OpenCV

2025

# INFO 703

# Master Advanced Mechatronics

**Luc Marechal**

UNIVERSITÉ SAVOIE MONT BLANC

POLYTECH ANNECY-CHAMBÉRY

ROS

## Lecture 6

## Vision - OpenCV

# Overview

- Interface camera in ROS
- Acquire images from the camera.
- Subscribe to an image camera topic
- Perform simple image processing with OpenCV

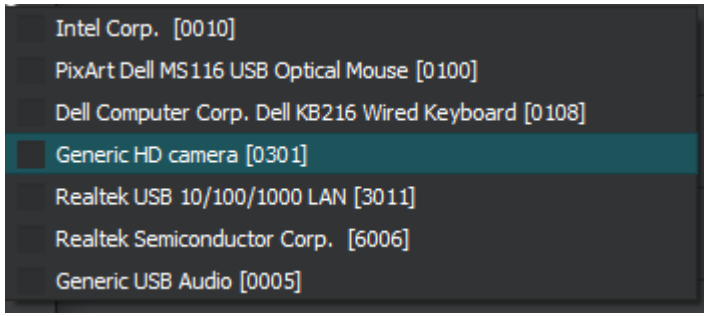
# Packages Related to USB Camera

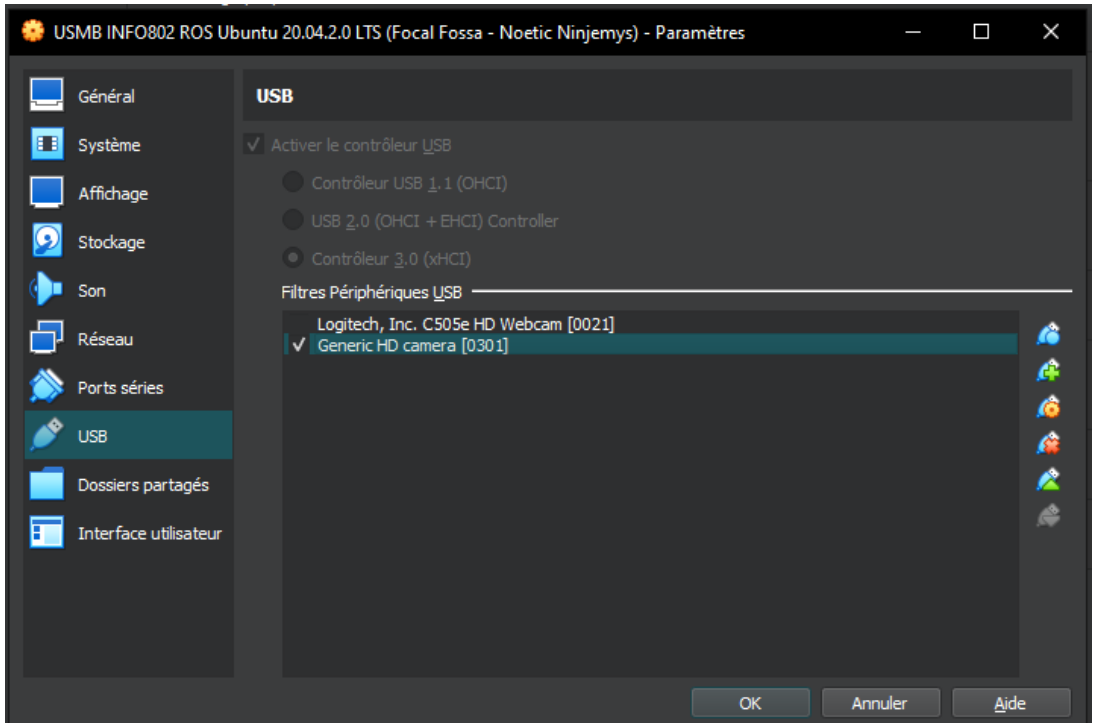
- **ibuvc-camera**: interface package for operating cameras with the UVC standard.
- **uvc-camera**: convenient package with relatively detailed camera settings. (ideal for stereo camera configuration).
- **usb-cam**: very simple camera driver for Bosch.
- **freenect-camera**, **openni-camera**, **openni2-camera**: for depth cameras like Kinect or Xtion. These sensors include a color camera, (also called RGB-D cameras). These packages are required to use their color images.
- **camera1394**: driver for cameras using IEEE 1394 standard FireWire.
- **prosilica-camera**: used in AVT's prosilica camera, (widely used for research purposes).
- **camera-calibration**: camera calibration package that applies OpenCV's calibration feature. Many camera-related packages require this package.

# Installing USB webcams in VirtualBox

In configuration >> USB : Add USB filter 

Then select Generic HD camera



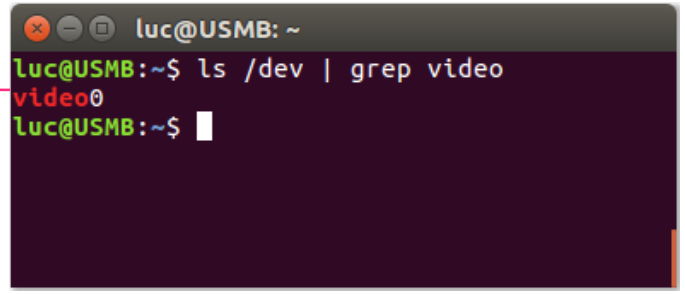



# Interfacing USB webcams in ROS

Find the video devices present on the system

```
> ls /dev | grep video
```

Found a device numbered #0




If no video found, click on Virtualbox menu
Devices > Webcams > Integrated Webcams

Known issue: https://doc.ubuntu-fr.org/v4l1

Test camera with Cheese.
if not installed:

```
> sudo apt install cheese
> cheese
```

If the camera is not detected with Cheese.
Then try guvcview

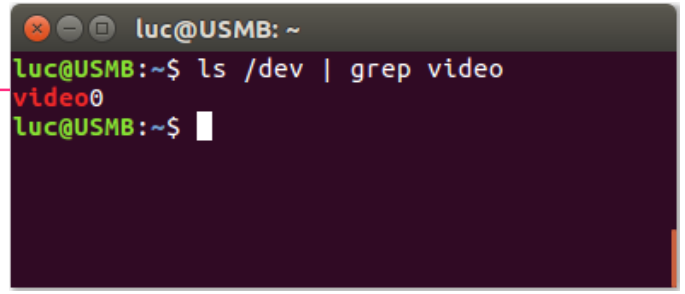
```
> sudo apt install guvcview
> guvcview
```

# Interfacing USB webcams in ROS

Find the video devices present on the system

```
> ls /dev | grep video
```

Found a device numbered #0




If no video found, click on Virtualbox menu
Devices > Webcams > Integrated Webcams

Known issue: https://doc.ubuntu-fr.org/v4l1

Test camera with Cheese.
if not installed:

```
> sudo apt-get install cheese
```

usb_cam driver package installation

```
> sudo apt-get install ros-noetic-usb-cam
```

uvc_camera driver package installation

```
> sudo apt-get install ros-noetic-uvc-camera
```

Image Related Package Installation

```
> sudo apt-get install ros-noetic-image-*
> sudo apt-get install ros-noetic-rqt-image-view
```

# Interfacing USB webcams in ROS

## Running uvc_camera node

```
> roscore
> rosrun uvc_camera uvc_camera_node
```

## Running usb_cam node

```
> roscore
> rosrun usb_cam usb_cam_node
```

## Verify Topic Message

```
> rostopic list
```

images are published in multiple ways, compressed and uncompressed (useful to send images to other ROS nodes and occupying little space)

```
/camera_info
/usb_cam/image_raw
/usb_cam/image_raw/compressed
/usb_cam/image_raw/compressed/parameter_descriptions
/usb_cam/image_raw/compressed/parameter_updates
/usb_cam/image_raw/compressedDepth
/usb_cam/image_raw/compressedDepth/parameter_descripti
/usb_cam/image_raw/compressedDepth/parameter_updates
/usb_cam/image_raw/theora
/usb_cam/image_raw/theora/parameter_descriptions
/usb_cam/image_raw/theora/parameter_updates
/rosout
/rosout_agg
```

```
/camera_info
/image_raw
/image_raw/compressed
/image_raw/compressed/parameter_descriptions
/image_raw/compressed/parameter_updates
/image_raw/compressedDepth
/image_raw/compressedDepth/parameter_descriptions
/image_raw/compressedDepth/parameter_updates
/image_raw/theora
/image_raw/theora/parameter_descriptions
/image_raw/theora/parameter_updates
/rosout
/rosout_agg
```

# Interfacing USB webcams in ROS

Visualize the image in another window with image view node

```
> rosrun image_view image_view image:=/usb_cam/image_raw
```

image:=<image topic> [image transport type]

Visualize with rqt_image_view Node

```
> rosrun rqt_image_view rqt_image_view image:=/usb_cam/image_raw
```

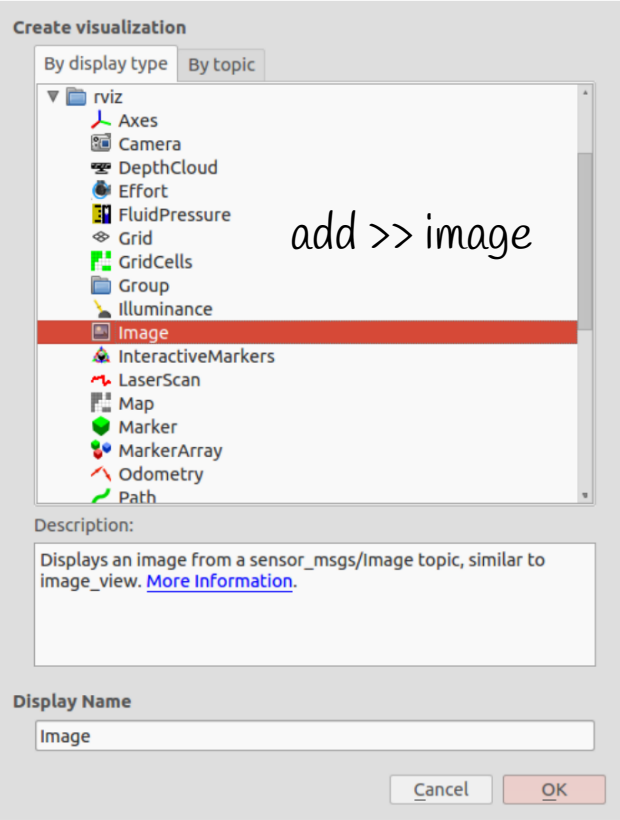
Republish the image in an other format on an other topic

```
> rosrun image_transport republish compressed in:=/usb_cam/image_raw
[output format] out:=/usb_cam/image_raw/republished
```

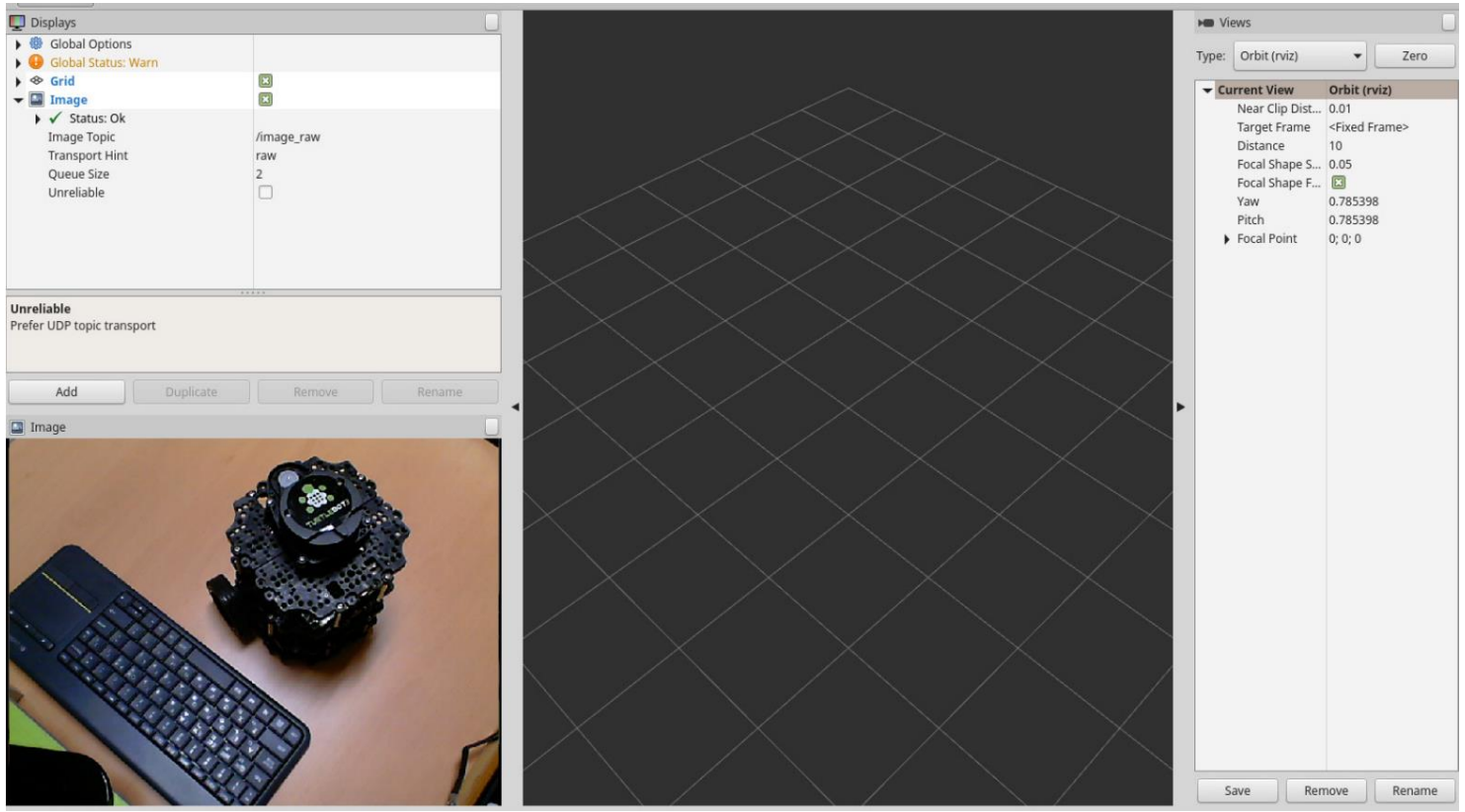
# Interfacing USB webcams in ROS

## Visualize with rviz

```
> rviz
```




select the topic /usb_cam/image_raw




Change the value of [Image Topic] in the [Image] option to '/usb_cam/image_raw'

# Interfacing USB webcams in ROS

The usb_cam-test.launch file can launch the USB cam driver with the necessary parameters

```
> roslaunch usb_cam usb_cam-test.launch
```

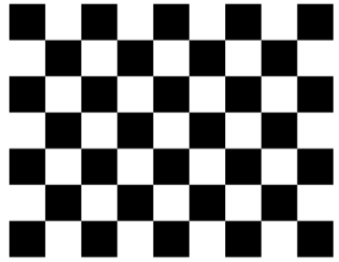
```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

you will get this warning message about the camera calibration, '[WARN] [1423194481.257752159]: Camera calibration file /home/xxx/.ros/camera_info/camera. yaml not found.'

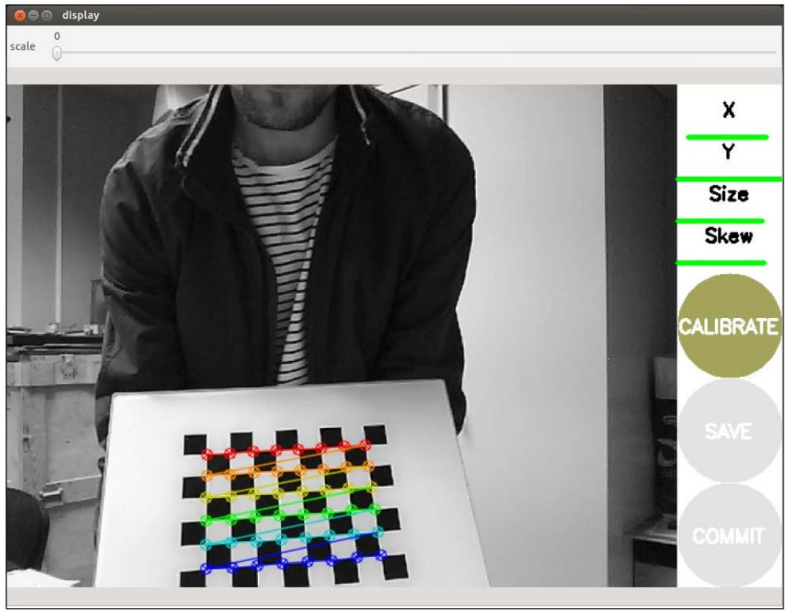
because the calibration file is missing. You can ignore this for now.

# ROS camera calibration

You need a large checkerboard with known dimensions. Calibration uses the interior vertex points of the checkerboard



http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf




Getting the dependencies and compiling the driver

```
> rosdep install camera_calibration
```

Running the Calibration Node

```
> rosrun camera_calibration cameracheck.py --size 8x6
monocular:=/forearm image:=image_rect
```

**More Info**

http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

# OpenCV

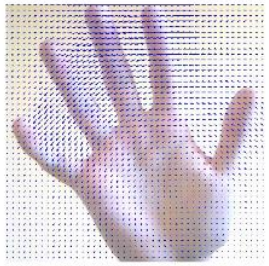
- OpenCV is a library of programming functions mainly aimed at real-time computer vision.
- Stands for the Open Source Computer Vision Library
- Developed by Intel in 1999.
- Supported by Willow Garage.
- Cross Platform
- Free for use under ‘’Open-Source BSD License’’.

# OpenCV Algorithm Modules Overview

Image Processing

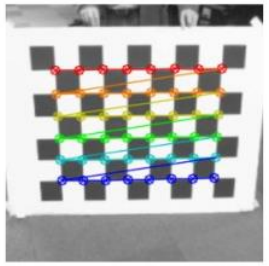
Transforms

Fitting

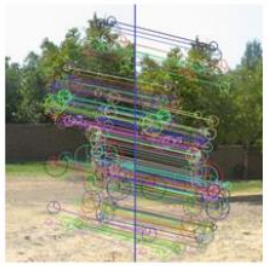


Optical Flow
Tracking

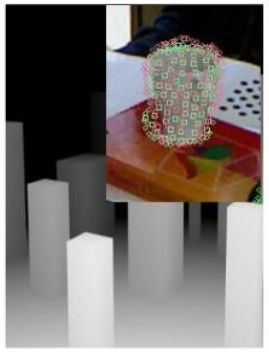
Segmentation



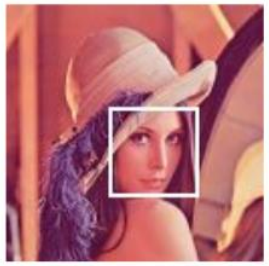
Calibration



Features
VSLAM



Depth, Pose
Normals, Planes,
3D Features



Object recognition
Machine learning

Computational
Photography

**Content source**
G. Bradsky
Willow garage

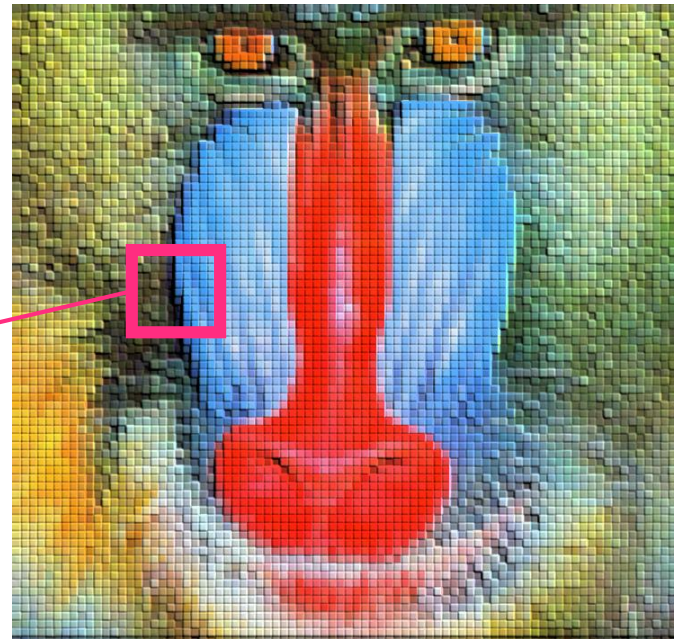
# Image Processing

## What is an Image?

- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

but the camera sees this

```
186 203 127 127 139
185 206 136 136 150
255  30 101 133 170
256 200 139 138  40
257 179 122 179  30
129 165 123 167  36
```



**Content source**

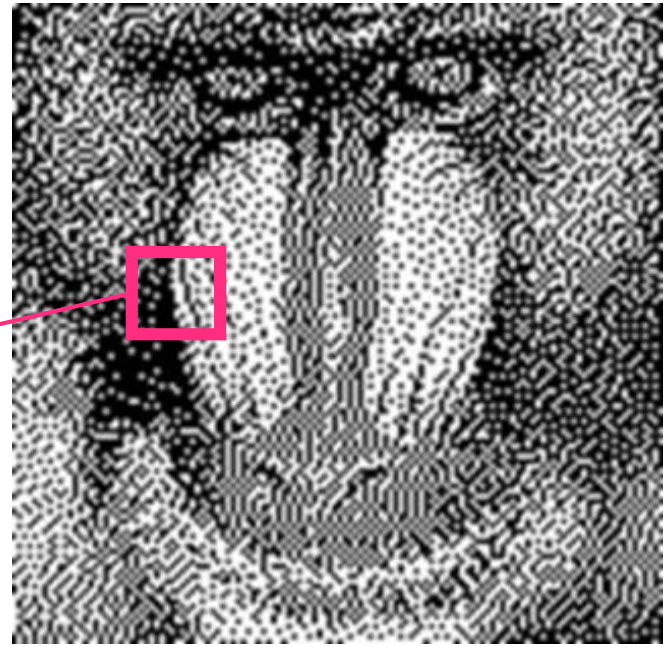
Rahul Sukthankar - The Robotics Institute, Carnegie Mellon

# Image Processing

## What is an Image?

- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits (black=0, white=1)
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

```
1 1 0 0 1 1 0 1
1 1 1 0 1 0 1 1
1 1 1 1 0 0 1 1
1 1 1 1 0 0 0 1
1 1 1 1 1 0 0 0
```



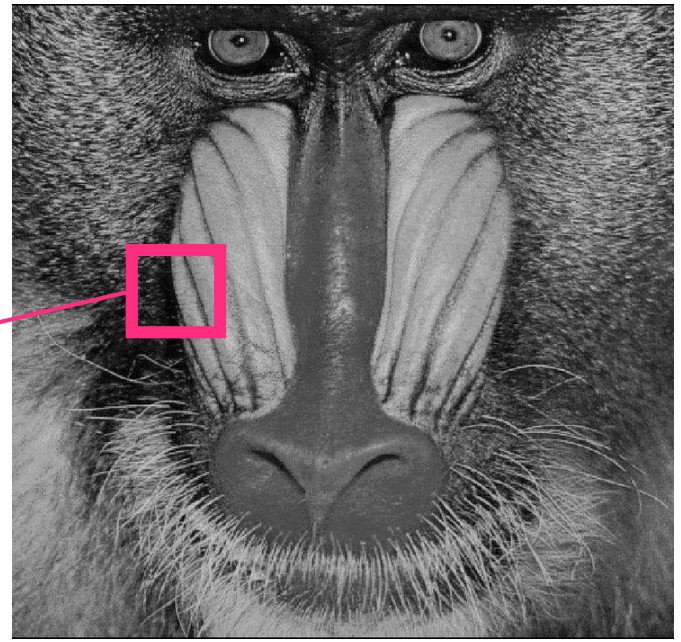
**Content source**
Rahul Sukthankar - The Robotics Institute, Carnegie Mellon

# Image Processing

## What is an Image?

- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

```
186 203 127 127 139
185 206 136 136 150
255  30 101 133 170
256 200 139 138  40
257 179 122 179  30
129 165 123 167  36
```



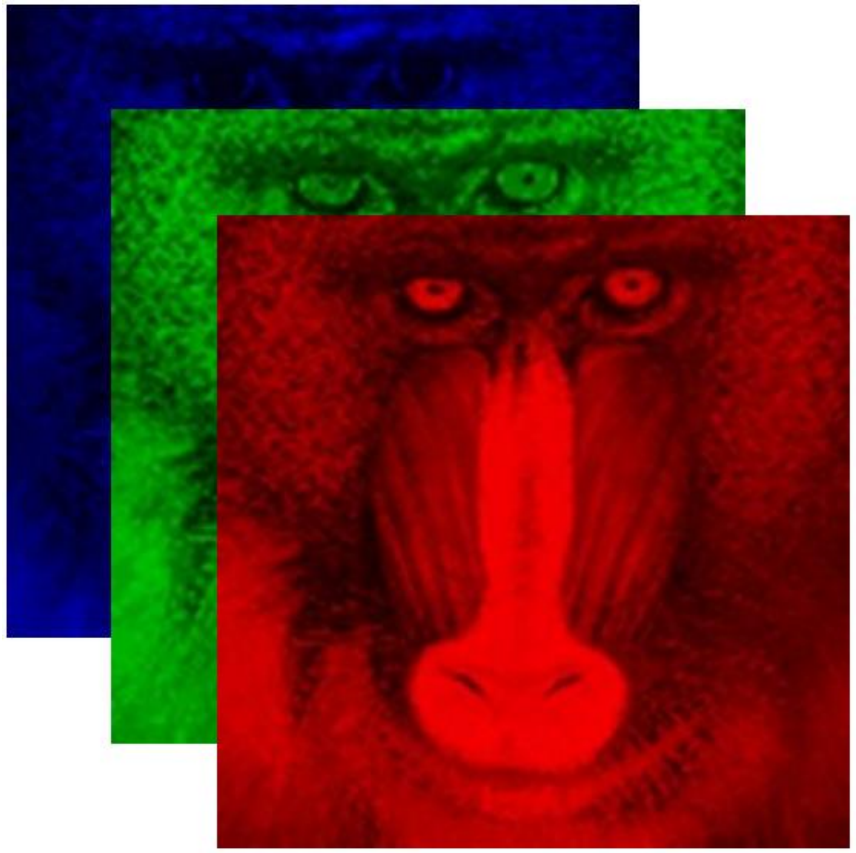
**Content source**

Rahul Sukthankar - The Robotics Institute, Carnegie Mellon

# Image Processing

## What is an Image?

- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR !!!
  - Sometimes includes Alpha



**Content source**
Rahul Sukthankar - The Robotics Institute, Carnegie Mellon

# OpenCV Image Processing

## Colors

- **BGR**: The default color. Normal 3 channel color
- **HSV**: Hue is color, Saturation is amount, Value is lightness. 3 channels
- **GRAYSCALE**: Gray values, Single channel

OpenCV requires that images to be in BGR or Grayscale in order to be shown

# HSV vs RGB

The most obvious approach is to find the red, green, blue (RGB) value of a yellow image pixel and filter for nearby RGB values. Unfortunately, filtering on RGB values turns out to be a surprisingly poor way to find a particular color in an image, since the raw RGB values are a function of the overall brightness as well as the color of the object.

Slightly different lighting conditions would result in the filter failing to perform as intended. Instead, a better technique for filtering by color is to transform RGB images into hue, saturation, value (HSV) images.

The HSV image separates the RGB components into hue (color), saturation (color intensity), and value (brightness). Once the image is in this form, we can then apply a threshold for hues near yellow to obtain a binary image in which pixels are either true (meaning they pass the filter) or false (they do not pass the filter).

**More Info**

Programming Robots with ROS, O'Reilly

# OpenCV Image Processing

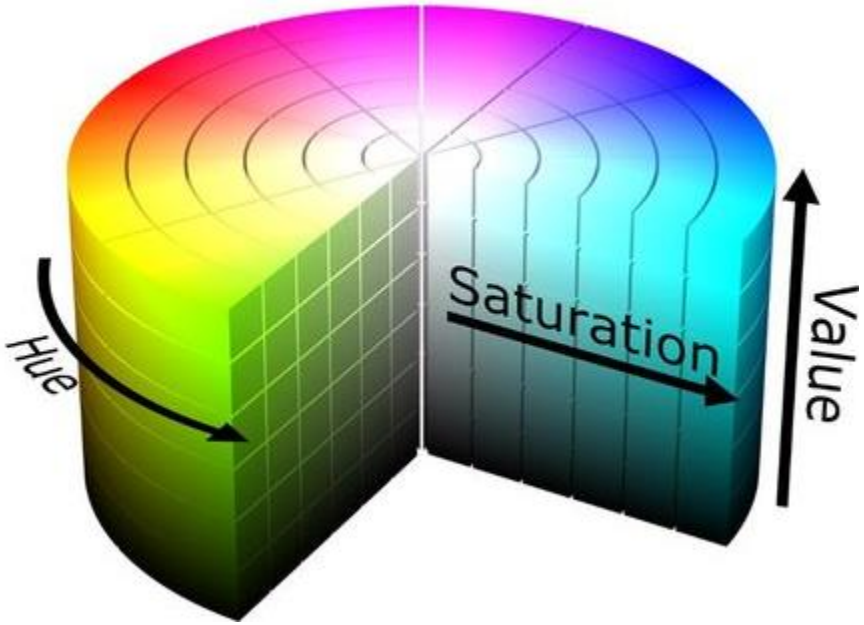
## Colors

**In OpenCv for HSV:**

Hue range is [0 , 179]

Saturation range is [0 , 255]

Value range is [0 , 255]



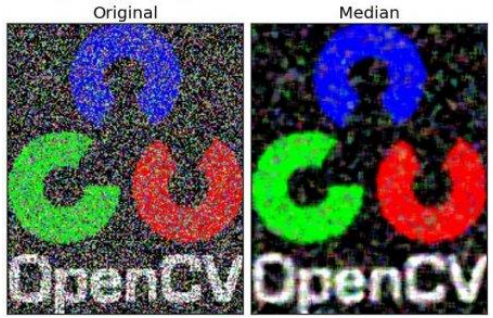

Find HSV values to track

```
> green = np.uint8([[[0,255,0 ]]])
> hsv_green = cv.cvtColor(green,cv.COLOR_BGR2HSV)
> print(hsv_green)
[[[ 60 255 255]]]
```

take [H-10, 100,100] and [H+10, 255, 255] as lower bound and upper bound respectively

# OpenCV Image Processing

## Basic Computer Vision Techniques

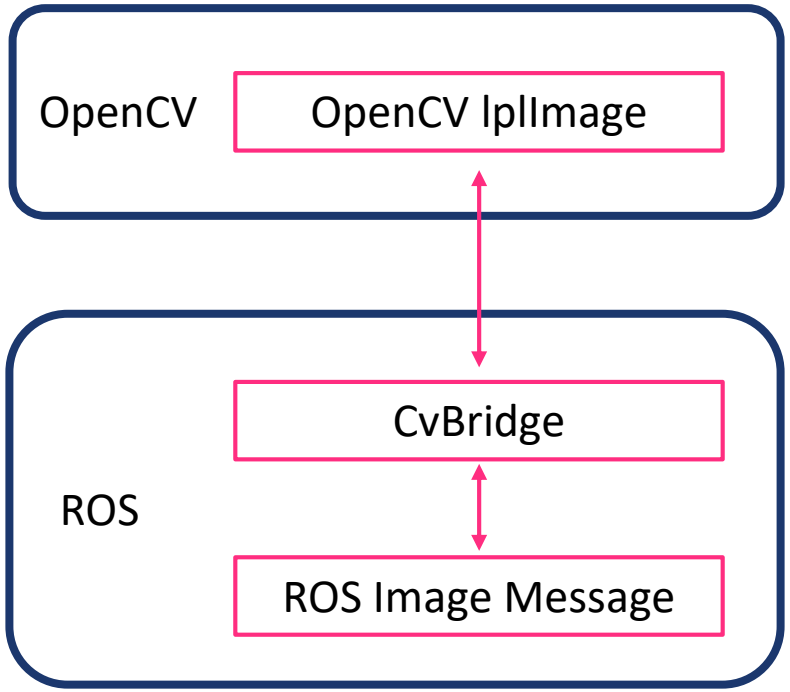



- Filtering
  - Remove noise from image
- Segmentation
  - Partition image into groups of pixels
  - Similarity can be decided based on intensity, color, pattern
- Feature detection
  - Extract interesting parts from the image
  - Edge detection, corner detection
  - For example in navigation systems it may prove useful to extract only floor lines from an image

**Content source**
Roi Yehoshua
- Bar Ilan University

# OpenCV in ROS

- OpenCV uses image native format from taken from the Intel Image Processing Library (IplImage)
- ROS passes images in its own *sensor_msgs/Image* message
- *cv_bridge* is a ROS package that provides functions to convert between ROS *sensor_msgs/Image* messages and the objects used by OpenCV



# OpenCV in ROS

Step-by-step procedures:

1. Subscribe the images from the camera driver from the topic /usb_cam/image_raw
   (or different topic depending on the driver used )

2. Convert the ROS images to the OpenCV image type using CvBridge

3. Process the OpenCV image using its APIs and find the edges on the image

4. Convert the OpenCV image type of the edge detection to the ROS image messages and publish into the topic /edge_detector/processed_image

# OpenCV in ROS

## 2. Converting ROS image messages to OpenCV images

To convert a ROS image message into an `cv::Mat`, module `cv_bridge.CvBridge` provides the following function:

Afficher/masquer les numéros de lignes

```
1 from cv_bridge import CvBridge
2 bridge = CvBridge()
3 cv_image = bridge.imgmsg_to_cv2(image_message, desired_encoding='passthrough')
```

The input is the image message, as well as an optional `encoding`. The encoding refers to the destination `cv::Mat` image.

If the default value of "passthrough" is given, the destination image encoding will be the same as the image message encoding. Image encodings can be any one of the following OpenCV image encodings:

- 8UC[1-4]
- 8SC[1-4]
- 16UC[1-4]
- 16SC[1-4]
- 32SC[1-4]
- 32FC[1-4]
- 64FC[1-4]

For popular image encodings, `CvBridge` will optionally **do color or pixel depth conversions as necessary**. To use this feature, specify the `encoding` to be one of the following strings:

- `mono8`: `CV_8UC1`, grayscale image
- `mono16`: `CV_16UC1`, 16-bit grayscale image
- `bgr8`: `CV_8UC3`, color image with blue-green-red color order
- `rgb8`: `CV_8UC3`, color image with red-green-blue color order
- `bgra8`: `CV_8UC4`, BGR color image with an alpha channel
- `rgba8`: `CV_8UC4`, RGB color image with an alpha channel

Note that `mono8` and `bgr8` are the two image encodings expected by most OpenCV functions.

# OpenCV in ROS

Installation

```
> sudo apt install libopencv-dev
```

Alternatively

```
> sudo apt install ros-kinetic-opencv3
```

Already installed on your Virtual Machine image

# Creating a Simple Image Subscriber Node (Python)

## Writing the Node

### Create package

```
> cd ~/catkin_ws/src/
> catkin_create_pkg vision_tutorials rospy roscpp
cv_bridge
```

### Edit script

```
> cd ~/catkin_ws/src/vision_tutorials
> mkdir scripts
> sudo code simple_camera_view_node.py
```

### Make script executable

```
> cd ~/catkin_ws/src/vision_tutorials/scripts
> chmod +x simple_camera_view_node.py
```

### Make package and source environment

```
> cd ~/catkin_ws
> catkin_make
> source ~/catkin_ws/devel/setup.bash
```

```python
#! /usr/bin/python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

bridge = CvBridge()

def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
    cv2.imshow("Usb camera video", cv2_img)
    cv2.waitKey(3)

def simple_camera_view_node():
    rospy.init_node('image_subscriber')
    image_topic = "/usb_cam/image_raw"

    rospy.Subscriber(image_topic, Image, image_callback)
    rospy.spin()

if __name__ == '__main__':
    simple_camera_view_node()
```

https://github.com/LucMarechal/ROS_Lectures/blame/3e8e45a6e527217b7a2f9c787f6523a5ae1d22b6/vision_tutorials/simple_camera_view_node.py

# Creating a Simple Image Subscriber Node (Python)

## Examining the Node

ROS Image message
ROS Image message -> OpenCV Image converter
Instantiates a *cvBridge* object
Converts ROS Image message to OpenCV2
Creates a window and display the image
Introduces a delay of n milliseconds while rendering images to windows
Defines the image topic
Setup *subscriber* and define its *callback** 
Spin until ctrl + C

```
#! /usr/bin/python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

bridge = CvBridge()

def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
    cv2.imshow("Usb camera video", cv2_img)
    cv2.waitKey(3)

def simple_camera_view_node():
    rospy.init_node('image_subscriber')
    image_topic = "/usb_cam/image_raw"

    rospy.Subscriber(image_topic, Image, image_callback)
    rospy.spin()

if __name__ == '__main__':
    simple_camera_view_node()
```

***Recall**: *callback* = function passed as an argument to an other function

# Creating a Simple Image Subscriber Node (Python)

## Running the Node

Launch the webcam driver

```
> roslaunch usb_cam usb_cam-test.launch
```

Run the `simple_camera_view_node` node

```
> rosrun vision_tutorials simple_camera_view_node.py
```

# Creating a Simple Image Processing Node (Python)

## Writing the Node

### Edit script

```
> cd ~/catkin_ws/src/vision_tutorials/scripts
> sudo code simple_image_processing_node.py
```

### Make script executable

```
> cd ~/catkin_ws/src/scripts
> chmod +x simple_image_processing_node.py
```

### Make package and source environment

```
> cd ~/catkin_ws
> catkin_make
> source ~/catkin_ws/devel/setup.bash
```

This node detects a specific color in the image and remove the others

https://github.com/LucMarechal/ROS_Lectures/blob/765e41aacd5c6428909f0e17e0e41b728617a580/vision_tutorials/simple_image_processing.py

# Creating a Simple Image Processing Node (Python)

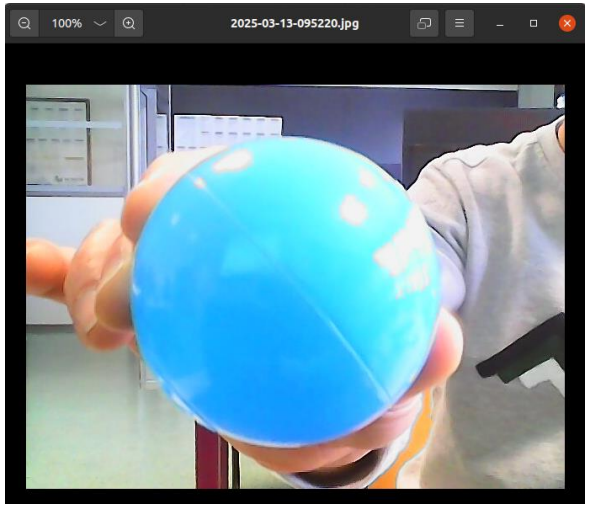
## Writing the Node

```python
#! /usr/bin/python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np

bridge = CvBridge()

def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
    frame = np.array(cv2_img, dtype=np.uint8)
    color_image = color_detection(frame)
    cv2.waitKey(3)

def color_detection(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    blue_lower = np.array([90,50,50],np.uint8)
    blue_upper = np.array([120,255,255],np.uint8)
    mask = cv2.inRange(hsv, blue_lower, blue_upper)
    cv2.imshow("image mask",mask)
    res = cv2.bitwise_and(frame,frame, mask= mask)
    cv2.imshow("color",res)

    return res


def simple_image_processing():
    rospy.init_node('image_subscriber')
    image_topic = "/usb_cam/image_raw"

    rospy.Subscriber(image_topic, Image, image_callback)
    rospy.spin()



if __name__ == '__main__':
    simple_image_processing()
```

# Defining the color code

## Defining the color code

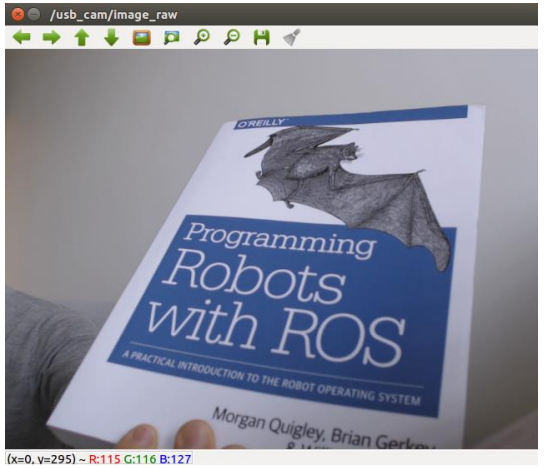
Take a snapshot of your object with Cheese

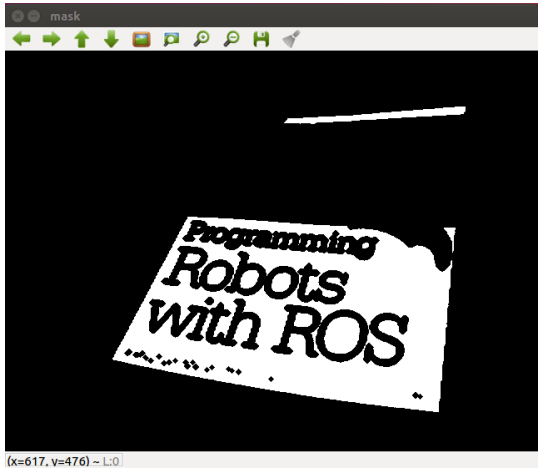



Open the image with Gimp

# Creating a Simple Image Processing Node (Python)

## Examining the Node




### Color detection

Converts images from BGR to HSV

Defining the range of Blue color

Finding the range blue colour in the image

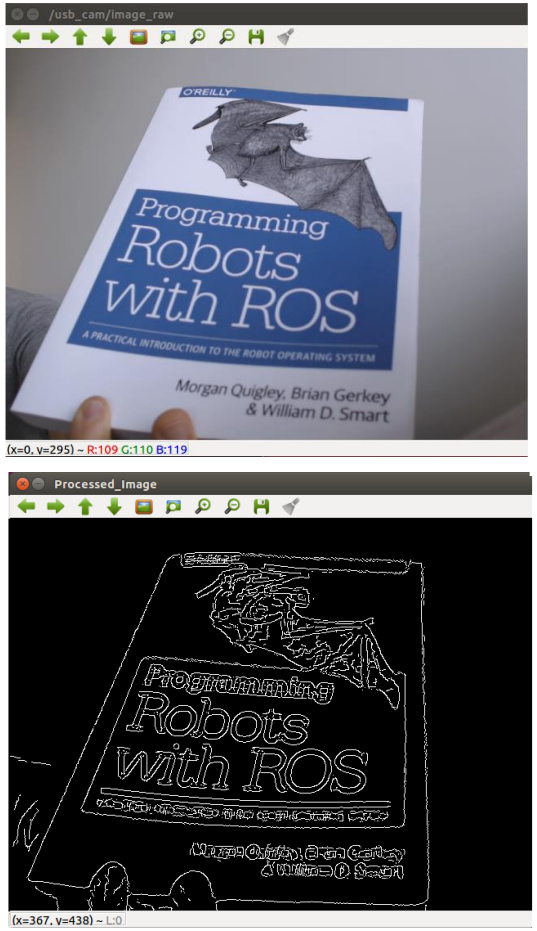
The bitwise and of the frame and mask is done so that only the blue coloured objects are highlighted and stored in res

code snippet

```
def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
    frame = np.array(cv2_img, dtype=np.uint8)
    color_image = color_detection(frame)
    cv2.waitKey(3)


def color_detection(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    blue_lower = np.array([90,50,50],np.uint8)
    blue_upper = np.array([120,255,255],np.uint8)
    mask = cv2.inRange(hsv, blue_lower, blue_upper)
    cv2.imshow("image mask",mask)

    res = cv2.bitwise_and(frame,frame, mask= mask)
    cv2.imshow("color",res)

    return res
```

# Creating a Simple Image Processing Node (Python)

## Examining the Node




Edges detection

Converts images from BGR to GREYscale

Computes edges using Canny edges filter

code snippet

```
def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
    frame = np.array(cv2_img, dtype=np.uint8)
    color_image = edges_detection(frame)
    cv2.waitKey(3)


def edges_detection(frame):
    grey = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.blur(grey, (7,7))
    edges = cv2,Canny(blurred, 15.0, 30.0)

    return edges
```

# Assignement

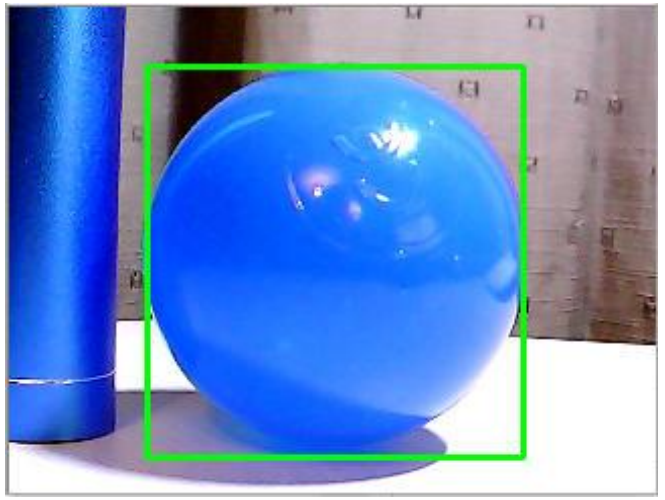
## Object Detection and Tracking

### Ball tracking

https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/

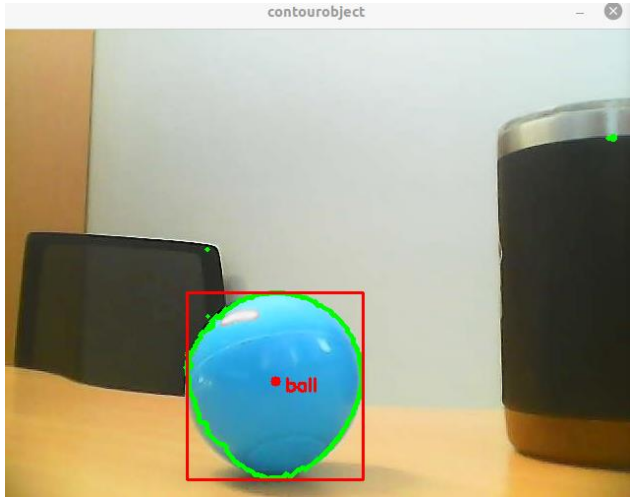
http://www.booppey.com/booppey/opencv-ball-detection-color-based/

https://docs.opencv.org/master/d2/d96/tutorial_py_table_of_contents_imgproc.html



# Ball tracking

## Examining the Node

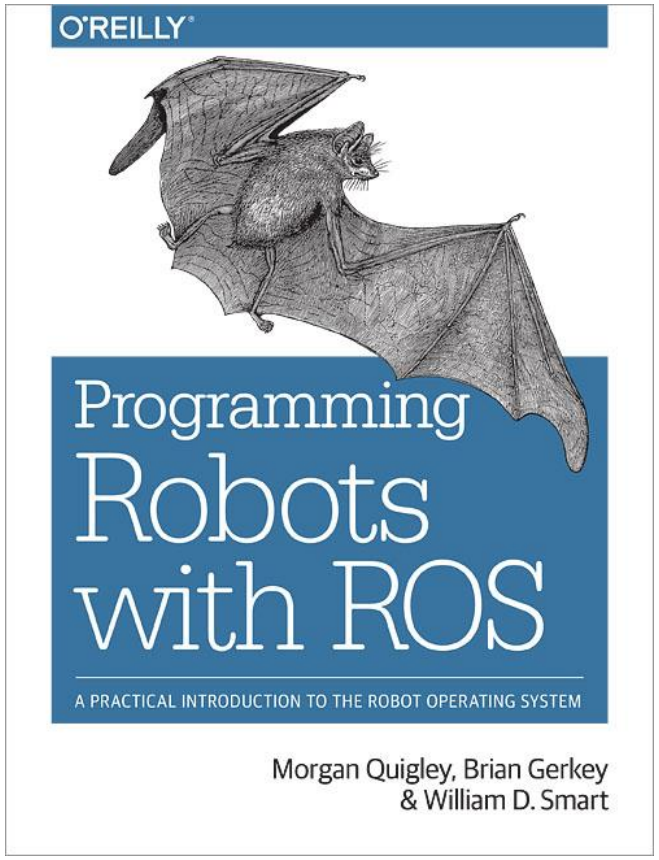



```python
#!/usr/bin/python3
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2, numpy

bridge = CvBridge()    # Creates a CvBridge object to handle image with OpenCv

def image_callback(msg):
    cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")      # Converting ROS sensor_msgs/Image messages into OpenCV image format
    frame = numpy.array(cv2_img, dtype=numpy.uint8)
    color_detection(frame)
    cv2.waitKey(3)


def color_detection(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    blue_lower = numpy.array([90,50,50],numpy.uint8)
    blue_upper = numpy.array([120,255,255],numpy.uint8)
    mask = cv2.inRange(hsv, blue_lower, blue_upper)    # produces a binary (black and white) image
    #blurred = cv2.GaussianBlur(mask,(7,7),0)          # produces a blurred binary (black and white) image
    #cv2.imshow("blur",blurred)

    contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Find the moments of the first contour (the bigest one)
    rospy.loginfo("Number of Contour found = %f\n", len(contours))


    frame_ct = cv2.drawContours(frame, contours, -1, (0,255, 0), 3)

    min_coutour_area = 500

    for c in contours:
        if (cv2.contourArea(c) > min_coutour_area):
            M = cv2.moments(c)
            if M['m00'] > 0:
                cx = int(M['m10']/M['m00'])
                cy = int(M['m01']/M['m00'])
                cv2.circle(frame, (cx,cy), 5, (0,0,255), -1)
                cv2.putText(frame, "ball", (cx+10,cy+10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,0,255), 2)
                x, y, w, h = cv2.boundingRect(c)
                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

    cv2.imshow("contourobject",frame_ct)
    cv2.imshow("image mask",mask)



def simple_image_processing():
    rospy.init_node('image_subscriber')
    image_topic = "/usb_cam/image_raw"

    rospy.Subscriber(image_topic, Image, image_callback)
    rospy.spin()



if __name__ == '__main__':
    simple_image_processing()
```

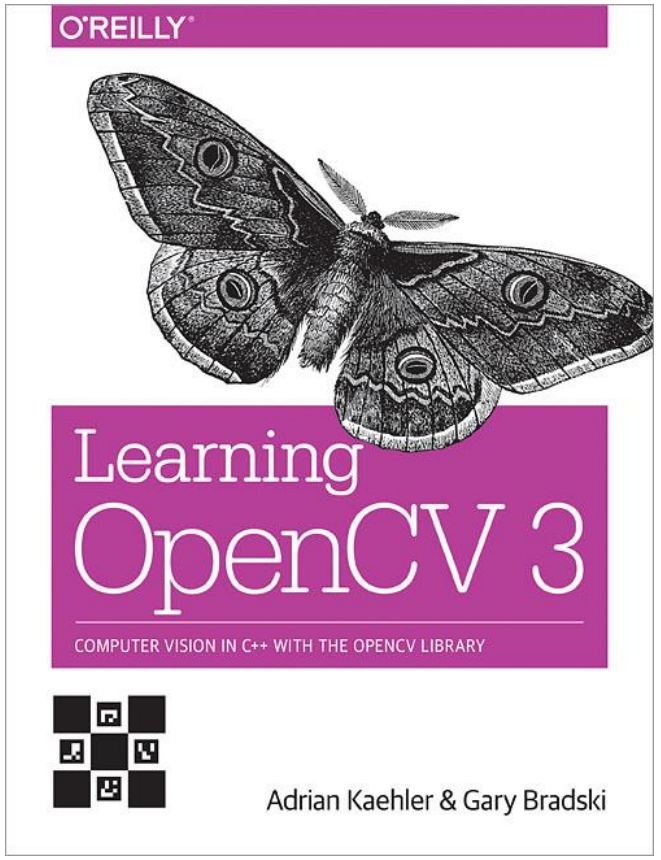
# Further References

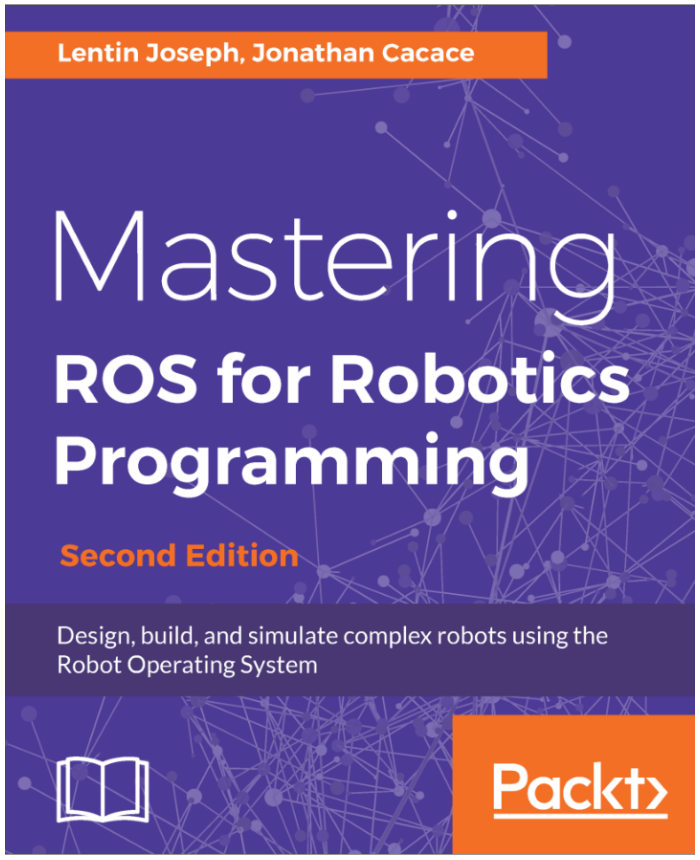
- OpenCV Image processing
  - https://docs.opencv.org/master/d2/d96/tutorial_py_table_of_contents_imgproc.html
- ROS Cheat Sheet
  - https://www.clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
  - https://kapeli.com/cheat_sheets/ROS.docset/

# Relevant books




Chapter 12




Chapter 10