In [1]:
```
import cv2
import glob
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import fftconvolve,convolve2d
import math
```

In [2]:
```
vidcap = cv2.VideoCapture(r'D:\Bhavya\1-1\CV\Assignment-3\video_1.mp4')
success,image = vidcap.read()
count = 0
while success:
    success,image = vidcap.read()
    if count%30==0 :
        image=cv2.flip(image,0)
        cv2.imwrite(r"D:\Bhavya\1-1\CV\Assignment-3\data\frame%d.jpg" % count, image)
        print('Read a new frame: ', success)
    count += 1
```

```
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
Read a new frame:  True
```

1.Normalized Co-relation

In [3]:
```
def ssd(A,B):
    squares = (A[:,:,:3] - B[:,:,:3]) ** 2
    return math.sqrt(np.sum(squares))
```

In [4]:
```
def norm_data(data):
    mean_data=np.mean(data)
    std_data=np.std(data, ddof=1)
    return (data-mean_data)/(std_data)


def ncc(data0, data1):
    return (1.0/(data0.size-1)) * np.sum(norm_data(data0)*norm_data(data1))
```

In [5]:
```
import cv2
import os
source_dest=r'D:\Bhavya\1-1\CV\Assignment-3\data'
image_list=[]
for file_name in os.listdir(r'D:\Bhavya\1-1\CV\Assignment-3\data'):
    image_list.append(file_name)
```

```
files = []
image_list=['D:/Bhavya/1-1/CV/Assignment-3/data/'+ s for s in image_list]
print(image_list)
images = [cv2.imread(file) for file in image_list]
```

```
['D:/Bhavya/1-1/CV/Assignment-3/data/frame0.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/dat
a/frame120.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame150.jpg', 'D:/Bhavya/1-1/C
V/Assignment-3/data/frame180.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame210.jpg',
'D:/Bhavya/1-1/CV/Assignment-3/data/frame240.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/dat
a/frame270.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame30.jpg', 'D:/Bhavya/1-1/CV/
Assignment-3/data/frame300.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame330.jpg',
'D:/Bhavya/1-1/CV/Assignment-3/data/frame360.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/dat
a/frame390.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame420.jpg', 'D:/Bhavya/1-1/C
V/Assignment-3/data/frame450.jpg', 'D:/Bhavya/1-1/CV/Assignment-3/data/frame60.jpg',
'D:/Bhavya/1-1/CV/Assignment-3/data/frame90.jpg']
```

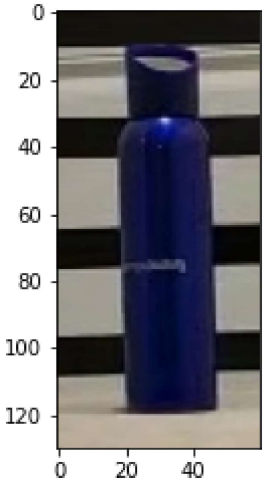
In [6]:
```
plt.imshow(cv2.cvtColor(images[0], cv2.COLOR_BGR2RGB))
print(images[0].shape)
```

```
(640, 352, 3)
```

In [7]:
```
cropped_image = images[5][150:280,120:180]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))

cv2.imwrite("Cropped Image.jpg", cropped_image)
```

Out[7]:
```
True
```



In [8]:
```
d=dict()
d_norm=dict()
for i in range(0,510,20):
    for j in range(0,292,20):
        #print(i,j)
        #print(i+130,j+160)
        d[str(i)+":"+str(i+130),str(j)+":"+str(j+60)]=ssd(cropped_image,images[5][i:i+
```

In [9]:
```
a=min(d.items(), key=lambda x: x[1])
y1,y2=map(int,a[0][0].split(':'))
x1,x2=map(int,a[0][1].split(':'))
```

In [10]:
```
plt.imshow(cv2.cvtColor(images[5][y1:y2,x1:x2], cv2.COLOR_BGR2RGB))
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x25b2d955fd0>
```

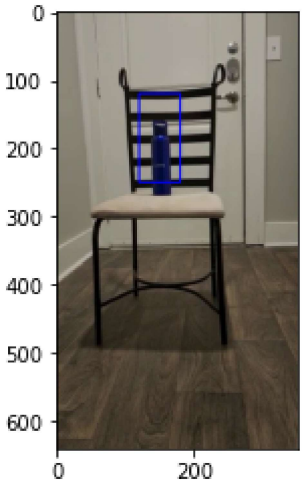
In [11]:
```
color = (255, 0, 0)

# Line thickness of 2 px
thickness = 2

# Using cv2.rectangle() method
# Draw a rectangle with blue line borders of thickness of 2 px
image = cv2.rectangle(images[5], (x1,y1), (x2,y2), color, thickness)
```

In [12]:
```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

Out[12]:
```
<matplotlib.image.AxesImage at 0x25b2d9bed00>
```



2. Motion Tracking Equation

In [13]:
```
Iref=cv2.imread(r'D:\Bhavya\1-1\CV\Assignment-3\data\frame420.jpg',cv2.IMREAD_GRAYSCAL
Inext=cv2.imread(r'D:\Bhavya\1-1\CV\Assignment-3\data\frame450.jpg',cv2.IMREAD_GRAYSCA
Iref=np.array(Iref).astype(np.float32)
Inext=np.array(Inext).astype(np.float32)
kernel_x = np.array([[-1., 1.], [-1., 1.]])*.25
kernel_y = np.array([[-1., -1.], [1., 1.]])*.25
kernel_t = np.array([[1., 1.], [1., 1.]])*.25
Iref = Iref / 255. # normalize pixels
Inext = Inext / 255. # normalize pixels
Ix=cv2.filter2D(Iref,-1,kernel=kernel_x)
Iy=cv2.filter2D(Iref,-1,kernel=kernel_y)
It=cv2.filter2D(Iref,-1,kernel=kernel_t)+cv2.filter2D(Inext,-1,kernel=kernel_x)
Ix,Iy,It=np.array(Ix),np.array(Iy),np.array(It)
```

In [14]:
```
motion_tracking_equation=np.divide(It,np.sqrt(np.square(Ix)+np.square(Iy)))
```

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_14588\2449423964.py:1: RuntimeWarning: divid
e by zero encountered in true_divide
  motion_tracking_equation=np.divide(It,np.sqrt(np.square(Ix)+np.square(Iy)))
C:\Users\Hp\AppData\Local\Temp\ipykernel_14588\2449423964.py:1: RuntimeWarning: inval
id value encountered in true_divide
  motion_tracking_equation=np.divide(It,np.sqrt(np.square(Ix)+np.square(Iy)))
```

In [15]:
```
motion_tracking_equation
```

Out[15]:
```
array([[       inf,        inf,        inf, ..., 259.99902 , 259.49902 ,
        167.99936 ],
       [       inf,        inf,        inf, ..., 259.99802 , 259.49805 ,
        167.99872 ],
       [       inf,        inf,        inf, ..., 162.22449 , 165.06978 ,
        167.49937 ],
       ...,
       [ 19.200003,  16.8     ,  17.650454, ...,  18.00169 ,  47.84756 ,
         48.29907 ],
       [ 11.490486,  10.194123,  10.778754, ...,  15.414932,  33.994488,
        158.39194 ],
       [ 10.416769,  10.036247,  18.593632, ...,  18.001696,  70.20257 ,
        159.80615 ]], dtype=float32)
```

2B.Image Registration

In [16]:
```
def image_registraion(align,ref):

# Open the image files.
    img1_color = align  # Image to be aligned.
    img2_color = ref    # Reference image.

  # Convert to grayscale.
    img1 = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)
    height, width = img2.shape

  # Create ORB detector with 5000 features.
    orb_detector = cv2.ORB_create(5000)

  # Find keypoints and descriptors.
  # The first arg is the image, second arg is the mask
  #  (which is not required in this case).
    kp1, d1 = orb_detector.detectAndCompute(img1, None)
    kp2, d2 = orb_detector.detectAndCompute(img2, None)

  # Match features between the two images.
  # We create a Brute Force matcher with
  # Hamming distance as measurement mode.
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)

  # Match the two sets of descriptors.
    matches = matcher.match(d1, d2)

  # Sort matches on the basis of their Hamming distance.
    matches=sorted(matches,key = lambda x: x.distance)

  # Take the top 90 % matches forward.
    matches = matches[:int(len(matches)*0.9)]
    no_of_matches = len(matches)

  # Define empty matrices of shape no_of_matches * 2.
    p1 = np.zeros((no_of_matches, 2))
    p2 = np.zeros((no_of_matches, 2))
    for i in range(len(matches)):
        p1[i, :] = kp1[matches[i].queryIdx].pt
        p2[i, :] = kp2[matches[i].trainIdx].pt

  # Find the homography matrix.
    homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)

  # Use this matrix to transform the
  # colored image wrt the reference image.
    transformed_img = cv2.warpPerspective(img1_color,
                      homography, (width, height))
    return transformed_img
```

In [17]:
```
# Creating a VideoCapture object to read the video
capture = cv2.VideoCapture(r"D:\Bhavya\1-1\CV\Assignment-3\video_1.mp4")
ret, frame = capture.read()

# Loop until the end of the video
```

```
while (capture.isOpened()):
    # Capture frame-by-frame
    prev=frame
    ret, frame = capture.read()
    disp=image_registraion(frame,prev)


    disp = cv2.resize(disp, (540, 380), fx = 0, fy = 0,
                        interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    frame=cv2.flip(frame,-1)
    cv2.imshow('Frame', disp)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
capture.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [17], in <cell line: 6>()
      8 prev=frame
      9 ret, frame = capture.read()
---> 10 disp=image_registraion(frame,prev)
     13 disp = cv2.resize(disp, (540, 380), fx = 0, fy = 0,
     14                     interpolation = cv2.INTER_CUBIC)
     16 # Display the resulting frame

Input In [16], in image_registraion(align, ref)
     24   matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)
     26 # Match the two sets of descriptors.
---> 27   matches = matcher.match(d1, d2)
     29 # Sort matches on the basis of their Hamming distance.
     30   matches=sorted(matches,key = lambda x: x.distance)

KeyboardInterrupt:
```

3. Optical Flow Vectors

In [18]:

```
def optical_flow(n):
    cap = cv2.VideoCapture(r"D:\Bhavya\1-1\CV\Assignment-3\video_1.mp4")
    feature_params = dict( maxCorners = 100,
                           qualityLevel = 0.3,
                           minDistance = 7,
                           blockSize = 7 )
    # Parameters for lucas kanade optical flow
    lk_params = dict( winSize  = (15, 15),
                      maxLevel = 2,
                      criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0
    # Create some random colors
    color = np.random.randint(0, 255, (100, 3))
    # Take first frame and find corners in it
    ret, old_frame = cap.read()
    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
```

```
    p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
    # Create a mask image for drawing purposes
    mask = np.zeros_like(old_frame)

    count=0
    while(1):
        ret, frame = cap.read()
        count+=1
        if not ret:
            print('!!! No frames grabbed!!!')
            break
        if(count%n==0):
            frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            # calculate optical flow
            p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **
            # Select good points
            if p1 is not None:
                good_new = p1[st==1]
                good_old = p0[st==1]
            # draw the tracks
            for i, (new, old) in enumerate(zip(good_new, good_old)):
                a, b = new.ravel()
                c, d = old.ravel()
                mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].to
                frame = cv2.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
            img = cv2.add(frame, mask)
            cv2.imshow('frame', img)
            if cv2.waitKey(1)==ord('q'):
                break
            # Now update the previous frame and previous points
            old_gray = frame_gray.copy()
            p0 = good_new.reshape(-1, 1, 2)
    cv2.destroyAllWindows()
```

First Frame as reference

In [19]:
```
optical_flow(1)
```

```
!!! No frames grabbed!!!
```

11th reference Frame

In [20]:
```
optical_flow(11)
```

```
!!! No frames grabbed!!!
```

31st reference Frame

In [21]:
```
optical_flow(31)
```

```
!!! No frames grabbed!!!
```

4.Feature based object detection

In [22]:
```
import cv2 as cv
import math as math
```

```
import numpy as np
org=cv.imread(r'D:\Bhavya\1-1\CV\Assignment-3\img.jpg')
ref=cv.imread(r'D:\Bhavya\1-1\CV\Assignment-3\img_cropped.jpg')
org = cv.cvtColor(org, cv.COLOR_BGR2GRAY)
ref=cv.cvtColor(ref, cv.COLOR_BGR2GRAY)
print(ref.shape)
print(org.shape)
```

```
(217, 212)
(404, 416)
```

In [23]:
```
def sum_squ_dis(cropped,orginal):
    squares = (cropped[:,:] - orginal[:,:]) ** 2
    return math.sqrt(np.sum(squares))
```

In [24]:
```
d=dict()
d_norm=dict()
for i in range(0,187,20):
    for j in range(0,204,20):
        d[str(i)+":"+str(i+217),str(j)+":"+str(j+212)]=sum_squ_dis(ref,org[i:i+217,j:
        #d_norm[str(i)+":"+str(i+100),str(j)+":"+str(j+100)]=ncc(norm_data(cropped_imc
```

In [25]:
```
a=sorted(d.items(), key=lambda x: x[1])
```

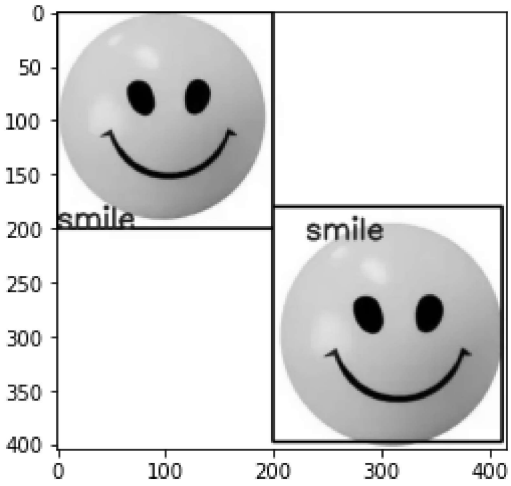
In [26]:
```
#print(d.items())
a=min(d.items(), key=lambda x: x[1])
y1,y2=map(int,a[0][0].split(':'))
x1,x2=map(int,a[0][1].split(':'))
```

In [27]:
```
color = (0, 255, 0)
thickness = 2
print(x1,y1)
image = cv.rectangle(org, (x1,y1), (x2,y2), color, thickness)
image=cv.rectangle(org, (0,200), (200,0), color, thickness)
cv.putText(image, 'smile', (x1+30, y1+30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12),
cv.putText(image, 'smile', (0, 200), cv.FONT_HERSHEY_SIMPLEX, 0.9, (36,255,12), 2)
```

Out[27]:
```
200 180
array([[  0,   0,   0, ..., 254, 254, 254],
       [  0,   0,   0, ..., 254, 254, 254],
       [  0,   0, 254, ..., 254, 254, 254],
       ...,
       [254, 254, 254, ..., 254, 254, 254],
       [254, 254, 254, ..., 254, 254, 254],
       [254, 254, 254, ..., 254, 254, 254]], dtype=uint8)
```

In [28]:
```
import matplotlib.pyplot as plt
plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x25b2da0d460>
```



5. Multiple Face Tracking Application

In [83]:
```
import os
import time

import imutils
detectorPaths = {
    "face": "face.xml"
}

print("Loading haar cascades...")
detectors = dict()

for (name, path) in detectorPaths.items():
    detectors[name] = cv2.CascadeClassifier(path)

print("[INFO] starting video stream...")
vs = cv2.VideoCapture(0)

while True:
    _,frame = vs.read()
    frame = imutils.resize(frame, width=500)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faceRects = detectors["face"].detectMultiScale(
        gray, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE)

    for (fX, fY, fW, fH) in faceRects:
        faceROI = gray[fY:fY + fH, fX:fX + fW]
        cv2.rectangle(frame, (fX, fY), (fX + fW, fY + fH),
                      (0, 255, 0), 2)
    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) == ord("q"):
        break

cv2.destroyAllWindows()
```

```
Loading haar cascades...
[INFO] starting video stream...
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [83], in <cell line: 19>()
     20 frame = imutils.resize(frame, width=500)
     21 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
---> 23 faceRects = detectors["face"].detectMultiScale(
     24     gray, scaleFactor=1.05, minNeighbors=5, minSize=(30, 30),
     25     flags=cv2.CASCADE_SCALE_IMAGE)
     27 for (fX, fY, fW, fH) in faceRects:
     28     faceROI = gray[fY:fY + fH, fX:fX + fW]

KeyboardInterrupt:
```

6.disparity based depth estimation in stereo-vision theory.

In [ ]:

```
import cv2

# function to display the coordinates of
# of the points clicked on the image
def click_event(event, x, y, flags, params):

    # checking for left mouse clicks
    if event == cv2.EVENT_LBUTTONDOWN:

        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img, str(x) + ',' +
                    str(y), (x,y), font,
                    1, (255, 0, 0), 2)
        cv2.imshow('image', img)

    # checking for right mouse clicks
    if event==cv2.EVENT_RBUTTONDOWN:

        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)

        # displaying the coordinates
        # on the image window
        font = cv2.FONT_HERSHEY_SIMPLEX
        b = img[y, x, 0]
        g = img[y, x, 1]
        r = img[y, x, 2]
        cv2.putText(img, str(b) + ',' +
                    str(g) + ',' + str(r),
                    (x,y), font, 1,
                    (255, 255, 0), 2)
        cv2.imshow('image', img)

# driver function
if __name__=="__main__":
```

```
    # reading the image
    img = cv2.imread(r'D:\Bhavya\1-1\CV\Assignment-3\image1.jpg', 1)

    # displaying the image
    cv2.imshow('image', img)

    # setting mouse handler for the image
    # and calling the click_event() function
    cv2.setMouseCallback('image', click_event)

    # wait for a key to be pressed to exit
    cv2.waitKey(0)

    # close the window
    cv2.destroyAllWindows()
```

```
1072   454
```

In [29]:
```
ul,vl=1067,453
ur,vr=783,461
b=172.3 # distance between left and right cameras
f=1329.30324 #focallength
z=(b*f)/(ul-ur) #distance of object
print('The distance is '+str(z)+'mm')
```

```
The distance is 806.4751699014084mm
```

7.Lucas-Kanade algorithm for motion tracking

In [30]:
```
cap = cv2.VideoCapture(0)
feature_params = dict( maxCorners = 100,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15, 15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.0
# Create some random colors
color = np.random.randint(0, 255, (100, 3))
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
while(1):
    ret, frame = cap.read()
    if not ret:
        print('No frames grabbed!')
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params
    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
    # draw the tracks
```

```
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), 2
        frame = cv2.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
    img = cv2.add(frame, mask)
    cv2.imshow('frame', img)
    if cv2.waitKey(1)==ord('q'):
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [30], in <cell line: 19>()
     17 mask = np.zeros_like(old_frame)
     18 while(1):
---> 19     ret, frame = cap.read()
     20     if not ret:
     21         print('No frames grabbed!')

KeyboardInterrupt:
```

In [ ]: