, er det kritisk. I løpet av de månedene programmet har vært i bruk har de enda ikke opplevd at strekkoder har blitt dekodet til feil siffer.

Løsning av problemstilling del 1

Automatisk deteksjon og dekoding av én strekkode per bilde vil redusere arbeidsmengden i flere av Vericos prosjekter betraktelig. En treffprosent på 98,4% fra viser at løsningen presentert i denne oppgaven mål. Verico har planlagt å ta i bruk denne løsningen i fremtidige prosjekter.

Løsning av problemstilling del 2

Detektering og dekoding av flere strekkoder per bilde er ikke noe Verico har behov for i sine prosjekter per i dag. Denne oppgaven viser at det er fullt mulig, noe som åpner for at Verico kan utnytte potensialet i oversiktsbildene de allerede har tatt på befaringene. Brukere av sluttproduktene kan få mulighet til å se bilder av hvor enhetene de søker befinner seg, i stedet for å bare få en veibeskrivelse i form av tekst.

Vedlegg

A Oversikt over innhold på vedlagt CD

Mappestruktur:

- `dekoding/`
  - `v11.MIDMAS.Masteroppgave.Console/` *(kildekode av konsollprogrammet)*
  - `v11.MIDMAS.Masteroppgave.GUI/` *(kildekode og JavaDOC av GUI-programmet)*
  - `BarcodeDecoderConsole.jar` *(kjørbar JAR-fil av konsollprogrammet)*
  - `BarcodeDecoderGUI.jar` *(kjørbar JAR-fil av GUI-programmet)*
  - `jfreechart-1.0.13.tar.gz` *(bibliotek for tegning av grafer)*
- `detektering/`
  - `oneBarcode.m` *(Matlab-script for detektering av én strekkode per bilde)*
  - `severalBarcodes.m` *(Matlab-script for detektering av flere strekkoder per bilde)*
- `eksempelbilder/`
  - `Bilde24.jpg` *(eksempelbilde med én strekkode)*
  - `VERx_10005.JPG` *(eksempelbilde med én strekkode)*
  - `VERx_10048.JPG` *(eksempelbilde med flere strekkoder)*
  - `VERx_10068.JPG` *(eksempelbilde med flere strekkoder)*
  - `VERx_10071.JPG` *(eksempelbilde med flere strekkoder)*
  - `VERx_10081.JPG` *(eksempelbilde med én strekkode)*
  - `VERx_10133.JPG` *(eksempelbilde med flere strekkoder)*
- `rapport.pdf` *(denne rapporten)*

B Matlab kildekode (deteksjon av strekkoder)

B.1 oneBarcode.m (deteksjon av én strekkode per bilde)

```
clear all; % Clear all variables
tic

folder = 'project1OneBarcode';
list = dir(folder);

% Create results-file
fid = fopen([folder 'Res/results.txt'], 'w'); % w=write
fprintf(fid, '');
fclose(fid);

% Loop Folder
for i=3:size(list),
    close all; % Close all figures

    file = list(i).name;
    figure(1); clf; set(figure(1),'color','white'); set(figure(1),'Position',[0 0 ...
        1080 1080]);
    set(gca,'LooseInset',get(gca,'TightInset'))

    % Read and Resize RGB-Image
    path = [folder '/' file];
    iOriginal  = imread(path);
    if (size(iOriginal,2) > size(iOriginal,1))
        scale = 480/size(iOriginal,1);
    else
        scale = 480/size(iOriginal,2);
    end
    iResized = imresize(iOriginal, scale);

    % Grayscale
    iGrayscale = rgb2gray(iResized);

    % Graythresh
    level = graythresh(iGrayscale);
    iGraythresh = im2bw(iGrayscale, level);

    % Morphological Skeletonization
    se = strel('square',5);
    iEroded = imerode(iGraythresh,se);
    iSkeleton = imsubtract(iGraythresh, iEroded);

    % Morphologically Open Image
    se = strel('line', 17, 90);
    iOpen = imopen(iSkeleton, se);

    % Morphologically Close Image
    se = strel('rectangle',[1,20]);
    iClose = imclose(iOpen,se);

    % Show region with the biggest area where (width/height)<2 (if exist)
    [labeled,numObjects] = bwlabel(iClose,4); % Label components

    if (numObjects>0)
        properties = {'Area', 'BoundingBox', 'Extrema', 'Centroid'};
        linedata = regionprops(labeled, properties);
        [sortedAreas sortedAreasIndex] = sort([linedata.Area], 'descend');
```

```
57
58          for j = 1:size(sortedAreas,2),
59              idx = sortedAreasIndex(j);
60              if( ((linedata(idx).BoundingBox(4)) / (linedata(idx).BoundingBox(3))) ...
                    < 2 )
61                  break;
62              end
63          end
64          iCleaned = ismember(labeled, idx);
65
66          % Show original image
67          h = subplot(3,3,8);
68          ax = get(h,'Position');
69          ax(2) = ax(2)+0.05;
70          set(h,'Position',ax);
71          imshow(iOriginal);
72
73          % Draw line
74          leftX = max( linedata(idx).Extrema(8,1), linedata(idx).Extrema(7,1) ); % ...
                max(left-topX, left-bottomX)
75          leftX = leftX-10;
76          rightX = max( linedata(idx).Extrema(3,1), linedata(idx).Extrema(4,1) ); % ...
                max(right-topX, right-bottomX)
77          rightX = rightX+10;
78          Y = linedata(idx).Centroid(2);
79          leftX = leftX/scale;
80          rightX = rightX/scale;
81          Y = Y/scale;
82          hold on;
83          plot([leftX,rightX], [Y,Y],'-','LineWidth',3,...
84              'MarkerEdgeColor','r',...
85              'MarkerFaceColor','g',...
86              'MarkerSize',5)
87          hold off;
88
89          % Decode Barcode
90          [status,result] = system(['java -jar BarcodeDecoderConsole.jar "' folder ...
                '/' file '" "' num2str(round(leftX)) '" "' num2str(round(rightX)) '" ...
                "' num2str(round(Y)) '"']);
91          title('h) Bl  linje p  originalbildet');
92      else
93          result = 'null';
94      end
95
96      h = subplot(3,3,1);
97      ax = get(h,'Position');
98      ax(2) = ax(2)-0.05;
99      set(h,'Position',ax);
100     imshow(iOriginal);
101     title(['a) imread (' num2str(size(iOriginal,2)) ' x ' ...
            num2str(size(iOriginal,1)) ' piksler)']);
102
103     h = subplot(3,3,2);
104     ax = get(h,'Position');
105     ax(2) = ax(2)-0.05;
106     set(h,'Position',ax);
107     imshow(iResized);
108     title(['b) Skalert bilde (' num2str(size(iResized,2)) ' x ' ...
            num2str(size(iResized,1)) ' piksler)']);
109
110     h = subplot(3,3,3);
111     ax = get(h,'Position');
112     ax(2) = ax(2)-0.05;
```

```
113     set(h,'Position',ax);
114     imshow(iGrayscale);
115     title('c) rgb2gray');
116 
117     h = subplot(3,3,4);
118     imshow(iGraythresh);
119     title('d) Bin rbilde');
120 
121     h = subplot(3,3,5);
122     imshow(iSkeleton);
123     title('e) Morfologisk omriss');
124 
125     h = subplot(3,3,6);
126     imshow(iOpen);
127     title('f) Morfologisk  pning ');
128 
129     h = subplot(3,3,7);
130     ax = get(h,'Position');
131     ax(2) = ax(2)+0.05;
132     set(h,'Position',ax);
133     imshow(iClose);
134     title('g) Morfologisk lukking');
135 
136     % Show Results
137     h = subplot(3,3,9);
138     ax = get(h,'Position');
139     ax(2) = ax(2)+0.05;
140     delete(h);
141     annotation( figure(1), 'textbox', ax, ...
142                     'String', ['i) Resultat fra dekoding:' char(10) char(10) ...
                            'Digits: ' result], ...
143                     'FitHeightToText', 'on' );
144 
145     % Save figure
146     [cdata, colormap] = getframe(gcf);
147     imwrite(cdata, [folder 'Res/' file]);
148 
149     % Write results to file
150     fid = fopen([folder 'Res/results.txt'], 'a'); % a=append
151     fprintf(fid, [path ';' result '\n']);
152     fclose(fid);
153 end
154 
155 % Write toc to file
156 fid = fopen([folder 'Res/results.txt'], 'a'); % a=append
157 fprintf(fid, ['\n\nElapsed time is ' num2str(toc) ' seconds.']);
158 fclose(fid);
```

B.2 severalBarcodes.m (deteksjon av flere strekkoder per bilde)

```
clear all;
close all;

% Settings
thresholdPercent = 2;
barcodeCover = 0.17;
properties = {'Area', 'Extrema', 'MinorAxisLength', 'MajorAxisLength'};

% List and loop dir
folder = 'project1SeveralBarcodes';
list = dir(folder);

fullscreen = get(0,'ScreenSize');
figure(1); clf; set(figure(1),'color','white'); set(figure(1),'Position',[0 -50 ...
    fullscreen(3) fullscreen(4)]);
set(gca,'LooseInset',get(gca,'TightInset'))

for i=3:size(list),
    file = list(i).name;
    path = [folder '/' file];

    if exist(path,'file')

        % Read, resize and view RGB-Image
        iOriginal  = imread(path);
        if (size(iOriginal,2) > size(iOriginal,1))
            scale = 1080/size(iOriginal, 1);
        else
            scale = 1080/size(iOriginal, 2);
        end
        iResized = imresize(iOriginal, scale);
        figure(1); imagesc(iResized); colormap(gray); axis equal;

        % Input from user
        [barcodeSizeX, barcodeSizeY] = ginput(2);
        title('Click on left, and then right side of a barcode.');
        barcodeWidth = sqrt(((barcodeSizeX(2)-barcodeSizeX(1))^2 + ...
            (barcodeSizeY(2)-barcodeSizeY(1))^2));

        % Grayscale
        iGrayscale = rgb2gray(iResized);

        % Vertical filter
        bvert = [ones(round(barcodeWidth/3), 1),-ones(round(barcodeWidth/3), 1)];
        iFilter2 = filter2(bvert, iGrayscale);

        % After thresholding
        iFilter2Sorted = sort(abs(iFilter2(:)), 'descend');
        threshold = iFilter2Sorted(ceil(thresholdPercent*numel(iFilter2Sorted)/100));
        iBinary = (abs(iFilter2) > threshold);

        % Remove regions that are to big/small
        barcodeLimMin = struct('Area', round(barcodeWidth/6));
        barcodeLimMax = struct('MajorAxisLength', round(barcodeWidth+10), ...
            'MinorAxisLength', round(barcodeWidth-10));
        iCleanedBinary = iBinary;
        cc = bwconncomp(iCleanedBinary, 4);
        stat  = regionprops(cc, properties);
        for j=1:cc.NumObjects
            if stat(j).Area < barcodeLimMin.Area
```

```
58                  iCleanedBinary(cc.PixelIdxList{j}) = 0;
59              end
60              if stat(j).MajorAxisLength > barcodeLimMax.MajorAxisLength
61                  iCleanedBinary(cc.PixelIdxList{j}) = 0;
62              end
63              if stat(j).MinorAxisLength > barcodeLimMax.MinorAxisLength
64                  iCleanedBinary(cc.PixelIdxList{j}) = 0;
65              end
66          end
67
68          % Check against barcode size
69          barcodeArea = ones(round(barcodeWidth/2.5), round(barcodeWidth));
70          iBarcode = filter2(barcodeArea/sum(barcodeArea(:)),double(iCleanedBinary));
71          iBarcode = (iBarcode > barcodeCover);
72
73          % Show figure(2)
74          figure(2); clf; set(figure(2),'color','white'); ...
                set(figure(2),'Position',[0 0 1080 1080]);
75
76          h = subplot(3,2,1);
77          ax = get(h,'Position');
78          ax(1) = ax(1)+0.033;
79          ax(2) = ax(2)-0.03;
80          set(h,'Position',ax);
81          imagesc(iOriginal); colormap(gray); axis equal;
82          title(['a) imread (' num2str(size(iOriginal,2)) ' x ' ...
                num2str(size(iOriginal,1)) ' piksler)']);
83
84          h = subplot(3,2,2);
85          ax = get(h,'Position');
86          ax(1) = ax(1)-0.033;
87          ax(2) = ax(2)-0.03;
88          set(h,'Position',ax);
89          imagesc(iGrayscale); colormap(gray); axis equal;
90          title(['b) rgb2gray av skalert bilde (' num2str(size(iResized,2)) ' x ' ...
                num2str(size(iResized,1)) ' piksler)']);
91
92          h = subplot(3,2,3);
93          ax = get(h,'Position');
94          ax(1) = ax(1)+0.033;
95          set(h,'Position',ax);
96          imagesc(iFilter2); colormap(gray); axis equal;
97          title('c) Filtrert med vertikalt filter');
98
99          h = subplot(3,2,4);
100         ax = get(h,'Position');
101         ax(1) = ax(1)-0.033;
102         set(h,'Position',ax);
103         imagesc(1-iBinary); colormap(gray); axis equal;
104         title('d) Etter terskling');
105
106         h = subplot(3,2,5);
107         ax = get(h,'Position');
108         ax(1) = ax(1)+0.033;
109         ax(2) = ax(2)+0.03;
110         set(h,'Position',ax);
111         imagesc(1-iCleanedBinary); colormap(gray); axis equal;
112         title('e) Fjernet regioner som er for store/sm ');
113
114         h = subplot(3,2,6);
115         ax = get(h,'Position');
116         ax(1) = ax(1)-0.033;
117         ax(2) = ax(2)+0.03;
```

```
118         set(h,'Position',ax);
119         imagesc(1-iBarcode); colormap(gray); axis equal;
120         title('f) Omriss av mulige strekkoder');
121
122         % save figure
123         [cdata, colormap1] = getframe(gcf);
124         imwrite(cdata, [folder 'Res/' file(1:end-4) '_2' file(end-3:end)]);
125
126         % Decode barcode regions
127         figure(1);
128         imshow(iOriginal);
129         cc = bwconncomp(iBarcode, 4);
130         stat  = regionprops(cc, properties);
131         hold on;
132         for j=1:cc.NumObjects
133             leftX = max( stat(j).Extrema(8,1), stat(j).Extrema(7,1) ); % ...
                    max(left-topX, left-bottomX)
134             leftX = leftX-5;
135             rightX = max( stat(j).Extrema(3,1), stat(j).Extrema(4,1) ); % ...
                    max(right-topX, right-bottomX)
136             rightX = rightX+5;
137             topY = max( stat(j).Extrema(1,2), stat(j).Extrema(2,2) );
138             bottomY = max( stat(j).Extrema(5,2), stat(j).Extrema(6,2) );
139             leftX = leftX/scale;
140             rightX = rightX/scale;
141             topY = topY/scale;
142             bottomY = bottomY/scale;
143
144             topY = max(ceil(topY),1);
145             bottomY = max(ceil(bottomY),1);
146             Y = topY + (bottomY-topY)/2;
147
148             if ((rightX-leftX) > 0.7*(size(barcodeArea,2)/scale)) && ...
                    ((bottomY-topY) > 0.7*(size(barcodeArea,1)/scale))
149                 diff = ((size(barcodeArea,2)+10)/scale) - (rightX-leftX);
150                 if diff>0
151                     leftX = leftX-(diff/2);
152                     rightX = rightX+(diff/2);
153                 end
154                 leftX = max(ceil(leftX),1);
155                 rightX = min(floor(rightX),size(iOriginal,2)-1);
156                 plot([leftX,rightX], [Y,Y],'-','LineWidth',1,...
157                         'MarkerEdgeColor','r',...
158                         'MarkerFaceColor','g',...
159                         'MarkerSize',5)
160                 [status,result] = system(['java -jar BarcodeDecoderConsole.jar "' ...
                        folder '/' file '" "' num2str(leftX) '" "' num2str(rightX) '" ...
                        "' num2str(round(Y)) '"']);
161                 if Y>(size(iOriginal,1)/2)
162                     text(leftX+((rightX-leftX)/2),...
163                         Y-((1.5*size(barcodeArea,1))/scale),...
164                         result,...
165                         'BackgroundColor',[.7 .9 .7],...
166                         'VerticalAlignment','bottom',...
167                         'HorizontalAlignment','center');
168                 else
169                     text(leftX+((rightX-leftX)/2),...
170                         Y+((1.5*size(barcodeArea,1))/scale), ...
171                         result, ...
172                         'BackgroundColor',[.7 .9 .7], ...
173                         'VerticalAlignment','top', ...
174                         'HorizontalAlignment','center');
175                 end
```

```
176             end
177         end
178         hold off;
179     end
180
181     % Save figure
182     [cdata, colormap1] = getframe(gcf);
183     imwrite(cdata, [folder 'Res/' file(1:end-4) '_1' file(end-3:end)]);
184
185     pause();
186
187 end
```

Referanser

[1] Verico AS. http://www.verico.com/.

[2] dlSoft. Barcode information for 1d barcodes. http://www.dlsoft.com/barcode_types/barcode_types_1D.htm.

[3] Simply Barcodes. Ean (european article number). http://www.officialeancode.com/.

[4] Simply Barcodes. Upc (universal product code). http://www.upccode.net/.

[5] Barcode Island. Ean-8 barcode. http://www.barcodeisland.com/ean8.phtml.

[6] Barcode Island. Upc-e barcode. http://www.barcodeisland.com/upce.phtml.

[7] Barcode Island. Upc-a barcode. http://www.barcodeisland.com/upca.phtml.

[8] Oracle. Java. http://www.java.com/en/.

[9] Object Refinery Ltd. Jfreechart. http://www.jfree.org/jfreechart/.

[10] Tråd på jfree-forum. http://www.jfree.org/phpBB2/viewtopic.php?f=3&t=27742.

[11] The Eclipse Foundation. Eclipse. http://www.eclipse.org/.

[12] ObjectAid. Objectaid uml explorer. http://www.objectaid.com/.

[13] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing, 4/E*. Prentice Hall, 2007.

[14] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 2007.

[15] MathWorks. Matlab. http://www.mathworks.com/products/matlab/.

[16] MathWorks. Image processing toolbox. http://www.mathworks.com/products/image/.

[17] Edward R. Dougherty and Roberto A. Lotufo. *Hands-on Morphological Image Processing*. SPIE Press, 2003.

[18] James Juett. Barcode localization using a bottom hat filter. Unpublished. http://digital.cs.usu.edu/~xqi/Teaching/REU09/Website/James/finalPaper.pdf, January 2010.

[19] Florian Knorn. M-code latex package. http://www.mathworks.com/matlabcentral/fileexchange/8015-m-code-latex-package.

[20] Chunhui Zhang, Jian Wang, Shi Han, and Mo Yi. Automatic real-time barcode localization in complex scenes. *IEEE International Conference on Image Processing*, pages 497–500, 2006.

[21] R.C. Gonzales, R.E. Woods, and S.L. Eddins. *Digital Image Processing Using MATLAB*. Prentice Hall, 2003.