DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: **Computer Science**	**Vårsemesteret, 2012** **Konfidensiell**
Forfatter: **Marita Lode Holbak**	..... (signatur forfatter)
Fagansvarlig: **Tom Ryen (UiS)** Veileder(e): **Jostein Øygarden (Verico AS)**	
Tittel på masteroppgaven: **Datafangst av merkeskilt på teknisk utstyr.** Engelsk tittel: **Data acquisition of informationplates on tecniqual equipment.**	
Studiepoeng: 30	
Emneord: **Invers Bilineær mapping, Java, matlabcontrol,** **merkeskilt, datafangst.**	Sidetall: **66** + vedlegg/annet: **CD ROM** **Stavanger, 14/6-2012**

Sammendrag

Dokumentasjon og verifikasjon er i den senere tid blitt en viktig del av kvalitetssikringen av produkter og tjenester inne industrien. Det finnes i dag en rekke hjelpemidler til bruk innen verifikasjonsprosessen. Denne oppgaven har som hensikt å se nærmere på en del verktøy og hjelpemidler innenfor et spesifikt fagfelt og forbedre disse.

Oppgaven er gjort i samarbeid med VericoA/S, som tilbyr sine kunder en fotobasert datainnsamling og datakvalitetssikring av teknisk utstyr. Dette gjøres ved at det blir tatt digitale bilder av merkeskiltene på det forskjellige utstyret, og disse bildene behandles deretter i VeriFacta, et program laget for dette formålet.

De digitale bildene går igjennom flere prosesser for å få det produktet kunden har betalt for. De forskjellige prosessene er i dag gjort manuelt, noen som er svært tidkrevende.

De siste årene har forskjellige studenter ved UiS arbeidet med å automatisere forskjellige deler av denne prosessen i henholdsvis prosjektoppgaver, bachelor oppgaver og master oppgaver. Noen slike oppgaver er også fremdeles under arbeid.

Under arbeidet med oppgavene har studentene brukt forskjellige typer verktøy, og de forskjellige elementene er egne applikasjoner eller kode-snutter som ikke er implementert i VeriFacta.

Det finnes i skrivende stund ingen oversikt over hvordan disse forskjellige elementene kan knyttes sammen, eller hva som eventuelt må utvikles for at man skal kunne få en fullstendig sammenhengende applikasjon som kan benyttes i datafangstprosessen.

I denne oppgaven er det utarbeidet en oversikt over hvordan disse forskjellige elementene passer inn i Verico AS datafangstprosess og det er også utviklet en applikasjon der de forskjellige metodene er kombinert på en slik måte at hele prosessen automatiserer så langt det per i dag er mulig. Applikasjonen er utviklet med tanke på at den skal kunne videreutvikles slik at det eneste manuelle arbeidet som tilslutt gjenstår er å mate de digitale bildene inn i programmet og kvalitetsjekke resultatene.

I dag brukes det svært mye tid på manuell inntasting og manuell registrering i datafangst prosessen, og dersom det er mulig å utvikle en applikasjon som kan gjøre hele eller deler av dette arbeidet automatisk, vil tids og kostnadsbesparelsene bli betydelige.

Innholdsliste

1 Innledning

1.1 Bakgrunn for oppgaven

Verico AS tilbyr sine kunder et produkt bestående av blant annet en database, der kundenes tekniske utstyr, tilhørende et anlegg, blitr dokumentert. Et anlegg kan for eksempel være en trafostasjon som blant annet består av transformatorer og fordelere. Disse er merket med skilt, som inneholder opplysninger om det enkelte apparatet. Denne informasjonen ønsker kunden å lagre i sin database, sammen med bilder av merkeskiltet og utstyr. Et eksempel på utstyr med tilhørende merkeskilt vises i figur 1.

Illustrasjon 1: Utstyr med tilhørende merkeskilt

Utstyret fotograferes og deretter behandles de digitale bildene i datafangst programmet VeriFacta, før det ferdige produktet leveres til kunden.

Dette produktet kan deretter benyttes for å få en oversikt over kundens utstyr, og kunden kan planlegge vedlikehold av utstyret uten å måtte reise ut til anlegget.

Selve behandlingen av bildene er en lang prosess som krever mye manuelt terminalarbeid.

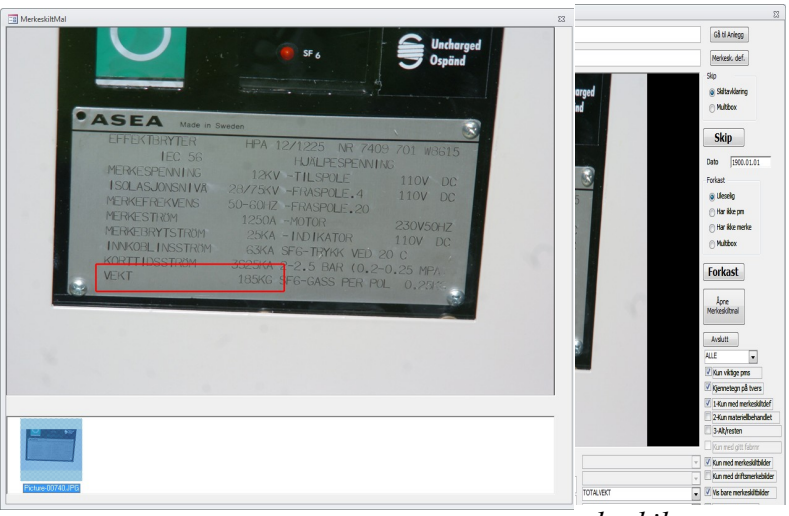
Dersom denne prosessen, helt eller delvis, kunne automatiseres, ville dette bety en betydelig effektivisering av arbeidet.

1.2 Problemstilling

For å kunne hente ut informasjonen som er trykket på merkeskiltene, må bildet av skiltet gjennomgå flere prosesser. Det første skrittet i dette arbeidet, er å klassifisere skiltet etter hvilket fabrikat og typebetegnelse utstyret hører inn under.

Dette gjøres ved at brukeren studerer skiltet og finner den nødvendige informasjonen som er trykket på skiltet. Deretter finner brukeren de samme verdiene i en liste og kobler skiltet opp mot riktig fabrikat og typebetegnelse.

Når klassifiseringen er utført, klippes de forskjellige feltene av merkeskiltet ut ved at brukeren markerer variabler på skiltet i henhold til en merkeskiltmal. I figur 2 vises et utsnitt av VeriFactas klippe funksjon. Brukeren har her valgt å klippe skiltets variabel for totalvekt, og denne er blitt markert med rød firkant. Når brukeren er fornøyd med klippet, velges neste merkeskilt, og total vekt markeres også på dette skiltet. Fordi klippingen ofte utføres av personer uten kjennskap til anleggene som skal dokumenteres, kan det noen ganger være vanskelig for brukeren å vite hvilket felt på skiltet som skal klippes. Av den grunn kan brukeren velge å åpne en merkeskiltmal som illustrert i figur 3 og bruke denne som mal for klippingen.



Illustrasjon 3: Merkeskiltmal *erkeskilt*

Når variablene er blitt klippet, må de de tastes inn. Dette gjøres ved at brukeren får opp den klipte variabelen på skjermen og taster denne inn manuelt. Alle inntastede verdier kvalitetssikres ved at brukeren kontrollerer den inntastede verdien mot den klippede.

Å innhente informasjon fra merkeskiltene på denne måten er i dag en svært tidkrevende prosess, og Verico AS ønsker derfor å utforske mulighetene for en effektivisering.

Det største potensialet for dette finner man i automatisering av klassifiseringen, klippingen og tastingen. Det er dette som er målet for denne oppgaven. For at prosessen skal bli mest mulig effektiv, blir datafangsten i dag utført etter samlebånd prinsippet. Dette vil være det foretrukne alternativet også etter en automatisering.

Et annet element som gjør datafangsten svært tidkrevende, er det faktum at hver enkelt variabel på merkeskiltet i den eksisterende prosedyren, behandles hver for seg. Dersom man kunne behandle alle variabler fra et merkeskilt i en enkelt operasjon, ville dette kunne effektivisere datafangstprosessen merkbart. Ved å utføre samme operasjon på alle merkeskilt før man går videre til neste, gjør man det i tillegg lettere å utføre en kvalitetssikring.

Kvalitetssikringen av det ferdige produktet vil fremdeles måtte utføres manuelt, da det er svært viktig for kundene at all informasjon i databasen er 100% korrekt. For at en variabel skal regnes som kvalitetssikret, må den ha blitt registrert med samme verdi to ganger. Dette gjøres i den eksisterende prosedyren ved at brukeren taster inn verdien, for så å gjennomgå verdien en gang til og bekrefte at den er korrekt. I noen tilfeller har kunden allerede registrert endel av variablene, og disse ligger da forhåndslagret i databasen. Dersom verdien som klippes og tastes inn er den samme som den allerede lagrede verdien, kan denne dermed regnes som kvalitetssikret. Dette er noe som kan utnyttes ved automatisering, og dermed kan også noe av tiden som i dag brukes på kvalitetssikring reduseres.

I dag er behandlingstiden for hver enkelt variabel i et merkeskilt, gjennomsnittlig på 190 sekunder. Dersom man antar at de automatiserte prosessene vil klare å behandle omtrent 50% av merkeskiltene, og at man i tillegg forbedrer de manuelle metodene, bør det være mulig å oppnå en tidsbesparelse på omkring 50%.

I den eksisterende prosedyren utføres alle trinnene som antydet ovenfor, altså manuelt i datafangst programmet VeriFacta. Da en automatisering av prosessene vil føre til en total

omlegging av programmets struktur, er det ønskelig for Verico AS at en ny applikasjon blir utviklet. VeriFacta er laget i Microsoft Access, men da Verico AS er i ferd med omlegge til MySQL database skal den nye applikasjonen i hovedsak utvikles i Java. På grunn av det store elementet av bildebehandling og avansert matematikk, bør noe av koden utvikles i MATLAB. Som en del av utviklingsprosessen bør det derfor vurderes om det vil være fornuftig å konvertere MATLAB til Java kode.

Hovedmålet med oppgaven er derfor først og fremst å få en oversikt over den eksisterende prosedyren, og deretter vurdere hvordan det vil være fornuftig å utforme og strukturere en automatisert versjon. Det vil også være nødvendig å betrakte de forskjellige eksterne funksjonenes nøyaktighet og vurdere hvordan eventuelle mangelfulle resultater skal håndteres. I tillegg må de elementene som skal knytte de eksterne funksjonene sammen til en helhetlig prosess, utvikles.

Den nye applikasjonen må være oversiktlig og utformet på en slik måte at nye funksjoner kan legges til dersom dette blir nødvendig. Applikasjonen må også kunne videreutvikles på en enkel måte.

1.3 Rapportens videre innhold

Rapporten er inndelt i ni kapitler.

1. Innledning – Introduksjon til oppgavegiveren og motivasjonen bak oppgaven.
2. Datafangstprosessen – Grundig gjennomgang av datafangstprosessen.
3. Oversikt – En oversikt over de forskjellige prosessene.
4. Praktiske og tekniske vurderinger – Her presenteres vurderinger som ble tatt før implementeringen.
5. Bildebehandling – Gjennomgang av metodene som er aktuelle i denne oppgaven.
6. Implementering og designmessige vurderinger – Her presenteres vurderinger gjort før og under implementeringen.
7. Vurdering av resultat – En vurdering av oppnådde resultater og en vurdering av tidsbesparelse.
8. Videreutvikling – Forslag til videre arbeid.
9. Konklusjon – Konklusjon med sammenligning av tidsbruk for gammel og ny prosedyre.

2 Datafangstprosessen

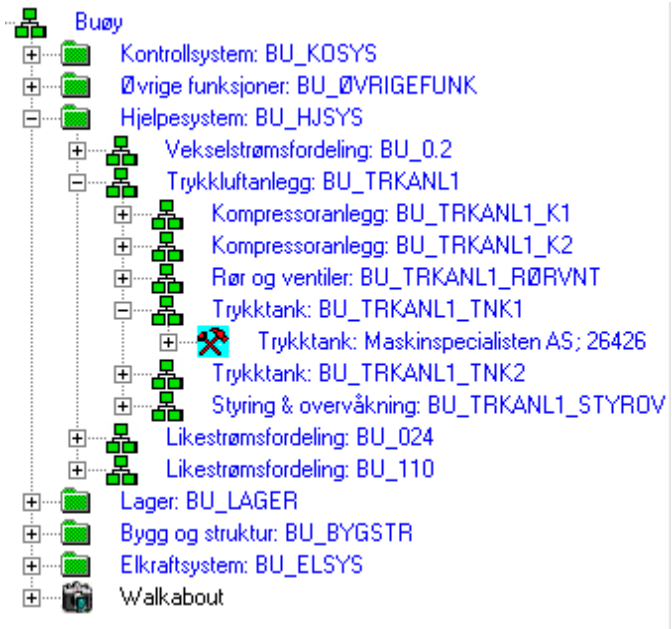
Her følger en gjennomgang av de forskjellige elementene i datafangstprosessen. Hovedvekten ligger på hvordan de forskjellige prosessene skal implementeres i applikasjonen utviklet i denne oppgaven men det vil også gis en kort beskrivelse av den eksisterende prosedyren.

Verico har erfart at den mest effektive måten å utføre datafangst ved hjelp av VeriFacta er å utføre en prosess på alle variabler av samme type før man går videre til neste. Det er derfor ønskelig at det nye programmet opererer på samme måte, men på ett og ett merkeskilt. Dette gir muligheter for å sette igang en prosess og la denne kjøre på alle merkeskilt uten bruker tilstede, for deretter å manuelt utføre prosessen på merkeskilt som er blitt behandlet feil, eller ikke behandlet i det hele tatt. Deretter kan neste prosess settes i gang.

2.1 Fotografering

Det første trinnet i prosessen er en grundig fotografering av anlegget hvor det blir tatt oversiktsbilder men også bilder av alt utstyr og alle merkeskilt

Disse bildene blir deretter sortert i en trestruktur av typen som er illustrert i figur 4.



Illustrasjon 4: Bilder sortert i trestruktur

Denne sorteringen foregår manuelt ved hjelp av «drag and drop» prinsippet, og hvert bilde får i tillegg en beskrivelse som forteller hva som er blitt fotografert. Denne måten å sortere bildene vil også bli benyttet i denne oppgaven.

De bildene som etter denne prosessen er blitt kategorisert som merkeskilt, skal deretter behandles for å hente ut og markere informasjonen funnet på skiltene. Prosessene som beskrives videre i dette avsnittet, forutsetter at denne bildesorteringen er blitt gjort og at hvert bilde derfor inneholder et merkeskilt.

2.2 Lagring av bildet

Bildene blir ikke fysisk sortert i forskjellige mapper, men det lages istedet en database der fil stien til hvert enkelt bilde lagres i en tabell. Et utsnitt av en slik tabell vises i figur 5.

ID_PicLink	Objekt_ID	PicSti	ThumbSti	PicType	Sortering	BirthDate	VF_Attr_ID	Tbl_Stj_Ref	PicIsCopy	Creator
34184	15808	Picture-00002.JPG	Picture-00002.JPG	100	1	04.01.2006 10:26:23	8184	1		Verico
34185	15814	Picture-00003.JPG	Picture-00003.JPG	100	1	04.01.2006 10:26:25	8185	1		Verico
34186	15820	Picture-00004.JPG	Picture-00004.JPG	100	1	04.01.2006 10:26:28	8186	1		Verico
34187	13470	Picture-00039.JPG	Picture-00039.JPG	100	1	04.01.2006 10:29:27	8196	1		Verico
34188	13470	Picture-00040.JPG	Picture-00040.JPG	100	2	04.01.2006 10:29:27	8197	1		Verico
34189	13470	Picture-00041.JPG	Picture-00041.JPG	100	3	04.01.2006 10:29:28	8198	1		Verico
34190	13470	Picture-00042.JPG	Picture-00042.JPG	100	4	04.01.2006 10:29:28	8199	1		Verico
34192	13194	Picture-00001.JPG	Picture-00001.JPG	100	1	05.01.2006 13:41:46	10915	1		Verico

Illustrasjon 5: Tabell fra VeriFactas database

I denne oppgaven benyttes ikke VeriFactas database, da Verico AS er i ferd med å gå over fra Microsoft Access til MySQL. Denne nye databasen er fremdeles under utvikling og det er derfor laget en egen enkel MySQL database som kun benyttes i denne oppgaven. Det er brukt rutiner for alle spørringer, slik at det er mulig og koble programmet opp mot en hvilken som helst database så lenge den inneholder tilsvarende rutiner.

2.3 Lokalisering av skiltet i bildet

Fordi bildene sorteres manuelt vil ethvert bilde som skal prosesseres videre inneholde et merkeskilt men det er ingen standard som forteller hvor i bildet skiltet befinner seg.

I den eksisterende prosedyren blir det ikke utført noen lokalisering av merkeskiltet, da alle prosessene utføres manuelt og det er derfor ikke nødvendig å definere hvor i bildet skiltet befinner seg.

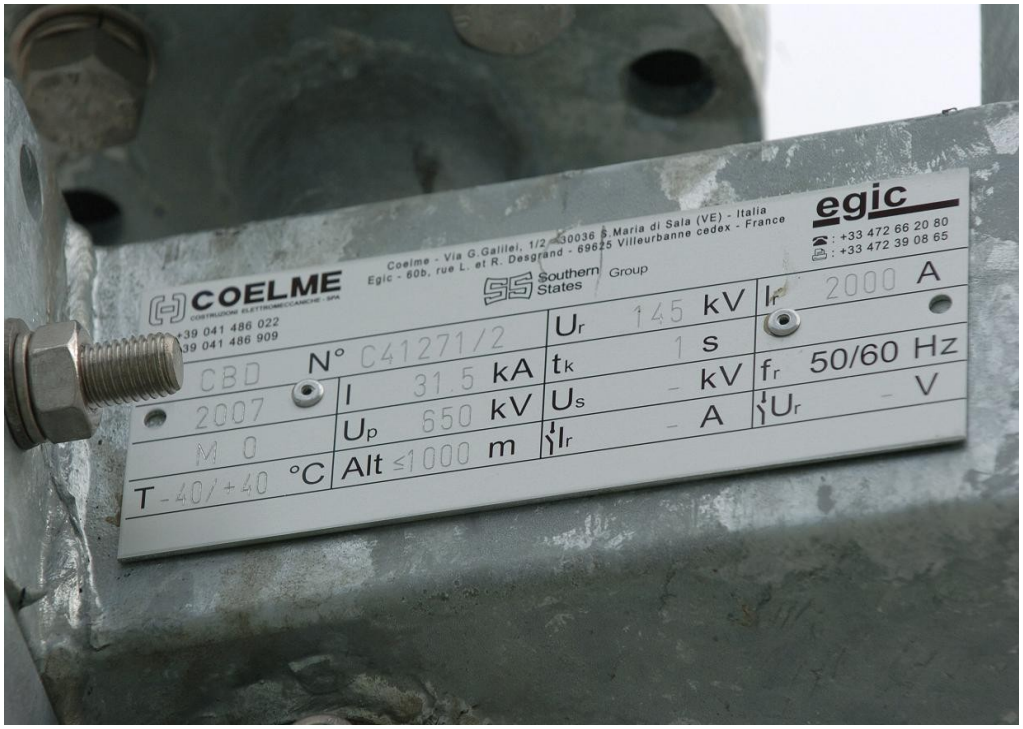
For å kunne klippe alle variabler i enkelt operasjon istedet for å måtte markere hver enkelt variabel slik det gjøres i den eksisterende prosedyren, er skiltet nødt til å ha samme form og størrelse som merkeskiltmalen. Å lokalisere skiltet er første steg i å oppnå denne formen.

For å vite hvor merkeskiltet befinner seg er det nok å finne de fire hjørnekoordinatene så fremt skiltet ikke er buet. Fordi det kun er svært få skilt som er buet er det i denne oppgaven ikke tatt hensyn til dette.

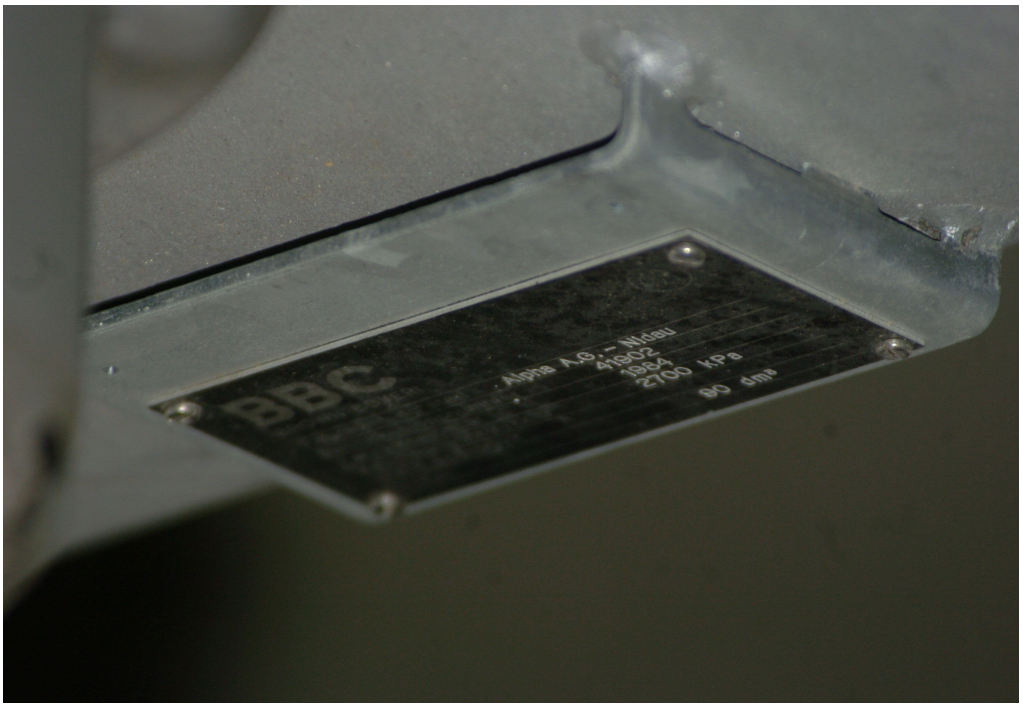
Det forutsettes i denne oppgaven at en MATLAB funksjon for å lokalisere merkeskiltet eksisterer, og at denne tar som input et bilde, og returnerer de fire hjørnekoordinatene. For å ta hensyn til tilfeller der den automatiske prosessen ikke klarer å lokalisere de fire hjørnekoordinatene, er det nødvendig at applikasjonen lar brukeren markere de fire hjørnene manuelt på de bildene der dette er nødvendig.

2.4 Warping

Anleggene Verico AS dokumenterer, inneholder ofte høyspenning eller andre hindringer som vanskeliggjør fotograferingen. Dette fører til at fotografene ofte blir tvunget til å ta bildene fra uheldige vinkler og avstander, noen som igjen fører til at merkeskiltene i bildene kan være forvrengt i større eller mindre grad. I figur 6 og 7 vises eksempler på både mild og ekstrem forvrengning.



Illustrasjon 6: Eksempel på mild forvrengning



Illustrasjon 7: Eksempel på ekstrem forvrengning

I tilfeller med ekstrem forvrengning, kan merkeskiltet være helt eller delvis uleselig. Det er allikevel nødvendig å behandle disse, da man ønsker å hente ut så mye informasjon som mulig fra merkeskiltet.

I den eksisterende prosedyren blir det ikke foretatt noen warping av bildene da dette ikke er nødvendig for å utføre manuell klipping og tasting. For at bildene skal kunne klassifiseres automatisk, noe som i sin tur er nødvendig for å kunne utføre automatisk klipping, er det derimot essensielt å eliminere denne forvrengningen. I denne oppgaven vil det derfor bli benyttet warping for å oppnå dette. Bildet og de fire punktene hentes da inn i en metode som warper bildet til et kvadrat eller et rektangel av en forhåndsbestemt størrelse og deretter klippes resten av bildet bort.

2.5 Klassifisering

Hvert merkeskilt har både et fabrikat og en typebetegnelse. To skilt kan ha samme fabrikat, men allikevel ha forskjellig typebetegnelse.

De forskjellige klassene har ofte markerte forskjeller i utseende, og det er dette som vil bli utgangspunktet for klassifiseringen. Ved hjelp av eliminasjonsmetoden vil man tilslutt stå igjen med ett eller flere alternativer. I denne oppgaven forutsettes det at en MATLAB funksjon som utfører dette allerede eksisterer, men for at denne skal kunne benyttes må merkeskiltene være "warpet" og ha en kvadratisk form.

I denne oppgaven sørges det for at merkeskiltet oppfyller kravene fra klassifiserings funksjonen. I tillegg utvikles det en manuell metode som håndterer de tilfellene der man står igjen med to eller flere alternativer. Brukeren må da kunne velge fra en liste hvilket fabrikat og typebetegnelse skiltet hører til under.

Det er svært viktig at merkeskiltet klassifiseres riktig, og det er derfor ønskelig at applikasjonen lar brukeren gå igjennom alle skilt som er blitt klassifisert av den automatiserte metoden og bekrefte at disse er korrekte.

2.6 Klipping

Hvert merkeskilt blir i VeriFacta klippet manuelt ved at brukeren markerer hver enkelt variabel med et rødt rektangel, og deretter godkjenner klippet. Dette gjøres for en og en variabel, og er en av de mest tidkrevende delene av datafangstprosessen. I denne oppgaven er målet at det skal være mulig å klippe alle variabler i en operasjon, noe som er mulig dersom merkeskiltet er blitt korrekt klassifisert og form og størrelse på merkeskiltet er tilnærmet lik det man finner i merkeskiltmalen.

Når et merkeskilt er blitt klassifisert vil det bli tilknyttet en merkeskiltmal. Denne malen inneholder informasjon om hva som er standard høyde og bredde for merkeskiltet, samt en tilhørende klippemal. Ved hjelp av høyden og bredden kan en ny "warp" utføres på merkeskiltet, slik at det får nøyaktig samme form som merkeskiltsmalen. Når dette er gjort kan klippemalen legges over merkeskiltet, slik at man kan få klippet alle variablene i en enkelt operasjon.

Fordi merkeskiltene i utgangspunktet kan ha vært mer eller mindre fordreid, og er blitt "warpet" til et format bestemt av merkeskiltmalen, er det ikke garantert at en klippemal alltid vil treffe nøyaktig. Det må derfor implementeres en metode som lar brukeren justere klippemalen der dette er nødvendig.

2.7 Optisk tegngjenkjenning (OCR)

Etter at variablene er klippet, blir de i VeriFacta tastet inn manuelt. For å effektivisere denne prosessen, er målet å implementere optisk tegngjenkjenning.

Dette er det allerede blitt arbeidet med av Andreas Waal i master oppgaven "Optical Character Recognition(OCR) on Electrical Specification Plates"[9], og denne er utviklet slik at den skal kunne lese inn feltene som er klippet ved bruk av klippemalen. OCR funksjonen er utviklet i MATLAB, og må enten konverteres til Java kode, eller implementeres slik den er.

Det er også nødvendig å gjøre det mulig for brukeren å korrigere verdien manuelt i de tilfellene der OCR funksjonen ikke klarer å gjenkjenne den klippede verdien.

2.8 Kvalitetssikring

Kvalitetssikring er svært viktig for Vericos kunder, da det er nødvendig at all informasjon som tilslutt blir lagret i databasen er 100% korrekt.

For å oppnå dette, ønsker Verico at kvalitetsikringen i stor grad fremdeles skal foregå manuelt, og det må derfor være mulig for brukeren å gå igjennom alle innleste eller inntastede verdier og bekrefte at disse er korrekte.

Et unntak fra dette er de situasjonene der kunden allerede har oppgitt verdier på forhånd. I disse tilfellene er det nok at de innleste eller inntastede verdiene stemmer med de som er gitt av kunden.

3 Oversikt

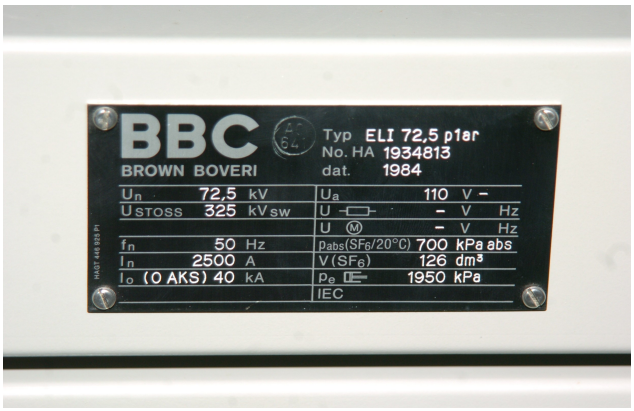
For å kunne utføre en automatisering av datafangstprosessen er det nødvendig å få en god oversikt over de forskjellige delene av prosessen og deretter vurdere hvordan disse må struktureres. Det er derfor viktig å få et overblikk over hvordan de forskjellige prosessene bearbeider bildene og hvilken form bildet har i det det forlater den enkelte prosess. Det er også viktig å kartlegge hvilken informasjon de forskjellige prosessene henter ut av bildet, og også hvilken informasjon de forskjellige prosessene er avhengige av for å kunne utføres.

3.1 Bildebehandling

I den eksisterende prosedyren er klippingen den eneste prosessen som utføres direkte på bildet av merkeskiltet. De andre prosessene henter kun informasjon fra skiltet ved manuell avlesning.

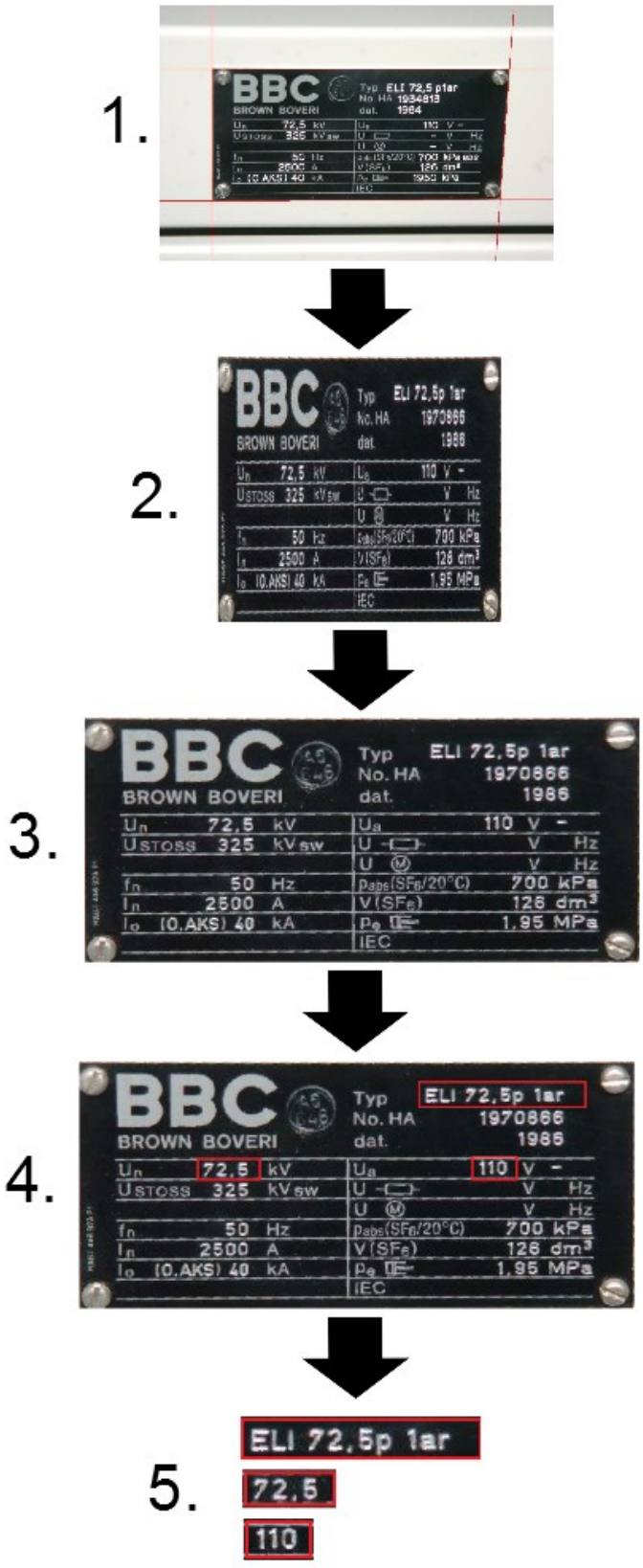
I applikasjonen utviklet i denne oppgaven, er det derimot nødvendig å utføre flere operasjoner på hvert bilde. Dette for at det skal være mulig å automatisk hente ut informasjonen.

Denne første oversikten konsentrerer seg om hvordan de forskjellige prosessene i applikasjonen behandler bildet og på hvilken form bildet er når det sendes videre til neste prosess. I figur 8 ser man et eksempel på et originalbilde, slik det opptrer idet det blir gitt som input til applikasjonen.



Illustrasjon 8: Eksempel på et originalbilde

Figur 9 viser de forskjellige prosessene utført på samme bilde og nedenfor finnes en beskrivelse av hver enkelt av disse prosessene.



Illustrasjon 9: Oversikt over prosessene utført på skiltet

1. Lokasjon av skiltet. Denne prosessen tar original bildet som input. De røde linjene brukes for å lokalisere de fire hjørnepunktene i skiltet. Disse hjørnepunktene lagres og brukes videre i programmet for å fortelle hvor i bildet skiltet befinner seg. Det blir ikke utført noen form for bildemanipulering i dette trinnet og bildet beholder derfor sin originale form.

2. Denne prosessen tar originalbildet og de fire hjørnekoordinatene som input. De fire hjørnekoordinatene brukes til å utføre en "warp" på skiltet og klipper deretter bort alt overflødig bildemateriell, slik at man sitter igjen kun med merkeskiltet. Størrelsen på skiltet blir justert til en forhåndsbestemt størrelse.

3. Bildet fra prosess nr. 2 gjennomgår her en klassifisering. Når fabrikat og typebetegnelse er blitt bestemt, tilordnes skiltet en merkeskiltmal og gjennomgår nok en "warp" som endrer størrelsen på skiltet slik at det får nøyaktig samme størrelse og form som merkeskiltmalen

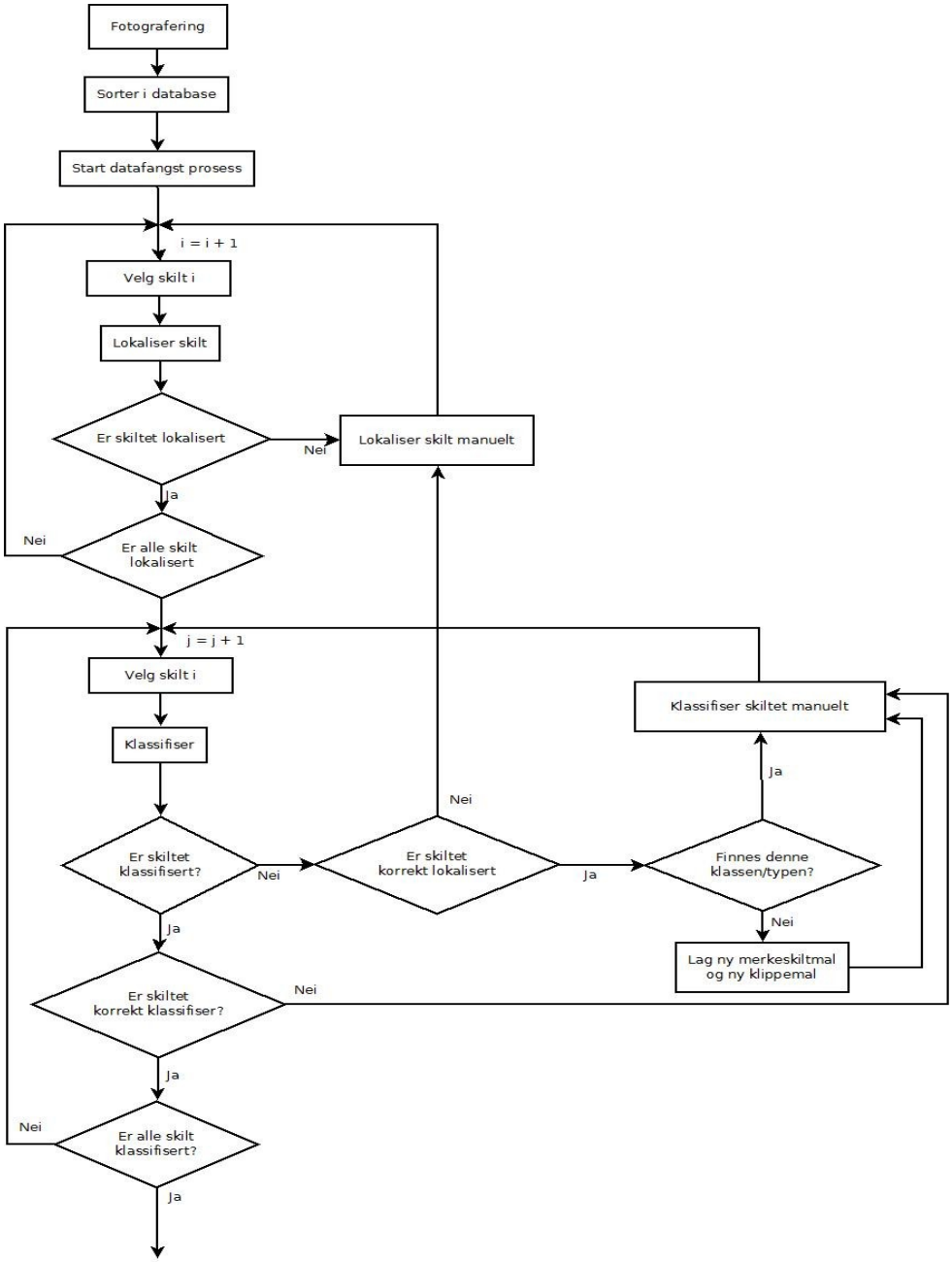
4. Denne prosessen legger en klippemal over bildet fra prosess nr 3. Denne klippemalen inneholder alle klipp som skal utføres på denne typen skilt. Hver skilttype har en egen klippemal.

5. Fra bildet i prosess nr 4 klippes de forskjellige variablene ut og er nå klare til å leses inn i databasen.

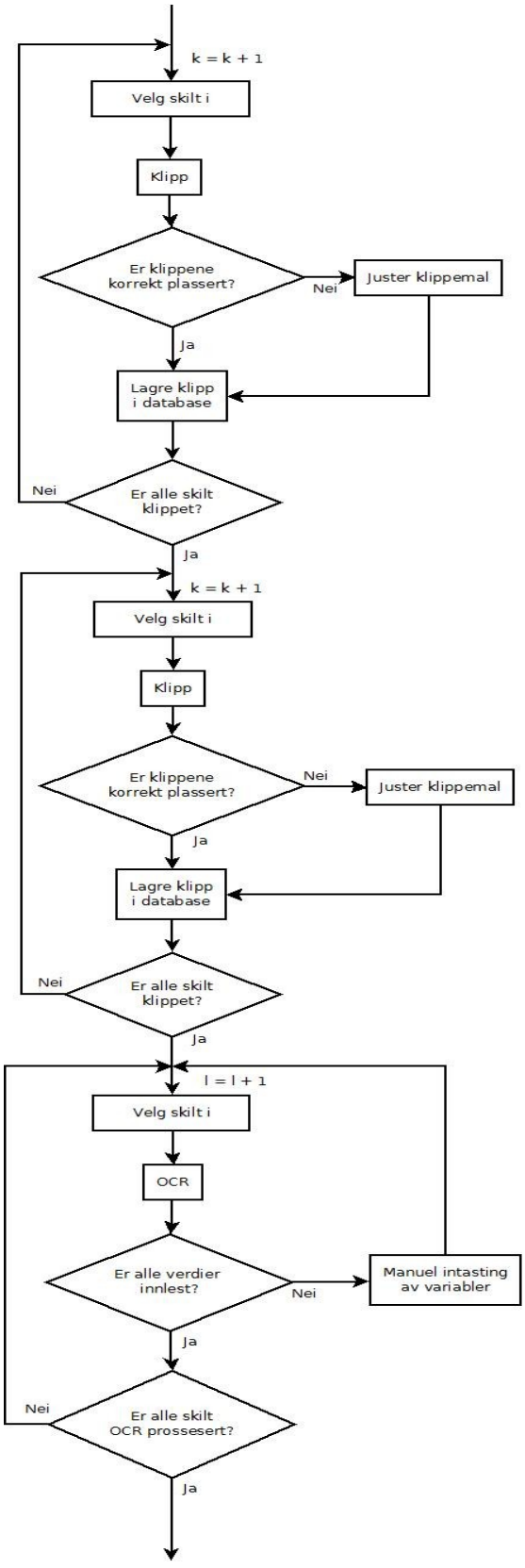
3.2 Prosessering

I figur 10, 11 og 12 vises en oversikt over hvordan ett merkeskilt skal prosesseres i den automatiserte prosedyren. Det er her verdt å merke seg at det for å effektivisere prosessen

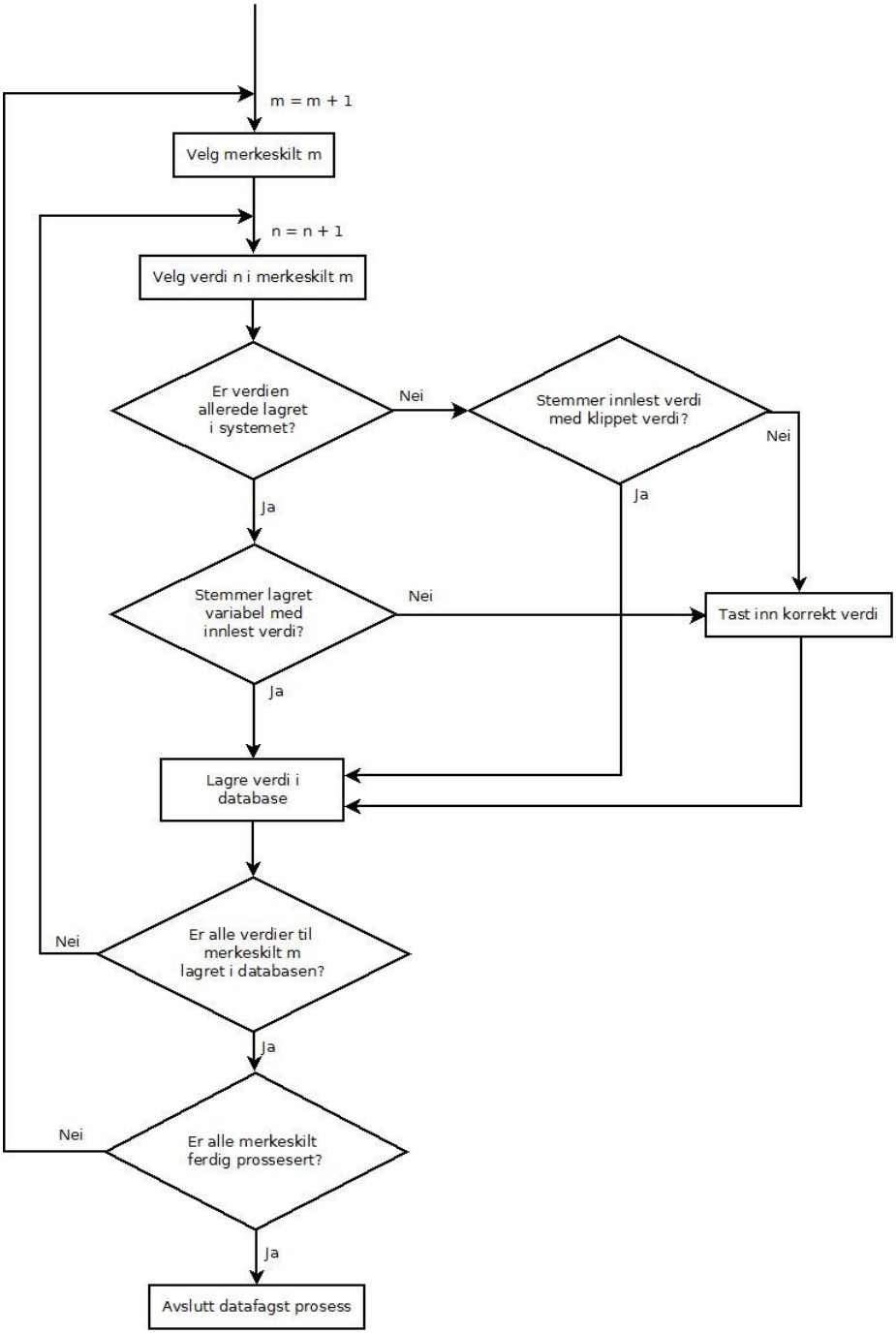
legges opp til at man behandler alle bilder som er tilgjengelige for en prosess før man går videre til neste prosess og gjør det samme her. Det blir også utført kvalitetskontroll i de prosessene der en feil ikke vil bli oppdaget automatisk i den påfølgende prosessen. Dette fordi en ikke skal få feil tidlig i datafangsten som ikke oppdages før man er gått igjennom flere av prosessene.



Illustrasjon 10: Datafangstprosessen del 1



Illustrasjon 11: Datafangstprosessen del 2



Illustrasjon 12: Datafangstprosessen del 3

4 Praktiske og tekniske vurderinger

I dette kapittelet gjennomgås de praktiske og tekniske vurderingene som ble foretatt før den automatiserte datafangst applikasjon ble utviklet.

4.1 Oppbygning

Applikasjonen er bygget opp med tanke på at det skal være enkelt å utvide og legge til nye moduler. Den er derfor utviklet på en slik måte at hver enkelt delprosess er en egen selvstendige applikasjon, som til slutt knyttes sammen i et felles brukergrensesnitt.

Det er også lagt vekt på å gjøre hver enkelt klasse så oversiktlig som mulig, da det vil være nødvendig å videreutvikle de forskjellige elementene for at applikasjonen skal kunne fungere som et fullverdig datafangstprogram.

I denne første prototypen av programmet er det derimot ikke lagt stor vekt på selve brukergrensesnittet. Det er istedet lagt vekt på å knytte de forskjellige elementene sammen på en fornuftig måte.

4.2 Kvalitetskontroll

Informasjonen som hentes inn for Vericos kunder lagres i en database slik at kunden kan bruke denne blant annet i forbindelse med vedlikeholdsarbeid og for å få en oversikt over de tekniske spesifikasjonene til forskjellig utstyr. For at dette skal være mulig, er det viktig for kunden at all informasjon som lagres er 100% korrekt. Det er bedre at en variabel ikke lagres enn at den inneholder feil informasjon.

Dette er håndtert i applikasjonen ved at det i noen tilfeller er blitt lagt til en mulighet for kvalitetskontroll, og i andre tilfeller at en slik kvalitetskontroll enkelt kan implementeres. Dette er blitt gjort på denne måten fordi man enda ikke kan vite nøyaktig hvordan de selvstendige MATLAB funksjonene vil fungere i applikasjonen når de er ferdig utviklet. Det er heller ikke satt opp noen kobling til Verico AS egen database, noe som gjør det umulig å foreta en kontroll mot allerede lagrede verdier.

4.3 Kobling mellom applikasjonen og eksterne metoder

Både funksjonen som lokaliserer merkeskiltet i bildet, funksjonen for klassifisering og den optiske tegngjenkjenningen, blir utviklet i MATLAB. Det er derfor nødvendig å bestemme hvordan disse forskjellige funksjonene skal implementeres i applikasjonen.

I denne oppgaven er det blitt vurdert tre forskjellige løsninger:

4.3.1 MATLAB Builder JA

MATLAB Builder JA [7], er MATLABs egen Java konverterer som krypterer MATLAB koden og pakker den inn i Java kode slik at den oppfører seg som en hvilken som helst Java klasse. Konverteren har også et grensesnitt som håndterer MATLAB figurer og gjør det mulig for brukeren å zoome, rotere og manipulere bildene.

Fordelen med MATLAB Builder JA, er blant annet at konverteringen blir gjort automatisk. Denne automatiseringen gjør det raskt og enkel måte å konvertere MATLAB kode til Java kode. Den konverterte koden kan etterpå implementeres direkte i Java applikasjonen.

På den andre siden, får man ikke automatisk en ryddig og oversiktlig kode ved en slik konvertering. Dette vil igjen gi endel utfordringer i forhold til eventuell videreutvikling av koden, dersom man ikke setter av tid til manuelt å gå inn og forbedre koden etter konverteringen. En annen bakdel med MATLAB Builder JA, er at dette er en relativt kostbar investering.

4.3.2 MatlabControl

Matlabcontrol[6], er en Java API med åpen kildekode som gjør det mulig å kalle MATLAB fra en Java-applikasjon. Kommunikasjonen mellom Java-applikasjonen og MATLAB foregår ved hjelp av en proxyserver.

Fordelen med denne tilnærmingen er at man ikke trenger å bruke tid på å skrive om MATLABkoden, men kan kalle den direkte slik den er.

Bakdelen med å gjøre det på denne måten, er at MATLAB må være installert på maskinen som kjører applikasjonen. I tillegg er tidsbruken ved å starte MATLAB, for så å kalle de

forskjellige funksjonene, vesentlig høyere en det man ville brukt dersom koden var implementert som Java klasser.

4.3.3 Manuell konvertering

Den tredje muligheten som ble vurdert i denne oppgaven, var å manuelt skrive om MATLABkoden til Java kode.

Manuell konvertering er mer tidkrevende, men resultatet blir en mer ryddig og oversiktlig kode en den man får med automatisk konvertering.

4.3.4 Konklusjon

Da applikasjonen i denne oppgaven er laget for å undersøke hvorvidt de forskjellige funksjonene lar seg sy sammen til en sammenhengende prosess, falt valget på matlabcontrol. Dette fordi det vil være unødvendig å bruke tid eller penger på å konvertere MATLABkoden til Java kode før man vet om det er mulig å sette sammen en automatisert applikasjon av de allerede eksisterende funksjonene.

Matlabcontrol gjør det mulig, på en enkel måte, å teste hvordan de forskjellige MATLAB funksjonene fungerer i applikasjonen. Dersom man velger å videreutvikle applikasjonen for kommersielt bruk, kan man vurdere om man skal investere i MATLAB Builder JA eller konvertere koden manuelt.

5 Bildebehandling

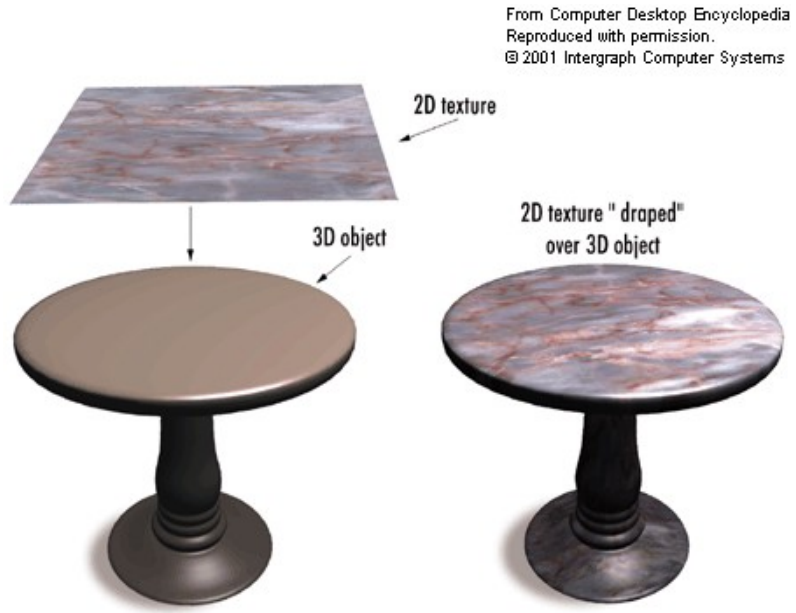
5.1 Geometriske operasjoner

Geometriske operasjoner[5], er operasjoner som transformerer piksel koordinatene i bilde I, til en ny plassering i bilde I', jamfør formel 1

$$I(x, y) = I'(x', y') \tag{1}$$

Typiske eksempler på slike operasjoner er vridning, skalering og rotering alle operasjoner der bildets geometri blir forandret.

I figur 13 illustreres et annet vanlig bruksområde for geometriske bilde operasjoner. Her brukes tekstur mapping til å legge et todimensjonalt bilde på en tredimensjonal flate.



Illustrasjon 13: Teksturmapping[8]

I denne oppgaven er mange av bildene tatt i vinkler som gjør at det avbildede skiltet blir forvrengt, og for at det skal kunne benyttes automatiserte metoder til klassifisering og klipping av skiltene, er det nødvendig at de transformeres til en standard form bestemt av

operasjonen som skal behandle skiltet. Til dette kan man benytte den geometriske operasjonen "warping".

5.2 Warping

I dagligtale betyr ordet "warp" å bøye eller vri noe ut av den formen det i utgangspunktet har. Image Warping[1] brukes om geometriske operasjoner som digitalt manipulerer et bilde slik at alle punkter i bildet flyttes fra en posisjon i det originale bildeplanet, til en annen posisjon i det nye bildeplanet. Dette kan blant annet brukes til å fjerne fordreininger i et bilde eller manipulere to bilder slik at de fremstår som et enkelt bilde.

En "warping" består av de to funksjonene som sees i funksjon 2 og 3,

$$x'(x, y) \tag{2}$$

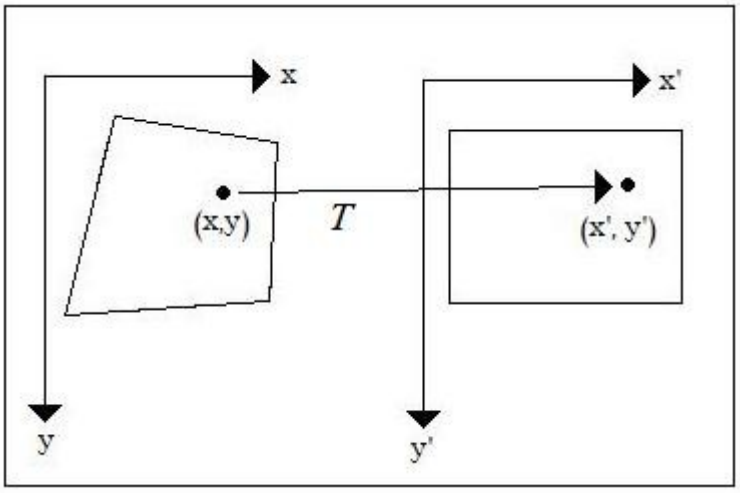
$$y'(x, y) \tag{3}$$

hvor (x,y) er koordinatene til et punkt i det første bildet, og (x',y') er koordinatene til samme punkt i det nye bildet.

I dette tilfellet er det nødvendig å transformere en hvilken som helst firkant til et rektangel av varierende dimensjon. Til dette formålet er det ideelt å bruke invers bilineær mapping.

5.3 Invers bilineær mapping

Bilineær mapping[2][3] er en ikke-affin transformasjon, som kan benyttes når man skal mappe en firkant til en annen firkant, der en eller begge kan være uregelmessige firkanter. Med uregelmessige menes det at både vinkler og lengden på sidene kan varieres uavhengig av hverandre. I denne oppgaven er kilden en uregelmessig firkant, mens destinasjons bildet er et rektangel, dette illustreres i figur 14 der T er den bilineære mapping funksjonen.



Illustrasjon 14: Bilineær mapping av firkant til rektangel

En affin transformasjon er en transformasjon som bevarer parallelle linjer. Dette vil si at alle rette linjer som før transformasjonen var parallelle, fortsatt vil være parallelle etter transformasjonen er utført. Bilineær mapping er en ikke-affin transformasjon, da parallelle linjer ikke bevares, dette ses eksempel på i figur 14.

Selve transformasjonen utføres ved at man for hvert punkt i kvadratet beregner koordinatene til punktets nye plassering i firkanten.

De nye koordinatene beregnes etter formlene 4 og 5.

$$x' = a_1 x + a_2 y + a_3 xy + a_4 \tag{4}$$

$$y' = b_1 x + b_2 y + b_3 xy + b_4 \tag{5}$$

En affin transformasjon er lineær. I bilineær mapping er det elementet xy, som gjør transformasjonen ikke-lineær og dermed ikke-affin.

De åtte parametrene (a_1 ... a_4 og b_1 ... b_4) er koeffisientene som representerer selve mappingen.

For å utføre en bilinær mapping er det først nødvendig å finne disse åtte koeffisientene. Dette gjøres ved at man setter opp ligningen etter formel 6.

$$\underline{x}' = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & x_1 y_1 & 1 \\ x_2 & y_2 & x_2 y_2 & 1 \\ x_3 & y_3 & x_3 y_3 & 1 \\ x_4 & y_4 & x_4 y_4 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = W \cdot \underline{a} \tag{6}$$

Der $\underline{x}'$ representerer hjørnekoordinatene i destinasjons-firkanten og $x_1 .... x_4$ og $y_1 ... y_4$ representerer hjørnene i kildefirkanten.
Og tilsvarende for $\underline{y}'$ i formel 7.

$$\underline{y}' = W \cdot \underline{b} \tag{7}$$

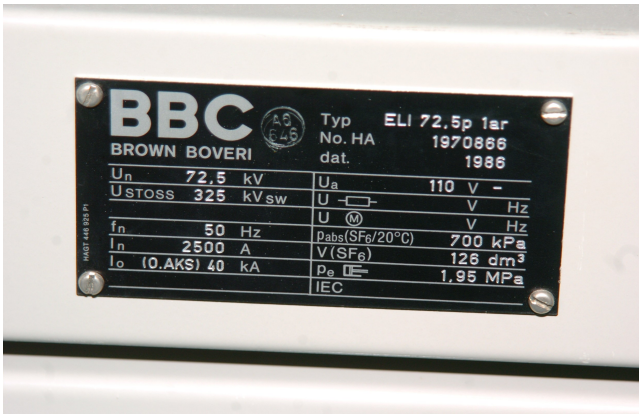
Der $\underline{a}$ og $\underline{b}$ er ukjente, mens $\underline{x}'$, $\underline{y}'$ og W er kjente. Dersom man antar at W er inverterbar kan man gjøre følgende operasjoner for å komme frem til formlene som gir de åtte koeffisientene.

$$\underline{x}' = W\,\underline{a} \qquad\qquad \underline{y}' = W\,\underline{b}$$

$$W^{-1}\underline{x}' = \cancel{W^{-1}W}\,\underline{a} \qquad\qquad W^{-1}\underline{y}' = \cancel{W^{-1}W}\,\underline{b}$$

$$\underline{a} = W^{-1}\underline{x}' \qquad\qquad \underline{b} = W^{-1}\underline{y}'$$

Deretter beregnes destinasjonskoordinatene for hvert enkelt punkt i kildefirkanten. Et eksempel på bilineær mapping utført på et merkeskilt illustreres i figur 15 og 16, der illustrasjon 15 viser originalbildet, og illustrasjon 16 viser merkeskiltet etter at bilineær mapping er blitt utført.

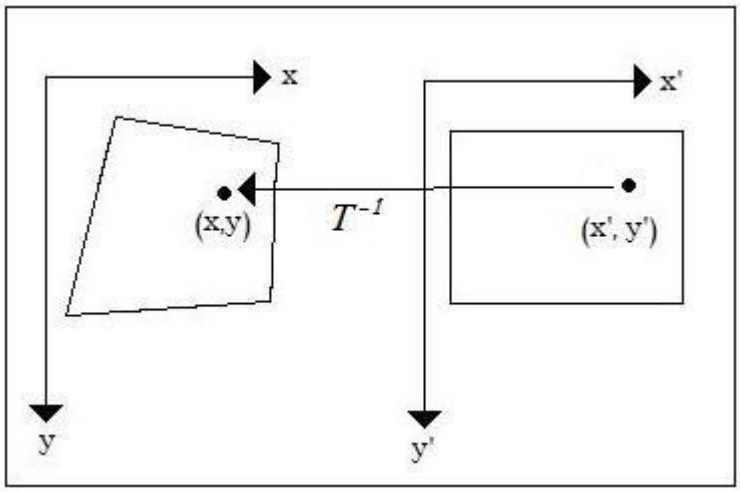


Illustrasjon 15: Originalbildet

Illustrasjon 16: Originalbilde etter bilineær mapping.

Man kan her se at det oppstår støy i form av sorte piksler på det "warpede" bildet. Dette er fordi vi her opererer med piksler og dermed diskrete posisjoner. Fordi bilineær mapping er en forover-mapping, beregner man for hvert piksel i kildebildet de nye koordinatene i destinasjonsbildet. De diskrete posisjonene til hver enkelt piksel fører dermed til at det i noen tilfeller vil være posisjoner i destinasjonsbildet som ikke vil bli tilegnet et piksel fra kildebildet.

Løsningen på dette problemet er å benytte bakover-mapping, eller også kaldt invers mapping. Dette vil si at man for hvert punkt i destinasjons bildeplanet beregner hvilken piksel-verdi fra kildebildet punktet skal få. Dette illustreres i figur 17, der T^{-1} er den inverse bilineære mapping funksjonen.



Illustrasjon 17: Invers bilineær mapping av firkant til rektangel.

Tilsvarende som for bilineær mapping må man for invers bilineær mapping også beregne de åtte koeffisientene, her representert med a' og b'. Disse beregnes på tilsvarende måte som i bilinear mapping, og vi ender dermed opp med formlene 8 og 9.

$$\underline{a}' = W'^{-1} \cdot \underline{x} \tag{8}$$

$$\underline{b}' = W'^{-1} \cdot \underline{y} \tag{9}$$

der W' kan ses i formel 10.

$$W' = \begin{bmatrix} x_1' & y_1' & x_1'y_1' & 1 \\ x_2' & y_2' & x_2'y_2' & 1 \\ x_3' & y_3' & x_3'y_3' & 1 \\ x_4' & y_4' & x_4'y_4' & 1 \end{bmatrix} \tag{10}$$

Deretter utføres følgende algoritme for å gi alle pikslene i destinasjonsbildet riktig piksel-verdi. Vi antar at pikselet i øvre venstre hjørne har posisjon (0,0) og pikselet i nedre venstre hjørne har posisjon (x'max, y'max)

```
for (x' = 0 : x'max)
        for (y' = 1 : y'max)
                Finn posisjon (x,y) i orginalbildet:
                x = [x'   y'   x'y'   1] · a
                y = [x'   y'   x'y'   1] · b
                x og y avrundes til nærmeste heltall.

                La pikselet i posisjon (x',y') få fargeverdien til pikselet
                i posisjon (x,y)
        end
end
```

I figur 18 vises et eksempel på et merkeskilt som er blitt "warpet" ved bruk av invers bilineær mapping. Dersom man sammenligner dette resultatet med merkeskiltet fra figur 15 ser man at kvaliteten er blitt merkbart forbedret da det ikke lenger oppstår støy i form av sorte punkter.

Illustrasjon 18: Merkeskilt etter "warp" ved bruk av invers bilineær mapping

5.4 Optisk tegngjenkjenning

Optisk tegngjenkjenning blir utført på bildet ved hjelp av en MATLAB funksjon laget av Andreas Wall i forbindelse med hans masteroppgave våren 2011 gitt av Verico AS. [4]

Funksjonen tar inn utklippene av verdiene fra merkeskiltet, og disse gjennomgår deretter en rekke operasjoner før funksjonen kommer med et forslag til verdier for hvert enkelt tegn.

Det første steget i denne prosessen er en bakgrunns segmentering, dette vil si at man fjerner bakgrunnen i bildet slik at man sitter igjen med et bilde der selve tallene er det eneste som gjenstår. Dette blir gjort ved bruk av "thresholding". Formelen for "thresholding" kan ses i formel 11.

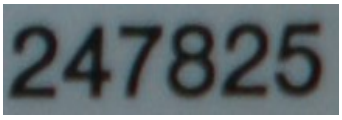
$$g(m,n)=\begin{cases} 1 & dersom \quad f(m,n)>t \\ 0 & ellers \end{cases} \tag{11}$$

Der *f* er input bildet, *g* er en binær versjon av bilde etter "thresholding", *(m,n)* er pikselkoordinatene og *t* er "threshold" verdien. For hver piksel i input bildet sjekkes det om

pikselverdien er høyere en den valgte *t,* og dersom dette er tilfellet blir verdien satt til 1 ellers blir den satt til 0. Som resultat får man dermed et sort hvitt binært bildet, der 0 representerer svart og 1 representerer hvit.

I den optiske tegngjenkjennings funksjonen benyttes flere utarbeidede formler for å kalkulerer den beste "threshold" verdien i hvert enkelt tilfelle.

Originalbildet med tilhørende binært bilde etter "thresholding" illustreres i figur 19 og 20.



Illustrasjon 19: Orginalbilde

Illustrasjon 20: Binærbilde etter "thresholding"

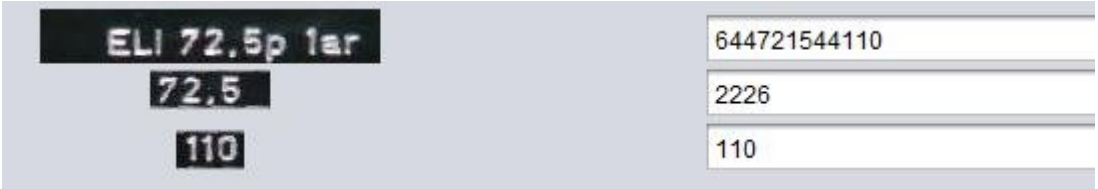
Når bakgrunns segmenteringen er utført, kombinerer metoden flere bilde operasjoner for å segmentere bildet opp i del-bilder, der hvert del-bilde inneholder et enkelt tegn.

For hvert tegn finner metoden deretter alle steder der linjer krysses, for så å segmentere ut alle linjene i tegnet ved å fjerne krysspunktene. Deretter kan retningen til hvert enkelt linjestykke beregnes. Hele tegnet blir så delt opp i 9 mindre deler hvor det for hver del beregnes en vektor med verdier kalkulert fra retning på linjer, lengden på linjer, krysspunkter etc.

Disse 9 vektorene brukes til å sammenligne tegnet med tegnene i trenings-dataen, og ved hjelp av k-nearest-neighbor bestemme verdien til tegnet. Den optiske tegngjennkjennings metoden kan per dags dato kun tolke tall.

På bakgrunn av dette kan man se at resultatet av den optiske tegngjenkjennings prosessen er avhengig av kvaliteten på på input-bildet. Det vil for eksempel være vanskelig å skille hvert enkelt tegn fra hverandre dersom bildet er så uklart at tegnene flyter sammen. Det vil også være vanskelig å skille mellom tallet fem og tallet seks dersom tegnene er uklare. I figur 21 vises et eksempel på den optiske tegngjenkjenningen utført på klipp fra et merkeskilt. Man kan her se at metoden har problemer med å tolke utklipp nummer en og to, men klarer å tolke

utklipp nummer tre helt korrekt. Dersom man studerer resultatet fra utklipp nummer en litt nærmere, kan man se at programmet faktisk har klart å tolke tallene, problemet er bare at bokstavene og kommaet mellom tallene også er blitt tolket som tall.



Illustrasjon 21: Optisk tegngjenkjenning av merkeskilt klipp

I applikasjonen utviklet i denne oppgaven, er det derfor viktig at de forskjellige bildeoperasjonene utført på merkeskiltet før den optiske tegngjenkjennings prosessen, påvirker bildekvaliteten så lite som mulig.

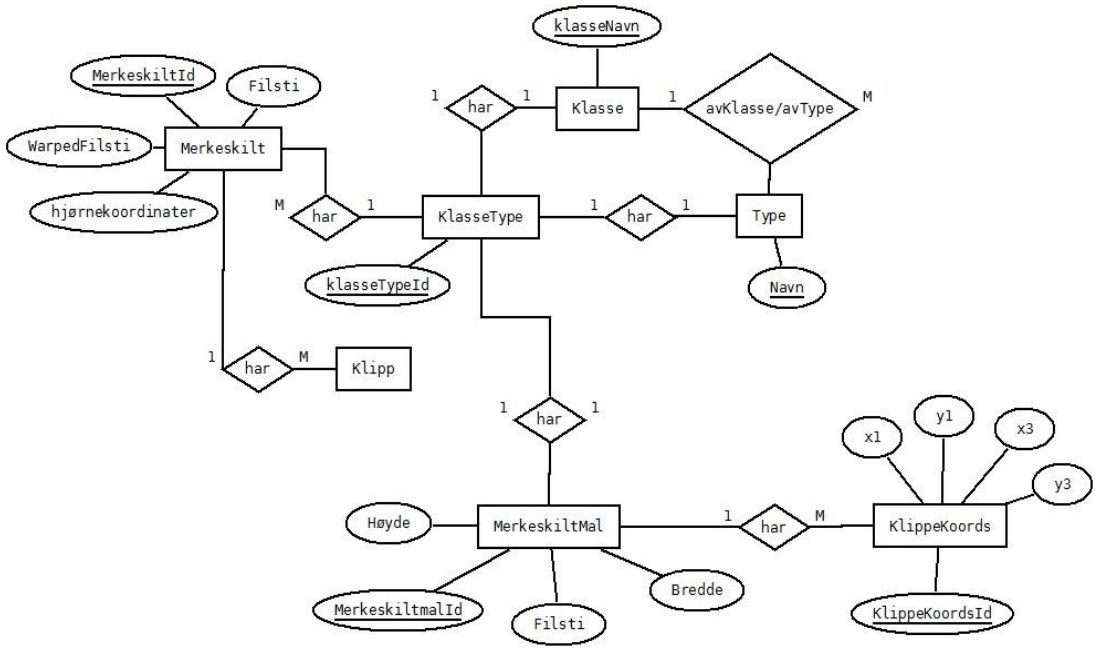
På grunn av faren for feiltolkning kombinert med kundens behov for 100% korrekt informasjon, er det nødvendig at applikasjonen muliggjør korrigere verdien som blir lest av den optiske tegngjenkjennings funksjonen. Når verdien skal legges inn i den endelige databasen, vil det også være nødvendig å legge inn en kvalitetskontroll der hvert tall gjennomgår en godkjennings prosess.

6 Implementering og designmessige vurderinger

6.1 Databasedesign

I skrivende stund benytter Verico AS en Access database til sin datafangst prosess, men de har også en MySQL database under arbeid. De to databasene er begge svært store og i mange tilfeller vanskelige å navigere i. I denne oppgaven har det derfor vært nødvendig å opprette en egen test database som programmet kan kjøre mot. Denne test databasen inneholder de tabellene som er nødvendige for å kunne kjøre programmet.

ER diagrammet til databasen er vist i figur 22.



Illustrasjon 22: ER diagram

For å gjøre programmet mest mulig uavhengig av databasen, er alle spørringer lagret i rutiner. På den måten kan programmet kommunisere med enhver database som har tilsvarende rutiner, og er aldri direkte i kontakt med databasen. Dette er en stor fordel, da dette gjør det mulig å koble programmet opp mot en database ved å opprette tilsvarende rutiner i databasen. Et eksempel på en slik rutine illustreres i figur 23.

```
1  -- --------------------------------------------------------------------------------
2  -- Denne rutinen tar som parameter klasseType variabelen til et merkeskilt og returnerer
3  -- merkeskilmalId-en til merkeskiltmalen som tilhører merkeskiltet.
4  -- --------------------------------------------------------------------------------
5  DELIMITER $$
6
7  CREATE DEFINER=`root`@`localhost` PROCEDURE `getMerkeskiltmalId`(IN merkeskiltKlasseType int, OUT merkeskiltmalIdOut int)
8  BEGIN
9  SELECT merkeskiltmal into merkeskiltmalIdOut
10 FROM `verifacta`.`klassetype`
11 WHERE klasseTypeId = merkeskiltKlasseType;
12 END
13
```

Illustrasjon 23: Eksempel på MySQL rutine

6.2 MatlabControl

For å kommunisere med MATLAB ved hjelp av matlabcontrol må man først sette opp en proxy som brukes til å kontrollere MATLAB. Dette gjøres på følgende måte:

```
MatlabProxyFactoryOptions options =
      new MatlabProxyFactoryOptions.Builder()
      .setUsePreviouslyControlledSession(true)
      .build();
MatlabProxyFactory factory = new MatlabProxyFactory(options);
proxy = factory.getProxy();
```

Deretter kan man kalle en .m funksjon ved hjelp av proxyens eval metode:

```
proxy.eval("addpath('"+mFilePath+"')");
Object[] arguments =
      proxy.returningEval("myfunc('"+imgPath.getAbsolutePath()+"')", 1);
Object firstArgument = arguments[0];
```

Variablene som returneres fra .m funksjonen lagres i "arguments" tabellen, og kan deretter hentes ut og brukes i Java applikasjonen.

6.3 Java applikasjonen

Datafangstapplikasjonen er implementert i Java, og består av 26 klasser organisert i 7 pakker, der hver pakke er knyttet til en konkret oppgave. De første fem pakkene er knyttet til de fem forskjellige bildeoperasjonene; lokalisering av skiltet, warping, klassifisering, klipping og OCR. En oversikt over disse fem pakkene vises under.

classification	**cutting**	**locateSign**	**ocr**	**warping**
Cl_ImagePanel	Cu_ImagePanel	LocateSign	OCR	BilinearMapping
Classification	Cutting.java	LS_ImagePanel	OCR_ImagePanel	Matrix
		LS_Manually	OCR_Matlab	Warping
		LS_Matlab		

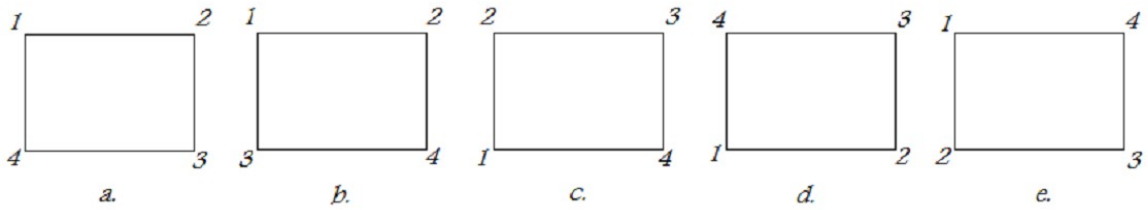
De to resterende pakkene vises under. Den første inneholder klasser knyttet til oppstart av applikasjonen, brukergrensesnittet og databasekommunikasjon. Den siste pakken inneholder alle objekter som benyttes i applikasjonen.

main	**objects**
DatabaseConnection	Cut
DataCaptureProgram	Fabrikat
LogoPanel	Input
MainPanel	Merkeskilt
RunProgram	Merkeskiltmal
SetUpPanel	Typebetegnelse
StartProcessPanel	

6.3.1 Lokaliser Skiltet

Lokalisering av skiltet i bildet består av to hoveddeler. Den ene er den manuelle lokaliseringen som klassen LS_Manually.java står for. Denne klassen viser bildet og venter på at alle fire hjørnene skal markeres. Når dette er gjort, lagres koordinatene for videre bruk i programmet.

En viktig oppgave for denne klassen er å registrere de fire hjørnekoordinatene. Dette blir gjort ved at brukeren med musen markerer hvert enkelt hjørne. En liten undersøkelse utført ved å spørre 20 tilfeldig valgte personer, viser at det ikke finnes en standard måte å nummerere hjørnene i en firkant på. I figur 24 vises de fem forskjellige metodene som ble foreslått.



Illustrasjon 24: Nummerering av hjørner

Fordelingen ble som følgende: 6 personer svarte at de ville valgt metode a, 6 personer ville valgt b, 2 ville valgt c, 2 ville valgt d og 1 ville valgt e.

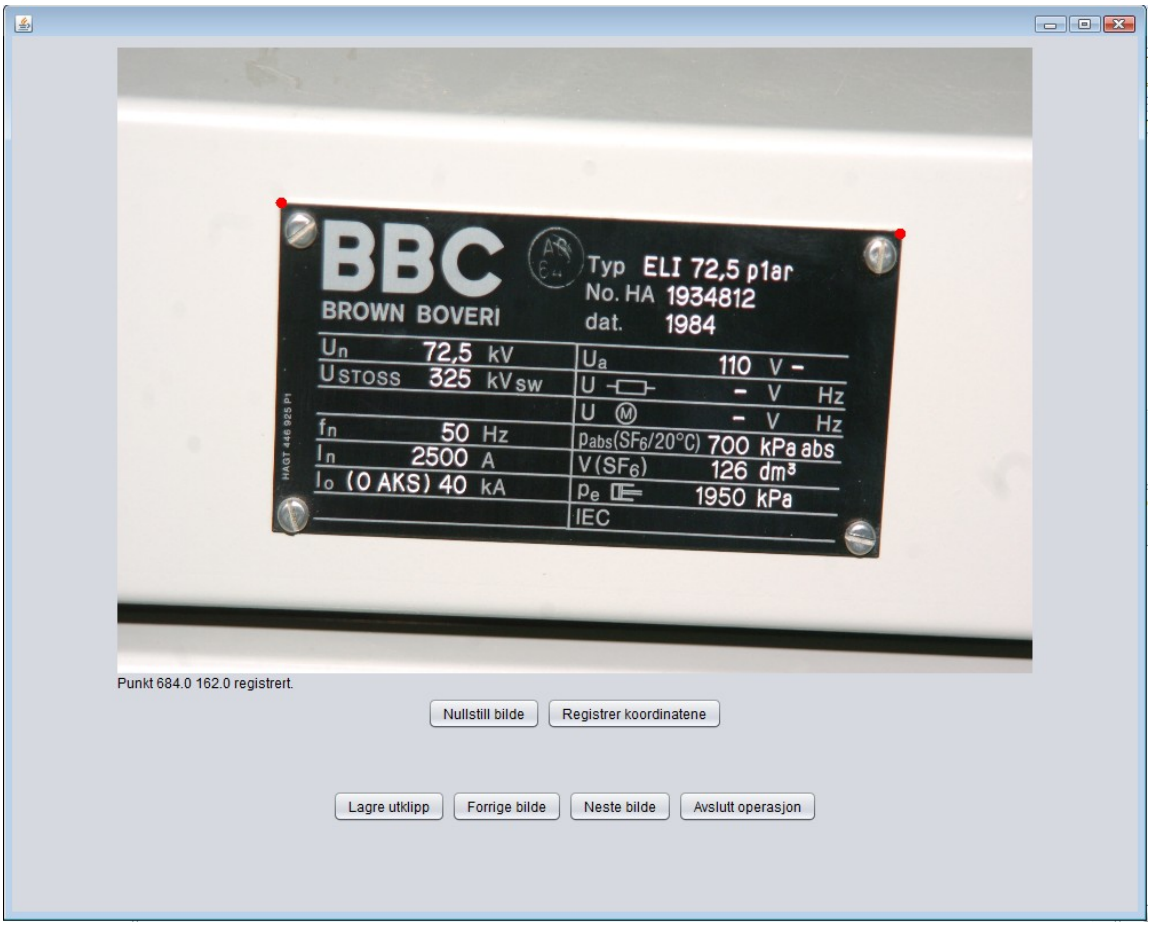
Av dette ser man at det er vanskelig å velge en standard på nummereringen, og løsningen ble derfor å utvikle en algoritme, som basert på de fire punktenes lokasjon i forhold til hverandre, bestemmer hvilket av hjørnene som skal knyttes til hvilket punkt.

```
for(i=0 : 4)
        for(j=1 : 4)
                Sammenlign koordinat i med koordinat j og sorter etter
                størrelsen på y-verdi.
        end
end
if (koordinat i plassering 1 sin x-verdi > koordinat i plassering 2 sin x-verdi)
        Bytt plass på de to verdiene
if (koordinat i plassering 3 sin x-verdi > koordinat i plassering 4 sin x-verdi)
        Bytt plass på de to verdiene
```

Dette implementeres i Java på følgende måte:

```
private void sortCoordinates() {

	Point2D temp;

	for(int i=0; i<srcCoords.length; i++){
		for(int j=i+1; j<srcCoords.length; j++){
			if(srcCoords[i].getY()>srcCoords[j].getY()){
				temp=srcCoords[i];
				srcCoords[i]= srcCoords[j];
				srcCoords[j]=temp;
			}
		}
	}

	if(srcCoords[0].getX()>srcCoords[1].getX()){
		temp=srcCoords[0];
		srcCoords[0]=srcCoords[1];
		srcCoords[1]=temp;
	}

	if(srcCoords[2].getX()>srcCoords[3].getX()){
		temp=srcCoords[2];
		srcCoords[2]=srcCoords[3];
		srcCoords[3]=temp;
	}
}
```

Et skjermbilde av den manuelle skilt lokaliseringen vises i figur 25.



Illustrasjon 25: Skjermbilde av manuell skiltlokalisering

Den andre hovedklassen i lokaliseringen er LS_Matlab.java som sender et kall til MATLAB med bildet som input og venter deretter på at de fire hjørnekoordinatene skal returneres. I denne versjonen av applikasjonen kaller LS_Matlab.java kun en dummy funksjon som returnerer fire forhåndsbestemte koordinater. Dette fordi metoden for automatisk lokalisering fremdeles er under arbeid. Denne klassen er derfor laget som en demonstrasjon på hvordan kommunikasjonen mellom applikasjonen og MATLAB funksjonen kan settes opp.

Både LS_Matlab.java og LS_manually.java kaller warping.java med bilde og hjørnekoordinater som input.

6.3.2 Warping

Warping.java er hovedklassen for "warping". Som nevnt kalles denne metoden for hvert skilt som blir lokalisert. Dette skiller seg fra de andre metodene som alle blir startet av brukeren.

Selve "warpingen" utføres av klassen bilineærMapping.java, der algoritmen fra kapittel 5.3 er implementert på følgende måte:

```
for(int x_=(int)dstPoints[0].getX(); x_<dstPoints[1].getX(); x_++){
      for(int y_=(int)dstPoints[0].getY(); y_<dstPoints[2].getY() ; y_++){
            int x = (int) (a1 * x_ + a2 * y_ + a3 * x_ * y_ + a4);
            int y= (int) (b1 * x_ + b2 * y_ + b3 * x_ * y_ + b4);
            int pixelValue = inputImage.getRGB(x, y);
            outputImage.setRGB(x_, y_, pixelValue);
      }
}
```

Der de åtte koeffisientene (a1 ... a4 og b1 ... b4) beregnes fra koden under. Og matriseoperasjonene utføres av klassen Matrix.java.

```
public void findCoeffisients(
      Point2D src1, Point2D src2, Point2D src3, Point2D src4,
      Point2D dst1, Point2D dst2, Point2D dst3, Point2D dst4)   {

      Matrix X =
            new Matrix(new double[][] {{src1.getX()},{src2.getX()},
            {src3.getX()},{src4.getX()}});
      Matrix Y =
            new Matrix(new double[][] {{src1.getY()},{src2.getY()},
            {src3.getY()},{src4.getY()}});

      Matrix M = new Matrix(new double[][]{
            {dst1.getX(), dst1.getY(), dst1.getX() * dst1.getY(), 1},
            {dst2.getX(), dst2.getY(), dst2.getX() * dst2.getY(), 1},
            {dst3.getX(), dst3.getY(), dst3.getX() * dst3.getY(), 1},
            {dst4.getX(), dst4.getY(), dst4.getX() * dst4.getY(), 1}
            });


      Matrix A = M.findUnknown(X);     //Calulates X = inverseM*A
      Matrix B = M.findUnknown(Y);     //Calulates Y = inverseM*B

      a1 = A.getValue(0,0);            b1 = B.getValue(0,0);
      a2 = A.getValue(1,0);            b2 = B.getValue(1,0);
      a3 = A.getValue(2,0);            b3 = B.getValue(2,0);
      a4 = A.getValue(3,0);            b4 = B.getValue(3,0);
}
```

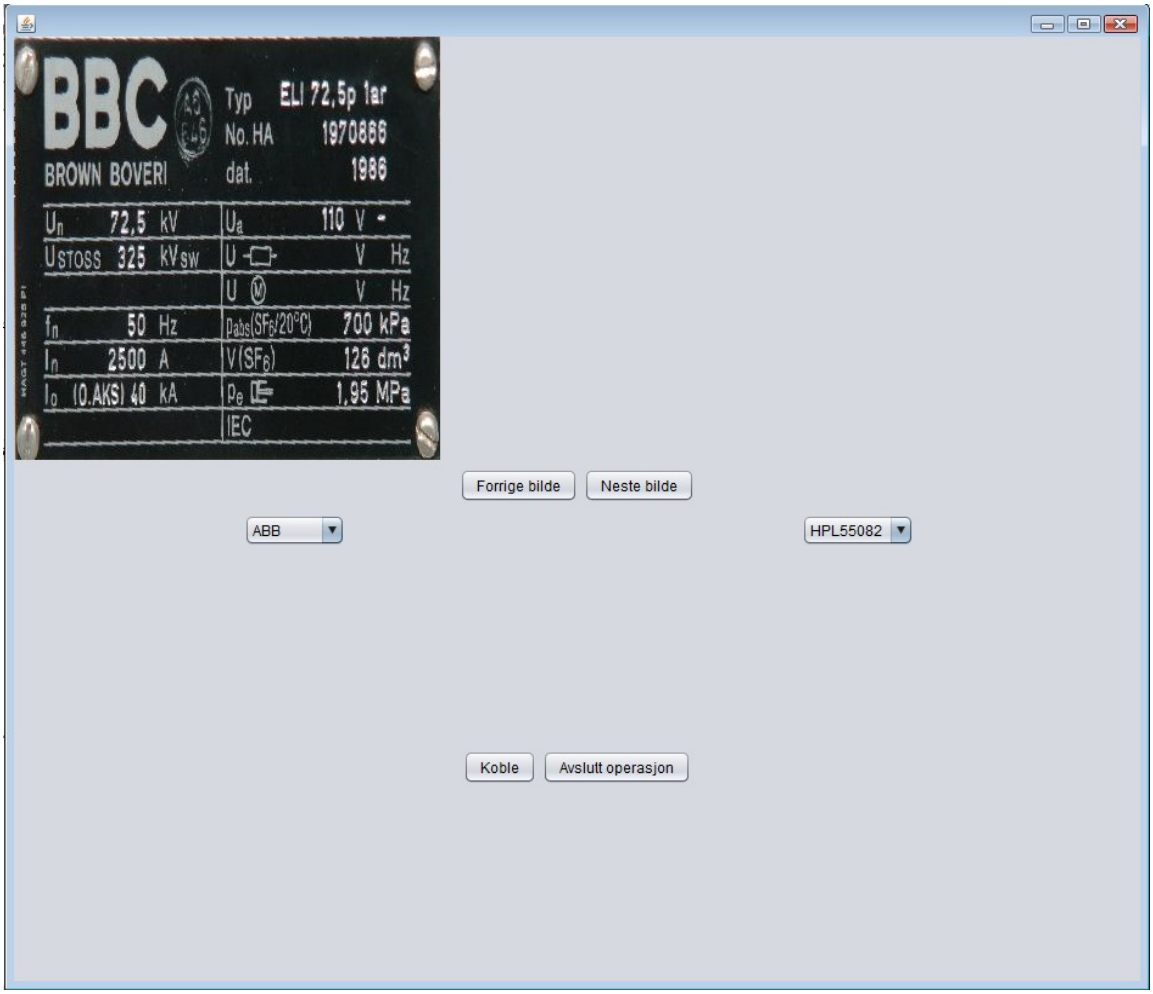
6.3.3 Klassifisering

MATLAB funksjonen for klassifisering er fremdeles under arbeid, og da denne vil bli kalt på samme måte som "lokaliser skilt" funksjonen, er det ikke implementert noe kall til en dummy funksjon i dette tilfellet.

Fokuset har istedet vært lagt på å implementere en alternativ klassifisering til de tilfellene der automatisk klassifisering ikke har kommet frem til et konkret resultat.

Klassen Classification.java, står for den manuelle klassefiringen ved å vise merkeskiltet sammen med alternativer for fabrikat og typebetegnelser. Brukeren kan dermed studere merkeskiltet og manuelt klassifisere dette. Fabrikat og typebetegnelse er knyttet sammen på en slik måte at når et fabrikat blir valgt, vises kun de typebetegnelsene som knyttes til dette konkrete fabrikatet.

I figur 26 vises skjermbildet for klassifiseringen. Merkeskiltet er her på samme kvadratiske form som kreves for den automatiske klassifiseringen. Dette ble vurdert til å være det mest praktiske, da merkeskiltene er tatt i forskjellige vinkler og på forskjellige avstander, noe som betyr at dersom man foretok en gjennomsnittlig beregning for å bestemme form på det "warpede" bildet, ville man få svært mange forskjellige resultater for skilt av samme fabrikat og type.



Illustrasjon 26: Skjermbilde for klassifisering

6.3.4 Klipping

Klippingen utføres ved at en klippemal legges over merkeskiltet og deretter justeres dersom dette er nødvendig. Gjennom klassifiseringen blir hvert merkeskilt koblet til en merkeskiltmal, og i klippingen brukes denne malen til å bestemme størrelsen på merkeskiltet. Deretter hentes klippemalen som hører til merkeskiltmalen og denne legges over merkeskiltet. Klippemalen er lagret i databasen som koordinater knyttet til et merkeskilt og tegnes opp som rektangler ved hjelp av Java biblioteket Graphics 2D.

Noen ganger kan man oppleve at klippemalen ikke treffer slik at alle tegn er innenfor rammen, og det er derfor nødvendig å kunne justere malen slik at man slipper å gå tilbake for

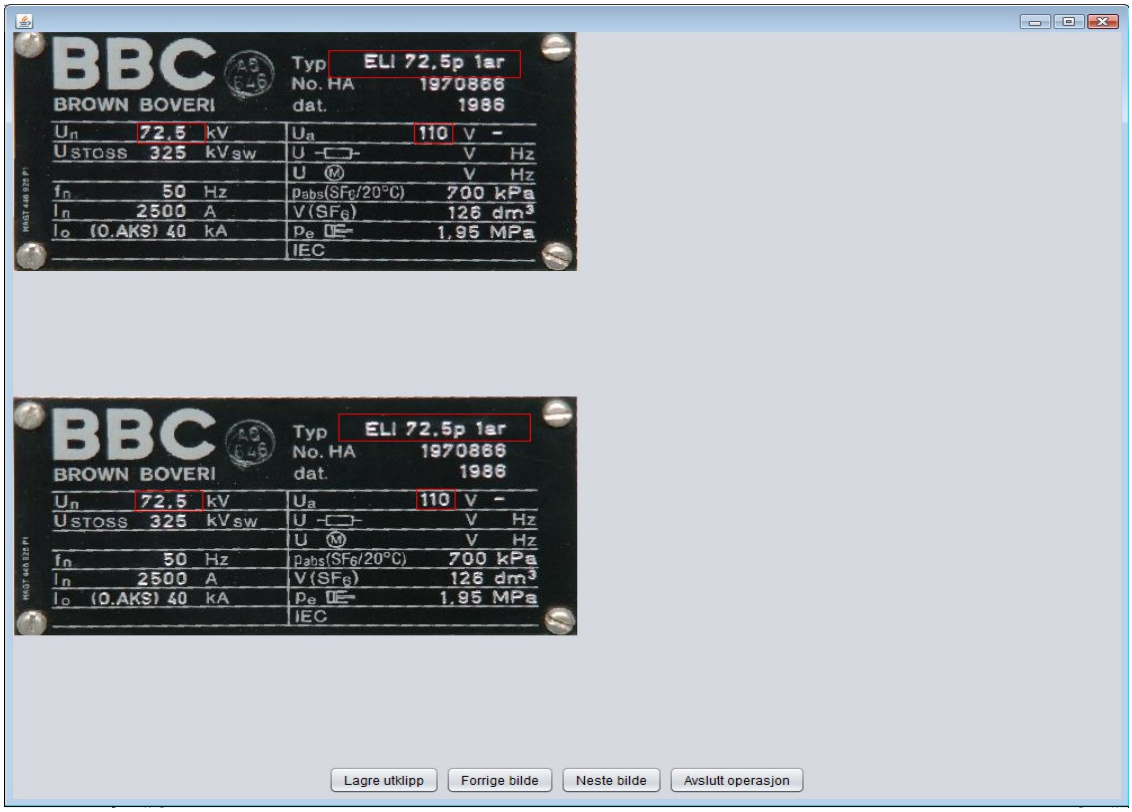
å korrigere tidligere prosesser. Dette løses ved at applikasjonen lar brukeren klikke på rektanglene og manuelt flytte disse etter behov. Java koden som sørger for å tegne opp klippemalen på skiltet er som følger:

```
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		Graphics2D g2 = (Graphics2D)g;

		g.drawImage(image,0,0, null);
		g.setColor(Color.red);
		for(int i=0; i<cutList.size(); i++){
			g2.draw(cutList.get(i));
		}

		g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
				RenderingHints.VALUE_ANTIALIAS_ON);

		if(isMerkeskilt){
			for(int i=0; i<cutList.size(); i++){
				int[] temp = new int[4];
				Rectangle r = cutList.get(i);
				temp[0] = (int)r.getX();
				temp[1] = (int)r.getY();
				temp[2] = (int)(r.getX()+r.getWidth());
				temp[3] = (int)(r.getY()+r.getHeight());
				cuts.get(i).setCutCoords(temp);
				merkeskilt.setCuts(cuts);
			}
		}
```

For å kunne justere klippemalen brukes bibliotekene MouseListener og MouseMotionListener. Et utdrag av koden for denne justeringen vises under.

```
	public void mousePressed(MouseEvent ev) {
		for(int i=0; i<cutList.size(); i++){
			if(isClicked(ev.getX(), ev.getY(),cutList.get(i))){
				Point p = ev.getPoint();
				Rectangle r = cutList.get(i);
				if(r.contains(p)) {
					rect=r;
					offset.x = p.x - r.x;
					offset.y = p.y - r.y;
					dragging = true;
				}
			}
		}
	}

	public void mouseDragged(MouseEvent ev) {
		if(dragging) {
			int x = ev.getX() - offset.x;
			int y = ev.getY() - offset.y;
			setRect(x, y);
		}
	}

	public void mouseReleased(MouseEvent ev) {
		dragging = false;
	}

	public boolean isClicked(int x, int y, Rectangle rect){
		if(x >= (int)rect.getX() && x < ((int)rect.getX() +
				(int)rect.getWidth()) && y >= (int)rect.getY() &&
				y< ((int)rect.getY() + (int)rect.getHeight())){
			return true;
		}
		return false;
	}
```

Figur 27 viser skjermbildet for klippingen. Øverst ser man merkeskiltmalen med tilhørende klippemal, og under vises merkeskiltet med samme klippemal. I noen tilfeller kan merkeskiltets utseende avvike fra merkeskiltmalen på en slik måte at klippemalen blir betydelig feilplassert. I slike tilfeller er det viktig for brukeren å kunne se merkeskiltmalen med tilhørende klippemal lagt over, slik at vedkommende manuelt kan justere malen på merkeskiltet til den klipper de riktige verdiene. Dette fordi brukeren av programmet mange ganger er en person uten kjennskap til det tekniske aspektet ved datafangsten.



Illustrasjon 27: Skjermbilde for klipping

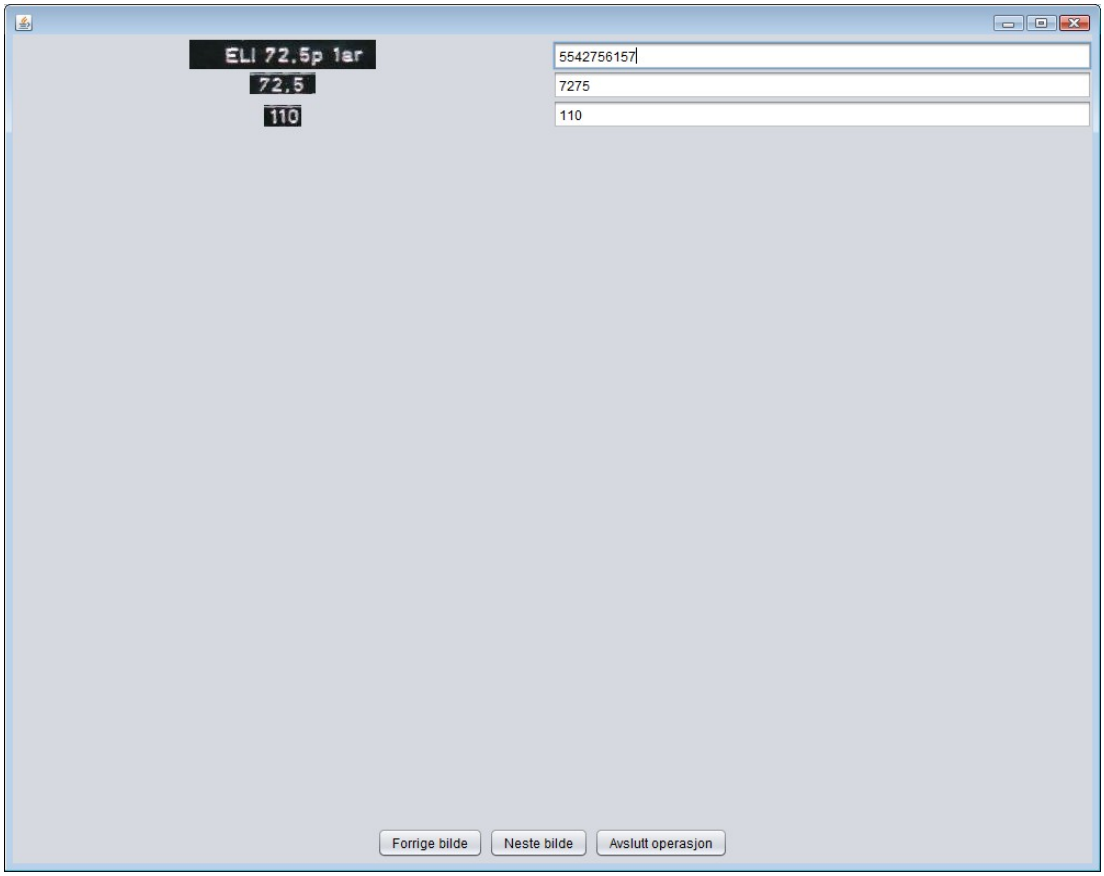
6.3.5 OCR

Klassen OCR_Matlab.java kaller MATLAB funksjonen som utfører den optiske tegngjenkjenningen, og gir filstien til de klippede verdiene som input til denne. MATLAB funksjonen returnerer deretter de gjenkjente verdiene. Dette gjøres i klassens konstruktør som vises under. Den returnerte verdien lagres her i variabelen "result".

```
public OCR_Matlab(String mFilePath, File imgPath) throws
      MatlabConnectionException, MatlabInvocationException{

      MatlabProxyFactoryOptions options =
                  new MatlabProxyFactoryOptions.Builder()
                  .setUsePreviouslyControlledSession(true)
                  .build();
      MatlabProxyFactory factory = new MatlabProxyFactory(options);
      proxy = factory.getProxy();

      proxy.eval("addpath('"+mFilePath+"')");

      Object[] arguments =
            proxy.returningEval("ocr1('"+imgPath.getAbsolutePath()+"')", 1);

      Object firstArgument = arguments[0];
      result = ((String[]) firstArgument)[0];

      proxy.disconnect();
}
```

Både bildene og verdiene vises så i et panel, slik at brukeren kan kontrollere at disse er riktige. Det gjenstår fremdeles en del arbeid for at den optiske tegngjenkjenningen skal fungere optimalt i applikasjonen, spesielt i forhold til kvaliteten på input-bildene. Dette drøftes nærmere i kapittel 7 og 8.3.

I figur 28 vises de klippede verdiene med de tilhørende verdiene gitt av den optiske tegngjennkjennings funksjonen.



Illustrasjon 28: Skjermbilde av den optiske tegngjenkjenningen.

7 Vurdering av resultater

Målet med en automatisering av datafangstprosessen er å oppnå en tidsbesparelse i forhold til eksisterende prosedyre. Det er derfor av interesse å foreta en vurdering av tidsbesparelsene prosessene, utviklet eller implementert i denne oppgaven, har gitt.

- **Lokalisering av skilt**
  - Denne funksjonen finnes ikke i den nåværende prosedyren og er et nytt trinn, nødvendig for å kunne utføre de automatiserte versjonene av klassifiseringen og klippingen.
  - MATLAB funksjonen er som nevnt fremdeles under utvikling, men det manuelle alternativet, utviklet i denne oppgaven tar i gjennomsnitt 60 sekunder per merkeskilt. Dersom man regner en gjennomsnitt på 5 variabler per merkeskilt, utgjør dette 10 sekund per variabel.

- **Warping**
  - "Warpingen" er også en funksjon som ikke eksisterer i den nåværende prosedyren, men som er nødvendig for å få merkeskiltene på den formen som kreves av klassifiserings og klippe prosessen.
  - "Warping" utføres i sammenheng med lokalisering av skiltet og bruker derfor ikke ekstra tid utover dette.

- **Klassifisering**
  - MATLAB funksjonen for klassifiseringen er som nevnt ikke ferdig utviklet, men dersom man regner med at omtrent halvparten av merkeskiltene kan klassifiseres av denne, og at resterende benytter den alternative manuelle metoden utviklet i denne oppgaven, ender man opp med å redusere klassifiseringstiden fra 30 sekunder til 20 sekunder per variabel.

- **Klipping**
  - I den nåværende prosedyren klippes alle variablene hver for seg. I applikasjonen utviklet i denne oppgaven klippes, derimot alle variabler i samme operasjon. Dersom man beregner 5 variabler per merkeskilt, fører dette til at gjennomsnittlig klippetid reduseres fra 20 sekunder per variabel til 5 sekunder per variabel.

- **Optisk tegngjenkjenning**
  - Treffprosenten på denne funksjonen er foreløpig ikke spesielt god. Dette fordi kvaliteten på bildene ikke er høy nok til at funksjonen fungerer optimalt. Dersom vi regner med at 10% av variablene gjenkjennes, reduseres tastetiden fra 10 til 9 sekunder per variabel. Potensialet for funksjonen er en treffsikkerhet på 98% dersom bildene er av høy kvalitet. Dersom man kan forbedre kvaliteten på merkeskilt klippene bør det derfor være mulig å oppnå en treffprosent på omtrent 50% i denne applikasjonen. Tidsbruken på kvalitetsjekkingen av de inntastede verdiene vil være den samme for både manuell inntasting og optisk tegngjenkjente verdier.

8 Videreutvikling

Dette kapittelet tar for seg forslag til videreutvikling av datafangst applikasjonen. Både når det kommer til utforming av brukergrensesnitt og til de tekniske løsningene.

8.1 Fullstendig implementering av MATLAB funksjoner

Av de forskjellige funksjonene som skal automatisere datafangsten, er to av dem fremdeles under arbeid, nemlig lokalisering av skilt og klassifisering. Av den grunn er det i denne applikasjonen kun implementert en dummy versjon av skilt lokaliseringen, for å undersøke hvorvidt matlabcontrol er en fornuftig teknologi å benytte. I videreutviklingen av denne applikasjonen, kan det derfor være fornuftig å vurdere om man skal konvertere MATLAB funksjonene til Java kode slik at man slipper å kjøre MATLAB i bakgrunnen. Dette fordi kallene fra Java til MATLAB tar noe lengre tid en metodekall inne i applikasjonen. Man vil også slippe å ha MATLAB installert på maskinen som skal kjøre applikasjonen. Dette er en stor fordel, da programmet skal installeres og kjøres på mange forskjellige maskiner, og matlabcontrol leverer foreløpig ingen løsning for å kalle MATLAB over nettverk. Det kan likevel være nyttig å bruke matlabcontrol inntil det punktet der applikasjonen er klar for kommersielt bruk.

8.2 Forbedring av brukergrensesnitt

I denne oppgaven har fokuset vært på å undersøke hvorvidt en automatisering av datafangstprosessen er mulig å gjennomføre med de funksjonene som er, eller er i ferd med å bli utviklet. Med dette som utgangspunkt har derfor ikke brukergrensesnittet for applikasjonen blitt prioritert.

En forbedring av brukergrensesnittet er nødvendig for å få en mer brukervennlig applikasjon. Dette er spesielt viktig, da Verico AS ansetter folk utenfra for å bidra i datafangstprosessen, noe som vil si at brukerne av applikasjonen ikke alltid har innsikt i det tekniske aspektet ved datafangsten. Det er derfor nødvendig å forbedre grensesnittet på en slik måte at brukeren kan utføre sin arbeidsoppgave mest mulig på egenhånd, uten å måtte få en grundig innføring i prosessen. Applikasjonen må gjøres med eller mindre selvforklarende.

Forbedring av brukergrensesnittet er også viktig for å effektivisere datafangstprosessen. Et forslag til en slik forbedring, er å redusere antall tastetrykk brukeren må gjennom i hver prosess. Når MATLAB funksjonene er blitt implementert i applikasjonen, bør det være mulig for brukeren å raskt kunne gå gjennom resultatene for å kvalitetssikre disse med så få tastetrykk som mulig.

8.3 Optimalisering av den optiske tegngjenkjenningen

Bildene som benyttes i datafangstprosessen er i utgangspunktet av høy kvalitet, men etter å ha gjennomgått en "warping" er de ikke lenger like skarpe. I tillegg er noen av bildene tatt på en relativt stor avstand fra selve skiltet, noe som fører til at tegnene som skal leses ikke alltid er like tydelige. Noen av skiltene holder også en svært dårlig kvalitet i utgangspunktet.

Alt dette utgjør tilsammen en stor utfordring for den optiske tegngjenkjenningen, noe som igjen fører til at man ikke får den suksess prosenten man kunne ønske.

En videreutvikling av applikasjonen bør derfor fokusere på å forbedre kvalitetene på bildene etter "warping". Enten ved å forbedre selve "warping" metoden, eller ved å utføre forskjellige bildeoperasjoner som for eksempel filtrering på de "warpede" bildene.

Den optiske tegngjenkjennings funksjonen som benyttes i denne oppgaven kan foreløpig kun behandle tall. Denne bør videreutvikles til også å kunne behandle bokstaver og tegn, da dette ofte forekommer på merkeskiltene.

8.4 Kvalitetskontroll

Som nevnt utføres kvalitetskontrollen i dag ved at hver verdi manuelt kontrolleres. Noen av verdiene ligger allerede lagret av kunden, og dersom man sjekker innlest/inntastet variabel mot denne, og aksepterer dette som en kvalitetskontroll vil man kunne redusere prosesseringstiden ytterligere.

8.5 Andre funksjoner

Applikasjonen er utviklet på en slik måte at det skal være enkelt å utvide den med flere funksjoner dersom dette skulle bli nødvendig. Under følger tre eksempler på deler av datafangstprosessen som kan forbedres.

- I skrivende stund sorteres alle bildene i mapper ved hjelp av "drag-and-drop" metoden. I tillegg blir bildene samtidig gjennomgått, og de ubrukelige bildene kassert. Dette er svært tidkrevende, og det kan være en ide å undersøke om det er mulig å lage en metode for å effektivisere denne delen av prosessen.
- I noen tilfeller er merkeskiltene plassert slik at det er umulig å fotografere hele skiltet fra en vinkel. Det er da nødvendig å ta flere bilder slik at man kan klippe noen variabler fra ett bilde, og noen fra et annet. Dersom det hadde vært mulig å utvikle en metode, som ved hjelp av bildeoperasjoner, kan sette sammen deler av forskjellige merkeskilt til et fullstendig skilt, vil dette være svært nyttig.
- Noen ganger er merkeskiltene plassert på sylindere, noe som gjør at skiltene krummer. Resultatet av "warpingen" vil derfor ikke være tilfredsstillende, og det kan være fornuftig å vurdere hvorvidt man skal forbedre "warping" funksjonen, slik at den kan håndtere dette spesial tilfellet. I midlertidig er det gjennomsnittlig svært få skilt som har en slik krumming. Det er derfor muligens en bedre ide å fokusere på å forberede den manuelle prosessen.

9 Konklusjon

I denne oppgaven er det først blitt utarbeidet en oversikt over de forskjellige prosessene som tilsammen skal utgjøre en automatisering av datafangstprosessen. Det er blitt gjort vurderinger av hvilke funksjoner som allerede eksisterer, hvilke som er under arbeid, og hvilke som fremdeles må utarbeides for at man skal kunne foreta en automatisering.

Den utarbeidede oversikten er blitt brukt som utgangspunkt for utviklingen av en applikasjon som kombinerer de forskjellige funksjonene til en sammenhengende prosess. Denne applikasjonen automatiserer helt, eller delvis de forskjellige prosessene og tilbyr manuelle alternativer der dette er nødvendig.

De forskjellige prosessene er:

Lokalisering av merkeskilt i bildet

MATLAB funksjonene for dette er under arbeid, og det er derfor satt opp en kobling mot denne funksjonen ved hjelp av matlabcontrol. I tillegg er det utviklet et manuelt alternativ til de tilfellene der MATLAB funksjonen ikke gir det ønskede resultatet.

Warping

En metode for å "warpe" bildene er blitt utviklet. Denne metoden benytter bilineær mapping til å "warpe" skiltene i bildene til et kvadrat eller et rektangel med en bestemt størrelse. Denne "warpingen" er nødvendig for at det skal være mulig å klassifisere og klippe merkeskiltet.

Klassifisering

MATLAB funksjonen for dette er under arbeid, og det er derfor utviklet en manuell kode til klassifiseringen som kan benyttes i de tilfellene der klassifiseringen er mislykket eller gir flere alternativer som resultat.

Klipping

En metode for å klippe et klassifisert merkeskilt er utarbeidet. Metoden legger en klippemal over merkeskiltet og lar brukeren manuelt justere malen, dersom dette er nødvendig. Deretter klippes alle variabler i en enkelt operasjon.

Optiske tegngjenkjenning

MATLAB funksjonen for optisk tegngjenkjenning er allerede utviklet. Det er derfor satt opp en kobling til denne ved hjelp av matlabcontrol.

Med løsningen Verico benytter i dag, bruker de i gjennomsnitt 190 sekund per variabel. Disse er fordelt på følgende måte:

- Materiellkobling (sek pr.) 30
- Merkeskiltdefinering (sek pr.) 120
- Klippetid (sek pr.) 20
- Tastetid (sek pr.) 10
- QA-tid (sek pr.) 10

Der "materiellkobling" er klassifiseringen av skiltet, og "merkeskiltdefinering" er prosessen der merkeskiltmalen defineres.

Med applikasjonen utviklet i denne oppgaven, er forventet tidsbruk dersom alle funksjoner ferdigstilles som følger:

- Lokalisering av skilt (sek pr.) 5
- Klassifisering (sek pr.) 20
- Klippetid (sek pr.) 5
- Tastetid (sek pr.) 9
- QA-tid (sek pr.) 10

Dette gir gjennomsnittlig 49 sekunder per variabel. Det er da beregnet at man benytter samme kvalitetskontroll som i den eksisterende prosedyre.

På bakgrunn av dette kan man konkludere med at dersom Verico AS velger å ferdigstille og benytte løsningene utarbeidet i denne oppgaven, vil det være mulig å effektivisere datafangstprosessen med rundt 75%.

Vedlegg

A Oversikt over innhold på vedlagt CD

Mappestruktur:

Datafangstapplikasjonen/

Kildekode/ *Kildekoden til datafangstapplikasjonen*

KjørbarApplikasjon/ *Kjørbar .jar fil.*

MerkeskiltMaler/ *Merkeskitlmal bildene*

Merkeskilt/ *Merkeskilt bildene*

Database/ *Databasen som .sql fil*

MATLAB/

Loklisering/ *MATLAB funksjon for lokalisering av skilt.*

OptiskTegngjenkjenning/ *MATLAB funksjonene knyttet til den optiske tegngjenkjenningen*

Installasjonsfiler/ *Filer nødvendig for installasjon*

Rapport.pdf *Denne rapporten*

B Installasjons veiledning

- **MySQL**

  For å kunne kjøre datafangstapplikasjonen må man først installere MySQL.
  Instalasjonsfilen finner man ved å følge linken
  http://www.mysql.com/downloads/installer/

- **Importering av database**

  Databasen ligger vedlagt på CDen i filen Database.sql.
  Denne importeres ved at man i ledeteksten først oppretter en tom database ved navn verifacta:

  - Start MySQL ledeteksten:

    C:>mysql -u «user» -p

    Der «user» er ditt brukernavn. Oppgi passord når dette bes om.
  - Opprett databasen:

    mysql> CREATE DATABASE verifacta

    mysql> Exit

  Deretter importeres databasen

  - C:>mysql - «user» -p verifacta < Database.sql

    Der «user» er ditt brukernavn. Oppgi passord når dette bes om

- **Java SE Development Kit 7**

  Deretter må Java SE Development Kit 7 installeres dersom dette ikke allerede er installert på maskinen.
  Filen for installering ligger på CDen, velg den som passer for operativsystemet.
  jdk-7u5-windows-i586 for 32 bit eller jdk-7u5-windows-x64 for 64 bit.

- **MATLAB**

  Dersom man ønsker å kjøre test mor MATLAB, må dette installeres.
  Dummy funksjonen for lokalisering av skilt og funksjonen
  for optisk tegngjenkjenning ligger på CDen.
  Filstien til MATLAB må da endres i klassene OCR.java og StartProcessPanel.java.

Referanser

[1] C.A. Glasbey and K.V. Mardia, A review of image warping methods *http://www.bioss.ac.uk/people/chris/warp.pdf*

[2] Paul S. Heckbert, Fundementals of Texture Mapping and Image Warping *http://www.cs.cmu.edu/~ph/texfund/texfund.pdf*

[3] Wilhelm Burger og Mark J. Burge, principles of digital image processing, core Algorithms *http://www.scribd.com/doc/58004056/143/Bilinear-Mapping*

[4] Andreas Waal, Optical Character Recognition(OCR) on Electrical Specification Plates

[5] Fritz Albregtsen, Geometriske operasjoner http://www.uio.no/studier/emner/emner/matnat/ifi/INF2310/v12/undervisningsmaterial e/forelesningsnotater/INF2310-2012-03-GeometriskeOperasjoner.pdf

[6] MatlabControl, A Java API to interact with MATLAB *http://code.google.com/p/matlabcontrol/*

[7] MATLAB builde JA *http://www.mathworks.se/products/javabuilder/description1.html*

[8] *http://encyclopedia2.thefreedictionary.com/texture+map*