A black and grey logo with a book and a graduation cap

AI-generated content may be incorrect.

ARUNACHALA HI-TECH ENGINEERING COLLEGE

**NAAN MUDHALVAN PROJECT**

A logo for a company

AI-generated content may be incorrect.

**Motion Tracking in Drone Footage Using FAST and ORB**

**Submitted by:**

Rejin M

Rahul R

Reno Ebinizer Roy

Reno Linse

Rohan L.R.

INDEX

* What is OpenCV…………………………………………………
* FAST…………………………………………………………………
* Feature detection using FAST in images…………………
* Feature detection using FAST in videos………………….
* ORB…………………………………………………………………
* Feature detection using ORB in images…………………
* Feature detection using ORB in videos………………….
* Motion tracking using FAST and ORB…………………….

**What is OpenCV**

OpenCV is a python library used to implement **Computer Vision**. Computer Vision allows a computer to monitor and identify objects individually.

* Image and Video Processing: Read, write, transform, and analyse images and video.
* Object Detection and Recognition: Face detection, pedestrian detection, and more.
* Camera Calibration: Useful in robotics and AR/VR.
* Machine Learning Support: Includes a basic ML module with classifiers like SVM, KNN, decision trees.
* Cross-Platform: Works on Windows, Linux, macOS, Android, and iOS.
* Bindings in Multiple Languages: Primarily C++, but also available for Python, Java, and more.

**Common Use Cases:**

1. Facial recognition systems
2. Augmented reality
3. Gesture recognition
4. Robotics and autonomous vehicles
5. Medical image analysis

**FAST**

* Used for feature detection
* Faster for real time application
* Robust

**WORKING OF FAST:**

1. Set a point of interest (p) and intensity(ip)
2. Choose threshold.
3. Choose 16 points around p if they are either all brighter(ip+t) or all darker(ip-t), p is a corner.
4. Initial filtering of bad candidates is done by selecting 4 pixels which are 90 degrees apart. Initially we check the 1st and the 9th pixels. If they are darker or brighter then we move on to the next 2 pixels which are 90 degrees apart, which are 5 and 13. At least 3 should pass to do the full check.

A close-up of a crossword puzzle

AI-generated content may be incorrect.

**Issues of FAST:**

>>Doesn’t reject many points for n<12

>> Choice of point is not optimal because it depends on the order of the checks and variation of corners and types

>> Results of initial filtering are thrown away

>> Multiple adjacent features are detected

The first three issues are addressed using a machine learning approach.

1. Get training data with points that are corners and not corners.
2. For each point, run the FAST algorithm to compute the intensities of the 16 points and split into 3 classes (Darker, similar, brighter).
3. Train decision tree to determine best way to split the data in the tree based on the minimizing entropy (to have less disorder/have pure class).
4. Use decision tree to have a faster detection of corners.

**Feature detection using FAST in images:**

**Source code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

# Read the image

img=cv2.imread("C:\\temerario011.jpg")

# Initiate FAST object

fast = cv2.FastFeatureDetector\_create()

# Detect keypoints

keypoints = fast.detect(img, None)

# Draw keypoints on the image

img\_with\_keypoints = cv2.drawKeypoints(img, keypoints, None, color=(255, 0, 0))

# Display the results

cv2.imshow('FAST Keypoints', img\_with\_keypoints)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Print FAST parameters

print("Threshold: ", fast.getThreshold())

print("NonmaxSuppression: ", fast.getNonmaxSuppression())

print("Total Keypoints De tected: ", len(keypoints))

**Input:**



**Output:**



**Feature detection using FAST in videos:**

**Source code:**

import cv2

import numpy as np

# Load the drone footage

video\_path = "C:\\Users\\rinku\\Downloads\\Untitled video - Made with Clipchamp (1).mp4"

cap = cv2.VideoCapture(video\_path)

# Check if video opened successfully

if not cap.isOpened():

print("Error: Could not open video.")

exit()

# Initialize FAST detector

fast = cv2.FastFeatureDetector\_create()

# Parameters for lucas-kanade optical flow

lk\_params = dict(winSize=(15, 15),

maxLevel=2,

criteria=(cv2.TERM\_CRITERIA\_EPS | cv2.TERM\_CRITERIA\_COUNT, 10, 0.03))

# Read the first frame

ret, old\_frame = cap.read()

if not ret:

print("Error: Could not read the first frame.")

exit()

old\_gray = cv2.cvtColor(old\_frame, cv2.COLOR\_BGR2GRAY)

# Detect keypoints using FAST

keypoints = fast.detect(old\_gray, None)

p0 = np.array([kp.pt for kp in keypoints], dtype=np.float32).reshape(-1, 1, 2)

# Create a mask image for drawing purposes

mask = np.zeros\_like(old\_frame)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

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

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)), (0, 255, 0), 2)

frame = cv2.circle(frame, (int(a), int(b)), 5, (0, 0, 255), -1)

img = cv2.add(frame, mask)

# Display the result

cv2.imshow('Motion Tracking', img)

# Break on 'q' key press

if cv2.waitKey(30) & 0xFF == ord('q'):

break

# Update previous frame and points

old\_gray = frame\_gray.copy()

p0 = good\_new.reshape(-1, 1, 2)

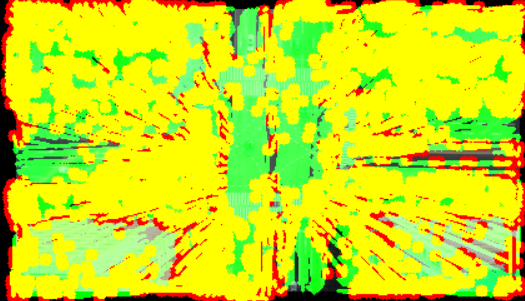
cap.release()

cv2.destroyAllWindows()

**Input:**

****

**Output:**



**ORB**

* ORB – Oriented FAST and Rotated BRIEF.
* It is a feature detection method.
* Mainly used for feature detection.
* It is very efficient, meaning requires only less amount of memory.

**WORKING OF ORB:**

**FAST Detector:**

* Run FAST detector to get a rough selection of features
* Narrow down N key-points by using Harris corners
* Apply scale pyramid to account for scale (steps a and b)

**Orientation by Intensity Centroid:**

* Moments are computed, which can be used to find the centroid and angle

**Rotation of BRIEF:**

* Use BRIEF to obtain the feature descriptor
* Steer BRIEF to find the correct orientation at 12 degree increments using a lookup table
* Run greedy algorithm to find unique features, by finding features with high variance and mean close to 0.5

**Matching:**

* Use LSH (Locality Sensitive Hashing) to approximate neighbour search. Idea is to group similar items in the same buckets with a high probability and same foe dissimilar items
* Hamming distance is also used to compare the distance between two descriptors, specifically between two binary descriptors

**Feature detection using ORB in images:**

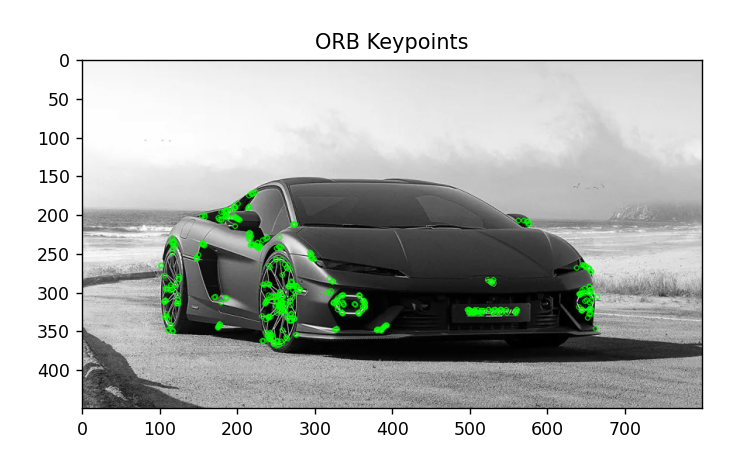
**Source code:**

import cv2  
import matplotlib.pyplot as plt  
  
# Read the input image  
image =cv2.imread("C:\\temerario01.webp",cv2.IMREAD\_GRAYSCALE)  
  
# Initialize ORB detector  
orb = cv2.ORB\_create(nfeatures=500)  
  
# Detect keypoints and compute descriptors  
keypoints, descriptors = orb.detectAndCompute(image, None)  
  
# Draw keypoints on the image  
output\_image = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0), flags=0)  
  
# Display the result  
plt.imshow(output\_image, cmap='gray')  
plt.title("ORB Keypoints")  
plt.show()

**Input:**



**Output:**



**Feature detection using ORB in videos:**

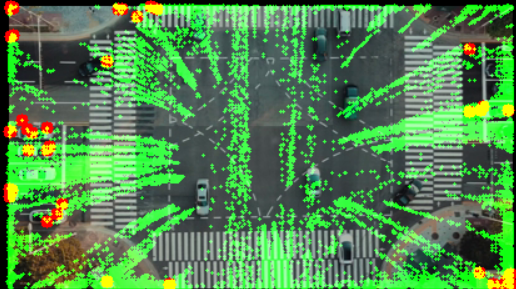
**Source code:**

import cv2  
import numpy as np  
  
# Load the drone footage  
video\_path = "C:\\Users\\rinku\\Downloads\\Untitled video - Made with Clipchamp (1).mp4"  
cap = cv2.VideoCapture(video\_path)  
  
# Check if the video opened successfully  
if not cap.isOpened():  
 print("Error: Could not open video.")  
 exit()  
  
# Initialize the ORB detector  
orb = cv2.ORB\_create(nfeatures=500)  
  
# Initialize the Brute-Force Matcher  
bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)  
  
# Read the first frame  
ret, prev\_frame = cap.read()  
if not ret:  
 print("Error: Could not read the first frame.")  
 exit()  
  
prev\_gray = cv2.cvtColor(prev\_frame, cv2.COLOR\_BGR2GRAY)  
  
# Detect keypoints and compute descriptors for the first frame  
prev\_keypoints, prev\_descriptors = orb.detectAndCompute(prev\_gray, None)  
  
# Create a mask for drawing purposes  
mask = np.zeros\_like(prev\_frame)  
  
while cap.isOpened():  
 ret, curr\_frame = cap.read()  
 if not ret:  
 break  
  
 curr\_gray = cv2.cvtColor(curr\_frame, cv2.COLOR\_BGR2GRAY)  
  
 # Detect keypoints and compute descriptors for the current frame  
 curr\_keypoints, curr\_descriptors = orb.detectAndCompute(curr\_gray, None)  
  
 if curr\_descriptors is not None and prev\_descriptors is not None:  
 # Match descriptors between previous and current frames  
 matches = bf.match(prev\_descriptors, curr\_descriptors)  
 matches = sorted(matches, key=lambda x: x.distance)  
  
 # Draw only the top 50 matches  
 for match in matches[:50]:  
 prev\_idx = match.queryIdx  
 curr\_idx = match.trainIdx  
  
 # Get the keypoint positions  
 prev\_pt = tuple(map(int, prev\_keypoints[prev\_idx].pt))  
 curr\_pt = tuple(map(int, curr\_keypoints[curr\_idx].pt))  
  
 # Draw the motion path  
 mask = cv2.line(mask, curr\_pt, prev\_pt, (0, 255, 0), 2)  
 curr\_frame = cv2.circle(curr\_frame, curr\_pt, 5, (0, 0, 255), -1)  
  
 # Combine the mask with the current frame  
 img = cv2.add(curr\_frame, mask)  
  
 # Display the output  
 cv2.imshow('Motion Tracking', img)  
  
 # Break on 'q' key press  
 if cv2.waitKey(30) & 0xFF == ord('q'):  
 break  
  
 # Update previous frame and keypoints/descriptors  
 prev\_gray = curr\_gray.copy()  
 prev\_keypoints = curr\_keypoints  
 prev\_descriptors = curr\_descriptors  
  
cap.release()  
cv2.destroyAllWindows()

**Input:**

****

**Output:**



**Motion tracking using FAST and ORB**

**M**otion tracking refers to the process of detecting and following the movement of objects over time in a video or a sequence of images. It's a core technique in computer vision, used in areas like surveillance, robotics, video analysis, and augmented reality.

**How Motion Tracking Works:**

1. Detection: Identify moving objects in the frame (often via background subtraction or feature detection).
2. Tracking: Follow those objects across frames (using algorithms that match features or contours).
3. Prediction (optional): Predict where the object will be in the next frame (using filters like Kalman or optical flow)

Motion tracking is useful in many sectors in our daily life. We can implement motion tracking in almost everything. Some of the uses are:

* Traffic control
* Object detection
* Rescue drone operations

**Motion tracking in using FAST and ORB:**

**Source code:**

import cv2  
import numpy as np  
def main(video\_path):  
 video\_path = ("C:\\Users\\rinku\\Downloads\\Untitled video - Made with Clipchamp (1).mp4")  
 cap = cv2.VideoCapture(video\_path)  
 if not cap.isOpened():  
 print("Error: Could not open video.")  
 return  
 fast = cv2.FastFeatureDetector\_create()  
 orb = cv2.ORB\_create()  
 bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)  
 prev\_frame = None  
 prev\_keypoints = None  
 prev\_descriptors = None  
 while True:  
 ret, frame = cap.read()  
 if not ret:  
 break  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 keypoints = fast.detect(gray, None)  
 keypoints, descriptors = orb.compute(gray, keypoints)  
 if descriptors is None:  
 prev\_frame = frame  
 prev\_keypoints = keypoints  
 prev\_descriptors = descriptors  
 continue  
 if prev\_descriptors is not None:  
 matches = bf.match(prev\_descriptors, descriptors)  
 matches = sorted(matches, key=lambda x: x.distance)  
 match\_img = cv2.drawMatches(  
 prev\_frame, prev\_keypoints, frame, keypoints,  
 matches[:50], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS  
 )  
 cv2.imshow("Feature Matches", match\_img)  
 prev\_frame = frame  
 prev\_keypoints = keypoints  
 prev\_descriptors = descriptors  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 cap.release()  
 cv2.destroyAllWindows()  
if \_\_name\_\_ == "\_\_main\_\_":  
 video\_path = "drone\_footage.mp4" # Replace with your video path  
 main(video\_path)

**Input:**

****

**Output:**

