OpenCV - II

OpenCV Object Detection

【110上】嵌入式系統技術實驗課程

TA: 陳翰群 hanz1211.ee09@nycu.edu.tw

Edge Detection

- OpenCV provides many edge-finding filters, including Laplacian, Sobel, Canny, and Scharr.
  - However, they are prone to misidentifying noise as edges.
- OpenCV also provides many blurring filters, including blur (a simple average), medianBlur, and GaussianBlur.
- The arguments for the edge-finding and blurring filters vary but always include ksize, an odd whole number that represents the width and height (in pixels) of a filter's kernel.

Edge detection with Canny

- OpenCV offers a handy function called Canny (after the algorithm's inventor, John F. Canny)
- You can do it in one line with OpenCV. In details, it is a five-step process:
  - Denoise the image with a Gaussian filter.
  - Calculate the gradients.
  - Apply non-maximum suppression (NMS) on the edges.
  - Apply a double threshold to all the detected edges to eliminate any false positives.
  - Analyze all the edges and their connection to each other to keep the real edges and discard the weak ones.
- Check canny.py for details

Binarize Image

- For a simple, high contrast image, you can threshold it that make edge detection easier.
- In contours_hull.py
  - You can check threshold result by imshow the thresh value
- cv2.threshold
  - 2nd argument is the threshold, pixel lower than this will be zero
  - 3rd argument is the output value for pixel larger than the threshold

Contour detection

- After finding Canny edges, we can do further analysis of the edges in order to determine whether they match a common shape, such as a line or a circle.
- In contours_hull.py

```
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

  - cv2.RETR_TREE, which tells the function to retrieve the entire hierarchy of external and internal contours
  - cv2.RETR_EXTERNAL tells the function only retrieve the most external contours
  - cv2.CHAIN_APPROX_SIMPLE will compress contours pixels to save memory

Approximate Bounding Polygon

- cv2.approxPolyDP can calculate the approximate bounding polygon of a shape
  - Douglas-Peucker Algorithm
- This function takes 3 parameters:
  - A contour
  - An epsilon value representing the maximum discrepancy between the original contour and the approximated polygon
    - The lower the value, the closer the approximated value will be to the original contour
  - A Boolean flag. If it is True, it signifies that the polygon is closed.
- Check contours_hull.py for details
  - Usually, epsilon is obtained from contour arc length. In this demo, we suggest 1% of the original arc length.
  - cv2.arcLength

Approximate Bounding Polygon

- A convex shape is one where there are no two points within this shape whose connecting line goes outside the perimeter of the shape itself.
- OpenCV also offers a cv2.convexHull function for obtaining processed contour information for convex shapes.

Green: Original contour
Cyan: Approximate polygon
Red: Convex shape bounding contour

Detect Other Shapes

- For line and circle, it's better using Hough transform
  - Detecting lines
    - cv2.HoughLines function or the cv2.HoughLinesP function.
    - The latter uses the probabilistic Hough transform
  - Detecting circles
    - cv2.HoughCircles
- Detecting other shapes
  - Combined use of cv2.findContours and cv2.approxPolyDP

Lab4-1 Get Star Contour

- lab4-1.py
- Draw the contour of the star
- Hint: Use threshold and define RoI before other processes can make it easier.

Detect Face

- Haar cascade classifier
  - Haar-like features are one type of feature that is often applied to real-time face detection
  - They were first used for this purpose in the paper:
    - Robust Real-Time Face Detection, by Paul Viola and Michael Jones (International Journal of Computer Vision 57(2), 137–154, Kluwer Academic Publishers, 2001)
- Haar cascades, as implemented in OpenCV, are not robust to changes in rotation or perspective

Detect Face

- Use cv2.CascadeClassifier to read a pretrained classifier
- In lab4-2.py
- One line to declare a classifier

```
face_cascade = cv2.CascadeClassifier(
    './cascades/haarcascade_frontalface_default.xml')
```

- After converting to gray image, run detectMultiScale

```
faces = face_cascade.detectMultiScale(gray, 1.2, 5)
```

Scale Factor: large value will perform faster
must >1.0
Min Neighbor: larger means higher confidence

- Draw the bounding box with cv2.rectangle

```
img = cv2.rectangle(img, (x, y), (x + w, y + h), (255, 255, 0), 2)
```

Lab4-2 Detect Faces

- lab4-2.py
- Modify the code, use detection on images/faces_2.jpg, you can remove eye detection
- You need to adjust parameter or exclude wrong bounding box
- These result are not accepted

Too few detected (need 75% or more)

Wrong bounding box not excluded

Using Image Descriptors

- Several algorithms can be used to detect and describe features
- The most used feature detection and descriptor extraction algorithms in OpenCV are as follows:
  - Harris: detecting corners.
  - SIFT: detecting blobs.
  - SURF: detecting blobs.
  - FAST: detecting corners.
  - BRIEF: detecting blobs.
  - ORB: This algorithm stands for Oriented FAST and Rotated BRIEF. It is useful for detecting a combination of corners and blobs.
- Matching features can be performed with the following methods:
  - Brute-force matching
  - FLANN-based matching

Using Image Descriptors

- orb_knn.py
- Get image key points and descriptors with ORB, similar with SIFT & SURF

```
orb = cv2.ORB_create()
kp0, des0 = orb.detectAndCompute(img0, None)
kp1, des1 = orb.detectAndCompute(img1, None)
```

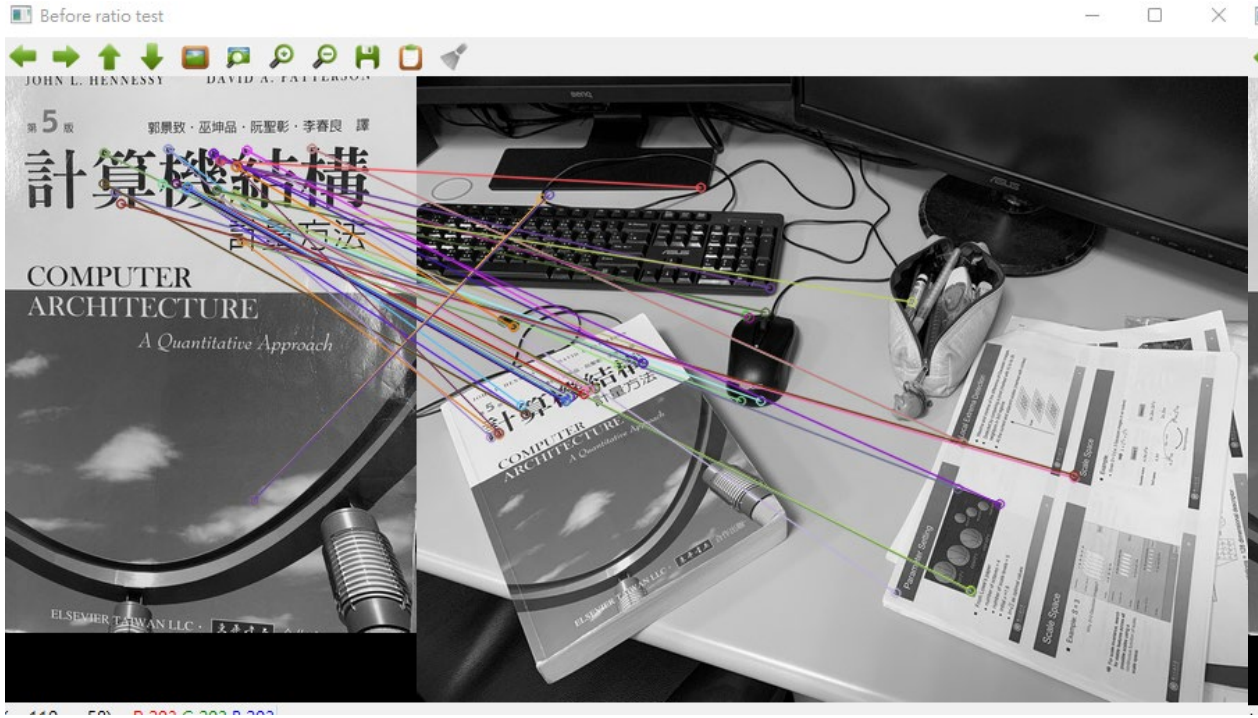
- Use brute-force KNN for matching

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
pairs_of_matches = bf.knnMatch(des0, des1, k=2)
```

- To ensure better matching quality, we normally apply ratio test after
  - First proposed by David Lowe, the author of the SIFT algorithm

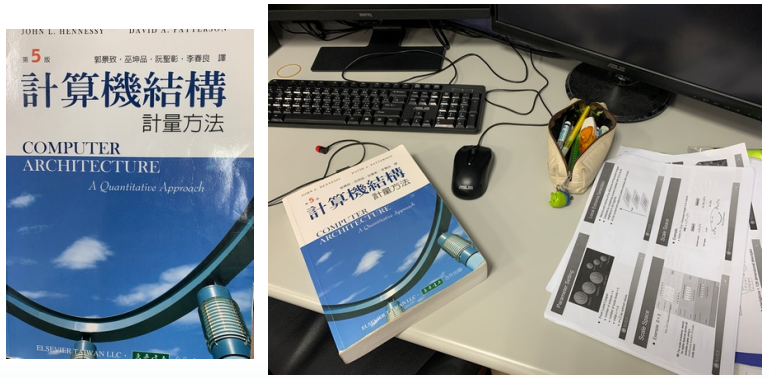
Using Image Descriptors

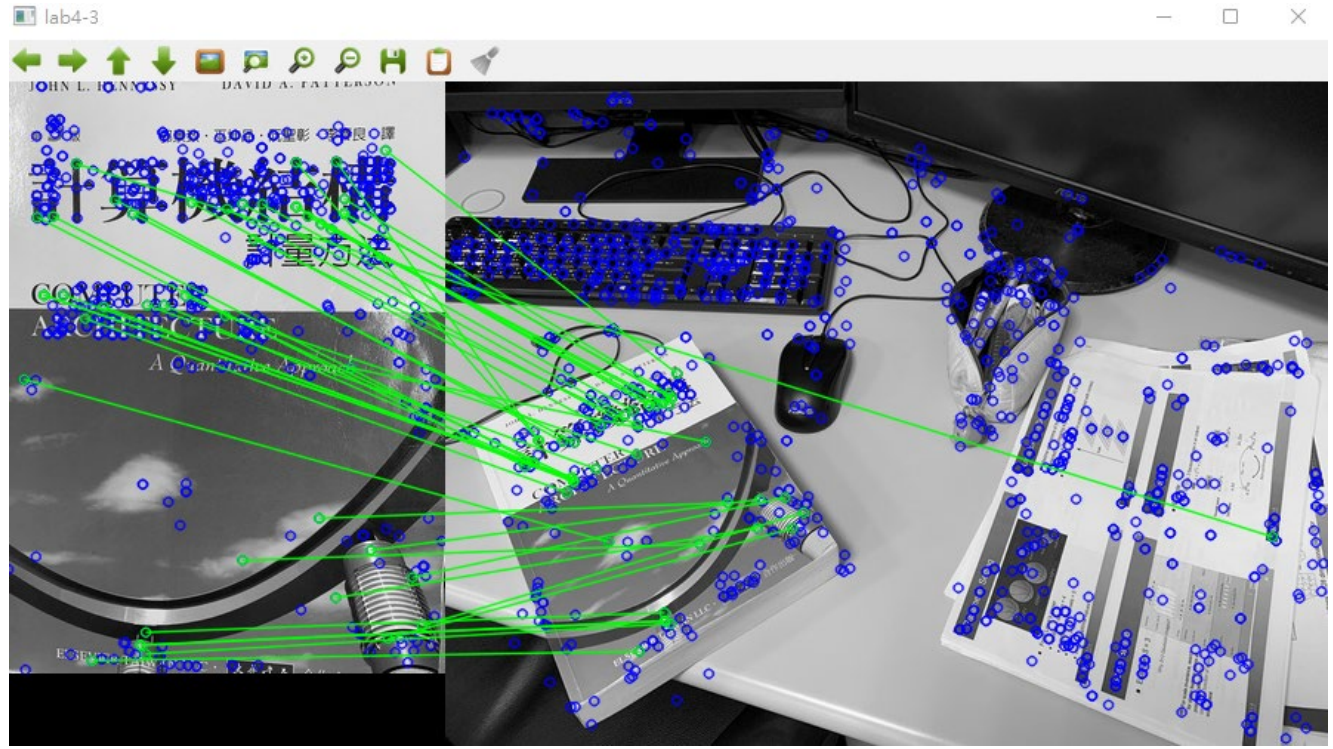
- orb_knn.py result



Lab4-3 SIFT & FLANN

- lab4-3.py
- Search the document online, use OpenCV implemented SIFT & FLANN-based matching algorithm.
- Hint: the methods you may find useful are
  - cv2.SIFT_create()
  - cv2.FlannBasedMatcher()
  - cv2.FlannBasedMatcher().knnMatch()
  - cv2.drawMatchesKnn()
- https://docs.opencv.org/4.5.3/





Using CNN Models

- OpenCV has methods for importing models built with deep learning frameworks.
  - cv2.dnn
- In lab4-4.py we will use the object detection model called SSD
- You can load Tensorflow model with cv2.dnn.readNetFromTensorflow
  - This method accepts a path to a file that contains a TensorFlow model in binary Protobuf (Protocol Buffers) format:

```
config = "ssd/ssd_mobilenet_v1_coco_2017_11_17.pbtxt.txt"
model = "ssd/frozen_inference_graph.pb"
detector = cv2.dnn.readNetFromTensorflow(model, config)
```

  - There is also cv2.dnn.readNetFromDarknet if you want to use YOLO family models, which accept darknet cfg.

Using CNN Models

- You can feed the input by:

```
detector.setInput(
    cv2.dnn.blobFromImage(
        img,
        size=INPUT_SIZE,
        swapRB=True,
        crop=False))
```

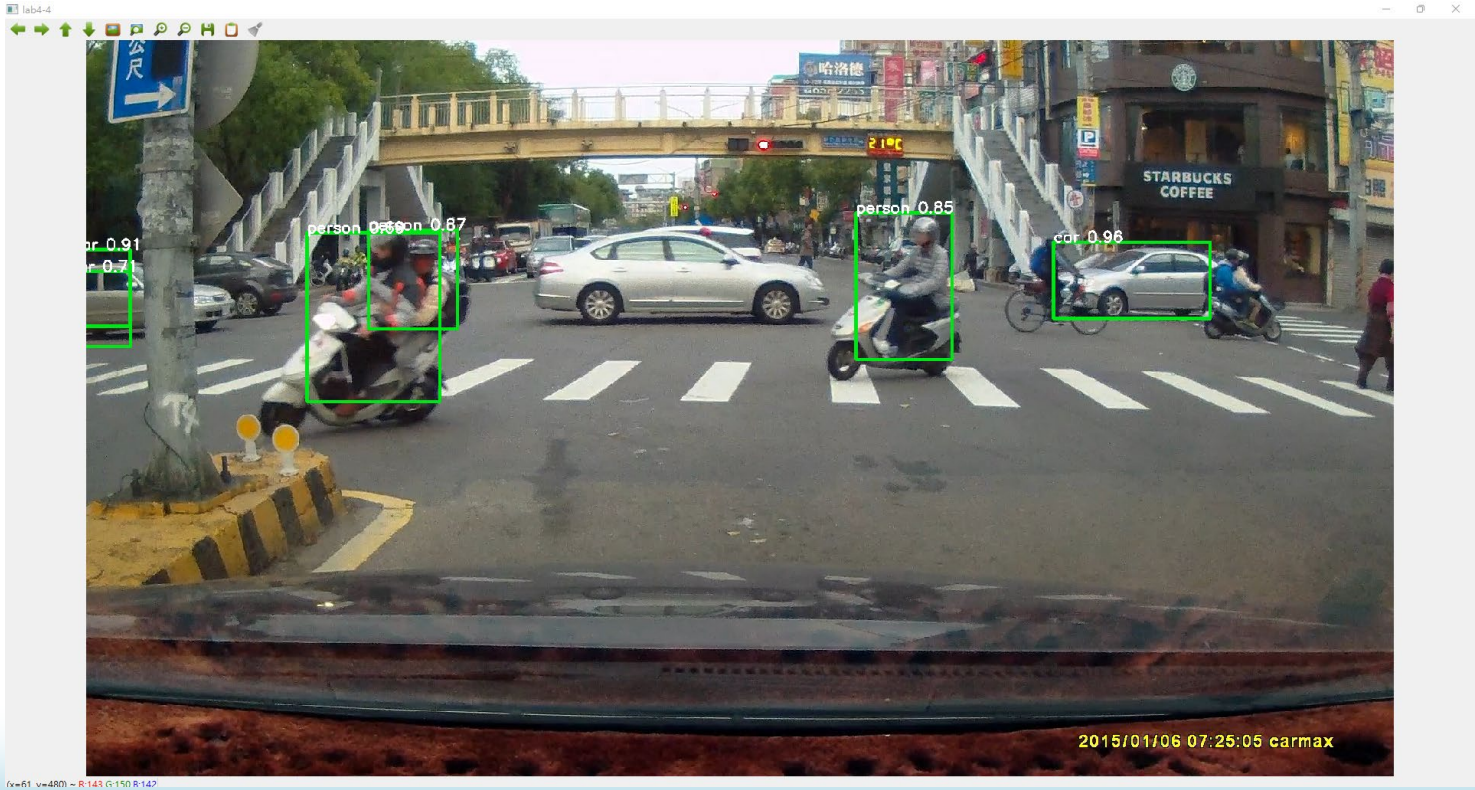
- And then do the inference:

```
detections = detector.forward()[0, 0, :, 1:]
```

  - forward stands for forward propagation.
  - The result is a 2-D array.
  - The first index of the array specifies the detection number
  - The second index represents a specific detection, which is expressed by the object class, score
  - The fourth values specifying two corner coordinates of the bounding box.

Lab4-4 SSD in OpenCV

- lab4-4.py
- Visualize the detector outputs.
- You only need to draw the objects that in TRACKED_CLASSES



Demo

- 本次Lab以個人為單位
- 配分
  - Lab4-1 : 40%
  - Lab4-2 : 20%
  - Lab4-3 : 20%
  - Lab4-4 : 20%
- Demo
  - 完成Lab後，請進視訊會議舉手呼叫助教們demo
  - 多個小題可以分次demo
  - 根據助教要求呈現程式執行結果
- 最後登記時間：21:20