

AI & Robotics

Computer Vision

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.

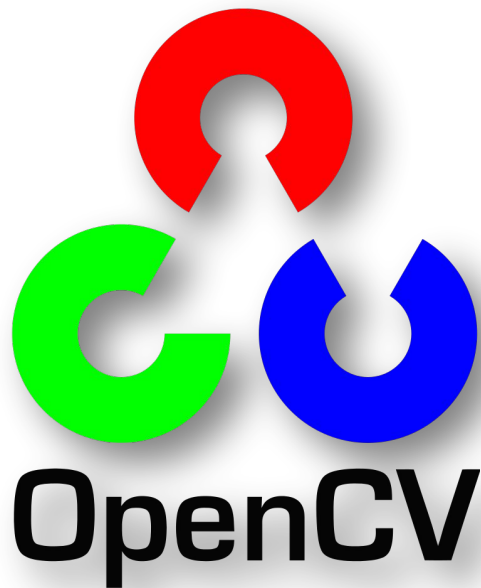
-- Wikipedia

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



[ABOUT](#) [NEWS](#) [EVENTS](#) [RELEASES](#) [PLATFORMS](#) [BOOKS](#) [LINKS](#) [LICENSE](#)

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

Quick Links

- [Online documentation](#)
- [Tutorials](#)
- [User Q&A forum](#)
- [Report a bug](#)
- [Build farm](#)
- [Developer site](#)
- [Wiki](#)

[Donate](#)

Latest news

OpenCV in the DECODED show

Jan 30, 2019

The Decoded Show came by to make a film about OpenCV and its use in film production at Array.com along with mention of its use in Silicon Valley in general and in healthcare apps in particular at Nestle in New York City.

OpenCV 4.0

Nov 20, 2018

Finally! OpenCV 4.0 has been released

Survey 2018

Nov 14, 2018

We've prepared a small survey to learn more about OpenCV usage. Your voice is appreciated!

OpenCV 4.0 release candidate

Nov 12, 2018

Getting closer: OpenCV 4.0 release candidate is out

[E-Mail](#)

[Slack](#)

[GitHub](#)

[Facebook](#)

[Google+](#)

[Wikipedia](#)

[Forum](#)

[IRC #opencv](#)

[SourceForge](#)

[Twitter](#)

[YouTube](#)

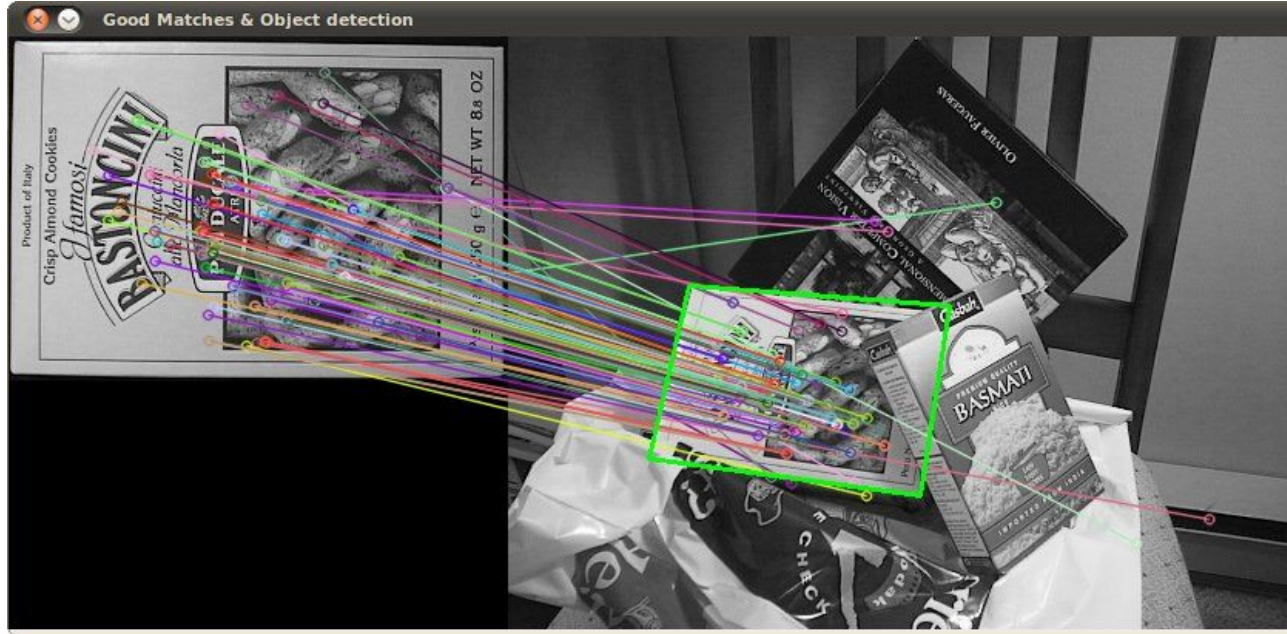
© Copyright 2019, OpenCV team

<http://opencv.org>

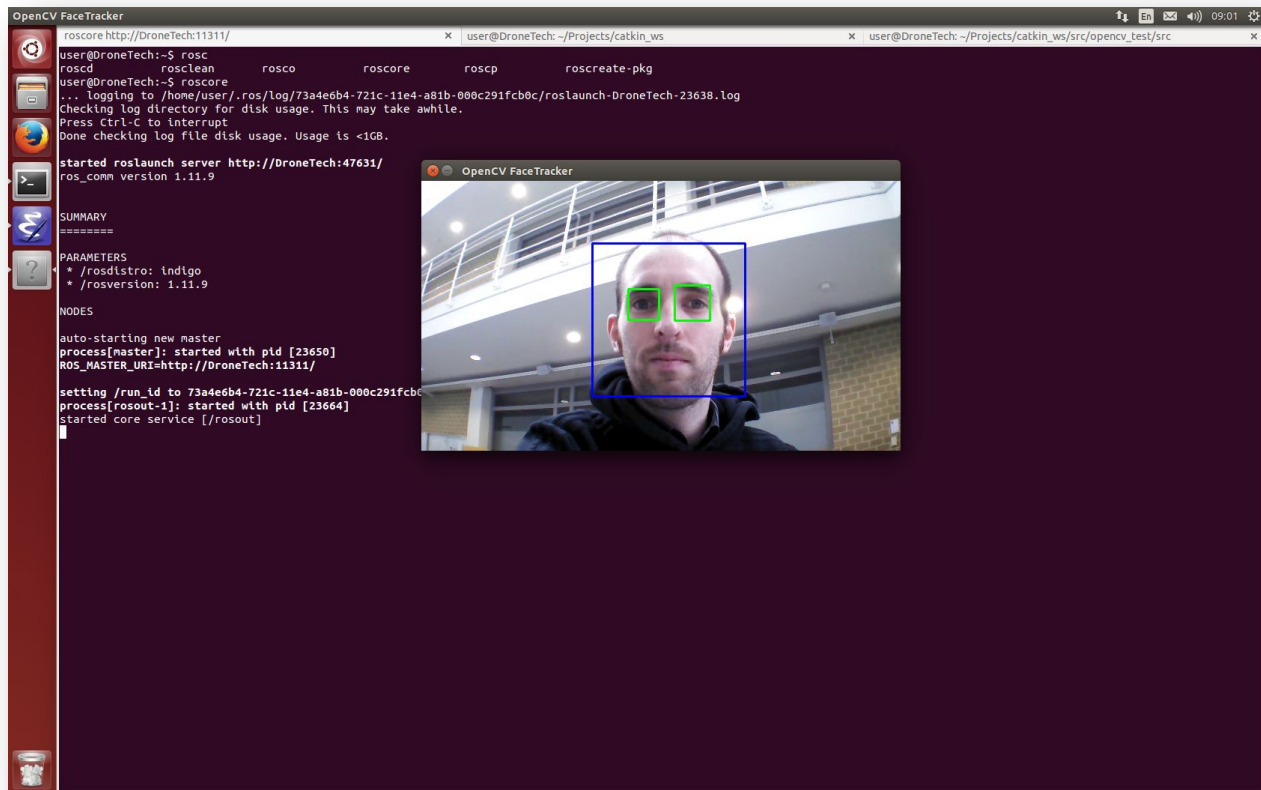
OpenCV Capabilities

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

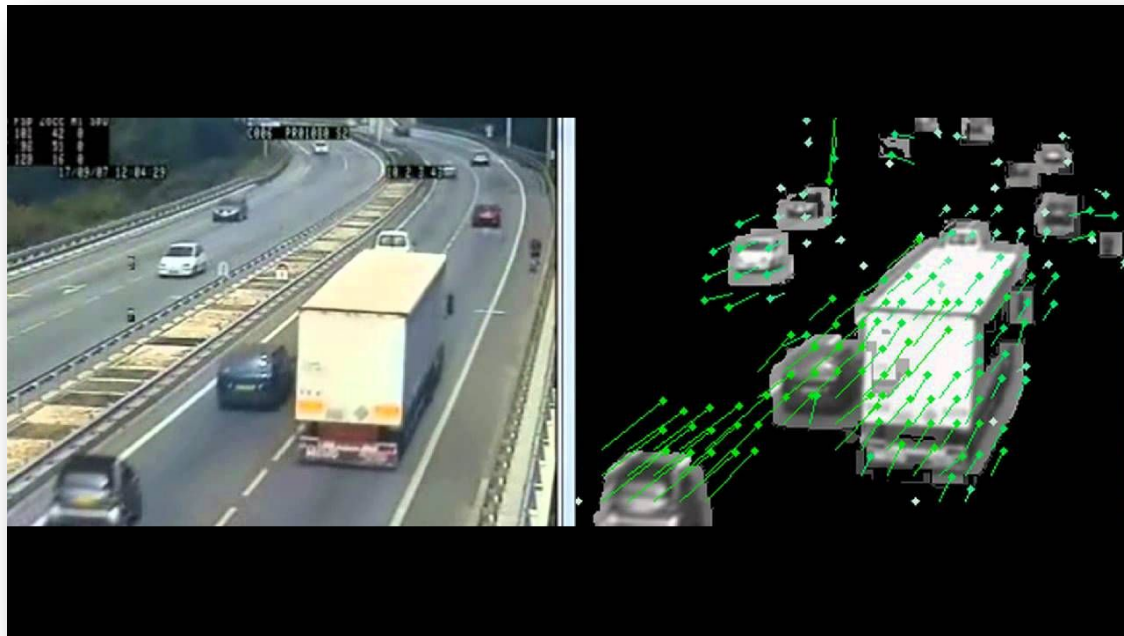
Object Identification



Face Recognition



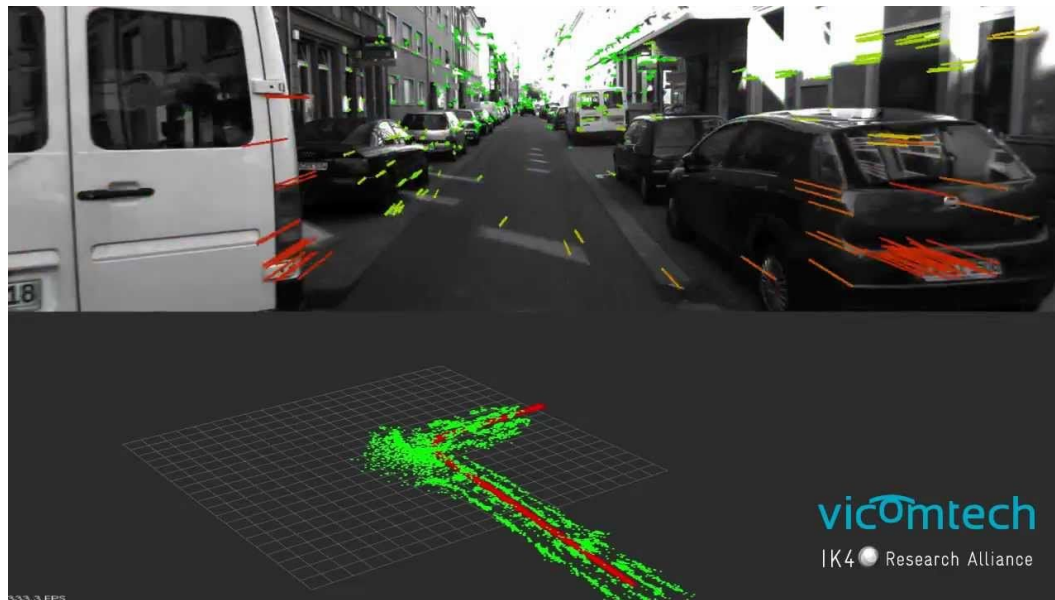
Optical Flow



<https://www.youtube.com/watch?v=4fMzvB4YLMl>

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.

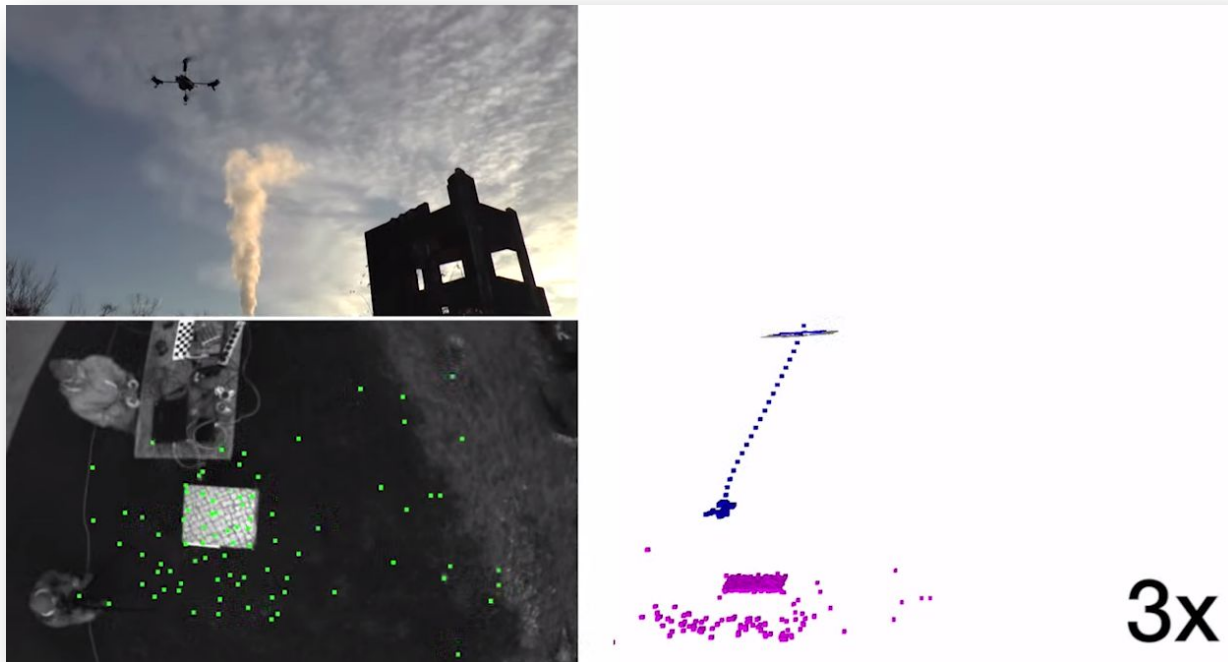
Visual Odometry (Egomotion Estimation)



<https://www.youtube.com/watch?v=ITQGTbrNssQ>

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

Visual Odometry (Egomotion Estimation)

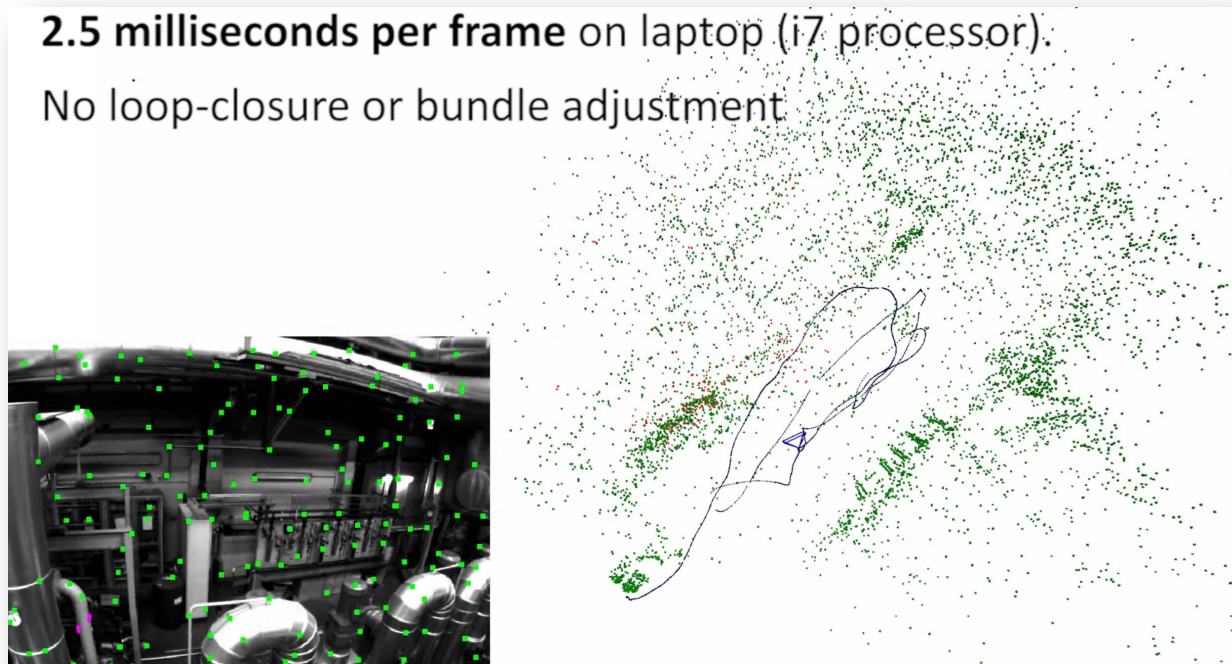


SVO: Fast Semi-Direct Monocular Visual Odometry

[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]

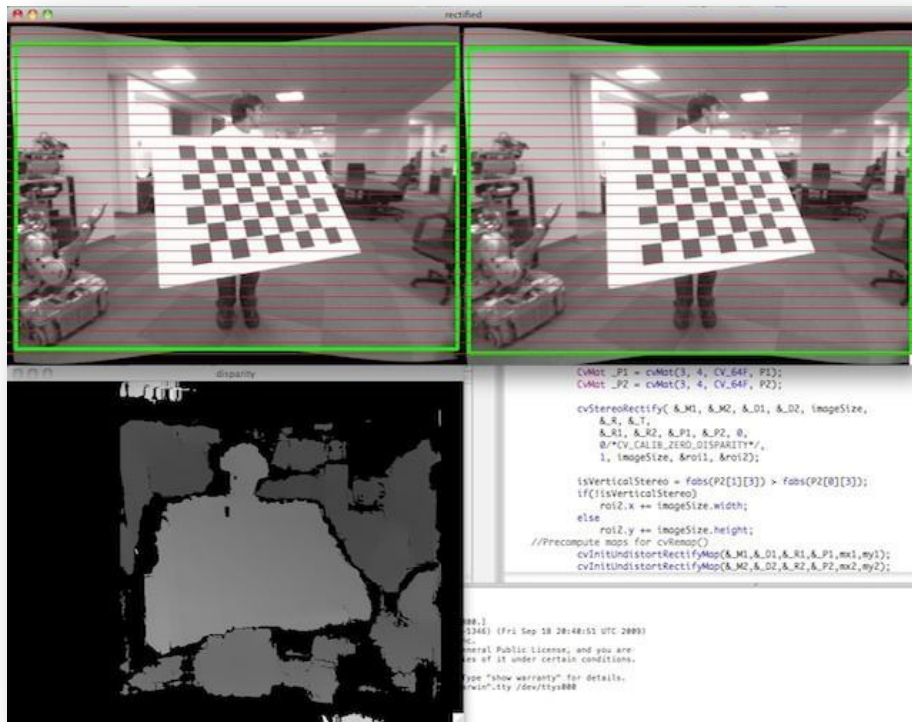
<https://www.youtube.com/watch?v=hR8ug1RTUfA>

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]

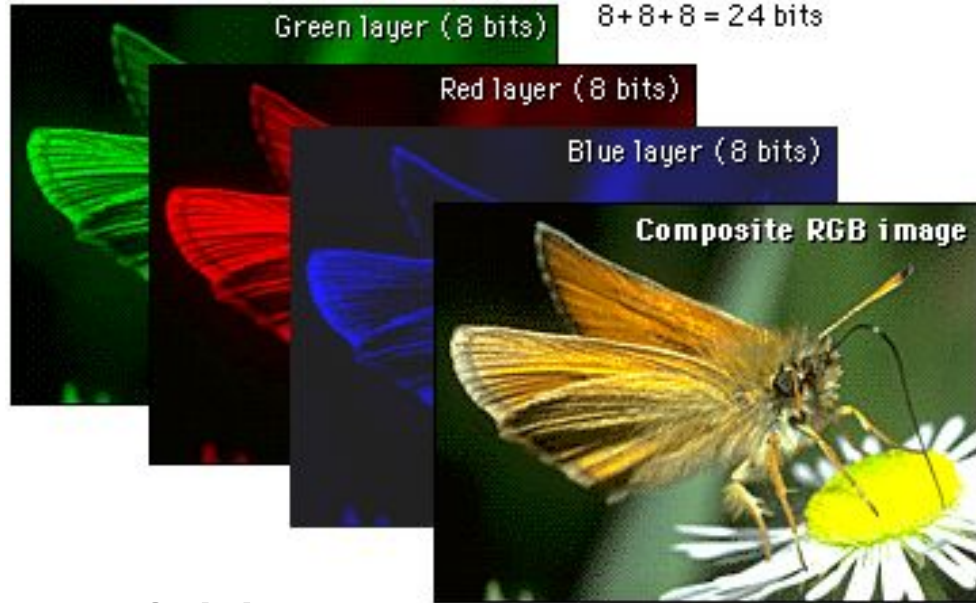
http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

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

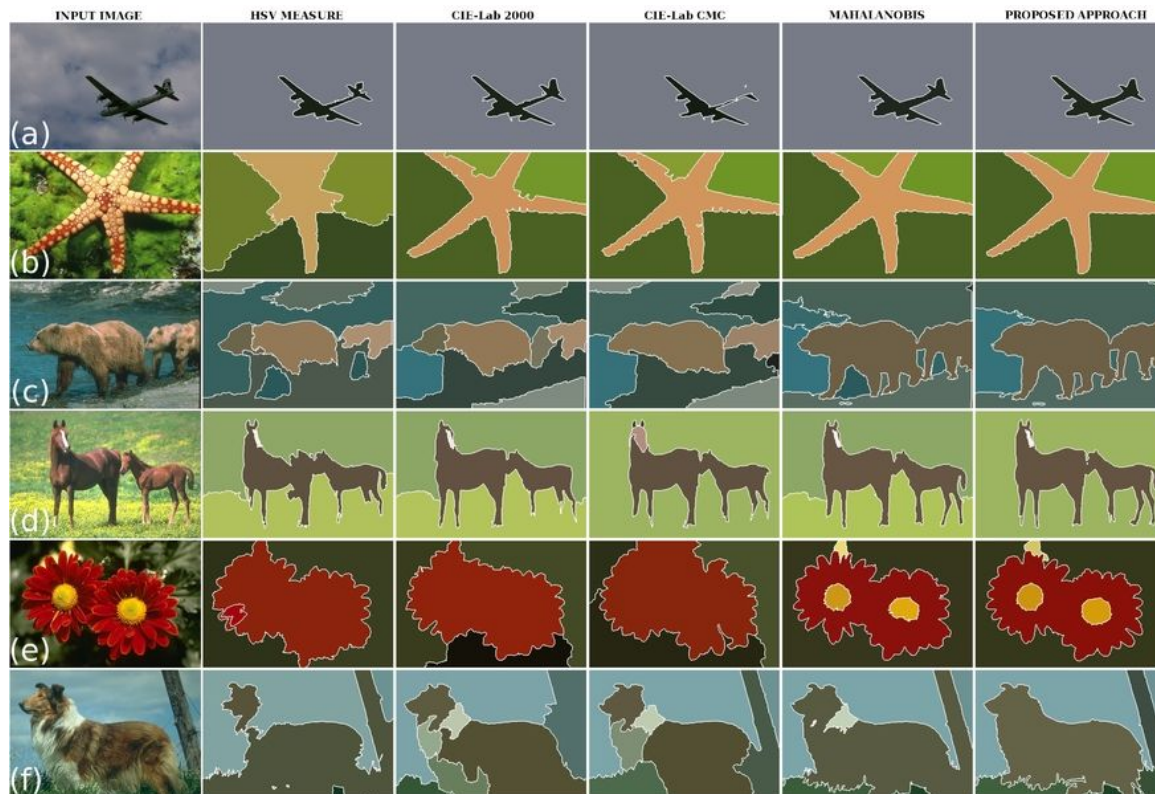
1 blue pixel = 8 bit

8 bit = 0 - 255

Color Detection: RGB

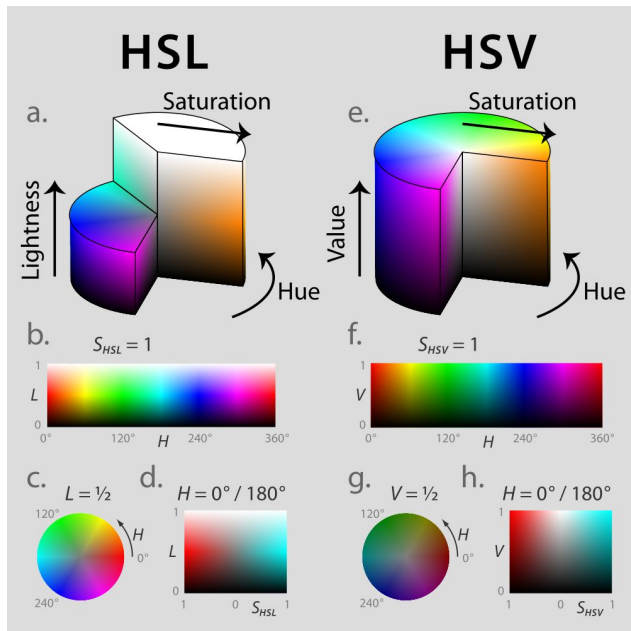
Not the best for color segmentation!

Color segmentation

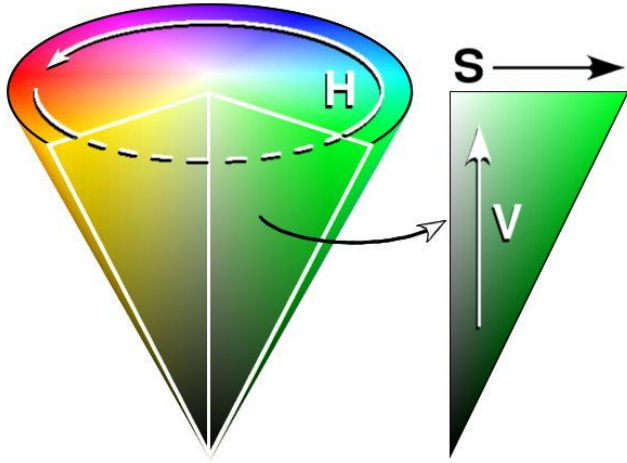


HSV

- 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)



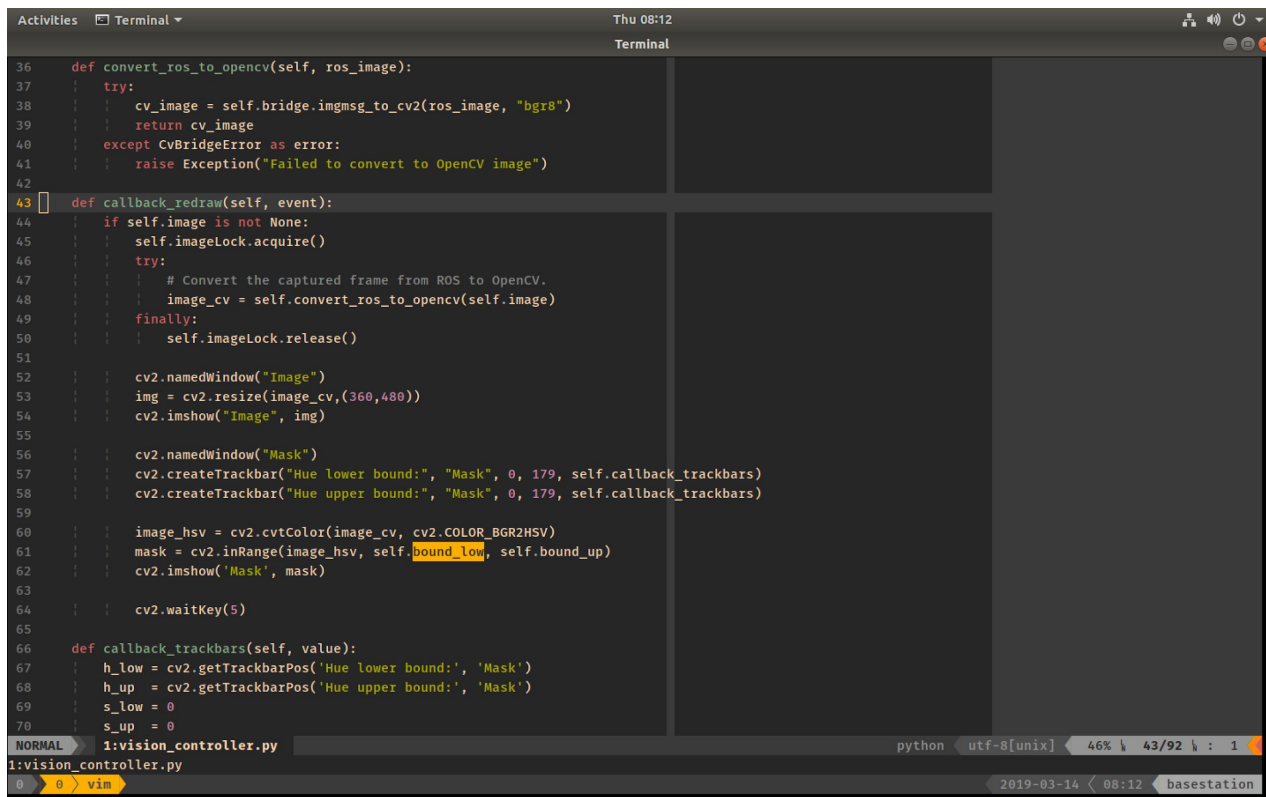
HSV in OpenCV



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

Hue: split @ Red!

Vision Controller Code Overview

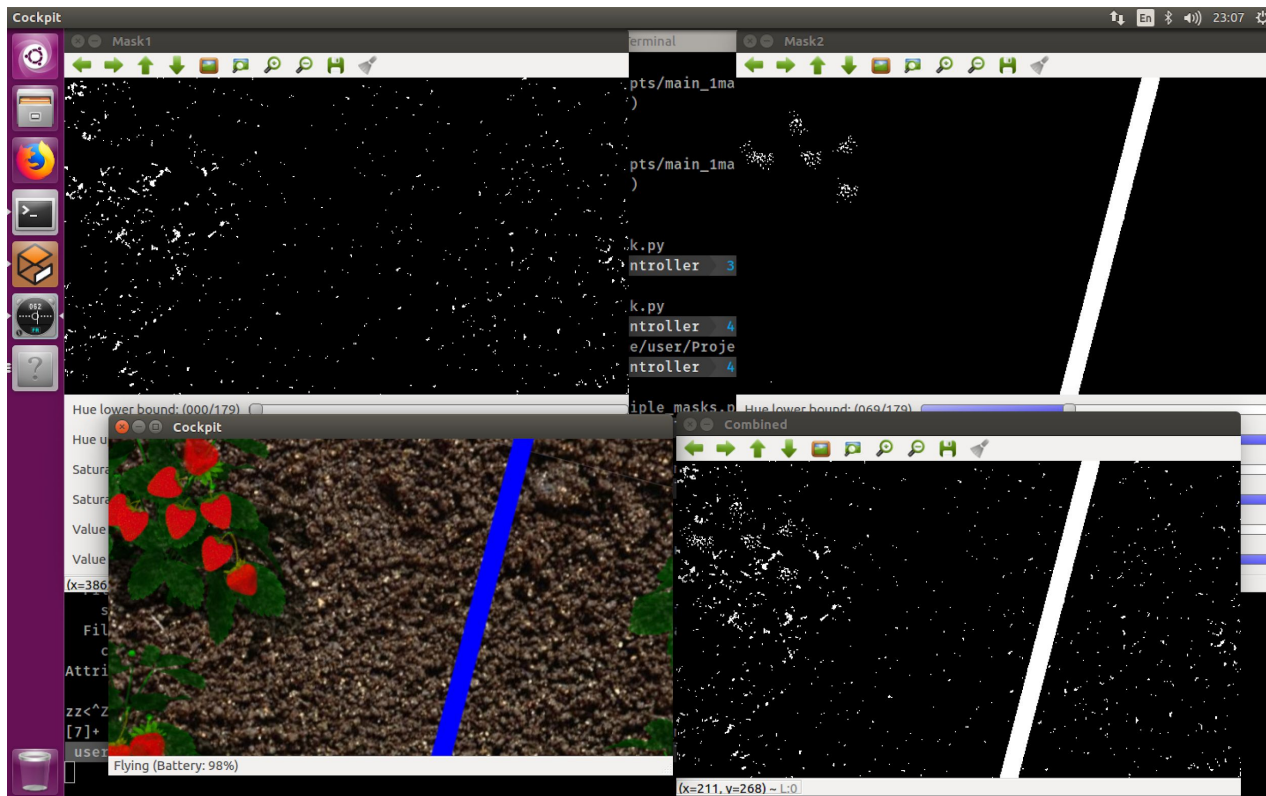


```
Activities  Terminal  Thu 08:12
Terminal
36 def convert_ros_to_opencv(self, ros_image):
37     try:
38         cv_image = self.bridge.imgmsg_to_cv2(ros_image, "bgr8")
39         return cv_image
40     except CvBridgeError as error:
41         raise Exception("Failed to convert to OpenCV image")
42
43 def callback_redraw(self, event):
44     if self.image is not None:
45         self.imageLock.acquire()
46         try:
47             # Convert the captured frame from ROS to OpenCV.
48             image_cv = self.convert_ros_to_opencv(self.image)
49             finally:
50                 self.imageLock.release()
51
52             cv2.namedWindow("Image")
53             img = cv2.resize(image_cv, (360, 480))
54             cv2.imshow("Image", img)
55
56             cv2.namedWindow("Mask")
57             cv2.createTrackbar("Hue lower bound:", "Mask", 0, 179, self.callback_trackbars)
58             cv2.createTrackbar("Hue upper bound:", "Mask", 0, 179, self.callback_trackbars)
59
60             image_hsv = cv2.cvtColor(image_cv, cv2.COLOR_BGR2HSV)
61             mask = cv2.inRange(image_hsv, self.bound_low, self.bound_up)
62             cv2.imshow('Mask', mask)
63
64             cv2.waitKey(5)
65
66 def callback_trackbars(self, value):
67     h_low = cv2.getTrackbarPos('Hue lower bound:', 'Mask')
68     h_up = cv2.getTrackbarPos('Hue upper bound:', 'Mask')
69     s_low = 0
70     s_up = 0
71
NORMAL 1:vision_controller.py python utf-8[unix] 46% 43/92 1
1:vision_controller.py
0 vim 2019-03-14 08:12 basestation
```

Merge Two Masks . . .


```
# Combine masks  
mask = mask1 + mask2  
  
cv2.imshow('Combined', mask)
```

Merge Two Masks . . .



Eroding and Dilating

OpenCV 2.4.12.0 documentation » OpenCV Tutorials » *imgproc* module: Image Processing » [previous](#) | [next](#) | [index](#)



Search

Table Of Contents

- Eroding and Dilating
 - Goal
 - Cool Theory
 - Morphological Operations
 - Dilation
 - Erosion
 - Code
 - Explanation
 - Results

Previous topic
Smoothing Images

Next topic
More Morphology Transformations

This Page
[Show Source](#)

Eroding and Dilating

Goal

In this tutorial you will learn how to:


- Apply two very common morphology operators: Dilation and Erosion. For this purpose, you will use the following OpenCV functions:
 - `erode`
 - `dilate`

Cool Theory

Note: The explanation below belongs to the book **Learning OpenCV** by Bradski and Kaehler.


Morphological Operations

- In short: A set of operations that process images based on shapes. Morphological operations apply a *structuring element* to an input image and generate an output image.
- The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :
 - Removing noise
 - Isolation of individual elements and joining disparate elements in an image.
 - Finding of intensity bumps or holes in an image
- We will explain dilation and erosion briefly, using the following image as an example:



Dilation

- This operations consists of convoluting an image *A* with some kernel (*B*), which can have any shape or size, usually a square or circle.
- The kernel *B* has a defined *anchor point*, usually being the center of the kernel.
- As the kernel *B* is scanned over the image, we compute the maximal pixel value overlapped by *B* and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*). Take as an example the image above. Applying dilation we can get:



[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]`, `hierarchy[i][1]`, `hierarchy[i][2]`, and `hierarchy[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 (x_1, y_1) and (x_2, y_2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, $\max(\text{abs}(x_1 - x_2), \text{abs}(y_2 - y_1)) = 1$.
- **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.

