Edge detection

Evan Cummings
CSCI 548 – Douglas W. Raiford – Pattern Recognition

November 16, 2016

1 Sobel edge-detection “violet” image :

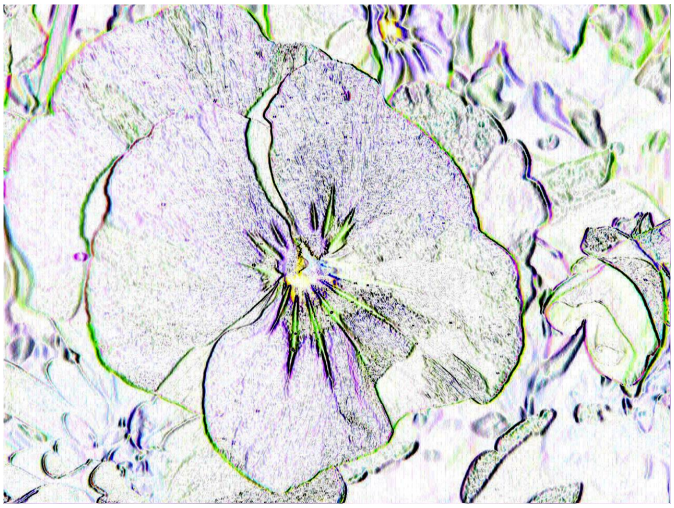

Figure 1: The originial “violet” image (left) and the edges detected by the *Sobel* method (right).

1.1 R source code :

```
# violet image edge-detection project
# Evan Cummings
# CSCI 548 - Pattern Recognition
# Douglas Raiford, Fall 2015

source("functs.r")
library(jpeg)

# store the image :
d = readJPEG("../../data/violet.jpg")

# get the edges detected by the 'Sobel' method :
s = sobelColor(d)

# save the resulting image :
writeJPEG(s, '../doc/images/violet_sobel.jpeg')
```

2 Laplacian edge-detection “Lenna” image :

Figure 2: The original “Lenna” image (above left), grey-scaled Lenna (above middle), blurred Lenna (above right), Laplacian of blurred Lenna with values less than 0.4 set to zero (bottom left), Laplacian of non-blurred-grey-scaled Lenna (bottom middle), and Laplacian for blurred-grey-scaled Lenna (bottom right).

2.1 R source code :

```
# lenna image edge-detection project
# Evan Cummings
# CSCI 548 - Pattern Recognition
# Douglas Raiford, Fall 2015

source("functs.r")
library(jpeg)

# store the image :
d = readJPEG("../../data/Lenna.jpg")

# convert to grey scale :
g = convert_to_grey(d)

# save the resulting image :
writeJPEG(g, '../doc/images/lenna_grey.jpeg')

# get the gaussian kernel :
G = gaussian_kernel(1.4, 5)

# blur the image with kernel :
B = blur_image(g,G)

# save the resulting image :
writeJPEG(B, '../doc/images/lenna_blurred.jpeg')

# get the edges by the Laplace method :
E = laplacian(g,5)

# save the resulting image Laplacian :
writeJPEG(E, '../doc/images/lenna_laplace.jpeg')

# get the edges by the Laplace method :
Eb = laplacian(B,5)

# save the resulting blurred Laplacian :
writeJPEG(Eb, '../doc/images/lenna_blurred_laplace.jpeg')

# get the edges by the Laplace method :
Et = laplacian(B,5,tol=0.4)

# save the resulting blurred Laplacian :
writeJPEG(Et, '../doc/images/lenna_blurred_tol_laplace.jpeg')
```

2.2 Common functions r source code :

```
sobelGrey = function(img)
{
  GX       = c(-1,-2,-1,0,0,0,1,2,1)
  dim(GX) = c(3,3)
  GY       = c(1,0,-1,2,0,-2,1,0,-1)
  dim(GY) = c(3,3)
  returnImg = img
  imgDims = dim(img)

  numRows = imgDims[1]
  numCols = imgDims[2]
  numMatrices = imgDims[3]

  # sumX = 0
  # sumY = 0
  # SUM = 0
  for(rowID in 1:(numRows - 1))
  {
    for(colID in 1:(numCols - 1))
    {
      sumX = 0
      sumY = 0
      SUM = 0
      if(rowID==1 || rowID==numRows-1)
      {
        SUM = 0;
      }else if(colID==1 || colID==numCols-1)
      {
        SUM = 0;
      }
      else
      {
        for(I in -1:1)
        {
          for(J in -1:1)
          {
            sumX = sumX + img[rowID + I, colID + J] * GX[I+2,J+2];
          }
        }
        for(I in -1:1)
        {
          for(J in -1:1)
          {
            sumY = sumY + img[rowID + I, colID + J] * GX[I+2,J+2];
          }
        }
        # GRADIENT MAGNITUDE APPROXIMATION (Myler p.218)
        SUM = abs(sumX) + abs(sumY);
        if(SUM > 1)
        {
          # cat("setting to 1, was ",SUM,"\n")
          SUM = 1;
        }
      }
      returnImg[rowID,colID] = SUM
    }
  }
  return(returnImg)
}


sobelColor = function(img)
{
  GX       = c(-1,-2,-1,0,0,0,1,2,1)
  dim(GX) = c(3,3)
  GY       = c(1,0,-1,2,0,-2,1,0,-1)
  dim(GY) = c(3,3)
  returnImg = img
  imgDims = dim(img)

  numRows = imgDims[1]
  numCols = imgDims[2]
  numMatrices = imgDims[3]

  # sumX = 0
  # sumY = 0
  # SUM = 0

  for(matrixID in 1:numMatrices)
  {
    for(rowID in 1:(numRows - 1))
    {
      for(colID in 1:(numCols - 1))
      {
        sumX = 0;
        sumY = 0;
        SUM = 0
        if(rowID==1 || rowID==numRows-1)
        {
          SUM = 0;
        }
        else if(colID==1 || colID==numCols-1)
        {
          SUM = 0;
        }
        else
        {
          for(I in -1:1)
          {
            for(J in -1:1)
            {
              sumX = sumX + img[rowID + I, colID + J,matrixID] * GX[I+2,J+2];
            }
          }
          for(I in -1:1)
          {
            for(J in -1:1)
            {
              sumY = sumY + img[rowID + I, colID + J,matrixID] * GX[I+2,J+2];
            }
          }
          # GRADIENT MAGNITUDE APPROXIMATION (Myler p.218)
          SUM = abs(sumX) + abs(sumY);
          # cat("sum is ", SUM,"\n")
          if(SUM > 1)
          {
            # cat("setting to 1, was ",SUM,"\n")
            SUM = 1;
          }
        }
        returnImg[rowID,colID,matrixID] = SUM
      }
    }
  }
  for(i in 1:numMatrices)
  {
    returnImg[,,i] = 1 - returnImg[,,i]
  }
  return(returnImg)
}


convert_to_grey = function(img)
{
  numMatrices = dim(img)[3]
  aveImg = img[,,1]
  if(numMatrices > 1)
  {
    for(i in 2:numMatrices)
    {
      aveImg = aveImg + img[,,i]
    }
  }
  aveImg = aveImg/max(aveImg)
  return(aveImg)
}

invertGrey = function(img)
{
  returnImg = 1 - img
  return(returnImg)
}

laplacian = function(img, p, tol=0)
{
  # create 5 x 5 discrete Laplacian operator :
  q = median(1:p)
  M = matrix(-1, p, p)
  M[q,q] = p*p - 1

  m = dim(img)[1]
  n = dim(img)[2]

  # edge image to return :
  E = matrix(0, m, n)

  for (i in q:(m-q))
  {
    for (j in q:(n-q))
    {
      t = M * img[(i-q+1):(i+q-1), (j-q+1):(j+q-1)]
      v = sum(t)
      if (v < tol)
        v = 0
      if (v > 1)
        v = 1
      E[i,j] = v
    }

  }
  return(1 - E)
}


gauss_weights = function(r,sigma)
{
  return(exp( -(r**2) / (2 * sigma**2) ))
}

gaussian_kernel = function(sigma,n)
{
  # form the distance matrix from the center pixel :
  K = matrix(0,n,n)
  p = median(1:n)
  for (i in 1:n)
  {
    for (j in 1:n)
    {
      K[i,j] = sqrt((i - p)**2 + (j - p)**2)
    }
  }

  # form the gaussian kernel :
  G = round(15*gauss_weights(K,sigma))
  G = G / sum(G)

  return(G)
}

blur_image = function(img, M)
{
  q = median(1:dim(M)[1])
  m = dim(img)[1]
  n = dim(img)[2]

  # blurred image to return :
  B = matrix(0,m,n)

  for (i in q:(m-q))
  {
    for (j in q:(n-q))
    {
      t = M * img[(i-q+1):(i+q-1), (j-q+1):(j+q-1)]
      v = sum(t)
      B[i,j] = v
    }

  }
  return(B)
}
```