OPENCV AND ITS APPLICATIONS

Data Science II, Fall 2023

Victor Qiu, Alex Fritz, Nilay Patel

01.

INTRODUCTION

An overview of OpenCV, computer vision, and related concepts

Presented by Alex Fritz

PART I BREAKDOWN

1.1
WHAT IS OPENCV?
An overview of OpenCV

1.2
A BRIEF HISTORY
The development of OpenCV

1.3
COMPUTER VISION
Importance and general applications of CV

1.4
OPENCV PROS AND CONS
Benefits and hindrances of OpenCV

1.5
PLATFORMS AND LANGUAGES
Where can you find OpenCV?

1.6
COMMUNITY & RESOURCES
Learn more and engage with other developers

ARTIFICIAL INTELLIGENCE (AI)

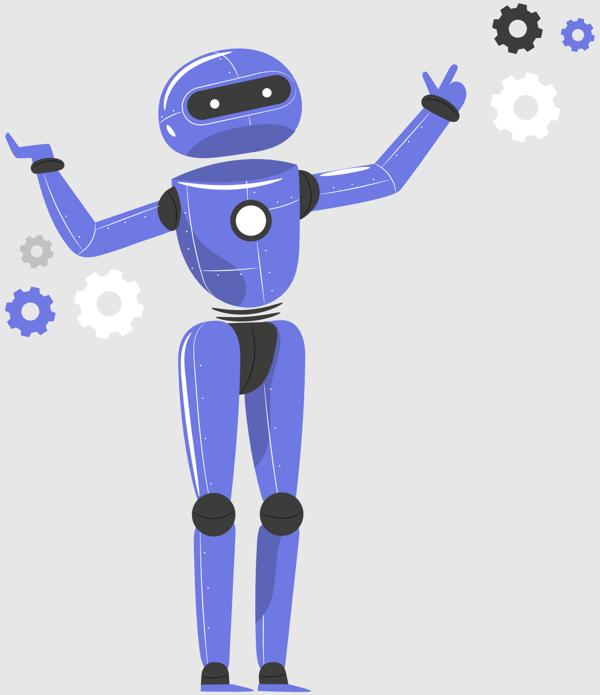
ARTIFICIAL
INTE
(AI)

WHAT IS OPENCV?

(AI)

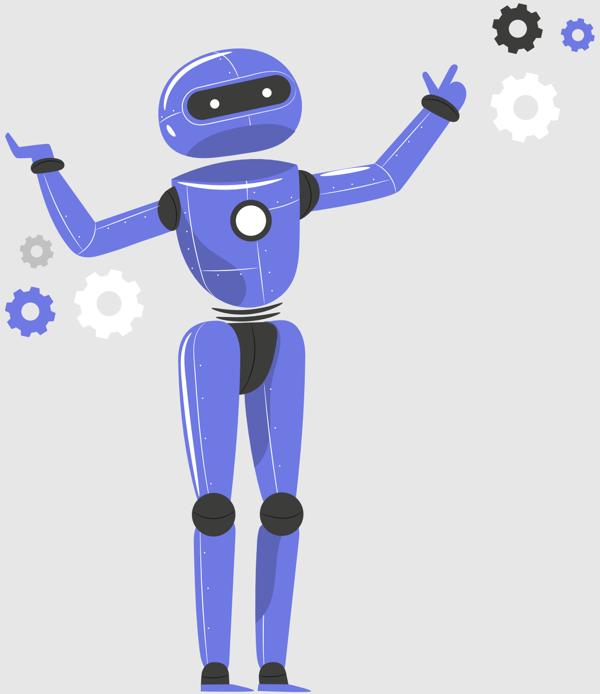
WHAT IS OPENCV?

- Actually short for **Open Computer Vision**, which refers to the project's open source nature.
- One of the most **popular and widely used** libraries for computer vision tasks
- Has **over 2500 optimized algorithms**
- Written in C and C++ for performance, but bindings are available **for many languages** like Python or Java.
- It's **cross-platform** and available for Android and iOS



WHAT IS OPENCV?

- Used for both **classic CV** and **machine learning**
- Has modules for **many specialized tasks** like image processing, video analysis, etc. (discussed later)
- Supports **GPU acceleration** for faster computation of intensive tasks
- Plenty of **extensive documentation** for beginners or experts alike
- **Continuously updated**; currently at version 4.8.0 (July 2, 2023)



ARTIFICIAL

INTE

(AI)

A BRIEF HISTORY

(AI)

A BRIEF HISTORY

- Originally developed by **Gary Bradski** at Intel's research center in California in the 1990s.
- Was released as **open-source** in the year 2000
- Over time, gained support from orgs like **Intel, Willow Garage, and Itseez**
- In 2019, OpenCV transitioned to **non-profit** under OpenCV.org
- Thanks to eager developers, OpenCV has maintained its place at **the forefront of computer vision tech**

GARY BRADSKI

Founder of OpenCV

ARTIFICIAL

INTE

(AI)

COMPUTER VISION

(AI)

IMPORTANCE OF COMPUTER VISION

- Computer vision is a field of **artificial intelligence** dedicated to enabling computers to **understand the visual world**
- ML models are trained on digital images and videos to recognize patterns, objects, and even emotions
- Ultimate goal of CV is to **mimic human vision** and translate pixel data into meaningful info
- Computer vision is a **transformative technology** helping to bridge the gap between human and computer

GENERAL APPLICATIONS OF COMPUTER VISION

TRANSPORTATION

- Self-Driving Vehicles
- Advanced Driver Assistance Systems
  - For things like parking or lane-changing
- Driver fatigue detection

FACIAL RECOGNITION

- Security and surveillance
- Personal technology interfacing
  - I.e., unlocking a smartphone or payment verification
- Location-based access control

ARTIFICIAL INTELLIGENCE (AI)

ARTIFICIAL INTELLIGENCE (AI)

GENERAL APPLICATIONS OF COMPUTER VISION

ENTERTAINMENT

- **Augmented reality**
  - I.e., superimposing digital images on the real world
- **Movie production**
  - Movement-captured CGI
- **Sports broadcasting**
  - Analyzing player movements, ball trajectories, etc.

AGRICULTURE

- **Precision agriculture**
  - Analyzing crop health or soil conditions
- **Disease detection**
  - Plants and animals
- **Livestock monitoring**
  - Provide behavioral insights


OPENCV FINDS ITS WAY INTO ALL OF THESE FIELDS.

ARTIFICIAL
INTE

(AI)

OPENCV PROS AND CONS

(AI)

PROS OF OPENCV

- As mentioned, OpenCV is both **open-source** and **cross-platform**
- **Totally free** of use (thanks to its BSD license)
- **Huge community** of developers and researchers contributing to its growth
- Like stated before, comes with **substantial documentation, tutorials, courses,** etc. for learning
- Users can **expand the library** by integrating it with other libraries or making **custom modules**

CONS OF OPENCV

- Despite documentation, can still come with a learning curve
  - CV is very math-intensive, so a conceptual understanding of modules is often required
- OpenCV is comprehensive, but that also means **very large**
  - Might bloat smaller applications
- Since it sometimes uses external software, there may be **compatibility issues**
  - Not too common
- Cons list is honestly difficult to compile, because despite these things:

OPENCV IS STILL WIDELY RENOWNED AS THE BEST BALANCE OF ACCESSIBILITY AND OPTIMIZATION IN COMPUTER VISION TOOLS

★ ★ ★ ★ ★

ARTIFICIAL INTE

(AI)

PLATFORMS AND LANGUAGES

(AI)

PLATFORMS & LANGUAGES

- We've discussed that OpenCV is **cross-platform**
  - Supports major operating systems like Windows, MacOS, Linux, and Unix
  - Also has some mobile support, like iOS and Android

- OpenCV provides **bindings** for various languages to let people work in their language of choice
  - Python
  - Java
  - Most recently, JavaScript (for web-based apps)

- Can be adapted to work on **embedded systems**
  - Suitable for IoT devices, robotics, and edge computing

- OpenCV's large API allows for integration with **other libraries and frameworks**
  - GUI design
  - Deep learning
  - Database integration

ARTIFICIAL

INTE

(AI)

COMMUNITY & RESOURCES

(AI)

COMMUNITY & RESOURCES

- OpenCV has **contributions from researchers, students, developers**, and others worldwide
  - A *huge* community spanning all sorts of countries and disciplines!
- Platforms like the OpenCV Q&A Forum, StackOverflow, and others have **entire dedicated sections to OpenCV** for troubleshooting and innovation
- OpenCV hosts **periodic challenges** where developers can work to solve real-world problems
- There are **workshops, conferences, and meetups** dedicated to OpenCV
  - Help to maintain the community
- OpenCV even has an **official blog** highlighting new features, success stories, & research breakthroughs

COMMUNITY & RESOURCES

DOCS REGULARLY UPDATED

Covers everything from basic to advanced modules

SO MANY TUTORIALS

Easily found with a quick Google search; there are written books as well!

COURSES EVERYWHERE

Several institutions/platforms have courses specifically on OpenCV

OFFICIAL GITHUB

OpenCV's repository also has tutorials, example projects, & discussions

02.

STRUCTURE & FUNCTIONALITY

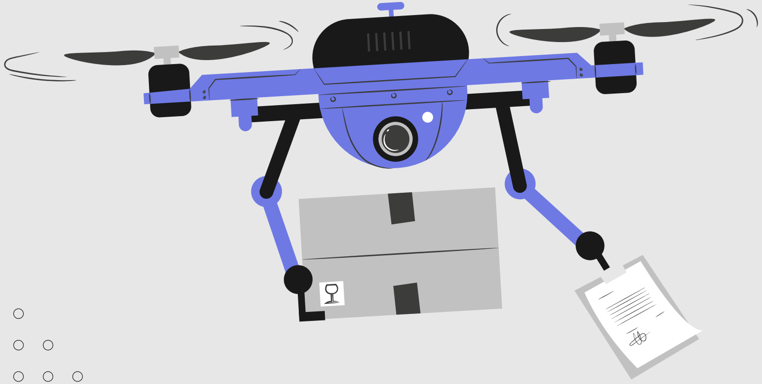
OpenCV is highly modular, and it comes with several main modules!

Presented by Nilay Patel

CORE MODULES

Main modules:

- core. Core functionality
- imgproc. Image Processing
- imgcodecs. Image file reading and writing
- videoio. Video I/O
- highgui. High-level GUI
- video. Video Analysis
- calib3d. Camera Calibration and 3D Reconstruction
- features2d. 2D Features Framework
- objdetect. Object Detection
- dnn. Deep Neural Network module
- ml. Machine Learning
- flann. Clustering and Search in Multi-Dimensional Spaces
- photo. Computational Photography
- stitching. Images stitching
- gapi. Graph API



CORE FUNCTIONALITY

Processing an Image & Storage

How is the image matrix stored in memory?

Grayscale:

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

RGB:

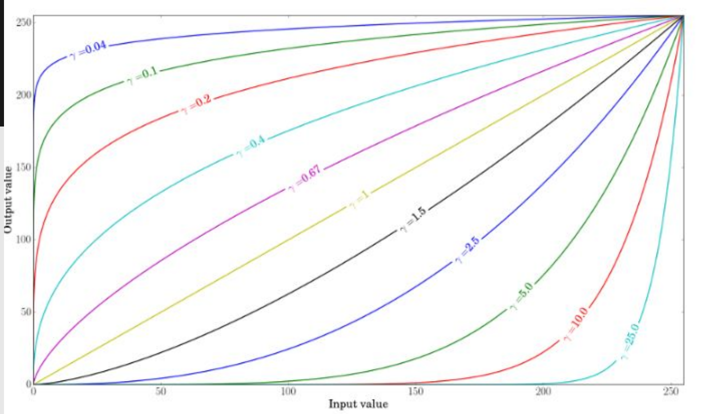
	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

```
Mat& ScanImageAndReduceC(Mat& I, const uchar* const table)
{
 // accept only char type matrices
 CV_Assert(I.depth() == CV_8U);

 int channels = I.channels();

 int nRows = I.rows;
 int nCols = I.cols * channels;

 if (I.isContinuous())
 {
 nCols *= nRows;
 nRows = 1;
 }

 int i,j;
 uchar* p;
 for( i = 0; i < nRows; ++i)
 {
 p = I.ptr<uchar>(i);
 for ( j = 0; j < nCols; ++j)
 {
 p[j] = table[p[j]];
 }
 }
 return I;
}
```

CORE FUNCTIONALITY

- **Changing the contrast and brightness of an image**
- These are linear transformations!
- $g(i,j)=\alpha \cdot f(i,j)+\beta$
  - This represents the transformation on each pixel.
  - α and β are regarded as gain and bias parameters and control contrast and brightness, and i and j represent the pixel location.
- A gamma component can be added:
  - $O=(I/255)^{\gamma}\times 255$

- Key Functions:
  - cv::convertTO():
  - `image.convertTo(new_image, -1, alpha, beta);`
- Changing contrast and brightness is key in being able to properly detect aspects of an image.



Plot for different values of gamma

The following image has been corrected with: $\alpha = 1.3$ and $\beta = 40$.

By Visem (Own work) [CC BY-SA 3.0], via Wikimedia Commons

IMAGE PROCESSING (IMGPROC)

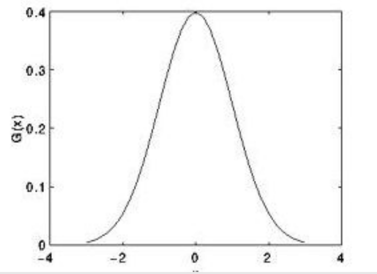
Image Smoothing

OpenCV uses **mathematical calculations** to smooth your image.

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Some ways to filter are:

- **Normalized Box Filter**
  - The simplest of filters; each pixel is the equally weighted mean of its kernel neighbors.
- **Gaussian Filter**
  - Regarded as the most popular filter; each filter modifies each point in the input array using a Gaussian distribution and them sums it to output the new image.
- **Bilateral Filter**
  - This is used when one wants to avoid smoothing edges. This is a more advanced implementation of the Gaussian filter, where it takes in differences in intensity between neighboring pixels.




```
for ( int i = 1; i < MAX_KERNEL_LENGTH; i = i + 2 )
{
medianBlur ( src, dst, i );
if( display_dst( DELAY_BLUR ) != 0 )
{
return 0;
}
}
```

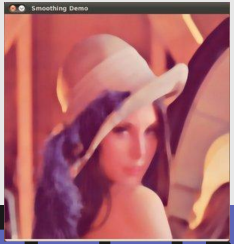


IMAGE PROCESSING (IMGPROC)

Laplace Operator (Border Detection)

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Simply called by using the function **Laplacian()**
Uses the gradient of images to calculate

Takes the first derivative to determine the maximum variance of pixel intensity, and then takes the second derivative to determine where 0 is, which is where the edge is outlined.

Filtering is sometimes necessary to remove misclassified edges.

Laplacian operator

```
Imgproc.Laplacian( src_gray, dst, ddepth, kernel_size, scale, delta, Core.BORDER_DEFAULT );
```

- The arguments are:
  - *src_gray*: The input image.
  - *dst*: Destination (output) image
  - *ddepth*: Depth of the destination image. Since our input is *CV_8U* we define *ddepth* = *CV_16S* to avoid overflow
  - *kernel_size*: The kernel size of the Sobel operator to be applied internally. We use 3 in this example.
  - *scale*, *delta* and *BORDER_DEFAULT*: We leave them as default values.

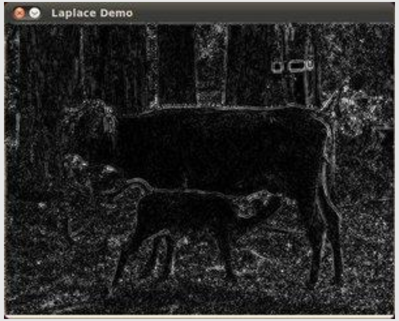
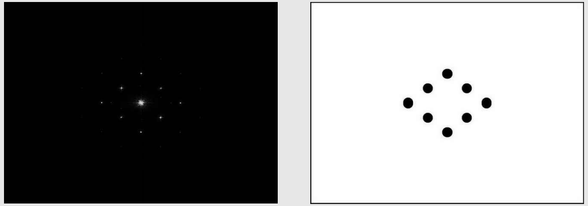


IMAGE PROCESSING (IMGPROC)

Periodic Noise Removing Filter

- Periodic Noise:
  - Refers to patterns that repeat at regular intervals in an image. Often appears as structured interferences like moire patterns.
- Frequency Domain:
  - One approach to periodic noise is to work with the frequency domain. Noise often shapes as distinct peaks in the frequency spectrum.
- OpenCV functions:
  - **dft(): Computes the Discrete Fourier Transform of an image.**
  - **idft(): Computes the Inverse Discrete Fourier Transform.**
  - Functions for creating masks/filters to apply in the frequency domain.
- Common Challenges:
  - Identifying the correct frequencies responsible for the noise.
  - Avoiding excessive blurring or loss of image details when filtering.

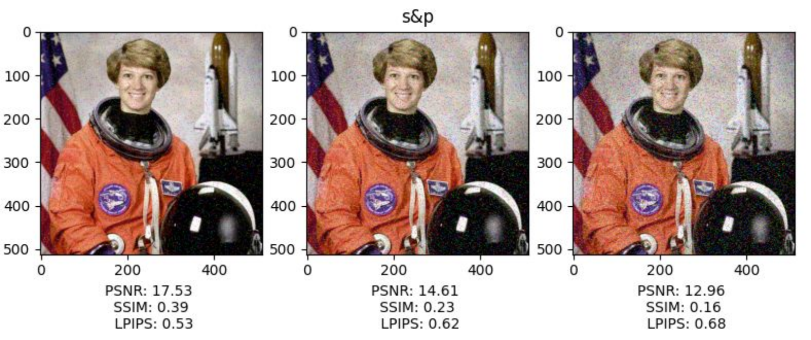


Power spectrum densities

VIDEO ANALYSIS

Video Input with OpenCV and similarity measurement

- Two common measures are used to check image similarity: PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index).
- Image Similarity - PSNR and SSIM:
  - PSNR is a quick method to calculate image similarity.
  - Higher PSNR values (typically between 30 and 50 for video compression) indicate there are more similarity between images
  - SSIM is a more complex method that aligns more with our visual perception. It provides a value between 0 (completely different) and 1 (completely the same) for each image channel.



ARTIFICIAL INTELLIGENCE (AI)

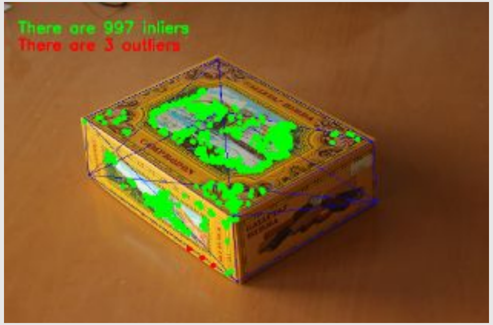
CAMERA CALIBRATION

Real Time pose estimation of a textured object

- Camera pose from n 3D-to-2D points
- General version of the problem requires estimating the six degrees of freedom of the pose and five calibration parameters:
  - Focal length, principal point, aspect ratio and skew.
  - It could be established with a minimum of 6 correspondences.

- OpenCV functions used:
  - findEssentialMat()
  - recoverPose()
  - solvePnP()
  - robustMatch()

- Applications of this can include:
  - Augmented reality
  - Gesture recognition



(AI)

03.

EXAMPLES & APPLICATIONS

How can we use OpenCV?

Presented by Victor Qiu

PART 3 BREAKDOWN

3.1

EXAMPLE 1

Oriented FAST and Rotated BRIEF (ORB Detector)

3.2

EXAMPLE 2

Trajectory and 3D point cloud with triangulation

ARTIFICIAL INTELLIGENCE (AI)

ARTIFICIAL

INTE

(AI)

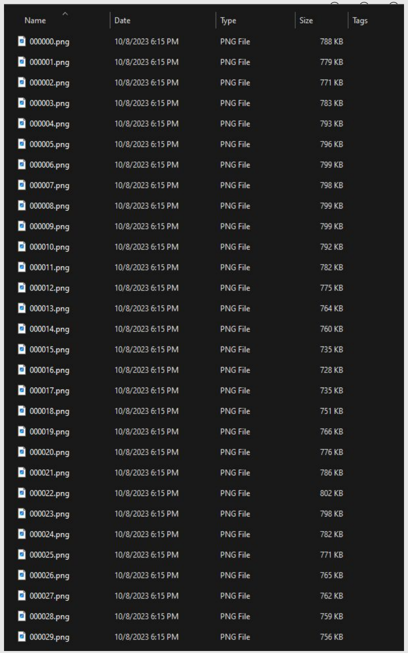
EXAMPLE 1

(AI)

<<<< EXAMPLE 1 >>>>

DETECT CORNERS WITH ORB DETECTOR

- Given 200 video frames, calculate keypoints in each frame and create a video showing the keypoints



Name	Date	Type	Size	Tags
000000.png	10/8/2023 6:15 PM	PNG File	788 KB	
000001.png	10/8/2023 6:15 PM	PNG File	779 KB	
000002.png	10/8/2023 6:15 PM	PNG File	771 KB	
000003.png	10/8/2023 6:15 PM	PNG File	783 KB	
000004.png	10/8/2023 6:15 PM	PNG File	793 KB	
000005.png	10/8/2023 6:15 PM	PNG File	796 KB	
000006.png	10/8/2023 6:15 PM	PNG File	799 KB	
000007.png	10/8/2023 6:15 PM	PNG File	798 KB	
000008.png	10/8/2023 6:15 PM	PNG File	799 KB	
000009.png	10/8/2023 6:15 PM	PNG File	799 KB	
000010.png	10/8/2023 6:15 PM	PNG File	792 KB	
000011.png	10/8/2023 6:15 PM	PNG File	782 KB	
000012.png	10/8/2023 6:15 PM	PNG File	775 KB	
000013.png	10/8/2023 6:15 PM	PNG File	764 KB	
000014.png	10/8/2023 6:15 PM	PNG File	760 KB	
000015.png	10/8/2023 6:15 PM	PNG File	735 KB	
000016.png	10/8/2023 6:15 PM	PNG File	728 KB	
000017.png	10/8/2023 6:15 PM	PNG File	735 KB	
000018.png	10/8/2023 6:15 PM	PNG File	751 KB	
000019.png	10/8/2023 6:15 PM	PNG File	766 KB	
000020.png	10/8/2023 6:15 PM	PNG File	776 KB	
000021.png	10/8/2023 6:15 PM	PNG File	786 KB	
000022.png	10/8/2023 6:15 PM	PNG File	802 KB	
000023.png	10/8/2023 6:15 PM	PNG File	798 KB	
000024.png	10/8/2023 6:15 PM	PNG File	782 KB	
000025.png	10/8/2023 6:15 PM	PNG File	771 KB	
000026.png	10/8/2023 6:15 PM	PNG File	765 KB	
000027.png	10/8/2023 6:15 PM	PNG File	762 KB	
000028.png	10/8/2023 6:15 PM	PNG File	759 KB	
000029.png	10/8/2023 6:15 PM	PNG File	756 KB	

EXAMPLE 1

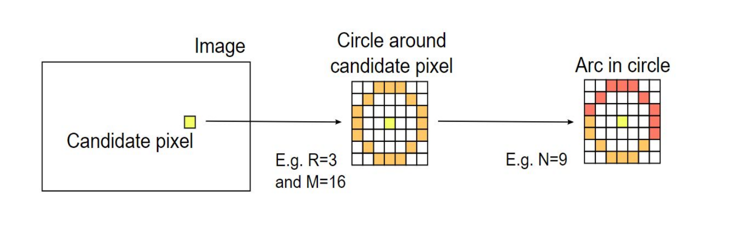
ORB DETECTOR

- Brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper ORB: An efficient alternative to SIFT or SURF in 2011
- **ORB** detector is an algorithm that calculates keypoints and descriptors of an image
- ORB uses a combination of the FAST corner detector and BRIEF descriptors to calculate the keypoints and descriptors of the image

EXAMPLE 1

FAST DETECTOR

1. Convert images into grayscale
2. Selects a pixel as a potential corner pixel
3. Compares the intensity of the pixel value to those around it
4. Determines the state of the pixel by comparing its values with at least N neighboring pixels



$$S_{p\to x} = \begin{cases} \text{brighter} & I_p + t \geq I_{p\to x} \\ \text{darker} & I_{p\to x} \leq I_p - t \\ \text{similar} & I_p - t < I_{p\to x} < I_p + t \end{cases}$$

$$I_{p\to x} = B(R, p)$$

EXAMPLE 1

BRIEF DESCRIPTOR

- Used for describing keypoints and comparing keypoints between different images
- After a keypoint is detected in an image, compare the intensity of the selected pixel with the neighbors around it. Assign 1 for greater and 0 if not to generate a fixed length binary “descriptor”
- Use Hamming distance to match keypoints from different images together. Keypoints with small Hamming distance (similar descriptors) are matched together
- Useful because it’s scale and rotation invariant

<<<< EXAMPLE 1 >>>>

USING ORB DETECTOR

```
img1 = cv2.imread(fileName)
img2 = cv2.imread(fileName2)

img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

keypoints_1, descriptors_1 = orb.detectAndCompute(img1_gray,None)
keypoints_2, descriptors_2 = orb.detectAndCompute(img2_gray,None)

gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# draw points on img1 and img2
img1 = cv2.drawKeypoints(gray1,keypoints_1,img1_gray)
img2 = cv2.drawKeypoints(gray2,keypoints_2,img2_gray)

# bruteforce matching
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
```

ARTIFICIAL

INTE

(AI)

EXAMPLE 2

(AI)

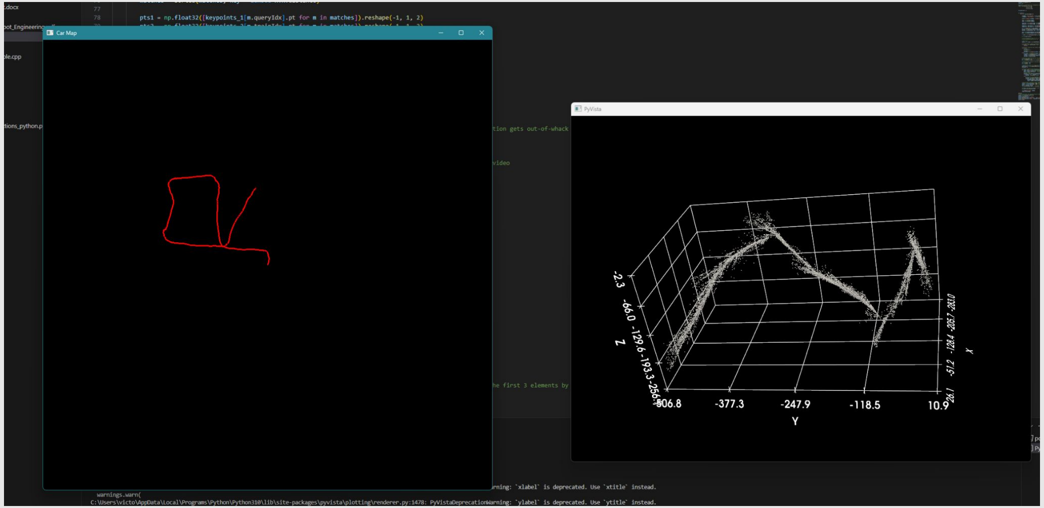
EXAMPLE 2

FIND THE TRAJECTORY MAP AND THE 3D POINT CLOUD GIVEN A LIST OF VIDEO FRAMES

- Given 1100 video frames, map out the trajectory of the car and plot the 3D point cloud of the pixels

EXAMPLE 2

FIND THE TRAJECTORY MAP AND THE 3D POINT CLOUD GIVEN A LIST OF VIDEO FRAMES



<<<< EXAMPLE 2 >>>>

CONVERTING PIXELS FROM 2D TO 3D

- First, need to know the intrinsic parameters of the camera (given) - focal length f_x and f_y, camera center c_x and c_y, and skew factor
- Use the intrinsic parameter matrix and the given 2D points to solve for the original 3D point
- P * [X, Y, Z, 1] = [x1, y1, w1]
- We know P (intrinsic parameter) and x1 and y1 (2D coordinate), solve for [X, Y, Z, 1]

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

<<<< EXAMPLE 2 >>>>

CONVERTING PIXELS FROM 2D TO 3D

```python
# Applies the process of triangulation to compute the 3D coordinates of a point in the scene
point_4d_hom = cv2.triangulatePoints(P1, P2, np.float32(pts1), np.float32(pts2))
Points3D = []

# Process to perform post processing on the 3D points obtained above.
for index, point in enumerate(point_4d_hom.T):
    point = point[:4]/point[-1]  #Converting homogeneous 3D points to non-homogeneous by dividing by the first 3 elements by the fourth
    point = point.T.dot(revX).T  # Reverse X direction
    # Filter points
    distance = math.sqrt(math.pow(point[0], 2) + math.pow(point[1], 2) + math.pow(point[2],2))
    if distance > 75 or distance < 5 or point[2] > 0:
        Point3D = np.array([0, 0, 0])
    else:
        # Calculates the non-homogeneous coordinates of the 3D point obtained through triangulation (point) in the world coordinate system by applying the current rotation and translation stored in currentRt.
        Point3D = currentRt.dot(point.reshape(4, 1))[:3]
        if pts2[index][0][1] < img2.shape[0] and pts2[index][0][0] < img2.shape[1]:
            Points3D.append(Point3D.flatten())
            color = img2[int(pts2[index][0][1]), int(pts2[index][0][0]), :]
```

THANKS!

We hope this presentation was able to give you a much deeper understanding of OpenCV and its applications!