1. Jupyter Notebook and NumPy Warmup [20pts]

We will make extensive use of Python's numerical arrays (NumPy) and interactive plotting (Matplotlib) in Jupyter notebooks for the course assignments. The first part of this assignment is intended as a gentle warm up in case you haven't used these tools before. Start by reading through the following tutorials:

If you haven't used Jupyter before, a good place to start is with the introductory documentation here:

https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server (https://jupyter-notebook.readthedocs.io/en/stable/notebook.html#starting-the-notebook-server)
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook (https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook
https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Running% (https://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Running%

This page gives a good introduction to NumPy and many examples of using NumPy along with Matplotlib:

http://www.scipy-lectures.org/intro/numpy/numpy.html (http://www.scipy-lectures.org/intro/numpy/numpy.html)

You should also get comfortable with searching through the documentation as needed

https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html (https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html) https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html)

NumPy Array Operations

Describe in words what each of each of the following statements does and what the value of `result` will be (i.e. if you were to execute `print(result)`). You should do this with out actually executing the code but instead just looking it and refering to the NumPy documentation.

[1.1]

```
import numpy as np
a = np.arange(5,15)
result = a[::3]
```

The first statement import numpy module and creates alias np, the second statement creates array a with values ranging from 5 to 14. The third statement slices array a to store the first value and the subsequent values will be the third value from the previous value. result=[5,8,11,14]

[1.2]

```
a = np.arange(1,5)
result = a[::-1]
```

The first statement creates array a with values 1 to 4. The second statement reverse the order of values stored in array a. result=[4,3,2,1]

[1.3]

```
f = np.arange(1840,1860)
g = np.where(f>1850)
result = f[g]
```

The first statement creates array f with values from 1840 to 1859. The second statement stores the index of the values in array f if the value is greater than 1850. The third statment stores the values in f whose index number are the values in array g. result=[1851,1852,1853,.....,1859]

[1.4]

```
x = np.ones((1,10))
result = x.sum(axis=1)
```

The first statement creates array x with 10 columns and one row and assgins 1 in all the array position.The second statement sums all the elements in row of array x. result=[10.]

NumPy Coding Exercises

Add or modify the code in the cells below as needed to carry out the following steps.

[1.5]

Use **matplotlib.pyplot.imread** to load in a grayscale image of your choice. If you don't have a grayscale image handy, load in a color image and then convert it to grayscale averaging together the three color channels (use **numpy.mean**).

Finally create an array A that contains the pixels in a 100x100 sub-region of your image and display the image in the notebook using the **matplotlib.pyplot.imshow** function.

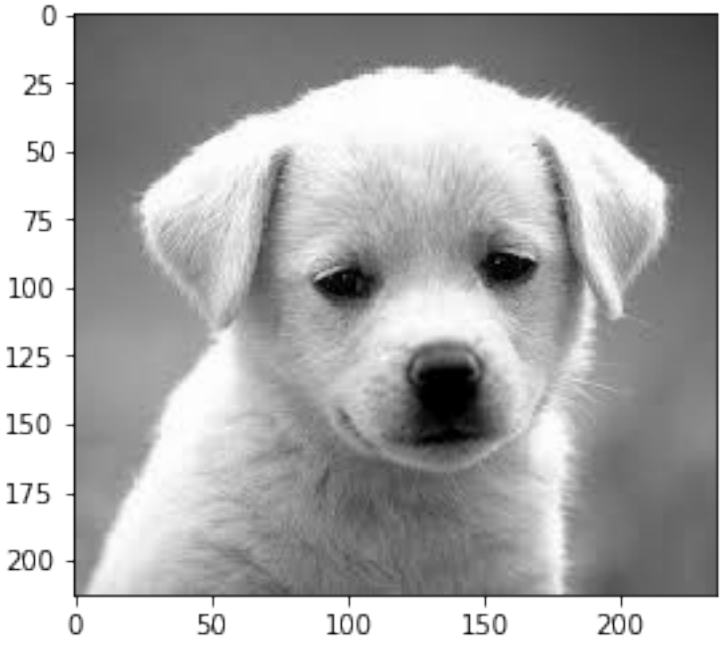
HINT: When loading an image with **imread** it is important to example the data type of the returned array. Depending on the image it may be that `I.dtype = uint8` or `I.dtype = float32` . Integer values range in [0..255] while floating point values for an image will be in [0..1]. A simple approach is to always convert images to floats, this will avoid much confusion and potential bugs later on.

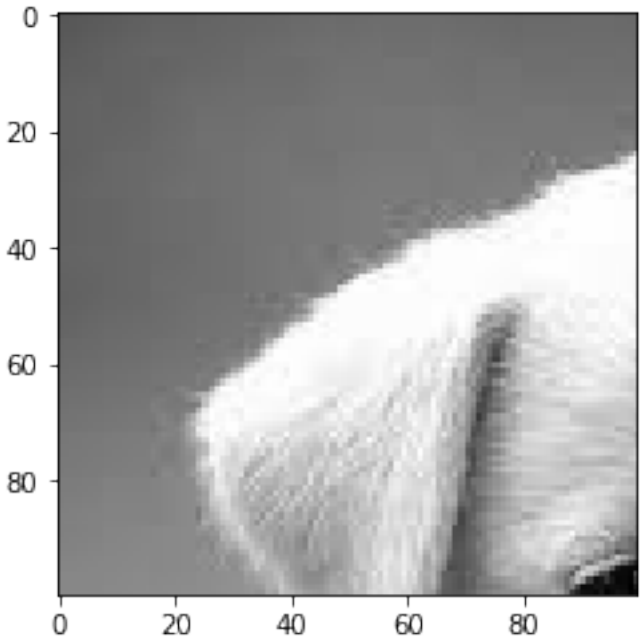
In [15]:

```
import numpy as np
import matplotlib.pyplot as plt

#load an image
I = plt.imread('grayscale_puppy.jpeg')

#display the shape of the array and data type
print("I.shape=",I.shape,"\nI.dtype=",I.dtype)

#convert to float data type and scale to [0..1] if necessary
if (I.dtype == np.uint8):
    I = I.astype(float) / 256

#I.dtype should now be float

#if your image is color (shape HxWx3), convert to grayscale by averaging togethe
r R,G,B values
grayscale=np.mean(I,axis=2)
#display the image in the notebook using a grayscale colormap
plt.imshow(I,cmap=plt.cm.gray)

#force matplotlib to go ahead and display the plot now
plt.show()
```

```
#select out a 100x100 pixel subregion of the image
A = I[:100,:100]
#display the selected subregion
plt.imshow(A,cmap=plt.cm.gray)
plt.show()
```

```
I.shape= (213, 236, 3)
I.dtype= uint8
```





[1.6]

In the cell below, describe what happens if you comment out the `plt.show()` lines?

How does the visualization of `A` change if you scale the brightness values (i.e. `plt.imshow(0.1*A,cmap=plt.cm.gray)`)?

Explain what is happening, referring to the **matplotlib** documentation as necessary (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html))

plt.show() displays a figure so if we comment out plt.show(), the figure is not displayed. In plt.imshow(0.1*A,cmap=plt.cm.gray), A is an array representation of an image which is in shape (M,N,3) i.e the image has RGB values.Each pixel is multiplied by 0.1 which increases the grayness of each pixels. After scaleing the brightness the gray pixels are almost black.

[1.7]

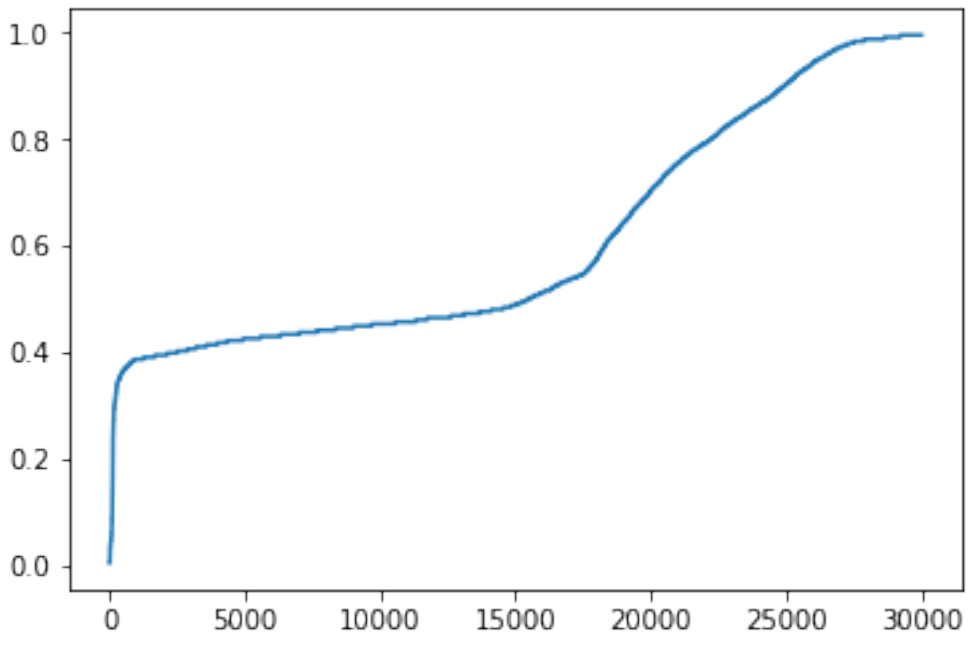
Write code in the cell below which (a) puts the values of `A` into a singe 10,000-dimensional column vector `x`, (b) sorts the entries in `x`, and (c) visualizes the contents of the sorted vector `x` by using the **matplotlib.pyplot.plot** function

In [16]:

```
#[your code here]
x=A.reshape(-1,1)
x=np.sort(x,axis=0)
plt.plot(x)
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x11be82940>]
```

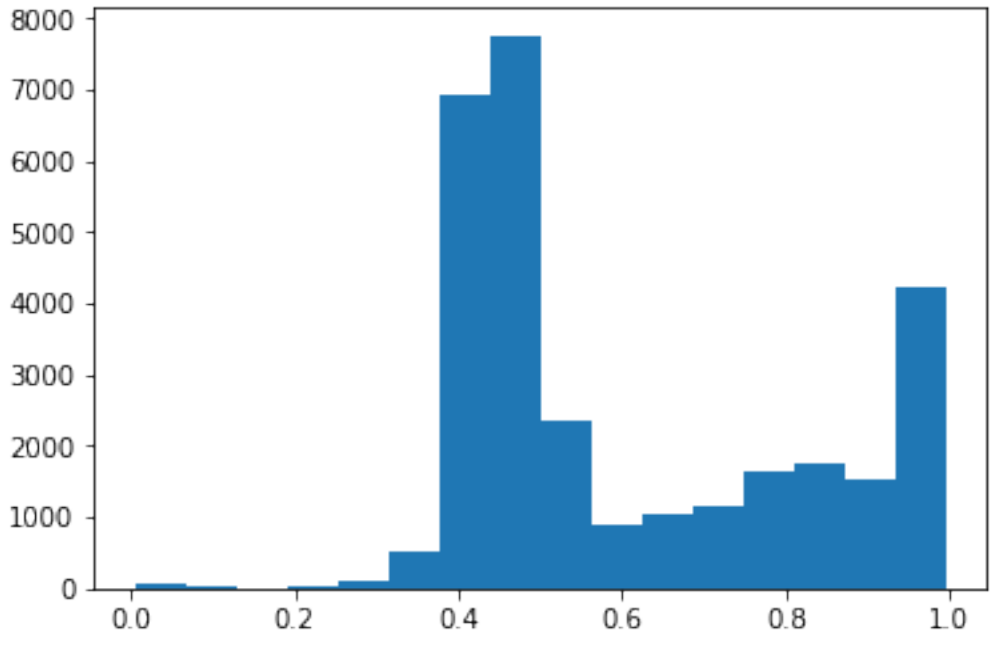


[1.8]

Display a figure showing a histogram of the pixel intensities in `A` using **matplotlib.axes.hist**. Your histogram should have 16 bins. You will need to convert `A` to a vector in order for the histogram to display correctly (otherwise it will show 16 bars for each row of A)

In [17]:

```
#[your code here]
#import matplotlib.axes as ax
a,bins,patches=plt.hist(x,bins=16)
```



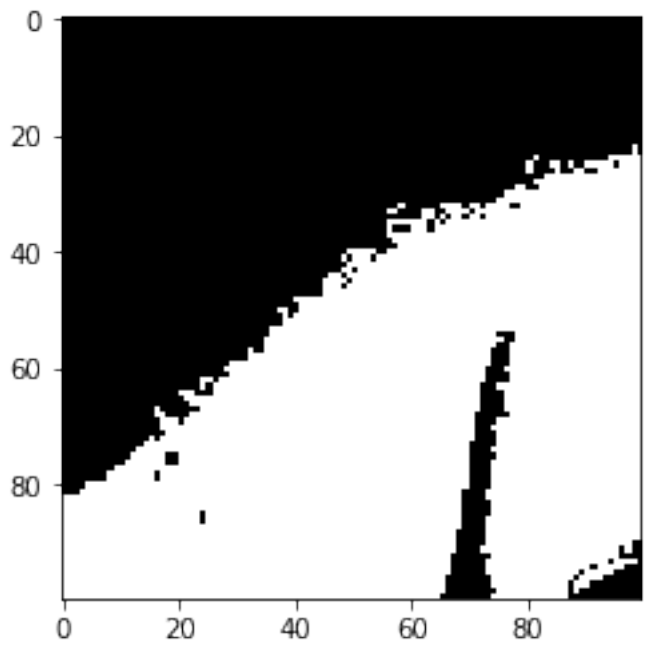
[1.9]

Create and display a new (binary) image the same size as `A`, which is white wherever the intensity in `A` is greater than a threshold specified by a variable `t`, and black everywhere else. Experiment in order to choose a value for the threshold which makes the image roughly half-white and half-black. Also print out the percentage of pixels which are black for your chosen threshold.

In [18]:

```
X=np.array(A)
threshold = 0.49
X[X>threshold]=1 #white
X[X<threshold]=0 #black
s=(X==0).sum()
print("The percentage of black pixels is:",round(s/X.size*100,2))
plt.imshow(X,cmap=plt.cm.gray)
plt.show()
```

```
The percentage of black pixels is: 50.13
```

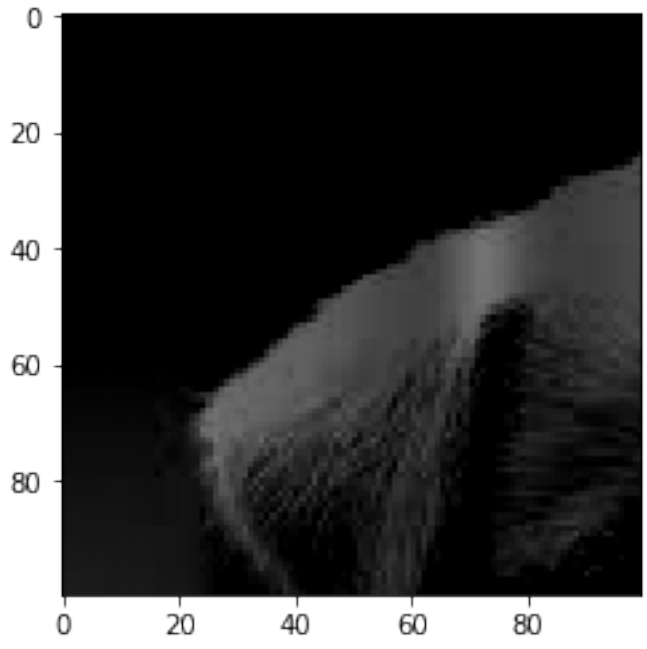


[1.10]

Generate a new grayscale image, which is the same as A, but with A's mean intensity value subtracted from each pixel. After subtracting the mean, set any negative values to 0 and display the result.

In [19]:

```
m=np.mean(A,axis=0)
X=np.array(A-m)
X[X<0]=0
plt.imshow(X,cmap=plt.cm.gray)
plt.show()
```



[1.11]

Let `y` be a column vector: `y = [1, 2, 3, 4, 5, 6]` so that `y.shape = (6,1)` . Reshape the vector into a matrix `z` using the **numpy.array.reshape** and (**numpy.array.transpose** if necessary) to form a new matrix `z` whose first column is `[1, 2, 3]` , and whose second column is `[4, 5, 6]` . Print out the resulting array `z`

In [20]:

```
y=np.array([1,2,3,4,5,6])
y.shape=(6,1)
z=np.reshape(y,(2,3))
z=z.T
print (z)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

[1.12]

Find the minimum value of `A`, if there are multple entries with the same minimum value it is fine to return the first one. Set `r` to be the row in which it occurs and `c` to be the column. Print out `r`, `c`, and `A[r,c]`

In [21]:

```
from numpy import unravel_index
index=unravel_index(A.argmin(), A.shape)
r=index[0]
c=index[1]
print (r)
print (c)
print (A[r,c])
```

```
98
98
[0.00390625 0.00390625 0.00390625]
```

[1.13]

Let v be the vector: `v = [1,8,8,2,1,3,9,8]`. Using the unique function, compute and print the total number of unique values that occur in `v`.

In [22]:

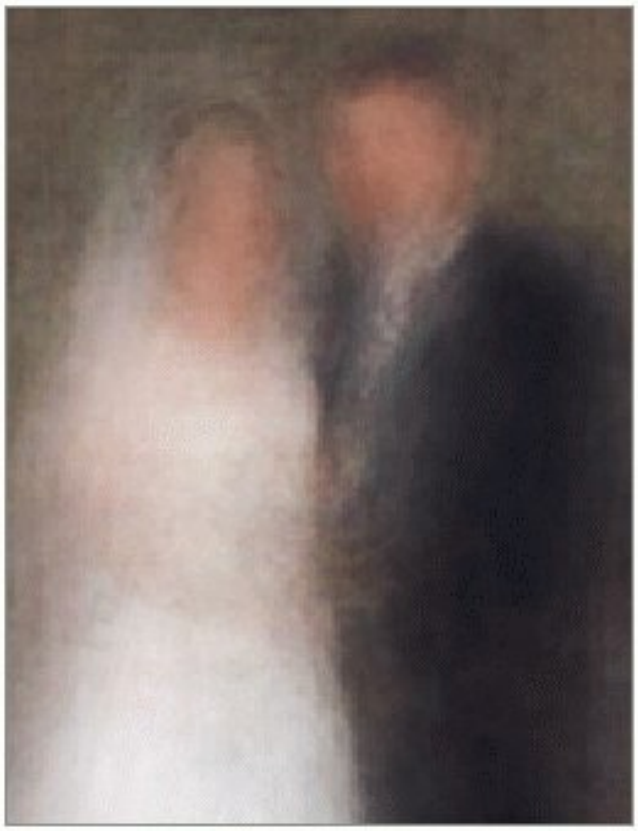
```
v = [1,8,8,2,1,3,9,8]
unique_values, frequency = np.unique(v, return_counts=True)
print (np.asarray((unique_values, frequency)).T)
print ("The number of unique values are:",unique_values.shape[0])
```

```
[[1 2]
 [2 1]
 [3 1]
 [8 3]
 [9 1]]
The number of unique values are: 5
```

2. Averaging Images [40pts]

In this exercise you will write code which loads a collection of images (which are all the same size), computes a pixelwise average of the images, and displays the resulting average.

The images below give some examples that were generated by averaging "100 unique commemorative photographs culled from the internet" by Jason Salavon. Your program will do something similar.



Newlyweds

Little Leaguer

Kids with Santa

The Graduate

Download the images provided on the Canvas course website for this assignment `averageimage_data.zip`. There are two sets, `set1` and `set2`. Notice that they are all the same size within a single set.

[2.1]

Write a function in the cell below that loads in one of the sets of images and computes their average. You can use the **os.listdir** to get the list of files in the directory. As you load in the images, you should compute an average image on the fly. Color images are represented by a 3-dimensional array of size (HxWx3) where the third dimension indexes the red, green and blue channels. You will want to compute a running average of the red, green and blue slices in order to get your final average color image.

You should encapsulate your code in a function called **average_image** that takes the image directory as an input and returns the average of the images in that directory. Your function should implement some error checking. Specifically your function should skip over any files in the directory that are not images (**plt.imread** will thrown an **OSError** if the file is not an image). It should ignore images that are not color images. Finally, it should also skip any images which are not the same height and width as the first color image you load in.

In [23]:

```
#
# these are the only modules needed for problem #2
#
import numpy as np
import os
import matplotlib.pyplot as plt
```

In [24]:

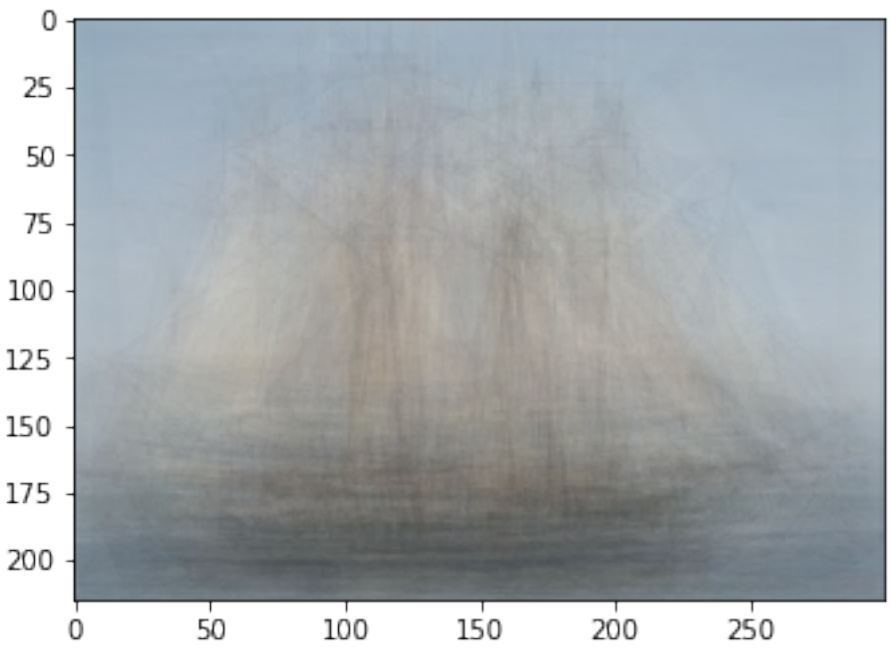
```
def average_image(dirname):
    """
    Computes the average of all color images in a specified directory and return
s the result.

    The function ignores any images that are not color images and ignores any im
ages that are not
    the same size as the first color image you load in

    Parameters
    ----------
    dirname : str
        Directory to search for images

    Returns
    -------
    numpy.array (dtype=float)
        HxWx3 array containing the average of the images found

    """
    #[your code here]
    count=0
    for file in os.listdir(dirname):
        filename = os.path.join(dirname,file)
        if os.path.isfile(filename):
            try:
                I = plt.imread(filename)
                #checking whether the image is colored or not
                width,height,channels=I.shape
                if(channels>=3):
                    width,height,channels=I.shape
                    if(count==0):
                        first_width=width
                        first_height=height
                        Iaverage=np.zeros(I.shape)
                    if((width==first_width)and(height==first_height)):
                        if (I.dtype == np.uint8):
                            I = I.astype(float) / 256
                        count+=1
                        Iaverage+=np.array(I)
            except OSError:
                print('not image file')
    Iaverage=Iaverage/count
    return Iaverage
```

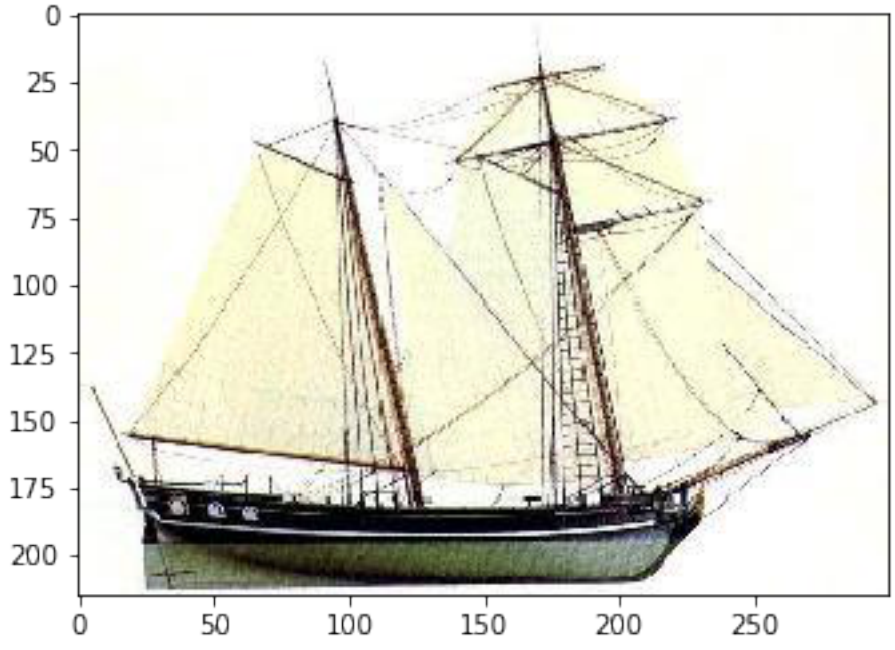
[2.2]

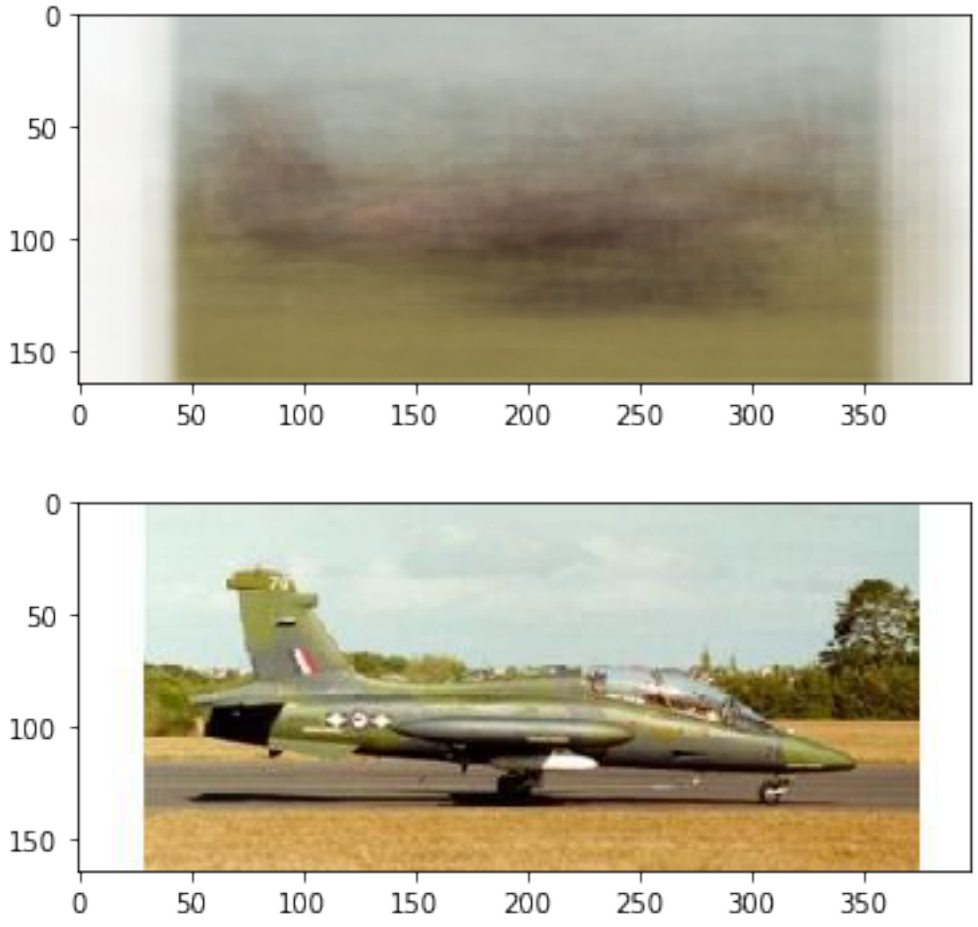
Write code below which calls your **average_image()** function twice, once for each set of images. Display the resulting average images. Also display a single example image from each set for comparison

In [25]:

```
Iav = average_image("averageimage_data/set1/")
plt.imshow(Iav)
plt.show()
img_example=plt.imread("averageimage_data/set1/im01.jpg")
plt.imshow(img_example)
plt.show()
Iav = average_image("averageimage_data/set2/")
plt.imshow(Iav)
plt.show()
img_example2=plt.imread("averageimage_data/set2/im01.jpg")
plt.imshow(img_example2)
plt.show()
```







[2.3]

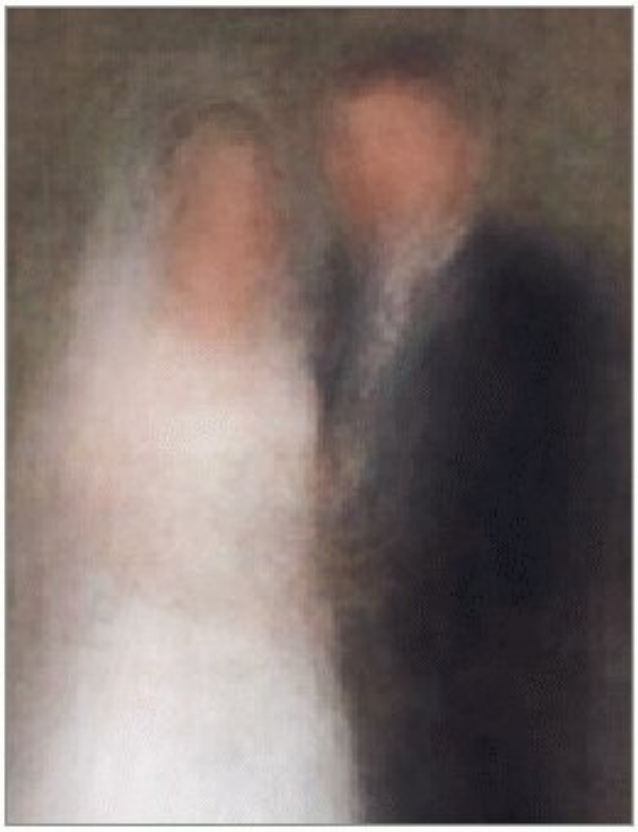
Provide a description of the appearance of the average images. Give an explanation as to why the average image does not look like the individual example images.

Image 1 is an image of boat whereas Image 2 is an image of aircraft. average_image function loaded multiple images of boats, summed all the red, green and blue channels from the images and took average of those channels which is the pixel value of the outputed image and theredfore does not look like the individula example image. Looking at image 1, we can say that almost all or most of the inputed images were boat since we can distinguish that image 1 is a boat. Similarly, for image 2, we can say that the inputed images were of aircraft.

---

3. Color sensor demosaicing [40pts]

As discussed in class, there are several steps to transform raw sensor measurements into nice looking images. These steps include Demosaicing, White Balancing and Gamma Correction. In this problem we will implement the demosaicing step. (see Szeliski Chapter 2.3) In the assignment data directory on Canvas there is a zip file containing raw images from a Canon 20D camera as well as corresponding JPEG images from the camera (`*.JPG`). The raw image files (`*.CR2`) have been converted to 16-bit PGM images (`*.pgm`) using David Coffin's dcraw program to make it easy to load them in as arrays using the supplied code below **read_pgm**



Newlyweds

Little Leaguer

Kids with Santa

The Graduate

Bayer RGGB mosaic.

The raw image has just one value per pixel. The sensor is covered with a filter array that modifies the sensitivity curve of each pixel. There are three types of filters: "red", "green", and "blue", arranged in the following pattern repeated from the top left corner:

```
R G . . .
G B
.
.
.
```

Your job is to compute the missing color values at each pixel to produce a full RGB image (3 values at each pixel location). For example, for each "green" pixel, you need to compute "blue" and "red" values. Do this by interpolating values from adjacent pixels using the linear interpolation scheme we described in class.

In [26]:

```
#
# these are the only modules needed for problem #3
#
import numpy as np
import matplotlib.pyplot as plt

#
# this function will load in the raw mosaiced data stored in the pgm file
#
def read_pgm(filename):
    """
    Return image data from a raw PGM file as a numpy array
    Format specification: http://netpbm.sourceforge.net/doc/pgm.html
    """
    infile = open(filename,'r',encoding="ISO-8859-1")

    # read in header
    magic = infile.readline()
    width,height = [int(item) for item in infile.readline().split()]
    maxval = infile.readline()

    # read in image data and reshape to 2D array, convert 16bit to float
    image = np.fromfile(infile, dtype='>u2').reshape((height, width))
    image = image.astype(float)/65535.
    return image
```

[3.1]

Implement a function `demosaic` which takes an array representing the raw image and returns a standard color image. To receive full credit, you should implement this using NumPy indexing operations like you practiced in the first part of the assignment. You should not need any for loops over individual pixel locations. You can accomplish this by either using array subindexing or alternately by using the imfilter function with the appropriate choice of filter.

In [27]:

```
def demosaic(I):
    """
    Demosaic a Bayer RG/GB image to an RGB image.

    Parameters
    ----------
    I : numpy.array  (dtype=float)
        RG/GB mosaic image of size  HxW

    Returns
    -------
    numpy.array (dtype=float)
    HxWx3 array containing the demosaiced RGB image

    """
    #slicing the image in RGB
    red=np.zeros(I.shape)
    green=np.zeros(I.shape)
    blue=np.zeros(I.shape)
    red[::2,::2]=I[::2,::2]
    green[1::2,::2]=I[1::2,::2]
    green[::2,1::2]=I[::2,1::2]
    blue[1::2,1::2]=I[1::2,1::2]

    #column wise calculation for red check for extract nums
    t1=np.zeros(red.shape)
    t2=np.zeros(red.shape)
    t1=red
    t2[:,2::2]=red[:,2::2]
    h,w=t2.shape
    extract_num=w-2
    extract=np.zeros(red.shape)
    extract[:,:extract_num]=t2[:,2:]
    addition=np.array((t1+extract)/2)
    final=np.zeros(red.shape)
    final[:,1::2]=addition[:,:extract_num+1:2]
    final_final=final+red

    #row wise calulation for red
    r1=np.zeros(red.shape)
    r2=np.zeros(red.shape)
    r1=red
    r2[2::2,:]=red[2::2,:]
    extract_num1=h-2
    e=np.zeros(red.shape)
    e[:extract_num1,:]=r2[2:,:]
    addition1=np.array((r1+e)/2)
    finala=np.zeros(red.shape)
    finala[1::2,:]=addition1[:extract_num1+1:2,:]
    red_final=finala+final_final


    #calculations for the green channel
```

```
    new_green=np.zeros((h+2,w+2))

    #padding zeros
    new_green[1:h+1,1:w+1]=green[:,:]
    new_green[1:h:2,1:w+1:2] = 0.25*new_green[0:h-1:2,1:w:2] + 0.25*new_green[2:
h+1:2,1:w:
    2] + 0.25*new_green[1:h+1:2,0:w:2] + 0.25*new_green[1:h+1:2,2:w+2:2]
    new_green[2:h+1:2,2:w+1:2] = 0.25*new_green[1:h:2,2:w+1:2] + 0.25*new_green[
3:h+2:2,2:
    w+1:2] + 0.25*new_green[2:h+1:2,1:w:2] + 0.25*new_green[2:h+1:2,3:w+2:2]

    #calculations for blue channel
    #padding zeros
    new_blue=np.zeros((h+1,w+1))
    new_blue[1:,1:] = blue
    new_blue[0,1:] = blue[1]
    new_blue[:,0] = new_blue[:,2]

    new_blue[2::2,1:w:2] = 0.5*new_blue[2::2,0:w-1:2] + 0.5*new_blue[2::2,2:w+1:
2]
    new_blue[1::2,2::2] = 0.5*new_blue[0:h-1:2,2::2] + 0.5*new_blue[2:h+1:2,2::2
]
    new_blue[1:h+1:2,1::2] = 0.25*new_blue[0:h-1:2,0:w-1:2] + 0.25*new_blue[0:h-
1:2,2::2]
    + 0.25*new_blue[2::2,0:w-1:2] + 0.25*new_blue[2::2,2::2]

    demosaic=np.zeros(shape=(h,w,3))
    green = new_green[1:new_green.shape[0]-1,1:new_green.shape[1]-1]
    blue= new_blue[1:,1:]
    demosaic[:,:,0] = red
    demosaic[:,:,1] = green
    demosaic[:,:,2] = blue

    return demosaic
```

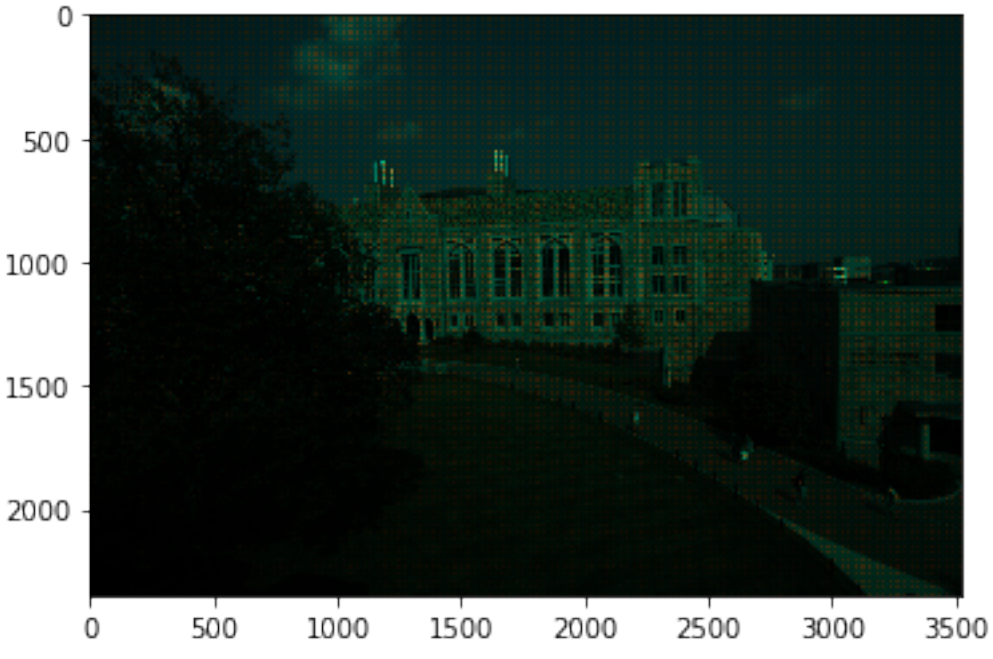
[3.2]

Write code and comments below that demonstrate the results of your demosaic function using `IMG_1308.pgm` . You are encouraged to include multiple examples for illustration. Since the images are so large, work with just the upper-left 500x500 pixel sub-block for illustrations.

You should display: (a) the original raw image with a grayscale colormap, (b) the resulting RGB image after demosaicing, (c) the corresponding part of the provided JPG file from the camera

In [28]:

```
Iraw = read_pgm("demosaic/IMG_1308.pgm")
plt.imshow(demosaic(Iraw))
plt.show()
#[enter your code here]
```



[3.3]

The correctly demosaiced image will appear darker than the JPG version provided. Provide an explanation of why this is the case based on your reading about the digital camera pipeline.

Digital camera usually creates JPEG image. According to digital camera pipeline,an image is captured then filtered to produce mosaic image which is then converted accordingly using various mechanisms with in camera to produce JPG version of image. There is loss of information in the process.Therefore, correctly demosaiced image will appear darker than the JPG image as the raw picture without any digital image processing will be carried out which somewhat corrects the luminace of the image.

In [ ]:

```
```