CSE527 Homework2

Due date: 23:59 on Oct 08, 2019 (Thuesday)

In this semester, we will use Google Colab for the assignments, which allows us to utilize resources that some of us might not have in their local machines such as GPUs. You will need to use your Stony Brook (*.stonybrook.edu) account for coding and Google Drive to save your results.

Google Colab Tutorial

Go to https://colab.research.google.com/notebooks/ (https://colab.research.google.com/notebooks/), you will see a tutorial named "Welcome to Colaboratory" file, where you can learn the basics of using google colab.

Settings used for assignments: ***Edit -> Notebook Settings -> Runtime Type (Python 3)***.

Local Machine Prerequisites

Since we are using Google Colab, all the code is run on the server environment where lots of libraries or packages have already been installed. In case of missing libraries or if you want to install them in your local machine, below are the links for installation.

- **Install Python 3.6**: https://www.python.org/downloads/ (https://www.python.org/downloads/) or use Anaconda (a Python distribution) at https://docs.continuum.io/anaconda/install/ (https://docs.continuum.io/anaconda/install/). Below are some materials and tutorials which you may find useful for learning Python if you are new to Python.
  - https://docs.python.org/3.6/tutorial/index.html (https://docs.python.org/3.6/tutorial/index.html)
  - https://www.learnpython.org/ (https://www.learnpython.org/)
  - http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html (http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html)
  - http://www.scipy-lectures.org/advanced/image_processing/index.html (http://www.scipy-lectures.org/advanced/image_processing/index.html)
- **Install Python packages**: install Python packages: `numpy` , `matplotlib` , `opencv-python` using pip, for example:

  ```
  pip install numpy matplotlib opencv-python
  ```

  Note that when using "pip install", make sure that the version you are using is python3. Below are some commands to check which python version it uses in you machine. You can pick one to execute:

  ```
  pip show pip
  ```

  ```
  pip --version
  ```

  ```
  pip -V
  ```

Incase of wrong version, use pip3 for python3 explictly.

- **Install Jupyter Notebook**: follow the instructions at http://jupyter.org/install.html (http://jupyter.org/install.html) to install Jupyter Notebook and familiarize yourself with it. *After you have installed Python and Jupyter Notebook, please open the notebook file 'HW1.ipynb' with your Jupyter Notebook and do your homework there.*

Description

In this homework you will experiment with SIFT features for scene matching and object recognition. You will work with the SIFT tutorial and code from the University of Toronto. In the compressed homework file, you will find the tutorial document (tutSIFT04.pdf) and a paper from the International Journal of Computer Vision (ijcv04.pdf) describing SIFT and object recognition. Although the tutorial document assumes matlab implemention, you should still be able to follow the technical details in it. In addition, you are **STRONGLY** encouraged to read this paper unless you're already quite familiar with matching and recognition using SIFT.

There are 3 problems in this homework with a total of 100 points. Two bonus questions with extra 5 and 15 points are provided under problem 1 and 2 respectively. The maximum points you may earn from this homework is 100 + 20 = 120 points. Be sure to read **Submission Guidelines** below. They are important.

Using SIFT in OpenCV 3.x.x in Local Machine

Feature descriptors like SIFT and SURF are no longer included in OpenCV since version 3. This section provides instructions on how to use SIFT for those who use OpenCV 3.x.x. If you are using OpenCV 2.x.x then you are all set, please skip this section. Read this if you are curious about why SIFT is removed https://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/ (https://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/).

We strongly recommend you to use SIFT methods in Colab for this homework, the details will be described in the next section.

However, if you want to use SIFT in your local machine, one simple way to use the OpenCV in-built function `SIFT` is to switch back to version 2.x.x, but if you want to keep using OpenCV 3.x.x, do the following:

1. uninstall your original OpenCV package
2. install opencv-contrib-python using pip (pip is a Python tool for installing packages written in Python), please find detailed instructions at https://pypi.python.org/pypi/opencv-contrib-python (https://pypi.python.org/pypi/opencv-contrib-python)

After you have your OpenCV set up, you should be able to use `cv2.xfeatures2d.SIFT_create()` to create a SIFT object, whose functions are listed at http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html (http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html)

Using SIFT in OpenCV 3.x.x in Colab (RECOMMENDED)

The default version of OpenCV in Colab is 3.4.3. If we use SIFT method directly, typically we will get this error message:

```
error: OpenCV(3.4.3) /io/opencv_contrib/modules/xfeatures2d/src/sift.cpp:1207: erro
r: (-213:The function/feature is not implemented) This algorithm is patented and is
excluded in this configuration; Set OPENCV_ENABLE_NONFREE CMake option and rebuild
 the library in function 'create'
```

One simple way to use the OpenCV in-built function `SIFT` in Colab is to switch the version to the one from 'contrib'. Below is an example of switching OpenCV version:

1. Run the following command in one section in Colab, which has already been included in this assignment:

    ```
    pip install opencv-contrib-python==3.4.2.16
    ```

2. Restart runtime by

    ```
    Runtime -> Restart Runtime
    ```

Then you should be able to use use `cv2.xfeatures2d.SIFT_create()` to create a SIFT object, whose functions are listed at http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html (http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html)

Some Resources

In addition to the tutorial document, the following resources can definitely help you in this homework:

- http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html (http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html)
- http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html (http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html)
- http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html?highlight=sift#cv2.SIFT (http://docs.opencv.org/3.0-beta/modules/xfeatures2d/doc/nonfree_features.html?highlight=sift#cv2.SIFT)
- http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html (http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html)

In [1]:
```
# pip install the OpenCV version from 'contrib'
pip install opencv-contrib-python==3.4.2.16
```

```
Requirement already satisfied: opencv-contrib-python==3.4.2.16 in /usr/local/
lib/python3.6/dist-packages (3.4.2.16)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist
-packages (from opencv-contrib-python==3.4.2.16) (1.16.5)
```

In [2]:
```
# import packages here
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
print(cv2.__version__) # verify OpenCV version
```

```
3.4.2
```

In [3]:
```
# Mount your google drive where you've saved your assignment folder
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, cal
l drive.mount("/content/gdrive", force_remount=True).
```

Problem 1: Match transformed images using SIFT features

{40 points + bonus 5} You will transform a given image, and match it back to the original image using SIFT keypoints.

- **Step 1 (5pt)**. Use the function from SIFT class to detect keypoints from the given image. Plot the image with keypoints scale and orientation overlaid.
- **Step 2 (10pt)**. Rotate your image clockwise by 60 degrees with the `cv2.warpAffine` function. Extract SIFT keypoints for this rotated image and plot the rotated picture with keypoints scale and orientation overlaid just as in step 1.
- **Step 3 (15pt)**. Match the SIFT keypoints of the original image and the rotated imag using the `knnMatch` function in the `cv2.BFMatcher` class. Discard bad matches using the ratio test proposed by D.Lowe in the SIFT paper. Use **0.1** as the ratio in this homework. Note that this is for display purpose only. Draw the filtered good keypoint matches on the image and display it. The image you draw should have two images side by side with matching lines across them.
- **Step 4 (10pt)**. Use the RANSAC algorithm to find the affine transformation from the rotated image to the original image. You are not required to implement the RANSAC algorithm yourself, instead you could use the `cv2.findHomography` function (set the 3rd parameter `method` to `cv2.RANSAC`) to compute the transformation matrix. Transform the rotated image back using this matrix and the `cv2.warpPerspective` function. Display the recovered image.
- **Bonus (5pt)**. You might have noticed that the rotated image from step 2 is cropped. Rotate the image without any cropping and you will be awarded an extra 5 points.

Hints: In case of too many matches in the output image, use the ratio of 0.1 to filter matches.

In [5]:
```
def drawMatches(img1, kp1, img2, kp2, matches):
    """
    My own implementation of cv2.drawMatches as OpenCV 2.4.9
    does not have this function available but it's supported in
    OpenCV 3.0.0

    This function takes in two images with their associated
    keypoints, as well as a list of DMatch data structure (matches)
    that contains which keypoints matched in which images.

    An image will be produced where a montage is shown with
    the first image followed by the second image beside it.

    Keypoints are delineated with circles, while lines are connected
    between matching keypoints.

    img1,img2 - Grayscale images
    kp1,kp2 - Detected list of keypoints through any of the OpenCV keypoint
              detection algorithms
    matches - A list of matches of corresponding keypoints through any
              OpenCV keypoint matching algorithm
    """

    # Create a new output image that concatenates the two images together
    # (a.k.a) a montage
    rows1 = img1.shape[0]
    cols1 = img1.shape[1]
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]

    # Create the output image
    # The rows of the output are the largest between the two images
    # and the columns are simply the sum of the two together
    # The intent is to make this a colour image, so make this 3 channels
    out = np.zeros((max([rows1,rows2]),cols1+cols2,3), dtype='uint8')

    # Place the first image to the left
    out[:rows1,:cols1] = np.dstack([img1, img1, img1])

    # Place the next image to the right of it
    out[:rows2,cols1:] = np.dstack([img2, img2, img2])

    # For each pair of points we have between both images
    # draw circles, then connect a line between them
    for mat in matches:

        # Get the matching keypoints for each of the images
        img1_idx = mat.queryIdx
        img2_idx = mat.trainIdx

        # x - columns
        # y - rows
        (x1,y1) = kp1[img1_idx].pt
        (x2,y2) = kp2[img2_idx].pt

        # Draw a small circle at both co-ordinates
```

```
        # radius 4
        # colour blue
        # thickness = 1
        cv2.circle(out, (int(x1),int(y1)), 4, (255, 0, 0), 1)
        cv2.circle(out, (int(x2)+cols1,int(y2)), 4, (255, 0, 0), 1)

        # Draw a line in between the two points
        # thickness = 1
        # colour blue
        cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), (0,255,0), 2
)

    # Also return the image if you'd like a copy
    return out

# Read image
img_input = cv2.imread('SourceImages/sift_input.JPG', 0)

# initiate SIFT detector
sift = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img_input, None)

# Darw keypoints on the image
# ===== This is your first output =====
res1 = cv2.drawKeypoints(img_input, kp1, outImage = img_input, flags=cv2.DRAW_
MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# rotate image
angle = -60 # because clockwise
rows, cols = img_input.T.shape
rot = cv2.getRotationMatrix2D((rows/2, cols/2), angle, 1)

cosA = abs(rot[0,0])
sinA = abs(rot[0,1])

new_rows = int(sinA * cols + cosA * rows)
new_cols = int(sinA * rows + cosA * cols)

rot[0, 2] += new_rows/2 - rows/2
rot[1, 2] += new_cols/2 - cols/2

output_img_shape = (new_rows, new_cols)

dst = cv2.warpAffine(img_input, rot, output_img_shape)


# find the keypoints and descriptors on the rotated image
kp2, des2 = sift.detectAndCompute(dst, None)

# Darw keypoints on the rotated image
# ===== This is your second output =====
res2 = cv2.drawKeypoints(dst, kp2, outImage = dst, flags=cv2.DRAW_MATCHES_FLAG
S_DRAW_RICH_KEYPOINTS)

# ====== Plot functions, DO NOT CHANGE =====
```

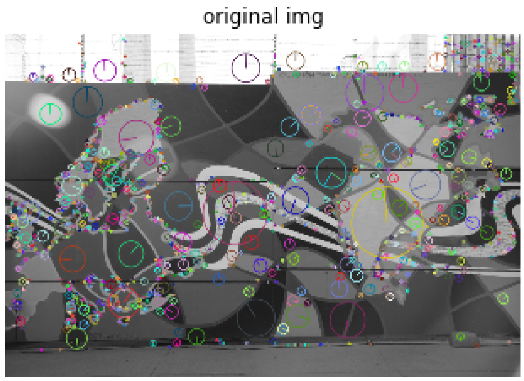
```
# Plot result images
plt.figure(figsize=(12,8))
plt.subplot(1, 2, 1)
plt.imshow(res1, 'gray')
plt.title('original img')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(res2, 'gray')
plt.title('rotated img')
plt.axis('off')
# ==========================================

# compute feature matching
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)


# Apply ratio test
good_matches = [] # Append filtered matches to this list

for m,n in matches:
    if m.distance < 0.1*n.distance:
        good_matches.append(m)
# draw matching results with the given drawMatches function
# ===== This is your third output =====
res3 = drawMatches(img_input, kp1, dst, kp2, good_matches)

# ====== Plot functions, DO NOT CHANGE =====
plt.figure(figsize=(12,8))
plt.imshow(res3)
plt.title('matching')
plt.axis('off')
# ==========================================

# estimate similarity transform
if len(good_matches) > 4:

    # find perspective transform matrix using RANSAC

    dstPoints = np.float32([kp1[m.queryIdx].pt for m in good_matches])
    srcPoints = np.float32([kp2[m.trainIdx].pt for m in good_matches])

    rot, mask = cv2.findHomography(srcPoints, dstPoints, cv2.RANSAC)
    print("Transformation Matrix = \n", rot)

    # mapping rotataed image back with the calculated rotation matrix
    # ===== This is your fourth output =====
    res4 = cv2.warpPerspective(src = dst, M = rot, dsize = (img_input.T.shape
))
else:
    print("Not enough matches are found - %d/%d" % (len(good_matches),4))

# ====== Plot functions, DO NOT CHANGE =====
# plot result images
plt.figure(figsize=(12,8))
plt.subplot(1, 2, 1)
```

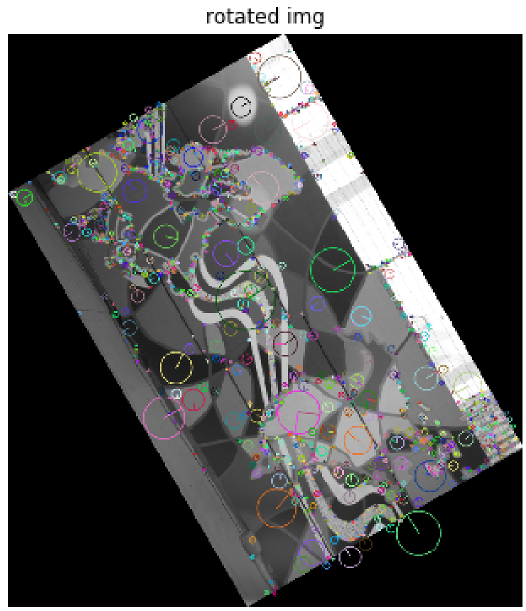
```
plt.imshow(img_input, 'gray')
plt.title('original img')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(res4, 'gray')
plt.title('recovered img')
plt.axis('off')
# ==========================================
```

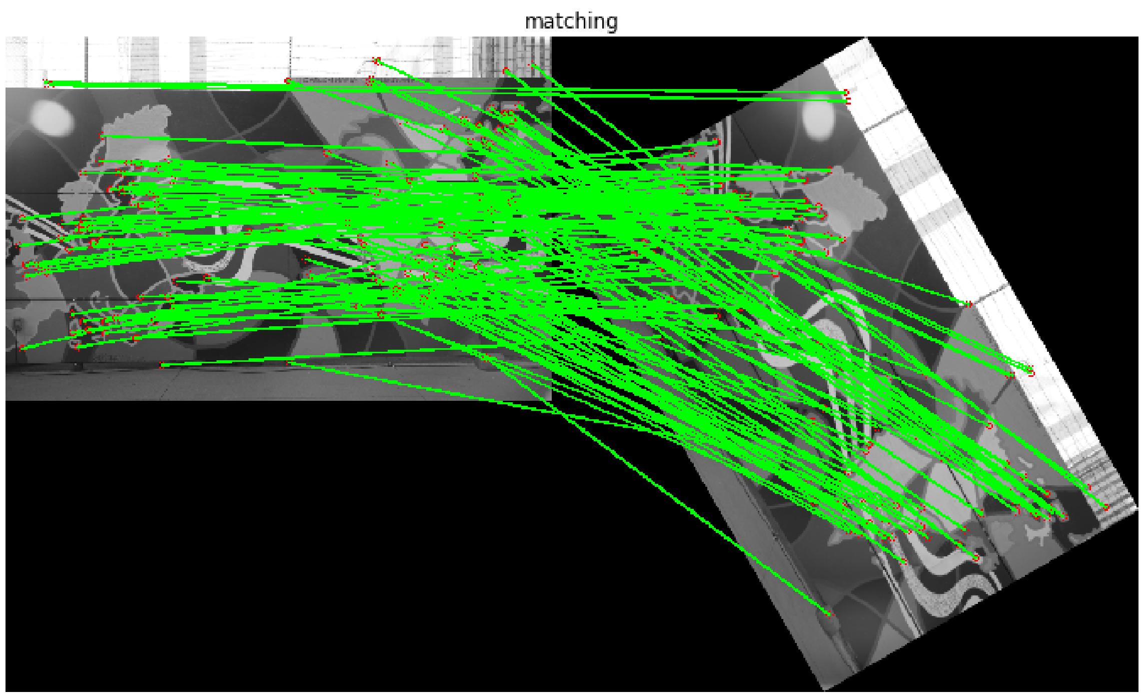
```
Transformation Matrix =
 [[ 4.99960632e-01  8.66397728e-01 -1.72992280e+02]
 [-8.66241420e-01  5.00213666e-01  3.00326429e+02]
 [-1.93612164e-07  6.03422656e-07  1.00000000e+00]]
```

Out[5]:

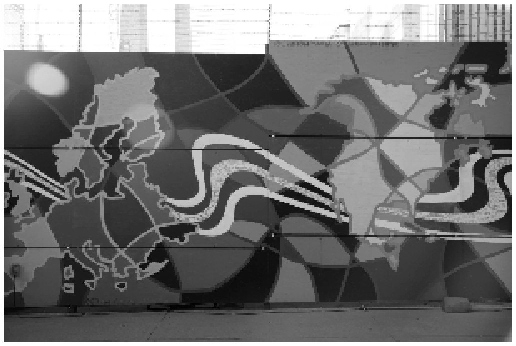
```
(-0.5, 599.5, 399.5, -0.5)
```







original img



recovered img

Problem 2: Scene stitching with SIFT features

{30 points + 15 bonus} You will match and align between different views of a scene with SIFT features.

Use `cv2.copyMakeBorder` function to pad the center image with zeros into a larger size. *Hint: the final output image should be of size 1608×1312.* Extract SIFT features for all images and go through the same procedures as you did in problem 1. Your goal is to find the affine transformation between the two images and then align one of your images to the other using `cv2.warpPerspective` . Use the `cv2.addWeighted` function to blend the aligned images and show the stitched result. Examples can be found at http://docs.opencv.org/trunk/d0/d86/tutorial_py_image_arithmetics.html (http://docs.opencv.org/trunk/d0/d86/tutorial_py_image_arithmetics.html). Use parameters **0.5 and 0.5** for alpha blending.

- **Step 1 (15pt)**. Compute the transformation from the right image to the center image. Warp the right image with the computed transformation. Stitch the center and right images with alpha blending. Display the SIFT feature matching between the center and right images like you did in problem 1. Display the stitched result (center and right image).
- **Step 2 (15pt)** Compute the transformation from the left image to the stitched image from step 1. Warp the left image with the computed transformation. Stich the left and result images from step 1 with alpha blending. Display the SIFT feature matching between the result image from step 1 and the left image like what you did in problem 1. Display the final stitched result (all three images).
- **Bonus (15pt)**. Instead of using `cv2.addWeighted` to do the blending, implement Laplacian Pyramids to blend the two aligned images. Tutorials can be found at http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_pyramids/py_pyramids.html (http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_pyramids/py_pyramids.html). Display the stitched result (center and right image) and the final stitched result (all three images) with laplacian blending instead of alpha blending.

Note that for the resultant stitched image, some might have different intensity in the overlapping and other regions, namely the overlapping region looks brighter or darker than others. To get full credit, the final image should have uniform illumination.

Hints: You need to find the warping matrix between images with the same mechanism from problem 1. You will need as many reliable matches as possible to find a good homography so DO NOT use 0.1 here. A suggested value would be 0.75 in this case.

When you warp the image with cv2.warpPerspective, an important trick is to pass in the correct parameters so that the warped image has the same size with the padded_center image. Once you have two images with the same size, find the overlapping part and do the blending.

In [6]:
```
imgCenter = cv2.imread('SourceImages/stitch_m.jpg', 0)
imgRight  = cv2.imread('SourceImages/stitch_r.jpg', 0)
imgLeft   = cv2.imread('SourceImages/stitch_l.jpg', 0)

# initalize the stitched image as the center image
imgCenter = cv2.copyMakeBorder(imgCenter,604,604,356,356,cv2.BORDER_CONSTANT)

# blend two images
def alpha_blend(img, warped, flag):

    rows,cols = img.shape
    reg_of_int = warped[0:rows, 0:cols]

    ret, mask = cv2.threshold(img, 10, 255, cv2.THRESH_BINARY)
    mask_inv = cv2.bitwise_not(mask)

    warped_bg = cv2.bitwise_and(reg_of_int, reg_of_int, mask = mask_inv) # the
portion of image which needs to be added

    weight = 0.5 if flag else 1 # for second time blending, weight = 1 is used
for stitched image which was used before with weight = 0

    blended = cv2.addWeighted(warped_bg, 0.5, img, weight, 0)
    return blended

def Laplacian_Blending(A, B, mask, num_levels=6):
    # assume mask is float32 [0,1]
#     mask = mask.astype(np.float32)
    # generate Gaussian pyramid for A,B and mask

    print(A.shape, B.shape, mask.shape)
    G = A.copy()
    gpA = [G]
    for i in range(6):
        G = cv2.pyrDown(G)
        gpA.append(G)

    G = B.copy()
    gpB = [G]
    for i in range(6):
        G = cv2.pyrDown(G)
        gpB.append(G)

    G = mask.copy()
    gpmask = [G]
    for i in range(6):
        G = cv2.pyrDown(G)
        gpmask.append(G)


    # generate Laplacian Pyramids for A,B and masks

    lpA = [gpA[5]]
    for i in range(5,0,-1):
        size = (gpA[i-1].shape[1], gpA[i-1].shape[0])
        GE = cv2.pyrUp(gpA[i], dstsize = size)
```

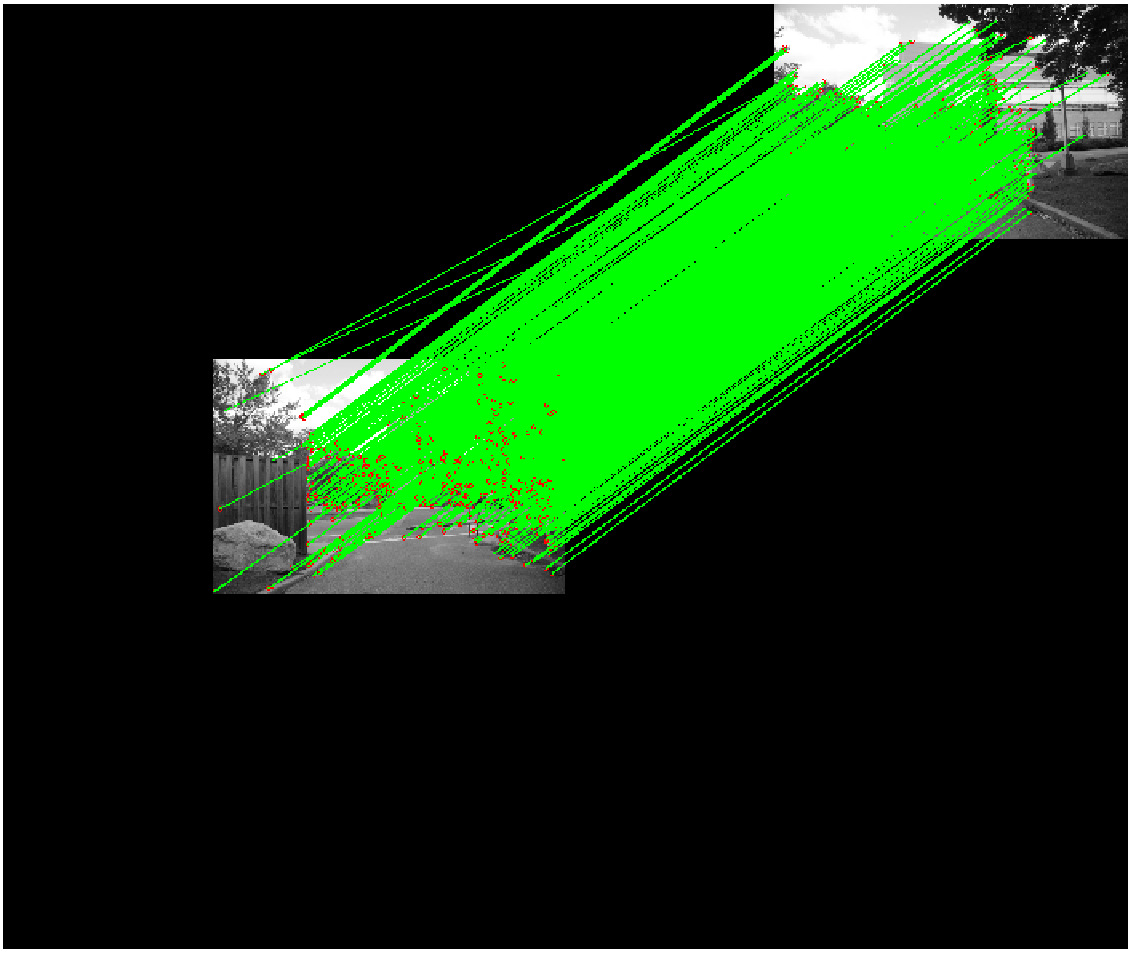
```
        L = cv2.subtract(gpA[i-1],GE)
        lpA.append(L)

    lpB = [gpB[5]]
    for i in range(5,0,-1):
        size = (gpB[i-1].shape[1], gpB[i-1].shape[0])
        GE = cv2.pyrUp(gpB[i], dstsize = size)
        L = cv2.subtract(gpB[i-1],GE)
        lpB.append(L)

    lpmask = [gpmask[5]]
    for i in range(5,0,-1):
        size = (gpmask[i-1].shape[1], gpmask[i-1].shape[0])
        GE = cv2.pyrUp(gpmask[i], dstsize = size)
        L = cv2.subtract(gpmask[i-1],GE)
        lpmask.append(L)

    # Now blend images according to mask in each level
    LS = []
    for la,lb,mask in zip(lpA, lpB, lpmask):
        ret, mask = cv2.threshold(la, 10, 255, cv2.THRESH_BINARY)
        mask_inv = cv2.bitwise_not(mask)

        rows,cols = la.shape
        reg_of_int = lb[0:rows, 0:cols]

        lb = cv2.bitwise_and(reg_of_int, reg_of_int, mask = mask_inv)

        ls = cv2.add(la, lb)
        LS.append(ls)


    # now reconstruct
    ls_ = LS[0]
    for i in range(1,6):
        size = (LS[i].shape[1], LS[i].shape[0])
        ls_ = cv2.pyrUp(ls_, dstsize = size)
        ls_ = cv2.add(ls_, LS[i])
    return ls_

def getTransform(img1, img2):
    # compute sift descriptors

    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    # find all mactches
    matches = bf.knnMatch(des1, des2, k = 2)


    # Apply ratio test
    good_matches = [] # Append filtered matches to this list

    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good_matches.append(m)
```

```
    # draw matches
    img_match = drawMatches(img1, kp1, img2, kp2, good_matches) # call given d
rawMatches function

    # estimate transform matrix using RANSAC
    if len(good_matches) > 4:

        # find perspective transform matrix using RANSAC

        dstPoints = np.float32([kp1[m.queryIdx].pt for m in good_matches])
        srcPoints = np.float32([kp2[m.trainIdx].pt for m in good_matches])

    else:
        print("Not enough matches are found - %d/%d" % (len(good_matches),4))

    H, mask = cv2.findHomography(srcPoints, dstPoints, cv2.RANSAC) # call cv2.
findHomography
#     print("Transformation Matrix = \n", H)

    return H, img_match

def perspective_warping(imgCenter, imgLeft, imgRight):

    # Get homography from right to center
    # ===== img_match1 is your first output =====
    T_R2C, img_match1 = getTransform(imgCenter, imgRight) # call getTransform
 to get the transformation from the right to the center image
#     test(imgRight, imgCenter, T_R2C)
    # Blend center and right
    # ===== stitched_cr is your second output =====

    warped = cv2.warpPerspective(src = imgRight, M = T_R2C, dsize = imgCenter.
T.shape)
    stitched_cr = alpha_blend(imgCenter, warped, flag = True) # call alpha_ble
nd

    # Get homography from left to stitched center_right
    # ===== img_match2 is your third output =====
    T_L2CR, img_match2 = getTransform(stitched_cr, imgLeft) # call getTransfor
m to get the transformation from the left to stitched_cr

    # Blend left and center_right
    # ===== stitched_res is your fourth output =====
    warped = cv2.warpPerspective(src = imgLeft, M = T_L2CR, dsize = stitched_c
r.T.shape)
    stitched_res = alpha_blend(stitched_cr, warped, flag = False) # call alpha
_blend

    return stitched_res, stitched_cr, img_match1, img_match2

def perspective_warping_laplacian_blending(imgCenter, imgLeft, imgRight):

    # Get homography from right to center
    T_R2C, img_match1 = getTransform(imgCenter, imgRight)
    # Blend center and right
    # ===== This is your first bonus output =====
    warped = cv2.warpPerspective(src = imgRight, M = T_R2C, dsize = imgCenter.
```

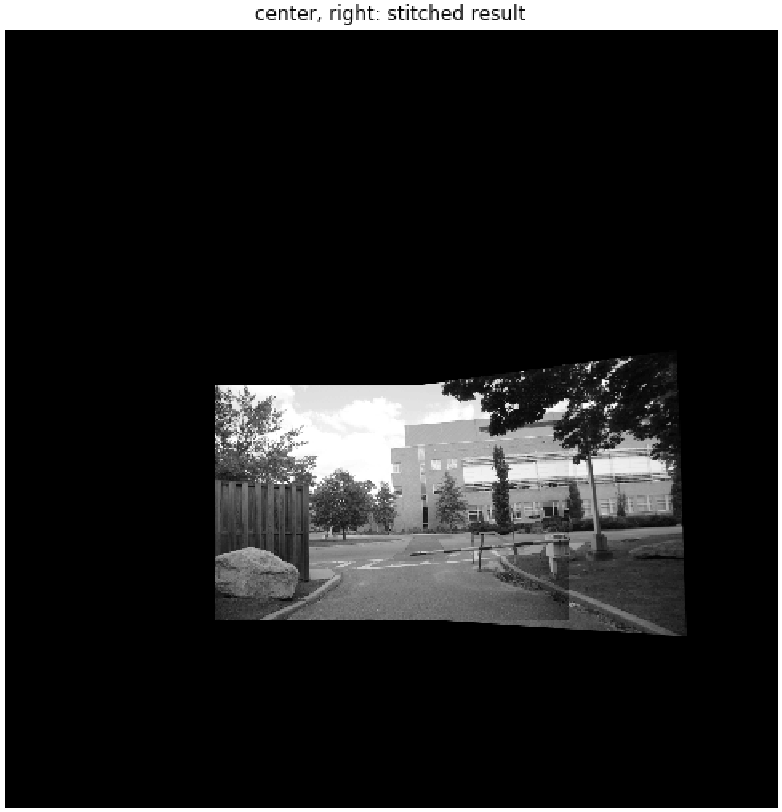
```
T.shape)

    ret, mask = cv2.threshold(warped, 10, 255, cv2.THRESH_BINARY)

    stitched_cr = Laplacian_Blending(imgCenter, warped, mask, num_levels=6) #
 call Laplacian_Blending to stitch the center and right image

    # Get homography from left to stitched center_right
    T_L2CR, img_match2 = getTransform(stitched_cr, imgLeft)
    # Blend left and center_right
    # ===== This is your second bonus output =====
    warped = cv2.warpPerspective(src = imgLeft, M = T_L2CR, dsize = stitched_c
r.T.shape)

    ret, mask = cv2.threshold(warped, 10, 255, cv2.THRESH_BINARY)

    stitched_res = Laplacian_Blending(stitched_cr, warped, mask, num_levels=6)
# call Laplacian_Blending to stitch the stitched_cr and left image

    return stitched_res, stitched_cr


# ====== Plot functions, DO NOT CHANGE =====
stitched_res, stitched_cr, img_match1, img_match2 = perspective_warping(imgCen
ter, imgLeft, imgRight)
stitched_res_lap, stitched_cr_lap = perspective_warping_laplacian_blending(img
Center, imgLeft, imgRight)

plt.figure(figsize=(25,50))
plt.subplot(4, 1, 1)
plt.imshow(img_match1, cmap='gray')
plt.title("center and right matches")
plt.axis('off')
plt.subplot(4, 1, 2)
plt.imshow(stitched_cr, cmap='gray')
plt.title("center, right: stitched result")
plt.axis('off')
plt.subplot(4, 1, 3)
plt.imshow(img_match2, cmap='gray')
plt.title("left and center_right matches")
plt.axis('off')
plt.subplot(4, 1, 4)
plt.imshow(stitched_res, cmap='gray')
plt.title("left, center, right: stitched result")
plt.axis('off')
plt.show()

plt.figure(figsize=(25,50))
plt.subplot(2, 1, 1)
plt.imshow(stitched_cr_lap, cmap='gray')
plt.title("Bonus, center, right: stitched result")
plt.axis('off')
plt.subplot(2, 1, 2)
plt.imshow(stitched_res_lap, cmap='gray')
plt.title("Bonus, left, center, right: stitched result")
plt.axis('off')
# =============================================
```

```
(1608, 1312) (1608, 1312) (1608, 1312)
(1608, 1312) (1608, 1312) (1608, 1312)
```

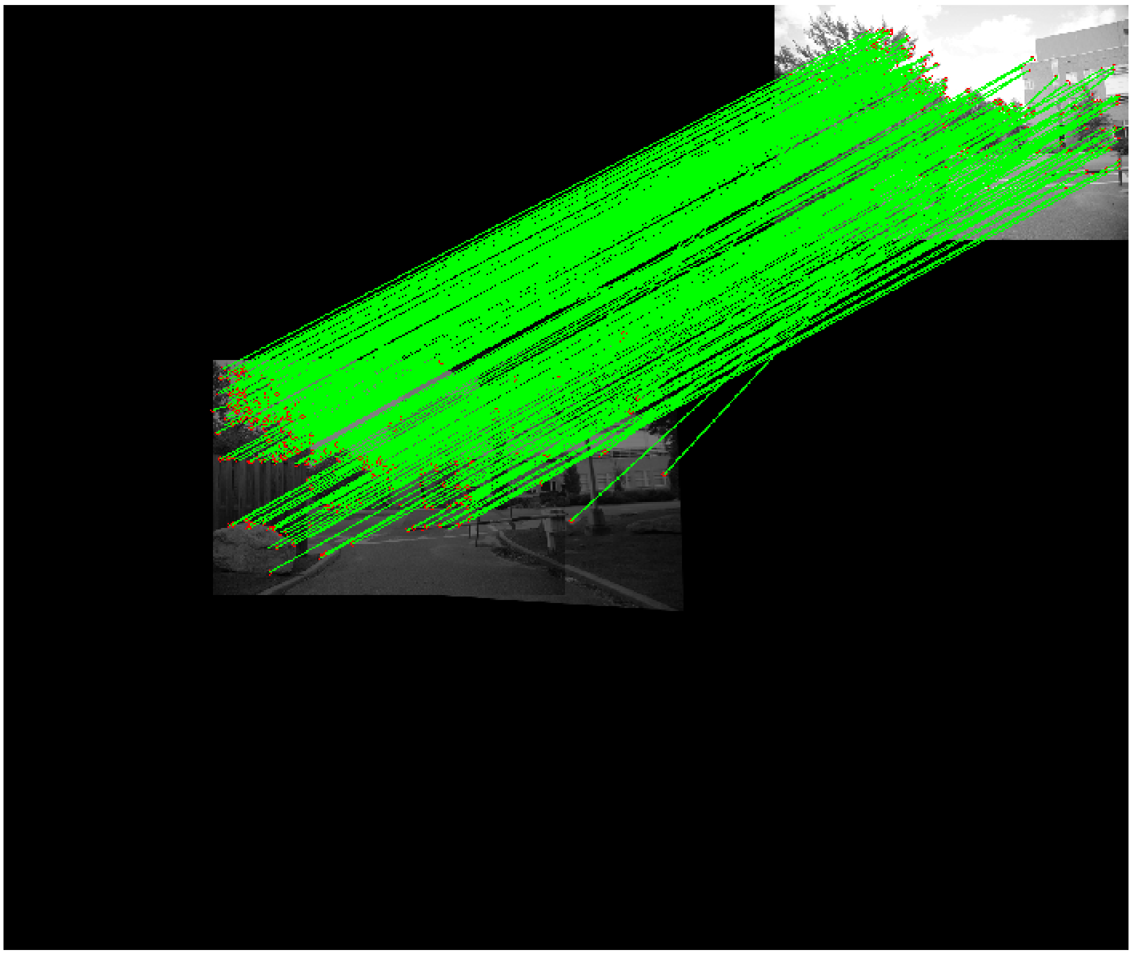
center and right matches



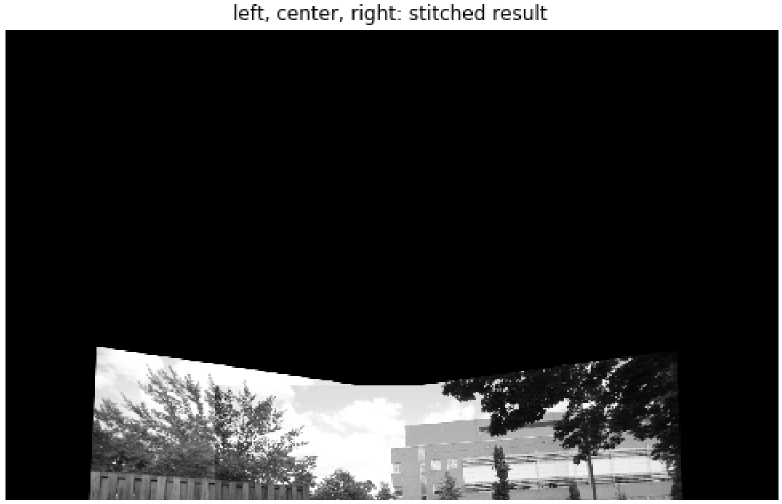
center, right: stitched result



left and center_right matches



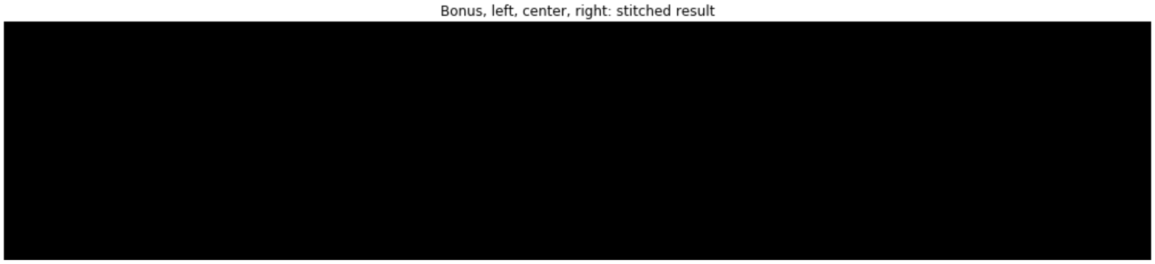
left, center, right: stitched result



Out[6]: (-0.5, 1311.5, 1607.5, -0.5)

Bonus, center, right: stitched result

Bonus, left, center, right: stitched result



Problem 3: Object Recognition with HOG features

{30 points} You will use the histogram of oriented gradients (HOG) to extract features from objects and recognize them.

HOG decomposes an image into multiple cells, computes the direction of the gradients for all pixels in each cell, and creates a histogram of gradient orientation for that cell. Object recognition with HOG is usually done by extracting HOG features from a training set of images, learning a support vector machine (SVM) from those features, and then testing a new image with the SVM to determine the existence of an object.

You can use `cv2.HOGDescriptor` to extract the HoG feature and `cv2.ml.SVM_create` for SVMs (and a lot of other algorithms). You can also use Python machine learning packages for SVM, e.g. `scikit-learn` and for HoG computation, e.g. `scikit-image`. Please find the OpenCV SVM tutorial at [https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/ (https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/)](https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/).

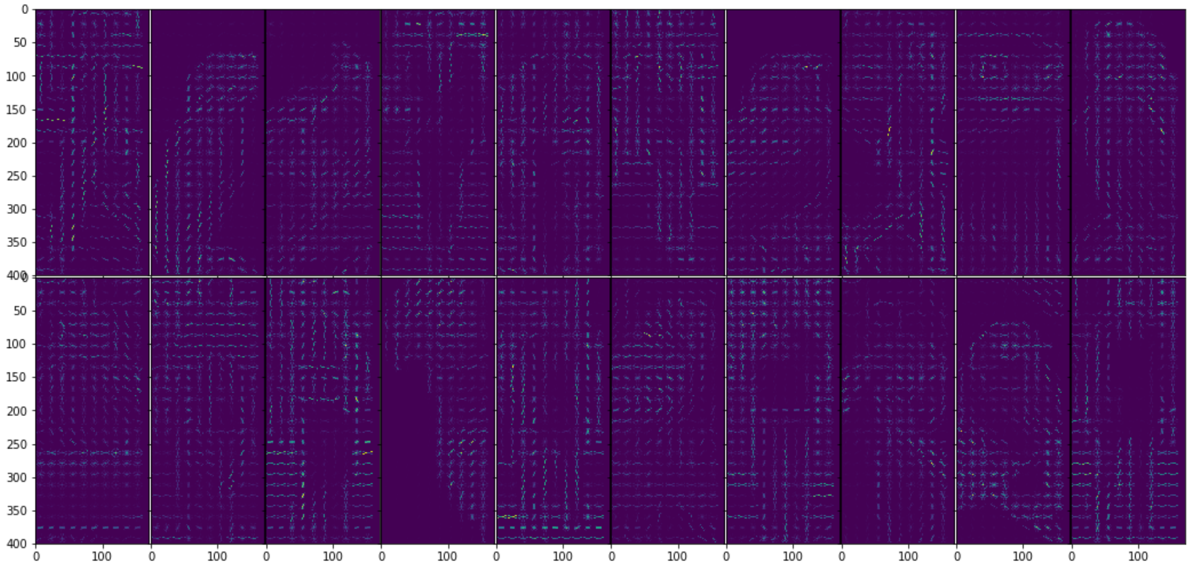
An image set located under SourceImages/human_vs_birds is provided containing 20 images. You will first train an SVM with the HoG features and then predict the class of an image with the trained SVM. For simplicity, we will be dealing with a binary classification problem with two classes, namely, birds and humans. There are 10 images for each class.

Some of the function names and arguments are provided, you may change them as you see fit.

- **Step 1 (5pts)**. Load in the images and create a vector of corresponding labels (0 for bird and 1 for human). An example label vector should be something like [1,1,1,1,1,0,0,0,0,0]. Shuffle the images randomly and display them in a 2 x 10 grid with figsize = (18, 15).
- **Step 2 (10pts)**. Extract HoG features from all images. You can use the OpenCV function `cv2.HOGDescriptor` or hog routine from `scikit-image`. Display the HoG features for all images in a 2 x 10 grid with figsize = (18, 15).
- **Step 3**. Use the first 16 examples from the shuffled dataset as training data on which to train an SVM. The rest 4 are used as test data. Reshape the HoG feature matrix as necessary to feed into the SVM. Train the classifier. **DO NOT train with test data.** No output is expected from this part.
- **Step 4 (15pts)**. Perform predictions with your trained SVM on the test data. Output a vector of predictions, a vector of ground truth labels, and prediction accuracy.

In [7]:
```
import skimage.exposure
from skimage.feature import hog
from sklearn.svm import LinearSVC

imgs_per_class = 10
seed = 42
# load data
def loadData(file):

    # Implement your loadData(file) here
    return [cv2.imread(file + str(i) + '.png') for i in range(1, imgs_per_clas
s + 1)]

x_birds = loadData('SourceImages/human_vs_birds/bird_')
x_humans = loadData('SourceImages/human_vs_birds/human_')

# ===== Display your first graph here =====

# create a vector of labels
# assume labels: bird = 0, human = 1


y_birds = np.zeros(imgs_per_class)
y_humans = np.ones(imgs_per_class)

x_data = np.concatenate([x_humans, x_birds])
y_data = np.concatenate([y_humans, y_birds])

permut = np.random.RandomState(seed).permutation(len(x_data))

x_shuffled_data = x_data[permut]
y_shuffled_data = y_data[permut]

import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
fig  = plt.figure(figsize=(18,15))

grid = ImageGrid(fig, 111, nrows_ncols=(2, 10))

for ax, im in zip(grid, x_shuffled_data):
  ax.imshow(im)

plt.show()



# fig, ax_lst = plt.subplots(2, 10)

# for i, img in enumerate(x_shuffled_data):
#   plt.imshow(img, axis=ax_lst[i])
# plt.show()
```

In [8]:
```
# Compute HOG features for the images
from skimage.feature import hog
def computeHOGfeatures(img):

    # Implement your computeHOGfeatures() here


    fd, hog_image = hog(img, orientations=8, pixels_per_cell=(16, 16),
                        cells_per_block=(1, 1), visualize=True, multichannel=True)
    return hog_image




# Compute HOG descriptors
HOGf = []
for img in x_shuffled_data:
  HOGf.append(computeHOGfeatures(img))

# ===== Display your second graph here =====


import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
fig  = plt.figure(figsize=(18,15))

grid = ImageGrid(fig, 111, nrows_ncols=(2, 10))

for ax, im in zip(grid, HOGf):
  ax.imshow(im)

plt.show()



# reshape feature matrix

HOGf = np.array(HOGf).reshape(len(HOGf),-1)

# Split the data and labels into train and test set

features_train, features_test = np.split(HOGf, [16])
labels_train, labels_test = np.split(y_shuffled_data, [16])

print(features_train.shape, labels_train.shape)
```



```
(16, 68000) (16,)
```

In [9]:
```python
# train model with SVM
from sklearn.svm import LinearSVC
# call LinearSVC
clf = LinearSVC()
# train SVM
clf.fit(features_train, labels_train)
# call clf.predict
estimated_labels = clf.predict(features_test)

# ===== Output functions ======
print('estimated labels: ', estimated_labels) # fill in here #)
print('ground truth labels: ', labels_test) # fill in here #)
print('Accuracy: ', np.mean(estimated_labels == labels_test))# fill in here #,
'%')
```

```
estimated labels:  [1. 0. 0. 1.]
ground truth labels:  [1. 0. 0. 1.]
Accuracy:  1.0
```

In [0]:
```python

```