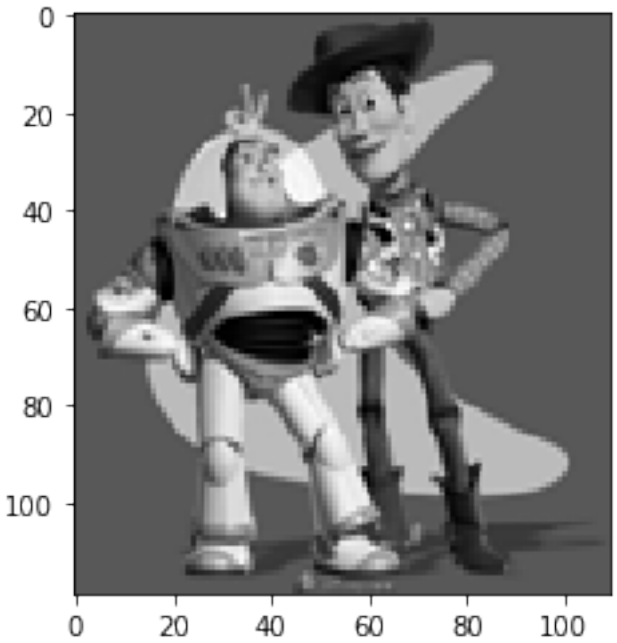
Project3

March 20, 2020

[1]:
```
import cv2
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from utils import *
import os
from scipy.sparse import lil_matrix
from scipy.sparse import linalg
from scipy.sparse import csr_matrix
import pdb
import time
```

[2]:
```
img = cv2.imread('samples/toy_problem.png')
toy_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
toy_img = cv2.cvtColor(toy_img, cv2.COLOR_BGR2GRAY).astype('double') / 255.0
plt.imshow(toy_img, cmap="gray")
```

[2]:
```
<matplotlib.image.AxesImage at 0x7fc0588fcd68>
```



0.1 Part 1 Toy Problem (20 pts)

[3]:
```
def toy_reconstruct(toy_img):
    """
    The implementation for gradient domain processing is not complicated, but␣
 ↪it is easy to make a mistake, so let's start with a toy example. Reconstruct␣
 ↪this image from its gradient values, plus one pixel intensity. Denote the␣
 ↪intensity of the source image at (x, y) as s(x,y) and the value to solve for␣
 ↪as v(x,y). For each pixel, then, we have two objectives:
    1. minimize (v(x+1,y)-v(x,y) - (s(x+1,y)-s(x,y)))^2
    2. minimize (v(x,y+1)-v(x,y) - (s(x,y+1)-s(x,y)))^2
    Note that these could be solved while adding any constant value to v, so we␣
 ↪will add one more objective:
    3. minimize (v(1,1)-s(1,1))^2

    :param toy_img: numpy.ndarray
    """
    start = time.time()
    im_h, im_w = toy_img.shape
    im2var = np.arange(im_h * im_w).reshape(im_h, im_w) # value is transposed??

#     print("im_h, im_w ", im_h, im_w)
#     print("im2var shape ", im2var.shape)
#     print("toy_img shape ", toy_img.shape)
    # Objective 1
    #  e = e + 1
    # A[e][im2var[y][x+1] = 1
    # A[e][im2var[y][x]] = -1
    # b[e] = im[y][x+1] - im[y][x]

    # A matrix bounds im_h * im_w
    # https://piazza.com/class/k5cumohrew35en?cid=362 ??
    n_constraints = 2 * im_h * im_w + 1 #number of constraints
    n_pixels = im_h * im_w
    A = lil_matrix((n_constraints, n_pixels), dtype = np.float64)
#     print("A ", A.shape)
    b = np.zeros(n_constraints, np.float64)
#     print("b ", b.shape)

    e = 0
    A[e, im2var[0][0]] = 1
    b[e] = toy_img[0][0]
#     pdb.set_trace()
    for y in range(im_h): #im2var.shape[0])
```

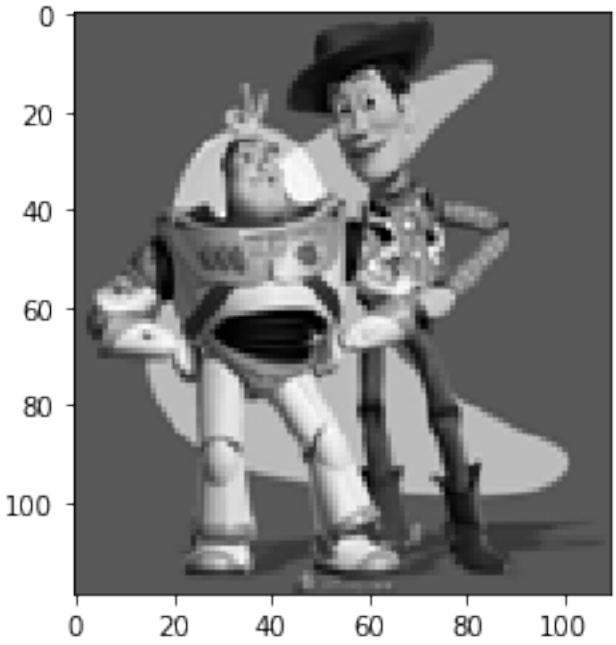
```
        for x in range(im_w): #im2var.shape[1])

            # Need to handle border cases. x == im_w-1 and y == im_h-1
            #Objective 1
            e = e + 1
            if(x != im_w-1):
                A[e, im2var[y][x+1]] = 1
                A[e, im2var[y][x]] = -1
                b[e] = toy_img[y][x+1] - toy_img[y][x]
            else:
                A[e, im2var[y][x]] = -1
                b[e] = -toy_img[y][x]

            #Objective 2
            e = e + 1
            if(y != im_h-1):
                A[e, im2var[y+1][x]] = 1
                A[e, im2var[y][x]] = -1
                b[e] = toy_img[y+1][x] - toy_img[y][x]
            else:
                A[e, im2var[y][x]] = -1
                b[e] = -toy_img[y][x]

    v = linalg.lsqr(csr_matrix(A), b)
    res = v[0].reshape(im_h, im_w)
    print("Total time taken ", time.time() - start)
    return res
```

[4]:
```
im_out = toy_reconstruct(toy_img)
# print("toy_img ", toy_img)
# print("im_out ", im_out)
if im_out.any():
    print("Error is: ", np.sqrt(((im_out - toy_img)**2).sum()))
```

```
Total time taken  1.4244844913482666
Error is:  0.0001901762385198699
```

[5]:
```
plt.imshow(im_out, cmap='gray')
```

[5]:
```
<matplotlib.image.AxesImage at 0x7fc058098390>
```



0.2 Preparation

[68]:
```
# Feel free to change image
# background_img = cv2.cvtColor(cv2.imread('samples/im2.jpg'), cv2.
 ↪COLOR_BGR2RGB).astype('double') / 255.0
# background_img = cv2.cvtColor(cv2.imread('samples/tajmahal.jpg'), cv2.
 ↪COLOR_BGR2RGB).astype('double') / 255.0
# background_img = cv2.cvtColor(cv2.imread('samples/vtz.jpg'), cv2.
 ↪COLOR_BGR2RGB).astype('double') / 255.0
background_img = cv2.cvtColor(cv2.imread('samples/wall.jpg'), cv2.
 ↪COLOR_BGR2RGB).astype('double') / 255.0
plt.figure()
plt.imshow(background_img)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[68]:
```
<matplotlib.image.AxesImage at 0x7fc0581bd9b0>
```

[70]:
```
# Feel free to change image
# object_img = cv2.cvtColor(cv2.imread('samples/penguin-chick.jpeg'), cv2.
 ↪COLOR_BGR2RGB).astype('double') / 255.0
```

```
object_img = cv2.cvtColor(cv2.imread('samples/dog2.jpg'), cv2.COLOR_BGR2RGB).
 ↪astype('double') / 255.0
import matplotlib.pyplot as plt
%matplotlib notebook
mask_coords = specify_mask(object_img)
```

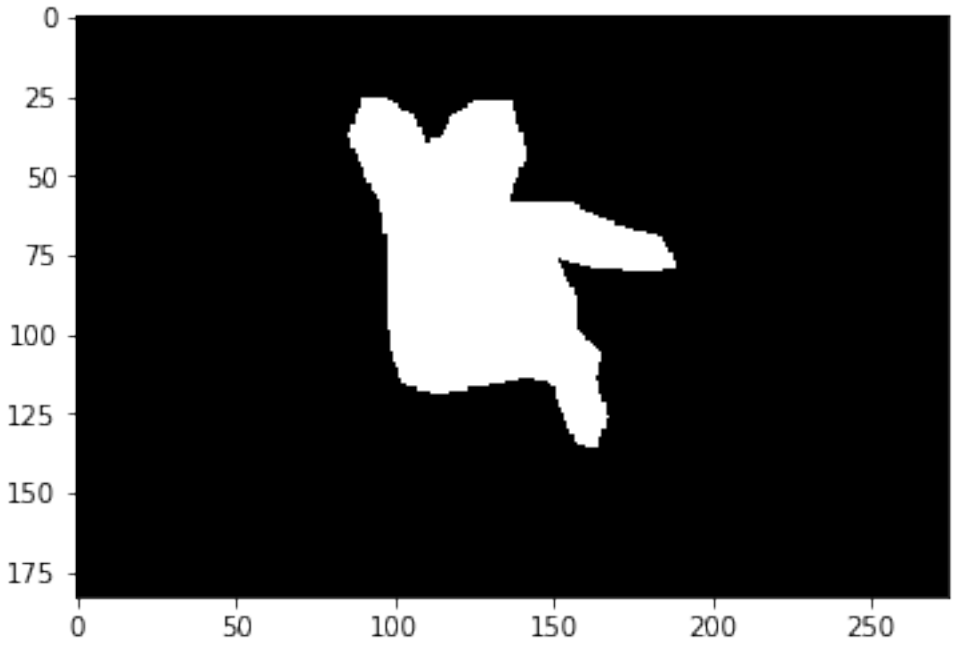
```
If it doesn't get you to the drawing mode, then rerun this function again.
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[71]:
```
xs = mask_coords[0]
ys = mask_coords[1]
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
mask = get_mask(ys, xs, object_img)
```

```
<Figure size 432x288 with 0 Axes>
```

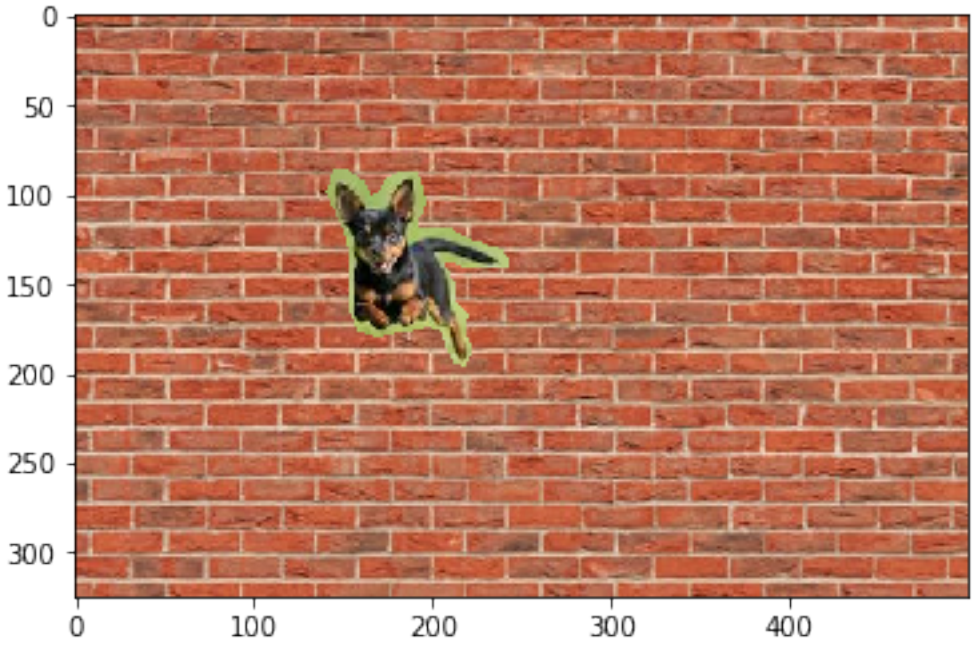


[72]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
```

```
bottom_center = specify_bottom_center(background_img)
```

```
If it doesn't get you to the drawing mode, then rerun this function again. Also,
make sure the object fill fit into the background image. Otherwise it will crash

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

[73]:
```
%matplotlib inline
import matplotlib.pyplot as plt
cropped_object, object_mask = align_source(object_img, mask, background_img,␣
 ↪bottom_center)
```



0.3 Part 2 Poisson Blending (50 pts)

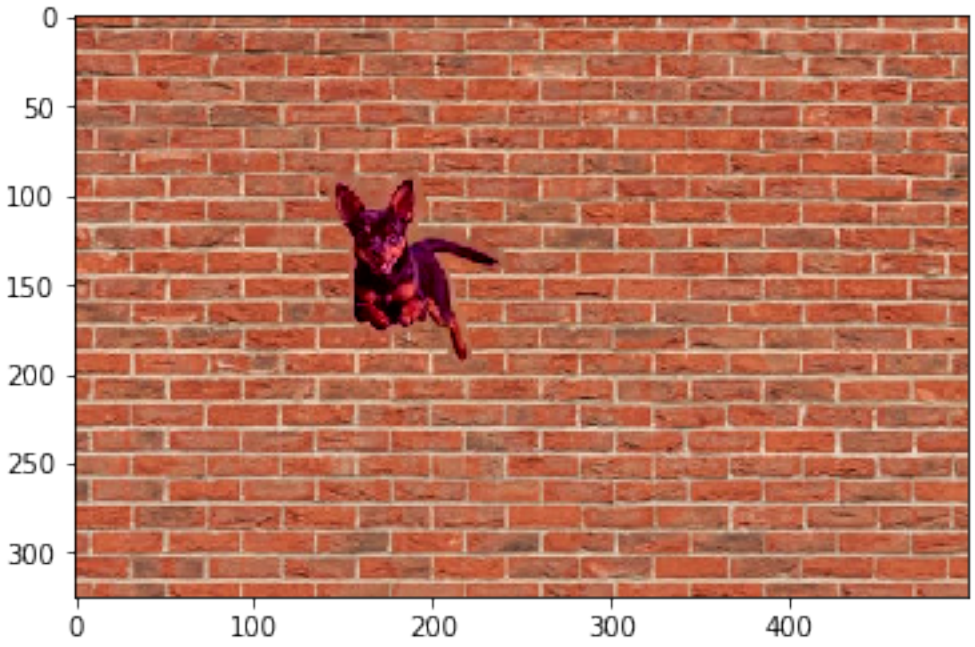
[80]:
```
def poisson_blend(cropped_object, object_mask, background_img):
    """
    :param cropped_object: numpy.ndarray One you get from align_source
    :param object_mask: numpy.ndarray One you get from align_source
    :param background_img: numpy.ndarray
    """
#     print("cropped_object: ", cropped_object.shape)
```

```
#     print("object_mask: ", object_mask.shape)
#     print("background_mask: ", background_img.shape)

    # We need to do for 3 channels
    #find co-ordinates of non-zero values in object_mask

    start = time.time()
    im = cropped_object.copy()
    co_h, co_w = cropped_object.shape[0], cropped_object.shape[1]

    # Generate co2var matrix
    co2var = np.arange(co_h * co_w).reshape(co_h, co_w)

    # number of pixels we need to process from object mask
    mask_coords = np.argwhere(object_mask != 0)
    n_constraints = 4 * len(mask_coords) + 1
    n_pixels =  co_h * co_w

    # final result
    res3Channels = np.zeros((co_h, co_w, 3))
    for ch in range (3): # for each channel

        start_time = time.time()
        print("processing channel ", ch)
        # Generata a A matrix n_contraints, n_pixels
        A = lil_matrix((n_constraints, n_pixels), dtype = np.float64)
        # Generate B Matrix with n_contraints
        b = np.zeros(n_constraints, np.float64)

        #https://piazza.com/class/k5cumohrew35en?cid=451
        #https://piazza.com/class/k5cumohrew35en?cid=466
        #https://piazza.com/class/k5cumohrew35en?cid=484
        print("time spent ", time.time() - start_time)
        start_time = time.time()
        print("apply constraints ")
        e = 0
        for (y, x) in mask_coords:

            if(object_mask[y][x+1] == 1): #x+1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y][x+1]] = -1
                b[e] = im[y][x][ch] - im[y][x+1][ch]
            else:
                A[e, co2var[y][x]] != 1
                b[e] = im[y][x][ch] - im[y][x+1][ch] +␣
 ↪background_img[y][x+1][ch]
#                 b[e] = background_img[y][x+1][ch]
```

```
            e = e + 1
            if(object_mask[y+1][x] == 1): #y+1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y+1][x]] = -1
                b[e] = im[y][x][ch] - im[y+1][x][ch]
            else:
                A[e, co2var[y][x]] = 1
                b[e] = im[y][x][ch] - im[y+1][x][ch] +␣
 ↪background_img[y+1][x][ch]
#                   b[e] = background_img[y+1][x][ch]

            e = e + 1
            if (object_mask[y][x-1] == 1): #x-1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y][x-1]] = -1
                b[e] = im[y][x][ch] - im[y][x-1][ch]
            else:
                A[e, co2var[y][x]] = 1
                b[e] = im[y][x][ch] - im[y][x-1][ch] +␣
 ↪background_img[y][x-1][ch]
#                   b[e] = background_img[y][x-1][ch]

            e = e + 1
            if (object_mask[y-1][x] == 1): #y-1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y-1][x]] = -1
                b[e] = im[y][x][ch] - im[y-1][x][ch]
            else:
                A[e, co2var[y][x]] = 1
                b[e] = im[y][x][ch] - im[y-1][x][ch] +␣
 ↪background_img[y-1][x][ch]
#                   b[e] = background_img[y-1][x][ch]
            e = e + 1

        print("calc lsqr ", time.time() - start_time)
        start_time - time.time()
        v = linalg.lsqr(csr_matrix(A), b)
        res = v[0].reshape(co_h, co_w)
        res3Channels[:,:,ch] = cv2.add(res, (1-object_mask)*background_img[:,:
 ↪,ch])
        print("channel ", ch , " processed in ", time.time() - start_time)

    print("poisson_blend total time ", time.time() - start)
    return res3Channels
```

[81]:
```
im_blend = poisson_blend(cropped_object, object_mask, background_img)
if im_blend.any():
    %matplotlib inline
    import matplotlib.pyplot as plt
    plt.imshow(im_blend)
```

```
processing channel  0
time spent  0.1333932876586914
apply constraints
calc lsqr  0.9930562973022461
channel  0  processed in  1.8847289085388184
processing channel  1
time spent  0.04763150215148926
apply constraints
calc lsqr  0.956810474395752
channel  1  processed in  1.779193639755249
processing channel  2
time spent  0.12368059158325195
apply constraints
calc lsqr  0.9884536266326904

Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).

channel  2  processed in  1.8233060836791992
poisson_blend total time  5.802590370178223
```



0.4 Part 3 Mixed Gradients (20 pts)

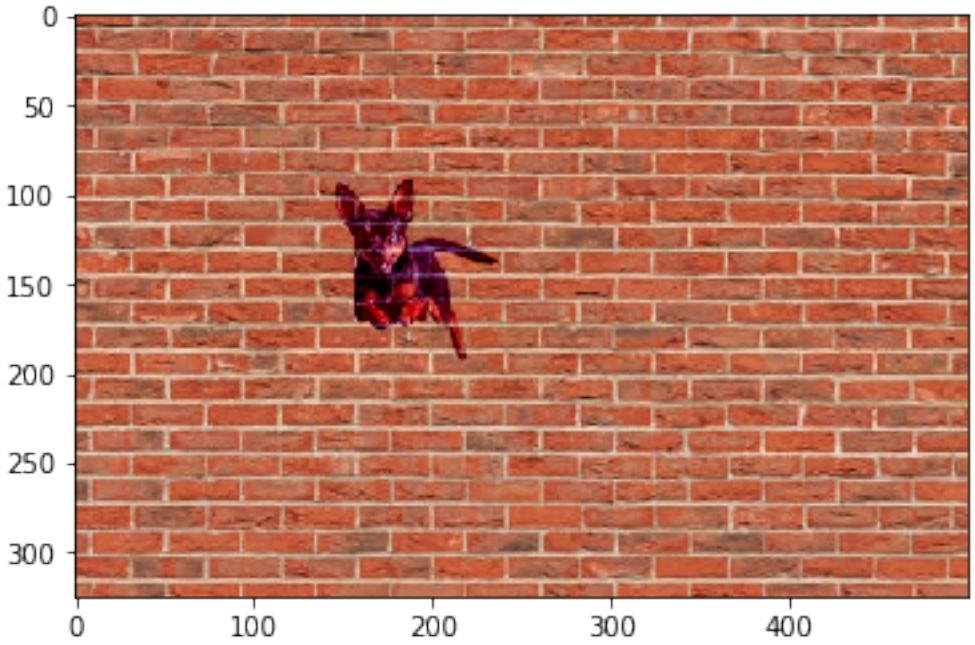
[82]:
```
def dij(si, sj, ti, tj):
#b[e] = im[y][x][ch] - im[y][x-1][ch] + background_img[y][x-1][ch]
#b[e] = im[y][x][ch] - im[y][x-1][ch]
    if (abs(si - sj) > abs(ti - tj)):
        return (si - sj)
    else:
        return (ti - tj)
#     return si - sj

def mix_blend(cropped_object, object_mask, background_img):
    """
    :param cropped_object: numpy.ndarray One you get from align_source
    :param object_mask: numpy.ndarray One you get from align_source
    :param background_img: numpy.ndarray
    """
#     print("cropped_object: ", cropped_object.shape)
#     print("object_mask: ", object_mask.shape)
#     print("background_mask: ", background_img.shape)

    # We need to do for 3 channels
    #find co-ordinates of non-zero values in object_mask

    start = time.time()
    im = cropped_object.copy()
    co_h, co_w = cropped_object.shape[0], cropped_object.shape[1]

    # Generate co2var matrix
    co2var = np.arange(co_h * co_w).reshape(co_h, co_w)

    # number of pixels we need to process from object mask
    mask_coords = np.argwhere(object_mask != 0)
    n_constraints = 4 * len(mask_coords) + 1
    n_pixels =  co_h * co_w

    # final result
    res3Channels = np.zeros((co_h, co_w, 3))
    for ch in range (3): # for each channel

        start_time = time.time()
        print("processing channel ", ch)
        # Generata a A matrix n_contraints, n_pixels
        A = lil_matrix((n_constraints, n_pixels), dtype = np.float64)
        # Generate B Matrix with n_contraints
        b = np.zeros(n_constraints, np.float64)
```

```
        #https://piazza.com/class/k5cumohrew35en?cid=451 almost code;
        #https://piazza.com/class/k5cumohrew35en?cid=466
        #https://piazza.com/class/k5cumohrew35en?cid=484
        print("time spent ", time.time() - start_time)
        start_time = time.time()
        print("apply constraints ")
        e = 0
        for (y, x) in mask_coords:

            bb = dij(im[y][x][ch], im[y][x+1][ch], background_img[y][x][ch],␣
↪background_img[y][x+1][ch])
            if(object_mask[y][x+1]) > 0: #x+1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y][x+1]] = -1
                b[e] = im[y][x][ch] - im[y][x+1][ch]
                b[e] = bb
            else:
                A[e, co2var[y][x]] = 1
                b[e] = bb + background_img[y][x+1][ch]
            e = e + 1

            bb = dij(im[y][x][ch], im[y+1][x][ch], background_img[y][x][ch],␣
↪background_img[y+1][x][ch])
            if(object_mask[y+1][x]) > 0: #y+1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y+1][x]] = -1
                b[e] = bb
            else:
                A[e, co2var[y][x]] = 1
                b[e] = bb + background_img[y+1][x][ch]
            e = e + 1

            bb = dij(im[y][x][ch], im[y][x-1][ch], background_img[y][x][ch],␣
↪background_img[y][x-1][ch])
            if (object_mask[y][x-1]) > 0: #x-1
                A[e, co2var[y][x]] = 1
                A[e, co2var[y][x-1]] = -1
                b[e] = bb
            else:
                A[e, co2var[y][x]] = 1
                b[e] = bb + background_img[y][x-1][ch]
            e = e + 1

            bb = dij(im[y][x][ch], im[y-1][x][ch], background_img[y][x][ch],␣
↪background_img[y-1][x][ch])
            if (object_mask[y-1][x] > 0): #y-1
                A[e, co2var[y][x]] = 1
```

```
                A[e, co2var[y-1][x]] = -1
                b[e] = bb
            else:
                A[e, co2var[y][x]] = 1
                b[e] = bb + background_img[y-1][x][ch]
            e = e + 1

        print("calc lsqr ", time.time() - start_time)
        start_time - time.time()
        v = linalg.lsqr(csr_matrix(A), b)
        res = v[0].reshape(co_h, co_w)
        res3Channels[:,:,ch] = cv2.add(res, (1-object_mask)*background_img[:,:
↪,ch])
        print("channel ", ch , " processed in ", time.time() - start_time)

    print("fix last mix_blend total time ", time.time() - start)
    return res3Channels
```

[83]:
```
im_mix = mix_blend(cropped_object, object_mask, background_img)
if im_mix.any():
    %matplotlib inline
    import matplotlib.pyplot as plt
    plt.imshow(im_mix)
```

```
processing channel  0
time spent  0.042109012603759766
apply constraints
calc lsqr  1.202021837234497
channel  0  processed in  2.1095380783081055
processing channel  1
time spent  0.1224663257598877
apply constraints
calc lsqr  1.2107105255126953
channel  1  processed in  2.1004998683929443
processing channel  2
time spent  0.04578566551208496
apply constraints
calc lsqr  1.1856167316436768

Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).

channel  2  processed in  2.068063974380493
fix last mix_blend total time  6.495405912399292
```



1 Bells & Whistles (Extra Points)

1.1 Color2Gray (20 pts)

[ ]:
```
color_img = cv2.imread('samples/colorBlind8.png')
plt.imshow(color_img)
```

[ ]:
```
gray_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_img, cmap="gray")
```

[ ]:
```
#taking the max among the 3 color channels solved it for me.
def gradient_norm(si, sj):
    max = 0
    for ch in range(3):
        if abs(si[ch] - sj[ch]) > abs(max):
            max = si[ch] - sj[ch]
    return max

def color2gray(img):
    im_h, im_w, _ = img.shape
    im2var = np.arange(im_h * im_w).reshape(im_h, im_w)

    n_constraints = 2 * im_h * im_w + 1 #number of constraints
    n_pixels = im_h * im_w
    A = lil_matrix((n_constraints, n_pixels), dtype = np.float64)
```

```
    b = np.zeros(n_constraints, np.float64)

    e = 0
    A[e, im2var[0,0]] = 1
    b[e] = gray_img[0,0]

    for y in range(im_h):
        for x in range(im_w):

            # Need to handle border cases. x == im_w-1 and y == im_h-1
            #Objective 1
            e = e + 1
            if(x != im_w-1):
                A[e, im2var[y][x+1]] = -1
                A[e, im2var[y][x]] = 1
                # rgb values are in here..
                # similar gradients to the original RGB
                # each value has 3 channels
                b[e] = gradient_norm(img[y,x], img[y,x+1])
            else:
                A[e, im2var[y][x]] = 1
#                 b[e] = -gray_img[y][x]

#             pdb.set_trace()
            #Objective 2
            e = e + 1
            if(y != im_h-1):
                A[e, im2var[y+1][x]] = -1
                A[e, im2var[y][x]] = 1
                b[e] = gradient_norm(img[y,x], img[y+1,x])
            else:
                A[e, im2var[y][x]] = -1
#                 b[e] = -gray_img[y][x]

    v = linalg.lsqr(csr_matrix(A), b)
    res = v[0].reshape(im_h, im_w)
    return res

grayed_image = color2gray(color_img.astype('float32'))
plt.imshow(grayed_image, cmap='gray')
```

1.2 Laplacian pyramid blending (20 pts)

[85]:
```
def laplacian_blend(img1, img2):

    mask = np.zeros_like(cropped_object)
```

```
    # create mask for 3 channels
    for i in range(3):
        mask[:,:,i] = object_mask.copy()

    levels = 121
    # generate a gaussian pyramid for A
    gImg1 = img1.copy()
    gpImg1 = [gImg1]
    for i in range(levels):
        gImg1 = cv2.pyrDown(gImg1)
        gpImg1.append(np.float32(gImg1))

    # generate a gaussian pyramid for B
    gImg2 = img2.copy()
    gpImg2 = [gImg2]
    for i in range(levels):
        gImg2 = cv2.pyrDown(gImg2)
        gpImg2.append(np.float32(gImg2))

    # generate a gaussian pyramid for Mask
    gMask = mask.copy()
    gpMask = [gMask]
    for i in range (levels):
        gMask = cv2.pyrDown(gMask)
        gpMask.append(np.float32(gMask))
    gpMask.reverse()

    #generate laplacian pyramid for Img1
    lpImg1 = [gpImg1[levels-1]]
    for i in range(levels-1, 0, -1):
        G = cv2.pyrUp(gpImg1[i])[:gpImg1[i-1].shape[0], :gpImg1[i-1].shape[1],:]
        L = np.subtract(gpImg1[i-1], G)
        lpImg1.append(L)

    #generate laplacian pyramid for Img2
    lpImg2 = [gpImg2[levels-1]]
    for i in range(levels-1, 0, -1):
        G = cv2.pyrUp(gpImg2[i])[:gpImg2[i-1].shape[0], :gpImg2[i-1].shape[1], :
↪]
        L = np.subtract(gpImg2[i-1], G)
        lpImg2.append(L)

    alphamatte = []
    for a, b, alpha in zip(lpImg1, lpImg2, gpMask[1:]):
        alphamatte.append(a * alpha + b * (1.0 - alpha))

    resAllLevels = alphamatte[0]
```

```
    for i in range(1, levels):
        lvl = cv2.pyrUp(resAllLevels)
        reslvl = lvl[:alphamatte[i].shape[0],:alphamatte[i].shape[1]]
        resAllLevels = cv2.add(reslvl, np.float32(alphamatte[i]))

    return resAllLevels
```

[86]:
```
out_lap = laplacian_blend(cropped_object, background_img)
plt.imshow(out_lap)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

[86]: `<matplotlib.image.AxesImage at 0x7fc0582d1fd0>`

1.3 More gradient domain processing (up to 20 pts)