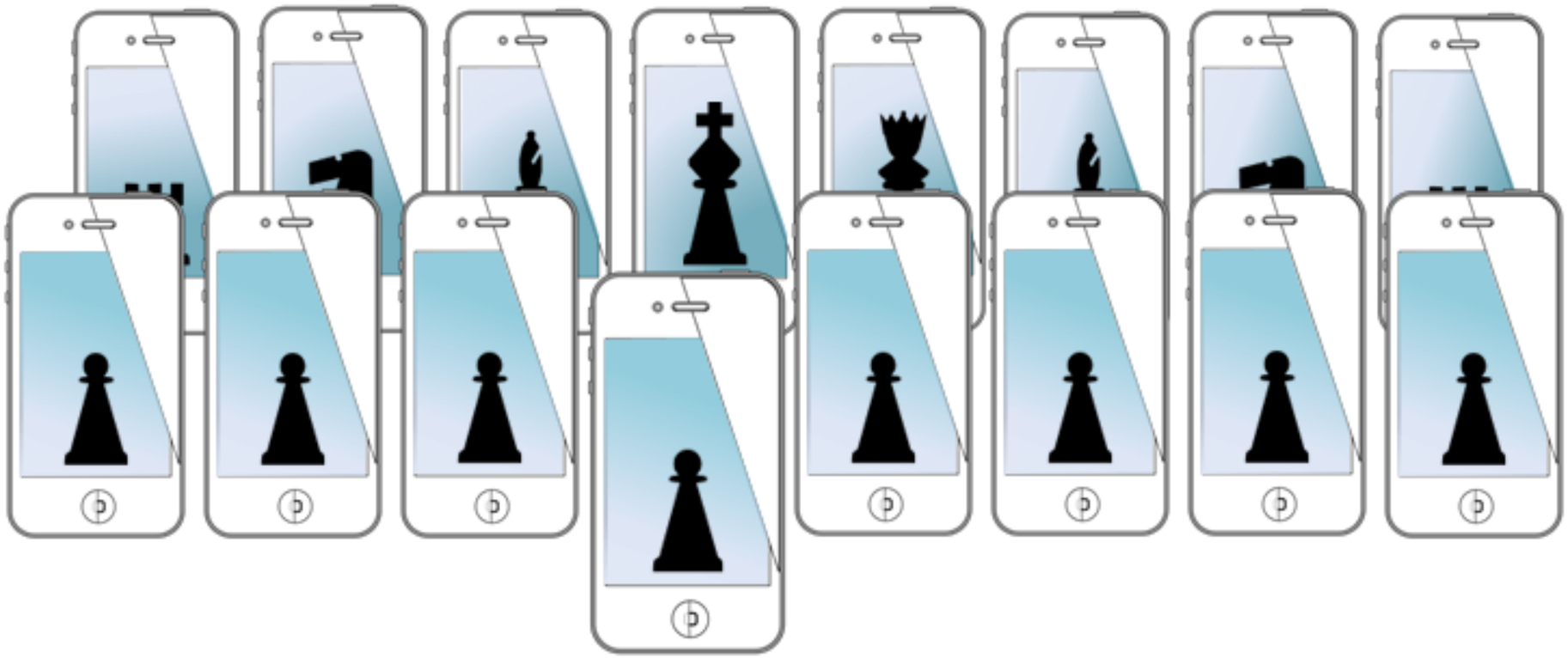
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture thirteen: computer vision

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

- A2 grades are up
- A3 grades soon
- A4 is due the Friday after spring break (~3 weeks)

Module A

Create an iOS application using the CoreImage template that:

- Reads and displays images from the camera in real time
- Highlights multiple faces in the scene
- Highlights eye and mouth position
  - hint: could use another filter for highlights
- (extra credit, up to 0.25 points) displays if the user is smiling or blinking (and with which eye)

Verify the functionality of the application to the instructor during lab time or office hours (or scheduled via email).

Module B

Create an iOS application using the iOpenCV template that:

- Uses video of the user's finger (with flash on) to sense a single dimension stream indicating the "redness" of the finger
- Uses the redness to measure the heart rate of the individual (coarse estimate)
- (optional, NOT extra credit) Display an estimate of the PPG signal

Verify the functionality of the application to the instructor during lab time or office hours (or scheduled via email).

agenda

- OpenCV in iOS
  - we will look at using the tool
  - focus on
    - outputs of each algorithm
    - how to use each method
- ignore most of what is under the hood

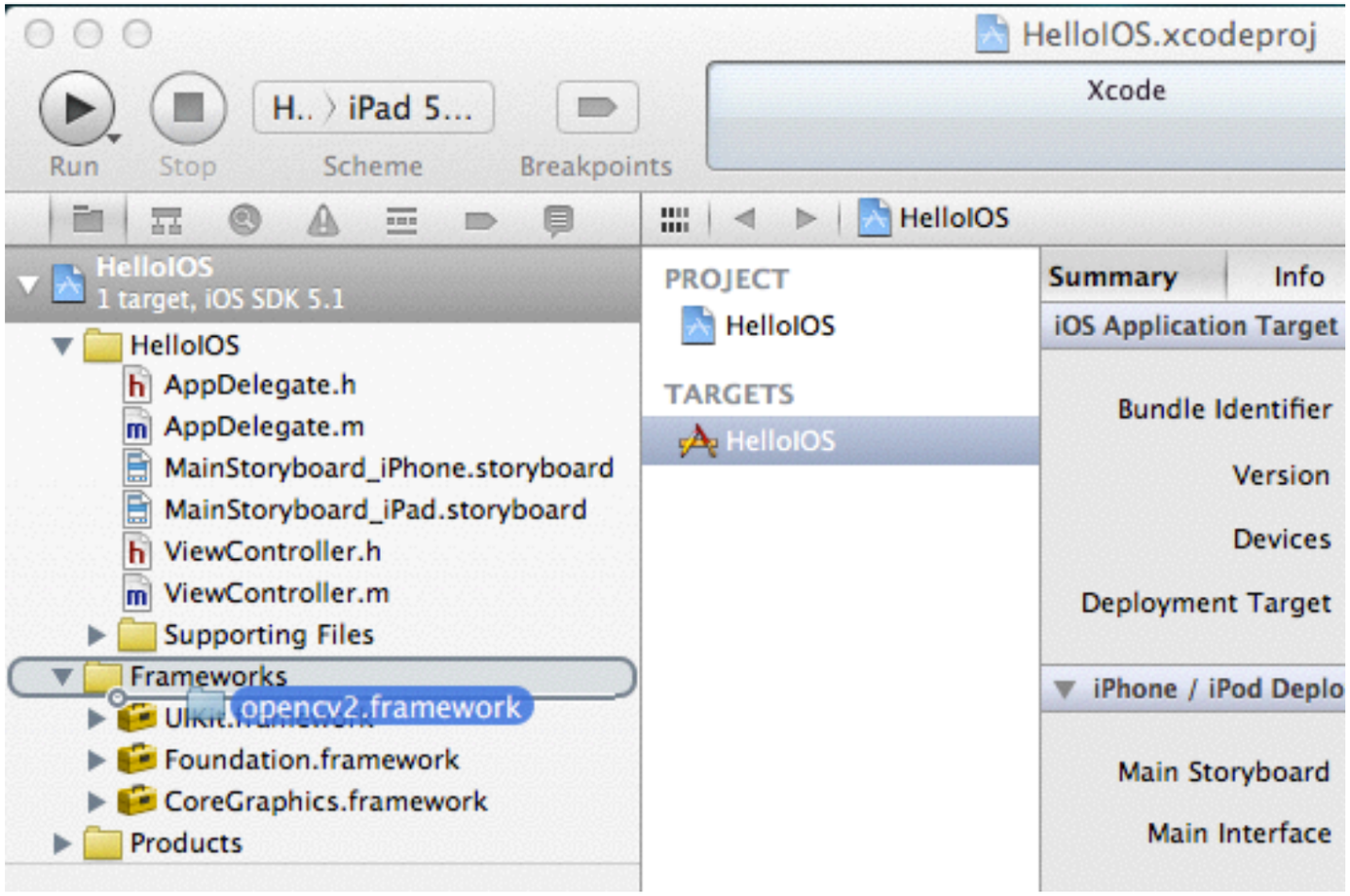
OpenCV in iOS

- open computer vision library
- released by intel
- many common functions are implemented
- written in c++, but has many wrappers
  - EMGU for .NET (c#,VC++,etc.), pycv2 for python, Java API for Android, and many, many more
- some hardware accelerations on iOS
  - not as many as core image
  - expect slightly slower processing, but still fast!

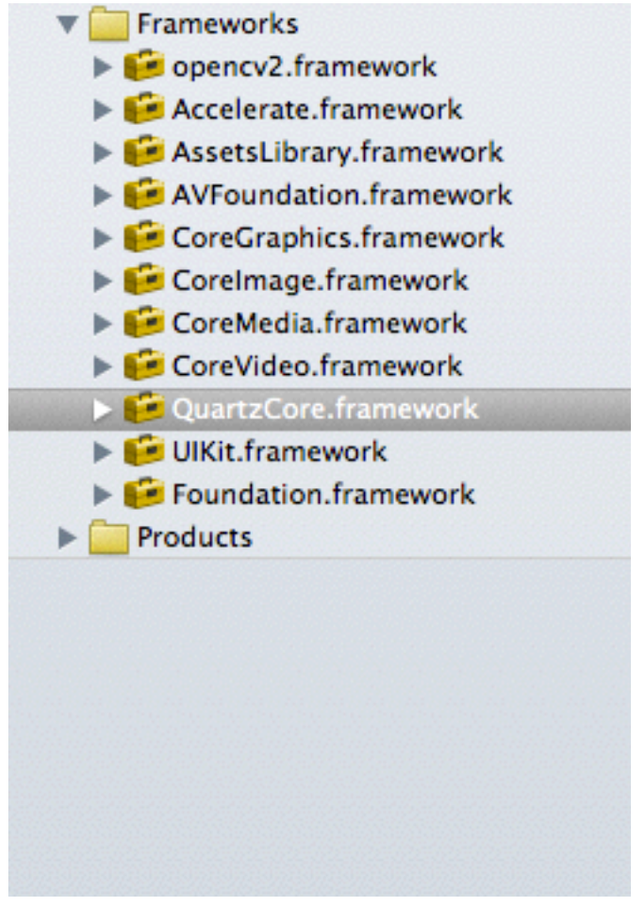
OpenCV installation

- download the opencv framework for iOS
- drag into project
- manually add a bunch of dependencies
- step by step instructions:
  - http://docs.opencv.org/doc/tutorials/ios/video_processing/video_processing.html
- remember to rename your view controller (or model) to .mm

OpenCV installation




```
#ifdef __cplusplus
#import <opencv2/opencv.hpp>
#endif
```



OpenCV video

- tutorial will also show you how to setup video capture
- you can use the delegate protocol and a cvVideoCamera

```
@interface YourViewController ()<CvVideoCameraDelegate>
@property (weak, nonatomic) IBOutlet UIImageView *imageView;
@property (nonatomic, strong) CvVideoCamera* videoCamera;
@end
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
   // Do any additional setup after loading the view, typically from a nib.
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:self.imageView];
    self.videoCamera.delegate = self;
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionBack;
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset352x288;
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;
    self.videoCamera.defaultFPS = 30;
    self.videoCamera.grayscaleMode = NO;
    [self.videoCamera start];
}
```

OpenCV video

```
- (void)viewDidLoad
{
    [super viewDidLoad];
  // Do any additional setup after loading the view, typically from a nib.
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:self.imageView];
    self.videoCamera.delegate = self;
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionBack;
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset352x288;
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;
    self.videoCamera.defaultFPS = 30;
    self.videoCamera.grayscaleMode = NO;
    [self.videoCamera start];
}


 #ifdef __cplusplus
 - (void)processImage:(Mat&)image;
 {
    Mat image_copy;
    cvtColor(image, image_copy, CV_BGRA2BGR); // get rid of alpha for processing
    // processing here to the image_copy
    cvtColor(image_copy, image, CV_RGB2BGRA); //add back for display
 }
 #endif
```

OpenCV video

- you are not on the main queue here
- OpenCV is mostly updated for iOS7
  - some functions they use are deprecated, but currently still work
  - you are at the mercy of the OpenCV community for implementing the updates (or update yourself!)

```
#ifdef __cplusplus
- (void)processImage:(Mat&)image;
{
   Mat image_copy;
   cvtColor(image, image_copy, CV_BGRA2BGR); // get rid of alpha for processing
   // processing here to the image_copy
   cvtColor(image_copy, image, CV_RGB2BGRA); //add back for display
}
#endif
```

OpenCV Demo

access torch

before we get too far…

```
- (IBAction)toggleTorch:(id)sender {
    self.torchIsOn = !self.torchIsOn;
    [self setTorchOn:self.torchIsOn];


}

- (void)setTorchOn: (BOOL) onOff
{
    AVCaptureDevice *device = [AVCaptureDevice
                               defaultDeviceWithMediaType:AVMediaTypeVideo];
    if ([device hasTorch]) {
        [device lockForConfiguration:nil];
        [device setTorchMode: onOff ? AVCaptureTorchModeOn : AVCaptureTorchModeOff];
        [device unlockForConfiguration];
    }
}
```

OpenCV operations

- your input is a matrix
- data is interleaved BGR
  - if setting color conversion to CV_BGRA2BGR
  - must get a pointer in the array for the row and column

```
image.ptr(row, column)
```

starts from upper left

```
for(int i=0;i<50;i++){
      image.ptr(i, i)[0] = 255;
      image.ptr(i, i)[1] = 0;
      image.ptr(i, i)[2] = 0;
 }
```

blue

green

red

```
for(int i=0;i<50;i++){
      uchar *pt = image.ptr(i, i);
      pt[0] = 255;
      pt[1] = 0;
      pt[2] = 0;
 }
```

```
for(int i=0;i<50;i++){
      uchar *pt = image.ptr(i, i);
      pt[0] = 255;
      pt[1] = 0;
      pt[2] = 0;
      pt[3] = 255;
      pt[4] = 0;
      pt[5] = 0;
 }
```

pixel at i,i

next pixel in row

fast image down-sizing

- for reducing image size quickly (e.g., to increase frame rate)
- pyramid
  - nearest neighbor (no interpolation)
  - integer multiple down-sampling (throws rows and columns)
  - anti-aliasing built in

```
cv::pyrDown(image_copy, image);
```

OpenCV operations

- filtering

```
Mat gauss = cv::getGaussianKernel(25, 3);
cv::filter2D(image_copy, image_copy, -1, gauss);
GaussianBlur(image_copy, image_copy, cv::Size(3, 3), 2, 2 );
```

- inversion

```
bitwise_not(image_copy, image_copy);
```

- statistics

```
Scalar avgPixelIntensity = cv::mean( image_copy );
avgPixelIntensity.val[0]
avgPixelIntensity.val[1]
avgPixelIntensity.val[2]
```

- color conversion

```
cvtColor(image, image_copy, CV_BGRA2BGR);
cvtColor(image, image_copy, CV_GRAY2BGR);
cvtColor(image, image_copy, CV_RGB2BGR);
cvtColor(image, image_copy, CV_BGR2HSV);
cvtColor(image, image_copy, CV_BGR2Lab);
cvtColor(image, image_copy, CV_BGR2Lab);
cvtColor(image, image_copy, CV_BGR2YCrCb);
cvtColor(image, image_copy, CV_BGR2YUV);
```

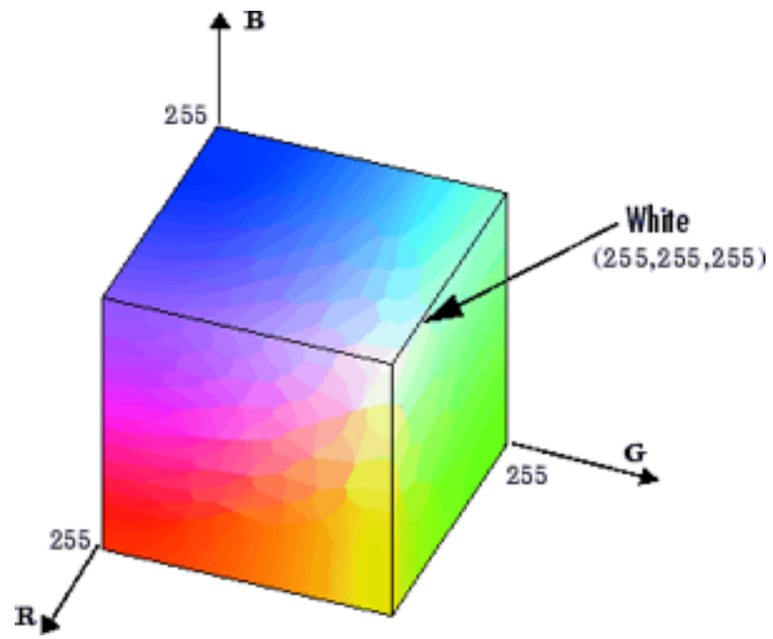
OpenCV Demo

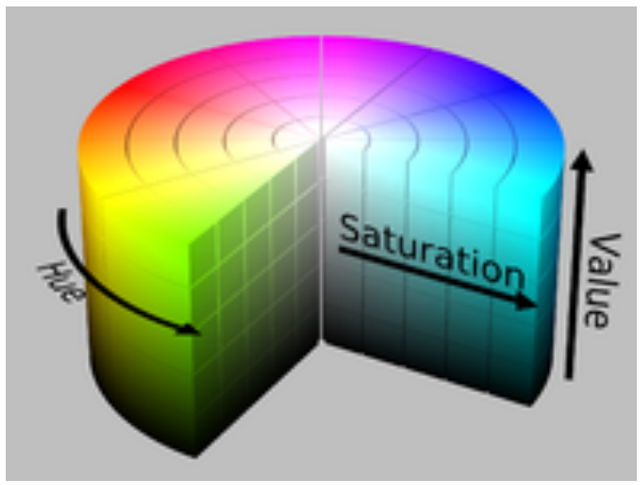
- basic operations

color conversion

- to display properly, use BGRA

```
cvtColor(image, image_copy, CV_BGRA2BGR);

cvtColor(image_copy, image, CV_BGR2BGRA);

cvtColor(image, image_copy, CV_BGRA2GRAY);

cvtColor(image_copy, image, CV_GRAY2BGRA);

cvtColor(image, image_copy, CV_BGRA2HSV);

cvtColor(image_copy, image, CV_HSV2BGRA);
```

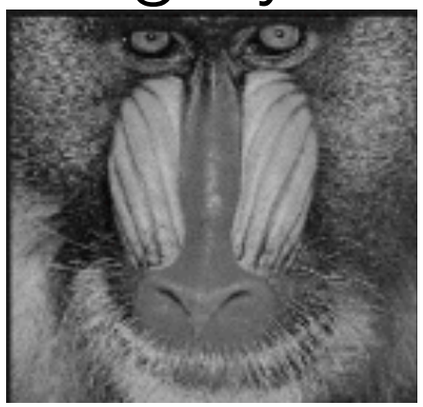


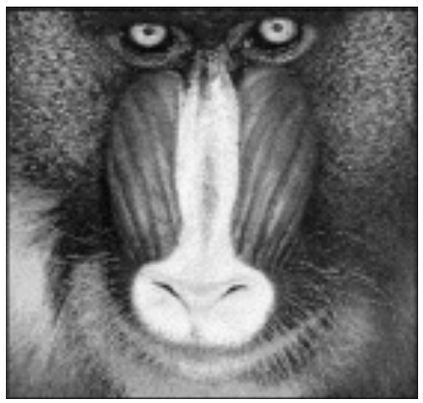


color conversion

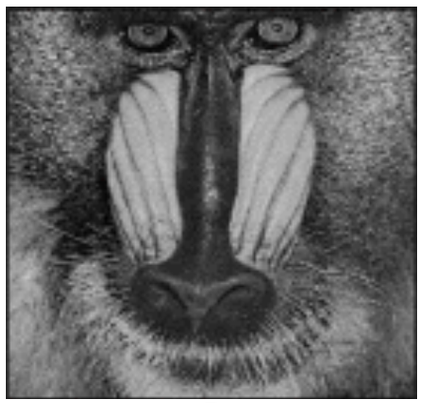
original

gray

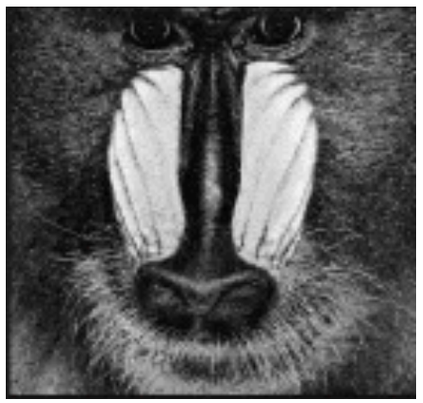




R



G



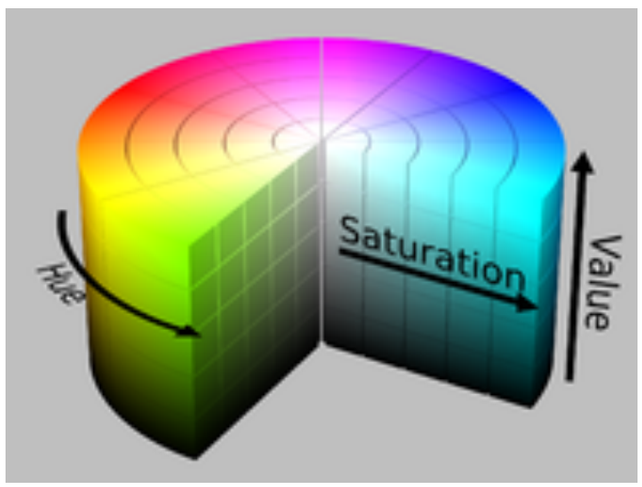
B

H

S

V

- **hsv**
  - what we perceive as color, rather than "sense" as color (sort of)
    - hue: the color value
    - saturation: the richness of the color relative to brightness
  - value: the intensity

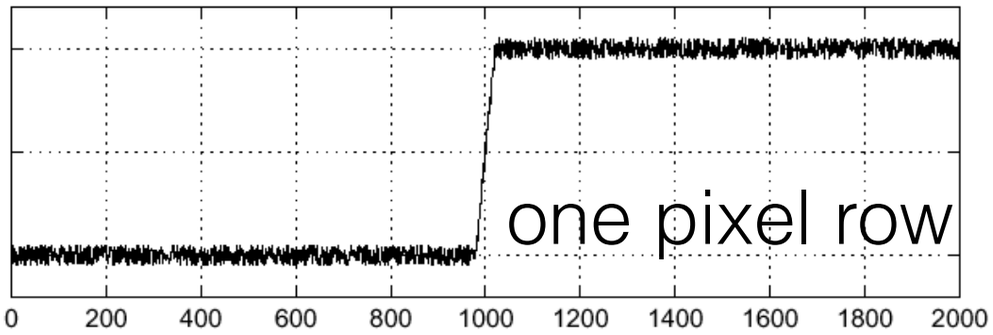


edge detection

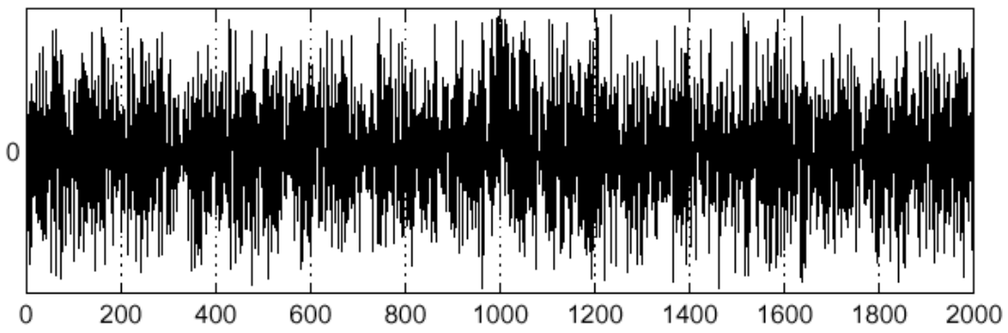
- can use linear filters to get gradient

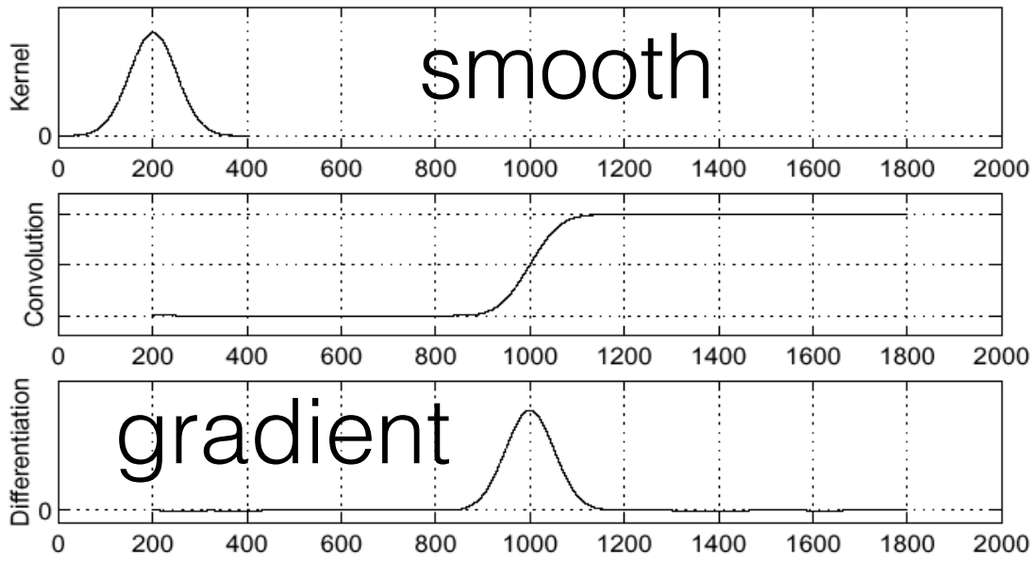
-1	0	1
-2	0	2
-1	0	1

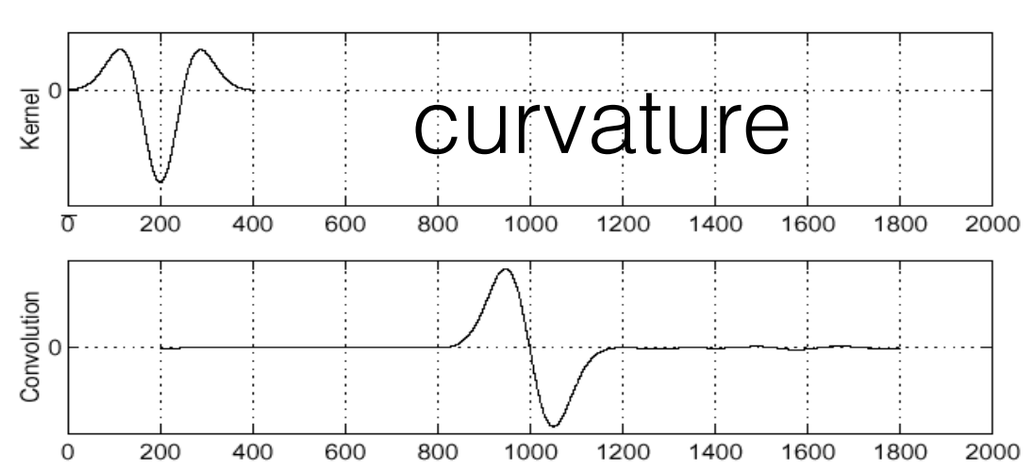
1	2	1
0	0	0
-1	-2	-1



$\frac{d}{dx} f(x)$







images courtesy of S. Narasimhan

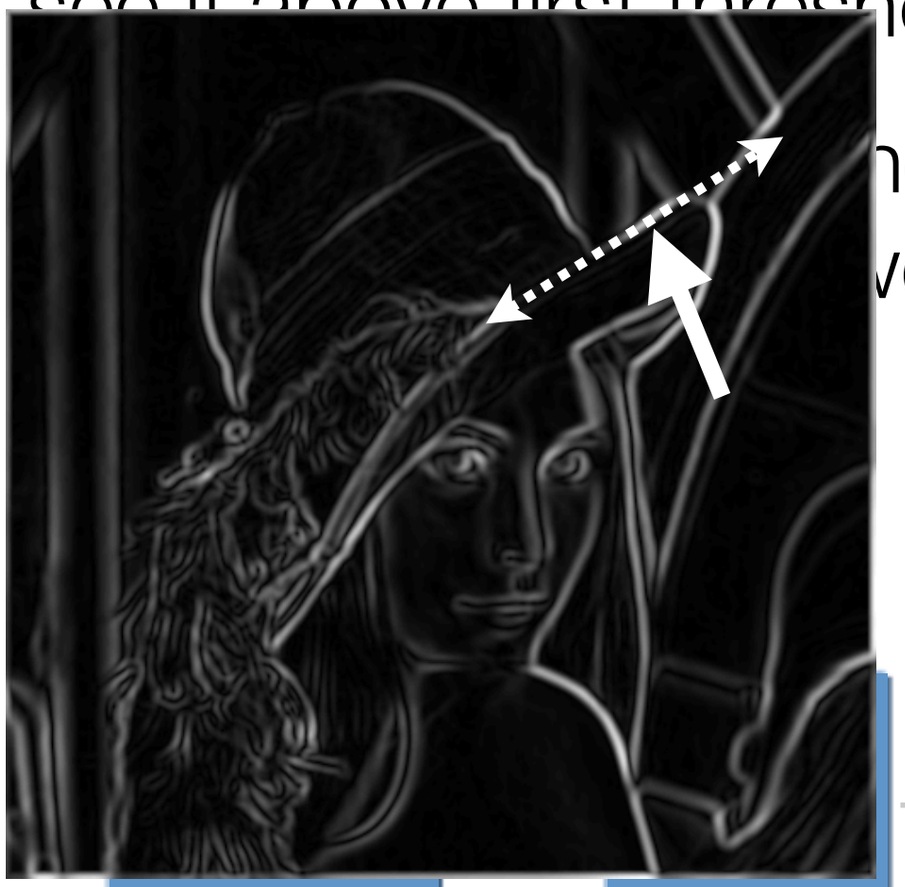
gradient example

can we do better?

check local maxima

canny edge detection

- get local maxima of gradient
- see if above first threshold
- [illegible] lar to [illegible]



$G_{x,y}$

From magnitude and direction of $G_{x,y}$

[illegible] sis [illegible] ding

hyst1 = image
hyst2 = along lines

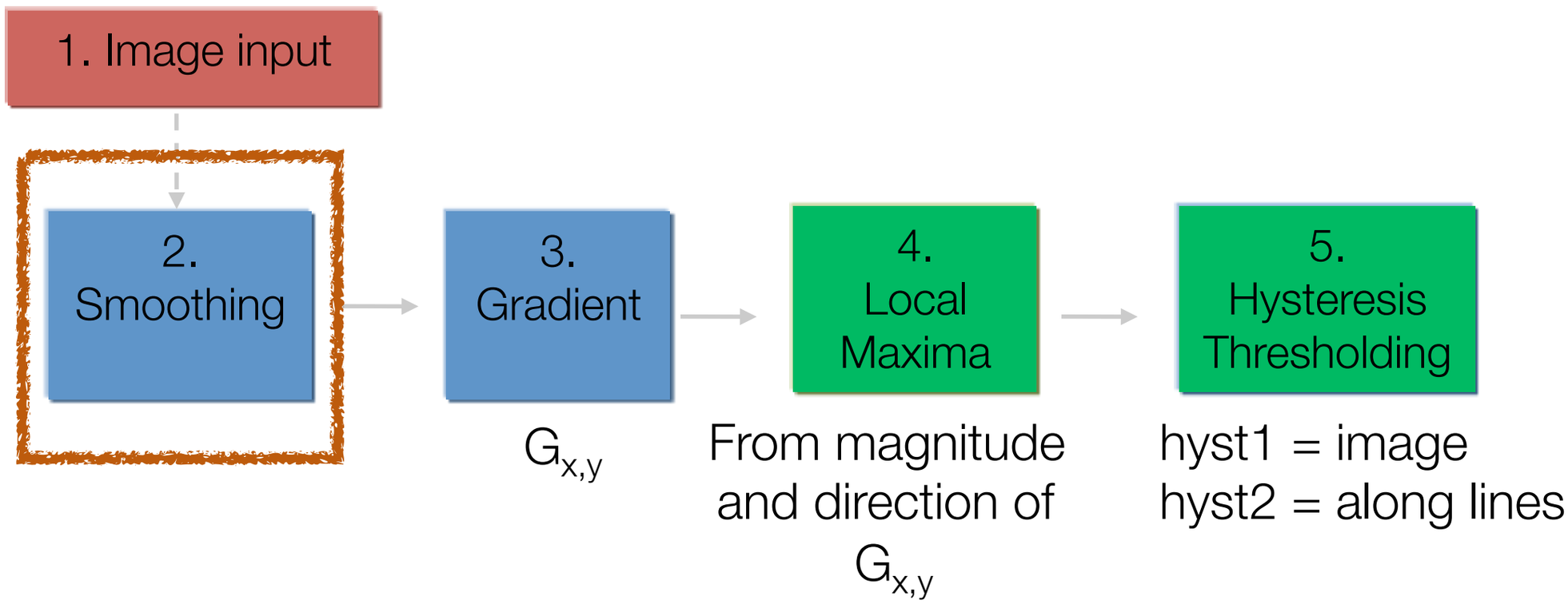
canny edge detection

```
const int kCannyLowThreshold = 300;
cvtColor(image, grayFrame, COLOR_RGB2GRAY);

// Perform Canny edge detection using sl
Canny(grayFrame, output,
          kCannyLowThreshold,
          kCannyLowThreshold*7,
          kFilterKernelSize);

// copy back for further processing
cvtColor(output, image, CV_GRAY2BGRA); //add back for display
```

hyst2: small threshold

hyst1: large threshold

size of gradient filters



edges to contours

- connected components search
- contour detection from “outside” of component

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	1	1	0	1	1	1	1	1	0
0	1	1	0	0	1	0	0	0	0	1	1	1	1	1	0
0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1
0	1	1	0	0	1	0	0	0	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

OpenCV edge demo

- edges
- contours

generic Haar cascade

- remember Haar cascade filtering?
- you can use any trained cascade of classifiers
  - just need a trained file to load in
- get some trained xml files, here is a start:
  - http://alereimondo.no-ip.org/OpenCV/34/
- or train your own (this is not trivial)
  - http://docs.opencv.org/doc/user_guide/ug_traincascade.html
  - database of positive example images
  - database of negative example images

Haar syntax for iOS

```
cv::CascadeClassifier classifier;

// load in custom trained Haar Cascade filter
// This one is a famous trained face detector from Rainer Lienhart
 // http://www.lienhart.de/Prof._Dr._Rainer_Lienhart/Welcome.html
 NSString *fileName = [[NSBundle mainBundle]
                pathForResource:@"haarcascade_frontalface_alt2" ofType:@"xml"];

 classifier = cv::CascadeClassifier([fileName UTF8String]);
```

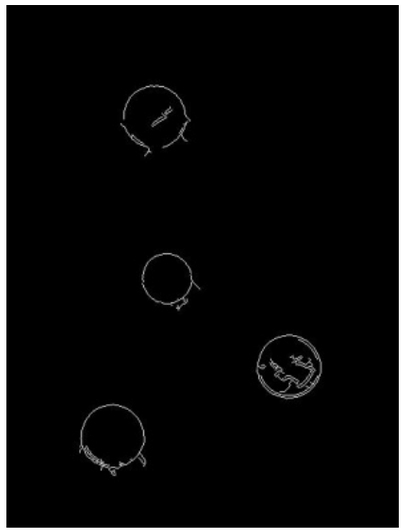
```
cvtColor(image, grayFrame, CV_BGRA2GRAY);
  vector<cv::Rect> objects;

  // run classifier
  classifier.detectMultiScale(grayFrame, objects);

  // display bounding rectangles around the detected objects
  for( vector<cv::Rect>::const_iterator r = objects.begin(); r != objects.end(); r++)
  {
      cv::rectangle( image,
            cvPoint( r->x, r->y ),
            cvPoint( r->x + r->width, r->y + r->height ),
            Scalar(0,0,255,255));
  }
```

OpenCV Demo

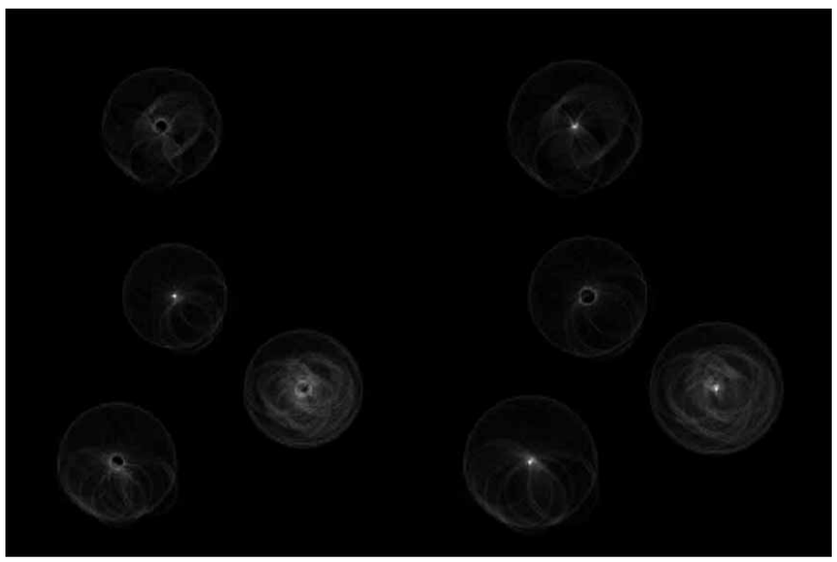
- user trained Haar cascades

hough transform

- In general, hough transform consists of
  - edge detection
  - for each detected point,
  - draw shape with different parameter
  - accumulation in parameter space
  - look for local maxima
  - for a circle, look for maxima in (x,y,R)



```
vector<Vec3f> circles;

/// Apply the Hough Transform to find the circles
HoughCircles( image_copy, circles,
              CV_HOUGH_GRADIENT,
              1, // downsample factor
              image_copy.rows/30, // distance between centers
              kCannyLowThreshold/2, // canny upper thresh
              40, // magnitude thresh for hough param space
              0, 0 ); // min/max centers
```



Open CV demo

- the hough

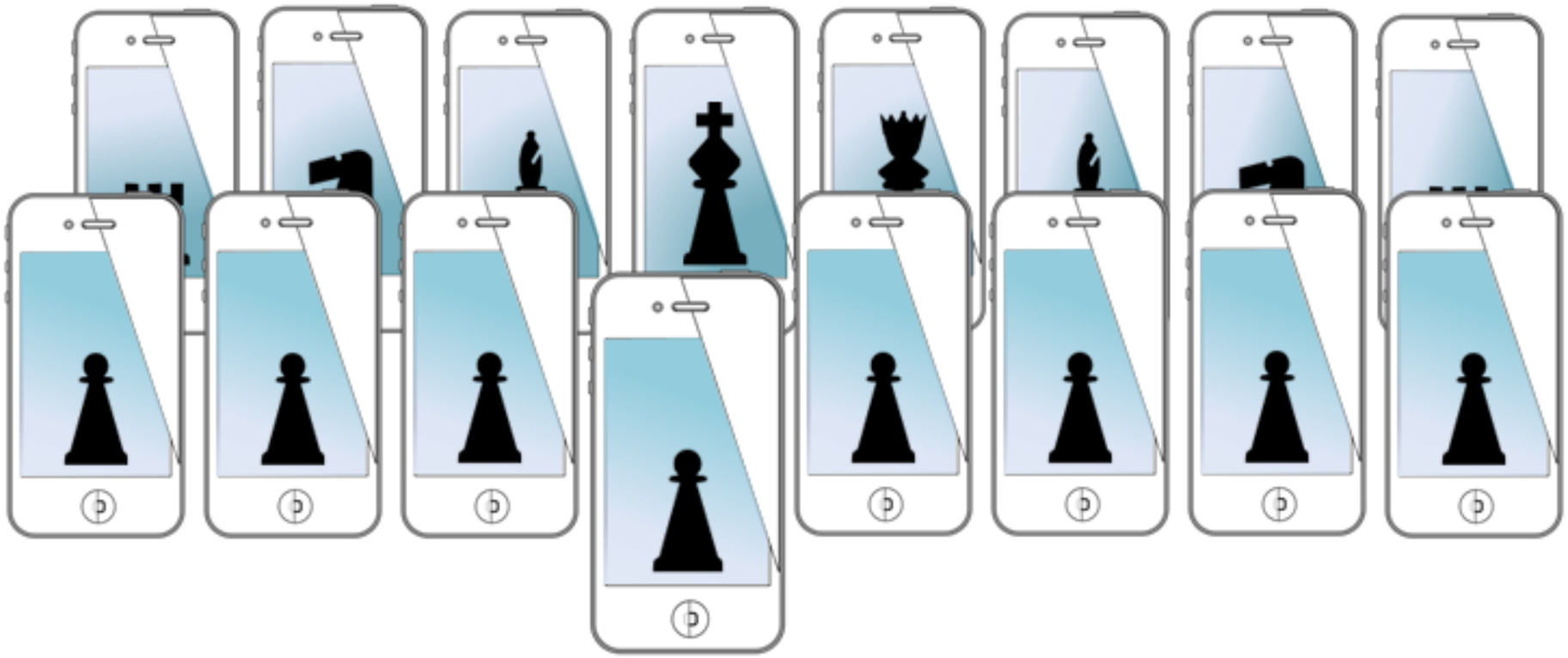
have fun with it!

- OpenCV is a powerful framework
- many contributors
  - each algorithm has (possibly) different semantics
  - highly comprehensive and updated
  - lots of examples
    - OpenCV is absolutely an industry standard

for next time…

- have a great spring break!
- then come back and learn about microcontroller basics

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture thirteen: computer vision

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University