Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Robotteknologi og signalbehandling	Vårsemesteret, 2020. Åpen / Konfidensiell
Forfatter: Axel Hansen Grøvdal	
Fagansvarlig: Morten Mossige Veileder(e): Morten Mossige	
Tittel på masteroppgaven: Generering av bane for lakkeringsrobot ved bruk av datasyn Engelsk tittel: Automatic trajectory planning using computer vision for spray paint robot	
Studiepoeng: 30	
Emneord: Robot Automasjon Computer vision	Sidetall: 54 + vedlegg/annet: 3 og .zip fil Stavanger, 13/07/2020 dato/år

Masteroppgave i Robotikk og Signalbehandling

GENERERING AV BANE FOR LAKKERINGSROBOT VED BRUK AV DATASYN

Axel Hansen Grøvdal

Universitetet
i Stavanger

13. juli 2020

Sammendrag

I dag produseres det mange varer i ulike fasonger som trenger ett lag eller to med spraylakk. Denne lakken kan forlenger levetiden til produktet, bidra til sikkerhet ved bruk av sterke farger, eller rett og slett la produktet fremstå mer estetisk. Det er mange grunner til å lakkere. Produkter finnes i alle forskjellige former og fasonger. Noen produkt er tre-dimensjonale, andre er tilnærmet flate. Derfor finnes forskjellige metoder for lakkering. Form og funksjoner spiller en viktig rolle for hvilken metode som benyttes for lakkeringen. En bilprodusent krever høy kvalitet for at bilen skal fremstå estetisk, mens en spade produsent stiller ikke de samme kravene. Der er det kanskje viktigst at lakken bidrar til overflate som beskytter metallet under for korrosjon.

En ulempe med lakkeringen er at det brukes en del resurser og tid for å få jobben gjort. Lakkeringsroboten ble derfor laget for å kutte ned på denne tidsbruken. En ulempe med roboten er at dens bane må planlegges. Når mange objekter med samme fasong lakkeres, brukes lite tid per objekt på bane-planleggingen. Når objektet er unikt, vil planleggingen ta mye tid.

Denne oppgaven ser på muligheten for å bruke data-syn kamera for å planlegge lakkeringsbane for tilnærmet flate objekt. Det vil ikke spille noen rolle om objektet er unikt eller ikke. Programmet som fremstås i denne oppgaven tar inn ett bilde av objektet som skal lakkeres, og lager banen utifra dette bildet. Ett kamera bidrar ikke med samme nøyaktighet som en bane laget av en tre-dimensjonell figur. Dette fremstår derfor som ett lavkost alternativ for produsenter som vektlegger sparing av kostnader på betraktning av tid.

Innhold

Kapittel 1

Introduksjon

Denne oppgaven vil se på en metode for å generere bane for lakkeringsrobot. Oppgaven bruker ett enkelt bilde fra ett datasyn kamera til å generere robot-banen. Denne banen er automatisk generert med tanke på den geometriske formen til objektet som skal lakkeres. Siden bare ett kamera benyttes, begrenses oppgaven til objekter som er flate eller tilnærmet flate.

Denne oppgaven er tillegnet en bruker som har stor variasjon av produkter som skal lakkeres. I industrien brukes blant annet cad-styrt lakkering for automatisk genererte lakkeringsbaner [1]. Ulempen med denne typen er at det kreves en cad fil for hvert objekt som skal lakkeres. Om hvert objekt som lakkeres er unikt, må det mates inn cad fil for hvert enkelt objekt, og det kan være tidkrevende. Metoden som skildres i denne oppgaven bruker ett enkelt bilde av objektet, og lager banen utifra dette. Genereringen av banen vil derfor kreve ingen annen informasjon om objektet enn akkurat ett bilde fra ett data-syn kamera.

1.1 Oppgave definisjon

Denne oppgaven går ut på å lage ett program som basert på datasyn, genererer robot-bane for lakkering av flater eller tilnærmet flate objekt. Denne oppgaven går ut på å lage ett program som tar inn ett tersklet bilde og gir ut koordinater som roboten skal gå etter. I inngangsbildet er objektet som skal lakkeres skilt ut, og bildet er fra ett kalibrert kamera.

Produktet av programmet skal være koordinater roboten skal gå etter for å lakkere objektet.

1.2 Kort om kapitlene

Kapittel 1: Informasjon rundt oppgaven

Kapittel 2: Bakgrunn. Bakgrunn og relevant teori rundt oppgaven.

Kapittel 3: Metoder og Implementering. Metoder handler om hvordan oppgaven blir strukturert og hvorfor de forskjellige valgene er tatt. Implementeringen går ut på hvordan de forskjellige algoritmene er skrevet.

Kapittel 4: Viser frem resultater og en diskusjondel rundt de forskjellige resultatene. Snakker også om videre arbeid.

Kapittel 5: Konklusjon av oppgaven

Kapittel 2

Bakgrunn

Den første lakkeringsroboten kom til verden i 1969, utviklet av Trallfa, nå ABB på Jæren. Banen til roboten ble da lært ved å føre roboten i banen den skulle følge. Banen ble da lagret på ett magnetbånd som kunne spilles av senere [2]. Slik ble den første roboten tatt i bruk for lakkering. Nå i dag har teknologien utviklet seg. Denne oppgaven ser på hvordan generere baner ved hjelp av kamera.

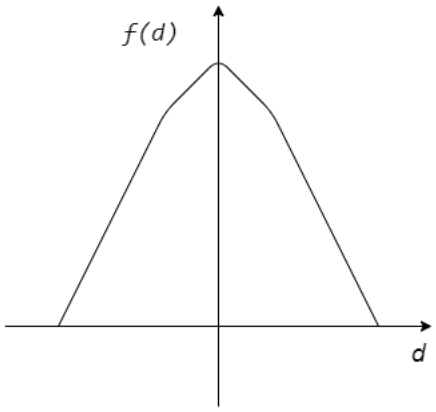
Dette kapittelet vil fokusere på spraylakkering og teknologier som brukes i sammenheng med å generere baner ved hjelp av kamera. Spraylakkeringen biten handler om hvordan denne oppgaven ser på optimal spraylakkeringsbane [1]. Resten av kapittelet er teori som baserer seg på siterte algoritmer og kjente metoder.

2.1 Spray maling

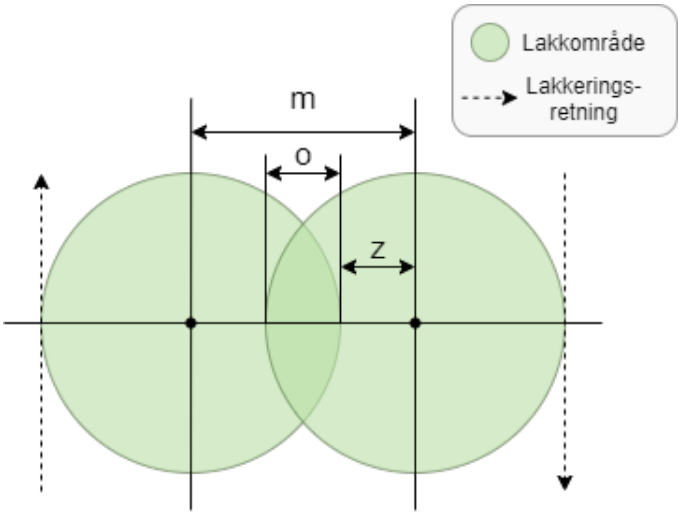
Lakkering med spray maling er å sende fargepartikler til en overflate med hjelp av lufttrykk. Målet med en god lakkeringsjobb er og få et så jevnt dekke av lakkeringsmedie som mulig. For å få til dette er det en del variabler som må være like i løpet av lakkeringen. Disse variablene kan være, optimal avstand mellom hvert strøk, jevn hastighet på strøk, likt trykk i lakkpistol med mer.

2.1.1 Lakkeringskjegle

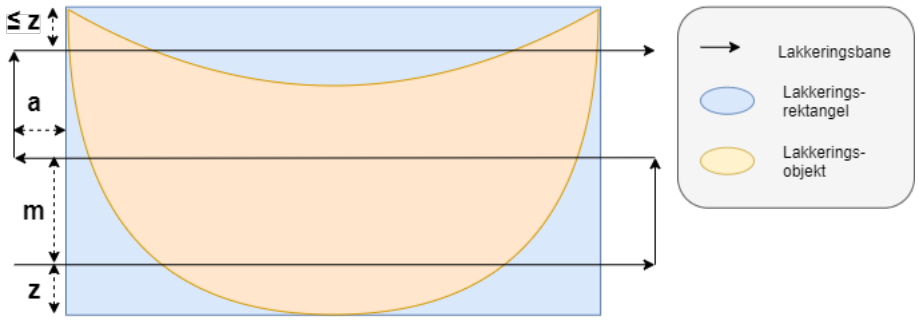
Nå er det slik at det er forskjellige typer maling-pistoler som har sine egenskaper. Disse forskjellige egenskapene gir forskjellig lakkeringskjegle. Figur 2.1 viser en generisk lakkeringskjegle. Figuren viser at $f(d)$ som er mengde lakk, er høgest konsentrert rundt senter av strålen. Ved større avstand d fra senter, jo mindre lakk havner i det området.



Figur 2.1: Generisk lakkeringskjegle: f(d) viser mengde lakk, d viser avstand fra senter av lakkeringsstråle



Figur 2.2: Viser to overlappende lakkeringskjegler sett ovenifra



Figur 2.3: Lakkeringsmønster som viser forskjellige attributter

2.1.2 Lakkerings mønster

Lakkeringsmønsteret sørger for at lakkeringsmediet dekker objektet jevnt. En viktig faktor med å bestemme dette er avstand mellom hver bevegelse. Feil avstand gir svingning i lakk tykkelse. Som figur 2.3 viser, er lakkeringsmønsteret spredt ut over en rektangel som dekker objektet. Rektangelet er dekket med ett mønster som lakkeringen følger for å minimere ujevnheter i lakken. Det er lagt inn akselerasjonsfelt med lengde 'a' for å sørge for at lakkeringsverktøyet holder jevn hastighet over rektangelet. For å sørge for at minst mulig lakk blir brukt og færrest mulig aktueringer med lakkpistolen blir gjennomført, er det ønskelig å lakkere langs den lange siden av rektangelet.

Symbol	Betydning
m	Hvor lang avstand det er mellom hver bevegelse i lakkeringsbanen
a	Hvor lang avstand roboten trenger for å akselerere opp til riktig hastighet
z	Startavstand fra kant.
o	Hvor lang er avstanden der to lakkeringskjegler overlapper

Tabell 2.1: Utfyller figur 2.2 og 2.3.

$$m = lakkeringskjegle\_diameter - overlapp \tag{2.1}$$

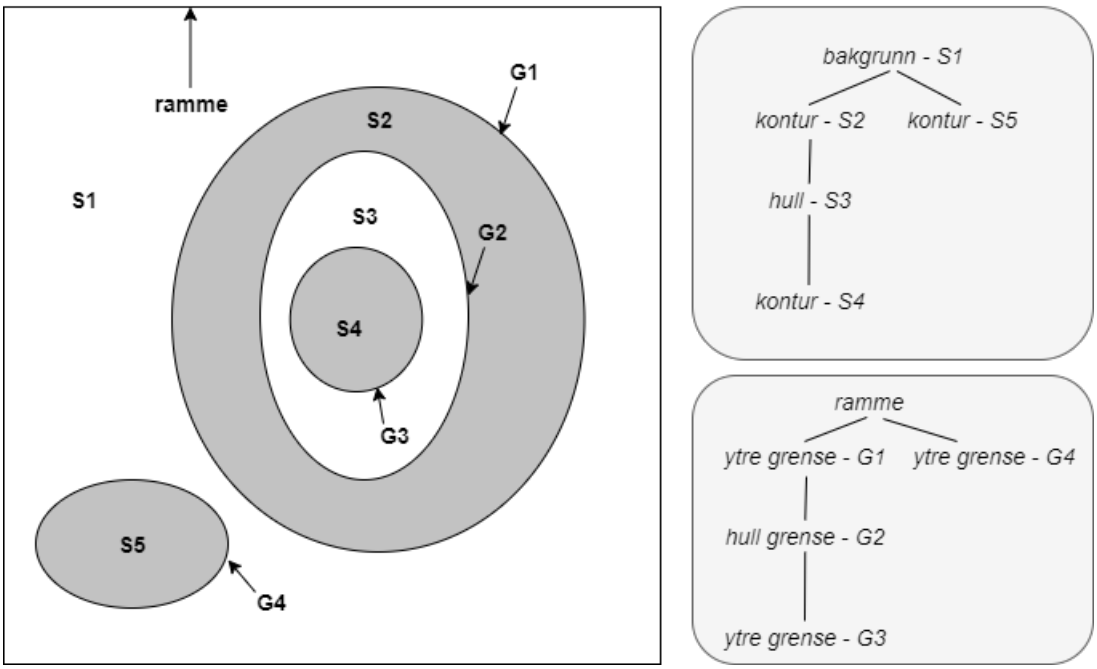
$$z = \frac{lakkeringskjegle\_diameter}{2} - overlapp \tag{2.2}$$

2.2 Kontur og topografi søk - findContour

Funksjonen 'findContour' bruker en algoritme som kalles 'Topological Structural Analysis of Digitized Binart Images by Border Following' og er utgitt av Satoshi Suzuki i 1985 [3]. Denne algoritmen brukes for å finne alle konturer innen ett binært bilde. Konturene som blir funnet, blir kategorisert og lokalisert.

Algoritmen blir brukt til å detektere alle konturer og hull i ett binært bilde. Algoritmen gir ut en liste med alle konturene, herunder posisjon og hierarki.

Algoritmen fungerer ved at den tar inn ett binært bilde. I bildet vil '0' tolkes som bakgrunn og '1' tolkes som objekt. Algoritmen søker gjennom bildet til den treffer en '1' for deretter å følge kanten til den går rundt. Deretter søkes det i konturen om det inneholder noen hull, og om hullet inneholder hull. Dette kan sees på figur 2.4



Figur 2.4: Figuren viser en grafisk fremstilling av 'findContur' algoritmen og hierarkiet den danner. [3]

Når funksjonen kjører og det er hentet frem en kontur fra ett objekt, består resultatet av en rekke med punkter. Disse punktene er koordinater til n-antall punkter som beskriver grensen

til den gitte konturen. Ligningene under viser hvordan strukturen på rekken er.

$$k_n = [(x_0, y_0), (x_1, y_1), \cdots (x_{n-1}, y_{n-1})] \tag{2.3}$$

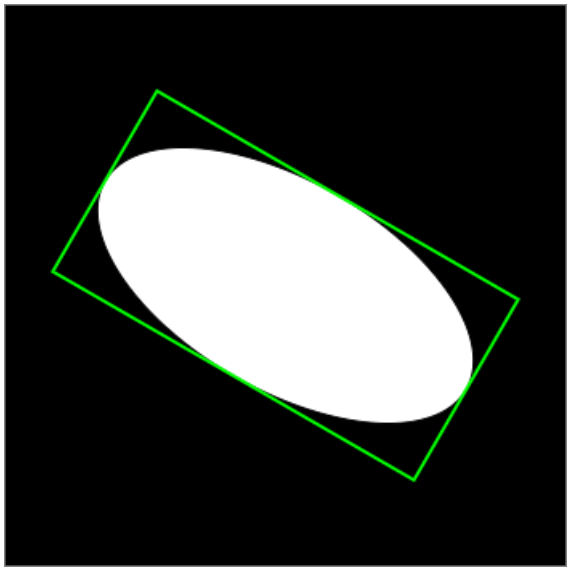
$$konturer = [k_0, k_1, \cdots k_{n-1}] \tag{2.4}$$

For å vise hvordan de forskjellige konturerene henger sammen hierarkisk, gies en matrise som beskriver relasjonene de forskjellige konturene har til hverandre.

2.3 Finne minste rektangel - minAreaRect

Funksjonen 'minAreaRect' finner det minste mulige rektangelet rundt ett objekt. Objektet kan være av hvilken som helst form. 'minAreaRect' er basert på 'Rotating Caliper' algoritmen [4]. 'Rotating Caliper' algoritmen ble originalt brukt til å finne diameter av en konveks polygon, men er videreutviklet til å finne minste dekkende rektangel. [5]

Det minste dekkende rektangelet av objektet har en side som er kolineær med en av kantene til objektet som skal dekkes. På grunn av dette, lages alle rektangler som har sider som er kolineær med hjørner av objektet som skal dekkes. Deretter plukkes ut det rektangelet som har minst areal. Kjøretiden på denne algoritmen er $O(n^2)$. [5]



Figur 2.5: Figuren viser resultat av 'minAreaRect' funksjonen

2.4 Avstand til kontur - pointPolygonTest

Avstand til kontur eller 'pointPolygonTest' er en funksjon laget av OpenCV [6]. Funksjonen sammenligner ett punkt på bildet mot en gitt konturgrense. Funksjonen har to moduser. En som retunerer svar om punktet er i eller utenfor grensen, og den andre modusen gir ett svar på hvor langt unna punktet er den gitte konturgrensen. Tabell 2.2 viser resultatet fra figuren i en oversikt. Figur 2.4 viser hva som menes med en kontur-grense.

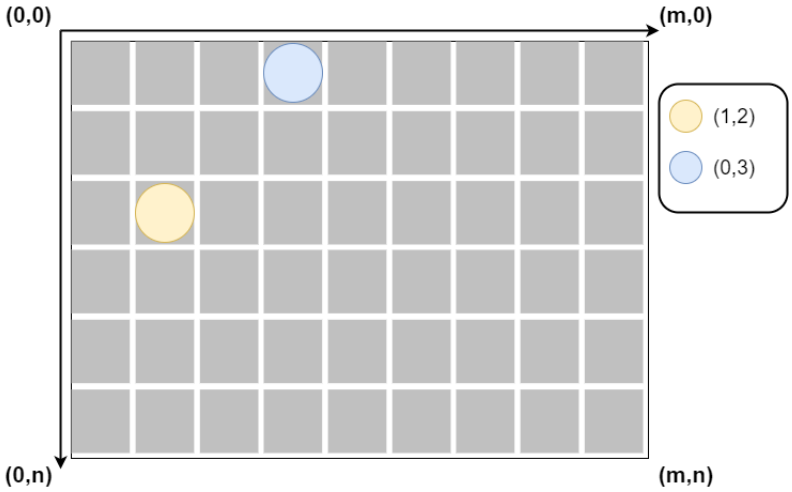
Plasering av punkt	Med avstand	Uten avstand
Utenfor grense	-n	-1
På grense	0	0
Innenfor grense	n	1

Tabell 2.2: Symbolet 'n' beskriver avstand fra punkt til grense

Hvordan denne funksjonen fungerer i dypden er ikke beskrevet av 'OpenCV'.

2.5 Koordinatsystem på bilder

Digitale bilder består av ett rutenett av piksler. Disse pikslene er det som fyller bildet med innhold. Måten de organiseres og identifiseres er ved hjelp av ett koordinatsystem. I motsetning til ett normal koordinatsystem som har positiv x- og y-akse, har bilder positiv x-aks og negativ y-akse. Aksene går heller ikke mot negative verdier. Dette vil si at posisjonen øverst til venstre har koordinat $(0, 0)$, og posisjonen nederst til høgre har posisjon (m, n). Dette kan tydes på figur 2.6.



Figur 2.6: Figuren viser en grafisk fremstilling av ett piksel rutenett.

2.6 Rotasjonsmatrise

Rotasjonsmatrise brukes for rotasjoner i euklidsk rom. Dette vil si at ett punkt, P roteres rundt origo, og punkt *'P"* dannes. Ligningen under viser rotasjonsmatrisen for rotasjon med klokken.

$$\begin{bmatrix} P'_x \\ P'_y \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} \tag{2.5}$$

Er det ønskelig å rotere rundt ett punkt som ikke er origo, ganges inn en affin transformasjonsmatrise til punktet. Ligning 2.6 viser transformasjonsmatrisen. I ligningen beskriver 't_x' og 't_y' koordinatene til punktet det er ønskelig å rotere rundt.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.6}$$

2.7 Massesentrum av polygon

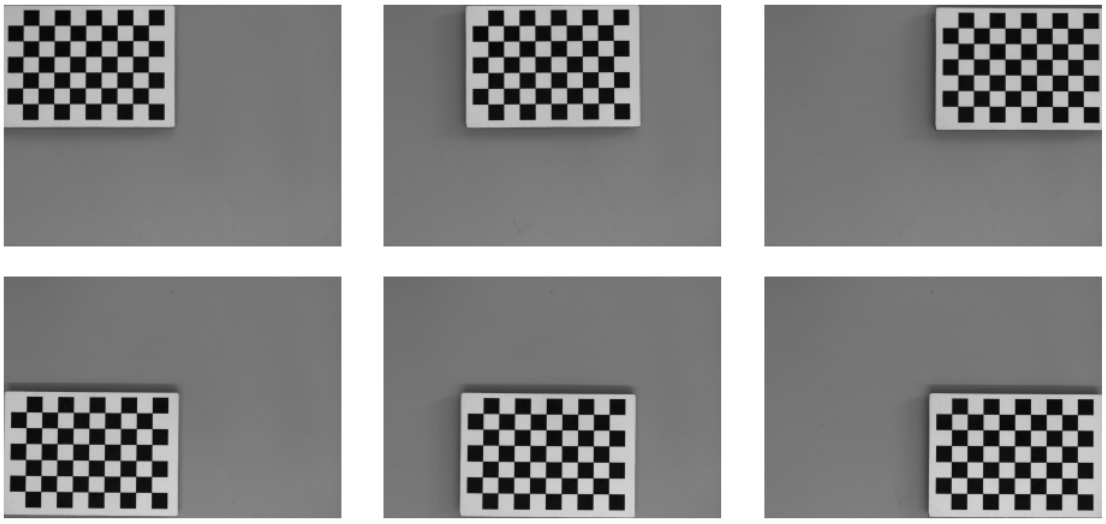
For å finne massesentrum til en vilkårlig polynom, finnes først konturen til polynomet. Konturen blir funnet ved hjelp av ett kontursøk som er beskrevet i avsnitt 2.2. Konturen

som er en rekke bestående av koordinatene til polynomet, blir kjørt gjennom en for-løkke. Gjennomsnitt-verdien for x- og y-akse blir regnet frem. Denne gjennomsnitt verdien er senter av masse til polygonet.

2.8 Kamera kalibrering

For å bruke digitale kameraer i en industriell setting, er det viktig at bildet som brukes er kalibrert. Dette vil si at objekter som vises på bildet, har lik form og fasong som objektet har i virkeligheten. På ett kamera er det flere faktorer som spiller inn i hvordan objektet fremstilles. Disse faktorene er blant annet forvrengninger skapt av linsen, og hvor paralell kamera-brikken er mot linsen. Alt dette må tas med når kamera skal kalibreres.

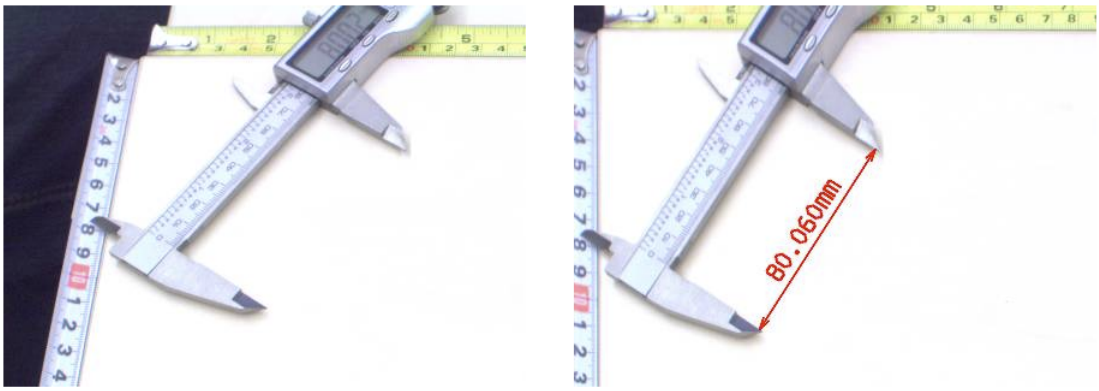
I denne oppgaves ses det på interaksjon mellom flate objekter og robot. For at roboten skal kunne brukes til å samhandle med objekter, er det essensielt at en presis kalibrering blir foretatt. Denne kalibreringen fører til at kamera-koordinatene er lineært fremstilt. Dette er en forutsetning for at kamera-koordinatene kan transformeres til verdens-koordinater som roboten benytter seg av.



Figur 2.7: Figuren viser eksmpelbilder av 'sjakk brett' brukt for kamera kalibrering [7]

For å gjennomføre denne kalibreringen tas det flere bilder av ett 'sjakk brett'. Dette brettet

plasseres på forskjellige steder i bilderammen for å få en så god kalibrering som mulig. Figur 2.7 viser hvordan disse bildene kan se ut. Legg merke til plasseringen av sjakkbrettet er i alle delene av bildet. Kalibrerings-bildene behandles av en algoritme og danner en kamera-matrise. Etter kalibrering vil bilde bli likerettet. Dette vil si at en cm i midten av bildet og en cm i en kant, vil dekke like mange piksler. Eksempel på dette er vist i figur 2.8.



Figur 2.8: Figuren viser eksmpelbilder av før og etter kalibrering. En programvare for å telle piksler er benyttet på det kalibrerte bildet for å vise oppnådd presisjon. [7]

Kapittel 3

Metoder og implementering

I dette kapittelet gåes det gjennom hva som er implementert av egne algoritmer og hvordan forskjellige problem er løst. Programmet plukkes ned i biter og forteller om de forskjellige valgene og hva som har ført til at disse valgene har blitt tatt.

3.1 Forkrav til system

For å få programmet så fleksibelt som mulig er det flere inngangsvariabler. Disse inngangsvariablene er beskrevet i 'tillegg A'. Resultatet til programmet er da koordinatene som roboten skal følge. Disse koordinatene er regnet frem med tanke på inngangsbildet fra kamera. Nå er det slik at programmet krever at bildet er prosessert på en spesiell måte for å fungere.

I korte trekk er disse stegene:

1. Korrigere bilde, fjerne forvrenginger.
2. Hente ut objekt fra bilde, gjøre bildet binært.
3. Gi riktige inngangsparametre inn i programmet med tanke på ønskede spesifikasjoner.

1. Korrigere bilde Bilde forvrenges på grunn av måten linsen fortrenger lyset og danner ett bilde. Disse forvrengningene gjør at bildet ikke gjengir virkeligheten. Typisk er dette mot kanten i bildet, hvor ett objekt kan virke breiere eller kortere enn hva det er. Dette vil føre

til at pikslene i bildet ikke kan virke som et koordinatsystem. I praksis vil en piksel dekke ett kvadratisk området på en gitt størrelse. Mot kanten av bildet vil denne gitte størrelsen være forskjellig. Når en skal hente størrelsen av ett objekt kan en da ikke stole på at størrelsen som vises på bildet, faktisk er objektets riktige størrelse. For å fjerne dette problemet må derfor bildet korrigeres. Se teori avsnitt 2.8 for mer info.

Om bildet ikke er kalibrert, eller ikke er kalibrert presist nok, minkes presisjonen fra bilde til robot. Det kan hende at en gitt linse gir ett godt nok bilde, men for å oppnå høgest presisjon må kamerasystemet kalibreres.

2. Hente ut objekt fra bilde, gjøre bildet binært For å kunne plukke ut objektet som skal lakkeres, må dette fremheves binært. Det vil si å skille ut objektet fra bakgrunnen. Objektet skilles ut ved å gjøre om bildet binært. Det gjør at bakgrunnen blir svart (0) og kun objektet som skal lakkeres blir hvitt (1). Dette gjør at objektet klart og tydelig skilles ut, og programmet vet sikkert hva som skal lakkeres.

Det er flere metoder å skille ut objekt på, og det må være opp til de som bruker programmet. En enkel måte å gjøre dette på er å sammenligne ett bilde med og uten objektet. Da kan en algoritmen for å hente ut objektet benyttes. Dette krever da at kameraet står på samme posisjon når de to bildene tas.

3. Parametre etter ønskede spesifikasjoner For å kjøre programmet er det parametre som styrer hvordan programmet lager lakkeringsbanen. Parametrene som har med lakkeringsbanen å gjøre er:

- Akselrasjons-bane
- Overlapp
- Diameter

Detaljer om lakkering finnes på kapittel 2.1. Disse parametrene er angitt i antall piksler. Dette vil si at brukeren må selv regne overgangen fra en måleenhet til piksler. Grunnen til at det er angitt i piksler og ikke i en måleenhet som meter er at programmet ikke vet spesifikasjonene til data-syn systemet til brukeren. Lakkeringsbanens parametre endres til

ønskede parametre, og programmet lager bane med hensyn til disse. Se tilleggskapital A for mer informasjon om alle parametrene.

3.2 Oversikt

I dette avsnittet gåes det gjennom trinnene programmet bruker fra inngangsbilde til lakkbane koordinater.

Trinnene kan deles inn i tre hovedkategorier.

- Identifisering og klassifisering av objekt
- Håndtering av hull i objekt
- Bane-utregning

Lakkeringsbanen gis i ett området som er rektangulært. Lakkeringsobjektet kan ha hvilken som helst form, i tillegg til former med hull. Siden lakkeringsområdet er rektangulært er det ønskelig å dele opp objektet i biter som er mest mulig rektangulært. Dette for å spare lakk.

3.2.1 Identifisere objekt

Bildet som blir sendt inn i programmet har forhåndskrav om at det er binært og at objektet er markert hvitt. Videre må objektet klassifiseres. Klassifiseringen går ut på å finne alle hullene som er i objektet.

Metode:

Metoden for å finne hull i objektet bruker topografi søk algoritmen beskrevet i 2.2. Algoritmen gjennomfører ett topografisøk og gir ett strukturbasert svar på hvilke hull som hører til hvilke objekt. Denne strukturen brukes til klassifisering på om objektet trenger videre behandling for hull, eller om objektet er klar for bane-utregning. Algoritmen gir også bildekoordinater på hvordan objektet er utformet. Denne informasjonen brukes ikke i klassifiseringen, men tas i bruk senere.

Flere metoder er ikke vurdert siden denne metoden er enkel, robust og effektiv.

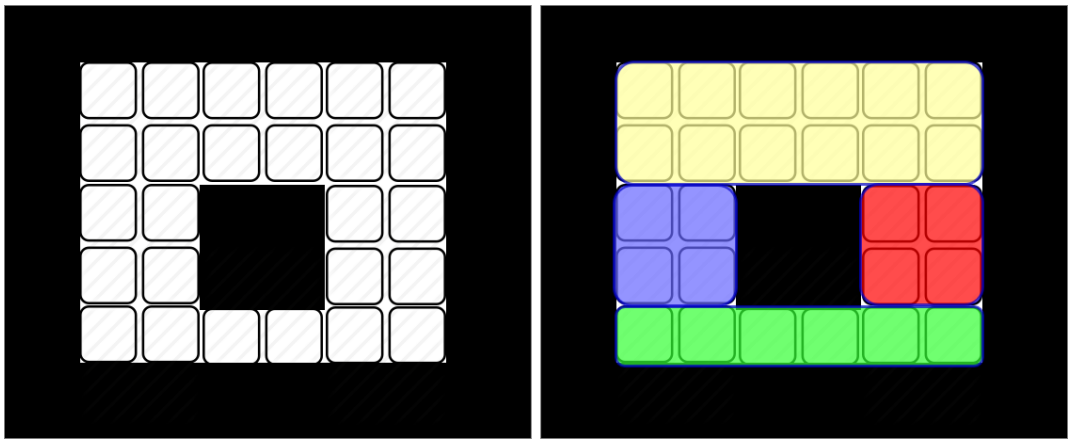
3.2.2 Håndtering av hull i objekt

Metoder

For å håndtere hull i objekt er flere metoder vurdert, men bare en er implementert og testet. Kriteriene for metodene er:

- Lakkbesparing. At lakken treffer objektet og ikke området rundt objektet.
- Antall aktuasjoner for robot-arm. Går ut på hvor langt verktøyet til roboten må forflytte seg. Dette sier noe om slitasje, men også noe om hvor lang tid lakkeringen tar.
- Antall aktuasjoner for lakkpistol. Sier noe om slitasje på robotens lakkpistol
- Kompleksitet. Hvor krevende er det å lage denne løsningen
- Robusthet.

Rutenett metoden Metoden går ut på å fylle objektet som skal lakkeres med ett rutenett. Objektet er allerede skilt ut fra bakgrunnen så det er lett å finne objektet. Dette rutenettet er hva som skal danne lakkeringsrektanglene som er beskrevet i teori kapittelet på avsnitt 2.1. Rutenettet starter med små ruter som dekker objektet. Disse rutene må kunne gruppere seg for å danne større ruter. For å finne hvilke ruter som skal slå seg sammen for å danne en stor rute, trengs det en algoritme. Denne algoritmen må se på nabo-ruter for å finne ut om de gruppere seg sammen eller ikke. Figur 3.1 viser ett tenkt eksempel på hvordan ett rutenett på 26 mindre ruter kan gruppere seg sammen til fire større ruter.



Figur 3.1: Viser eksempel på rutenettmetoden

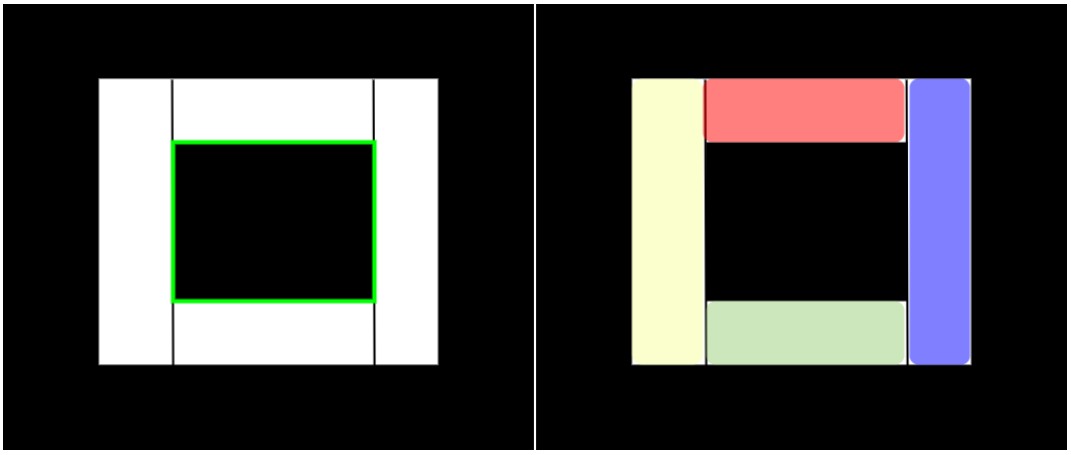
Begrunnelse

De tenkte fordelene med denne metoden er at systemet er veldig robust mot forskjellige geometriske former. Objektet kan ha hvilken som helst form, og om det er hull i objekt vil det ikke ha noen påvirkning. Dette gjør at 'vanskelige' geometriske funksjoner uten hull, deles opp i flere rektangler. En annen gode med denne metoden er at områdene som lakkeres, ikke har mulighet til å bli lakkert to ganger. Med en god algoritme for å slå sammen firkanter, kan denne metoden gi en generelt god løsning.

Ulempene med denne metoden er at den er noe innflukt. Det er ingen gode bibliotek lett tilgjengelig, som fører til att alt må kodes fra grunnen av.Kompleksiteten blir derfor noe høg. Om denne algoritmen ikke blir skrevet godt nok, kan det føre til at det dannes unødvendig mange rektangler. Dette vil igjen føre til unødvendig mange aktuasjoner for roboten. En annen faktor å trekke frem er at alle rektanglene kun kan stå horisontalt eller vertikalt på rutenettet. Dette fører til at om objektet har form som en rombe, vil metoden ha begrenset effekt.

Fyll metoden Denne metoden tar utgangspunkt i at objektet har ett eller flere hull i seg. Metoden går ut på å lage ett rektangel som fyller ut hullet på en best mulig måte. Når dette rektanglet er funnet, tegnes to parallelle streker som ligger på to av rektangelets sider. Dette gjør at ett objekt med ett hull, er nå kuttet opp i fire biter. Disse fire bitene kan nå

behandles videre som fire uavhengige objekt uten hull. Figur 3.2 viser eksempel på denne metoden.



Figur 3.2: Viser eksempel på fyllmetoden hvor den grønne rektangelen har fylt hullet og ut ifra denne delt opp objektet i fire biter.

Begrunnelse Fordelene med denne metoden er at den er forholdsvis enkel å implementere og forholdsvis robust. Metoden er forholdsvis enkel å implementere fordi at det eneste problemet å løse, er å finne en rektangel på innsiden av ett polygon. Å finne ett matematisk korrekt svar på dette viser seg å være noe utfordrene [8], men ett tilnærmet svar vil fungere i denne sammenhengen. Å lage en tilnærming vil være gjennomførbart. Robustheten til denne metoden burde være brukbar, men på hull som er 'c' formet vil ikke denne metoden gi noe godt svar. Grunnen er at ett rektangel vil bare greie å utfylle mindre deler av hullet. Antallet aktuasjoner for robot-arm og lakkpistol er igjen avhengig av hvordan objektet blir delt opp. Generelt er det to problemer med denne metoden. En, at lakkbesparelsen av denne metoden er avhengig av formen på hullet og figuren. To, at algoritmen for å finne rektangelet i hullet er avgjørende siden det ligger i grunn for hvordan objektet skal kuttes opp.

Resultatet gir da middels til høy lakkbesparing, spørs etter objektets geometriske form. Aktuasjoner for robot-arm og lakkpistol går mye etter hvordan objektets hull er, og hvordan den geometriske formen er. Er objektet og hullet rektangulært formet gir denne metoden svært gode resultater. Om objektet har en vanskelig form, vil resultatene bli middels dårlig.

Det som er viktig er at metoden vil gi et svar alternativ som løser oppgaven uansett form. Hvor god utførelsen er, avhenger av objektets form.

Ignorere hull med dynamisk lakkerinspistol Denne metoden tar utgangspunkt i 'minAreaRect' funksjonen beskrevet i kapitel 2.3) for å danne en minimal rektangel rundt objektet. På dette rektangelet blir det tegnet lakkeringsmønster over. Istedenfor å lakkere hele rektangelet, vil lakkeringspistolen bare være på når den er over objektet. Dette vil da føre til optimal lakkbesparelse.

Begrunnelse Fordelen med denne metoden er at den er forholdsvis simpel. Trenger kun en funksjon som sjekker om objektet er under lakkeringspistolen. Siden bare objektet blir lakkert, vil denne metoden også føre til svært god lakkbesparelse

Ulempen med denne metoden er at det kreves veldig mange aktuasjoner. For noen geometriske former med lite areal som er hull vil denne metoden fungere godt. Det vil fungere siden robotarmen bevege seg i samme mønsteret som den ellers hadde gjort, bare med flere aktuasjoner på lakkeringspistol. For geometriske former som har mye hull-areal fører dette til mye ekstra bevegelse for robotarmen og aktuasjoner for lakk-pistol. Eksempel for en slik form kan være en bilderamme. Siden denne metoden bare fungere godt under svært begrensede former, blir den sett på som svært lite robust.

Konklusjon Fyll metoden ble valgt på bakgrunn av at metoden vil kunne gi svært gode resultater på enkle geometriske former, og gi et ok resultat på vanskeligere geometriske former. Metoden er av middels kompleksitet, hvor det komplekse i denne metoden er å utvikle en god algoritme for å fylle hull. Det er også mulig å videreutvikle denne metoden med å legge til ekstra funksjonalitet.

3.2.3 Bane-utregning

Bane-utregning går ut på å finne banen til roboten. Dette er da banen som lakkeringspistolen skal følge for å lakkere objektet. På dette steget er området som skal lakkeres identifisert. Vinkler og lengder til objektet må finnes. Denne informasjonen må brukes til å regne frem banen. Banen må dekke hele objektet i henhold til spesifikasjonene nevnt i kapittel 2.1.

Denne banen blir kalt for 'lakkeringsbanen' siden det er banen roboten skal følge for å lakkere objektet. Siden banen alltid dekker ett rektangulært område, kalles dette for et 'lakkerings-rektangel'.

Det som er viktig i denne sammenhengen, er å lage en robust metode som består av to trinn:

1. Finne og hente informasjon for alle objekt i bildet
2. Planlegge lakkerings-banen

1. Finne og hente informasjon for alle objekt i bildet

For å finne alle objektene brukes kontursøk beskrevet i avsnitt 2.2. Dette er en metode som vil gi alle konturene i bildet. Deretter kan 'minAreaRect' funksjonen hente ut de optimale rektanglene til alle objektene. På hvert rektangel identifiseres lengste side og hvilken vinkel den har i forhold til x-aksen til bilde-koordinatene. Det trengs en algoritme for å bestemme ett fast 'start hjørne' til utregningen av lakkeringsbanen. Dette hjørne må indentifiseres slik at metoden som brukes for å tegne lakkeringsbane vet hvilken vei den skal tegne. Dette er en metode for å gjøre algoritmen robust nok til å takle alle vinklene rektangelet kan ha.

2. Planlegge lakkerings-bane

Objektet som skal lakkeres er nå funnet. Det er dannet ett rektangulært området over objektet. Dette området er det som lakkeres. For at det skal lakkeres, trenger det å dannes en bane roboten skal følge. Denne banen, også kalt lakkeringsbanen må regnes frem. Lengde, bredde og vinkel til dette rektangulære området, også kalt lakkeringsrektangelet, er funnet i punkt 1. Nå skal denne informasjonen brukes til å tegne en bane over rektangelet. Det som er utfordrende å få til er å kunne tegne banen uansett hvilken vinkel rektangelet har.

Det er tenkt frem tre metoder for å løse dette problem. Metodene må oppfylle kravene som er satt, og blir rangert etter:

- Robusthet
- Kompleksitet

Metode 1: Rotere bildet etter rektangel Metoden går ut ifra å rotere bildet slik at en gitt side av lakkeringsrektangelet ligger parallelt med bildets koordinatsystem. Deretter tegnes på bane, og bildet roteres tilbake. Grunnen til at banen tegnes på når rektangelet ligger parallelt med bildet, er at tegningen av banen gjøres mye enklere. Siden det er snakk om å tegne en tilsvarende bane som vises på figur 2.3, er det enkelt å regne frem hvor de forskjellige punktene ligger. Grunnen er altså at mellom hvert punkt på banen, er det en vinkel på 90°. Når sidene ligger vinkelrett, er det da addering/subtraksjon som trengs for å regne ut alle punktene. Når punktene er plassert, roteres bildet tilbake. Og punktene ganges inn med rotasjonsmatrisen for å finne punktene når bildet er rotert tilbake. Grunnen til at bildet må roteres tilbake er at det kan være flere objekt som skal lakkeres. Å rotere tilbake gjør prosessen ryddigere.

Begrunnelse: Fordelen med denne er at den er forholdsvis enkel, men har noen tekniske utfordringer. Når bildet roteres frem og tilbake fjernes informasjonen rundt kantene.Dette er fordi bildet roteres rundt senter av bildet og deler av bildet vil havne på utsiden av bildet. Dette vil igjen føre til at den effektive delen av bildet forminskes mye. Dette gjør denne metoden lite robust og derav lite egnet.

Metode 2: Rotere koordinatene til rektangel Metode 2 er ganske lik i utførelse fra metode 1. Denne metoden baserer seg på samme teknikk, men istedenfor å rotere hele bildet etter lakkeringsrektangelet som i metode 1, roteres kun lakkeringsrektangelet. Rektangelets koordinatpunkter roteres slik at rektangelets korte side er parallelt med bildets y-akse. Dette gjør at det tas utgangspunkt i koordinater, og ikke i selve bildet. Dette gjør at koordinatene kan ha verdier som er utenfor bilderammen, i tillegg til negative tall. Det er disse koordinatene som er sluttproduktet og som blir brukt videre.

Gangen i metoden blir da å først finne 'minAreaRect' rektangelet til objektet som skal lakkeres (punkt 1). Deretter roteres rektangelets koordinater som gjør lakkbane utregningen enklere. Dette av samme grunn som beskrevet i metode 1. Etter lakkbanen er regnet frem, roteres koordinatene til lakkeringsbanen tilbake over objektet. Teorien til roteringsmatrisen finnes på kapittel 2.6.

Begrunnelse Denne metoden er simpel i den form av at kun koordinatene roteres ved hjelp av rotasjonsmatrisen. Den er også robust da koordinatene ikke trenger å stemme overens med bildet. Koordinatene har kun tatt utgangspunkt fra bildet, og er ikke avhengig av det. Resultatet som gies ut av funksjonen kan da brukes videre. Ulempen med denne metoden er at den kun kan tegne streker som beveger seg langs x eller y aksen. Slik programmet er laget nå, er det ikke behov for forskjellige vinkler mellom hvert overdrag med lakkeringspistolen. Dette er fordi programmet finner rektangler og lakkerer de. Rektanglene har alltid hjørner som er 90 °. Om programmet videreutvikles, kan det være behov for ekstra funksjonalitet. Eksempelvis om programmet går utifra å lakkere former som ikke har 90° på alle vinklene sine.

Metode 3: Regne frem hvert enkelt punkt Denne metoden går utifra å bruke trigonometriske egenskaper til å regne frem hvert enkelt punkt. Alle avstander og vinkler er kjente. Dette gjør det mulig å regne frem hvor punktene skal være. Tar utgangspunkt til ett hjørnet av lakkeringsrektangelet, og utifra denne posisjonen er resten kjente vinkler og avstander.

Begrunnelse Å ta utgangspunkt i at alle punkter regnes frem fra grunnpunkt gjør at denne metoden er meget robust. Sammenlignet med de andre metodene er denne den mest robuste. Grunnen til dette er at vinklene mellom hvert punkt ikke er avhengig av å være vinkelrett på x- eller y-aksen. De er en vinkel og en avstand fra forrige punkt. Nå er ikke denne egenskapen noe verdifull i denne sammenhengen, og kan derfor neglisjeres. Det å regne frem trigonometriske egenskaper til hvert punkt krever noe kode og kan ses på som noe komplisert logikk.

Konklusjon Metode 2 ble sett på som best egnet til denne oppgaven. Dette er fordi den er robust nok til å takle de oppgavene som den skal utføre, og er relativt ukomplisert. Rotasjon og transformasjon er matematiske funksjoner som er lett å kode og bidrar til å holde programmet enkelt. Metode 3 er mer robust, men krever en del kode for å få til. At metode 3 har en mer komplisert logikk bidrar negativt. Det er ikke noe poeng i å gjøre oppgaven mer komplisert enn hva den trenger å være. Rotasjon og transformasjon av punkt

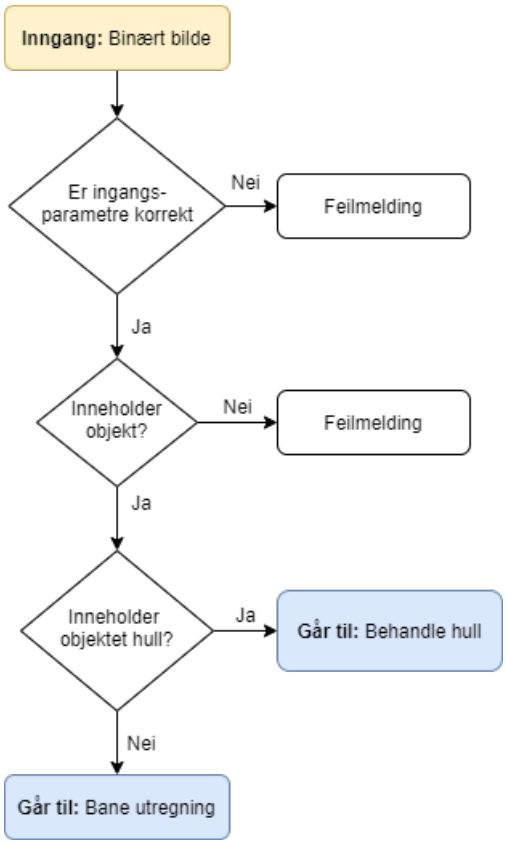
er beskrevet på avsnitt 2.6. Metode 1 er sett bortifra siden metode 2 utfyller metode 1 sine ulemper.

3.3 Implementering

Denne seksjonen viser metodene brukt i det ferdige produktet. Metodene er de som ble beskrevet og bestemt i forrige seksjon. Her gås det i detaljer på hvordan de er oppbygd.

3.3.1 Identifisering og klassifisering av objekt

Programmet startes ved å sjekke for feil på inngangsvariablene. Her sjekkes det for negative verdier, og om 'overlapp' er større enn 'diameter'. Dette vil da gi feilmelding. Deretter starter identifiseringen ved å kjøre 'findContour' funksjonen beskrevet i avsnitt 2.2. Om funksjonen ikke finner noen objekter i bildet kjøres ut feilmelding. Altså om bildet er helt svart. Om det ikke er hull i objektet vil bane utregningen starte, om det er hull, vil analysen av hull starte. Beskrivelsen av funksjonen vises på figur 3.3.



Figur 3.3: Oversikt over 'Identifisering og Klassifisering'

3.3.2 Behandle hull

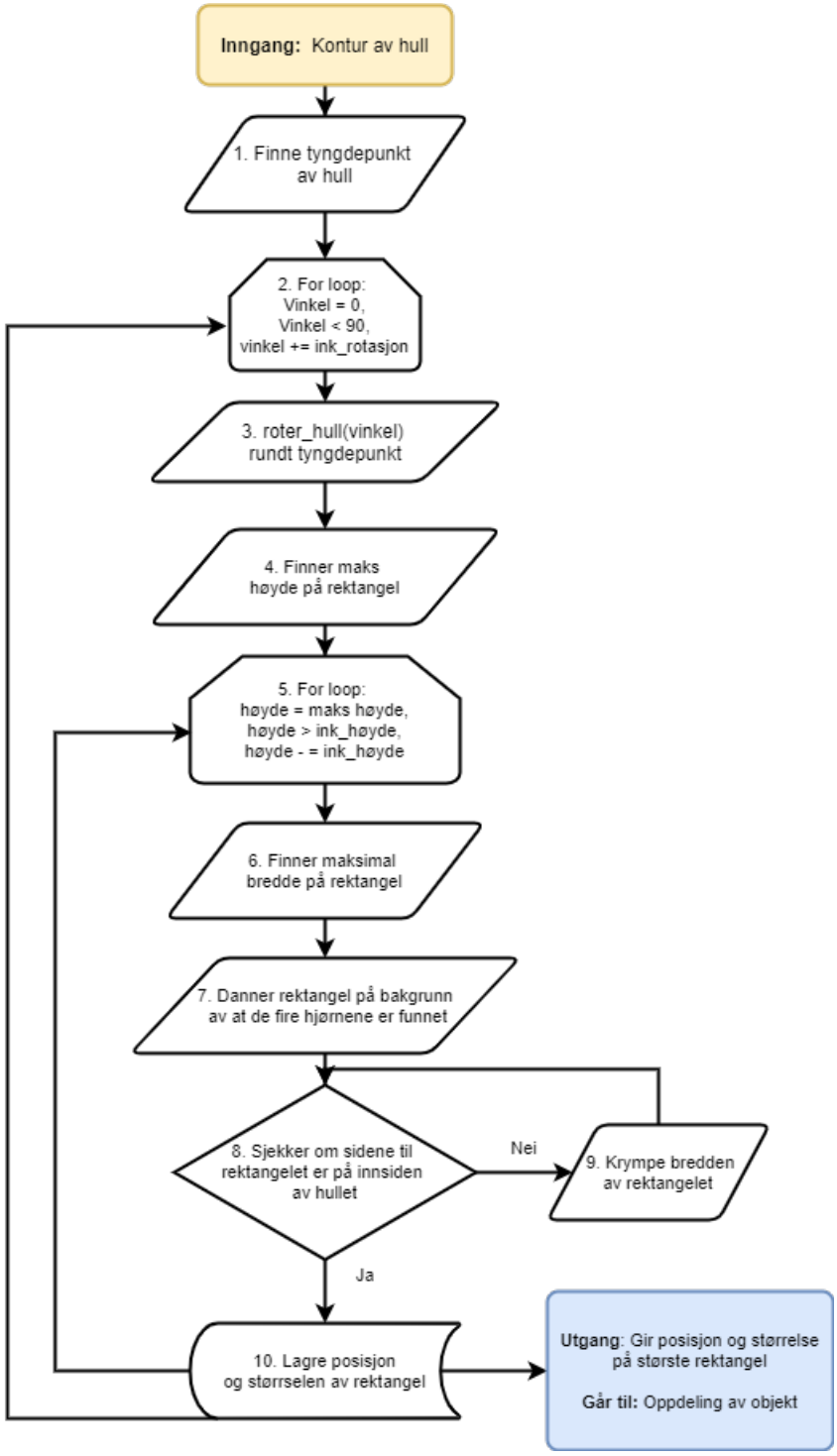
Behandle hull er prosessen for å dele opp objekt som har ett eller flere hull i seg. For hvert hull, går algoritmen gjennom to trinn. Dette vil si at om ett objekt har flere hull, blir denne prosessen repetert like mange ganger det er hull i objektet. De to trinnene er:

1. Estimere største rektangel
2. Dele opp objekt

'Behandle hull' bruker metoden som er diskutert i seksjon 3.2.2. Når det detekteres hull fra forrige avsnitt, sendes objektet med hull videre til denne funksjonen. Denne funksjonen tar hånd om hull i objekt og deler opp objektet. Dette fungerer ved å finne ett rektangel som tar mest mulig plass i hullet, og bruker to av rektangeletes parallelle sider til å dele opp objektet. Rektangelet som fyller hullet er et estimat på det største mulige rektangelet. Å

regne frem til det optimale største rektangelet er mulig og det kommer frem i artikkelen 'Largest inscribed rectangles in convex polygons' hvordan det kan gjøres [8]. Artikkelen beskriver en omfattende metode. Siden metoden er beskrevet matematisk, vil det kreve en del arbeid å få kodet metoden til Python slik at den kan brukes i denne sammenhengen. Istedenfor å bruke denne metoden er det laget ett estimat på den største rektangelen. Å lage ett estimat er godt nok til å få denne metoden til å fungere.

1. Estimere største rektangel Gangen i algoritmen er å finne det største rektangelet som får plass i hullet, hvor vinkelen er 0° på y-aksen. Rektangelet finnes ved å detektere mulige hjørner til rektangelet. Deretter sjekkes om linjen som går fra hjørne til hjørne ikke krysse konturgrensen. Om grensen ikke krysses er rektangelet gyldig. Om det ikke er gyldig, krympes rektangelet til det passer på innsiden av hullet. Nå er det største rektangelet funnet for denne vinkelen. Etter dette roteres selve hullet rundt sitt tyngdepunkt med ett inkrement. Samme metode brukes for å finne ett nytt rektangel. Dette gjentages til hullet har rotert 90°. Av de rektanglene som er funnet, er det med størst areal trukket frem som svar.



Figur 3.4: Oversikt over algoritmen som finner største rektangel

Figur 3.4 beskriver algoritmens struktur. Listen under utfyller figuren. Vær obs på forløkkene vist i figuren.

1. Finne massesenteret til hullet. Dette blir da senter av rektangelet som skal fylle hullet. Teorien til dette er beskrevet i avsnitt 2.7.

2. Siden det største rektangelet kan ha hvilken som helst vinkel, må alle vinkler testes. Siden algoritmen benytter seg av søk i lengde- og bredde-retning, roteres hullet kun 90 grader, da $90° \cdot 2 \cdot 2 = 360°$. Derfor vil alle vinkler testes, selv om hullet kun roteres 90°. Dette realiseres i form av en for-løkke.

3. Gitt vinkelen fra for-løkken, roteres hullet. Funksjonen for å rotere hullet er laget av N.V.S Abhilash [9]. Funksjonen tar inn konturen til hullet, og roterer gitt grader rundt hullets massesenter. Se avsnitt 2.7 for teori. Grunnen til at hullet roteres og ikke søkefunksjonen, er at denne måten er minst komplisert. Ulempen er at ved rotasjon av hullet, som er oppgitt ved n-antall koordinatpunkter, er at noen ganger blir det avrundingsfeil i rotasjonen. Avrundingsfeilen fører til at grensen til hullet får feil form. På grunn av dette vil 'pointPolygonTest' funksjonen ikke fungere som den skal. Dette fører til falske positive svar. Grunnen er at funksjonen tar inn kontur-grensen som er beskrevet på avsnitt 2.2.Siden denne kontur-grensen inneholder koordinater. Denne avrundingsfeilen som er beskrevet skjer sjeldent, men det er en teoretisk mulighet for feil.

4. Finner høyde av rektangel. Høyden finnes ved å følge y-aksen opp fra massesenteret til den treffer grensen til hullet. Deretter følges y-aksen nedover fra massesenteret til den treffer grensen til hullet. Den avstanden som er kortest av de to, blir gjeldene maks avstand fra massesenteret i begge retninger. Avstanden er den samme i begge retninger siden massesenteret skal være i senter av rektangelet. Eksempel på dette vises i figur 3.5

5. For-løkke for å minske høyde på antatt rektangel. Siden den maksimale høyden til en rektangel ikke nødvendigvis gir det største arealet, er det ønskelig å teste flere høyder. Dette er spesielt for polygoner, at den lengste størrelsen ikke nødvendigvis gir det

størst arealet.

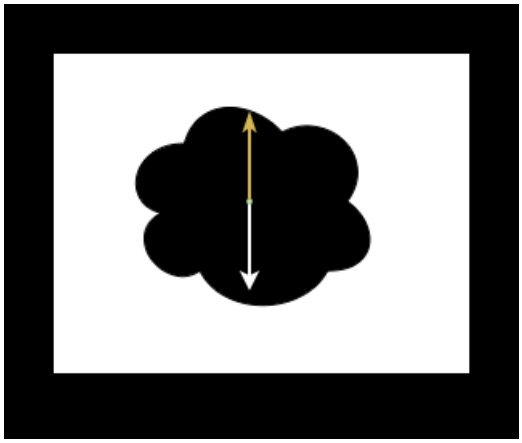
6. Finne maksimal bredde på rektangelet, gitt høyde. Topp- og bunn-punkt funnet i steg 4, brukes for å regne ut bredden. Det er fire punkter som skal finnes, topp høgre, topp venstre, bunn høgre og bunn venstre. Den korteste lengden av de blir den gjeldende lengden for alle. Se figur 3.6 for grafisk fremstilling.

7. Danner ett rektangel med de fire hjørnene som ble funnet i punkt 6.

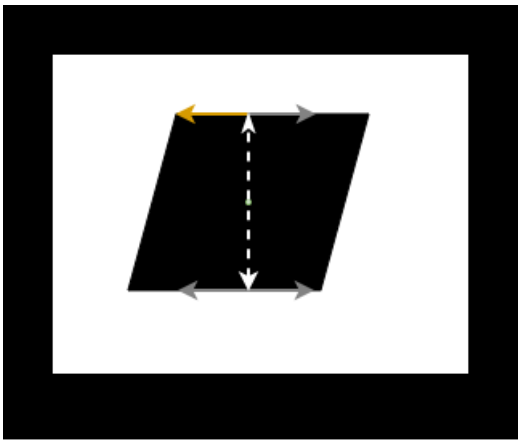
8. Sjekker om sidene til rektangelet er på innsiden av hullet. De korte sidene, er garantert å være på innsiden av rektangelet på måten punktene finnes. De lange sidene til rektangelet er ikke sjekket om de har krysset grensene til hullet, og må derfor sjekkes.

9. Om de lange sidene krysser grensen til hullet, minkes de med ett gitt antall piksler. Prosessen gjentar seg til rektangelet er på innsiden

10. Lagrer de fire hjørnene til det største rektangelet. Når vinkler og høyde er ferdig iterert gis ut det største rektangelet som brukes videre.

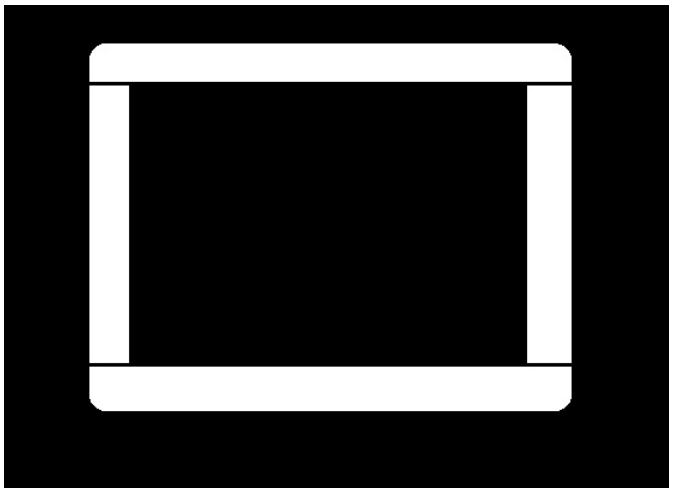


Figur 3.5: Hvordan algoritmen finner maks høyde i hull. Den oransje pilen viser at grensen er nådd, og høyde er bestemt. De to pilene har samme lengde.



Figur 3.6: Hvordan algoritmen finner maks bredde til rektangelet. Den oransje pilen viser at grensen er nådd, og bredden er bestemt. De fire pilene har samme lengde.

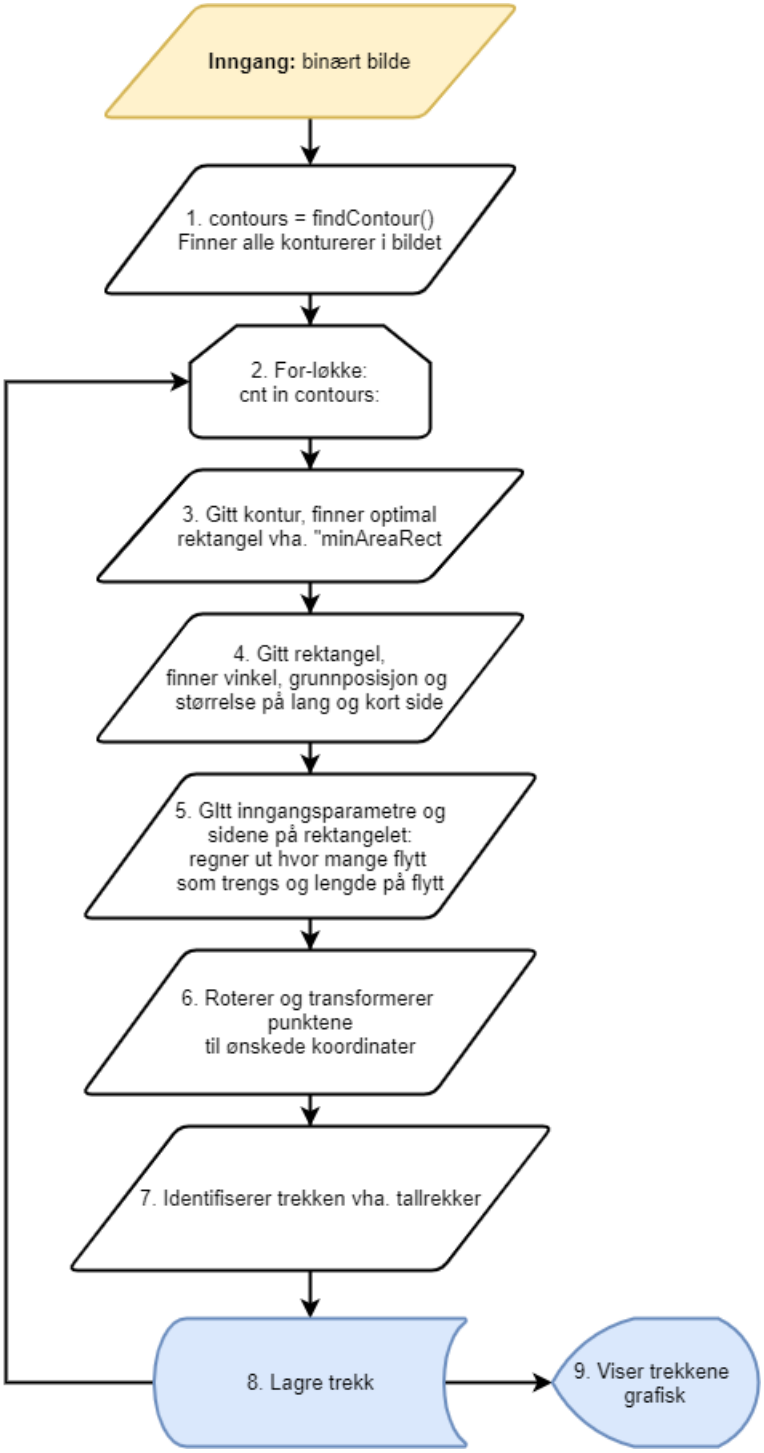
2. Dele opp objekt Etter det største estimerte rektangelet er funnet lages to kopier av inngangsbildet. På den ene kopien deles objektet opp langs lengderetningen til rektangelet, og bredde retningen på den andre kopien. Poenget med å laget to kutt, er å finne ut hva som egner seg best. Dette er opp til objektets geometriske form, og derfor testes begge deler. Etter objektet er kuttet opp, lages det ved hjelp av 'minAreaRect' funksjonen rektangler som dekker alle objektene. Det kombinerte arealet til alle rektanglene dannet av 'minAreaRect' måles opp mot det originale arealet til objektet. Det kuttet som gir ett resultat som er nærmest 100% blir valgt. Grunnen til at arealet kan bli under 100% er om det er mange hull i objektet. Siden 'minAreaRect' funksjonen baserer seg på å finne alle objektene i bildet, er det viktig at det er god nok seprasjon mellom objektene. Etter testing er det funnet ut at kuttet må ha bredde på to piksler. Flere kutt på to piksler bredde vil føre til at objektets totale areal er minket. For å unngå at for mye av objektet blir borte, kan en margin legges på rundt rektanglene som dekker objektene. Denne marginen gjør at rektanglene som dekker delene vokser med n antall piksler. Hvor mange n piksler er, bestemmes når programmet initieres. Initieringsparameterene beskrives på tilleggskapittel A og 'minAreaRect' funksjonen beskrives på avsnitt 2.3.



Figur 3.7: Figuren viser ett objekt som er kuttet langs langsiden til rektangelet.

3.3.3 Bane utregning

Bane utregningen går ut på å lokalisere alle konturer og regne ut lakkeringsbanen for konturene. Dette gjøres ved å finne ett rektangel som dekker objektet, for deretter å regne ut banen basert på rektangelet. Figur 3.8 går gjennom alle punktene algoritmen bruker for å regne ut lakkeringsbanen til objektet.



Figur 3.8: Figuren viser alle trinnene som går med for å lokalisere og regne ut lakkeringsbane

1. Søker og finner alle konturer i bildet. Søker bare etter ytre grenser. Dette vil si at om et objekt har hull, vil det bli sett bortifra. Resultatet blir da en rekke med konturer, der hver kontur beskriver sin plassering og form. Se avsnitt 2.2 for mer utfyllende info.

2. For-løkken går gjennom hver kontur funnet i steg 1.

3. Finner optimal rektangel med 'minAreaRect' funksjonen. Dette gir den minste rektangelet som legger seg på utsiden konturen. Les mer om denne funksjonen på avsnitt 2.3.

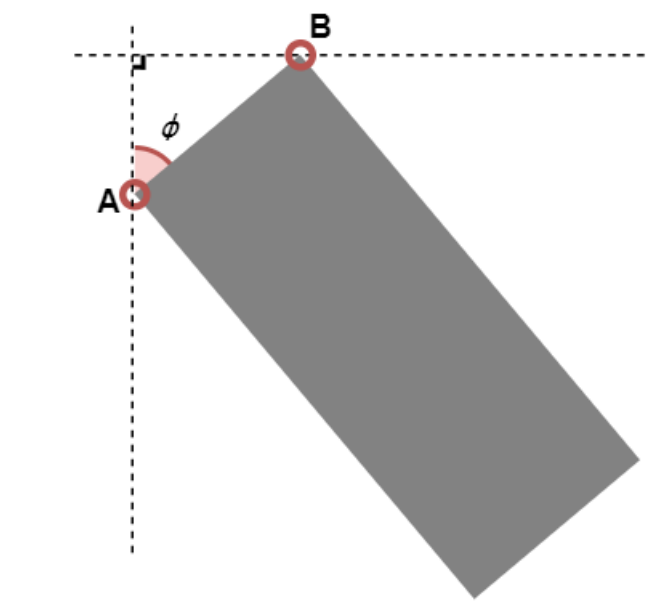
4. Regner ut grunnposisjon, lengde på sider og vinkel til rektangelet funnet i punkt 3. Grunnposisjonen er et bestemt hjørne på rektangelet. Dette hjørnet er hvor lakkeringen skal starte.

5. Regner ut hvordan lakkerings-banen skal være. Dette er basert på grunnpunktet, parametrene til lakkeringspistolen og størrelsen på lakkeringsrektangel. Disse punktene til lakkerings-banen er koordinater og de plasseres med grunnpunkt origo. Grunnen til at dette gjøres er at det forenkler prosessen videre. Les mere om dette på avsnitt 3.2.3.

6. Flytter lakkerings-banen funnet i punkt 5, til der den skal være; over lakkeringsrektangelet. Måten dette gjøres på er at punktene først roteres, deretter flyttes fra grunnpunkt origo til grunnpunkt lakkerings-rektangel. Dette gjøres ved bruk av transformasjonsmatrisen som beskrives i avsnitt 2.6. Figur 3.9 viser grunnpunkt og hvordan vinkel identifiseres.

7. Punktene som er regnet frem er nå listet i en rekke. Listen er ikke sortert på noen måte, men ved hjelp av tallrekker sorteres trekkene etter om streken går over området som skal lakkeres eller ikke.

8. Når trekkene er stortert vises lakkeringsmønsteret på toppen av inngangsbildet. Figur 3.10 viser hvordan produktet ser ut. De blå sirklene viser punktene som er regnet ut, rød pil viser hvor lakkpistolen beveger seg uten at den er over lakkeringsrektangelet, gul pil viser hvor den beveger seg mens den er over lakkeringsrektangelet.



Figur 3.9: A viser grunnpunkt og ϕ viser vinkel mellom A og B

3.3.4 Behandla data videre

Når programmet er ferdig kjørt, lages en liste med alle punktene av interesse (PoI). Disse punktene er de som danner lakkerings-banen. I figur 3.10 er PoI markert med blå sirkel. Disse punktene kommer over endring i lakkeringsbanen. Enten at punkene markerer akselrasjonsfelt, start på lakkeringsrektangel, slutt på lakkeringsrektangel, eller oppstilling til neste akselrasjonsfelt. I figurene som er vist, er det piler fra punkt til punkt. Pilene peker på hvilken vei robot vil bevege seg om den følger punkt for punkt. For å lagre punktene, er det satt opp en 'pickle' funksjon for å lagre punktene. 'Pickle' funksjonen lagrer data som en 'datastream', og det trengs en 'pickle-load' funksjon for å lese denne dataen igjen. Grunnen til at 'Pickle' er brukt er at rekkens struktur holder seg når det blir lest. Dette er mer beskrevet i tilleggskapittel A.

Ligningen under viser hvordan datastrukturen for lakkeringsbanen til figur 3.10 er. Legg merke til at det er fire lakkeringsrektangler som brukes for å dekke figuren.

$$lakkRektangel = [(x_0, y_0), (x_1, y_1), \cdots (x_{n-1}, y_{n-1})]$$

$$Figur = [lakkRektangel_1, lakkRektangel_2, lakkRektangel_3, lakkRektangel_4]$$

3.3.5 Tilleggsfunksjoner

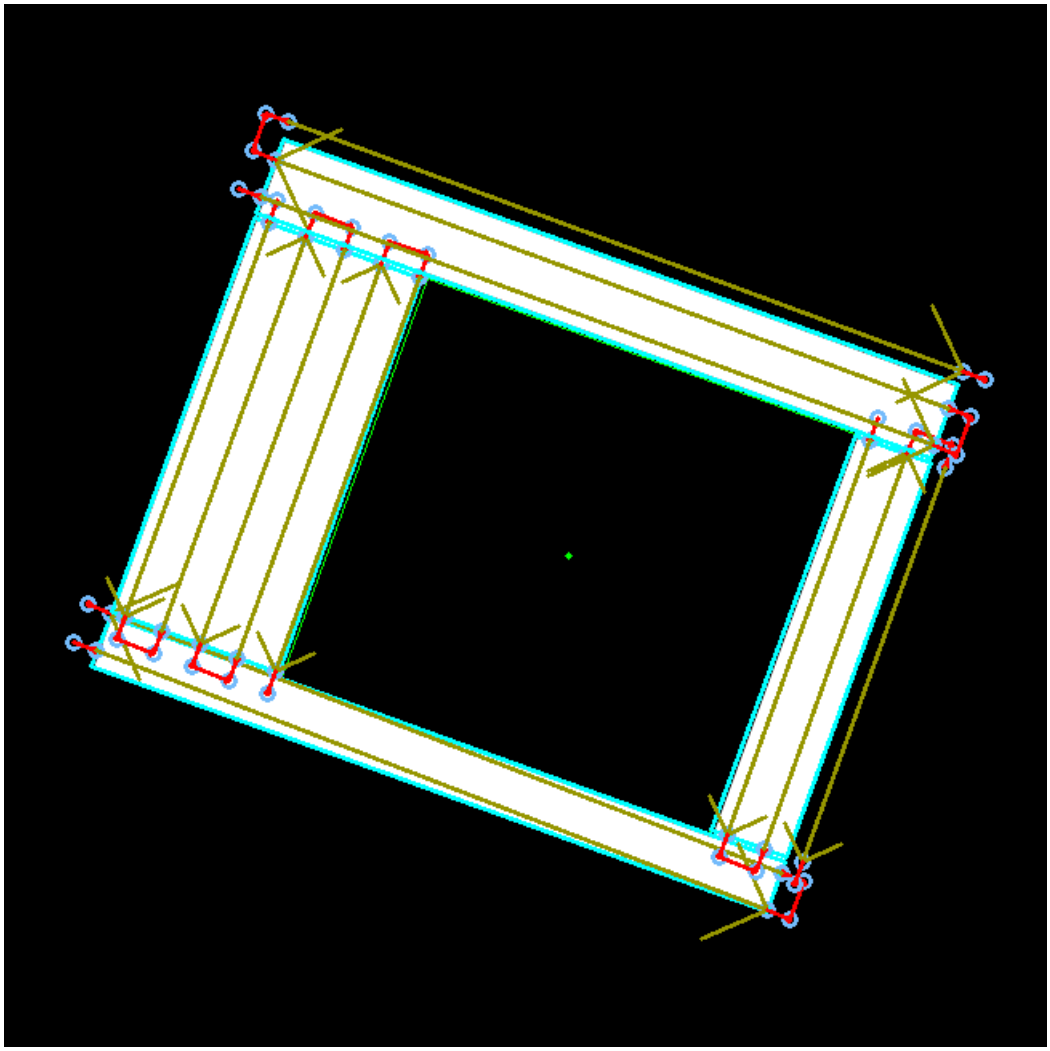
For å jobbe rundt artefakter som kan oppstå under ulike geometriske former er det to tilleggsfunksjoner for å motvirke noen, og for å gi ett bedre resultat.

Lakkering av små objekt

Etter kutting av ett objekt med flere hull, kan det oppstå fliker som er tynnere en avstand 'm'. Avstand 'm' er avstand mellom hvert flytt for lakkeringspistolen beskrevet på avsnitt 2.1. Disse tynne flikene kan være ønskelig å kjøre lakkeringspistolen over, men ved noen tilfeller er det ønskelig å ignorere disse. Dette er fordi de fører til ekstra bevegelse for roboten. Om fliken står ved siden av ett annet lakkerings-rektangel er det fullt mulig at lakkeringsmønsteret går over denne lille fliken uansett. Til dette er det en inngangs-parameter som kan settes 'True/False'. Mer om inngangsparametrene på tilleggskapittel A.

Ekstra margin til lakkerings-rektangel

Når ett objekt kuttes opp, vil deler av objektet kuttes vekk. Hvert enkelt kutt har en bredde på to piksler. Dette er en liten del av 'kaka', men om objektet har flere vindu som kuttes ved siden av hverandre kan det bli flere kutt som legger seg ved siden av hverandre. For å få disse kuttene lakkert er det mulig å øke størrelsen til lakkerings-rektangelet. Dette gjøres ved 'margin-funksjonen'. Denne marginen velges ved en av inngangsparametrene til programmet. Se mer om inngangsparametrene på tilleggskapittel A.



Figur 3.10: Viser hvordan ett objekt ser ut når den har fått lakkeringsmønster plassert over seg.

Kapittel 4

Resultater med diskusjon og videre arbeid

4.1 Resultater

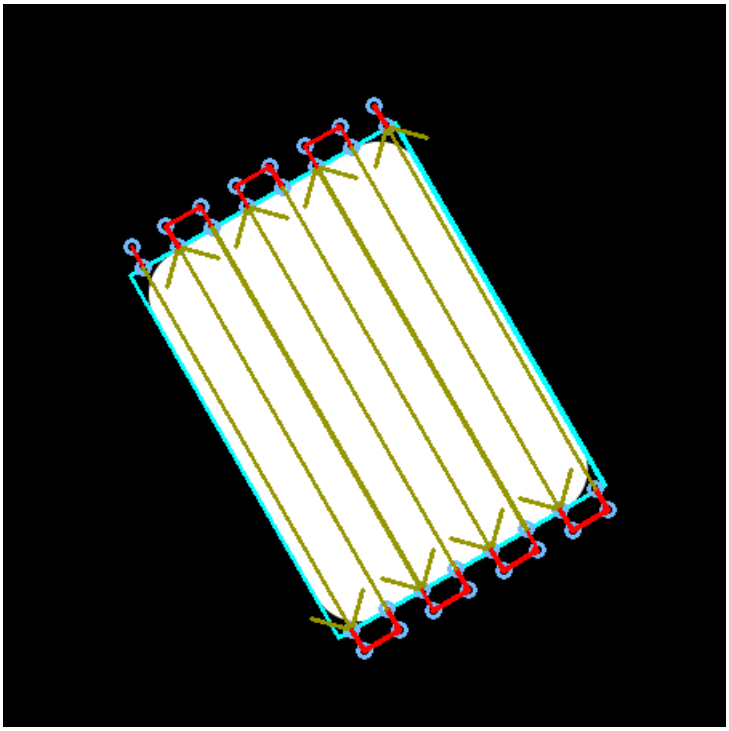
Diskuterer og sammenligner resultater fra forskjellige geometriske former. De ønskede egenskapene til algoritmen er mest mulig spart lakk, og minst mulig aktuasjoner for robot og for lakkeringspistol. Mest mulig spart lakk går utifra hvor effektive lakkerings-rektanglene er. Minst mulige aktuasjoner er gitt ved flest mulig lange bevegelser over objektet.

Måten lakkbesparelse måles på er ved å sammenligne areal til lakkerings-rektanglene opp mot arealet på objektet. Selve lakkeringen er ikke testet, så det er ingen reelle tester på hvor mye lakk som faktisk er brukt. Det er viktig å påpeke at lakkeringsrektanglene gir en indikasjon på hvor effektiv programmet er.

4.1.1 Rektangulære former

Rektangelets geometriske former gir et meget godt resultat siden lakkerings-rektangelet utfyller objektet godt. Figur 4.1 viser hvordan algoritmen tolker denne figuren. Lakkerings-rektangelet tilsvarer 102.2 % av opprinnelig areal. Dette vil si at 2.2 % av området som lakkerings-rektangelet dekker er ikke figuren. Dette tyder på effektiv lakkering. At objektet

er satt opp i en vinkel, gir ingen endring av resultat. Grunnen til at objektet ikke får 100%, er at hjørnene er avrundet.



Figur 4.1: Figuren viser lakkeringsmønster over rektangel med runde hjørner.

4.1.2 Rektangulære former med vindu

Rektangler med vindu som ligger parallelt med objektet gir gode resultater, som figur 4.2. Disse resultatene ligger nært 100%. Objekt med vindu som ikke ligger parallelt, kan gi særs dårlige resultater som figur 4.3. Som vist i figur 4.2 og 4.3, så fyller vindu-rektangelet vinduet oppimot 100%. Dette tyder på at vindu-algoritmen fungerer godt mot rektangulære vindu.

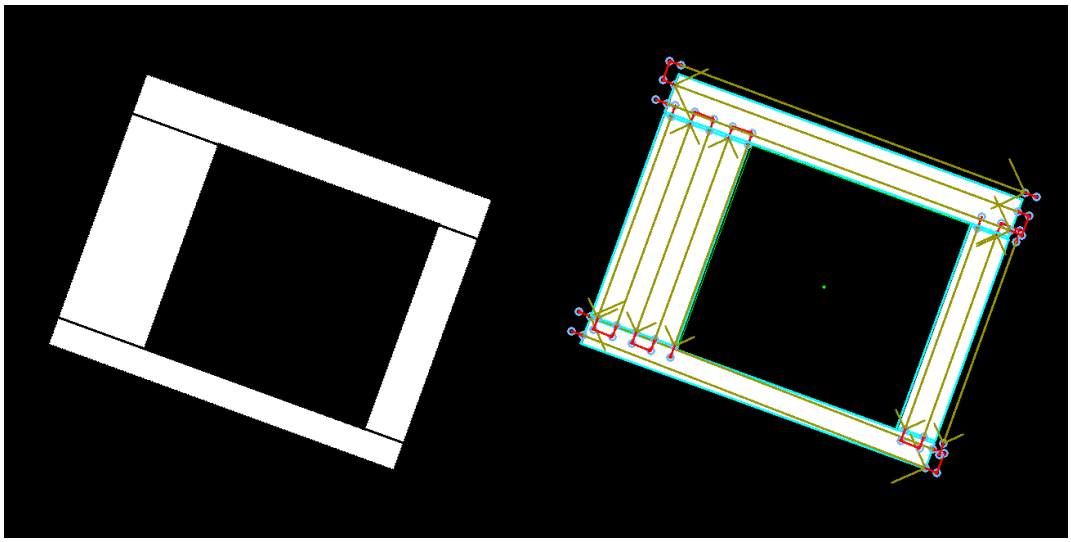
Figur 4.2 gir ett godt resultat. Figurens form gir den en effektivitet på 101.3% og rektanglene er lange som fører til få aktuasjoner. Det som fører til det gode resultatet er at 97.6% av vinduets originale størrelse er fylt. Uten vinduet hadde derav effektivitetsgraden vært på 201.7%, som vil si at det nesten hadde blitt brukt dobbelt så mye lakk som nødvendig.

Når det gjelder figur 4.3 er det ett større området som lakkeres dobbelt, samt lakk som ikke

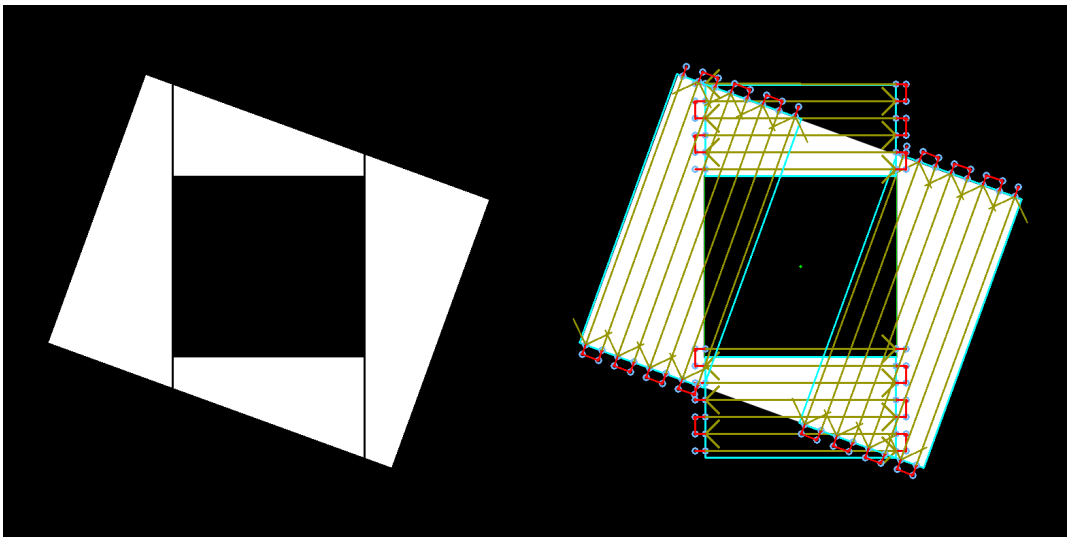
treffer objekt. Denne metoden ga ett lite effektivt svar. Om en ser på tabell 4.1, viser den at resultatet av å ikke behandle vindu gir ett bedre resultat. Dette vil si å ignorere vindu, og bare behandle objektet som ett objekt uten vindu.

Figur	**Total effektivitet**	**Vindu effektivitet**	**Behandle uten vindu**
Figure 4.2	*101.3%*	97.6%	201.7%
Figure 4.3	*161.2%*	98.6%	152.1%

Tabell 4.1: Tabellen viser grad av effektivitet



Figur 4.2: Figuren viser lakkeringsmønster over rektangel med rektangulært vindu. Vinduet er også parallelt med objektet. Objektet på venstre side er oppkuttet og på høyre side er all informasjon med.



Figur 4.3: Figuren viser lakkeringsmønster over rektangel rektangulært vindu som står med vinkel mot moder-objektet.

4.1.3 Ikke rektangulære former

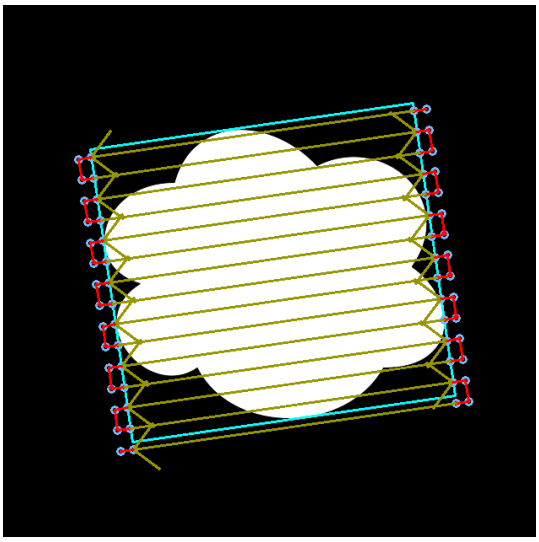
Objektene her har ikke en rektangulær form. Dette vil si at det ikke er mulig å få 100% effektivitet. Det som fører til hvor effektiv disse figurene er, er hvor lik de er rektangler.

Figur 4.4 viser en ok effektivitet på 134.5%. Ikke det beste resultatet, men det kunne vært verre. Antall aktuasjoner er holdt på ett minimum.

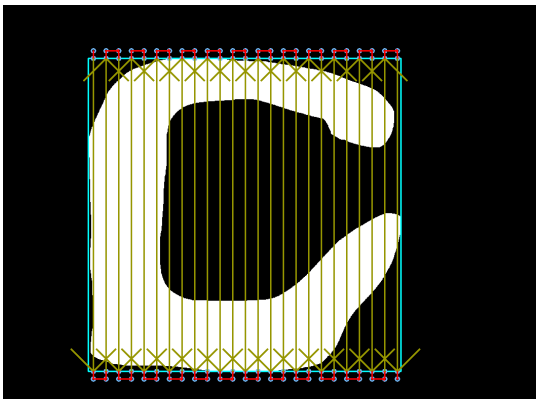
Figur 4.5 har en dårlig effektivitetsgrad på 198%. Nesten dobbelt areal som lakkeres. Algoritmen for bane-utregning takler denne typen geometrisk form dårlig. Dette er grunnet at figurer uten hull ikke deles opp. Antallet aktueringer er meget høyt, siden roboten må bevege seg ganske langt på grunn av den dårlige effektivitetsgraden.

Figur	Total Utfyllese
Figure 4.4	134.5%
Figure 4.5	198.0%

Tabell 4.2



Figur 4.4: Figuren viser lakkeringsmønster over et polygon formet som en sky.



Figur 4.5: Figuren viser lakkeringsmønster over et polygon formet som en "c".

4.1.4 konkav polygon med vindu

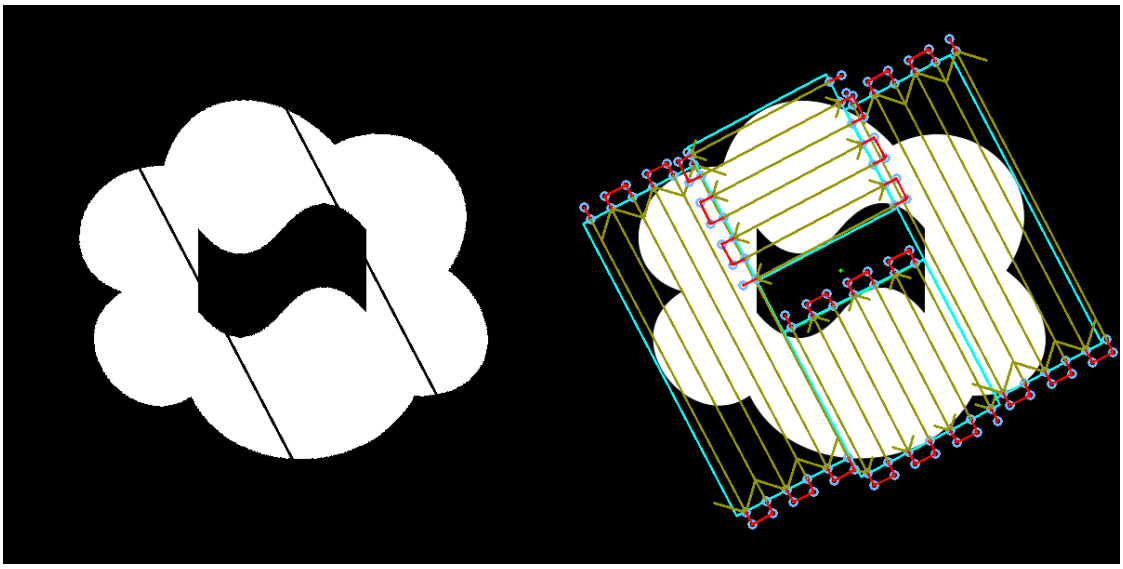
Objektene her har en form som inneholder en konkav polygon med to forskjellige vinduer som også er polygoner. Her blir vindu funksjonen testet i å finne største rektangel. Å finne største rektangel på innsiden av ett konkav polygon er en forholdsvis komplisert oppgave å finne ett optimalt svar på. Derfor er det brukt en tilnærmingsmetode for å finne dette svaret.

Figur 4.6 og 4.7 tolkes på forholdsvis like måter. Algoritmen for å tilpasse rektangel i

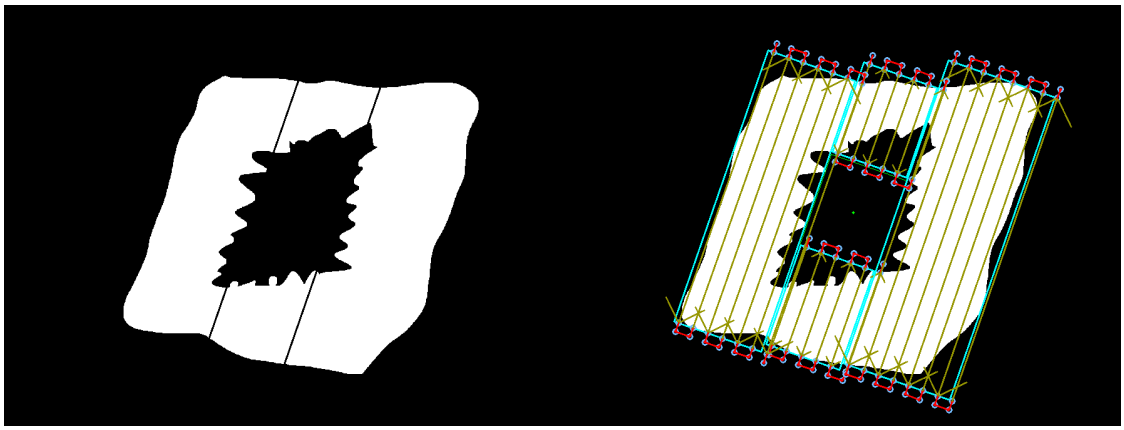
vindu fungerer slik som den skal i dette tilfellet. Figurene deles opp i biter som minner om rektangler, og derfor oppnår forholdsvis høy effektivitetsgrad. Lakkerings-rektanglene har relativt lange sider som fører til ett normalt nivå aktueringer.

Figur	**Total effektivitet**	**Vindu effektivitet**	**Behandle uten vindu**
Figure 4.6	*134.5%*	66.0%	154.9%
Figure 4.7	*127.5%*	48.6%	153.9%

Tabell 4.3



Figur 4.6: Figuren viser lakkeringsmønster over en konveks polygon med konvekst vindu.



Figur 4.7: Figuren viser lakkeringsmønster over en konveks polygon med konvekst vindu

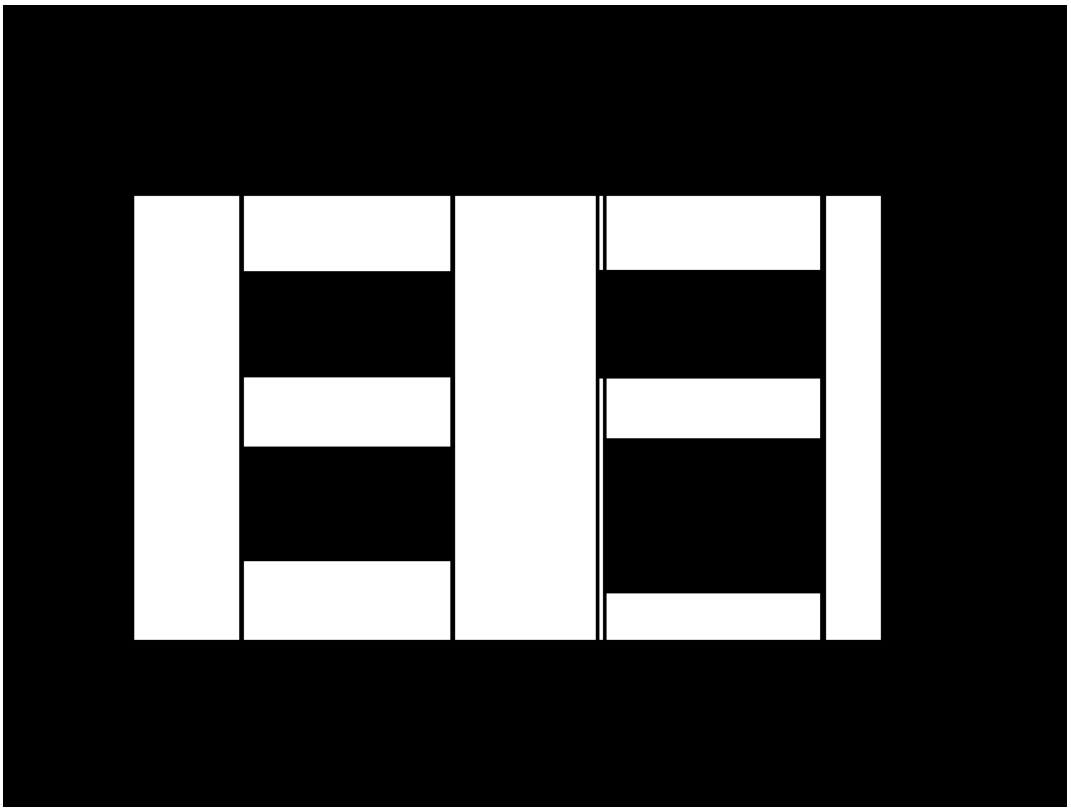
4.1.5 Figur med flere vinduer

Når det er flere vindu blir figuren kuttet opp flere ganger. Figur 4.8 viser ferdig oppkuttet objekt og figur 4.9 er lakkeringsrektanglene lagt over objektet. På figuren er vinduene på venstre side plassert med samme breddekoordinater. Dette gjør at kuttet ligger oppå hverandre og kuttet ikke er merkbart breiere enn normalt, sammenlignet med ett enkelt vindu. På høgre siden er de bare nesten samme bredde. Dette gjør at området rundt denne delen har noen artefakter. På høgresiden er forskjellen mellom de lagt slik at kuttet blir dobbelt så bredt som det skal være på høgre side. På venstre siden er kuttet gjort at en flik på 3 piksler står alene. Denne fliken står ved siden av en større rektangel, og i utgangspunktet skal fliken høre til denne rektangelet. Siden fliken står som en egen bit, vil denne få sitt eget lakkeringsmønster lagt over. Mønsteret dekker da i grunn ett mye større område enn hva fliken er. Som skrevet om på avsnitt 3.3.5, er det laget en funksjon som kan ignorere fliker. Denne funksjonen styres ved en inngangsvariabel. Som sett på figur 4.9, er fliker ignorert. Selv om fliken er ignorert, overlapper lakkeringsmønsteret for rektangelet til venstre for flikene. Om fliken ikke hadde fått ett overlappende lakkeringsmønster, er det også mulig å bruke margin for å utvide alle lakkerings-rektangelene for figuren. Som siste utvei er det mulig å velge at fliker skal få eget lakkeringsmønster. Hadde fliker ikke blitt ignorert i dette eksempelet ville det ført til ekstra aktuasjoner for robot samt mer lakk som hadde gått med. Når det gjelder utregningen av effektivitet, er ikke margin tatt med

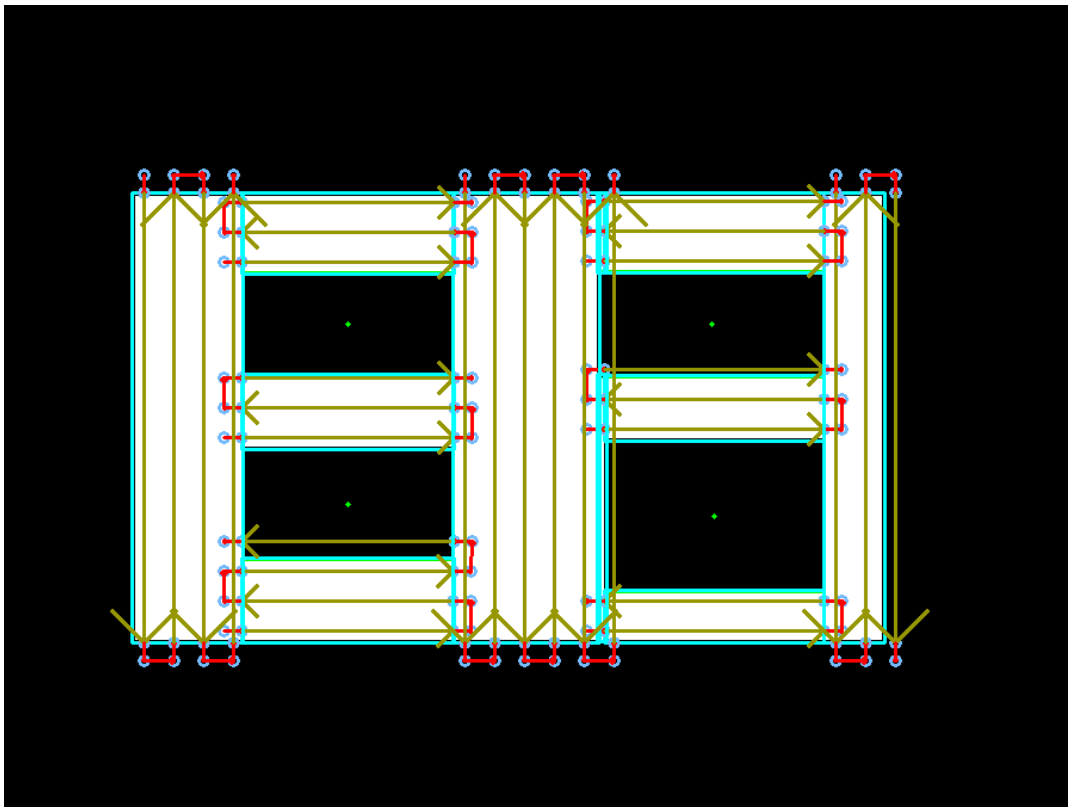
i utregningen. Fliker er med i utregningen siden effektivitets utregningen er definert som: $Total\_effektivitet = \frac{Samlet\_lakkerings-rektangler\_areal}{Totalt\_areal}$.

Figur	**Total effektivitet**	**Vindu effektivitet**	**Behandle uten vindu**
Figure 4.9	*96.2%*	96.5%	148.2%

Tabell 4.4



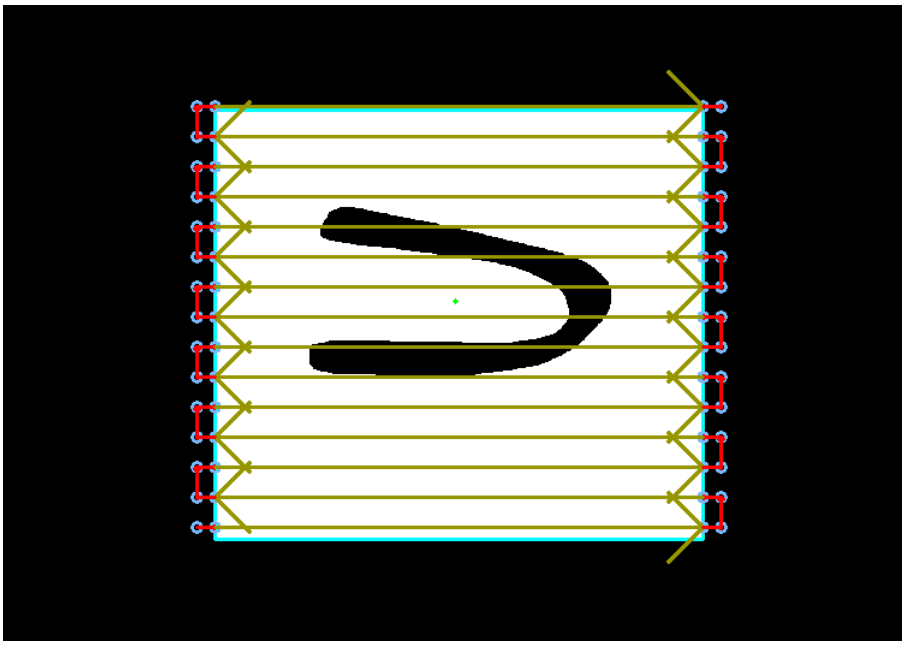
Figur 4.8: Figuren viser objektet som er oppkuttet slik at ingen konturer inneholder ett vindu.



Figur 4.9: Figuren viser lakkeringsmønster over en figur med flere vindu. Legg merke til at mindre objekt er ignorert og at størrelsen på lakkeringsrektanglene er 3 piksler lengre i alle retninger.

4.1.6 Vindu hvor tyngdepunkt er utenfor figur

Når tyngdepunktet til ett vindu ligger på utsiden av vinduet vil vindu-algoritmen ikke fungere. Dette er fordi algoritmen bare lager vindu-rektangel ut ifra tyngdepunktet til vinduet. Når tyngdepunktet er på utsiden, finner algoritmen ingen punkter som er på innsiden av ett vindu. Dette fører til at algoritmen gir en feilmelding, og går videre til bane-utregning. Baneutregningen tar bare for seg kanten til objektet og lager derfor ett lakkerings-mønster som går over objektet. Figur 4.10 er ett eksempel på dette. Legg merke til at tyngdepunktet til vinduet er markert med en grønn prikk.



Figur 4.10: Viser objekt hvor vinduet har tyngdepunkt på utsiden av vinduet

4.1.7 Sammendrag fra resultater

Programmet yter godt under gode forhold, men yter under standard ved vanskelige forhold. En gjenganger er at om vinduet, og figuren har en form som er tilnærmet lik ett rektangel blir resultatet svært godt og dekker opp mot 100% av objektet. Når figuren ligner romber eller har rektangler som står 45° på hverandre, blir det mindre gode resultat. Grunn funksjonaliteten er på plass, men den har sine begrensninger. Med litt videre utvikling, kan dette programmet virke svært godt. Slik programmet er nå, gir det greie resultat, men det har klare begrensninger.

4.2 Videre arbeid

Deler opp videre arbeid i to seksjoner. En for videre utvikling av funksjonalitet som vil forbedre eller utfylle programmet mer. Den andre seksjonen for funksjonalitet som fungerer, men som ikke fungerer optimalt med tanke på effektivitet av databruk.

4.2.1 Ønskede funksjoner

1. Bytte ut lakkerings-rektangler med ett polynom med to parallelle sider. De parallelle sidene skal tilsvare rektangelets lengste sider, og kortsidene skal følge objektet. Dette gjør at effektiviteten vil drastisk gå opp og aktuasjoner på robot holdes lavere. Dette krever at en ny baneplannleggingsalgoritme skrives. Siden systemet er modulært, kreves det ikke noen endringer med klassifisering eller vindu-algoritmen.

2. Endre vindu-algoritmen til å finne største parallellogram, istedenfor største rektangel. Dette gjør at algoritmen fyller flere vindu, og effektiviteten vil øke for enkelte former. Grunnen til akkurat parallellogram er å prøve å kutte opp ett objekt i former som er mest mulig rektangle. Til sammenligning kan ett trapes brukes, men kan da bare kutte figuren langs trapesets parallelle sider. Dette fjerner noe av den ønskede funksjonaliteten. Ett parallellogram derimot kan kuttes langs horisontale og vertikale linjer. Resten av vindu-algoritmen blir som den er. Resten av programmet trenger ingen modifikasjoner for å fungere. Derimot er det ønskelig at punkt nr 1 er realisert for å kunne ta maksimalt utbytte av denne ekstra funksjonaliteten.

3. Vindu-algoritmen tar nå bare utgangspunkt i ett senterpunkt til vinduet. Å sjekke for flere punkter kan gi ett forbedret svar, spesielt når vinduet har formen til ett polygon.

4. Senterpunktet til vindu-algoritmen er alltid senterpunktet til rektangelet som fyller vinduet. Dette er fordi lengden alltid er lik fra senter og opp som fra senter og ned. Samme med sidene. Om avstand fra senter og opp er forskjellig enn fra senter og ned, kan det føre til at vinduet dekkes i en større grad. Dette er snakk om vinduer som har former som ikke er rektangulære eller kvadratiske. For disse formene er allerede det optimale svaret brukt, som er tyngdepunktet av vinduet. For polynomer er ikke alltid tyngdepunktet det optimalet svaret, og derfor kan det forventes forbedret ytelse for disse. Spesielle former som figur 4.10, hvor tyngdepunktet ikke er i figuren, krever også en annen metode for å få ett godt svar.

5. Vindu-algoritmen sjekker om rektangelet som fyller vinduet er på innsiden av vinduet. Dette gjøres ved å sjekke forskjellige piksler om de er på innsiden. Denne funksjonen

er opencv funksjonen som er skrevet om i avsnitt 2.4. Denne funksjonen gir ikke alltid ett korrekt svar. For å utbedre denne funksjonen, kan en sammenligne svaret med en test om at pikselen er hvit eller sort. Dette kan gjøres siden det er snakk om et binært bilde, hvor vinduet alltid er sort, og objektet alltid er hvitt. Grunnen til at sort/hvit sjekken ikke kan brukes alene, er at den ikke sjekker om punktet er på innsiden av vinduet.

6. Når vindu-algoritmen er ferdig og har delt opp ett objekt med vindu, kan det oppstå mindre biter som står alene. De mindre bitene står ofte tett på en annen større bit. Å kunne slå sammen biter av objektet slik at programmet blir mer effektivt hadde vært en ønsket funksjon. Dette er en funksjon som må lages fra grunnen av, og kan føre til at programmet blir noe mer effektiv og sparer aktuasjoner for robot.

7. Programmet har en algoritme for å dele opp objekter med vindu, men mangler en funksjon for å dele opp objekter som ikke har vindu. Objekt som figur 4.5 har stort forbedringspotensiale for å kunne splittes opp i mindre biter. En algoritme som splitter opp slike objekter på en god måte gir stor forbedring av effektivitet for slike objekt.

4.2.2 Optimalisering

1. Funksjonen som tester om punkter ligger på innsiden av en kontur, skrevet om i avsnitt 2.4 er tidkrevende. Funksjonen brukes aktivt i vindu-algoritmen og kan bli kalt noen tusen ganger per objekt. At denne funksjonen er resurskrevende gjør at programmet kan bruke inntill 5 sekkunder per objekt. Dette spørs veldig fra objekt til objekt, men ett høyoppløst objekt bruker lengere tid enn ett som er lavere oppløst. Som nevnt i punkt 5 under "Ønskede funksjoner", er ikke denne funksjonen korrekt hele tiden. Kombinasjonen av en tung funksjoen med feil, gjør at denne har nytte av å bli utbedret.

2. Ved starten av vindu-algoritmen velges startpunkt. Dette startpunktet er tyngdepunktet til vinduet. Om tyngdepunktet ikke er på innsiden av vinduet, kjøres fortsatt søkfunksjonen som resulteres i feilmelding. En funksjon for å sjekke om startpunktet er gyldig, vil gjøre kjøretiden kortere for objekt av denne typen.

Kapittel 5

Konklusjon

Hovedobjektet med denne oppgaven, var å lage programvarebiten for å generere baner for lakkeringsrobot ved hjelp av ett data-syn kamera. Dette har blitt gjennomført. Oppgaven beskriver ett program som greier å løse oppgaven den var satt til å gjøre. Resultatet har varierende resultat med tanke på den geometriske formen til objektet som skal lakkeres. Når objektets form ligner en rektangel gir programmet en meget god løsning som diskutert i kapittel 4. Om objektet tilegner seg form som eksempelvis en "C", løser programmet oppgaven på en lite effektiv måte.

Siden resultatene er av varierende effektivitet, er det mulig å effektivisere metoden som er brukt. På "Videre arbeider det diskutert måter på hvordan lage programmet mer effektivt. Disse metodene legger til rette for at grunnprinsippet er det samme, men metodene er endret slik at resultatet vil bli bedre.

Figurer

Bibliografi

[1] Chen Wei and Zhao Dean. Tool trajectory optimization of robotic spray painting. In *2009 Second International Conference on Intelligent Computation Technology and Automation*, volume 3, pages 419–422. IEEE, 2009.

[2] 50 år siden jærbuenes banebrytende oppfinnelse av lakkeringsroboten. `https://kommunikasjon.ntb.no/pressemelding/50-ar-siden-jaerbuenes-banebrytende-oppfinnelse-av-lakkeringsroboten?publisherId=5310709&releaseId=17872282`. Accessed: 09-07-2020.

[3] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[4] Michael lail Shamos. *COMPUTATIONAL GEOMETRY*. PhD thesis, Yale University, 1978.

[5] Godfried T Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, page A10, 1983.

[6] About OpenCV. https://opencv.org/about/, 2020.

[7] Machine Vision Guide - Camera Calibration and World Coordinates. `https://docs.fab-image.com/current/studio/machine_vision_guide/CameraCalibrationAndWorldCoordinates.html`. Accessed: 22-06-2020.

[8] Christian Knauer, Lena Schlipf, Jens M Schmidt, and Hans Raj Tiwary. Largest inscribed rectangles in convex polygons. *Journal of discrete algorithms*, 13:78–85, 2012.

[9] N.V.S. Abhilash. Opencv contour scale rotate. `https://github.com/nvs-abhilash/tutorials`, 2019.

Tillegg A

Starte program

Nødvendig programvare:

Program	***Version***	***Info***
Python	3.8	Generell Python 3 vil nok fungere
OpenCV	4.8	Eldre versjoner vil nok fungere
Numpy	1.18	—

Beskrive innganger til program, 'lakkeringsbane'

Navn	**Krever**	**Tilleggsinfo**
img	cv.imread('bilde.png')	Bildet som programmet skal lage bane over
aks	int	Ant piksler som kreves for akselrasjonslengde
diameter	int	Piksler diameter på lakkeringskjegle
overlapp	int, mindre enn 'diameter'	Piksler overlapp mellom lakkeringskjegler
smaaObjekt	bool	Velge om små objekt skal være med eller ikke. Små objekt er rektangler der den korte siden er tynnere enn diameter
inkX	int	Piksler inkremet for fylling av hull, x-akse. Optimalt→ 1, Raskest → eks. 3
inkY	int	Piksler inkremet for fylling av hull, y-akse. Optimalt→ 1, Raskest → eks. 3
inkVink	int	Grader inkrement for rotasjon av rektangel som fyller hull. Optimalt → 1, Raskest→ eks. 3
margin	int	Piksler margin som legges til hver side av lakkeringsrektangel

Eksempel på hvordan starte programvaren. Her blir alle robot-koordinatene lagret i rekken 'bevegelser' og lagret som en 'pickle'.

```
import cv2
from lakkeringsbane import lakkeringsbane
import pickle

ink = 1
bevegelser =    lakkeringsbane(cv2.imread('bilde.png'),"bildenavn", aks=15,
                diameter=30, overlapp=5, smaaObjekt=False, inkX=ink,
                inkY=ink, inkVink=ink, margin=3)

with open('listfile.data', 'wb') as filehandle:
    # store the data as binary data stream
    pickle.dump(bevegelser, filehandle)

cv.waitKey(0)
cv.destroyAllWindows()
```