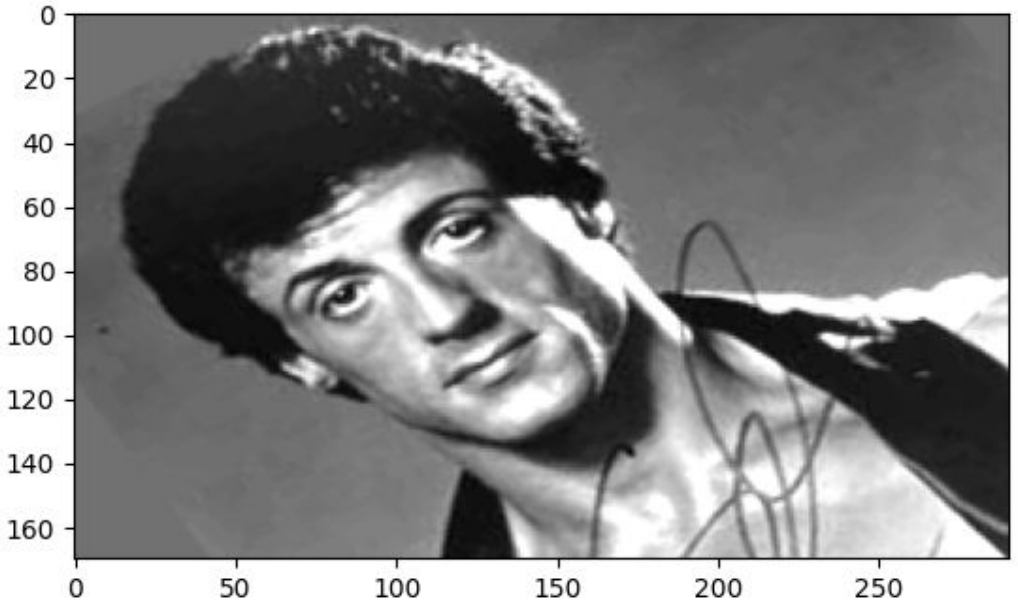
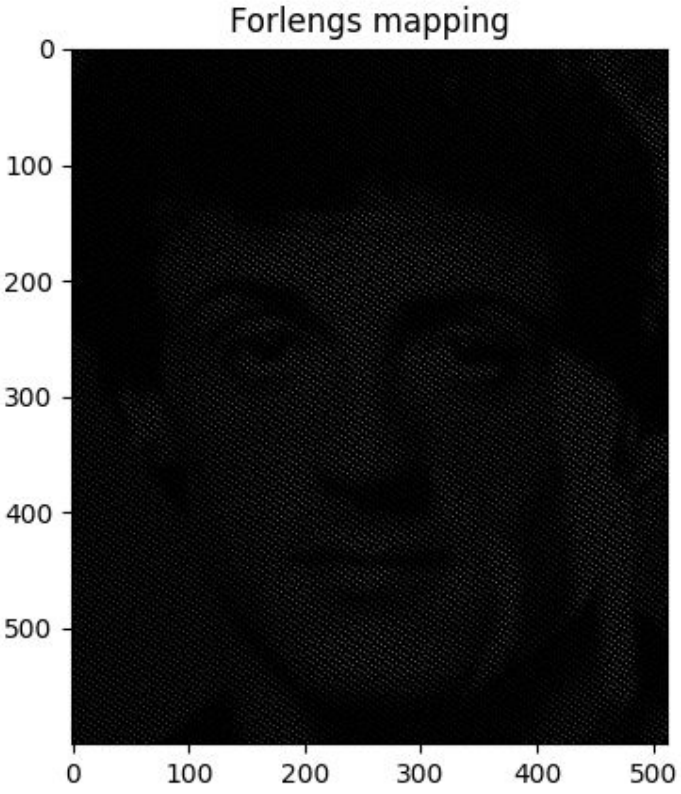
Oppgave 1

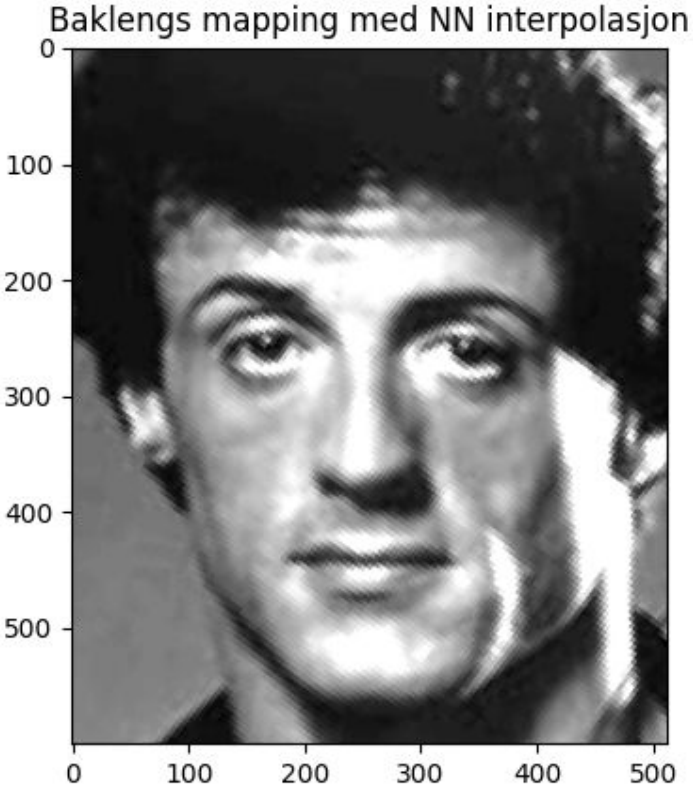
Resultatbilde fra 1a:

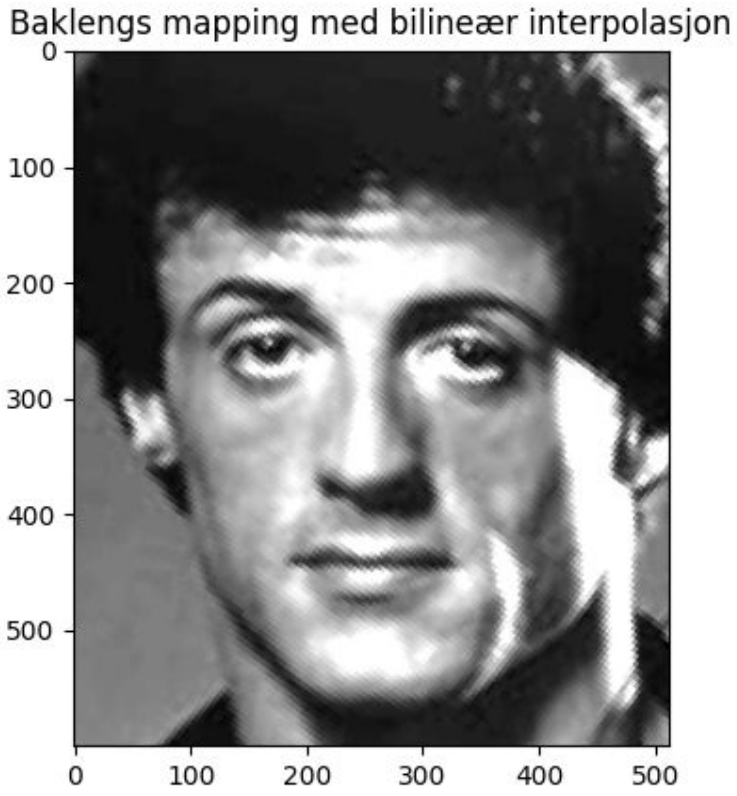


I oppgave a blir enten middelverdien og standardavviket feil, eller så får bildet verdier utenfor intervallet [0, 255], men det virker ikke som at det er mulig å oppfylle begge kravene. Resultatbildet blir uansett veldig likt.

Videre fant jeg koeffisientene til den affine transformasjonen i b) ved å finne fire like punkter i portrett.png og geometrimaske.png (begge øynene og munnvikene) og fant koordinatene manuelt ved å vise bildene i matplotlib. Deretter brukte jeg utregningen fra forelesningen til å finne transformasjonsmatrisen (s. 28 fra geometrioperasjoner).


Forlengs mapping
0
100
200
300
400
500
0
100
200
300
400
500


Baklengs mapping med NN interpolasjon
0
100
200
300
400
500
0
100
200
300
400
500



Det er stor forskjell på forlengs-mapping og baklengs-mapping, og dette er fordi forlengs-mapping ikke fyller inn hullene en ofte får med affine transformasjoner (spesielt forstørring), og en får da mange svarte piksler. Dette er tydelig, da bildet er veldig mørkt. De to baklengs-mappede bildene er veldig like, og begge fyller inn pikslene der informasjonen mangler, enten med nærmeste nabo, eller med en litt mer komplisert utregning.

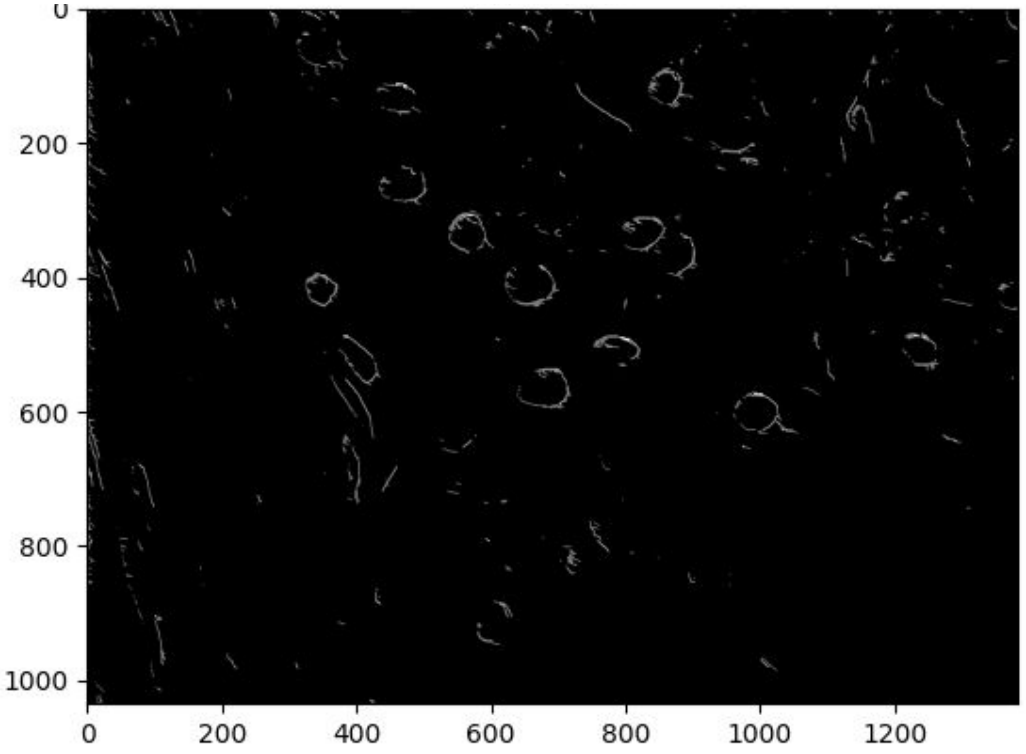
Oppgave 2

2a) består av funksjonene rotate180, utvid_bilde og konvolusjon. Først kalles konvolusjon, som utfører konvolusjon på et gitt bilde med et gitt filter, men først kaller på utvid_bilde for å legge til kanter til innbildet og kaller på rotate180 og roterer filtereret 180 grader. Rotate180 fungerer ganske intuitivt, og utvid_bilde looper gjennom innbildet og kopierer pikslene over, i tillegg til å legge inn nærmeste nabo om pikselen er utenfor det originale bildet. Konvolusjon funksjonen følger oppskriften gitt i forelesningen.

Videre i 2b) har vi gauss_filter, som lager et gaussfilter av en gitt sigma, og regner dermed ut sidelengdene, finner summen av alle vektene (fra forelesningsliden) og setter A (1/sum) slik at summen av vektene blir 1. Videre har vi calc_gradient som lager et bilde av gradient magnituden og retningen ved å finne gx og gy som i forelesningen, og deretter bruke atan() og sqrt(x**2 + y**2). Videre tar funksjone trim_edges bildet av magnituden, og sjekker det opp mot bildet av retningen, og fjerner/setter til 0 de som har en åtte nabo med eller mot gradient-retningen med større magnitude. Til slutt har vi hysterese funksjonen som looper gjennom det fortynnede bildet og setter alle pikslene med større verdi enn Th til å være

maks (Kunne brukt 1 bits bilde og satt til 0 og 1, men satte bare til 255), og de som er mellom Tl og Th og nabo til en markert piksel til å være maks også. Resten blir satt til å være 0.

Resultatbilde:



Resultatet ble ganske bra, men har fortsatt en del “støy” og en del kanter som ikke er fylt ut.
Parametre:

- Sigma: påvirker verdiene i gauss-filteret, samt størrelsen. Ettersom konvolusjonen tar en del tid, kommer programmet til å ta altfor lang tid å kjøre om sigma blir noe særlig større enn 1. Det er derfor vanskelig å si noe spesifikt om hva sigma endrer når det kommer til utseende til bildet, men ettersom gauss-filteret blir nok resultatet mer detaljert.
- Tl og Th: Tresholdene bestemmer hvilke verdier som blir markert (hvite) i hysteresen, og hvilke som blir svarte. Ettersom alt over Th automatisk blir markert, så påvirker mest hvilke nye kanter som kommer med i utbildet, mens ettersom alt mellom Tl og Th må være i kontakt med en markert piksel for å bli markert, så påvirker Th mer hvor utfylt de eksisterende kantene blir. Ettersom tynningen av kantene allerede har satt mye av kantene til å være 0, viste det seg at en veldig lav Tl funket best. Tl og Th ble respektivt satt til å være 2 og 46 i bildet over.