NTNU

Kunnskap for en bedre verden

Fakultet for informasjonsteknologi og elektroteknikk

IMAG2024 - Matematikk for ingeniørfag 2 D

Matteprosjekt rapport

Authors:
Benjamin Willadsen — Mustafa Kesen — Sethushan Selaiah — Sofiya Yasim
Ibrahim Ali — Roel Bakke

Vår 2024

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduksjon

Denne rapporten utforsker bruken av bildebehandlingsteknikker for å klargjøre detaljer i et objekt på et bilde. Disse teknikkene går ut på å redusere støyen i et bilde og deretter forsøke å identifisere kantene til et objekt uten at bakgrunnen forstyrrer. Ville det da være mulig å effektivt fjerne støy fra bildet og samtidig bevare objektkantene uten at bakgrunnsstøy forstyrrer?
Dette er bildet oppgavene dreier seg om, dette svart-hvit bildet av dette insektet.

Figure 1: Oppgave bildet

2 Teori knyttet til oppgavene

Gradient i bilder:
I bildebehandling brukes gradienten til å analysere endringer i farge og intensitet over et bilde. Dette er viktig for å kunne identifisere egenskaper som kanter, mønste og tekstur. Gradienter markerer hvor det skjer endringer i intensitet og er derfor brukbar for teknikker som kantdeteksjon hvor gradienten bidrar med å skille høy grad av endring i lysstyrke eller farge. Dette hjelper med å skille forgrunn fra bakgrunn, identifisere objekter ved hjelp av kanter eller for å understreke visse kanter og mønster.

Matematisk formulering av gradienten:
Når man analyserer et bilde ser vi på endringer i farge eller intensitet langs x-aksen (horisontalt) og y-aksen (vertikalt). Endringene forteller oss blant annet hvordan bildet er satt sammen og hvordan de kan anvendes i ulike prosesser som kandeteksjon.

Hvis vi tar funksjonen f(x,y) for å beskrive endring i intensitet, hvor:

$$z = f(x, y)$$

For å forstå hvordan intensiteten endres i et bilde kan man bruke de partielle derverte av f med hensyn på x og y.

f_x er den partielle deriverte av f med hensyn til x og måler endringen av intensitet langs x-aksen (horisontalt). Dette forteller oss hvordan intensiteten endres fra venstre mot høyre i bildet.

$$f_x(x,y) = \frac{\partial f}{\partial x} \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

f_y er den partielle deriverte av f med hensyn til y og måler endringen av intensitet langs y-aksen (vertikalt).

$$f_y(x,y) = \frac{\partial f}{\partial y} \approx \frac{f(x,y+1) - f(x,y-1)}{2}$$

Bruk av gradienten:
Ved å regne ut både f_x og f_y, kan vi finne gradientvektoren ∇f ved hvert enkelt punkt i bildet.

$$\nabla f = \sqrt{f_x^2 + f_y^2}$$

Vektoren indikerer retning og størrelse på den største endringen i intensitet. Dette er nyttig for å blant annet oppdage kanter siden de svarer til steder hvor lengden av gradientvektoren er stor.

Diffusion:
Diffusjons teori er ofte tilknyttet til kjemi og biologi innen atomlære, der diffusjon er bevegelse av stoffer til et sted der mengden av det stoffet er mindre, slik at det blir en spredning på hvor stoffene befinner seg. Slik som hvordan sukker sprer seg når det blir blandet med te, eller saft blir blandet med vann, men hva har dette med bildebehandling å gjøre?

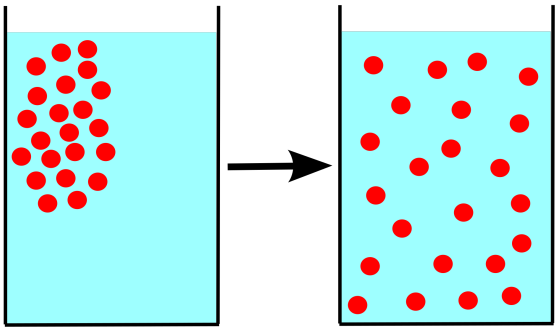

Figure 2: Diffusjon

Diffusjon innen bildebehandling:
Bilder er bare masse partikler med farger over et portrett.Så i konteksten til bildebehandling så handler diffusjon om prosessen der verdiene til en piksel endrer seg over tid basert på visse algortimer/regler, dette er for å da oppnå ulike formål som kantutgjevning og støyreduksjon (noice reduction).

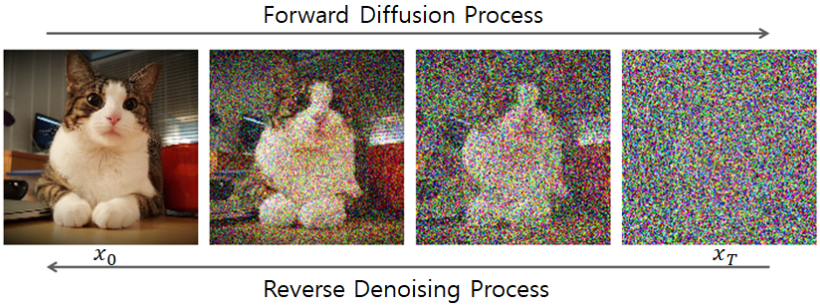

Figure 3: Diffusjon prossesen i et bildet. Her blir bilde av katta diffusert med tanke på støy reduksjon/økning

Edge Detection:
Edge detection (Kant deteksjon på norsk) er enkelt forklart hvordan en maskin kan se kanter på en figur eller et bildet. For oss er dette en vesentlig prosses og noe vi ble vant til siden fødsel. Men det samme kan ikke sies for en datamaskin basert på ren kode, derfor bruker matematiske metoder for å skille mellom bakgrunn og objektet for å da finne kantene rundt en viss figur. Disse kant detekterings metodene er oftest brukt for objekt gjennkjenning og datakompresjon innen bildebehandling og datamaskinsyn.

Hvordan benytter man Edge Detection?
Kantene i et bilde representerer overganger i intensitet eller farge, og disse overgangene kan ofte ses som endringer i gradienten til pikslene.Ved bruk av algoritmer så kan man finne ut hvor kantene til et objekt i et bildet er ved bruk av gradienten av farge pikslene. Dette kommer vi mer inn til i oppgavene som følger.

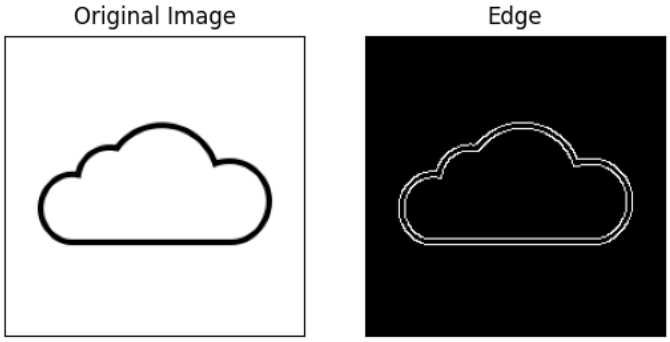

Figure 4: Edge detection skiller skyen med bakgrunnen

3 Oppgave 1 - *Gradienten og kantene i et bilde*

Oppgave a: Denne oppgaven omhandler beregning av den partiellderiverte f_x som representerer gradienten i x-retning. Oppgaven løstes ved å anvende sentraldifferanse metoden, som ga en tilnærming av endringen i intensiteteten/fargen i x-retning for hver punkt i bildet. Dette førte til

en nøyaktig beregning av f_x for alle pikslene i bildet. Se figur: 5

Oppgave b: Deretter ble den partiellderiverte f_y beregnet, som representerer gradienten i y-retning. På samme måte som f_x, ble sentraldifferansen anvendet for å estimere endringen i intensitet i y-retning for hvert punkt i bildet. Se figur: 6

Etter disse to oppgavene får man en forståelse av gradienten i både horisontal og vertikal retning i bildet.

I selve koden ble numpy sin gradient funksjon np.gradient brukt, for å beregne den partiellderivert f_x og f_y i x- og y-retning. Ved å bruke 'axis=1' spesifiseres at gradienten skal beregnes langs den horisontale aksen i bildet, og 'axis=0' spesifiserer den vertikale aksen i bildet.

image var navnet på bildet som ble brukt, så dette førte til at gradienten i både x- og y-retning kunne bli beregnet, ved å spesifisere hvilken akse på bildet det var snakk om. Resultatene ble lagret i matrisene f_x og f_y.

Noe som er verdt å merke seg er at numpy sin isnan funksjon ble brukt til å sette de udefinerte verdiene i matrisene er lik 0. Dette ble gjort for å håndtere verdier som ikke var definert i gradienten. Siden i prosessen med å beregne gradienten, spesielt nær kantene i et bilde, mangler det enten en piksel til høyre eller venstre, eller over og under som man trenger for å utfører beregningen. Da kan man unngå for eksempel støyete områder rundt kantene, eller noe annet som kan påvirke resultatet av kantdeteksjonen

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread('prosjektbilde.png')


# a)Regn ut verdiene for f_x og legg dem i en matrise med navn fx.
# Ta gradienten av bildet i x-retning
f_x = np.gradient(image, axis=1)

# Sett verdier til 0 der de ikke er definert
f_x[np.isnan(f_x)] = 0

# Tegn deretter bilde til matrisen fx.
plt.title("Oppgave 1a) Matrise f_x")
plt.imshow(f_x)
plt.show()

# b) Regn ut verdiene for f_y og legg dem i en matrise med navn fy.
# Ta gradienten av bildet i y-retning
f_y = np.gradient(image, axis=0)

# Sett verdier til 0 der de ikke er definert
f_y[np.isnan(f_y)] = 0

# Plot bildet fy
plt.title("Oppgave 1b) Matrise f_y")
plt.imshow(f_y)
plt.show()
```

Listing 1: Gradientverdi i x- og y-retning

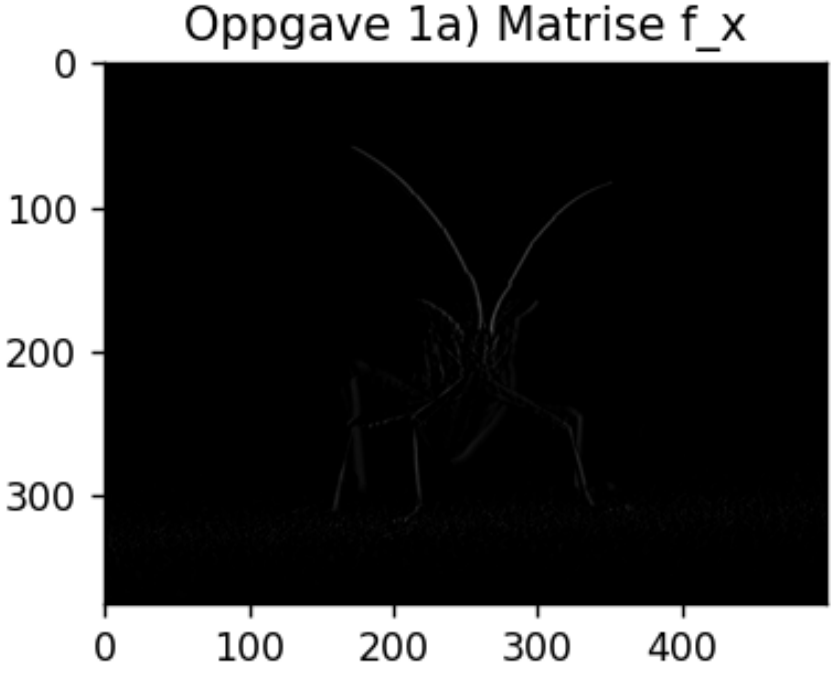

Figure 5: Beregning av f_x for pikslene i bildet

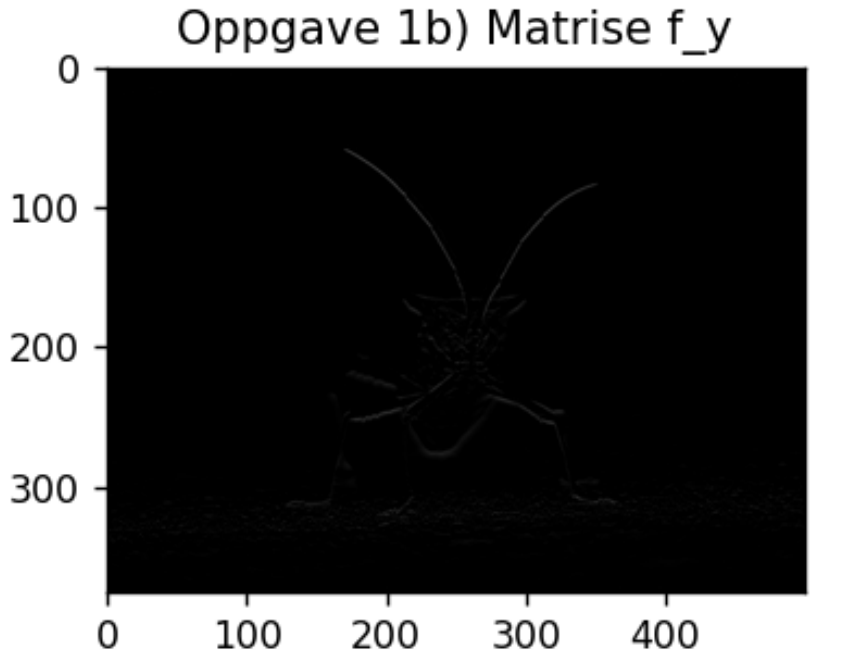

Figure 6: Beregning av f_y for pikslene i bildet

Oppgave c: I denne oppgaven ble lengden av gradienten altså Δf beregnet i hvert punkt, og lagt til i nye matrisen **fgradabs**. f_x og f_y ble kombinert ved bruk av Pytagoras, og den totale lengden av gradienten ble beregnet i hvert punkt av bildet. Dette førte til at man kunne se gradientens totale styrke i pikslene. Ved å se på resultatet kan man se den dramatiske intensiteten av gradienten, siden endringen av farge i hver piksel blir mer tydeligjort. Se figur: 7.

Oppgave d: Videre ble minste og største verdien i **fgradabs** beregnet og verdiene ble skalert innenfor et bestemt intervall, 0 og 1. Dette førte til at det ble lettere å visualisere gradienten. numpy funksjonene min og max ble brukt (np.min/ np.max) for å finne minste og største verdien i

matrisen, slik at det var mulig å skalere alle verdiene i **fgradabs** til å ligge mellom 0 og 1. Skaleringen av matrisen ble et viktig trinn for å få en forståelse av forholdet mellom intensitetsendringene, som man kan se på resultatet. Se figur: 8

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread('prosjektbilde.png')


# c) Regn ut lengden av gradienten i hvert punkt ved hjelp av pytagoras
#og legg dem i en matrise med navn fgradabs.
fgradabs = np.sqrt(f_x**2 + f_y**2)

# Sett verdier til 0 der de ikke er definert
fgradabs[np.isnan(fgradabs)] = 0

# Tegn deretter bilde til matrisen fgradabs.
plt.title("Oppgave 1c) Matrise fgradabs")
plt.imshow(fgradabs)
plt.show()

# d) # Finn minste og største verdi i matrisen fgradabs
min_value = np.min(fgradabs)
max_value = np.max(fgradabs)

# Skaler verdiene
fgradabs_scaled = (fgradabs - min_value) / (max_value - min_value)

# Tegn fgradabs bildet på nytt.
plt.title("Oppgave 1d) - fgradabs scaled")
plt.imshow(fgradabs_scaled)
plt.show()
```

Listing 2: Lengden av gradienten og skalering av verdiene

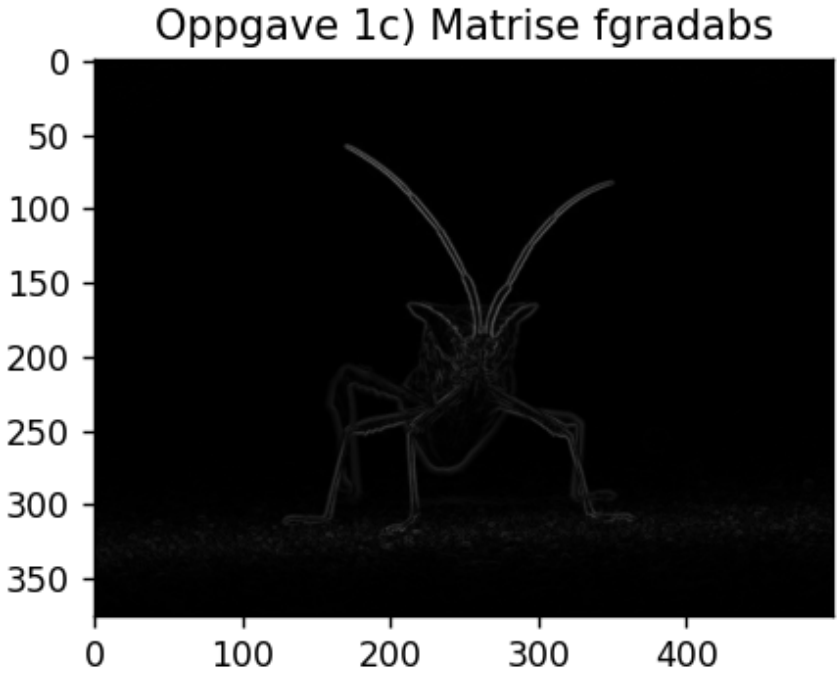

Figure 7: Lengden av gradienten i hvert punkt (Δf)

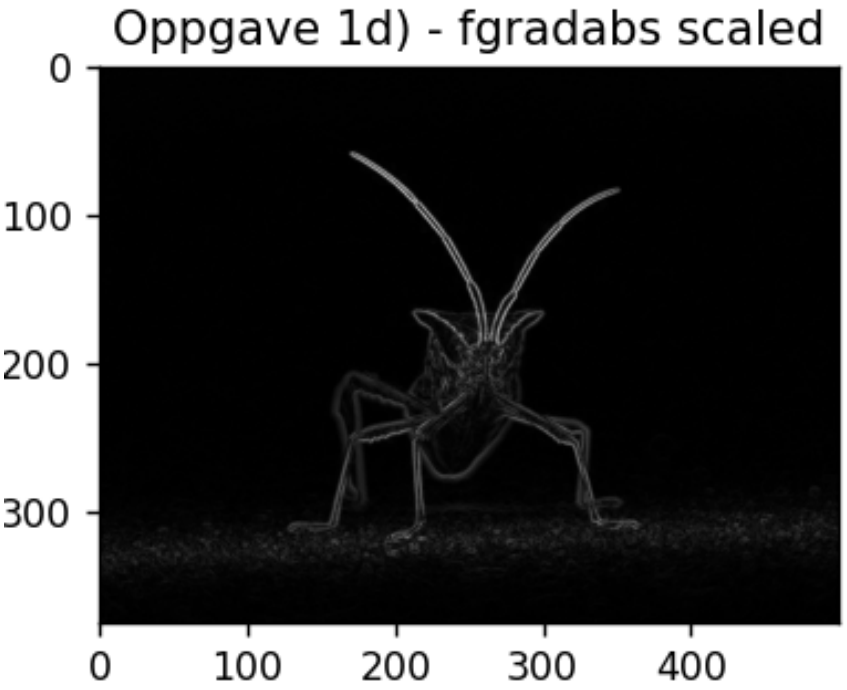

Figure 8: Skalering av pikslene i bildet

Oppgave e: Kanter markerer overganger mellom ulike objekter, som mennesket klarer å identifisere relativt kjapt på grunn av synet sitt. Når det er snakk om kanter i et bildet, er det snakk om områder der det er en plutselig endring i intensitet eller farge. Disse endringene gir informasjon om struktur og form.

I oppgavene tidligere ble gradienten beregnet, som representerer endringen i intensitet og farge.

Gradient-magnituden indikerer styrken av kanten i en piksel (Kleppe, 2021). Når man ser på gradienten i et bilde, ser man etter der gradienten er stor, noe som indikerer en brå endring i intensitet, som anses som en kant.

Ved å analysere gradienten, kan man dermed klassifisere en piksel som enten en kant, eller ikke en kant. Gradienten fungerer dermed som et godt verktøy for kantdeteksjon.

Oppgave f: I denne oppgaven ble det presentert en ferdigstilt funksjon med navn 'kutt', som tok verdien x som input og returnerte 0.0 eller 1.0, avhengig om x var mindre enn eller lik parameteren c. Videre ble funksjonen 'np.vectorize' brukt, for å gjøre 'kutt' funksjonen til en vektorisert funksjon, som betyr at alle pikslene/punktene kan brukes på alle elementene i matrisen **fgradabs_scaled** samtidig. Deretter ble denne vektoriserte funksjonen brukt til å lage en ny matrise kalt 'kanter'. Denne matrisen inneholder nullere og enere, avhengig om gradientveriden i **fgradabs_scaled** er under eller over terskelverdien 'c'.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
image = mpimg.imread('prosjektbilde.png')


# f)
def kutt(x):
    c = 0.1  #terskel
    if x <= c:
        return 0.0
    else:
        return 1.0

kutt = np.frompyfunc(kutt, 1, 1)
# Bruker np.vectorize for å bruke funksjonen kutt på alle elementene i matrisen fgradabs
kanter = np.vectorize(kutt)
# Bruker funksjonen kutt til å lage en matrise med navn kanter.
kanter = kanter(fgradabs_scaled)
plt.title("Oppgave f) - Kanter")
# Tegn kanter bildet.
plt.imshow(kanter)
```

Listing 3: Algoritme for vektorisering av kutt funksjon

Justering av terskelverdien c spilte en stor rolle i kantdeteksjonen i denne oppgaven.

Når det ble satt en **høy verdi av c**, for eksempel 0.8, hadde bildet færre markerte kanter. Dette er fordi når terskelverdien er høy, betyr det at kun gradientverdier som er betydelig større enn terskelen 0.8 vil bli ansett som kanter. Dette resulterer til at færre punkter i bildet, blir ansett som kanter, siden terskelen er så høy.

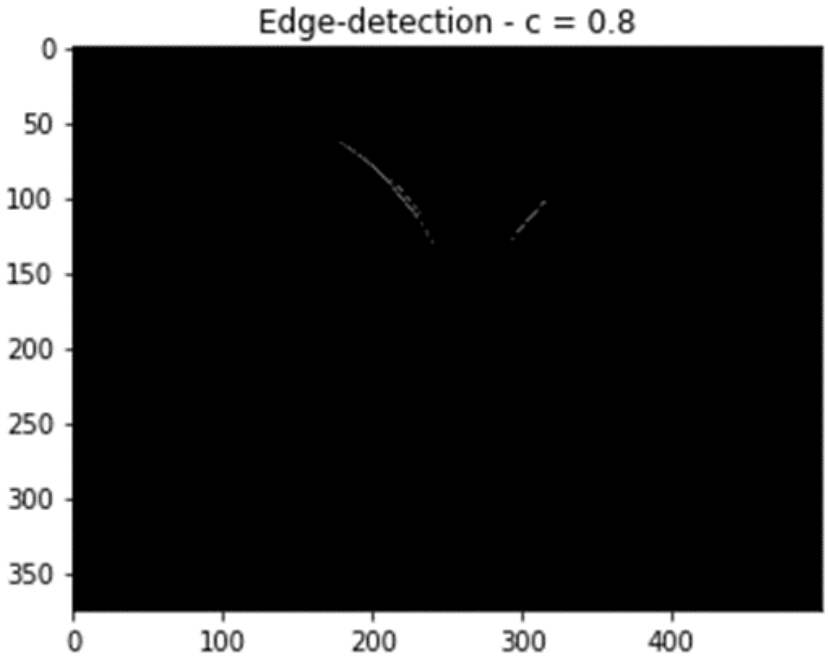

Figure 9: Høy terskelverdi = lite detaljer

Når det ble satt en **lav verdi av c**, for eksempel 0.003, endte bildet opp med altfor mye støy og mange små detaljer som også ble markert som kanter. Dette er fordi når terskelverdien er lav, betyr det at absolutt alle gradientverdier som er større enn terskelen 0.003 vil bli ansett som kanter. Dette fører til at flere punkter i bildet, vil bli ansett som kanter, siden terskelen er så lav.

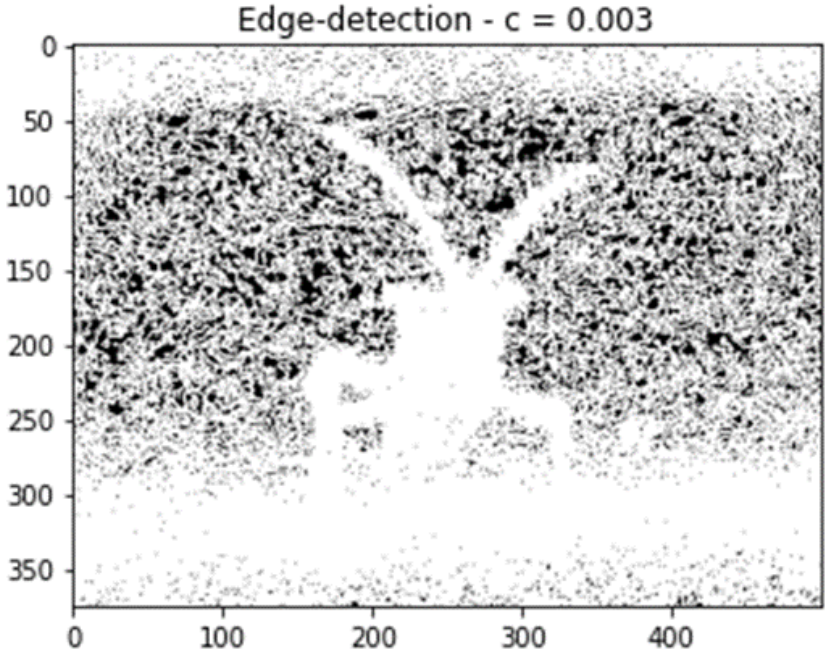

Figure 10: Lav terskelverdi = mye støy

Når det ble satt en **passende verdi av c**, for eksempel 0.1, endte bildet opp med å vise de viktigste kantene, slik at strukturen og formen til insektet ble tydelig. En optimal terskelverdi bør balanseres mellom å markere viktige kanter, samtidig som den reduserer støy og unødvendige

detaljer. Dette fikk vi til ved å prøve oss fram og forstå at den optimale terskelverdien er høy nok til å redusere støy, men også lav nok til at de viktigste punktene ble ansett som kanter.

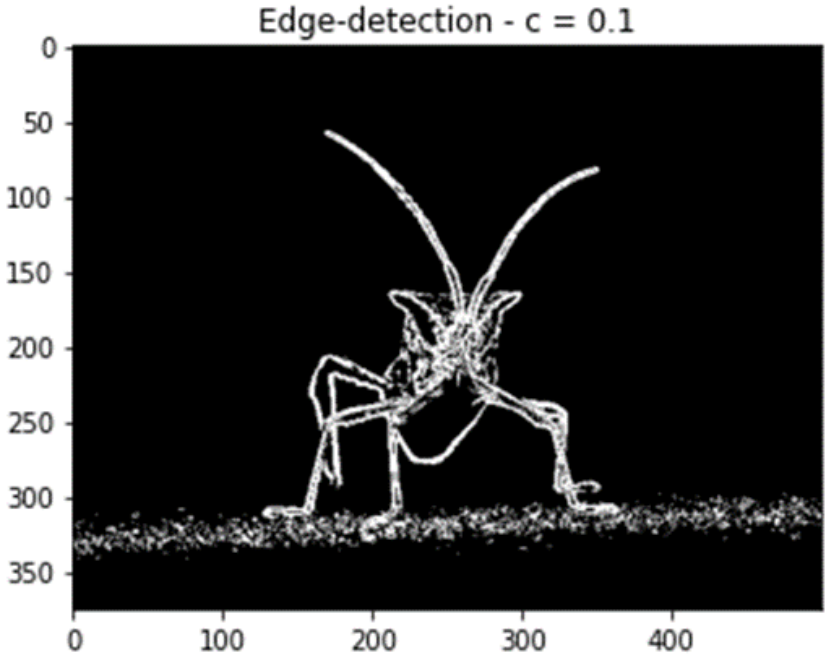

Figure 11: Passende terskelverdi

4 Oppgave 2 - *Diffusjon og endringer i tid*

Ficks lover beskriver prinsippene bak diffusjon, som er spredningen av partikler eller molekyler fra områder med høy konsentrasjon til områder med lavere konsentrasjon. Følgene oppgaver skal gå igjenom de ulike lovene til Ficks.

Oppgave a:
Ficks første diffusjonslov kan uttrykkes på flere måter, men oppgaven utaler den som:

$J = a\frac{\partial F}{\partial x}$

Her representerer J diffusjonsstrømmen (mengden materiale som diffunderer per areal og tid), a er diffusjonskoeffisienten, og $\frac{\partial F}{\partial x}$ er konsentrasjonsgradienten (endringen i konsentrasjon med avstand). Hvor F er konsentrasjonen og x er posisjonen.

Oppgave b:
Ficks andre diffusjons lov defineres oftes som:

$$\frac{\partial F}{\partial t} = a(\frac{\partial^2 F}{\partial^2 x})$$

Her er $\frac{\partial^2 F}{\partial^2 x}$ den andre ordens partiske derivasjonen av konsentrasjonen av stoffet F med hensyn til t (tid). Dette representerer endringen i konsentrasjonsgradienten langs x-aksen og indikerer hvordan konsentrasjonen endres over rommet på grunn av diffusjon.

Oppagve c:

Flukser av type 1 (første ordens) og type 2 (andre ordens) representerer henholdsvis lineær og

kvadratisk avhengighet av konsentrasjonen med hensyn til romlig variasjon.

De ulike flux typene passer avhengig av hvilken situasjon man er I. Type 1 passer mer når man gjennomgår enkel diffusjon på et stiff som har konstant konsentrasjon, mens type 2 passer mer når det er sterk interaksjon imellom partiklene som skal difusseres.

Oppgave d:
Når alle tre former for fluksen kombineres, indikerer det komplekse diffusjonsprosesser der flere faktorer spiller en rolle samtidig. Dominansen av en bestemt mekanisme, enten lineær (type 1), ikke-lineær (type 2) eller romlig gradient (type 3), avhenger av systemets egenskaper og miljøforholdene.

Oppgave e:

Vi kan oppgi bevaringsloven for et stoff (F(x,y,t)) i ved bruk av formelen.
$\frac{\partial F}{\partial t} + \frac{\partial I}{\partial x} + \frac{\partial J}{\partial y} = 0$

I denne ligningen er $\frac{\partial F}{\partial t}$ endringen i stoffmengden (F) over tid (t), og siden vi ønsker å finne fluxen i vektorform ($J(vektor) = (I, J)$) så represensenterer I og J strømmen av stoffet i x og y retningene. Derfor kan vi sette opp $\frac{\partial I}{\partial x} + \frac{\partial J}{\partial y}$ som to andre faktorer som totalt med $\frac{\partial F}{\partial t}$ må bli 0. Derfor er likningen formelen satt opp slik, for å sikre at mengden av stoffet forblir konstant over tid i det todimensjonale området, med J som fluksen av stoffet gjennom områdets grenser.

5 Oppgave 3 - *Diffusjon på bilde*

Oppgave a: Oppgaven involverte diffusjon av et bilde ved hjelp av forlengs Euler-integrasjon kombinert med Laplace-operatøren. Denne teknikken gjorde det mulig å simulere hvordan intensitetene i bildet endres over tid ved å spre pikselverdier gradvis. Gjennom denne prosessen ble det oppnådd et nytt bilde hvor hver piksel hadde oppdaterte intensitetsverdier.

Bruken av forlengs Euler-integrasjon ga innsikt i hvordan matematiske konsepter kan anvendes praktisk for å løse bildemanipuleringsoppgaver. Dette vil være nyttig for lignende problemer i fremtiden.

I implementasjonen ble det benyttet en for-løkke for å iterativt oppdatere bildets pikselverdier gjennom et bestemt antall iterasjoner. Hver piksel ble oppdatert basert på Laplace-operatøren, som beregner endringer i pikselintensiteter basert på nærliggende piksler, og resultatet ble skalert med tidssteget (dt).

```python
from matplotlib import pyplot as plt
from matplotlib import image as mpimg
import numpy as np

def kutt(x):
    c = 0.15 # Grense for skalering
    if x<=c:
        return 0.0
    else:
        return 1.0

kutt = np.frompyfunc(kutt, 1, 1)
kanter = np.vectorize(kutt)

# Initialisereing av parametre og innlasting av bilde
num_iterations = 100
dt = 0.1
image = mpimg.imread("image001.png")
image2 = np.copy(image)

# Forlengs Euler-integrasjon for diffusjon
for i in range (num_iterations):
    f_x = np.gradient(image2, axis=1)
    f_y = np.gradient(image2, axis=0)
    f_xx = np.gradient(f_x, axis=1)
    f_yy = np.gradient(f_y, axis=0)
    image2 += dt*(f_xx + f_yy) # Oppdatering av bildet basert på Laplace-operatøren


# Visualisering av det diffuserte bildet
plt.imshow(image2, cmap='gray')
plt.title("Diffusion result")
plt.show()


# Skalerer det diffuserte bildet og utfører kantdeteksjon
f_x2 = np.gradient(image2, axis=1)
f_y2 = np.gradient(image2, axis=0)
fgradabs2 = np.sqrt(f_x2**2 + f_y2**2)
fgradabs2_scaled = (fgradabs2 - fgradabs2.min())/(fgradabs2.max() - fgradabs2.min())

# Kaller på kutt-funksjonen for å utføre kantdeteksjon
kanter = kanter(fgradabs2_scaled)
plt.title("Kanter")
plt.imshow(kanter)
```

Listing 4: Koden som ble brukt for å løse Oppgave 3

For beregning av de partielle deriverte, ble Numpy sin gradientfunksjon (np.gradient) brukt, som i dette tilfelle ble brukt til å derivere langs begge aksene i bildet. 'image2' ble det diffuserte bildet, hvor image2 += dt * ($f_x x + f_y y$) representerer oppdateringen av bildet ved å legge til produktet av tidssteget og summen av de partielle andrederiverte langs x og y aksene (Laplace-operatøren).

```
# Forlengs Euler-integrasjon for diffusjon
for i in range (num_iterations):
    f_x = np.gradient(image2, axis=1)
    f_y = np.gradient(image2, axis=0)
    f_xx = np.gradient(f_x, axis=1)
    f_yy = np.gradient(f_y, axis=0)
    image2 += dt*(f_xx + f_yy) # Oppdatering av bildet basert på Laplace-operatøren
```

Listing 5: Metode for å generere det diffuserte bildet

Linjen `image2 += dt * (f_xx + f_yy)` oppdaterer bildet `image2` ved å justere hver piksels verdi basert på Laplace-operatøren, som representeres av summen `f_xx + f_yy`. Denne operatøren beregner endringene i pikselintensiteter fra omkringliggende piksler, og er nøkkelen til å forstå hvordan intensiteter sprer seg over bildet. Produktet av tidssteget `dt` og Laplace-operatøren skalerer disse endringene, kontrollerer diffusjonshastigheten og sikrer en jevn spredning. Ved hver iterasjon anvendes denne oppdateringen for å simulere spredning av intensiteter, noe som er en sentral del av Euler-metoden for numerisk løsning.

Oppgave b: I denne oppgaven ble det eksperimentert hvordan bildene ville se ut med ulike verdier for Δ T i a).

Lav Δ T :

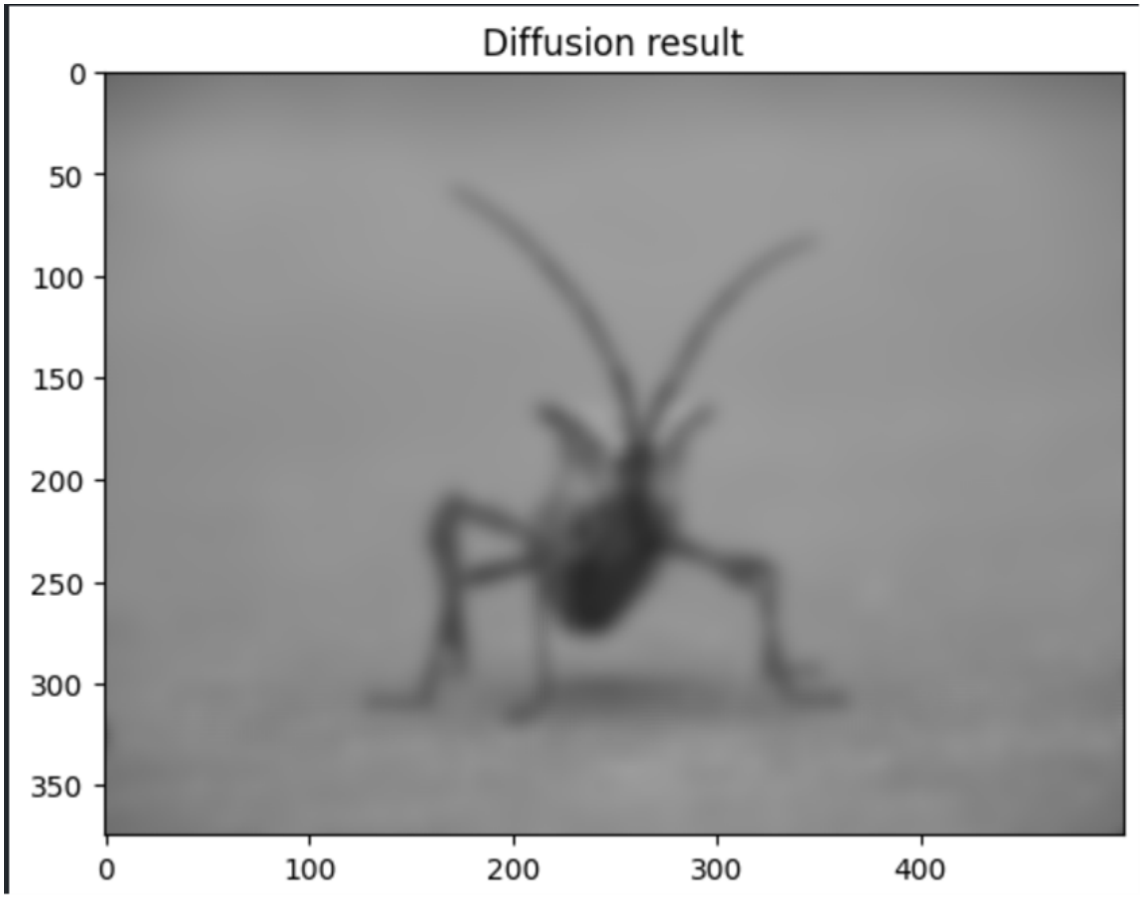

Figure 12: Δ T = 0.1

Høy Δ T :

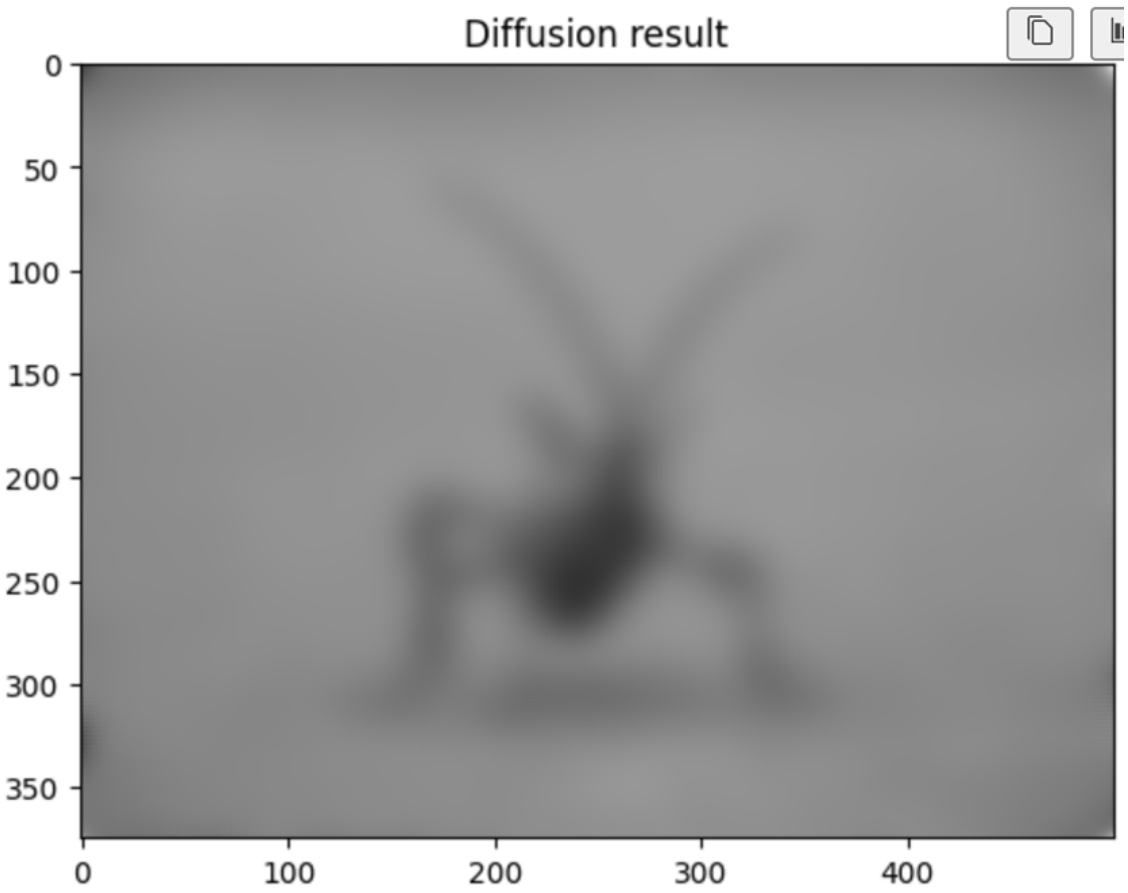

Figure 13: Δ T =1

Med en høyere Δ T ser man at bildet blir mer uklart. Dette skyldes hvordan tidsskrittet påvirker den numeriske løsningen i diffusjkonslikningen. Diffusjonslikningen er en partiell differensiallikning som beskriver i dette tilfelle bildeintensiteter. Hvert iterasjonstrinn simulerer spredning av intensiteter over en lengre tidsperiode. Dette betyr at hver oppdatering gjør en større endring i bildets intensiteter, noe som resulterer i en raskere spredning av intensitetene fra områder med høy intensitet til områder med lav intensitet. Målet med diffusjonsprosessen er å glatte ut intensitetsendringer for å fjerne støy eller andre oppgaver som kantdeteksjon. Når Δ T er for stor, kan denne glattingen bli overdreven, og detaljene og skarpheten i bildet kan reduseres kraftig.

Oppgave c: Formålet med denne oppgaven var å finne en god grenseverdi, Δ T, og antall iterasjoner slik at algoritmen fra Oppgave 1 kombinert med det diffuserte bilde ga en god beskrivelse av kantene i bildet.

Etter mye eksperimentering ble det beste resultatet oppnåd med grensen 'c=0.15' sammen med 'Δ T =0.1' og med 100 iterasjoner.

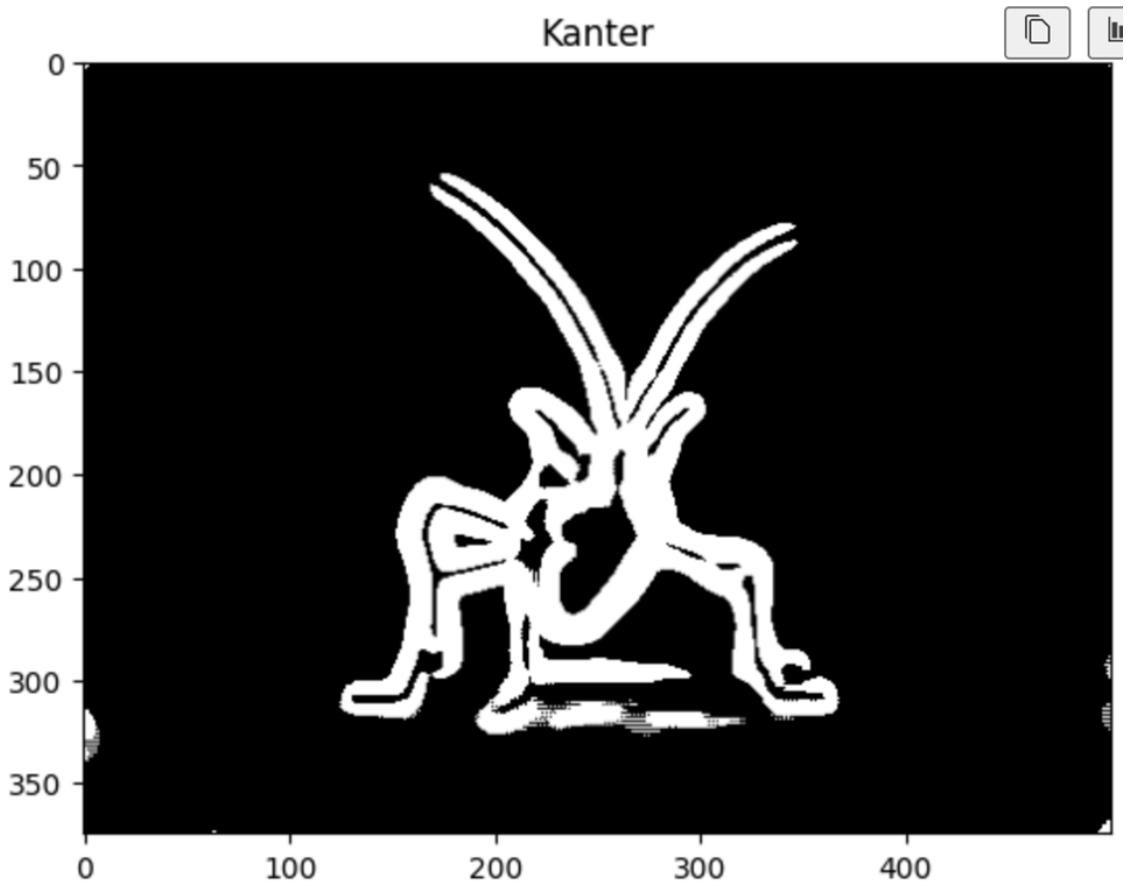

Figure 14: Parametre: c=0.15 , Δ T =0.1 og 100 iterasjoner

6 Oppgave 4 - *Mer diffusjon på et bilde*

I oppgave 3 var diffusjonen av bildet gitt ved differensiallikningen
$F_t = I_x + J_y$ hvor $I = g \cdot F_x$, og $J = g \cdot F_y$,
men istedenfor at g er konstanten 1, er den nå funksjonen $g(|\nabla F|) = \dfrac{1}{\sqrt{1 + \dfrac{|\nabla F|^2}{\lambda^2}}}$ for et vilkårlig tall λ. Ideen for denne oppgaven er å velge en passende verdi for λ slik at diffusjonen bevarer de skarpe kantene lengre i diffusjonen.

6.1 Oppgave a

Går ut på å tegne grafen til $g(x)$ for ulike verdier av x.

Koden for å plotte grafen:

```python
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

# Oppgave 4 a)
# Tegn grafen til g(x) for ulike verdier av .

lambda_values = np.linspace(0.1, 10, 5)
x = np.linspace(0, 50, 1000)

def g_x(x, lmbda):
    return 1 / (np.sqrt(1 + (x)**2 / (lmbda**2)))

for lmbda in lambda_values:
    g_values = g_x(x, lmbda)
    plt.plot(x, g_values, label=f' = {lmbda}')

plt.xlabel('x')
plt.ylabel('g(x)')
plt.title('g(x) for different values of ')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 6: Kode for tegning av grafen til $g(x)$

Her lages det to tabeller ved hjelp av np.linspace funksjonen, **lambda_values** og x. I **lambda_values** tabellen lagres det fem forskjellige lambda verdier mellom 0.1 og 10, mens i tabellen x lagres det 1000 punkter for x mellom 0 og 50. Så defineres funksjonen $g(x)$ med to variabler, hvor vi setter inn tabellene våre med lambda og x verdiene, når vi itererer gjennom de ulike lambda verdiene med for-løkken. Deretter plottes $g(x)$ for de ulike lambda verdiene i forhold til x verdiene.

Denne koden gir oss denne grafen:

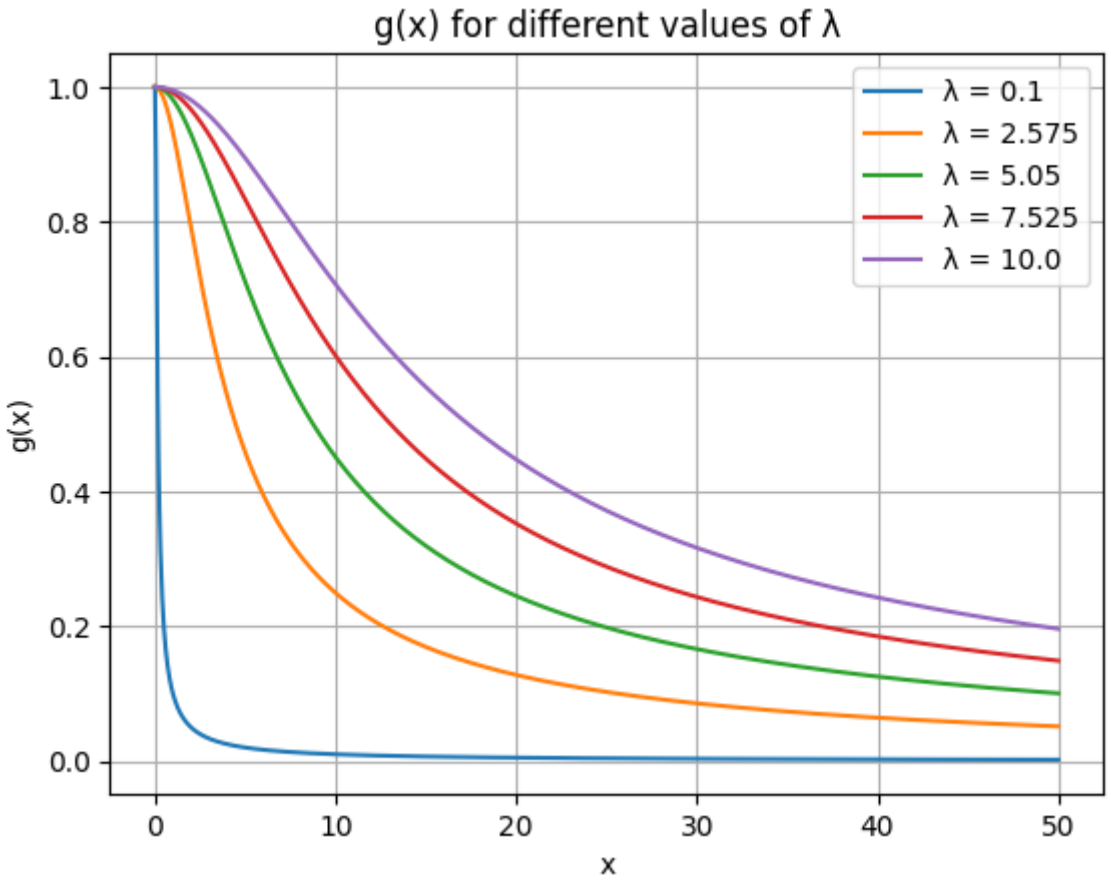

Figure 15: Grafen for g(x) for x-verdier i intervallet [0, 50]

6.1.1 i)

Som vi husker, er funksjonen vår $g(|\nabla F|) = \dfrac{1}{\sqrt{1 + \dfrac{|\nabla F|^2}{\lambda^2}}}$.

For store verdier av $|\nabla F|$, vil brøken $\left(\dfrac{|\nabla F|}{\lambda}\right)^2$ dominere nevneren.

Da vil $g(|\nabla F|)$ nærme seg $\dfrac{1}{\sqrt{\left(\dfrac{|\nabla F|}{\lambda}\right)^2}} = \dfrac{1}{\dfrac{|\nabla F|}{\lambda}} = \dfrac{\lambda}{|\nabla F|}$.

Dette betyr at når $|\nabla F|$ blir veldig stor, vil $g(|\nabla F|)$ nærme seg $\dfrac{\lambda}{|\nabla F|}$.

Dette tilsier da at $g(|\nabla F|)$ vil avta og nærme seg null når $|\nabla F|$ blir veldig stor i forhold til λ.

6.1.2 ii)

Derimot når vi får veldig små verdier av $|\nabla F|$, vil brøken $\dfrac{|\nabla F|}{\lambda}$ nærme seg 0, ettersom nevneren blir større enn telleren. Når $\dfrac{|\nabla F|}{\lambda}$ nærmer seg 0, vil også $\left(\dfrac{|\nabla F|}{\lambda}\right)^2$ nærme seg 0, så i nevneren av

funksjonen $g(|\nabla F|)$ vil vi få $\sqrt{1+\left(\frac{|\nabla F|}{\lambda}\right)^2} \approx \sqrt{1+0} = 1$.

Dette betyr at når vi får veldig små verdier av $|\nabla F|$, vil $g(|\nabla F|)$ nærme seg $\frac{1}{\sqrt{1+0}} = 1$.

6.1.3 iii)

Hvilken innflytelse har λ for hva som skjer i i) og ii)?

I funksjonen vår, må vi også velge en passende λ verdi, ettersom den også har noe å si for funksjonsverdien til $g(|\nabla F|)$.

For store verdier av $|\nabla F|$, har det seg slik at jo større λ er, jo langsommere vil $g(|\nabla F|)$ nærme seg null.

For små verdier av $|\nabla F|$ derimot, blir forholdet $\frac{|\nabla F|}{\lambda}$ neglisjerbart lite. Dermed vil nevneren $\sqrt{1+\left(\frac{|\nabla F|}{\lambda}\right)^2}$ nærme seg 1, og $g(|\nabla F|)$ vil nærme seg 1.

6.2 Oppgave b og c

Implementer integrasjonen beskrevet over og vis bildene som lages

For å utføre integrasjonen og for å plotte bildet etter diffusjonen, tas i bruk gradientene fra tidligere.

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread("Matteoppgave.png")
image3 = np.copy(image)

def g_x(x, lmbda):
    return 1 / (np.sqrt(1 + (x)**2 / (lmbda**2)))

lmbda = 0.01
dT = 0.1
c = 0.1
num_iterations = 100

for i in range(num_iterations):
    f_x = np.gradient(image3, axis=1)
    f_y = np.gradient(image3, axis=0)
    f_xx = np.gradient(f_x, axis=1) * g_x(f_x, lmbda)
    f_yy = np.gradient(f_y, axis=0) * g_x(f_y, lmbda)
    image3 += dT * (f_xx + f_yy)

plt.title(f"Diffusjon med  = {lmbda}" + f" og t = {dT}")
plt.imshow(image3)
plt.show()
```

Listing 7: Beregning og visualisering av diffusjonen av bildet

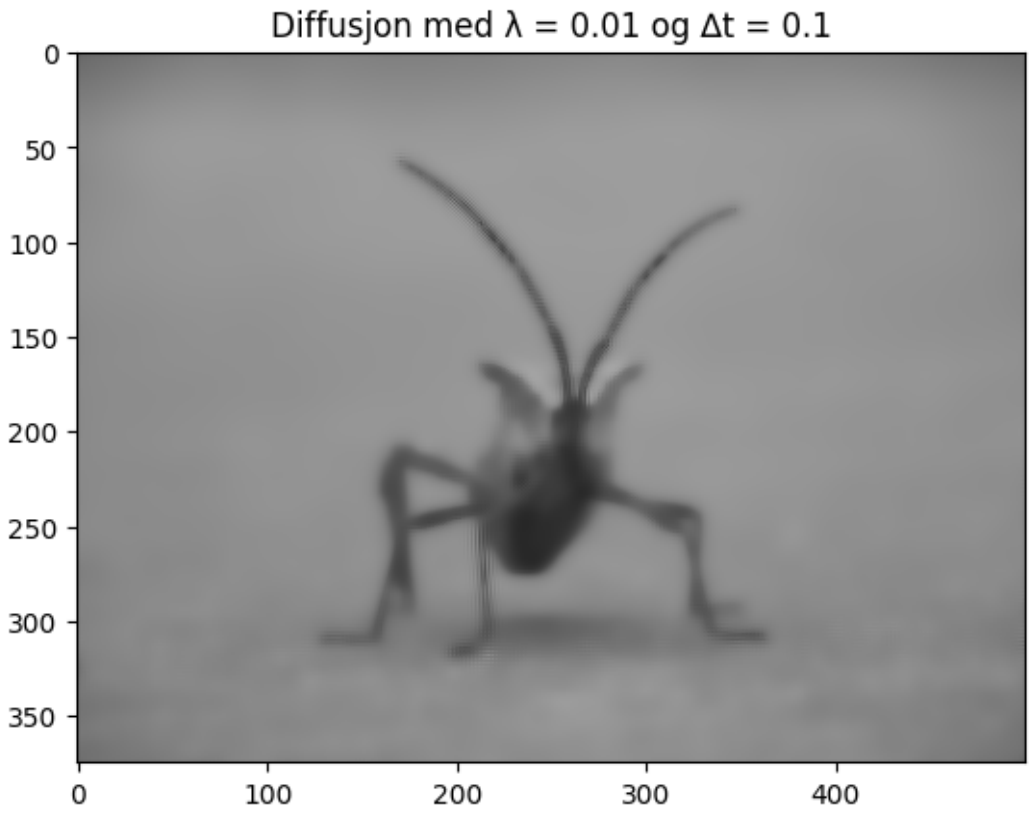

Figure 16: Diffusjon med $\lambda = 0.01$ og $\Delta t = 0.1$

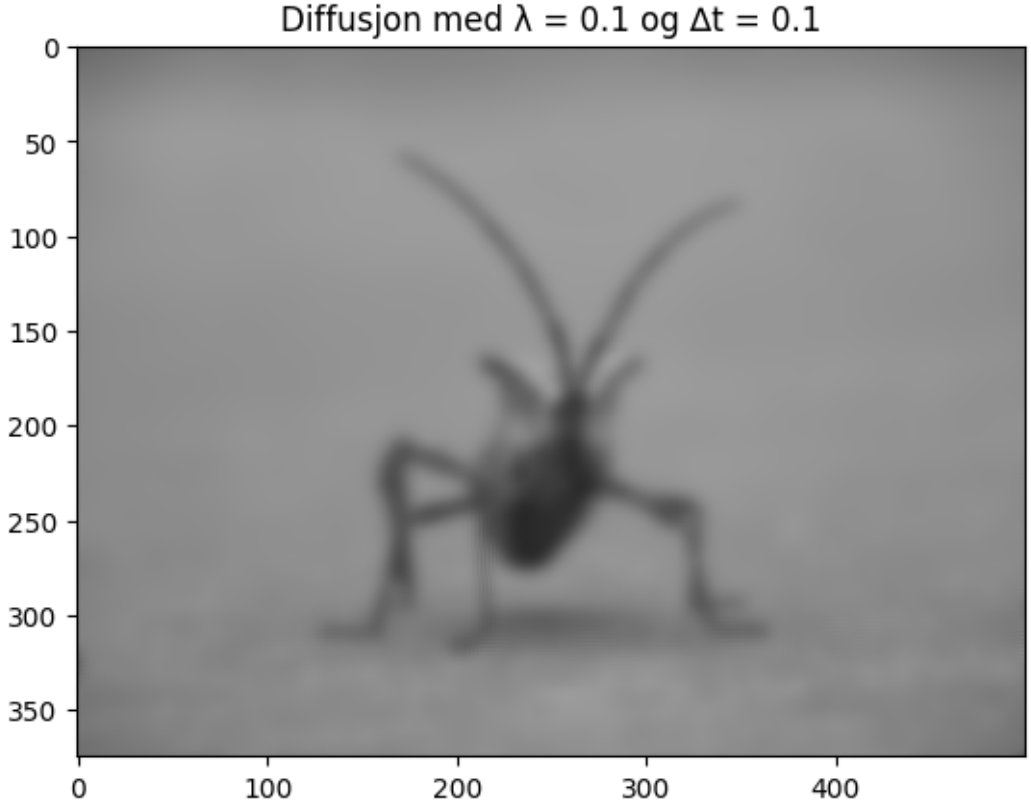

Figure 17: Diffusjon med $\lambda = 0.1$ og $\Delta t = 0.1$

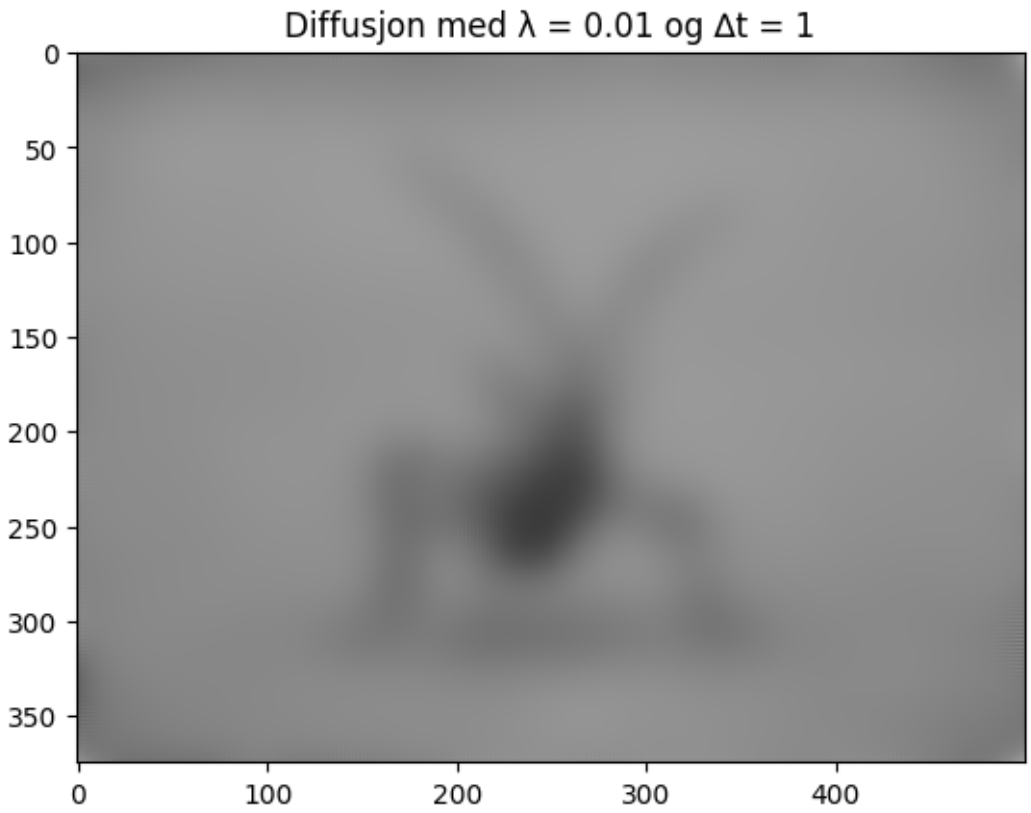

Figure 18: Diffusjon med $\lambda = 0.01$ og $\Delta t = 1$

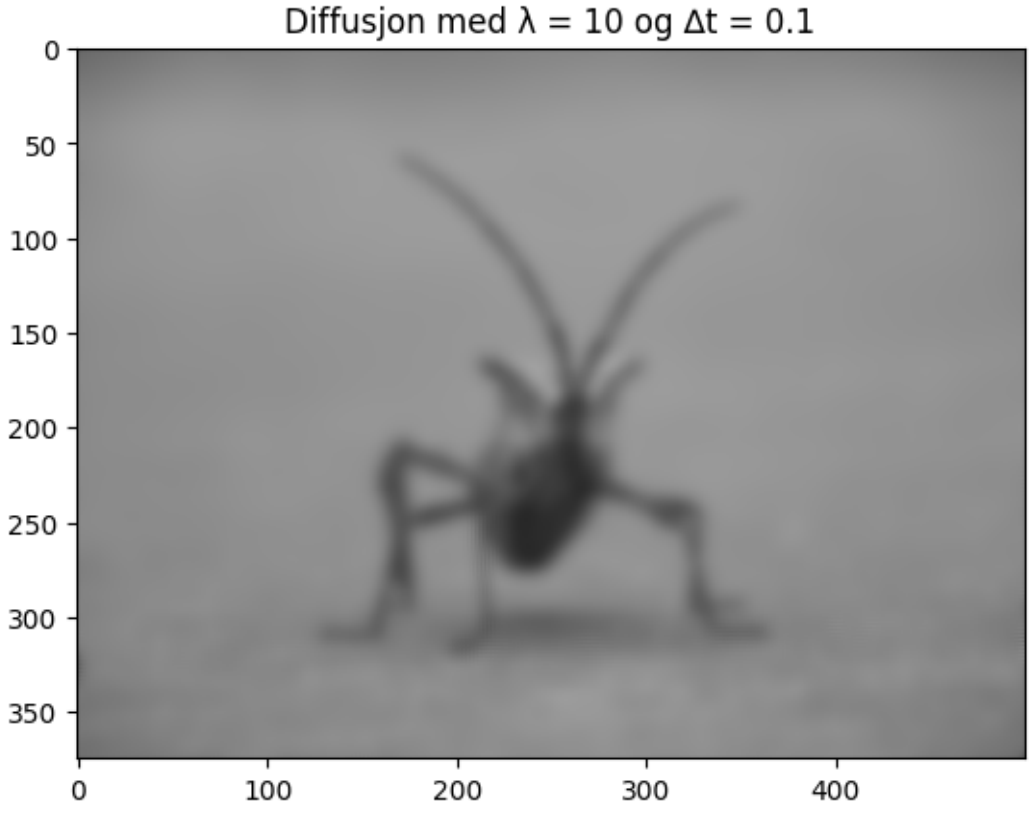

Figure 19: Diffusjon med $\lambda = 10$ og $\Delta t = 0.1$

Figurene 16-19 viser bildet etter diffusjonen med ulike λ og Δt verdier.
Med en $\lambda = 0.01$ og $\Delta t = 0.1$ blir bildet uklart og støyen reduseres, men detaljene beholdes til en viss grad, derimot når λ blir endret til 0.1 som i figur 17, vil støyen reduseres enda mer men på bekostning av detaljene. Hvis vi øker Δt til en verdi av 1, vil vi se at bildet blir veldig uskarpt og detaljene er nesten helt borte. Dette er fordi når Δt blir for stor, vil hver iterasjon i diffusjonsprosessen gjøre større endringer og være litt for aggressiv, som fører til en uskarphet i bildet. Derfor gir en Δt-verdi på 0.1 ganske gode resultater. Til slutt viser figur 19 at med en λ-verdi på 10, vil man få et ganske likt bilde som i figur 17. Dette skyldes at terskelen for et godt resultat trolig ligger mellom 0.01 og 0.1, derfor vil bildene med $\lambda = 10$ og 0.1 ende opp ganske like, mens bildet med $\lambda = 0.01$ ender opp bedre i vårt tilfelle.

Med en stor λ-verdi, diffusjonsprosessen være mer glattende. Dette betyr at detaljer i bildet vil bli jevnet ut, og resultatet blir da et bilde som er mer uskarpt og med mindre støy.
Med en liten λ-verdi, vil diffusjonsprosessen være mindre glattende. Da vil man bevare flere detaljer i bildet som betyr flere kanter, men kan også bety mere støy på bildet. Derfor er det viktig å velge en passende λ-verdi så resultatet blir slik man ønsker det.

Eksperimenter med ulike verdier av Δ t og lambda i b. Beskriv hva du observerer. Hvilken innflytelse har ulike verdier av lambda på resultatene?

6.3 Oppgave d

Etter å ha utført diffusjonsprosessen, kan kand deteksjonsalgoritmen brukes til å vise kantene i bildet:

```
def kutt(x):
    if x <= c:
        return 0.0
    else:
        return 1.0

f_x3 = np.gradient(image3, axis=1)
f_y3 = np.gradient(image3, axis=0)
fgradabs3 = (f_x3**2 + f_y3**2)**0.5
fgradabs_scaled3 = (fgradabs3 - fgradabs3.min())/(fgradabs3.max()-fgradabs3.min())
kanter3 = np.vectorize(kutt)
kanter3 = kanter3(fgradabs_scaled3)
plt.title("Kantene i bildet etter diffusjon | " + f" = {lmbda}" + f", t = {dT}" + f" og c = {c}")
plt.imshow(kanter3)
plt.show()
```

Listing 8: Kant deteksjon algoritme og visualisering

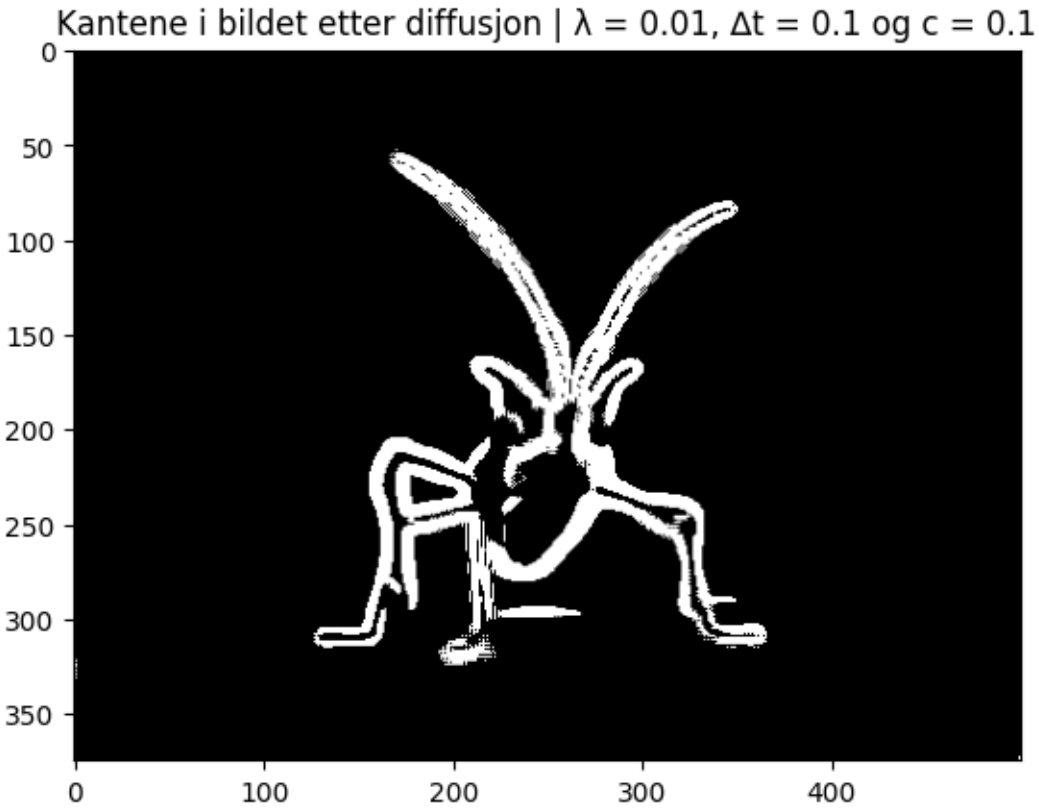

Figure 20: Kanter i bildet med $\lambda = 0.01$, $\Delta t = 0.1$ og $c = 0.1$

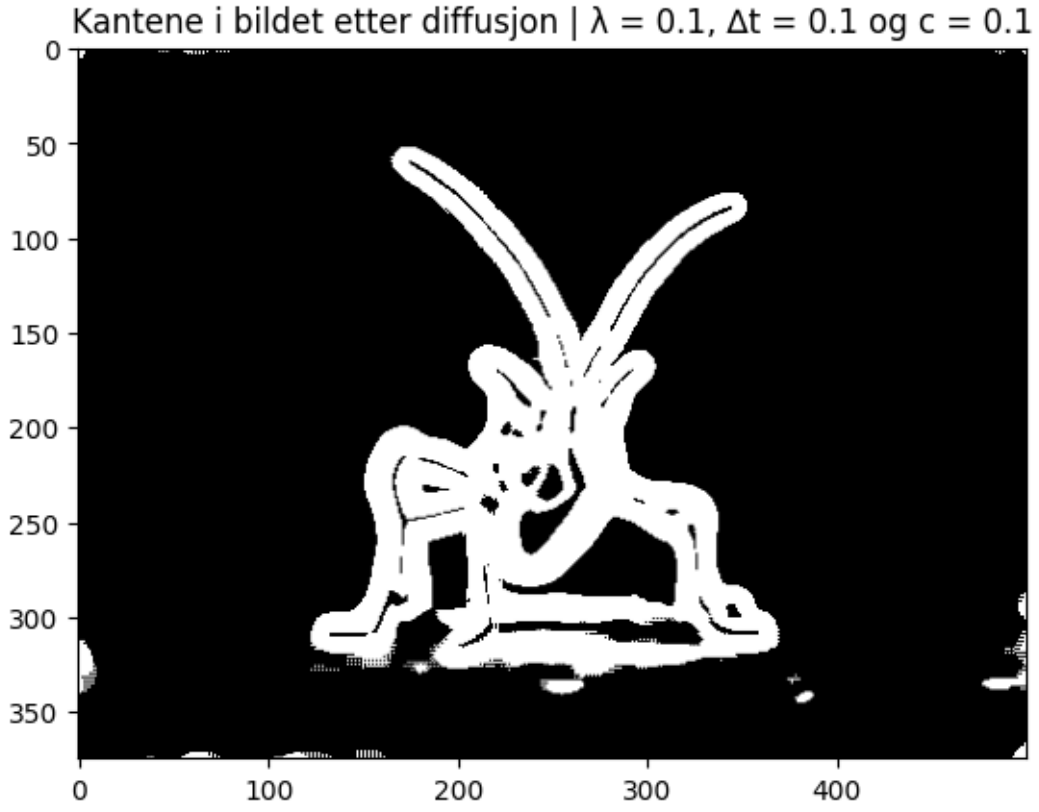

Figure 21: Kanter i bildet med $\lambda = 0.1$, $\Delta t = 0.1$ og $c = 0.1$

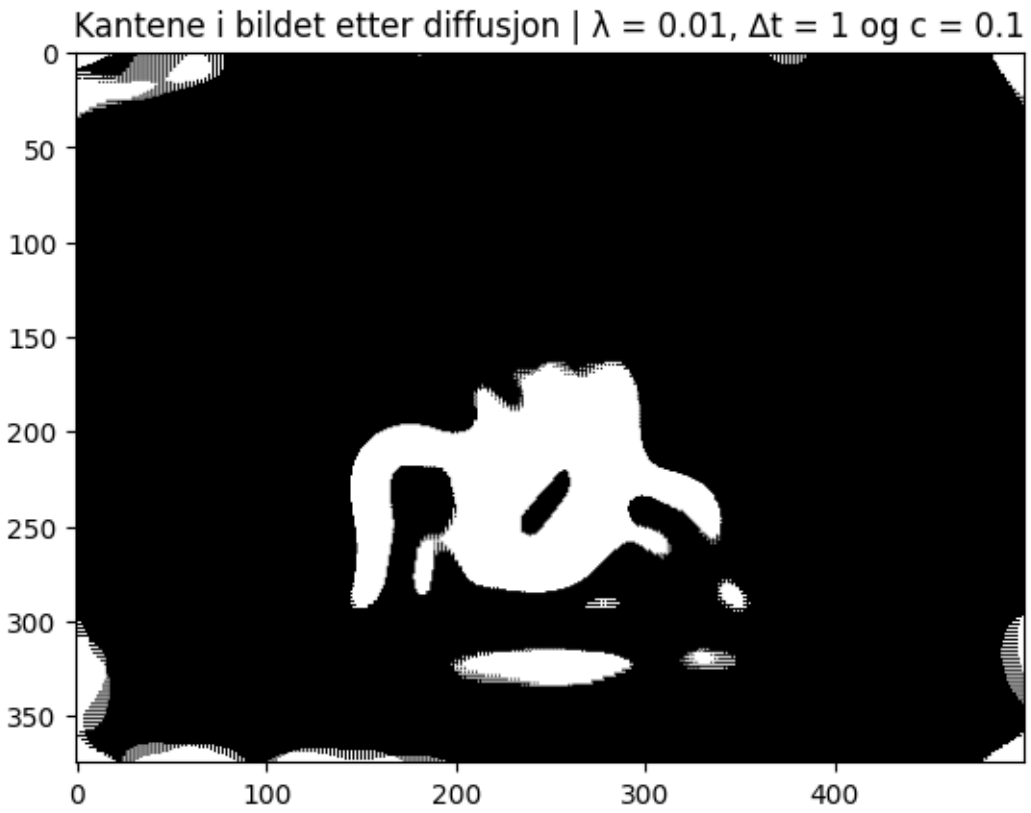

Figure 22: Kanter i bildet med $\lambda = 0.01$, $\Delta t = 1$ og $c = 0.1$

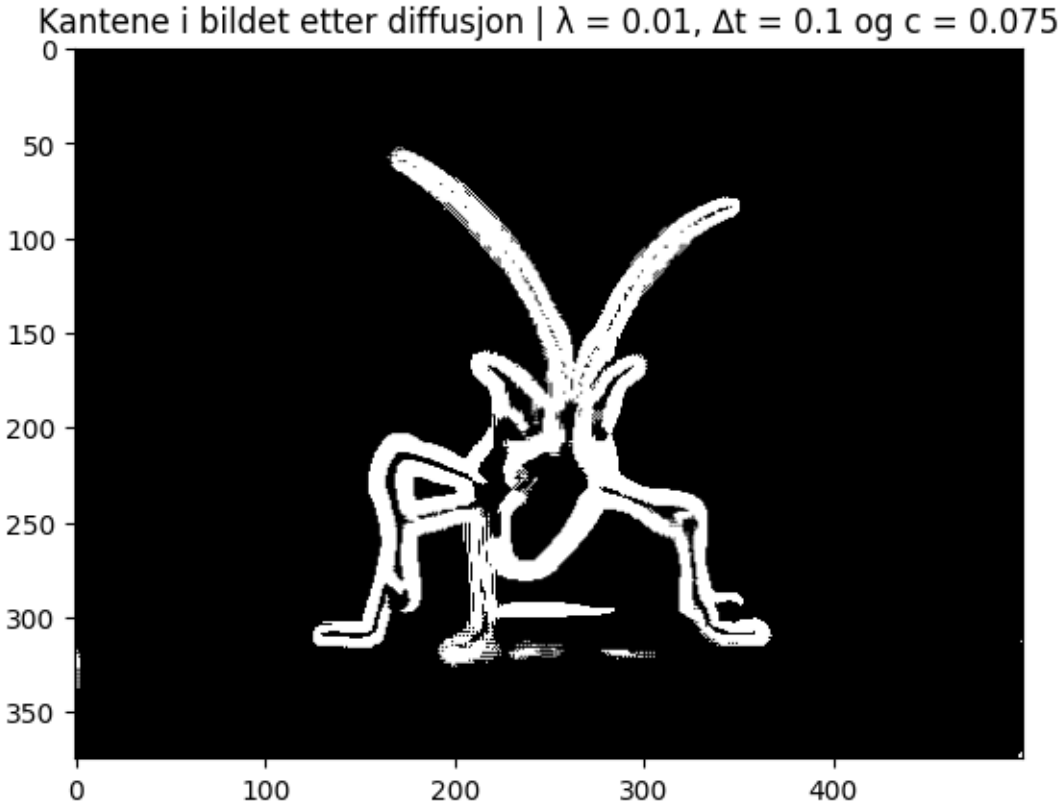

Figure 23: Kanter i bildet med $\lambda = 0.01$, $\Delta t = 0.1$ og $c = 0.075$

Figurene 20-23 viser kantene i bildet etter diffusjonsprosessen med utvalgte λ og Δt verdier. Kantene som vises varierer veldig av betingelsene man velger. De to beste resultatene, figur 20 og 23 viser tydelig kantene til insektet som er i fokus med litt mangler her og der, spesielt for figur 20. Men ettersom c er satt til 0.075 på figur 23, vil man få med litt støy som man kan se under insektet. En god verdi for c vil da trolig være innenfor $c = [0.075, 0.1]$.
Figur 21 viser kantene med λ-verdi på 0.1. Som nevnt tidligere, er ikke dette den beste verdien for λ, ettersom man mister detaljene og bildet blir mer uskarpt når kantene skal bli oppdaget.
Og til slutt har vi figur 22, som viser bildet med en Δt-verdi på 1. Da vil bildet bli alt for uskarpt og detaljene nesten helt borte for at kantene skal bli oppdaget.

7 Avsluttende Refleksjoner

Denne delen av rapporten tar for seg de oppnådde resultatene og deres relasjon til forskningsspørsmålet som ble introdusert i begynnelsen: "Er det mulig å effektivt fjerne støy fra bildet og samtidig bevare objektkantene uten at bakgrunnsstøy forstyrrer?" Effektiviteten og påliteligheten til de anvendte metodene vil bli vurdert, eventuelle begrensninger diskutert, og retninger for videre forskning foreslått.

Drøfting av Metoder

Kantdeteksjonsteknikker basert på gradienten til pikslene har vist seg å være effektive for å identifisere objektkanter. Valget av terskelverdier for gradientintensitet er imidlertid kritisk og kan avgjøre nøyaktigheten av deteksjonen. Finjustering av disse parameterne er nødvendig for optimale resultater, og endringer i terskelverdier kan dramatisk påvirke kvaliteten på kantdeteksjonen.

Støyreduksjon ble oppnådd gjennom diffusjonsmetoder, som effektivt reduserte bakgrunnsstøy. Det er likevel rom for forbedringer i hvordan disse metodene håndterer bildebehandling, ettersom aggressiv støyreduksjon noen ganger kan viske ut viktige detaljer.

Begrensninger

Metodene som ble implementert adresserte delvis forskningsspørsmålet. Selv om støy ble redusert uten å ofre mye av kantenes skarphet, var det situasjoner hvor kantdeteksjonen mislyktes under visse støyforhold. Dette antyder at ytterligere arbeid er nødvendig for å forbedre robustheten av algoritmene under varierte forhold.

Forbedringer og Videreutvikling

Under prosjektet ble det klart at algoritmen kunne forbedres ved å finjustere parameterne mer nøye. Videre utvikling kunne fokusere på å utvikle en tilnærming hvor terskelverdier og andre parametere justeres dynamisk basert på bildets støyprofil.

I tillegg åpnet prosjektet opp for potensiell anvendelse av kantdeteksjonsalgoritmen på uskarpe bilder for å avdekke abstrakte detaljer. Dette er et spennende felt som kan utforskes videre, gitt tilstrekkelige ressurser og tid. En slik utvidelse ville ikke bare forbedre forståelsen av bildebehandlingsteknikker, men også utvide anvendelsesområdene for arbeidet som er utført.

Etter litt utforskning også på nettet ble metoden Laplacian of Gaussian oppdaget, en mer robust versjon av Laplace som inkluderer Gauss-glatting(Kleppe, 2021). Denne metoden kan være nyttig for å forbedre kantdeteksjon i fremtidige prosjekter. Selv om dette ble oppdaget etter at de opprinnelige oppgavene var fullført, indikerer det et potensial for å integrere avanserte metoder for å løse mer komplekse bildebehandlings-utfordringer i fremtiden.

Med disse resultatene og forslagene håpes det å bidra og inspirere videre forskning på bildebehandling.

8 Konklusjon

Resultatene viser at det er fult mulig å redusere støy i bilder samtidig som objektkantene bevares. Så lenge diffusjonsparameterne lambda og tidssteget er justert nøye, vil en lavere lambda-verdi skape bedre kantdetaljer og mer støy, mens en høyere lambda-verdi mye mindre støy men tap av detaljer. Det er derfor viktig å justere diffusjonsparameterne og tilhørende verdier til bilde nøyaktig for å få ønsket utfall. Funnene indikerer at selv om metodene viser seg å være effektive, viser andre teknikker som Laplacian of Gaussian at det må ytterlig forskning til for å forbedre metodenes robusthet og nøyaktighet i bildebehandling.

References

[1] Kleppe, Andreas. (2021, 03. mars). *Digital bildebehandling.* Universitetet i Oslo. Hentet 17.04.2024 kl 14:00 fra: https://www.uio.no/studier/emner/matnat/ifi/IN2070/v21/undervisningsmateriale/forelesning/in2070_2021_08_filtrering_ii.pdf

[2] Carillo, Alejandro. (2020, 01. desember). *How computer vision works.* Khan Academy. Hentet 17.04.2024 kl 15:37 fra: fra:https://www.khanacademy.org/computing/code-org/x06130d92:how-ai-works/x06130d92:what-is-ai/v/how-computer-vision-works

Bilde referanser

- Figure 2: Diffusjon, SNL https://snl.no/diffusjon
- Figure 3: Diffusjon prossesen i et bildet. Her blir bilde av katta diffusert med tanke på støyreduksjon/økning, https://xoft.tistory.com/32
- Figure 4: Edge detection skiller skyen med bakgrunnen, henta fra https://stackoverflow.com/questions/67157150/python-opencv-delete-outermost-edges-after-canny-edge-detection