FEATURE DETECTION, MATCHING & AUGMENTATION

COMPUTER VISION - IT-420

SUBMITTED BY

Name	VASU SHARMA
Roll No.	2K21/IT/193

Name	SAAR AGRAWAL
Roll No.	2K21/IT/153

Name	VIKASH
Roll No.	2K21/IT/196

AUGMENTED REALITY

Augmented Reality (AR) is the integration of digital information with the user's environment in real time. Which creates a totally artificial environment. AR users experience a real world environment with generated perceptual information overlaid on top of it.

It has variety of uses, from assisting in decision making process to entertainment. It is use to either visually change natural environments in some way or to provide additional information to users.

ORB (Oriented FAST and Rotated BRIEF)

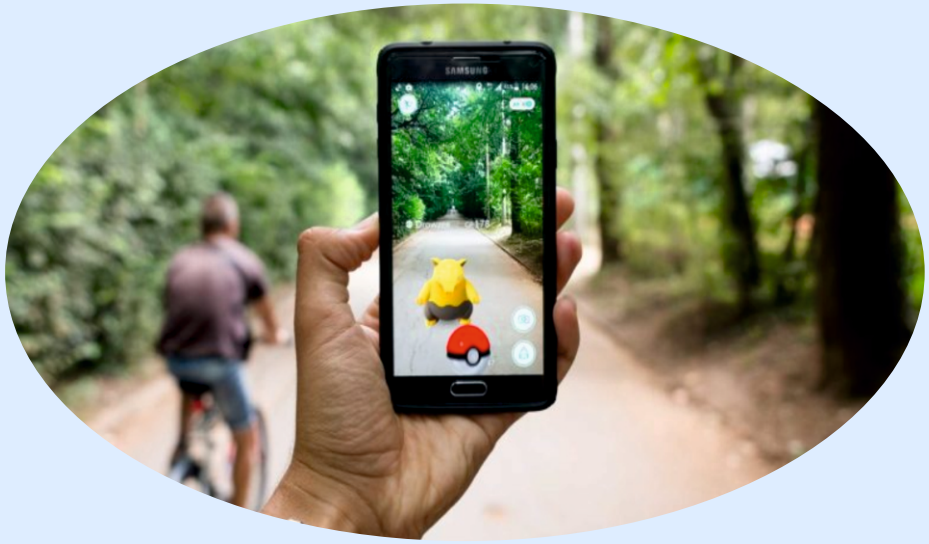
Oriented FAST and Rotated BRIEF (ORB) was developed at OpenCV labs. It is a fusion of FAST keypoint detector and BRIEF descriptor with many modification to enhance the performance It also use pyramid to produce multi scale features.

It compute the intensity weighted centroid of the patch with the located corner at center. The direction of the vector form this corner point to centroid gives the orientation.

It is used in feature detection because of its robustness, relative simplicity & computational efficiency.

DIFF b/w AR & VR

- AR uses a real world setting while VR is completely virtual.
- AR users can control their presence in the real world; VR users are controlled by the system.
- VR requires a headset device, but AR can be accessed with a smart phone.
- AR enhances both the virtual & real world while VR only enhances a fictional reality.



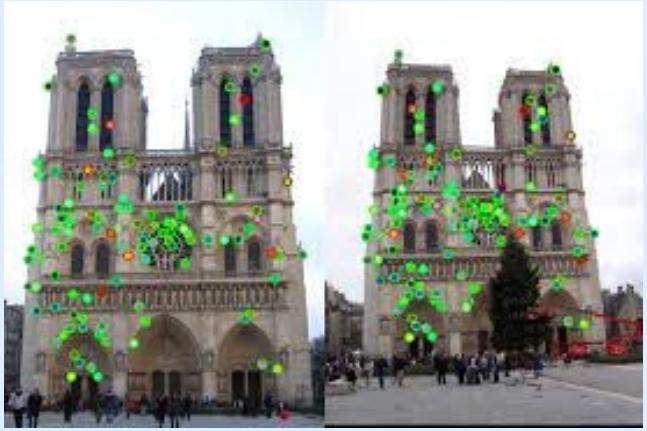
AUGMENTED REALITY

VIRTUAL REALITY

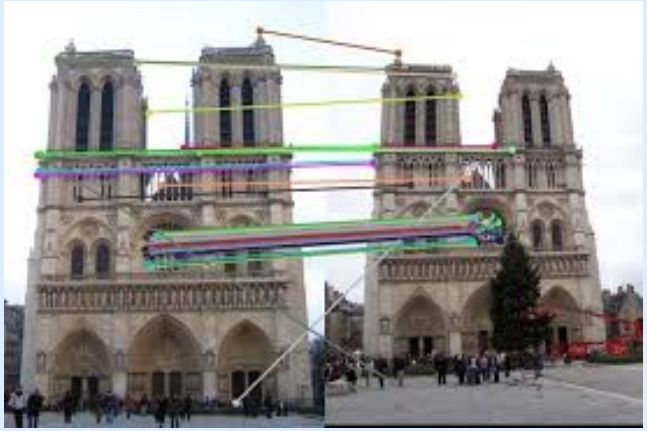
FEATURE MATCHING

Feature Matching or generally image matching, a part of many computer vision application such as image registration, camera calibration and object recognition. It is the task of establishing correspondence between two images of the same scene/object.

A common approach to image matching consists of detecting a set of interest points each associated with image descriptors from image data. Once the features and their descriptors have been extracted from two or more images, the next step is to establish some preliminary feature matches between these images.



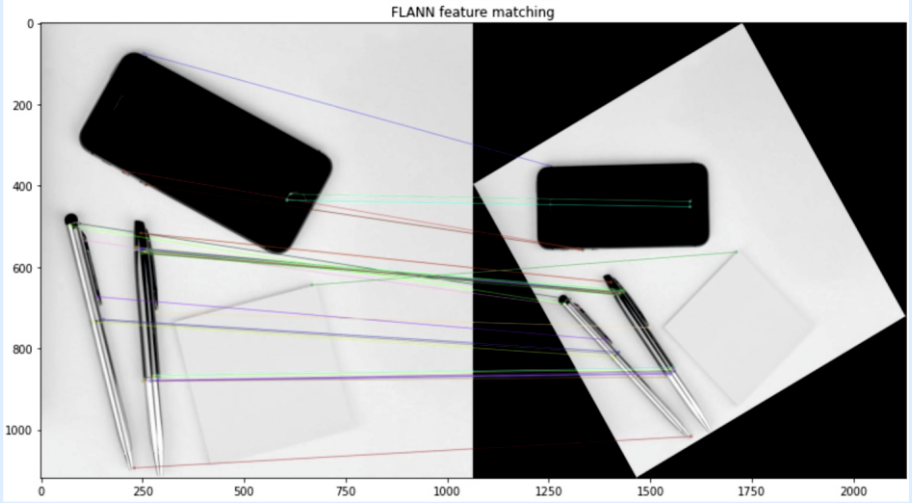
INTEREST POINTS



FEATURES MATCHING

FLANN

Flann stands for Fast Library for Approximate Nearest Neighbors. It is actually used for the system to choose the best algorithm and optimum parameters depending on the dataset. FlannBasedMatcher is also used to match or search for the features of one image to another image. this function is available in the OpenCV library. It uses Nearest Neighbors Approach and usually runs faster than BruteForceMatcher for various datasets. It also works great with large datasets. Here the feature descriptors searched for the nearest numerical values for feature match and its neighbor's distance computation is easy and done fastly. For FlannBasedMatcher we need to pass dictionaries which are index_params and search params.



AUGMENTATION

Augmentation in computer vision refers to the technique of artificially increasing the size of a dataset by applying various transformations to the existing images. These transformations can include rotations, translations, scaling, flipping, cropping, changing brightness and contrast, adding noise, or any other operation that doesn't change the semantic content of the image.

Some of its advantages are :-

- Increased Robustness: By introducing variations in the training data, augmentation helps the model become more robust to real-world conditions such as changes in lighting, viewpoint, scale, and occlusions.
- Generalization: Augmentation prevents the model from overfitting to the training data by exposing it to a wider range of examples.
- Data Efficiency: Augmentation allows you to generate a larger and more diverse training dataset without the need for collecting additional labeled data.
- Regularization: Augmentation serves as a form of regularization by adding noise to the training process. This helps prevent the model from memorizing the training examples and encourages it to learn more robust representations of the data.

HOMOGRAPHY MATRIX

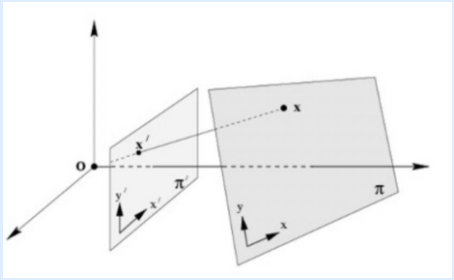
The planar Homography matrix relates the transformation between two planes.

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

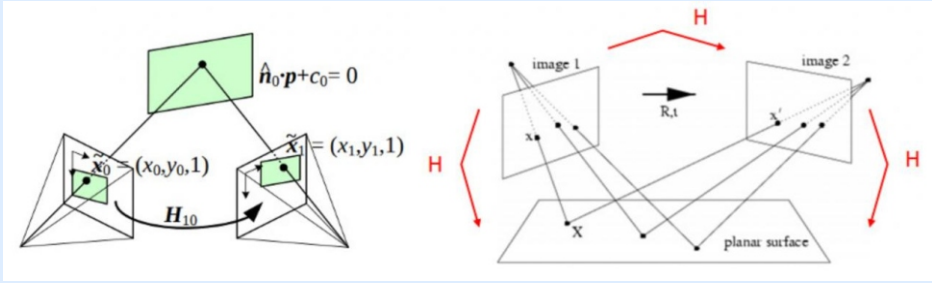
The homography matrix is a 3x3 matrix but with 8 DoF (degrees of freedom) as it is estimated up to a scale.

The following examples show different kinds of transformation but all relate a transformation between two planes.

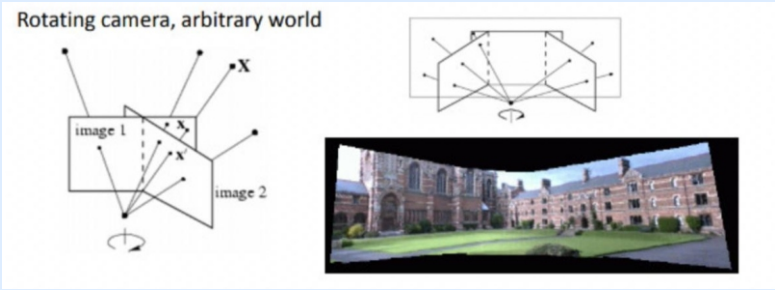
a planar surface and the image plane



a planar surface viewed by two camera positions



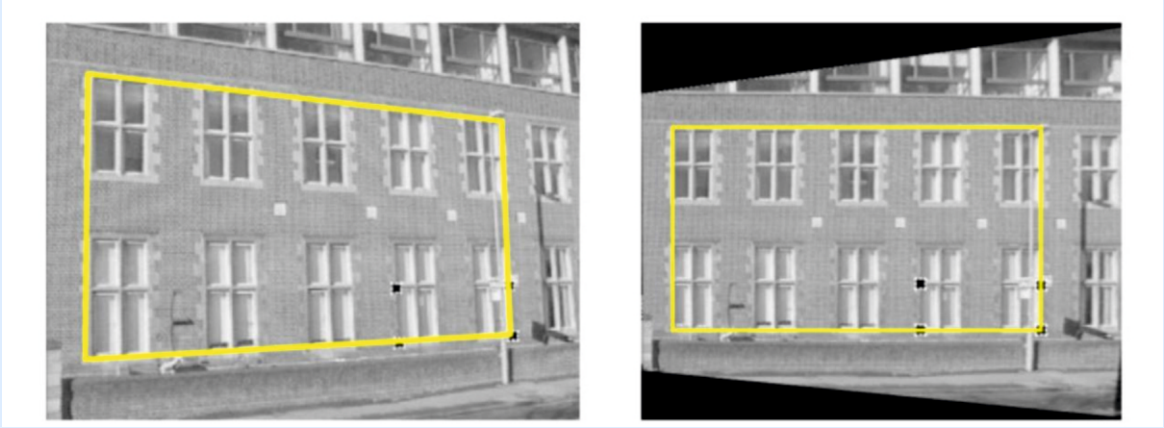
a rotating camera around its axis of projection, equivalent to consider that the points are on a plane at infinity



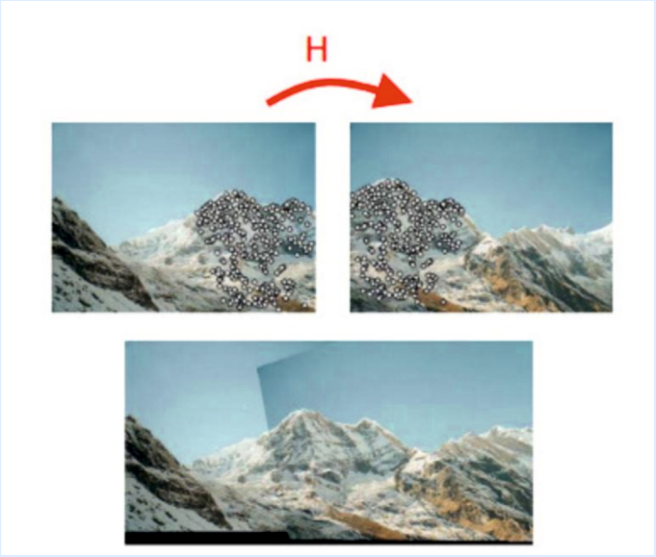
ADVANTAGES OF HOMOGRAPHY

Camera pose estimation from coplanar points for augmented reality with marker for instance

Perspective removal / correction



Panorama stitching



Augmentation

test3.py > ...

```python
import cv2
import numpy as np

MIN_MATCHES = 20
detector = cv2.ORB_create(nfeatures=5000)

FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=100)
flann = cv2.FlannBasedMatcher(index_params,search_params)
def load_input():
    input_image = cv2.imread('physics.jpg')
    augment_image = cv2.imread('iphoneimg.jpg')
    input_image = cv2.resize(input_image, (300,400),interpolation=cv2.INTER_AREA)
    augment_image = cv2.resize(augment_image, (300,400))
    gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    # find the keypoints with ORB
    keypoints, descriptors = detector.detectAndCompute(gray_image, None)
    return gray_image,augment_image,keypoints, descriptors

def compute_matches(descriptors_input, descriptors_output):
    # Match descriptors
    if(len(descriptors_output)!=0 and len(descriptors_input)!=0):
        matches = flann.knnMatch(np.asarray(descriptors_input,np.float32),np.asarray(descriptors_output,np.float32),k=2)
        good = []
        for m,n in matches:
            if m.distance < 0.69*n.distance:
                good.append(m)
        return good
    else:
        return None

if __name__=='__main__':
    #Getting Information form the Input image
    input_image, aug_image, input_keypoints, input_descriptors = load_input()

    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()
    while(ret):
```

test3.py > ...

```python
        ret, frame = cap.read()
        if(len(input_keypoints)<MIN_MATCHES):
            continue
        frame = cv2.resize(frame, (600,450))
        frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        output_keypoints, output_descriptors = detector.detectAndCompute(frame_bw, None)
        matches = compute_matches(input_descriptors, output_descriptors)
        if(matches!=None):
            if(len(matches)>10):
                src_pts = np.float32([ input_keypoints[m.queryIdx].pt for m in matches ]).reshape(-1,1,2)
                dst_pts = np.float32([ output_keypoints[m.trainIdx].pt for m in matches ]).reshape(-1,1,2)

                #Finally find the homography matrix
                M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
                #matchesMask = mask.ravel().tolist()
                pts = np.float32([ [0,0],[0,399],[299,399],[299,0] ]).reshape(-1,1,2)
                dst = cv2.perspectiveTransform(pts,M)
                M_aug = cv2.warpPerspective(aug_image, M, (600,450))

                #getting the frame ready for addition operation with Mask Image
                frameb = cv2.fillConvexPoly(frame,dst.astype(int),0)
                Final = frameb+M_aug

                #output_final = cv2.polylines(frame,[np.int32(dst)],True,255,3, cv2.LINE_AA)
                cv2.imshow('Final Output', Final)
                #cv2.imshow('Finallli', Final)
            else:
                cv2.imshow('Final Output', frame)
        else:
            cv2.imshow('Final Output', frame)
        key = cv2.waitKey(15)
        if(key==27):
            break
```

Feature Detection

test.py > ...

```python
import cv2
import numpy as np

#Getting the Image ready for feature detection
input_image = cv2.imread('physics.jpg')
input_image = cv2.resize(input_image, (400,550),interpolation=cv2.INTER_AREA)
gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
# Initiate ORB object
orb = cv2.ORB_create(nfeatures=1000)

# find the keypoints with ORB
keypoints, descriptors = orb.detectAndCompute(gray_image, None)

# draw only the location of the keypoints without size or
final_keypoints = cv2.drawKeypoints(gray_image, keypoints,input_image,(0,255,0))

cv2.imshow('ORB keypoints', final_keypoints)
cv2.waitKey()
```

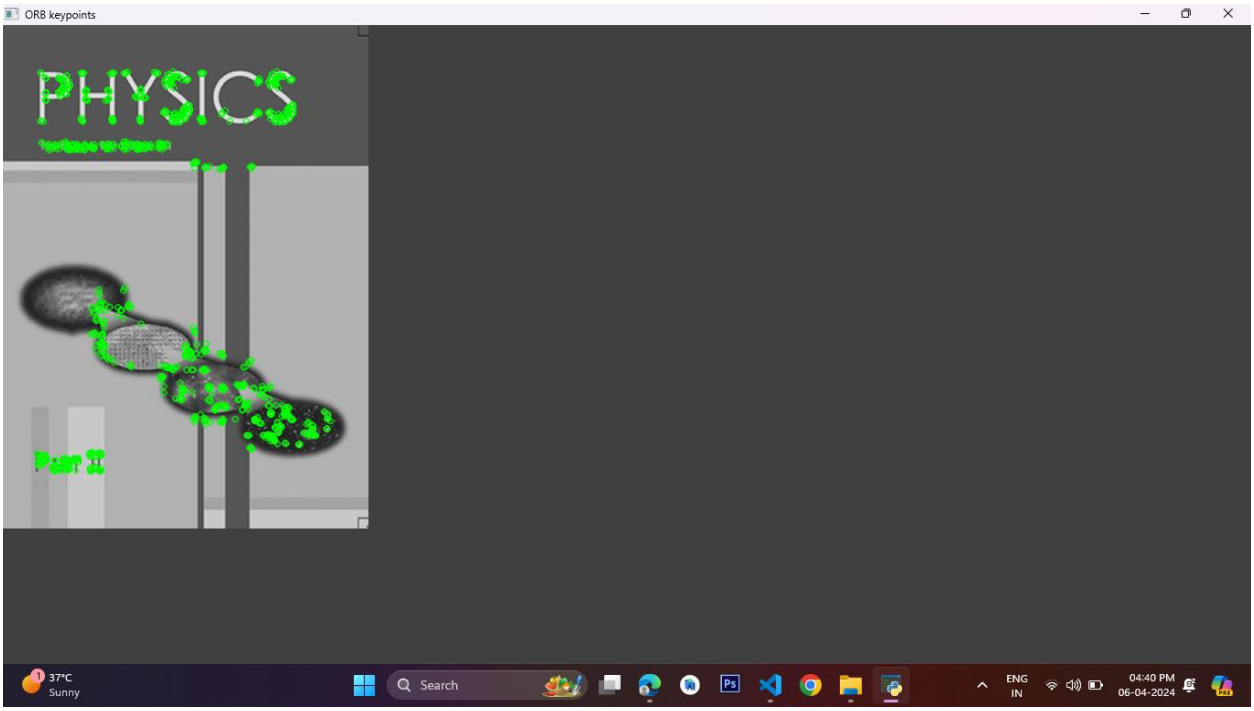
Feature Matching

test2.py > ...

```python
import cv2
import numpy as np

#Initilizing the ORB Feature Detector
MIN_MATCHES = 20
detector = cv2.ORB_create(nfeatures=5000)
#Preparing the FLANN Based matcher
index_params = dict(algorithm = 1, trees=3)
search_params = dict(checks=100)
flann = cv2.FlannBasedMatcher(index_params,search_params)

#Function for Loading input image and Keypoints
def load_input():
    input_image = cv2.imread('physics.jpg')
    input_image = cv2.resize(input_image, (400,550),interpolation=cv2.INTER_AREA)
    gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    # find the keypoints with ORB
    keypoints, descriptors = detector.detectAndCompute(gray_image, None)
    return gray_image,keypoints, descriptors

#Function for Computing Matches between the train and query descriptors
def compute_matches(descriptors_input, descriptors_output):
    if(len(descriptors_output)!=0 and len(descriptors_input)!=0):
        matches = flann.knnMatch(np.asarray(descriptors_input,np.float32),np.asarray(descriptors_output,np.float32),k=2)
        good = []
        for m,n in matches:
            if m.distance < 0.68*n.distance:
                good.append([m])
        return good
    else:
        return None
```

test2.py > load_input

```python
#Main Working Logic
if __name__=='__main__':

    #Getting Information form the Input image
    input_image, input_keypoints, input_descriptors = load_input()

    #Getting camera ready
    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()

    while(ret):
        ret, frame = cap.read()
        #Condition Check for error escaping
        if(len(input_keypoints)<MIN_MATCHES):
            continue
        #Resizing input image for fast computation
        frame = cv2.resize(frame, (700,600))
        frame_bw = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        #Computing and matching teh Keypoints of Input image and query Image
        output_keypoints, output_descriptors = detector.detectAndCompute(frame_bw, None)
        matches = compute_matches(input_descriptors, output_descriptors)

        if(matches!=None):
            output_final = cv2.drawMatchesKnn(input_image,input_keypoints,frame,output_keypoints,matches,None,flags=cv2.DrawMatchesFlags_NOT
            cv2.imshow('Final Output', output_final)
        else:
            cv2.imshow('Final Output', frame)
        key = cv2.waitKey(5)
        if(key==27):
            break
```



Project Working