AI & Robotics

Computer Vision

PXL AI & ROBOTICS LAB

Goals

The **junior-colleague**

- can use OpenCV to analyse image (streams) with Python in projects.

Computer Vision

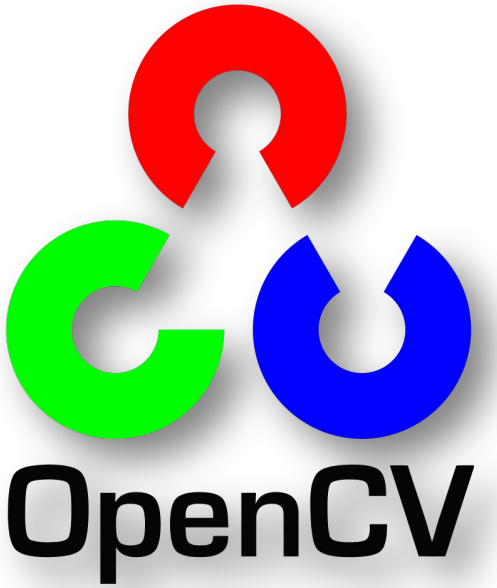
is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

-- Wikipedia

Computer Vision

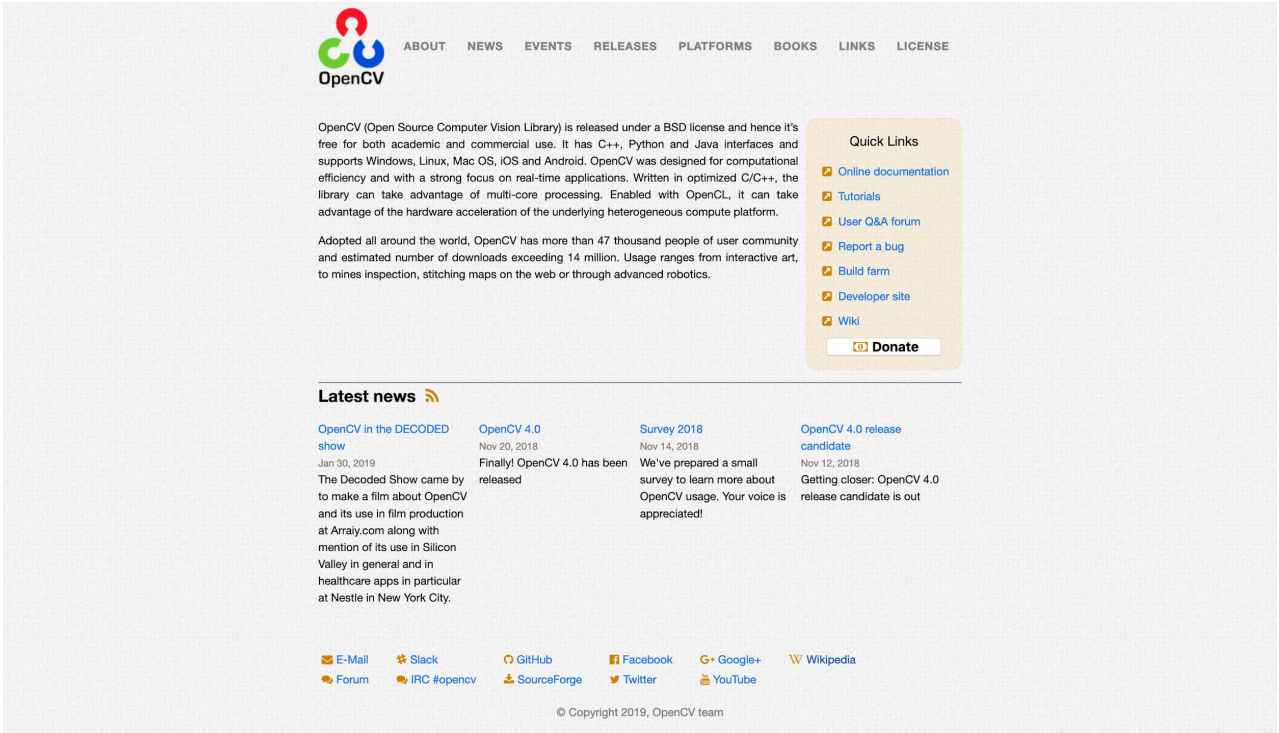
- Ability of computers to see
- Acquiring
- Processing
- Analyzing
- Understanding
- Picture = 1000 words

OpenCV



- Open Source Computer Vision
- Cross-platform library
- Real-time Computer Vision
- Originally by Intel
- Well documented
- Multiple bindings
  (C++, Java & Python)

OpenCV

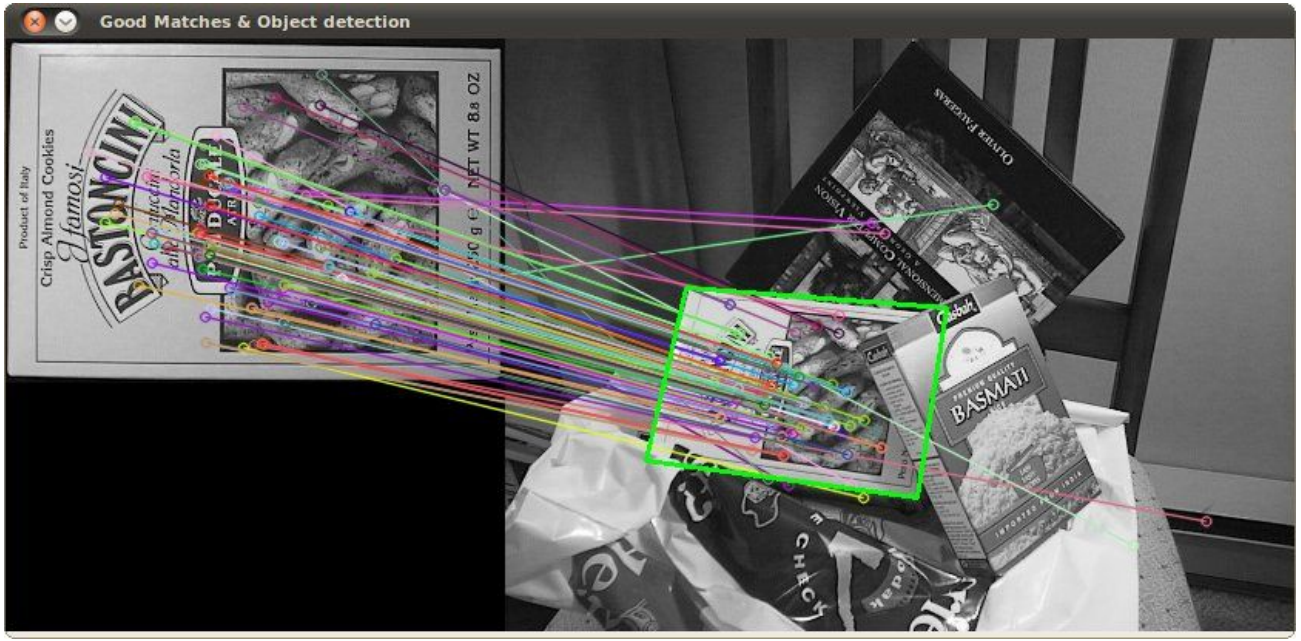


http://opencv.org

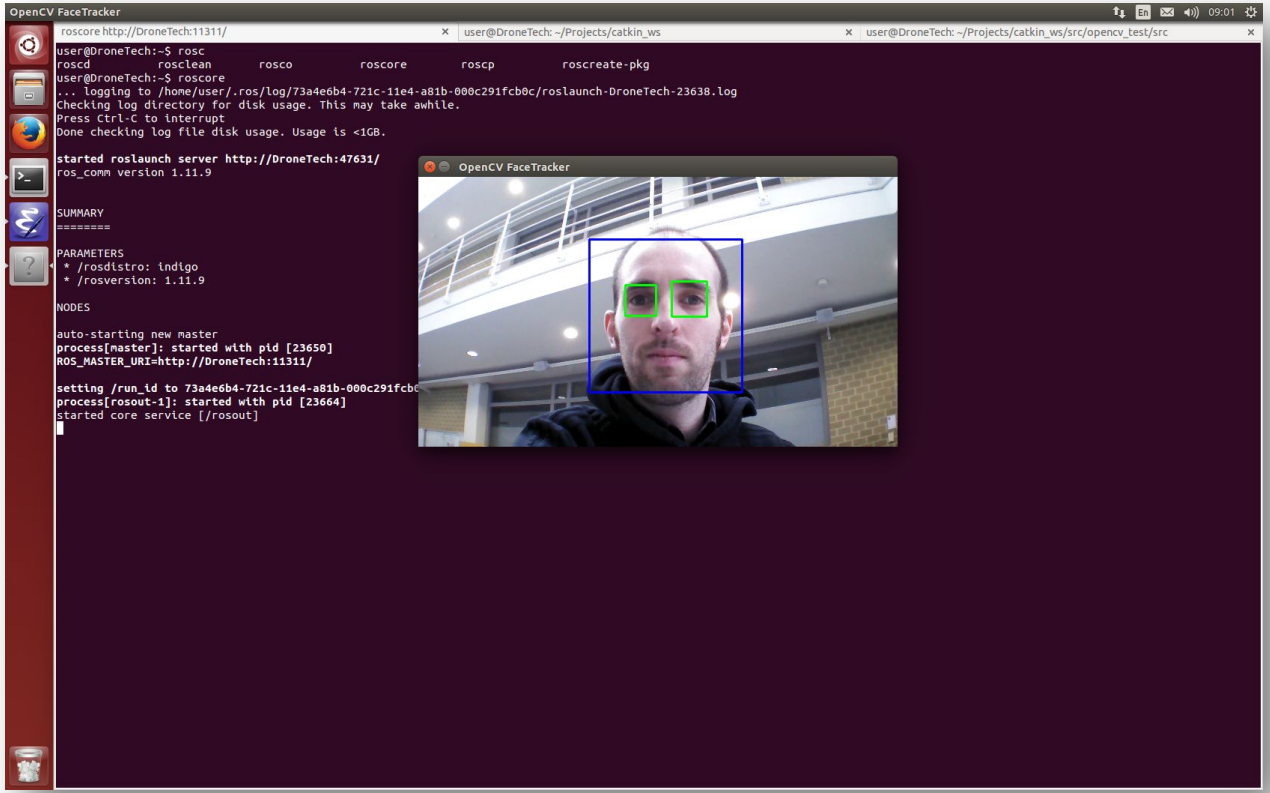
OpenCV Capabilities

- Object identification
- Face and gesture recognition
- Optical flow
- Visual Odometry (Egomotion Estimation)
- Structure from motion (SFM)
- Stereo and multi-camera calibration
- Depth computation
- . . .

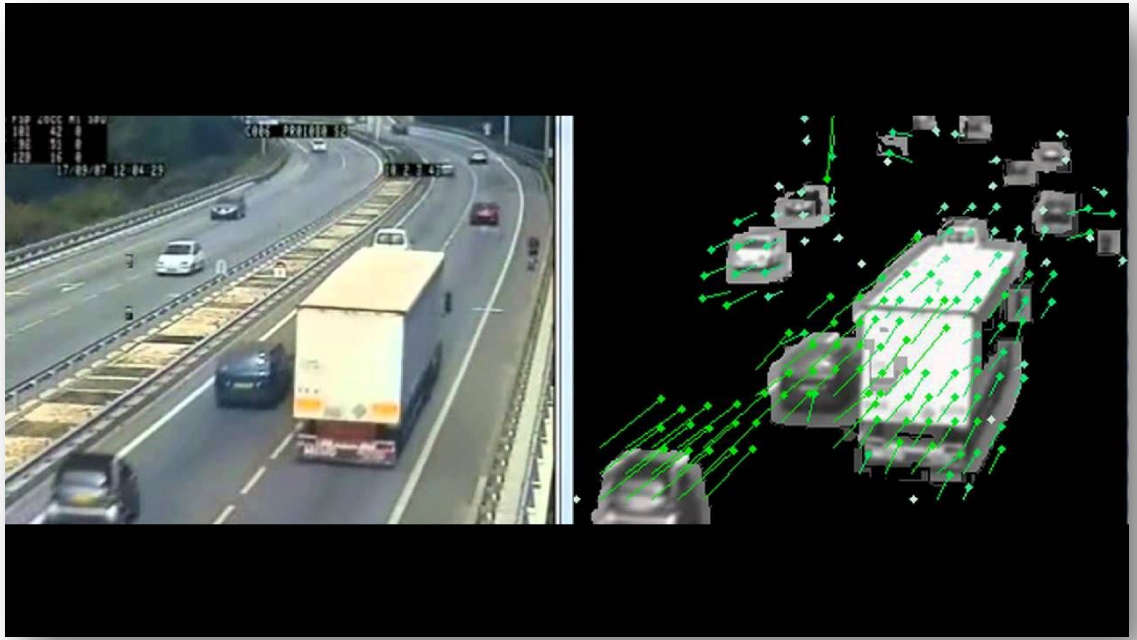
Object Identification



Face Recognition



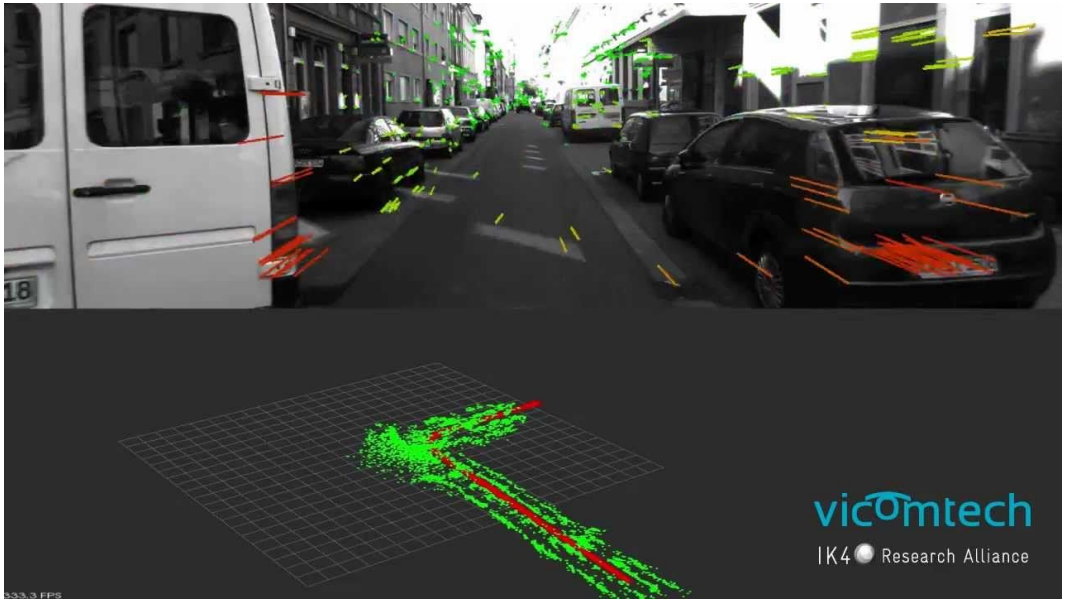
Optical Flow



https://www.youtube.com/watch?v=4fMzvB4YLMI

The pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer
(an eye or a camera) and the scene.

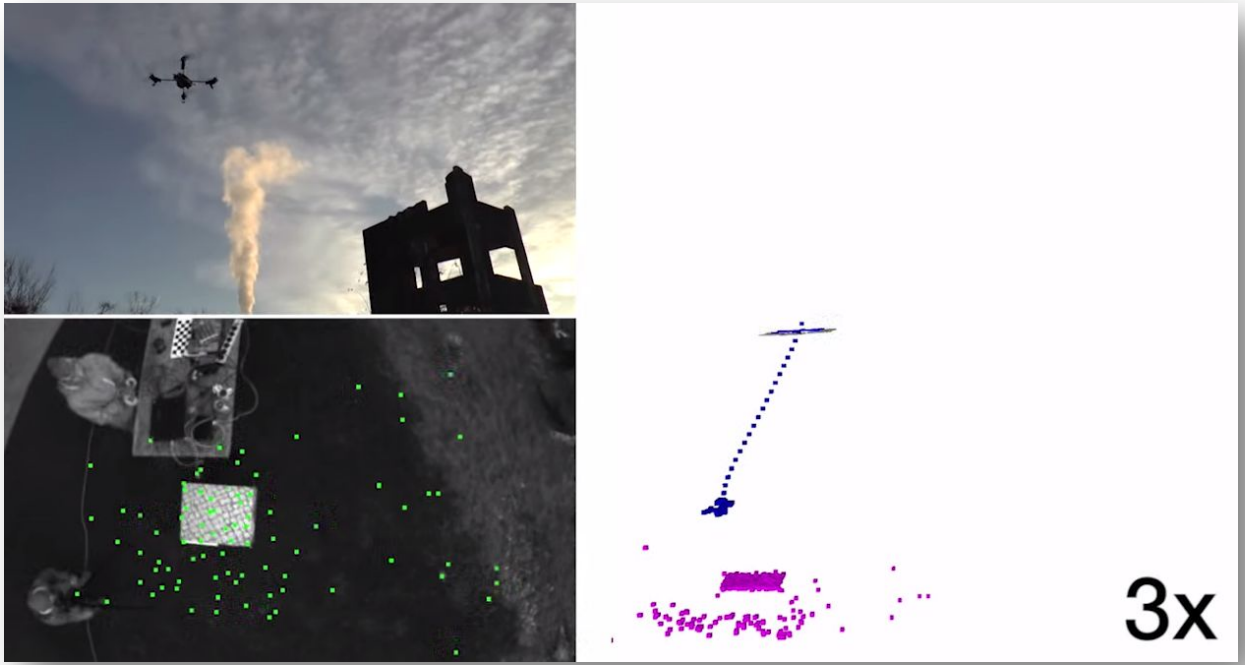
Visual Odometry (Egomotion Estimation)



https://www.youtube.com/watch?v=lTQGTbrNssQ

Determining position and orientation of a robot by analyzing its camera's image sequence.

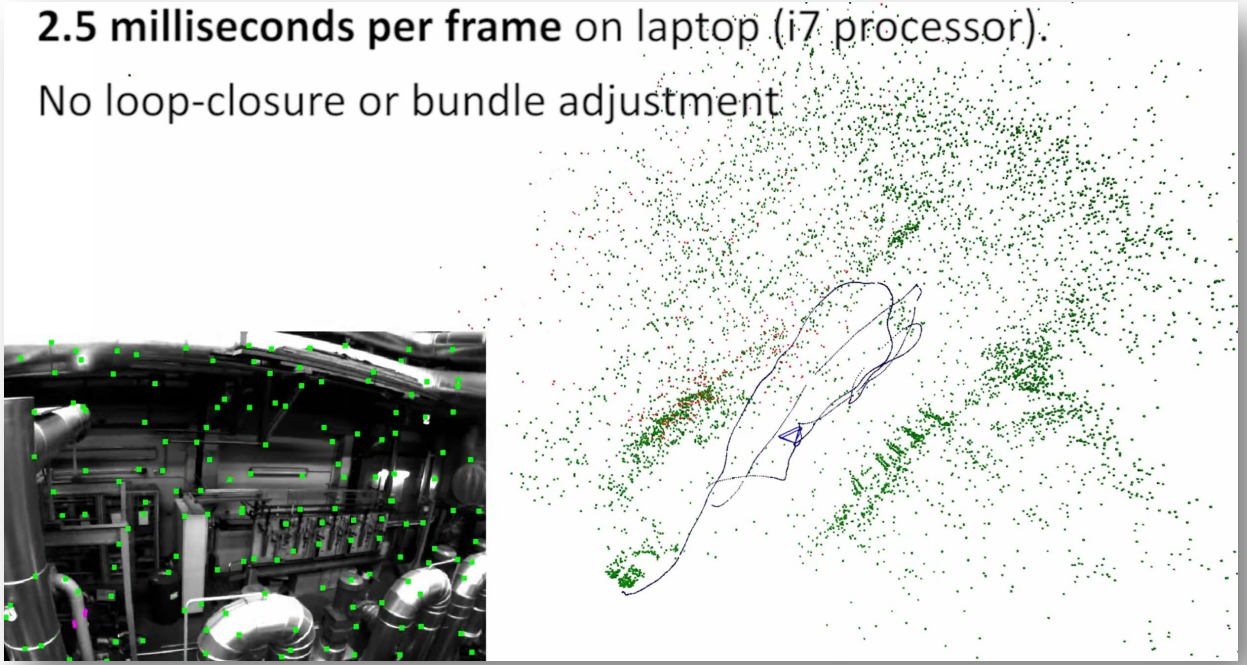
Visual Odometry (Egomotion Estimation)



SVO: Fast Semi-Direct Monocular Visual Odometry

[SOURCE]
https://www.youtube.com/watch?v=2YnIMfw6bJY

Visual Odometry (Egomotion Estimation)



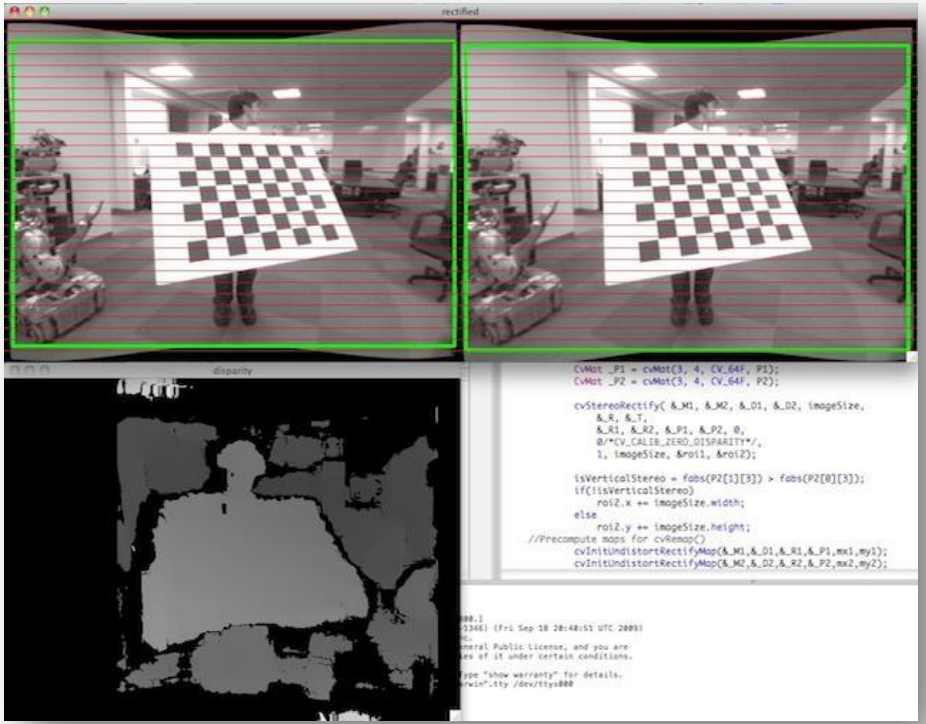
SVO 2.0: Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems

[SOURCE]
https://www.youtube.com/watch?v=hR8uq1RTUfA

Structure from Motion

- SfM
- Estimating 3D structure
- From 2D image sequences
- Based on the phenomenon by which humans can recover 3D structure from the projected 2D (retinal) motion field of a moving object or scene.

Stereo and multi-camera calibration



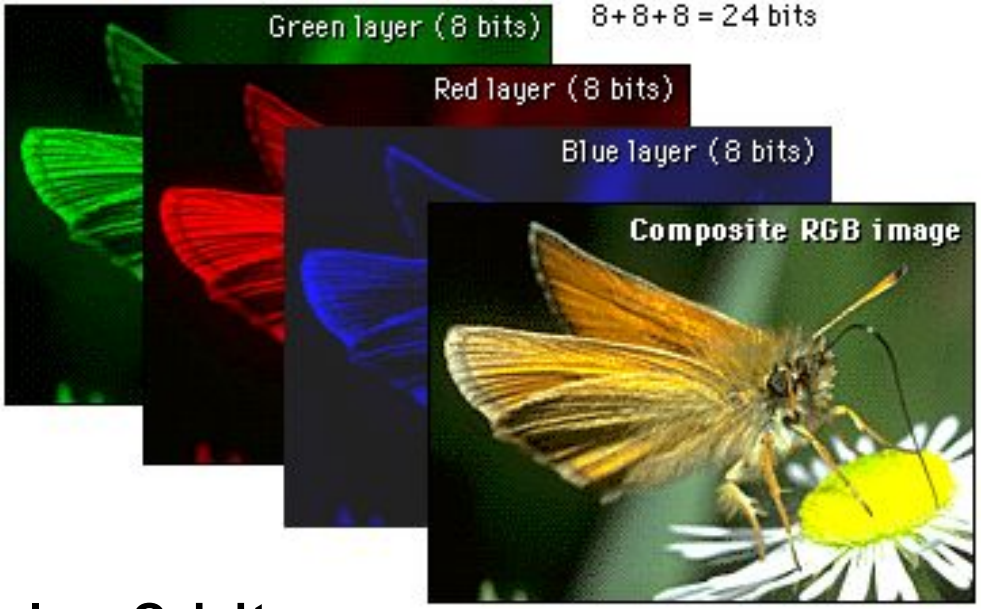
[INFO]
http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

New Package: Vision Controller

```
$ catkin_create_pkg vision_controller sensor_msgs \
> cv_bridge rospy std_msgs
```

+ vision_controller.py

Color Detection: RGB



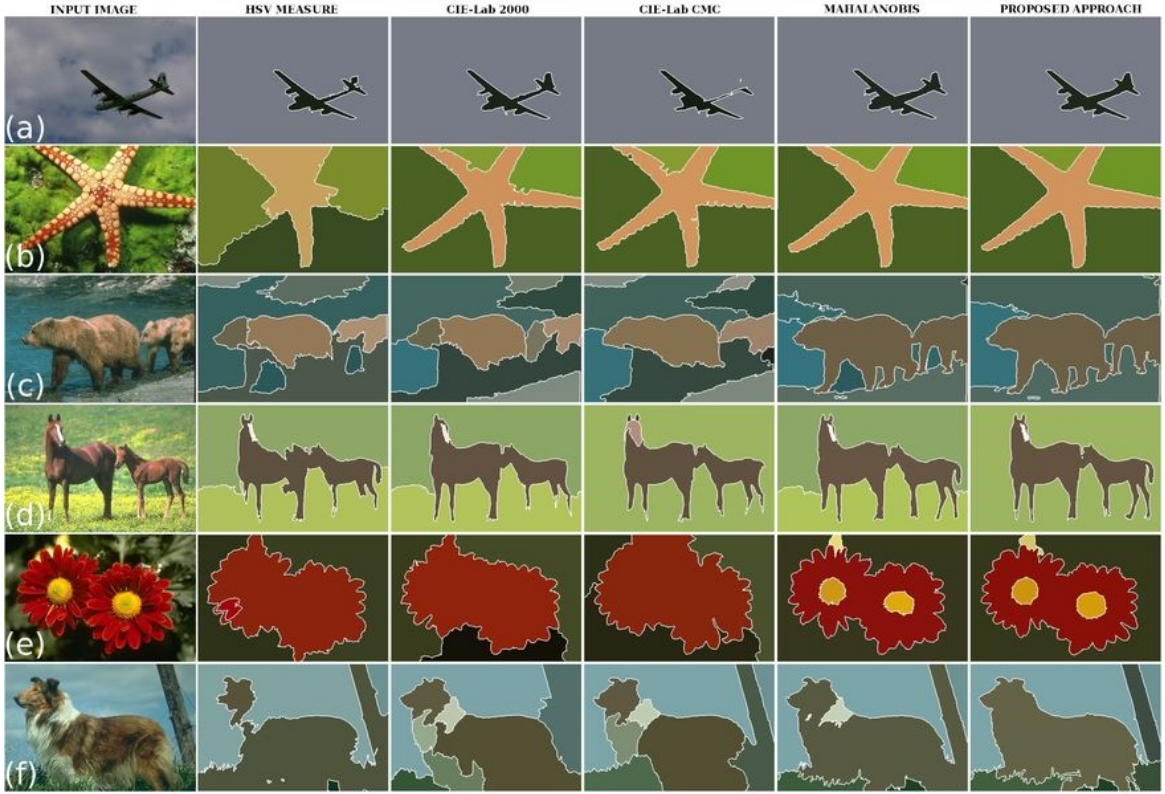
1 green pixel = 8 bit
1 red pixel = 8 bit
1 blue pixel = 8 bit

8 bit = 0 - 255

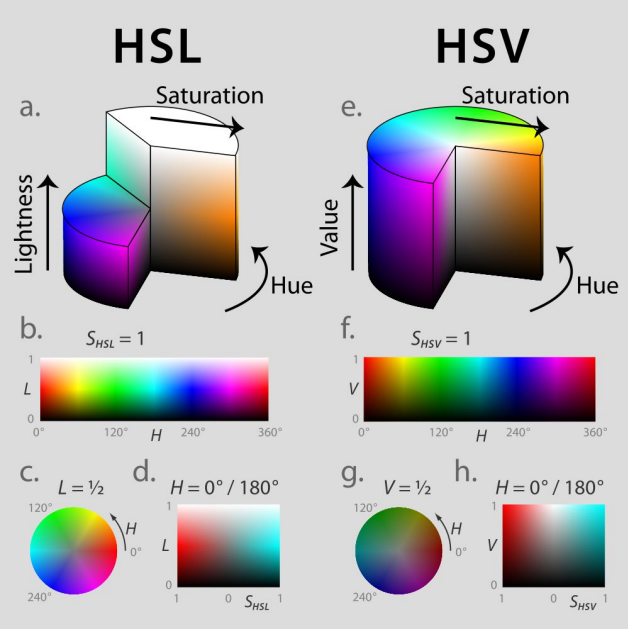
Color Detection: RGB

Not the best for color segmentation!

Color segmentation

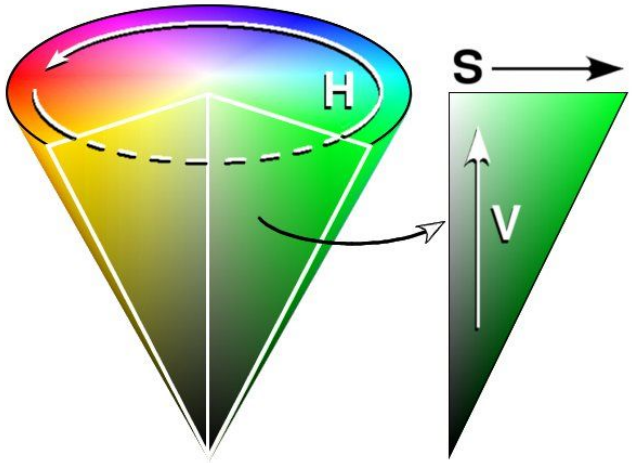


HSV



- Based on how humans perceive color
- 3 Components
  - Hue
    Color
  - Saturation
    Amount of white in the color
  - Value (Lightness)
    Amount of black in the color (how dark is the color)

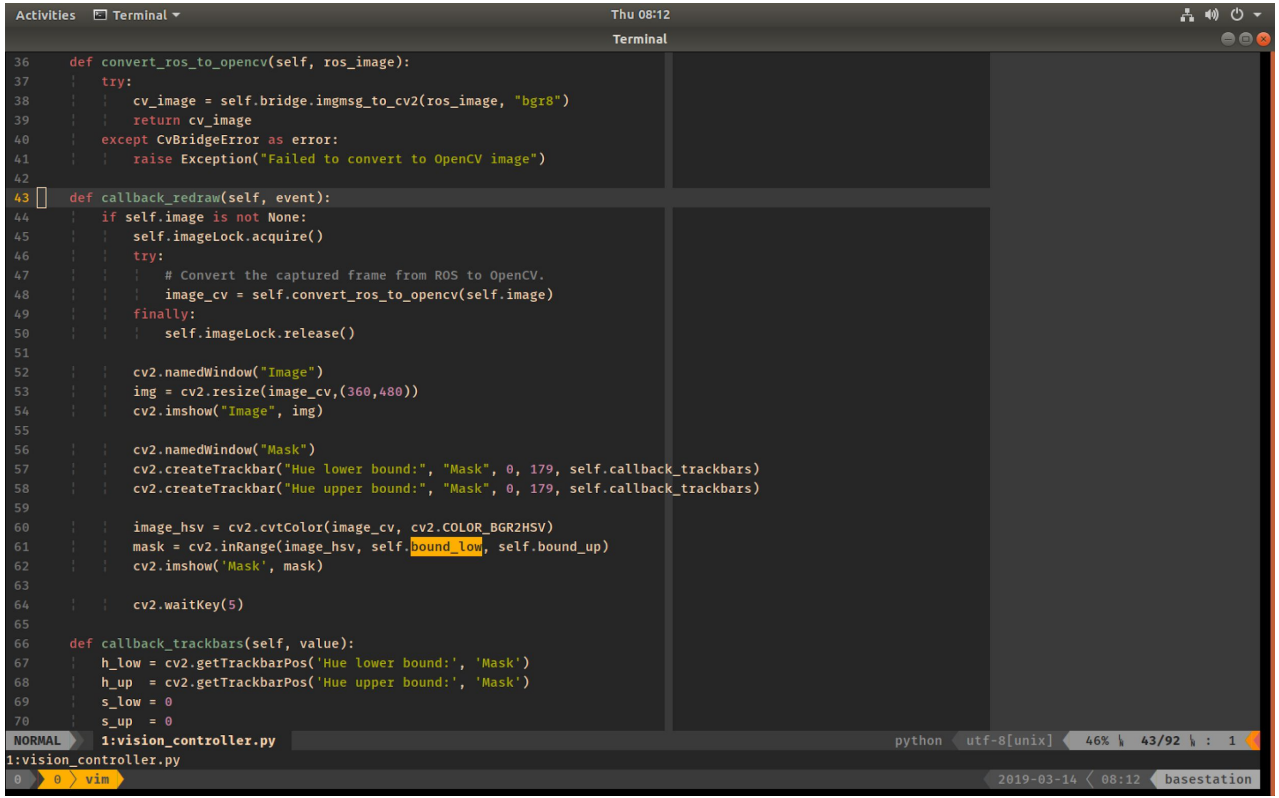
HSV in OpenCV



- Hue: 0 – 179
- Saturation: 0 – 255
- Value: 0 – 255

Hue: split @ Red!

Vision Controller Code Overview

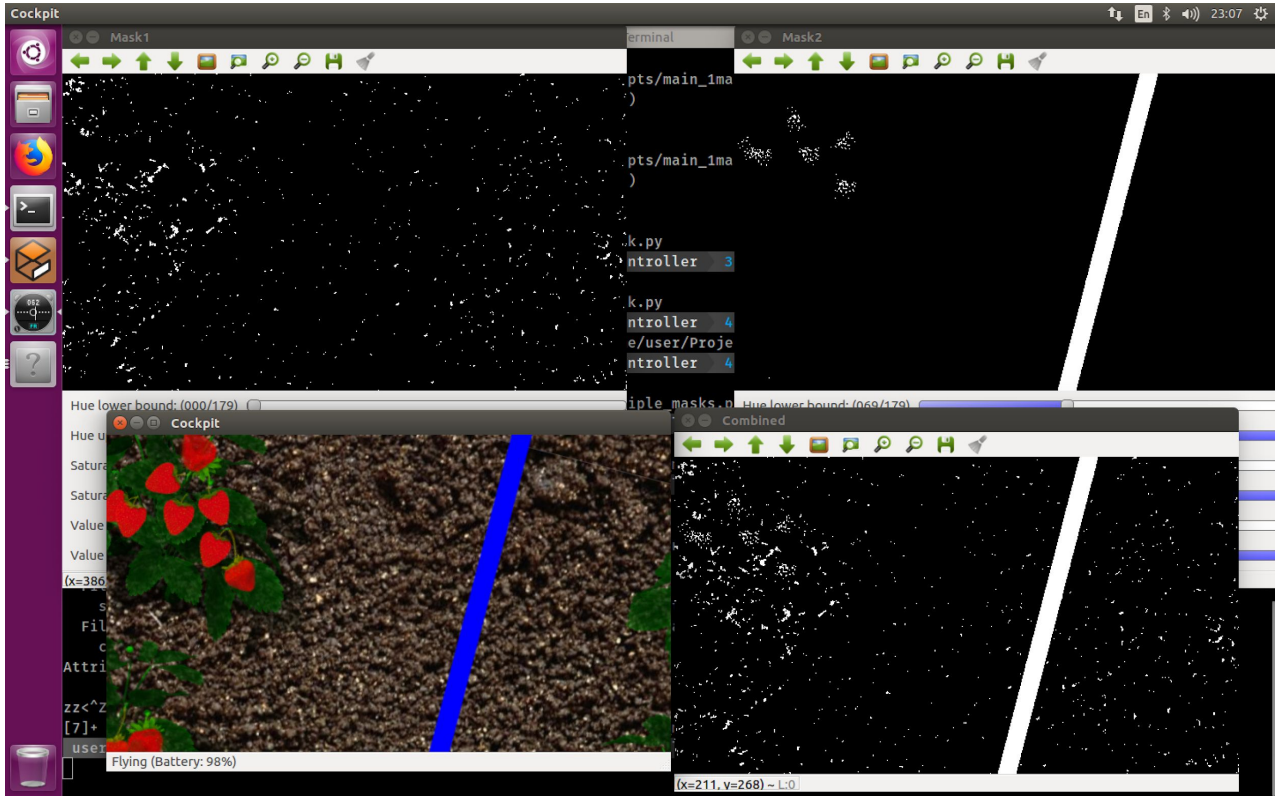


Merge Two Masks . . .

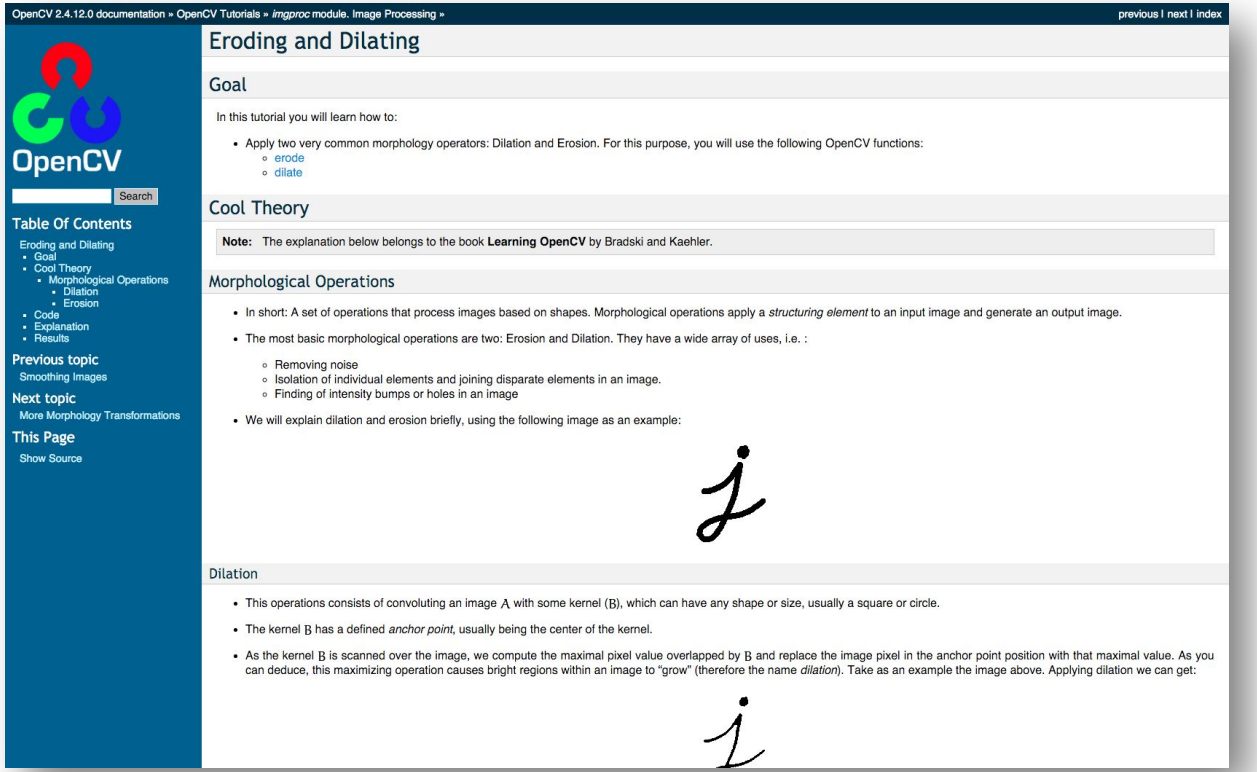
```
# Combine masks
mask = mask1 + mask2

cv2.imshow('Combined', mask)
```

Merge Two Masks . . .

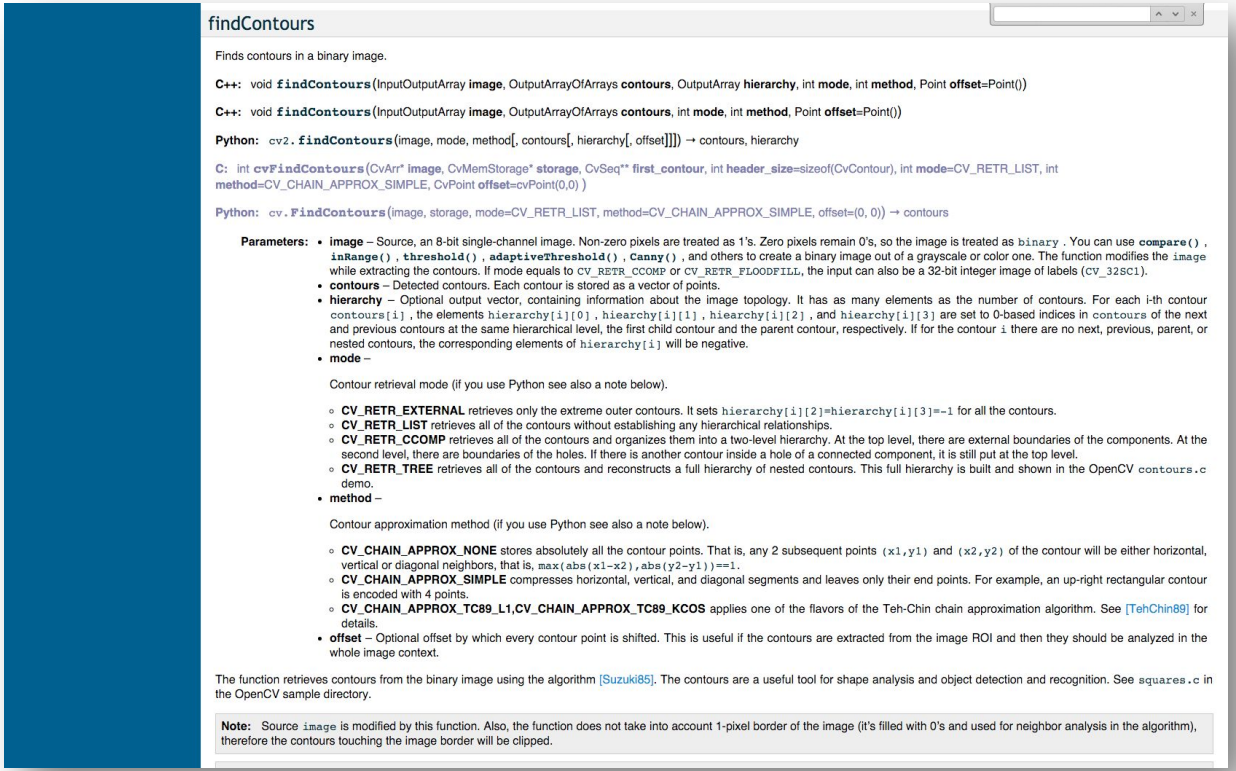


Eroding and Dilating



[INFO]
http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

Contours



findContours

Finds contours in a binary image.

C++: void **findContours**(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())

C++: void **findContours**(InputOutputArray **image**, OutputArrayOfArrays **contours**, int **mode**, int **method**, Point **offset**=Point())

Python: `cv2.findContours`(image, mode, method[, contours[, hierarchy[, offset]]]) → contours, hierarchy

C: int **cvFindContours**(CvArr* **image**, CvMemStorage* **storage**, CvSeq** **first_contour**, int **header_size**=sizeof(CvContour), int **mode**=CV_RETR_LIST, int **method**=CV_CHAIN_APPROX_SIMPLE, CvPoint **offset**=cvPoint(0,0))

Python: `cv.FindContours`(image, storage, mode=CV_RETR_LIST, method=CV_CHAIN_APPROX_SIMPLE, offset=(0, 0)) → contours

Parameters:

- **image** – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as `binary` . You can use **compare()** , **inRange()** , **threshold()** , **adaptiveThreshold()** , **Canny()** , and others to create a binary image out of a grayscale or color one. The function modifies the `image` while extracting the contours. If mode equals to `CV_RETR_CCOMP` or `CV_RETR_FLOODFILL`, the input can also be a 32-bit integer image of labels (`CV_32SC1`).
- **contours** – Detected contours. Each contour is stored as a vector of points.
- **hierarchy** – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour `contours[i]` , the elements `hierarchy[i][0]` , `hiearchy[i][1]` , `hierarchy[i][2]` , and `hiearchy[i][3]` are set to 0-based indices in `contours` of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour `i` there are no next, previous, parent, or nested contours, the corresponding elements of `hierarchy[i]` will be negative.
- **mode** –

  Contour retrieval mode (if you use Python see also a note below).

  - **CV_RETR_EXTERNAL** retrieves only the extreme outer contours. It sets `hierarchy[i][2]=hierarchy[i][3]=-1` for all the contours.
  - **CV_RETR_LIST** retrieves all of the contours without establishing any hierarchical relationships.
  - **CV_RETR_CCOMP** retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
  - **CV_RETR_TREE** retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV `contours.c` demo.
- **method** –

  Contour approximation method (if you use Python see also a note below).

  - **CV_CHAIN_APPROX_NONE** stores absolutely all the contour points. That is, any 2 subsequent points `(x1,y1)` and `(x2,y2)` of the contour will be either horizontal, vertical or diagonal neighbors, that is, `max(abs(x1-x2),abs(y2-y1))==1`.
  - **CV_CHAIN_APPROX_SIMPLE** compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
  - **CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS** applies one of the flavors of the Teh-Chin chain approximation algorithm. See [TehChin89] for details.
- **offset** – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

The function retrieves contours from the binary image using the algorithm [Suzuki85]. The contours are a useful tool for shape analysis and object detection and recognition. See `squares.c` in the OpenCV sample directory.

Note: Source `image` is modified by this function. Also, the function does not take into account 1-pixel border of the image (it's filled with 0's and used for neighbor analysis in the algorithm), therefore the contours touching the image border will be clipped.

[**INFO**]
http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html