OpenCV

Gabriella Johnson, Kyle Reinholt

History

- Created out of a Intel Research Initiative [1]
- Alpha release in 1999 [1]
- Official release in 2006 [1]

OpenCV was originally a project at Intel, where they were looking to advance CPU intensive applications [1].
OpenCV was created to make computer vision infrastructure universally available [1]

What is OpenCV, Computer Vision?

OpenCV is much like the Facade Pattern we learned about in OOAD this year.

It is a collection of objects that **hide** all of the extremely **complex details** inside of computer vision tasks.

If you are a person with vision, you don't have to apply much of your processing power to recognize objects, faces, or colors in your everyday lives.
Computer Vision utilizes concepts from the human visual system to achieve tasks like detecting where an edge of an object starts and the edge ends. This may seem like a simple task, but a lot of programming goes into "attempting" to solve these problems.

Motivations

Advance vision research by providing not only open but also optimized code for basic vision infrastructure. **No more reinventing the wheel** [6].

Disseminate vision knowledge by providing a **common infrastructure** that developers could build on, so that code would be more readily **readable and transferable** [6].

It should be mentioned that source [2] is the original book written about OpenCV, which is why it appears frequently. This project was implemented using OOP because it is much easier to maintain and create new functionality. The library has more than 2500 optimized algorithms [10], could you imagine a procedure based library with all these functions?

Purpose

The purpose of OpenCV is to **provide high end computer vision algorithms** to users and developers (its a community of 40,000+) for computational visual processing needs.

Before OpenCV, there was not a state-of-the-art, open source library to essentially **"plug and play" CV**.

If you have ever had to implement your own CV algorithms, you would really appreciate all of the hard work people have put into producing a well supported/documented code base like OpenCV.

Well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, employ the library [9]. This is huge. Intel started opencv as a means for people to start working toward a common goal.

OpenCV and OOP

Although OpenCV was written using the Object Oriented Paradigm, it can be utilized by functional languages if the correct infrastructure is in place. For example, Haskell is a purely functional language [11], but still has the ability to use object function calls from OpenCV.

OpenCV is primarily written and interfaced through C++

Bindings exist in Python, Java, and Matlab

Web applications can use OpenCV in Javascript as of 2017 [6]

Wrappers exist in C#, Perl, Ch, Haskell, Ruby, and Go

GoCV - Go and OpenCV 4 computer vision library [15]

This slide is to talk about the type of languages that support use by OpenCV. This is to try to tie in object oriented concepts vs supported languages. We focus on the three most popular languages.

Classes

Image Processing

- Image Filters
    - class BaseFilter

Image Stitching

- Surf Features
    - class CV_EXPORTS SurfFeaturesFinder : public FeaturesFinder
        - Shows Inheritance

Machine Learning

- Statistical Modeling
    - class CvStatModel

Object Detection

- Latent SVM
    - class LatentSvmDetector

Click here for Demo
https://docs.opencv.org/3.4/annotated.html

It should definitely be mentioned here there are over 2500 optimized algorithms in OpenCV. Since OpenCV is object oriented, this gives us a clue as to how many classes there could be if each class is devoted to one algorithm.
There are a large number of base/derived classes.
Here is a link to all of the classes https://docs.opencv.org/3.4/annotated.html
You can change the detail level on the page, there are thousands of classes.
This is a list of the classes within OpenCV, which demonstrates an extremely small limit of its capabilities

Applications

Disclaimer

There are a ton of concepts in computer vision

OpenCV implements a lot of these concepts.

There are a huge amount of projects that utilize OpenCV.

Therefore, it would be impossible to talk about all of them.

We present four applications that represent some core concepts of computer vision.

Image Processing

- Image processing is another common application of OpenCV
- Common image processing tools [17]:
  - Changing the color of the image
  - Geometric Transformations
  - Smoothing/Blurring Images
  - Morphological Transformations
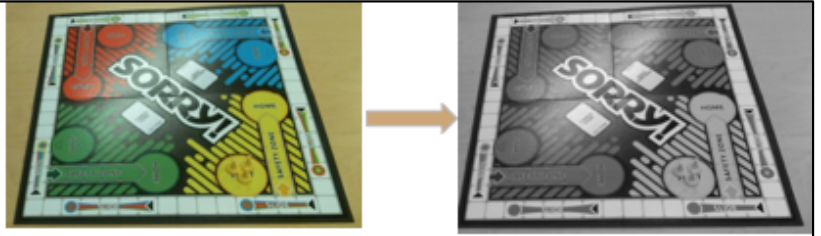
Changing the color of an image is a crucial step in OpenCV as a lot of the functions require binary images for example. Therefore a majority of code using OpenCV has a step where they convert the image into grayscale like you will see in the future slides showing actual code

Image resource: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

Image Stitching

The most common application of image stitching has to do with creating high resolution panoramic images or mosaics.

The OpenCV environment has been used to progress research in different approaches to image stitching [10], as well as supporting different methods.

Some people don't know that Google Street View actively utilizes image stitching [9][1]. The most common solution to image stitching uses a homography matrix to map pixels between one image plane and another.
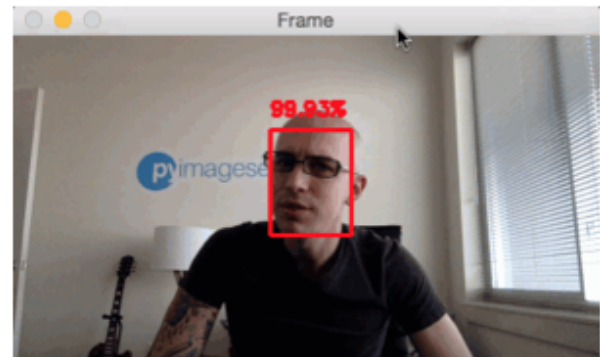Gif source: https://i.gifer.com/TfZO.gif

Face Detection

OpenCV has API specifically for face detection. There are more updated approaches with OpenCV 3.x+ and supported frameworks: TensorFlow, PyTorch, Caffe. Object Detection classes can also be used to detect faces [8].

Class Examples:
EigenFaceRecognizer
FisherFaceRecognizer
LBPHFaceRecognizer

Each of the three FaceRecognizer classes inherit from the FaceRecognizer base class. The derived classes can be used to determine which approach you would like to use, i.e. Fisher vs. Eigen.
This comes from the documentation found here:
https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#eigenfaces
Gif resource: https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/
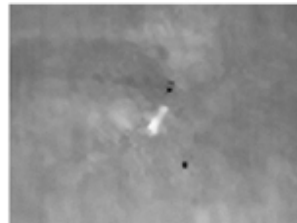
Wildlife Detection

There is an increased interest on the use of Unmanned Aerial Vehicles (UAVs) for wildlife and feral animal monitoring around the world[7].

These vehicles could be deployed in national parks to monitor populations of endangered animals or used to track migration patterns. There is endless applications for using computer vision to wildlife without having to disturb the animal's routine.

Ward et al. uses OpenCV in drones to automatically detect the target (animals). See the graphic to the right, this shows an example of the system

(A)

(B) (C)

Image resource:
https://eprints.qut.edu.au/92309/1/Autonomous_UAVs_Wildlife_Detection_Using_Thermal_Imaging__Predictive_Navigation_and_Computer_Vision1.pdf

Supported Deep Learning Frameworks

TensorFlow

TensorFlow, an object detection API, is a framework for creating deep learning networks that solve object detection problems.

TensorFlow is implemented in C++, but has the most support for Python . TensorFlow can be used in some C-based languages through the use of a C API [12].

OpenCV needs an extra configuration file to import object detection models from TensorFlow. This file varies on the model used in TensorFlow.

OpenCV primarily uses TensorFlow for trained models for recognition tasks.

PyTorch

An open source deep learning platform that provides a seamless path from research prototyping to production deployment.

It's a Python-based scientific computing package targeted at two sets of audiences:

- A replacement for NumPy to use the power of GPUs
- a deep learning research platform that provides maximum flexibility and speed

PyTorch uses trained classifiers to detect objects in images and videos.

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors [14].

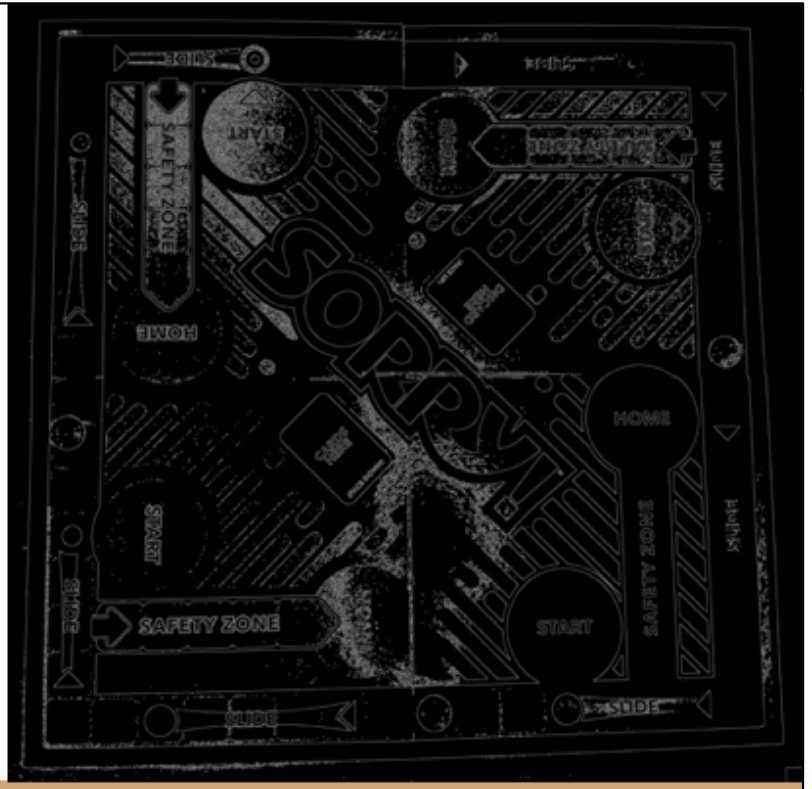**OpenCV can use trained models from Caffe** to recognize objects.

Code

Gabriella

Canny Edge Detection


```
image = cv2.imread('Board.jpg')

grayScale = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)
   , thresh = cv2.threshold(grayScale, 105, 255, 0)
cannyEdge = cv2.Canny(thresh, 125, 255)
```

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages. With the final output that is shown here being what it believes to the "strong" edges in the image. [17]

Contours

```
image = cv2.imread('Board.jpg')

grayScale = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(grayScale, 105, 255, 0)
cannyEdge = cv2.Canny(thresh, 125, 255)

contours, hierarchy =
cv2.findContours(cannyEdge.copy(),
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cv2.imwrite("NewBoard.jpg", image)
```


Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection. [17] In the code here we are finding the contours in a specific image and then drawing them

Color Detection

```python
image = cv2.imread('Board.jpg')

hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# defining the Range of blue color
blueLowerBound = np.array([99, 115, 150], np.uint8)
blueUpperBound = np.array([110, 255, 255], np.uint8)

# defining the Range of yellow color
yellowLowerBound = np.array([20, 190, 20], np.uint8)
yellowUpperBound = np.array([30, 255, 255], np.uint8)

# finding the range of blue and yellow color in the
image
blue = cv2.inRange(hsv, blueLowerBound,
blueUpperBound)
yellow = cv2.inRange(hsv, yellowLowerBound,
yellowUpperBound)

# Morphological transformation, Dilation
kernal = np.ones((5, 5), "uint8")

blue = cv2.dilate(blue, kernal)
res1 = cv2.bitwise_and(image, image, mask=blue)

yellow = cv2.dilate(yellow, kernal)
res2 = cv2.bitwise_and(image, image, mask=yellow)
```

```python
# Tracking the Blue Color
contours, hierarchy = cv2.findContours(blue,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if area > 300:
        x, y, w, h = cv2.boundingRect(contour)
        ar = w / float(h)
        image = cv2.rectangle(image, (x, y), (x + w, y + h),
(255, 0, 0), 2)
        cv2.putText(image, "blue color", (x, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0))

# Tracking the yellow Color
contours, hierarchy = cv2.findContours(yellow,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if area > 300:
        x, y, w, h = cv2.boundingRect(contour)
        ar = w / float(h)
        image = cv2.rectangle(image, (x, y), (x + w, y + h),
(0, 255, 255), 2)
        cv2.putText(image, "yellow_color", (x, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255))

cv2.imwrite('NewBoard.jpg', image)
```

Color Detection is used to detect colors within a specific HSV (hue, saturation, value) value in an image or video. The code shown here: sets the range for the colors blue and yellow and then searches within the given image for those colors and puts a bounding box around the pixels it believes matches the range of values provided, which is shown on the next slide.

Here you can see the program works pretty well with this image, putting a bounding box around the majority of the blue and yellow components of the board. One reason that there is normally some variability in its accuracy is the change in lighting. As you can see the bottom of the board is in a shadow and its having trouble detecting the blue of that slide bar. This is a very common challenge when trying to use computer vision

References

1. Gary Bradski and Adrian Kaehler. Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc., 2008.
2. Taheri, Sajjad (2018-05-31). "Computer Vision for the Masses: Bringing Computer Vision to the Open Web Platform". eeTimes. Retrieved 26 november 2018.
3. Brahmbhatt, Samarth. Practical OpenCV. Apress, 2013.
4. Agam, Gady. "Introduction to programming with OpenCV." Online Document 27 (2006).
5. https://opencv.org/
6. Taheri, Sajjad, et al. "OpenCV. js: Computer vision processing for the Web." Univ. California, Irvine, CA, USA, Tech. Rep. CECS TR (2017): 17-02.
7. Ward, Sean, et al. "Autonomous UAVs wildlife detection using thermal imaging, predictive navigation and computer vision." Proceedings of the 2016 IEEE Aerospace Conference. IEEE, 2016.
8. Blomgren, Staffan, and Marcus Hertz. "Facing the differences between Facebook and OpenCV: A facial detection comparison between Open Library Computer Vision and Facebook." (2015).
9. Anguelov, Dragomir, et al. "Google street view: Capturing the world at street level." Computer 43.6 (2010): 32-38.
10. Chen, Kaili, and Meiling Wang. "Image stitching algorithm research based on OpenCV." Proceedings of the 33rd Chinese Control Conference. IEEE, 2014.
11. Thompson, Simon. Haskell: the craft of functional programming. Vol. 2. Addison-Wesley, 2011.
12. https://www.tensorflow.org/guide/extend/bindings
13. Nvidia, C. U. D. A. "Programming guide." (2010).
14. Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014.
15. https://gocv.io/
16. http://simplecv.org/
17. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html
18. https://docs.opencv.org/3.4/d0/de3/tutorial_py_intro.html

Questions?