**Date:15/2/24 EXPERIMENT 6**

**Name:Rithika shree**

**21R231**

**Stereo Matching and Point Cloud Reconstruction**

**Aim:**

To perform feature detection and matching, disparity map creation and point cloud reconstruction using OpenCV.

# **Software/ Packages Used:**

1. Pycharm IDE
2. Libraries used:
   * NumPy
   * opencv-python
   * matplotlib
   * scipy

# **Programs:**

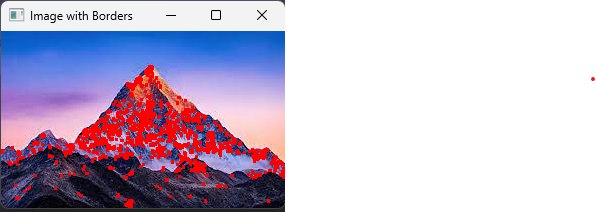
# **Feature Detection and Matching:**

**HARRIS CORNER DETECTION METHOD (IMAGE)**

**INPUT**

# Python program to illustrate   
# corner detection with   
# Harris Corner Detection Method   
  
# organizing imports   
import cv2   
import numpy as np   
  
# path to input image specified and   
# image is loaded with imread command   
image = cv2.imread('mountain.jpg')  
  
# convert the input image into   
# grayscale color space   
operatedImage = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)   
  
# modify the data type   
# setting to 32-bit floating point   
operatedImage = np.float32(operatedImage)   
  
# apply the cv2.cornerHarris method   
# to detect the corners with appropriate   
# values as input parameters   
dest = cv2.cornerHarris(operatedImage, 2, 5, 0.07)   
  
# Results are marked through the dilated corners   
dest = cv2.dilate(dest, None)   
  
# Reverting back to the original image,   
# with optimal threshold value   
image[dest > 0.01 \* dest.max()]=[0, 0, 255]   
  
# the window showing output image with corners   
cv2.imshow('Image with Borders', image)   
  
# De-allocate any associated memory usage   
if cv2.waitKey(0) & 0xff == 27:   
 cv2.destroyAllWindows()

**OUTPUT**

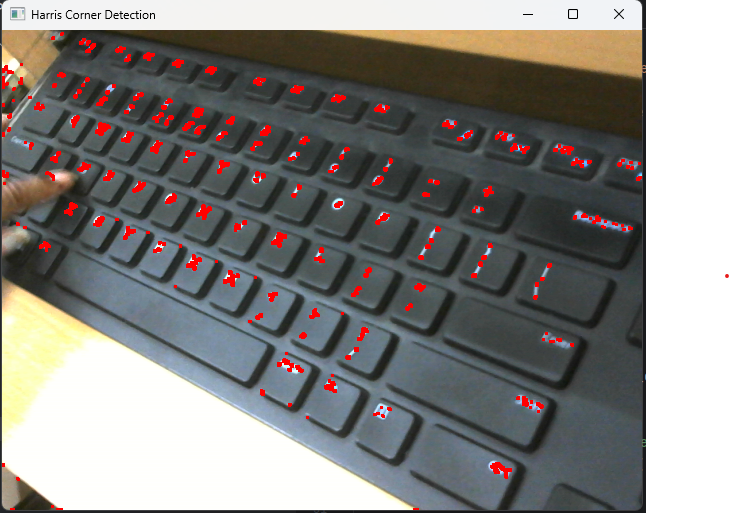


**HARRIS CORNER DETECTION METHOD (VIDEO)**

**INPUT**

import numpy as np  
import cv2  
  
# Function to detect Harris corners  
def detect\_corners(frame):  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 gray = np.float32(gray)  
 dst = cv2.cornerHarris(gray, 2, 3, 0.04)  
  
 # Dilate to mark the corners  
 dst = cv2.dilate(dst, None)  
  
 # Threshold for an optimal value, it may vary depending on the image  
 frame[dst > 0.01 \* dst.max()] = [0, 0, 255] # Mark detected corners in red  
 return frame  
  
# Capture video from camera  
cap = cv2.VideoCapture(0)  
  
while(True):  
 # Capture frame-by-frame  
 ret, frame = cap.read()  
  
 # Detect corners  
 corners\_detected\_frame = detect\_corners(frame)  
  
 # Display the resulting frame  
 cv2.imshow('Harris Corner Detection', corners\_detected\_frame)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the capture  
cap.release()  
cv2.destroyAllWindows()

**OUTPUT**

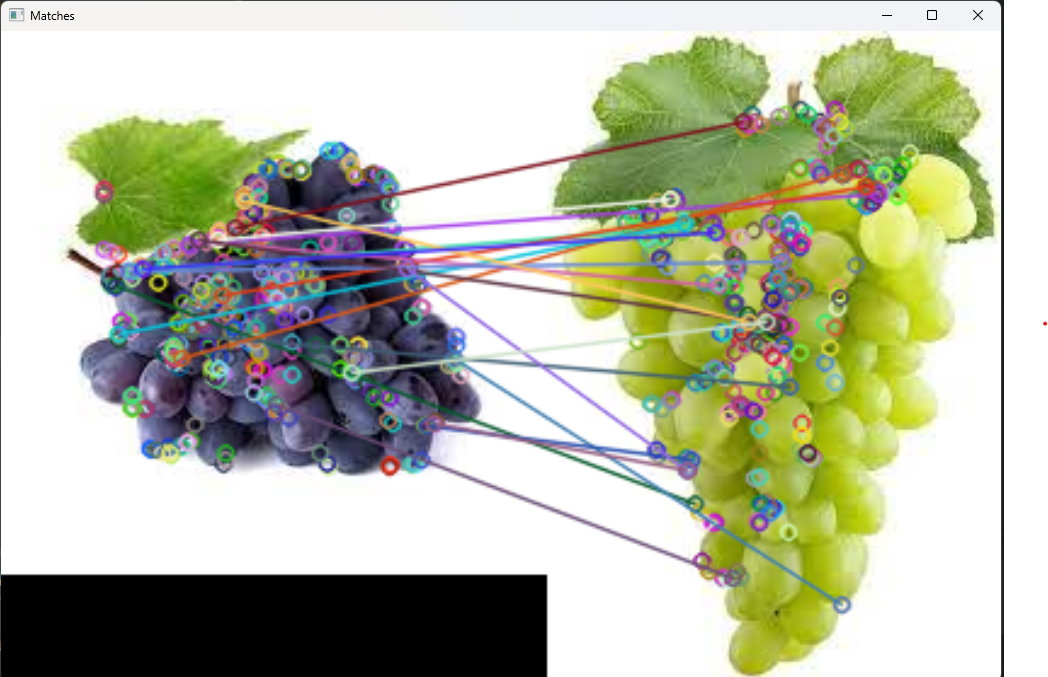


**ORB (image)2 images**

**INPUT**

import numpy as np  
import cv2  
  
# Read the query image as query\_img  
# and train image This query image  
# is what you need to find in train image  
# Save it in the same directory  
# with the name image.jpg  
query\_img = cv2.imread('grapes.jpg')  
train\_img = cv2.imread('grapes2.jpg')  
  
# Convert it to grayscale  
query\_img\_bw = cv2.cvtColor(query\_img, cv2.COLOR\_BGR2GRAY)  
train\_img\_bw = cv2.cvtColor(train\_img, cv2.COLOR\_BGR2GRAY)  
  
# Initialize the ORB detector algorithm  
orb = cv2.ORB\_create()  
  
# Now detect the keypoints and compute  
# the descriptors for the query image  
# and train image  
queryKeypoints, queryDescriptors = orb.detectAndCompute(query\_img\_bw, None)  
trainKeypoints, trainDescriptors = orb.detectAndCompute(train\_img\_bw, None)  
  
# Initialize the Matcher for matching  
# the keypoints and then match the  
# keypoints  
matcher = cv2.BFMatcher()  
matches = matcher.match(queryDescriptors, trainDescriptors)  
  
# draw the matches to the final image  
# containing both the images the drawMatches()  
# function takes both images and keypoints  
# and outputs the matched query image with  
# its train image  
final\_img = cv2.drawMatches(query\_img, queryKeypoints,  
 train\_img, trainKeypoints, matches[:20], None)  
  
final\_img = cv2.resize(final\_img, (1000, 650))  
  
# Show the final image  
cv2.imshow("Matches", final\_img)  
cv2.waitKey(3000)

**OUTPUT**



**ORB (video)1 image 1 video**

**INPUT**

import cv2  
import numpy as np  
  
# Create an ORB object  
orb = cv2.ORB\_create()  
  
# Create a BFMatcher object  
bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)  
  
# Read the input image  
input\_image = cv2.imread('phone.jpg', cv2.IMREAD\_GRAYSCALE)  
input\_kp, input\_des = orb.detectAndCompute(input\_image, None)  
  
# Open the camera  
cap = cv2.VideoCapture(0, cv2.CAP\_DSHOW)  
  
while True:  
 # Read the next frame  
 ret, frame = cap.read()  
  
 # Check if the frame is valid  
 if not ret:  
 print("Error: Failed to capture frame from the camera.")  
 break  
  
 # Convert frame to grayscale  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Detect and compute keypoints and descriptors for the current frame  
 kp, des = orb.detectAndCompute(gray, None)  
  
 # Check if descriptors are empty  
 if des is None or len(des) == 0:  
 print("Error: Descriptors not computed for the current frame.")  
 continue  
  
 # Check if input descriptors are empty  
 if input\_des is None or len(input\_des) == 0:  
 print("Error: Descriptors not computed for the input image.")  
 continue  
  
 # Match descriptors  
 matches = bf.match(input\_des, des)  
  
 # Sort matches by distance  
 matches = sorted(matches, key=lambda x: x.distance)  
  
 # Draw matches  
 img\_matches = cv2.drawMatches(input\_image, input\_kp, frame, kp, matches[:10], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)  
  
 # Display the frame  
 cv2.imshow('Frame', img\_matches)  
  
 # Check for exit  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the camera and close all windows  
cap.release()  
cv2.destroyAllWindows()

**OUTPUT**



**2 videos**

import cv2  
import numpy as np  
  
# Initialize the first camera (change index as needed)  
cap1 = cv2.VideoCapture(0)  
  
# Check if the camera opened successfully  
if not cap1.isOpened():  
 print("Error: Could not open camera 1")  
 exit()  
  
# Initialize the second camera (change index as needed)  
cap2 = cv2.VideoCapture(1)  
  
# Check if the camera opened successfully  
if not cap2.isOpened():  
 print("Error: Could not open camera 2")  
 cap1.release()  
 exit()  
  
# Initialize ORB detector  
orb = cv2.ORB\_create()  
  
while True:  
 # Capture frames from both cameras  
 ret1, frame1 = cap1.read()  
 ret2, frame2 = cap2.read()  
  
 # Convert frames to grayscale  
 gray1 = cv2.cvtColor(frame1, cv2.COLOR\_BGR2GRAY)  
 gray2 = cv2.cvtColor(frame2, cv2.COLOR\_BGR2GRAY)  
  
 # Find keypoints and descriptors  
 kp1, des1 = orb.detectAndCompute(gray1, None)  
 kp2, des2 = orb.detectAndCompute(gray2, None)  
  
 # Initialize brute force matcher  
 bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)  
  
 # Match descriptors  
 matches = bf.match(des1, des2)  
  
 # Sort matches based on distance  
 matches = sorted(matches, key=lambda x: x.distance)  
  
 # Draw matches  
 img\_matches = cv2.drawMatches(frame1, kp1, frame2, kp2, matches[:10], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)  
  
 # Display result  
 cv2.imshow('Matches', img\_matches)  
  
 # Break loop if 'q' is pressed  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the cameras and close OpenCV windows  
cap1.release()  
cap2.release()  
cv2.destroyAllWindows()

**1 image**

# Importing the libraries  
import cv2  
# Reading the image and converting into B/W  
image = cv2.imread('snow.jpg')  
gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
# Applying the function  
orb = cv2.ORB\_create(nfeatures=2000)  
kp, des = orb.detectAndCompute(gray\_image, None)  
# Drawing the keypoints  
kp\_image = cv2.drawKeypoints(image, kp, None, color=(0, 255, 0), flags=0)  
cv2.imshow('ORB', kp\_image)  
cv2.waitKey()

# **Disparity Map:**

**INPUT**

import numpy as np  
import cv2  
  
# Load left and right images  
left\_image = cv2.imread('ak.jpg', cv2.IMREAD\_GRAYSCALE)  
right\_image = cv2.imread('ak1.jpg', cv2.IMREAD\_GRAYSCALE)  
  
# Resize images to have the same dimensions  
height, width = left\_image.shape  
right\_image = cv2.resize(right\_image, (width, height))  
  
# Create StereoBM object  
stereo = cv2.StereoBM\_create(numDisparities=16, blockSize=15)  
  
# Compute disparity map  
disparity\_map = stereo.compute(left\_image, right\_image)  
  
# Normalize the disparity map for visualization  
disparity\_map\_visual = cv2.normalize(disparity\_map, None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)  
  
# Display the disparity map  
cv2.imshow('Disparity Map', disparity\_map\_visual)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

# **INPUT**

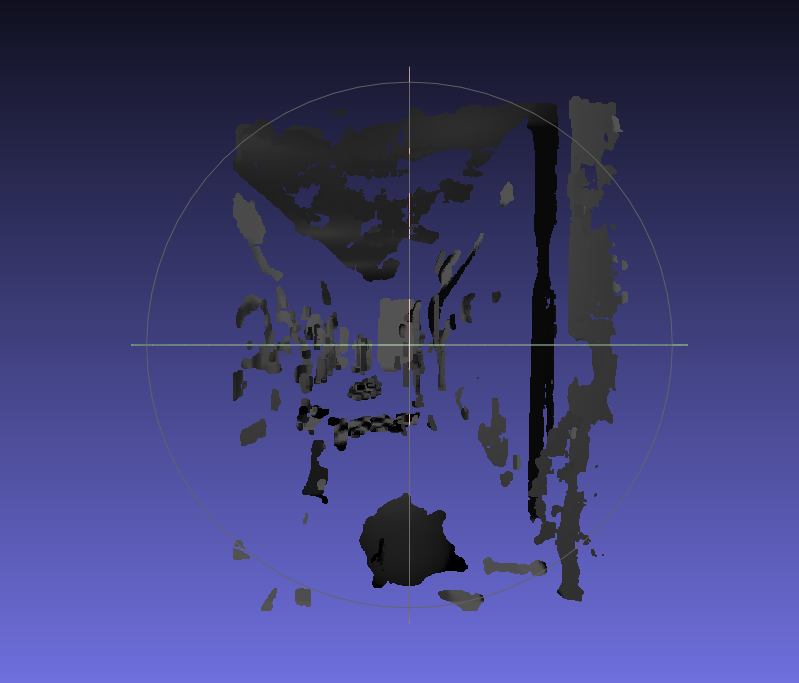
# 

**OUTPUT**

# 

**DISPARITY IMAGE (MESH LAB)**

from \_\_future\_\_ import print\_function  
import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
  
ply\_header = '''ply  
format ascii 1.0  
element vertex %(vert\_num)d  
property float x  
property float y  
property float z  
property uchar red  
property uchar green  
property uchar blue  
end\_header  
'''  
  
def write\_ply(fn, verts, colors):  
 verts = verts.reshape(-1, 3)  
 colors = colors.reshape(-1, 3)  
 verts = np.hstack([verts, colors])  
 with open(fn, 'wb') as f:  
 f.write((ply\_header % dict(vert\_num=len(verts))).encode('utf-8'))  
 np.savetxt(f, verts, fmt='%f %f %f %d %d %d ')  
  
def main():  
 print('loading images...')  
 frameWidth = 480  
 frameHeight = 480  
 imgL = cv.imread('ambush\_5\_right.jpg', 0)  
 imgR = cv.imread('ambush\_5\_left.jpg', 0)  
 imgL = cv.resize(imgL, (frameWidth, frameHeight))  
 imgR = cv.resize(imgR, (frameWidth, frameHeight))  
  
 window\_size = 3  
 min\_disp = 16  
 num\_disp = 112 - min\_disp  
 stereo = cv.StereoSGBM\_create(minDisparity=min\_disp, numDisparities=num\_disp,  
 blockSize=16, P1=8 \* 3 \* window\_size \*\* 2,  
 P2=32 \* 3 \* window\_size \*\* 2, disp12MaxDiff=1,  
 uniquenessRatio=10, speckleWindowSize=100, speckleRange=3)  
 print('computing disparity...')  
 disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0  
  
 print('generating 3d point cloud...')  
 h, w = imgL.shape[:2]  
 f = 0.8 \* w # guess for focal length  
 Q = np.float32([[1, 0, 0, -0.5 \* w], [0, -1, 0, 0.5 \* h], [0, 0, 0, -f], [0, 0, 1, 0]])  
  
 points = cv.reprojectImageTo3D(disp, Q)  
 colors = cv.cvtColor(imgL, cv.COLOR\_GRAY2RGB)  
 mask = disp > disp.min()  
 out\_points = points[mask]  
 out\_colors = colors[mask]  
 out\_fn = 'out.ply'  
 write\_ply(out\_fn, out\_points, out\_colors)  
 print('%s saved' % out\_fn)  
  
 plt.figure(figsize=(20, 10))  
 plt.subplot(1, 3, 1)  
 plt.title("left image")  
 plt.imshow(imgL, cmap='gray')  
 plt.axis('off')  
 plt.subplot(1, 3, 2)  
 plt.title("right image")  
 plt.imshow(imgR, cmap='gray')  
 plt.axis('off')  
 plt.subplot(1, 3, 3)  
 plt.title("disparity image")  
 plt.imshow((disp - min\_disp) / num\_disp, cmap='gray')  
 plt.axis('off')  
 plt.show()  
  
 print('Done')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()  
 cv.destroyAllWindows()



**IMAGE CLASSIFIER**

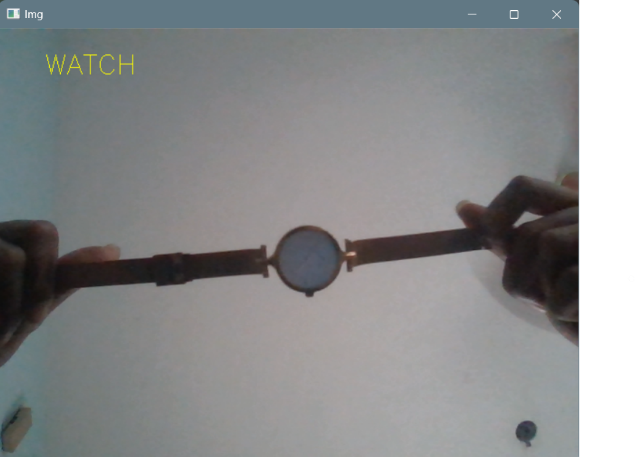
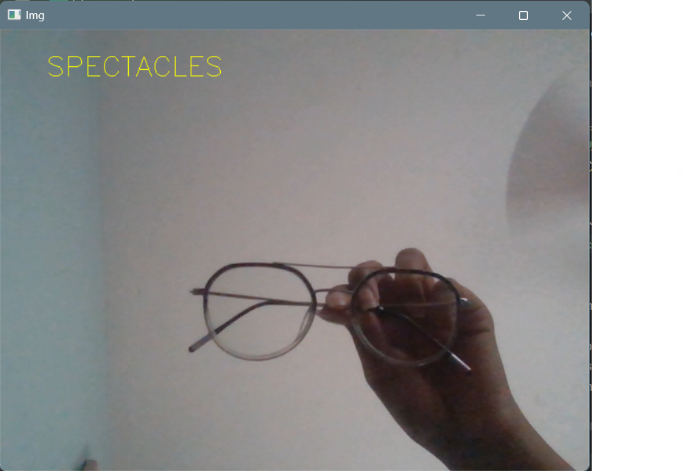
**INPUT**

import cv2 as cv  
import numpy as np  
import os  
#pre-trained images  
path = 'C:\studies\sem 6\AI & vision systems lab\IMAGE'  
orb= cv.ORB\_create(nfeatures=1000)  
images=[]  
classNames=[]  
ml= os.listdir(path)  
print("Total classes detected: ", len(ml))  
  
for cl in ml:  
 img= cv.imread(f'{path}/{cl}',0)  
  
 images.append(img)  
 classNames.append(os.path.splitext(cl)[0])  
print(classNames)  
#descriptors  
def fd(images):  
 desList=[]  
 for img in images:  
 kp,des = orb.detectAndCompute(img,None)  
 desList.append(des)  
 return desList  
def fin(img,desList, thres=4):  
 kp2,des2 = orb.detectAndCompute(img,None)  
 bf = cv.BFMatcher()  
 matchList=[]  
 finval= -1  
 try:  
 for des in desList:  
 matches = bf.knnMatch(des,des2,k=2)  
 good=[]  
 for m,n in matches:  
 if m.distance<0.75 \* n.distance:  
 good.append([m])  
 matchList.append(len(good))  
 except:  
 pass  
 if len(matchList)!=0:  
 if max(matchList)>thres:  
 finval = matchList.index(max(matchList))  
 return finval  
desList = fd(images)  
print(len(desList))  
cap = cv.VideoCapture(0)  
while True:  
 success, img2 = cap.read()  
 imgo= img2.copy()  
 img2= cv.cvtColor(img2, cv.COLOR\_BGR2GRAY)  
 ia= fin(img2,desList)  
 if ia!=-1:  
 cv.putText(imgo,classNames[ia],(50,50),cv.FONT\_HERSHEY\_SIMPLEX,1,(0,255,255),1)  
  
  
 cv. imshow('Img',imgo)  
 cv.waitKey(1)

**INPUT IMAGE**



**OUTPUT**

****

# 

**Capturing video**

import cv2

# Initialize the camera

cap = cv2.VideoCapture(0)

# Check if the camera opened successfully

if not cap.isOpened():

print("Error: Could not open camera")

exit()

while True:

# Capture frame-by-frame

ret, frame = cap.read()

# Display the resulting frame

cv2.imshow('Frame', frame)

# Break the loop if 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the camera and close OpenCV windows

cap.release()

cv2.destroyAllWindows()

**Post Lab Questions:**

1. **What is Correspondence problem? Which approach is used for finding corresponding pixels?**

* The correspondence problem arises in computer vision and image processing when attempting to find matching points or features between different images or frames of a video.
* One common approach for finding corresponding pixels or features is through feature matching techniques. Feature descriptors such as SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and ORB (Oriented FAST and Rotated BRIEF) are commonly used for this purpose. These descriptors encode information about key points or features in an image, which can then be matched across different images to establish correspondences.

1. **How 2D image is converted to 3D?**

* Converting a 2D image to 3D typically involves depth estimation or reconstruction techniques.
* One common method is stereo vision, which uses multiple images of the same scene taken from different viewpoints (usually two), along with geometric principles, to estimate the depth information of objects in the scene.
* Other techniques include structured light methods, where a known pattern is projected onto the scene and its deformation is used to infer depth, and time-of-flight methods, where the time it takes for light to travel from a source to the scene and back is used to estimate depth.

1. **How ORB differs from FAST?**

* ORB (Oriented FAST and Rotated BRIEF) and FAST (Features from Accelerated Segment Test) are both feature detection and description algorithms used in computer vision.
* FAST is a corner detection algorithm that identifies keypoints in an image based on the intensity difference around a pixel.
* ORB builds upon FAST and BRIEF (Binary Robust Independent Elementary Features). ORB adds orientation assignment and a more robust feature descriptor to FAST-BRIEF, making it more suitable for real-time applications.
* One significant difference is that ORB computes oriented keypoints, which makes it more robust to image rotations compared to FAST.

1. **Compare ORB, Sift and Surf**

**ORB (Oriented FAST and Rotated BRIEF):**

Fast computation and relatively good performance.

Provides orientation assignment to keypoints.

Uses binary descriptors, making it efficient for real-time applications.

**SIFT (Scale-Invariant Feature Transform):**

Provides scale-invariant keypoints.

Robust to changes in scale, rotation, and illumination.

Uses a floating-point descriptor, which may be slower than binary descriptors like ORB.

**SURF (Speeded-Up Robust Features):**

Designed to be faster than SIFT while maintaining similar performance.

Uses integral images to speed up computations.

Provides scale and rotation-invariant features.

# 

# **Result:**

Thus the Feature Detection and Matching, Disparity Map Creation and Point Cloud Reconstruction were done using OpenCV.