```
import numpy as np
import cv2

#def im2double(im):
#    min_val = np.min(im.ravel())
#    max_val = np.max(im.ravel())
#    out = (im.astype('float') - min_val) / (max_val - min_val)
#    return out

#def FgAssign(dtrimap, FgMask):

#       for x in range(dtrimap.shape[1]):
#               for y in range(dtrimap.shape[0]):
#                       if dtrimap[y][x]>0.99:
#                               FgMask[y][x]=True
#
#       return FgMask


#def BgAssign(dtrimap, BgMask):

#       for x in range(dtrimap.shape[1]):
#               for y in range(dtrimap.shape[0]):
#                       if dtrimap[y][x]<0.01:
#                               BgMask[y][x]=True
#       return BgMask

def findBoundaryPixels(trimap, a, b, result):

#       dtrimap= im2double(trimap)
#       FgMask=trimap
#       BgMask=trimap
#       FgMask= FgAssign(dtrimap, FgMask)
#       BgMask= BgAssign(dtrimap, BgMask)
#       UnMask= FgMask
#       for x in range(trimap.shape[1]):
#               for y in range(trimap.shape[0]):
#                       UnMask[y][x]= ~FgMask[y][x] and ~BgMask[y][x]

        print "inside find boundary pixels"

        for x in range(1, (trimap.shape[1]-1)):
                for y in range(1, (trimap.shape[0]-1)):
                        if trimap[y][x]==a:
                                if (trimap[y-1][x] ==b or trimap[y+1][x] ==b or trimap[y][x-1] == b or
trimap[y][x+1]) ==b:
                                        info=(x,y)
                                        result.append(info)
        print "Arsehole"
        print result
        return result

#This is equation 2 of the paper

def calculateAlpha( F, B, I):

        result = 0.0
        div = 1e-6
        for c in range(3):
                f= F[c]
                b= B[c]
                i= I[c]

                result  = result + (i-b) * (f-b)
                div  = div + (f-b) * (f-b)
        return min(max(result/div, 0.0), 1.0);

#This is equation 3 of the paper

def colorCost(F, B, I, alpha):
        result=0.0
        for c in range(3):
```

```
                f= F[c]
                b= B[c]
                i= I[c]

                result  = result + np.square((i - (alpha * f + (1 - alpha) * b)))

        return np.sqrt(result)

#This is equation four of the paper
def distCost(p0, p1, minDist):
        dist= np.square(p0[0]-p1[0]) + np.square(p0[1] - p1[1])

        return np.sqrt(float(dist)/minDist)

#def colorDist(I0, I1) :
#       result = 0.0
#       for c in range(3) :
#               result  = result + np.square(int(I0[c]) - int(I1[c])
#       return np.sqrt(float(result))

# for sorting the bondary pixels according to intensity

#class IntensityComp(object):
#       def __init__(self, x, img):
#               self.x=x
#               self.img=img
#
#       def returnintensity(self):
#               imgr=self.img
#               return -(imgr[x[1]][x[0]][0] + imgr[x[1]][x[0]][0] + imgr[x[1]][x[0]][2])

#def IntensityComp(x):
#       return -(img[x[1]][x[0]][0] + img[x[1]][x[0]][0] + img[x[1]][x[0]][2])

#def IntensityComp(img):
#       return sum(img[item1[1]][item1[0]])

def nearestDistance(boundary, p ):
        minDist2= 999999999

        for i in range(len(boundary)):
                dist2= np.square(boundary[i][0]- p[0]) + np.square(boundary[i][1]- p[1])
                minDist2 = min(minDist2, dist2)
        return np.sqrt(float(minDist2))


def calcAlphaPatchm(img, trimap, fgbound, bgbound, samples):

        w= img.shape[1]
        h= img.shape[0]
        fg=np.uint32(np.int32(len(fgbound)))
        bg=np.uint32(np.int32(len(bgbound)))

        for y in range(h):
                next=[]
                for x in range(w):
                        info={}
                        info['fi']=None
                        info['bj']=None
                        info['df']=None
                        info['db']=None
                        info['cost']=None
                        info['alpha']=None
                        next.append(info)
                samples.append(next)


        for y in range(h):
                for x in range(w):
                        if trimap[y][x]==128:
                                p = (x, y)
```

```
                                samples[y][x]['fi']= np.uint32(fg*np.random.random())
                                samples[y][x]['bj']= np.uint32(bg*np.random.random())
                                samples[y][x]['df'] = nearestDistance(fgbound, p)
                                samples[y][x]['db'] = nearestDistance(bgbound, p)
                                samples[y][x]['cost'] = float("inf")


        coords = []
        for j in range(w*h):
                coords.append((0,0))


        for y in range(h):
                for x in range(w):
                        coords[x + y*w] = (x, y)

        for iter in range(10):
                np.random.shuffle(coords)

                for i in range(len(coords)):
                        x= coords[i][0]
                        y= coords[i][1]

                        if (trimap[y][x] != 128):
                                continue

                        I = img[y][x]

                        s= samples[y][x]

                        for y2 in range(y-1, y+2):
                                for x2 in range(x-1, x+2):

                                        if x2<0 or x2>= w or y2<0 or y2>=h :
                                                continue

                                        if trimap[y2][x2] != 128:
                                                continue

                                        s2= samples[y2][x2]
                                        xi= s2['fi']
                                        yi= s2['bj']
                                        try:
                                                fp = fgbound[xi]
                                                bp = bgbound[yi]
                                        except IndexError:
                                                break
                                        F= img[fp[1]][fp[0]]
                                        B= img[bp[1]][bp[0]]

                                        alpha = calculateAlpha(F, B, I )

                                        cost= colorCost(F, B, I , alpha) + distCost(p, fp , s['df']) +
distCost(p, bp, s['db'])


                                        if cost< s['cost'] :
                                                s['fi'] = s2['fi']
                                                s['bj'] = s2['bj']
                                                s['cost'] = cost
                                                s['alpha'] = alpha

                #random walk

                w2= int(max(len(fgbound), len(bgbound)))

                for y in range(h):
                        for x in range(w):
                                if trimap[y][x]!= 128:
                                        continue

                                p = (x, y)
```

```
                                I= img[y][x]

                                s= samples[y][x]
                                k=0

                                while(1) :
                                        r=w2*pow(   , k)
                                        k=k+1
                                        if r<1:
                                                break

                                        di = int(r * np.random.uniform(-1, 1))
                                        dj= int(r* np.random.random(-1, 1))

                                        fi = s['fi'] + di
                                        bj = s['bj'] + dj
                                        if fi<0 or fi >= len(fgbound) or bj<0 or bj>= len(bgbound):
                                                continue
                                        try:

                                                fp = fgbound[fi]
                                                bp = bgbound[bj]

                                                F= img[fp[0]][fp[1]]
                                                B= img[bp[0]][bp[1]]

                                                alpha = calculateAlpha(F, B, I)
                                                cost= colorCost(F, B, I, alpha) + distCost(p, fp, s
['df']) + distCost(p, bp, s['db'])


                                                if (cost< s['cost']):
                                                        s['fi'] = fi
                                                        s['bj'] = bj
                                                        s['cost']= cost
                                                        s['alpha'] = alpha
                                        except IndexError:
                                                break



def globalmattinghelper(img, trimap, foreground, alpha, conf):

        print "Printing trimap"
        print trimap
        result=[]
        fgbound= findBoundaryPixels(trimap, 255, 128, result)
        print "fgbound"
        result=[]
        print fgbound
        bgbound= findBoundaryPixels(trimap, 0, 128, result)

        n= len(fgbound) + len(bgbound)

        for i in range(n) :
                x= np.random.randint(trimap.shape[1])
                y= np.random.randint(trimap.shape[0])


                if trimap[y][x] == 0:
                        bgbound.append((x, y))
                elif trimap[y][x] == 255:
                        fgbound.append((x,y))
        def IntensityComp(x):
                return -(int(img[x[1]][x[0]][0]) + int(img[x[1]][x[0]][1]) + int(img[x[1]][x[0]][2]))

        sorted(fgbound, key=IntensityComp)
        sorted(bgbound, key=IntensityComp)

        samples= []
        #... some coding stil left to do
        calcAlphaPatchm(img, trimap, fgbound, bgbound, samples)
```

```
        for y in range(alpha.shape[0]):
                for x in range(alpha.shape[1]):

                        if trimap[y][x]==0:
                                alpha[y][x]=0
                                conf[y][x]=255
                                foreground[y][x]=0

                        elif trimap[y][x]==128:
                                try:
                                        alpha[y][x] = 255* np.int32(samples[y][x]['alpha'])
                                except TypeError:
                                        break

                                an= -samples[y][x]['cost']
                                print an
                                pn = (an)/6
                                print conf[y][x]
                                conf[y][x]= 255 * np.exp(pn)
                                p = fgbound[samples[y][x]['fi']]
                                foreground[y][x]= img[y][x]

                        elif trimap[y][x]== 255:
                                alpha[y][x]= 255
                                conf[y][x]=255
                                fgbound[y][x]= img[y][x]



def globalmatting(img, trimap, foreground, alpha, conf=[]):


        conf=np.zeros(trimap.shape)

        globalmattinghelper(img, trimap, foreground, alpha, conf)
```