Oblig 1 2070

Mohamed Omar Atteyeh

March 26, 2021

Problem 1

Standardiseringen

Jeg brukte de forlesnings notatetene for å omgjøre mye av bilde detaljene standardesert. Det jeg brukte var en gråtone transformasjon som vises under ligningen 'oppgave 1' i koden. Jeg fikk en middelverdi på 127 og en standardavvik på 64. Bildet ser følgende ut etter transformen :

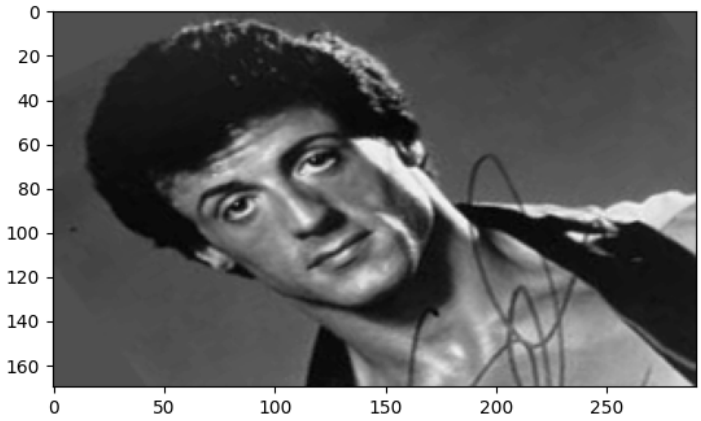
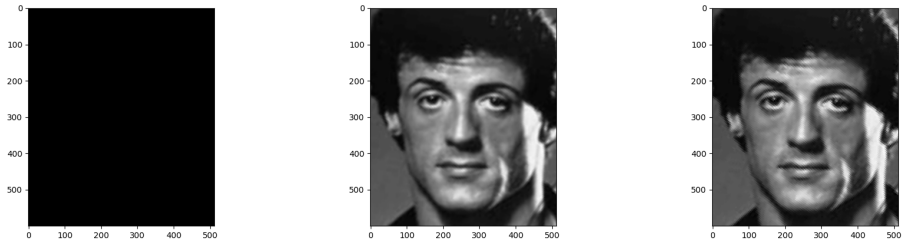


Figure 1: standardiseringen

Affintransformasjonen

For affintransformen visste jeg bildet, og valgte tre verdier på både maske bildet og portretten. Verdien var de verdiene som var på munnen, og begge øynene. Jeg skrev inn verdiene under (Tranfosmasjons matrice) i koden. Brukte dot produktet mellom de to og kom fram til et rimlig resultat etter implementasjonen ved bruk av forlengs mapping og bakleng mappingen:



(a) Forlengs (b) Bilinear (c) nærmeste nabo

Figure 2: Resultatet for både forlengs og baklengs mappingen

Diskusjon

Forlengs mapping ga meg et bildet som er helt nullet ut, mens begge baklengs transformene ga meg et lysere bildet. Antagelsen min er at forlengs mappingen min under transformasjonen nuller ut veldig mange av innbildet, og dette gjør da at hele bildet ser svart ut. I baklengs for vi bedre resultater fordi vi går gjennom alle piksler i utbildet og finner hvilken piksler fra innbildet må hentes. De er forskjellige operasjoner hvor i baklengs noen piksler hents og ikke alle, mens i forlengs alle hentes og overlappes med utbildet.

Problem 2

a)

Konvolusjonen fungerte slik at vi tok et bildet som beskrives som en array i python, og vi påvirker hver og en element i bildet slik at det resulterende bildet er forskjellig fra det opprinlige bildet. Konvolusjon er et matematisk operasjon, der array-et som kan sees som en matrice få en type interaksjon med en annen matrice. Det interaksjonet gir et nytt matrice, og det matrice

er hvor nye bildet. I denne oppgaven ble vi bedt om å konvolere og padde. Konvolusjonen kan man se under funksjonen konvolusjon() i koden. I koden oppstår det en interaksjon mellom bildet som er den første matrisen, og filtret som er den andre matrisen. Vi er nødt til å lage en ramme rundt bildet med nærmeste piksel verdier for å håndtere bildetrandproblemet, og den rammen da gjøre at vi utvider bildet. Den utvidelsen hjelper med interaksjonen med filtret som vi snur 180 grader. Vi lager en loop som går gjennom padded bildet og ganger hver piksel med filtret og summere de. resultatet blir et nytt bildet med nye piksel verdier.

b)

jeg kan fordele Canny algoritmen til 4 deler :

Gaussian blur

Gauss funksjonen skulle være filtret som jeg skulle bruke. Vi finner den ved bruk av en to dimensjonal gauss funksjon. Denne bruker som et filter for å konvolere bildet. Kant detektering kan lett påvirkes av støy som er på bildet, så vi bruker det for å hindre å ta med feil kanter, eller kanter som ikke er så nødvendige. Denne kjøres under (gauss(sigma)) funksjonen i koden. filtret vårt har en nxn størelse der $n = 1 + 8 \cdot \sigma$ og filtret har formen :

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Gradient

Den er en funksjon som er relatert til den derivert av intensitet verdiene til et bildet. Ved å partielle derivere en funksjon kan man finne gradienten. Her bruker vi gradienten for å finne både retningen og intensiteten av kantene. Vi bruker noe som heter sobel filtre for å finne gradienten i x og y retingen. I koden bruker vi den symmetriske 1D-operatoren hx og hy. Deretter konvolerer vi resultat bildet fra først konvolusjonen med gaussain blur som filter, med de 1D-operatorene. Videre bruker vi resultatet for å regne ut intensiteten av kantene og vinklene. Til slutt få vi verdier for vinklene, og intensiteten av kantene. Dette skjer under(gradient(innbildet)) i koden.

Non-maximum suppression

Dette er en teknikk for å gjøre kantene i det bildet vi ser tynnere. Dette gjør vi ved å analysere de fire retningene som er omringet pikselen. Man kan diskutere at de er 8 rettninger når man ta med seg de negative rettningene. Siden vi får vinklene og intensitetene, vi bruker de for å sjekke om det er en mulighet for å gjøre kantene mere spisse. Vi sjekker vinkelen av gradienten til hver piksel, og hvis pikselen har en høyere intensitet enn naboene på alle 8 vinkler, nuller vi naboene og beholder vi intensiteten til det pikslet, og resultatet blir tynnere kanter. Dette skjer under (NMS(m1,theta)) i koden.

Threshold og hysterese

Tilslutt bruker vi threshold for å gjøre kanten enda mer tydelig. Vi katogorisere pikslene i to katogorier, svake og sterke. Vi ønsker da at alle piksler med svake verdier til å bli like sterke som de sterke pikslene, og vi ønsker også å nulle alle de pikslene som har vardier svakere enn et piksel verdi. Det gjør at kantene blir mere tydlige og lettere og se. Her må vi ha en parameter, og parameteret skal bare brukes for å beregne piksel intensitetene som vi ønsker. Det som er optimalt er at kantene av alle cellekjerne til å bli tydlige i resultat bildet, og at støyet i bildet er så lite som mulig. Jeg fant ut at det høyeste intensiteten jeg har er cirka 30, og dermed valgte en parameter som er [9,14]. Denne delen kjøres under (hysterese(tl,th,innbildet)) i koden, og dette er resultat bildet :

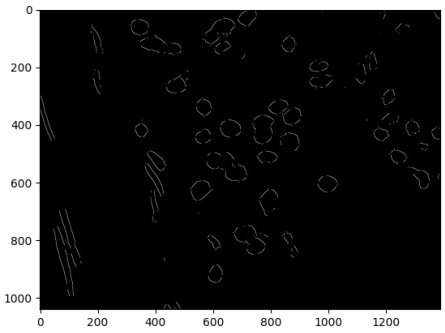


Figure 3: Canny algoritmen med (Tl = 9), (Th=14), og $\sigma = 5$

Diskusjon

Resultate kunne vært bedre, siden noen kanter av cellekjernene er ikke med i det endelige resultatet. Antagelsen min er at kanskje min Non-maximum-suppression delen av koden eliminere litt mer enn ønsket. Det er litt støy i bildet også, og det kan betyr at det kanskje trenges en annen bluring algortitme. Dette opplever vi faktisk når vi leker med sigma verdien for gaussian-blur. Økningen fjerner mer støy, men gjør kantene litt bredere. Siden mitt største intensitet er på cirka 30, Th måtte være en god del lavere for å kunne ta med de kantene som har litt svakere intensiteter, og Tl måtte være høy nok til å ikke ta med støyet og deler av bildet som ikke er så mye ønsket.