Oppgave 1)

**2)** Vi bruker minste kvadraters løsning til samregistreringen, side 27 i forelesningsnotater om geometriske operasjoner.For å finne den affine transformasjonen, sender vi inn referansepunkter (fra masken) og tilsvarende i innbildet. Vi plasserer x og y-koordinatene til innbildet i en 3x3-matrise, samt x og y-koordinatene til referansebildet i to forskjellige 3x1 matriser. Vi gjør utregningen i to omganger, en for x-koordinatene og en for y-koordinatene. Resultatet blir transformasjonsmatrisen med koeffisientene på de to øverste radene, samt 0, 0, 1 på siste rad.

**4)** På grunn av begrensningene ved forlengs mapping, blir forlengstransformasjonen vår svært begrenset og med mange hull, da denne mappingen ikke fyller utbildet direkte. Ved invers mapping så inverterer vi forovertransformasjonen og fyller heller in utbildet piksel for piksel. Her plukker vi opp begrensingene til forlengstransformen ved å tilnærme en verdi som skal plasseres i utbildet. Ved nærmeste nabo runder vi bare av tallet som kommer ut av transformasjonen, mens vi med bilineær interpolasjon tilnærmer denne verdien ved å forestille oss en funksjon mellom pikslene som gir en gradvis overgang i gråtoneverdi mellom pikslene, i stedet for en hard terskling som ved nærmeste nabo.

Oppgave 2)

**6) a:** Her implementer vi en konvolusjon. Først må vi utvide bildet vi skal filtrere, dette må utvides med størrelsen på filteret rundet ned til nærmeste heltall (fordi vi går ut ifra at filterets størrelse er av odde lengde). Videre roterer vi filteret med 180 grader, før vi plasserer origo til filteret på første piksel i det opprinnelige bildet. Videre iterer vi gjennom bildet, summerer med filterets verdi som overlapper bildet og tilegner en ny verdi på et utbilde.

**B:** Først skriver vi en funksjon der vi oppretter et gauss-filter som senere skal brukes i konvolusjonen vår. Vi regner så ut magnituden til gradienten og gradientens retning ved hjelp av henholdsvis lengden på magnitudevektoren og arktangens-vinkelen mellom magnitudevektorens y og x. Disse verdiene lagres i to todimensjonale arrays som er like store som bildet.

Gradientmagnituden gir en fremhevet kant som ofte er større enn det vi behøver, så vi tynner ut disse kantene ved å finne ‘kjernen’ der kanten begynner. Ved hjelp av gradientretningen finner vi hvor kanten fortsetter og kvitter oss med unødvendige kantpiksler.

Dette tynne bildet sendes inn i en funksjon der vi manuelt har satt inn variabler for en høy og en lav terskel. Høy terskel er terskelen hvor, hvis pikselverdien er større eller lik den, denne definitivt er en del av kanten vi er ute etter. Pikselverdiene som faller mellom lav og høy terskel er vi mer i tvil om, og lagres i det nye behandlede bildet på en spesiell pikselverdi. Disse ‘svake’ pikslene skal vi håndtere ved hjelp av hysteresis.

I hysteresis er formålet å behandle disse svake pikslene. Vi ser på disse og finner ut om vi skal skrote de, eller gjøre de om til sterke piksler. Det bestemmer vi ved å se på 8-naboer og om noen av disse er sterke. Er en av de sterke, så blir den svake pikselen omgjort til sterk, ellers settes den lik null.

**8)** Resultatbilde ser bra ut, man kan se fra bilde «full\_plot» at det skjer en markant kantdetektering gjennom stegene som blir utført. Her blir Cannys algoritme brukt. Den har en fin kombinasjon mellom støyredusering og kantlokalisering. Hvis vi for eksempel hadde brukt LoG-metoden, ville det mest sannsynlig vært litt mer støy, men ikke mer enn hvis vi hadde brukt Laplace-metoden (veldig god kantdeteksjon, mye støy). LoG-metoden er en slags bedret versjon av Laplace-metoden. Det er en del støy i cellekjerner.png og derfor er Cannys fint til å komprimere mellom god støyreduksjon, men også en god kantdeteksjon!

Endring av tersklene førte enten til at det ble for mye støy eller så ble det rett og slett fjernet for mye kanter. Etter mye frem og tilbake endte tersklene på et nivå som ga fine kanter og lite støy.