# Java

Java delen af projektet skal kunne læse en billedfil fra en given destination på computeren, hvorefter billedet vil blive lavet om til en String og efterfølgende sendt videre til vores PLC program, som er i direkte kontakt med tegnerobotten. Java-koden gennemgås i denne del af rapporten med fokus på kodens main class, *PictureinOOP*, hvor der vil blive henvist til de andre classes i koden.

**PictureinOOP**  
I starten af main-klassen befinder hele koden sig inde i et while(true)-loop for at den kan køres gentagende gange hvis ønsket af brugeren. Det første der sker inde i dette loop er kreering af et objekt *sc*, af klassen *scanner*, som stammer fra det importerede bibliotek *java.util.Scanner* fra linje 17 i klassen. Dette bliver benyttet til at tage imod input fra brugeren.  
Kodeblokken på linje 19-32 læser en billedfil fra projektmappen og konverterer filen til et objekt af *BufferedImage*, via et import af biblioteket *java.awt.image.BufferedImage*. Kode blokken starter med at deklarere variablet *name* af typen String. Derefter deklareres et objekt af *BufferedImage*, *cImage* initializes med værdien null. *fileName* og *cImage* deklareres her, da de skal bruges i koden senere, og initializes med en brugbar værdi i et while(true)-loop derefter. I det føromtalte while(true)-loop prompter programmet til at skrive et filnavn tilhørende et billede i projektmappen. *fileName* bliver her sat lig det input, og via en try multi-catch læses filen. I kodens try-del af statementet sætter vi vores før-deklarerede *cImage* lig en billedfil med samme navn som der er blevet skrevet i brugerens input tidligere. Dette bliver gjort via importering af *java.io.file* og *javax.imageio.ImageIO*. Kodens catch tager højde for *IOExceptions* og *IllegalArgumentExceptions,* da de to metoder fra det importerede bibliotek kan kaste dem. Via ImageIO’s statiske metode read læasses en billedefil der inde i metoden read kreeres, som et objekt af *File*, hvor *fileName* er et parametre. File laver fileName om til en fil og read kan derfra læse sagte fil, read retunere et *BufferedImage* og *cImage* kan altså sættes lig det read retunerer. *ImageIO* kan læse PNG og JPEG, som blev benyttet til at teste programmet og robotten, men den kan også læse BMP, WBMP og GIF filer. *File* kan kaste en *nullPointerException* hvis *fileName* er lig null, men da vi giver den en værdi via brugerinput, kan den ikke have det på det tidspunkt i programmet, altså gribes den exception ikke i catch-statementen. For at kunne gribe *IOExceptions* er *java.io.IOException* importeret. *IOExceptions* gribes hvis *cImage* bliver sat lig null og *IllegalArgumentExceptions* gribes hvis en error sker i løbet af læsningen af billedet. I denne kode kunne det fx ske hvis noget forkert er indtastet af brugeren ved brugerinput.

*pictureArray*, et multi-array af typen byte, kreeres på linje 36 i koden, hvis længde er sat lig bredden af vores valgte billede i pixels. På linje 40 til 45, laves der et objekt af klassen *Image*, og der benyttes en af dens metoder, *assembleImage*, der udfylder det nævnte multi-array (*pictureArray*).På linje 48 printes der på konsollen til brugeren, hvilket limit der er sat for mængden af pixels et billede må havde før det skaleres, samt hvor mange pixels det valgte billede har. Brugeren bliver promptet i linjerne 51-85 til at vælge imellem det fulde billede, eller en udvalgt del af billedet. Hvad end billedet skal skaleres eller ej, promptes der om det samme, og svarene brugeren må give er også ens. Uden at brugeren nødvendigvis ved af det, skaleres billedet hvis det overskrider billedets pixel limit. Limit er som der nu, sat for at teste programmet, videre forklaring kommer i diskussionen.  
  
På linje 88-90 deklareres og initialiseres objektet *roboC* af klassen *RobotClient*. Via variablerne port af typen int som er sat lig 12345 og hostName af typen String som er sat lig "192.168.0.103" *RobotClient* er et library givet af vores professor. Setuppet af denne server og TCP forklares i PLC delen af rapporten.

På linje 92 kreeres et objekt, *scaledImage* af klassen *Scale*, så objektet kan benyttes længere nede i koden, inde i de følgende if-statements til at skalere billedet.

På linje 94-221, afhængigt af om brugerens input på linje 51-85 er FP (*full picture*) eller SP (*specific part of picture*), og om billedets originale mængde af pixels overskrider det satte limit, kan der forekomme fire forskellige udfald.   
Det første mulige udfald, her for forklaringens skyld kaldt FP1, konverterer hele billedet til en String via objektet *code* af klassen *Message,* via metoden *convertToMessage*. Derefter tegner den, for brugeren skyld, billedet i hhv. 1 og 0, med mellemrum bagved hvert tegn, via objektet *photo*, og printer omtalte String. Endvidere forbinder den til PLC’en, via en server og sender billedets String, derefter hvis den er forbundet til serveren, bryder den forbindelsen afhængigt af brugerinput.  
På linjerne 104-117 sendes det fulde billede til PLC’en, I denne kode blok sendes den sammensatte String der sammensættes i metoden *convertToMessage* fra klassen Message til PLC’en. Metoden connect forbinder til serveren og videre til PLC’en via objektet *roboC*.  
Endvidere, såfremt programmet fortsat er forbundet til PLC’en, prompter det brugeren til at bestemme, hvorvidt den skal afbryde forbindelsen, og såfremt brugeren skriver alt andet end “YES”, prompter den brugeren igen via dens while(true) loop. Såfremt brugeren skriver “YES”, bliver forbindelsen afbrudt, og via et break hopper koden ud af while(true)-loopet.   
I det tredje mulige udfald her kaldt SP1, prompter koden brugeren til at specificere hvilket område af billedet som skal tegnes, og via objektet *part* af klassen *PartImage*, sættes variablen *partMessage* af typen String til billedets kode for det specifikke område. Endvidere forbinder den til PLC’en, sender billedets String, og via brugerinput afbryder den.  
Et fjerde udfald er FP2*,* hvor koden skalerer billedet og beregner forholdet hvorved der skal skaleres via objektet *scaledImage,* der blev kreeret på linje 92 i koden. Ved hjælp af *scaledImage* tegnes billedet for brugeren med hhv. 1 og 0, med mellemrum bagved, hvorefter billedet bliver konverteret til en String, og denne String udskrives til brugeren. Endvidere forbinder programmet til PLC’en, sender billedets String, og via brugerinput afbryder den.

Det sidste udfald, kaldt SP2*,* skalerer koden billedet og beregner forholdet hvorved der skal skaleres via objektet *scaledImage,* der blev kreeret på linje 92 i koden. Brugeren promptes til at specificere det område af billedet som skal tegnes, og via objektet *scaledImage* tegnes billedet i hhv. 1 og 0, med mellemrum bagved samtlige tegn, og variablen *scaledMessagePart* af typen String sættes lig billedets kode for det specifikke område og printes til brugeren. Endvidere forbinder programmet til PLC’en, sender billedets String, og via brugerinput afbryder den.

Ved linje 224-232 bliver brugeren promptet om hvorvidt de vil fortsætte programmet eller lukke programmet, og ud fra hvorvidt brugerens input er “NO” eller alt andet, hopper processen ud af *while true*-loopet som hele koden er i, eller programmet starter loopet forfra.  
 **Andre klasser   
Image**  
Klassen *Image* indeholder 2 attributes, 1 constructor og 2 metoder. De to private attributter er *iHeight* og *iWidth* af typen int, som i constructoren, respektivt bliver sat lig med parametrene *newIheightt* og *newIwidth* begge af typen int, som I programmets main, *PictureinOOP*, respektivt bliver sat til billedets højde og bredde.

Denne klasse indeholder metoden *assembleImage*, der benytter parametrene byte *pictureArray*[ ][ ] af typen byte, og *image* af typen *BufferedImage*, til at udfylde et multi-array med hhv. 0 og 1, castet til typen byte, for henholdsvis hvid og sort. *pictureArray* er af typen byte, for at spare på pladsen, da der ikke er brug for mere plads for et 1 og 0. Metoden begynder med at kreere objektet *ofPicture* af klassen *color*, dette gøres for at gemme værdierne et samlet sted. Et dobbel for-loop påbegyndes for at trappesere *pictureArray* fra main. Inde i det ydre for-loop, der gennemgår billedets bredde, kolonne for kolonne af pixels, sætter et nyt array med en størrelse lig højden af billedet ind i hver indeks, så *pictureArray*, værdimæssigt, så kan fyldes ud. For, hvert *y*- og *x*-værdi der er inde for grænserne sat af billedets bredde (*y*) og højde (*x*), sker der følgende:  
*Color* harattributterne *rgb, red, green, blue,* og de har hver deres accesmetoder. *Rgb* sættes via *getRGB*, en metode fra *BufferedImage*, der giver en farvekode for farven af et pixel, eller her en index i multi-arrayet. Et fx på hvordan de tre farver sættes er, ”*ofPicture.red = (ofPicture.rgb>>16)&0xff;*”. Man bruger >> til bitvis at gå gennem farvekoden. Den specifikke farve værdi kan findes byte vis og koden gennemgås bitvis derfor med 0, 8 og 16 til at finde værdien for farverne rød, grøn og blå i hvert pixel i billedet. Af disse farver findes gennemsnittet, via objektet *ofPicture*, så vi kan se billedets pixel i hvor ”lyse” de er, og dermed ikke i farver. Ud fra dette sættes en ny *RGB*, som er gennemsnittet for hvert pixel, via *getRGB*, en metode fra *BufferedImage*. På linje 43 til 47 sættes hvert pixel/index lig 0 eller 1 casted til byte, afhængig af om gennemsnittet er lig med eller højere end 160, eller om det er lavere. Hvis det er lavere, er farven tæt nok på sort til at koden konvertere det til 1 og omvendt hvis det er over eller lig med 160, konverteres det til 0.

Metoden *drawImage* har parameteren *pictureArray* af typen byte og står på linje 54 til 65 i denne klasse. Den benytter dobbelt for-loopet til at trappesere *pictureArray* og printer et 0 eller 1 (visuel reportage af pixels og dets farve) i konsollen, afhængig af arrayets værdi i det givne index i arrayet.  
For det ydre for-loopet gælder det, at int *y* får den givne startværdi 0, og såfremt dens værdi er lavere end højden af billedet, så vil der blive lagt +1 oveni værdien for y. Såfremt disse kriterier bliver opfyldt, vil det indre for-loop blive gennemgået, For det indre for-loop gælder det samme, blot med værdien for int *x* og bredden af billedet, i stedet for *y* og højden af billedet.   
På linje 57, såfremt at værdien i indekset som processen er nået til, er lig 0, så vil der blive printet et 0 efterfulgt af et mellemrum og såfremt dette ikke er tilfældet, vil der blive printet et 1, ligeså efterfulgt af et mellemrum.   
Til sidst på linje 63, sørger koden for at efter hver række af pixels er gennemgået og der hoppes ud til det ydre for-loop igen. Her sørger koden for at der printes på næste linje af konsollen og dobbelt for-loopet er færdig med en række. Heraf tegnes billedet række for række.   
  
**Color**Klassen Color indeholder 5 attributterne *rgb, red, green, blue* og *average*, som står på linje 5-9 og alle af typen int og alle er private. Klassen indeholder desuden en no-arg constructor, samt 6 metoder, 2 accessmetoder og 4 mutatormetoder.  
Accessmetoderne i denne kode er som følger: *getAverage*  der returnerer attributen *average*, da den er private. Derimod lidt atypisk for accessmetoder, beregner den *average* i metoden, og på en måde er den dermed også en mutatormetode. En anden metode er *getCrgb* der returnerer attributen *rgb*, da dette er en private attribute. *Crgb* står for *RGB* værdien i klassen *Color*, og benyttes da det importerede library, *java.awt.image.BufferedImage*, allerede har en *getRGB* metode der bruges i klassen *Image*.  
De 4 mutatormetoder gør det muligt at sætte attributterne *rgb, red, green* og *blue* til en værdi indenfor grænserne af typen int, hvilket benyttes i klassen *Image*.

**Robot Control Code**‘D’ står for *Down* og at robotten skal tegne her, samt skifte plads til næste pixel på x-aksen. ‘U’ står for *Up* og at der ikke skal tegnes, men der kun bevæges til næste pixel på x-aksen. Up og Down beskriver vores y akses position, hvorvidt pixelen er sort eller hvid. N står for *Next Line* og robotten skifter til næste række af pixels af billedet. ‘Q’ står for *Quit*, som betyder at tegningen er færdiggjort og at robotten skal stoppe med at tegne. Det kan forekomme, at der i den kreerede String kan stå et ‘N’ før ‘Q’, hvilket ville få robotten til at skifte linje blot for derefter at afslutte, men i dette tilfælde vil robotten blot afslutte fuldstændigt, robotten ikke kan forstå sagte kombination, “NQ”. Dette er umiddelbart en fejl i koden, men det har ikke nogen indflydelse på det færdige resultat, da robotten alligevel skal stoppe.  **Message**Klassen *Message* indeholder 2 private attributter på hhv. linje 6-7, som lyder som følgende:  
*upOrDown* af typen char, og *message* af typen String. Derudover indeholder klassen også en no-arg constructor på linje 9-10, som sætter attributten *message* lig en tom String.   
Klassen indeholder også metoden *convertToMessage* på linje 13, med parametrene byte *pictureArray* og BufferedImage *image*.  
*convertToMessage* sætter en String sammen, baseret på *pictureArray* fra main og kombinationen af følgende bogstaver: ‘D’, ‘U’, ‘N’ og ‘Q’.

Den sætter sagte String sammen på følgende måde. Metoden trappeserer *pictureArray*, via et dobbelt for-loop der er på linje 14-31. Det ydre for-loop gennemgår billedets pixels via et varible af typen int. Billedet gennemgås række for række, en række for hver y-værdi, y’s maximum værdi er altså billedets højde. For hver række af pixels eller for hver y-værdi, gennemgår det indre for-loop, via variablet *x* af typen int, hver kolonne, der tilsammen er bredden af billedet, hvilket altså er *x*’s maximum.   
Inde i det indre for-loop sker følgende: Hvis *x* er under højden af billedet,og *x* er under bredden af billedet, så sættes attributen *upOrDown* lig enten ‘U’ for “Up”, eller ‘D’ for “Down”, afhængig af om det givende indeks, som koden er nået til, er lig 0 eller ej. Derefter bliver *upOrDown* sat i slutningen af attributen *message*. Dette har en begrænsning via et if-statement for at undgå en *OutOfBoundsException* når billedet ikke har lige lange sider. Altså kan koden kun sende en String for enten et kvadratisk billede, eller kun et kvadratik stykke af billedet, hvis størrelse afhænger af dets bredde eller højde, afhængigt af hvilken der er mindst. I det ydre for-loop, efter gennemgangen af det indre for-loop, sættes der et ‘N’ på enden af *message*, som står for at robotten skal tegne næste linje. Efter hele dobbelt for-loopet sættes et ‘Q’ i enden af *message*, hvilket står for at robotten skal stoppe.

**PartImage**Klassen *PartImage* har 2 private attributes: *message* af typen String og *upOrDown* af typen char.Klassen indeholder også 1 no-arg constructor der sætter *message* lig en tom String og 2 metoder. På linje 13-21 står metoden *messagePart* der via parametrene int *x1*, int *y1*, int *x2*, int *y2* og byte[ ][ ] *pictureArray* sætter en String sammen ud fra samme bogstavs kombination som metoden *convertToMessage* fra klassen *Message*. Den sammensætter koden på samme måde, men med ændringen, at *y* og *x* i dobbelt for-loopet har startværdier og slutværdier der er sat via brugerinput i main. Metoden *drawImage* har samme ændringer som *convertToMessage*, og gør altså det samme som drawimage fra klassen Image, men ud fra to intervaller af x- og y-værdier.

**Scale**Denne klasse består bl.a af 7 private attributes, 3 af typen int, 2 af typen long, *upOrDown* af typen char og *message* af typen String. Klassen består også af en constructor og 8 metoder.  
Constructoren på linje 18-23 benytter sig af parameteren *BufferedImage image* til at sætte attributen *iHeight* og *iWidth* af typen long, til respektivt billedets højde og bredde. Constructoren sætter attributen *message* af typen String til en tom String, sådan at den har en værdi før den benyttes i metoder længere nede i koden. Constructoren beregner også mængden af pixels i billedet, ved at gange *iHeight* og *iWidth*.  
Metoden *amountOfScaling* på linje 40-56 udregner forholdet hvormed billedet skal skaleres og sætter attributten *n* til denne værdi, sådan at resten af de metoder der er i denne klasse har lettere adgang til den. Metoden beregner *n* ved at dividere mængden af pixels i billedet med et variabel der skjuler attributen *n*, altså en variabel med samme navn. Derefter hvis *spAmount,* som afhænger af variablen *n*, er under det limit vi har på mængden af pixels, eller hvis *n* er lig 2, så er variablens værdi godkendt. *spAmount* afhænger af *n* da den beregnes ved at trække *pAmount* divideret med *n* fra *pAmount*.  
Variablens værdi starter på 1000 og formindskes med 1 for hver gennemgang af det while(true)-loop koden er indkapslet i, indtil det bliver godkendt. Attributten *n* sættes lig variablen *n* når værdien er godkendt og der hoppes ud ad while(true)-loopet via et break. n kan være lig 2 og blive godkendt i *amountOfScaling* fordi at enten er 2 det eneste tal der går op i billedets pixels, eller ingen gør, med undtagelse af 1, og 2 sættes dermed som standard for billedet.  
*getValueN* er en accesmetode der returnerer *n*, så der er adgang til den i main, og derfra via metodernes parametre, har de også adgang. *getValueSPAmount* er en metode der returnerer attributten *spAmount,* og ligeså returnerer *getValuePAmount* attributten *pAmount*.  
På linje 58-76 er der en metode, *drawImage,* der printer billedet ud i 1’er og 0’er, med et mellemrum bagpå, ligesom *drawImage* i *Image*-klassen, men dette sker i forhold til *n*. Når *pictureArray* trappeseres via et dobbelt for-loop, gennemgås det indre for-loop, hvis *y* divideret med *n* ikke har en rest på 0. Yderligere køres koden inde i det indre for-loop kun, hvis x divideret med *n* ikker giver 0. Derved fås den effekt at alle rækker hvor *n* går op i *y* og alle de kolonner hvor *n* går op i x, forsvinder.  
På linje 78-104 står metoden *convertToMessage.* Den sætter en String sammen ligesom *convertToMessage* i klassen *Message*, men med brug af *n* på samme måde som denne klasses *drawImage*.  
På linje 109-131 står metoden, *scaledMessagePart, der* sætter en string sammen ligesom *messagePart* i *PartImage* klassen, men med samme ændringer som i ovenstående metode, *convertToMessage* i denne klasse.  
På linje 134-149 står der metoden, *drawImage,* som er en overloading af metoden *drawImage* længere oppe i klassen på linje 58-76. Denne metode printer et billede på samme måde som *drawImage* fra klassen *PartImage*, men med samme ændringer som *drawImage* på på linje 58-76.

**Diskussion**  
Interfaces og abstrakte klasser:Java kodens programmeringsfase startede før der var styr på hvad og hvordan abstrakte klasser og interfaces fungerer, derfor er de ikke inkorporeret. Hvis tiden var til det, ville et fokus være blevet sat på interfaces, da koden har metoder som *drawImage* der benyttes i næsten alle klasser. Det kunne også hjælpe på at få sat et system op for programmets klasser. Mange af klasserne kunne have været sat sammen, eller en klasse som Scale kunne være blevet splittet op, da der som det er nu, er klasser med få metoder og klasser med mange, og rent orden mæssigt er det lidt et rod. Her kunne interface og abstrakte klasser have sat et hierarki og dermed gjort det lettere og mere naturligt at få holdt en hvis orden på klasserne.

Vedrørende orden på klasser, så er der en klasse, *Color*. Den blev skrevet, da den første virkende version af koden i starten af projektet, ikke specifikt var objekt orienteret, og derfor skulle omskrives. Det første forsøg på det, var *Color*. Den blev senere unødvendig og blev til intet andet end en samling af farve-værdier og beregning af gennemsnittet af farven i et pixel. Den er altså ikke nødvendig, men forblev, da tiden ikke var til at fjerne den fra koden og da den ingen negativ virkning har, udover øgelse af dårlig klasse orden som er omtalt længere oppe i diskussionen.  
Måden skaleringen i programmet fungerer, er ved at sammensætte en String til robotten og tegne på konsollen, hvor pixels hoppes over i processen og derved forsvinder rækker og kolonner af pixels i billedet. En anden måde at gøre dette, som bibeholder formen af motivet i det originale billede sammenlignet med den nuværende metode, der direkte kun fjerner pixels, er hvor der tages højde for de omkringliggende pixels. Det pixel der er i fokus får sin farve sat i forhold til de omkringliggende pixels som i stedet fjernes.  
I den nuværende måde at skalere på, er det ikke muligt at skalere billeder til mindre end halv størrelse. Når et billede bliver skaleret til halv størrelse, fjernes der hver anden værdi i arrayet på y, hvorefter de bliver fjernet på x. Det er ikke muligt for koden at fjerne f.eks. to af tre pixels på både x og y. Den størst mulige skalering ved nuværende metode, hvor koden kun køres en gang, er halv størrelse. Hvis et billede skulle skaleres yderligere ville det være ved at skalere to gange. Det ville muligvis kræve at der laves et array ud fra det skalerede billede for at kunne skalere mere end en gang. Altså ville man skalere en gang med det originale array og igen med det nye array. Afhængigt af hvor mange gange der skal skaleres, skal der laves samme mængde nye arrays.

Når programmet tegner et billede i konsollen, ligger billedet på siden. Hvis tiden var til det, kunne det sandsynligvis rettes ved at ændre på rækkefølgen af gennemgangen af indekser. Derudover tegnes billedet via robotten og via konsollen, spejlvendt. Grunden kan ikke siges med sikkerhed, men da det ikke er meningen, er det sandsynligt når billedet læses i main, at det spejles. Dette, hvis tiden var til det, kunne sandsynligvis rettes ved at gennemgå rækkerne af pixels den modsatte vej i alle String sammensætnings- og tegnemetoder. Altså ved at lade x i de indre for-loops gå fra maksimumværdi mod 0, i stedet for at gå mod maksimum, som koden gør nu.

Endnu et problem koden har er at, af grunde ikke helt klare, kan metoderne der sammensætter en String på baggrund af et helt billede når “FP” er valgt af brugeren i main på linje 57-85, ikke håndterer et billede der ikke er kvadratisk, som ønsket. Det vil sige at de dobbelt for-loops gennemgår kun er en kvadratisk del af billedet, for ikke at få en *ArrayIndexOutOfBoundsException*, når der af en ukendt grund, spørges efter et index ude for grænserne sat af billedets højde og bredde. Altså har programmet en løsning, men en bedere en, som, hvis tiden var til det, kunne indføres ville være at gennemgå resten af billedet kvadratisk, stykke for stykke, indtil hele billedet er gennemgået. Hvis brugeren valgte “SP” så kan koden godt håndtere ikke kvadratiske billeder, og dette valg kan benyttes hvis brugeren kender til billedets længde og bredde, eller som anden forsøg hvis koden ikke gav det ønskede output.

“SP” valget kan benyttes som en løsning i ovenomtalte problem, “SP” har dog et problem da, hvis det bliver valgt, så promptes brugeren til at indskrive værdier af typen int, til at vælge hvilken del af billedet der skal tegnes. Hvis værdierne endten ikke er valgt inde for billedets grænser (højde og bredde) eller hvis de ikke er af typen af int, kan det skabe problemer, som fx kan det kaste en *ArrayIndexOutOfBoundsException*. Hvis tiden var til det, kunne det fixes med et while(true)-loop eller et try-catch-statement.

Til sidst i main, når brugeren promptes til om programmet skal køre forfra eller slukke på linje 244-250, kan et problem opstå afhængigt af om der vælges FP1, FP2, SP1 eller SP2. Programmet venter ikke på brugerens input, men starter automatisk programmet forfra, hvis SP1 er valgt og selvom det ikke er testet, kan SP2 have samme problem.

Byte benyttes som typen på arrayet *pictureArray* på grund af at arrayets størrelse ikke behøvede at være større for at indeholde 1 og 0, castet til byte og derfor er der ikke en grund til at benytte større datatyper. Det var tanken bag det, men et problem opstod i forhold til at programmet ikke kunne håndtere et billede når dets breddes eller længdes størrelse er omkring 256 eller højere. Der var på grund af andre problemer der dukkede op på samme tid, ikke tid til at teste præcis det maksimum der er, men det formodes at hvis de arrays der ligger i multi-arrayet *pictureArray* er større end 256, så er størrelsen af byte ikke stor nok. Altså hvis billedets højde er længere end 256 pixels, så kan et array på den længde ikke være i multi-arrayet.  
Den limit programmet har for om billedet skal skaleres, er specifikt sat til 257 gange 257. Dette er en sat værdi, som skulle være blevet brugt til at teste den limit der sættes af byte i *pictureArray*. Meningen er dog at hvis tiden var til det, ville limit afhænge af regnestykket, 65535 - antal af ‘N’er i dens sammensatte String minus 1 for ‘Q’. Altså for hver række af pixels i billedet skal der fjernes et pixel fra det maksimum limit har, sat af TCP, når der sendes til PLC’en, og da ‘Q’ står for at robotten stopper, fjernes der også et pixel for det. Da limit er sat til at et billede kan havde 257 som højden, så er der et hul i programmet hvor at et billede programmet ikke kan håndtere ikke skaleres til et billede der kan håndteres. Lidt alla det samme kan ske hvis et billede skaleres, men ikke skaleres til under 256 gange 256 fordi det originalt er for stort. Altså kan programmet komme i problemer afhængig af størrelsen af billedet eller højden af billedet.

Koden kan forbedres på forskellige måder, her er nogle af de forbedringer der med mere tid ville have blevet indført.  
Greyscale delen af koden kan udvides. *assempleImage* definerer farven ved en gennemsnitlig RBG værdi. Hvid defineres med et RBG gennemsnit på 161-255 og sort med et RBG gennemsnit på 0-160. Det ville være muligt at opdele farverne i flere kategorier, hvor opdelingerne er udsnit af RBG gennemsnittet. Herefter ville det være muligt at tildele de nye farveværdier til et sæt nye trykværdier til PLC’en. Hvilket ville gøre det muligt at skabe billeder med flere grå nuancer i stedet for kun sort og hvid. Dog ville det kræve at der blev opstillet flere tegn i vores String, for at definere hvilken grå værdi der skal gives videre til PLC’en, ‘U’ og ‘D’ ville ikke længere være nok.   
En anden forbedring er at få robotten til at tegne begge veje. Når robotten tegner billedet, gennemgår den hver række fra dens startpunkt på x-aksen til billedets fulde bredde, og når en række er gennemgået, før den gennemgår den næste, går den tilbage til dens startpunkt på x-aksen. Dette bruger den en masse unødvendig tid på og det ville være muligt at forkorte den tid ved at skifte række efter at en række er tegnet og dernæst tegne den næste række af billedet når robotten kører tilbage til dens startposition i x-aksen. Dette ville eliminere den tid der spildes på at robotten bevæger “blyanten” tilbage til startpunktet på x-aksen.

Ved at ændre vores strengs værdier, til omvendt rækkefølge for hvert linjeskift, ville det være muligt at løse dette problem på Java siden af processen. Det ville dog også kræve en ændring af PLC programmeringen og den måde den reagerer på sagte String.  
  
Robotten bruger nogle gange unødvendig tid på at gennemgå billedet. en forbedring ville altså være at mindske denne tid.   
Et billede har hele rækker eller store dele af rækker, hvor dets pixels er hvide, så spilder robotten en masse tid på at gennemgå disse pixels, som ikke skal tegnes. Det ville derfor være en optimering at robotten skiftede til næste pixel når der kun er ‘U’er tilbage, før det næste ‘N’ i den String der sendes til PLC’en. På denne måde kan hele rækker erstattes med at gå til næste linje og det samme for det resterende af en række pixels, hvor resten er hvide. Dette ville dog hjælpe mere på Java siden af processen, da der så skal sendes en meget mindre String og altså kan maksimum på billedets størrelse øges. Man ville gøre dette ved den String der sendes til PLC’en bliver sat sammen baseret på bl.a om der, hen til slutningen af en række, fra det nuværende indeks, kun er 0’er. Hvis det er sandt, så sættes der kun et ‘N’ og ikke et stort og unødvendigt antal ‘U’er.

**Linkliste**

Farvekode i klassen Image:

<https://stackoverflow.com/questions/6126439/what-does-0xff-do>

<https://www.dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-grayscale-image-in-java>

<https://stackoverflow.com/questions/1050989/double-greater-than-sign-in-java>

<http://www.rapidtables.com/web/color/RGB_Color.htm>

Read file fra main klassen, PictureinOOP:  
<https://docs.oracle.com/javase/tutorial/displayCode.html?code=https://docs.oracle.com/javase/tutorial/2d/images/examples/LoadImageApp.java>

<https://docs.oracle.com/javase/tutorial/2d/images/loadimage.html>

<https://docs.oracle.com/javase/7/docs/api/java/io/File.html#File(java.lang.String)>

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#compareTo(java.lang.String)>

<https://docs.oracle.com/javase/7/docs/technotes/guides/language/catch-multiple.html>

<https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html#read(java.io.File)>

Array opbygning i klassen Image:  
<https://stackoverflow.com/questions/5925426/java-how-would-i-load-a-black-and-white-image-into-binary>

<https://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html>

<http://www.codejava.net/java-se/swing/show-simple-open-file-dialog-using-jfilechooser>

<https://stackoverflow.com/questions/5374311/convert-arrayliststring-to-string-array>

<https://stackoverflow.com/questions/28564157/how-to-fill-a-2d-array-with-a-1d-array>