









Featuring **ESC**
The Embedded Systems Conference


Use OpenCV Computer Vision to Test Your Embedded System

Matt Liberty
Jetperch LLC

#eelive

Produced by EE Times









 ESC
EMBEDDED SYSTEMS
CONFERENCE
  BLACK HAT
EMBEDDED
  INTERNET OF
THINGS
  HARDWARE
STARTUP
  ANDROID
ENGINEERING
  FPGA
ENGINEERING
  SUPER C++
TUTORIAL
  UBM
Tech



Featuring **ESC**
The Embedded Systems Conference

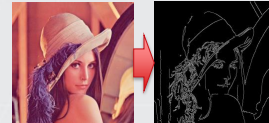
Agenda

- Introduction
 - Computer Vision
 - OpenCV & Python
- Examples
 - Recognize an icon on a screen
 - Recognize when an LED is illuminated
 - Measure Android touchscreen latency
- Tips and pitfalls
- Conclusion and questions

 ESC
EMBEDDED SYSTEMS
CONFERENCE
  BLACK HAT
EMBEDDED
  INTERNET OF
THINGS
  HARDWARE
STARTUP
  ANDROID
ENGINEERING
  FPGA
ENGINEERING
  SUPER C++
TUTORIAL
  UBM
Tech

Computer Vision

- Acquire
 - Single images
 - Video (time-related sequence of images)
- Process
 - Classical filtering
 - Morphological transformations
 - Geometric and perspective transformations
- Analyze & understand
 - Motion estimation
 - Pattern and feature detection
 - Object recognition and object tracking
 - Machine learning



OpenCV Modules

- | | |
|--|---|
| • core. The Core Functionality | • gpu. GPU-accelerated Computer Vision |
| • imgproc. Image Processing | • photo. Computational Photography |
| • highgui. High-level GUI and Media I/O | • stitching. Images stitching |
| • video. Video Analysis | • nonfree. Non-free functionality |
| • calib3d. Camera Calibration and 3D Reconstruction | • contrib. Contributed/Experimental Stuff |
| • features2d. 2D Features Framework | • legacy. Deprecated stuff |
| • objdetect. Object Detection | • ocl. OpenCL-accelerated Computer Vision |
| • ml. Machine Learning | • superres. Super Resolution |
| • flann. Clustering and Search in Multi-Dimensional Spaces | |

See <http://docs.opencv.org/>



OpenCV

- Open Source Computer Vision Library (OpenCV)
- Written in C++ as a cross-platform, portable library
- Runs on Linux, Windows, Mac OS X, Android
- Fast: can use MMX, SSE, OpenCL & CUDA (GPU support)
- Includes 2500+ algorithms spanning acquisition, processing, analysis and machine learning
- Liberal BSD license
- Large active community, originally started by Intel
- Professionally maintained by itseez

<http://opencv.org/>



Python & OpenCV

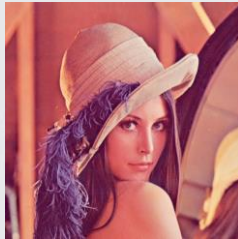
- Python bindings (cv2) are well-supported by the OpenCV project
- Get speed close to native C++ with flexibility of Python
- Allows for faster iteration time
- Access to numpy and scipy libraries for other numerical processing

http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html



Quick OpenCV Python Example

```
import cv2
img = cv2.imread('lena.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edge = cv2.Canny(gray, 1000, 3000, apertureSize=5)
cv2.imwrite('lena_edge.jpg', edge)
```



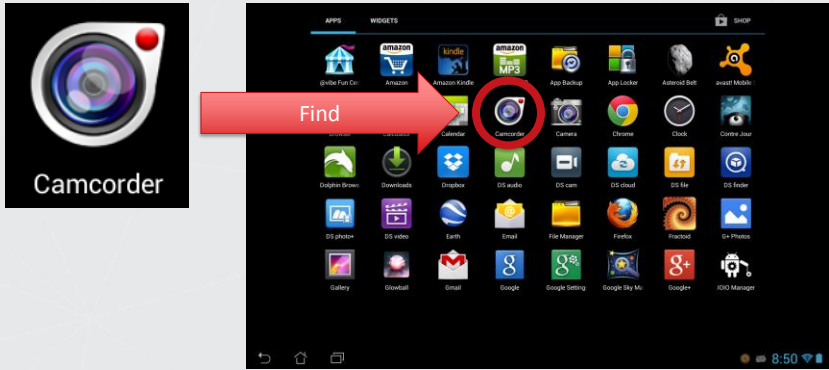
Agenda

- Introduction
 - Computer Vision
 - OpenCV & Python
- Examples
 - Recognize an icon on a screen
 - Recognize when an LED is illuminated
 - Measure Android touchscreen latency
- Tips and pitfalls
- Conclusion and questions

Source code available on github:
https://github.com/mliberty1/cv2_product_test

EE Live! Featuring **esc**
The Embedded Systems Conference

Detect an Icon



The slide illustrates the process of finding a specific icon within a set of application icons. On the left, a large icon labeled 'Camcorder' is shown. On the right, a grid of various application icons is displayed. A red arrow labeled 'Find' points from the 'Camcorder' icon to its corresponding icon in the grid, which is circled in red.

esc
EMBEDDED SYSTEMS CONFERENCE

BLACK AND WHITE
CONVERSION

OPEN SOURCE
SOFTWARE

CONNECTION
ANALYSIS

POWER
MANAGEMENT

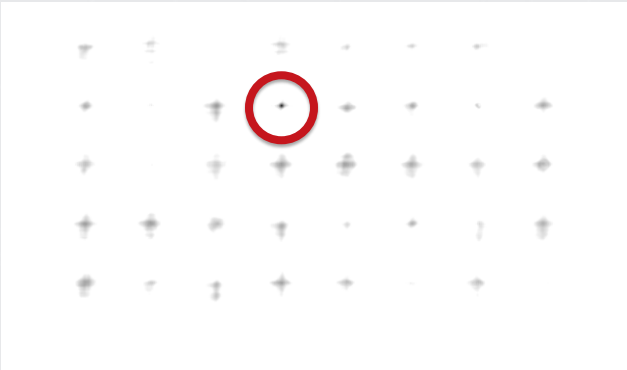
SYSTEMS
ARCHITECTURE

UBM
Tech

EE Live! Featuring **esc**
The Embedded Systems Conference

Detect an Icon (Exact)

- Search pixel by pixel computing the sum total difference between the icon and the overlapping image area.



The slide illustrates the process of detecting an icon exactly. It shows a grid of application icons. One icon, which is a red circle with a black dot in the center, is circled in red.

esc
EMBEDDED SYSTEMS CONFERENCE

BLACK AND WHITE
CONVERSION

OPEN SOURCE
SOFTWARE

CONNECTION
ANALYSIS

POWER
MANAGEMENT

SYSTEMS
ARCHITECTURE

UBM
Tech

Detect an Icon (Exact)

- Search pixel by pixel computing the sum total difference between the icon and the overlapping image area.

```
icon = cv2.imread(icon_filename)
image = cv2.imread(image_filename)
result = cv2.matchTemplate(image, icon, cv2.TM_SQDIFF_NORMED)
idx = np.argmin(result)
metric = np.ravel(result)[idx]
if metric > threshold:
    print('ERROR: %s' % metric)
    sys.exit(1)
coord = np.unravel_index(idx, result.shape)[-1::-1]
print('SUCCESS: %s at %s' % (metric, coord))
```

- bin/find_icon_exact.py cv2_product_test/test/android.png
cv2_product_test/test/android_camcorder.png

Detect an Icon (Inexact)

- Extract “features” from the icon and the image.
- Search for any image area where the icon features match the image features.
- The feature extraction and search can be mathematically designed to work with rotation and perspective distortion!

Detect an Icon (Inexact)



```
~/dev/opencv-2.4.8/samples/python2/find_obj.py --feature=sift-flann cv2_product_test/test/android_camcorder.png cv2_product_test/test/android.png
```

Detect an Icon (Inexact)

- Many different feature extraction methods exists, and many of the most robust are covered by patents.
- OpenCV API requires creating a detector, a matcher, searching through the results, and finding the homography to convert the icon's coordinates into the image coordinates.
- Or, just use the simplified API from our example:

```
cv2_product_test.find.find(icon, image, spec)
```

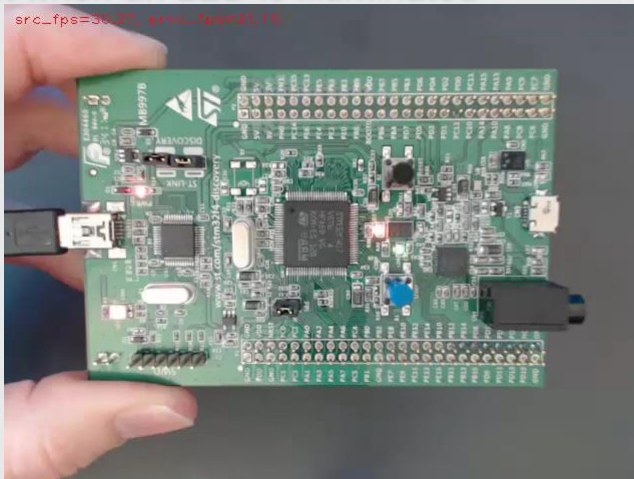

Detect when an LED is Illuminated

- Get the next frame of video as an image
- Detect board using features (same as previous example)
 - Draw the region on the image
- Detect LED (geometric relationship to board)
 - Draw the region on the image
- Detect pixel intensity inside LED region
 - If sufficient, draw "ON" in the image
 - Print all state changes to stdout
- Display the image to the GUI

stdout:

```
1393172372.270: LED ON
1393172372.376: LED OFF
1393172372.545: LED ON
1393172372.587: LED OFF
1393172372.625: LED ON
1393172372.783: LED OFF
1393172372.843: LED ON
1393172372.941: LED OFF
```

Detect when an LED is Illuminated



Detect when an LED is illuminated

- DetectLed class
 - rectangular_region_callback: Select board and LED regions
 - onDraw: process each image
 - onKeyPress: Enable capture of images and video to file
 - ledValue: Compute an LED metric for on/off detection:

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
quad = quad.reshape(-1, 2)
quad -= np.float32([x, y])
mask = np.zeros(img.shape, np.uint8)
quad = np.int32(quad.reshape(1, -1, 2))
cv2.drawContours(mask, quad, 0, (1), cv2.cv.CV_FILLED)
# Use percentile matching for statistical robustness
values = img[mask != 0]
V2 = 0 if len(values) < 5 else np.percentile(values, 90)
```

Measure Android touchscreen latency

- Objective: determine the time from touch to corresponding onscreen graphics on an Android tablet
- Run app that displays a magenta dot at touch location
- Image processing
 - Get the next frame of video as an image
 - Crop to the active area
 - Take negative and threshold each color
 - Detect dot as center of "green" (negative of magenta)
 - Detect finger as "white" and tip of finger as the smallest y-axis value
 - Record dot location and finger location and draw on image
 - Display latency onscreen, if available
- Latency computation: least squares fit of latency, x-axis offset and y-axis offset.

EE Live! Featuring **ESC**
The Embedded Systems Conference

Measure Android touchscreen latency

src_fps=29.76, proc_fps=29.94

esc ESC Embedded Systems Conference

BLACK AND WHITE

OPEN SOURCE

TECHNICAL

ANDROID

FREE

CC BY

UBM Tech

EE Live! Featuring **ESC**
The Embedded Systems Conference

Measure Android touchscreen latency: Image processing

- separate_targets
 - Use each color as a separate image
 - Use [cv2.erode](#) to eliminate noise ([tutorial](#))
- finger_image_to_coordinates
 - Use `numpy.max` and `numpy.where` to find the first row with a non-zero pixel (the most vertical point on the finger)
- touchscreen_image_to_coordinates
 - Use [cv2.moments](#) to find the centroid of the touchscreen dot image

esc ESC Embedded Systems Conference

BLACK AND WHITE

OPEN SOURCE

TECHNICAL

ANDROID

FREE

CC BY

UBM Tech

Measure Android touchscreen latency: Latency computation

- Select valid data
 - Only use detected points and ignore others
 - Presume that not too much fingertip data is missing
- Use [scipy.optimize.leastsq](#) to perform nonlinear least-squares fit
 - Use [numpy.interp](#) to time shift x-axis and y-axis curves
 - Allow for offset in both x and y to account for the arbitrary shift between fingertip and finger touch point on the screen

Agenda

- Introduction
 - Computer Vision
 - OpenCV & Python
- Examples
 - Recognize an icon on a screen
 - Recognize when an LED is illuminated
 - Measure Android touchscreen latency
- Tips and pitfalls
- Conclusion and questions

Computer Vision Pitfalls

- Incorrect focus
- Insufficient lighting or changing lighting
- Camera lens distortion [[wikipedia](#)]
- Camera image compression
 - JPEG, MJPEG, MPEG
 - Color compression like YUV is usually acceptable
- Rolling shutter [[wikipedia](#)]

OpenCV Pitfalls

- Data types to cv2 functions: often `<var>.astype(np.float32)` is required
- The cv2 image color order is [Blue, Green, Red], not the more typical [Red, Green, Blue]
- cv2 uses numpy matrices with row (y), column(x) order.
- Numpy slicing operations like `a[:, :, 2]` create a view into the original np.ndarray. cv2 needs a full matrix, not a view, so you need to `.copy()` the slice.
- Ubuntu's OpenCV packages exclude the non-free libraries.

Tips

- OpenCV includes some great tutorials and examples. Use them!
- Avoid getting stuck on the image processing “magic”. You can use the algorithms for many applications without knowing the math.
- Can easily use numpy and scipy processing on cv2 images since the images are already numpy ndarrays!
- OpenCV 3.0 is due out soon
- An inexpensive [copy stand](#) can be very useful.

Conclusion

- OpenCV provides a free, solid foundation for image processing with many advanced algorithms
- Applying OpenCV to common image and video processing problems can be reasonably straightforward
- Testing the indicators and user interface of embedded systems software on physical hardware is often a challenge, and OpenCV can help close the gap!

EE Live! Featuring **ESC**
The Embedded Systems Conference

More information



- OpenCV website: <http://opencv.org/>
 - Python Tutorials: http://docs.opencv.org/trunk/doc/py_tutorials/py_tutorials.html
 - C++ Tutorials: <http://docs.opencv.org/doc/tutorials/tutorials.html>
 - Reference Manual: <http://docs.opencv.org/modules/refman.html>
- Over 10 books! See <http://opencv.org/books.html>



- Source code: https://github.com/mliberty1/cv2_product_test

esc EMBEDDED SYSTEMS CONFERENCE

BLACK HAT EMBEDDED

INTERNET OF THINGS

HARDWARE STARTUP

ANDROID ENGINEERING

FPGA ENGINEERING

SUPER C++ TUTORIAL

UBM Tech

EE Live! Featuring **ESC**
The Embedded Systems Conference

Use OpenCV Computer Vision to Test Your Embedded System

Matt Liberty
Jetperch LLC
matt.liberty@jetperch.com

#eelive

Produced by EE Times

esc EMBEDDED SYSTEMS CONFERENCE

BLACK HAT EMBEDDED

INTERNET OF THINGS

HARDWARE STARTUP

ANDROID ENGINEERING

FPGA ENGINEERING

SUPER C++ TUTORIAL

UBM Tech