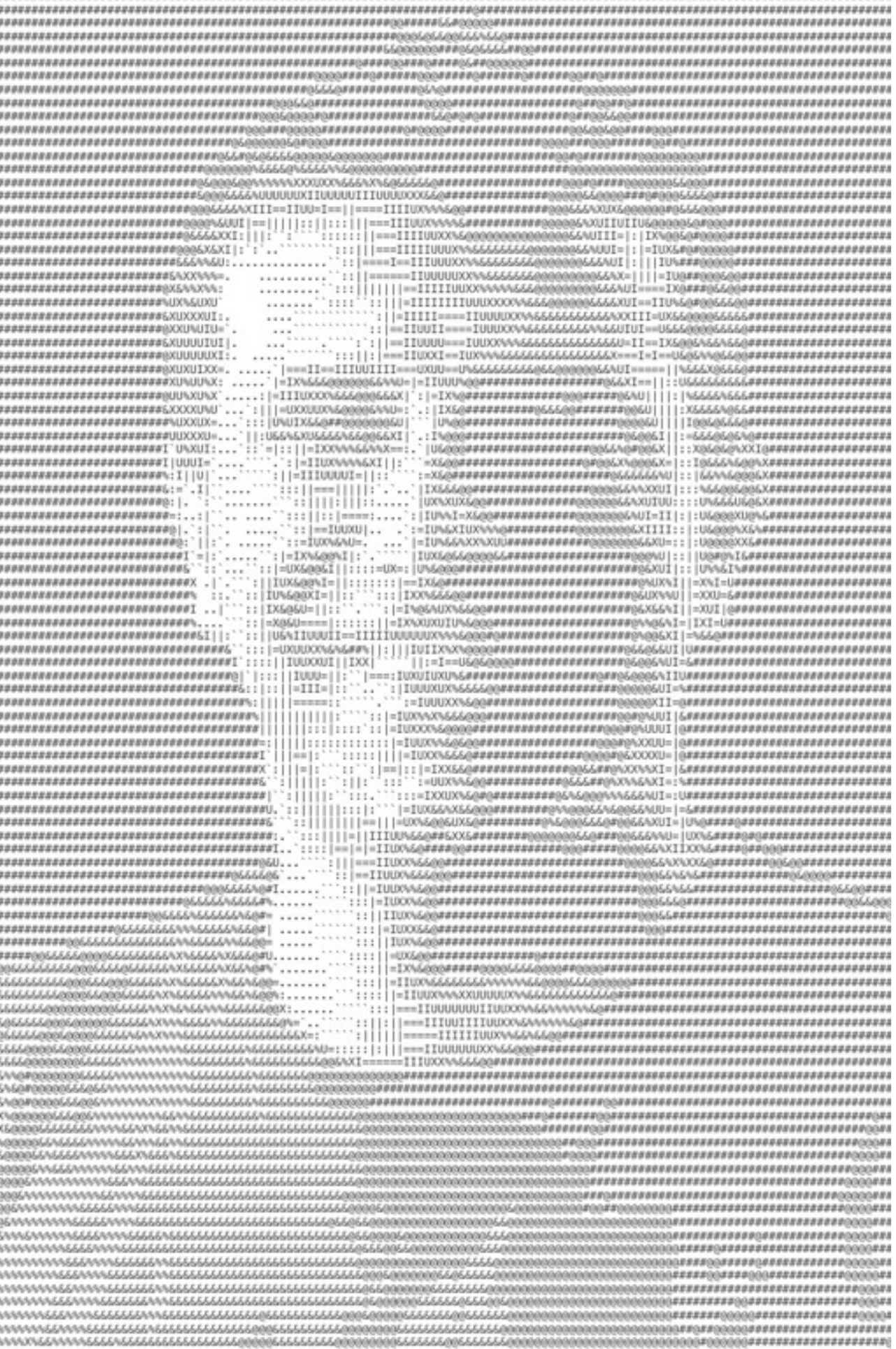
# DRIVING ASSISTANT SOLUTIONS WITH ANDROID

*@giorgionatili*



# ABOUT ME

- Engineering Manager at Akamai Technologies
- Organizer of DroidconBos (www.droidcon-boston.com)
- Organizer of Swiftfest 2017 (www.swiftfest.io)
- Organizer of Hashcon 2018 (www.hascon.io)
- Founder of Mobile Tea (www.mobiletea.net)

# AGENDA

- Computer vision essentials
- The fastest ever introduction to OpenCV
- Setup the OpenCV Android SDK and C++ integration
- How to detect (traffic) lights
- How to detect driving lanes
- Detecting features from a video source
- Challenges and next steps

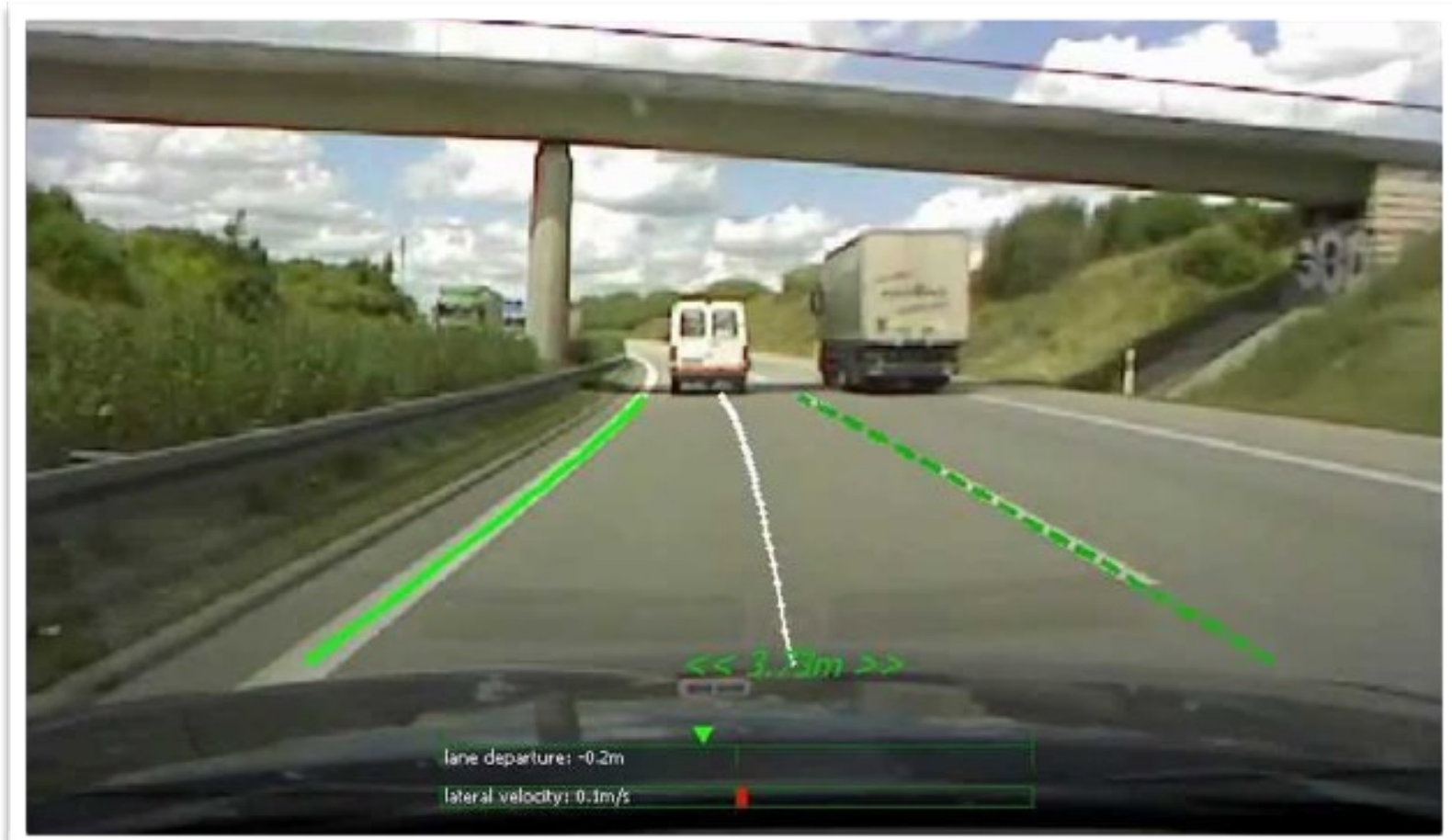
# THE PROBLEM

*What we are trying to do*

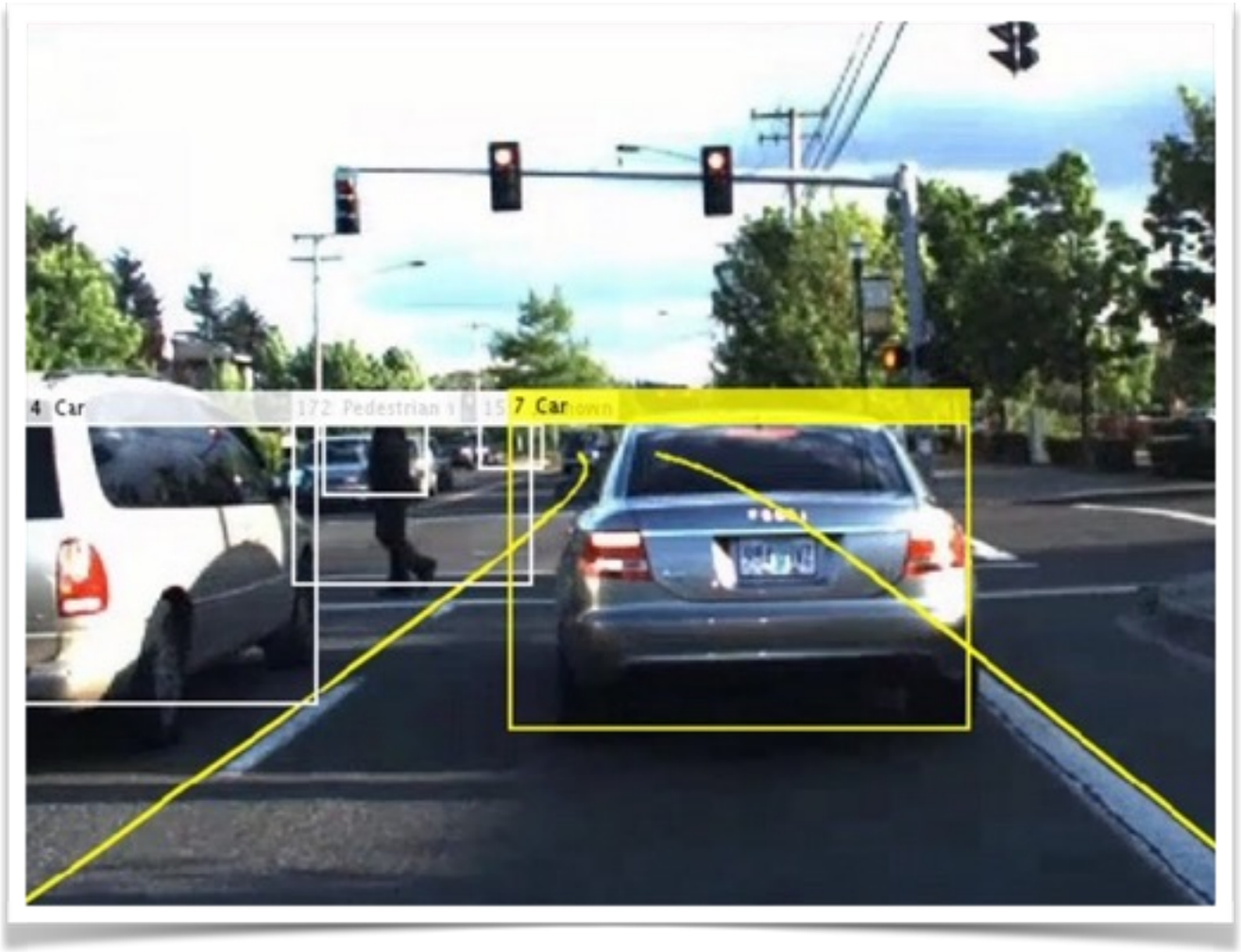
# GOALS

- Provide driving assistance solutions to everyone
- Improve the driving experience of everyone *(signals interpreting, contextual information, etc.)*
- Reduce the number of accidents due to distractions
- Monitor driving styles to train offline models

# AM I INSIDE THE LANES?



# THERE ARE OBSTACLES I HAVEN'T NOTICED?



# CAN I PARK HERE?


@DroidconBos

@giorgionatili

# PONY DRIVE

# COMPUTER VISION

*Exploring the building blocks of computer vision*

*Computer Vision combines magic and science to create illusions.*

*-Marco Tempes*

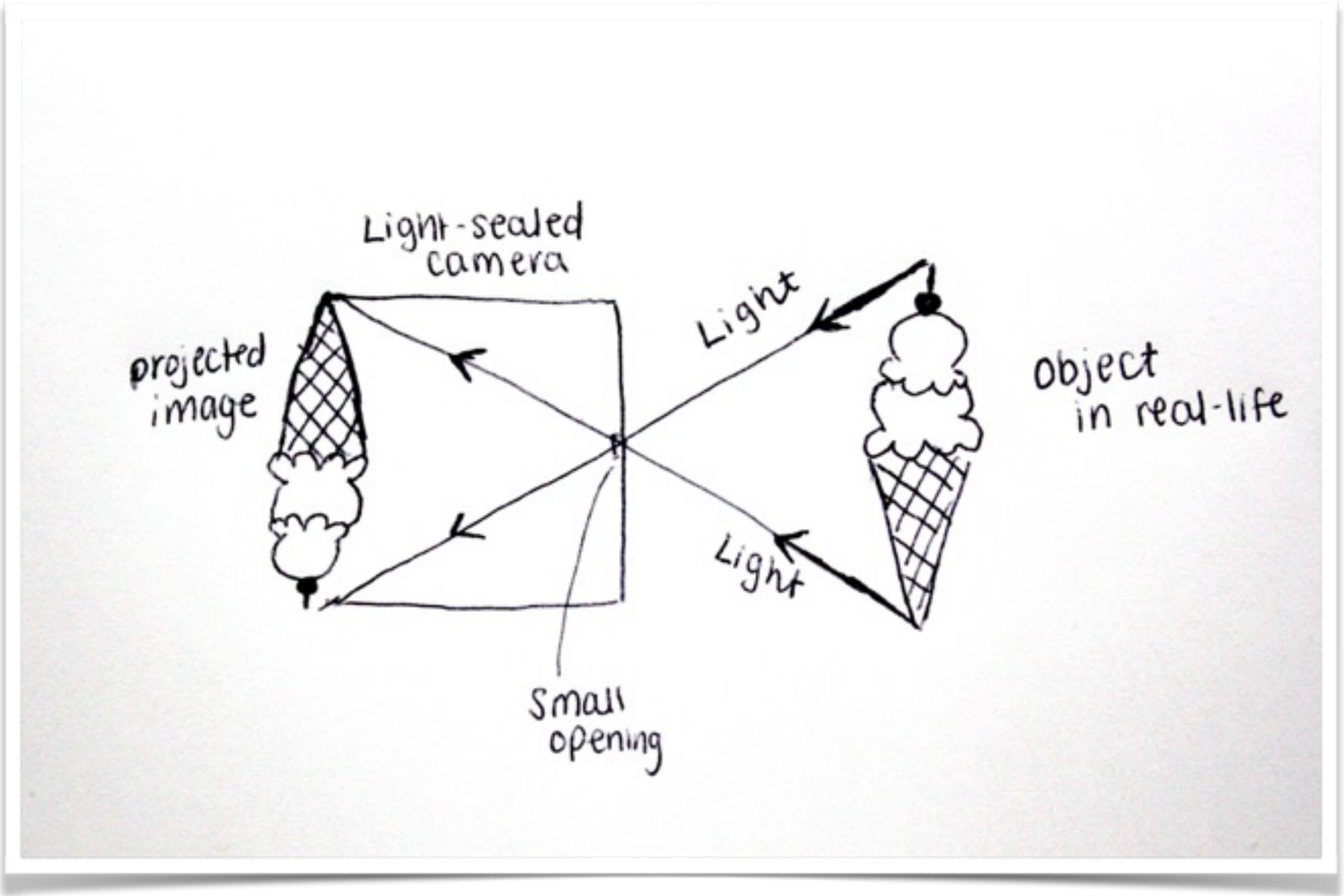
# DATA SOURCES

- Images are a source of information, the way a software transform them is crucial to get a clean data set
- A good source of data should be clean, that's what you achieve trough image processing

# DATA MANIPULATION

- An image is a 2D projection of a 3D scene expressed as the quantized result of `f(i, j)`
- Digital images are created by sampling a continuos image into discrete elements

# HOW THE CAMERA WORKS



# IMAGES AND FRAMES

- Images and frames are made up of pixels organized in a 2D array
- In a color image each pixel can be anything from 8 to 32 bits wide
- A grayscale images use 8 bits per pixel
- A frame is a single image in a video sequence

# IMAGES, PIXELS & SAMPLING

- Digital images are created by sampling a continuos image into discrete elements
- Digital images sensors consists of a 2D array of photosensitive elements *(pixels/image points/sampling points)*
- The number of samples in an image limit the objects that can be distinguished or recognized in the image

# BITS AND CHANNELS

- Each pixel in a digital image is the representation of the brightness of a given point in the scene
- Typically the number of levels per channel is $k=2^b$ where $b$ is the number of bits *(often 8)*

*Why this is important?*

*Because a reduced number of bits*

*save memory while preserving the*

*most meaningful information!*


YEAH!

# COLOR IMAGES

- Color images are a very rich set of information
- Color analysis can help to locate traffic signs, luminance and chrominance are a source of interesting information
- However, colors are not always important to determine the content of an image

# YOU CAN STILL RECOGNIZE THE ROSTER

*@DroidconBos*

*@giorgionatili*

# GRAY SCALE IMAGES

- Gray scale images are smaller in size and easier to analyze
- Luminance is by far more important than chrominance in distinguish visual features
- In most cases luminance provides enough information to detect the edges of an image
- Processing gray scale images reduce code complexity and learning curve

# EFFECTIVE GRAY SCALE IMAGES

- Scale the image to 25% of the original size
- Determine a region of interest to process
- Apply a gray scale transformation

```
image =  cv::imread("puppy.bmp", cv::IMREAD_GRAYSCALE);
```

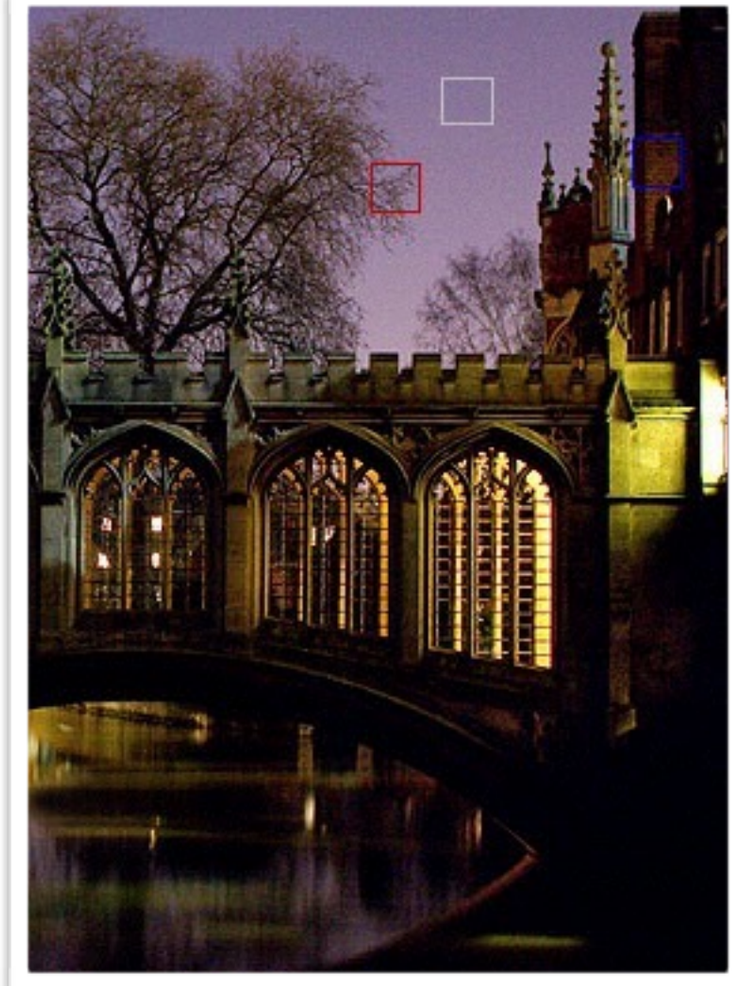
# IMAGE NOISE

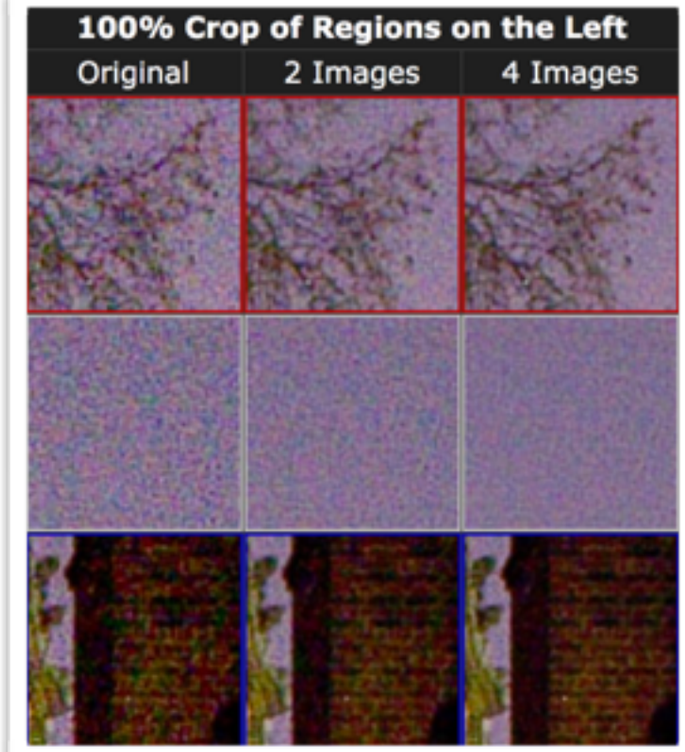
- Image noise is the measurement of the degradation of an image
- Image noise is usually an aspect of electronic noise and it can be produced by the sensor that is acquiring the image (aka the camera)

# REDUCING NOISE THROUGH IMAGE AVERAGING

- Image averaging works on the assumption that the noise in your image is truly random
- This way, random fluctuations above and below actual image data will gradually produce an average image

# IMAGE AVERAGING

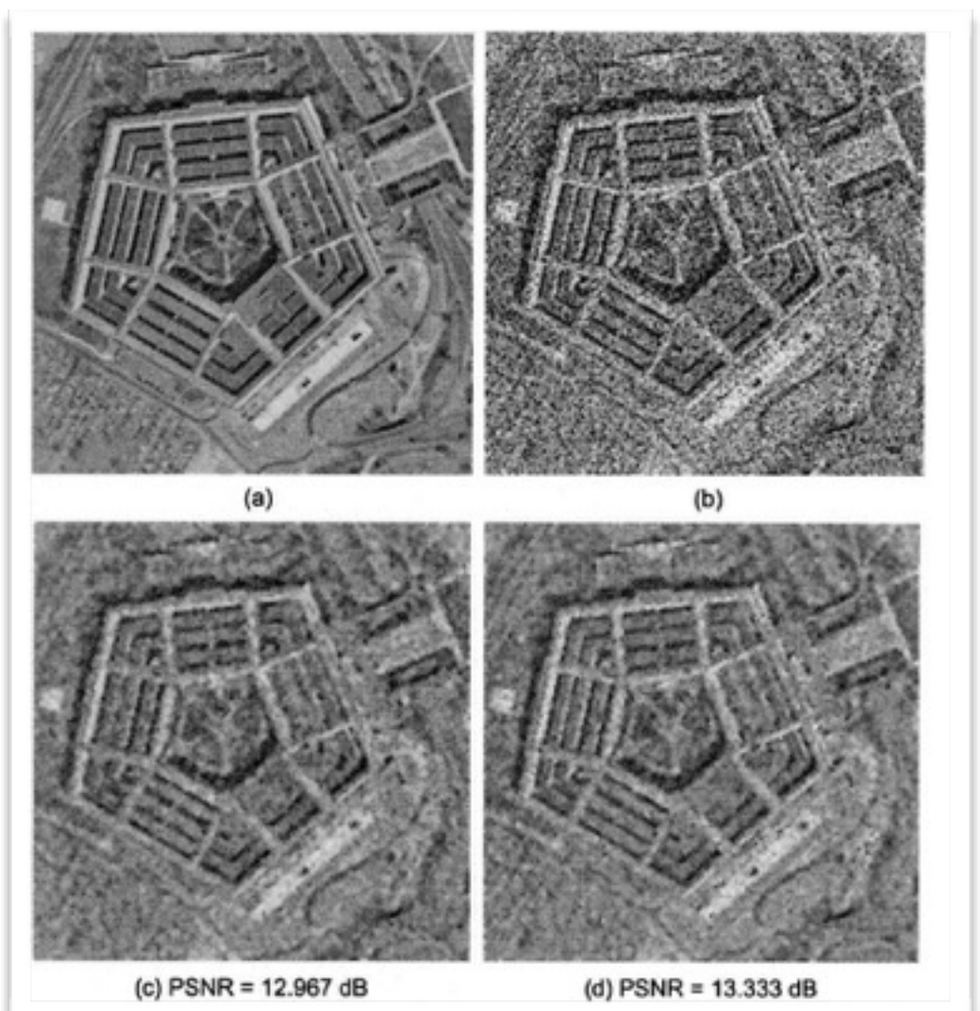





*source: cambridgeincolour.com*

# OTHER TECHNIQUES

*Gaussian filter*
*noise reduction*



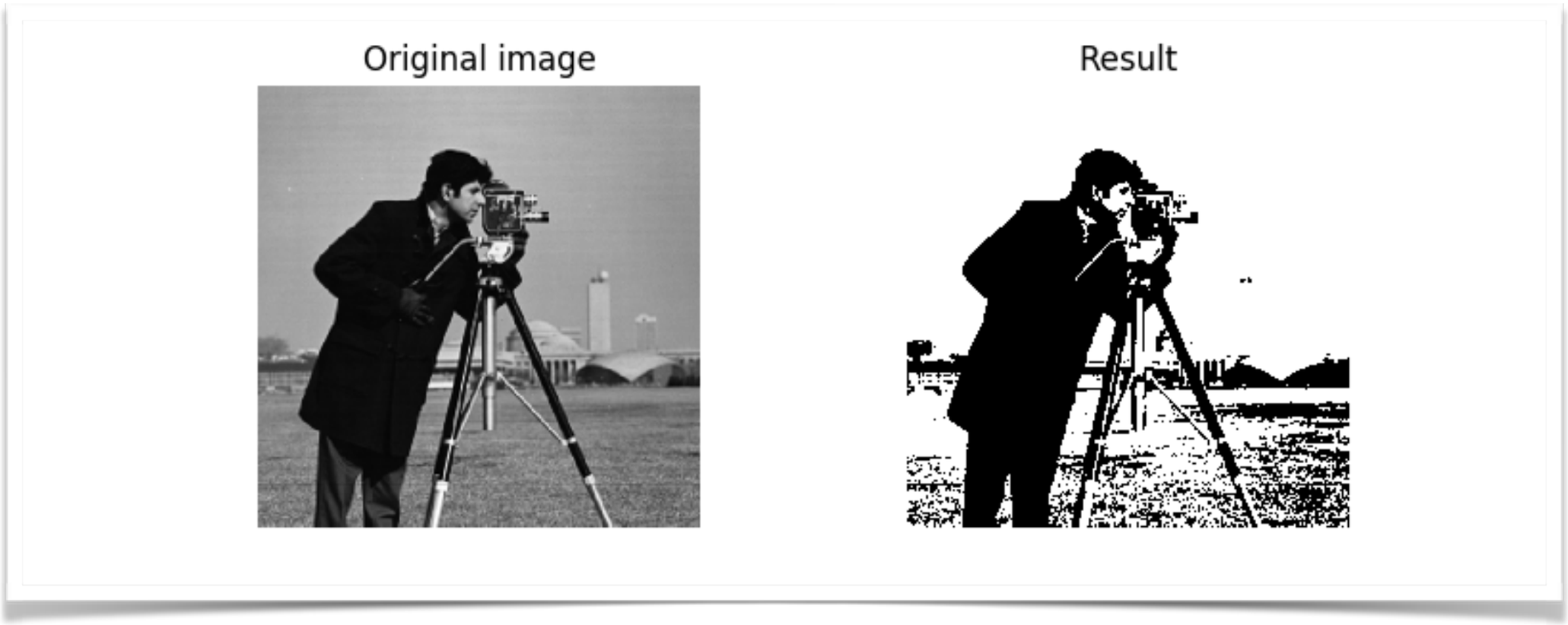

*Salt an pepper filter*
*noise reduction*

# IMAGE PROCESSING

# THRESHOLD

- When the source image is not of good quality, thresholding techniques can be used to separate the object of interest from the background
- Thresholding techniques imply the usage of algorithms and the processing lighting information

# IMAGE THRESHOLDING




*Thresholding used to create a binary representation of a grayscale image*

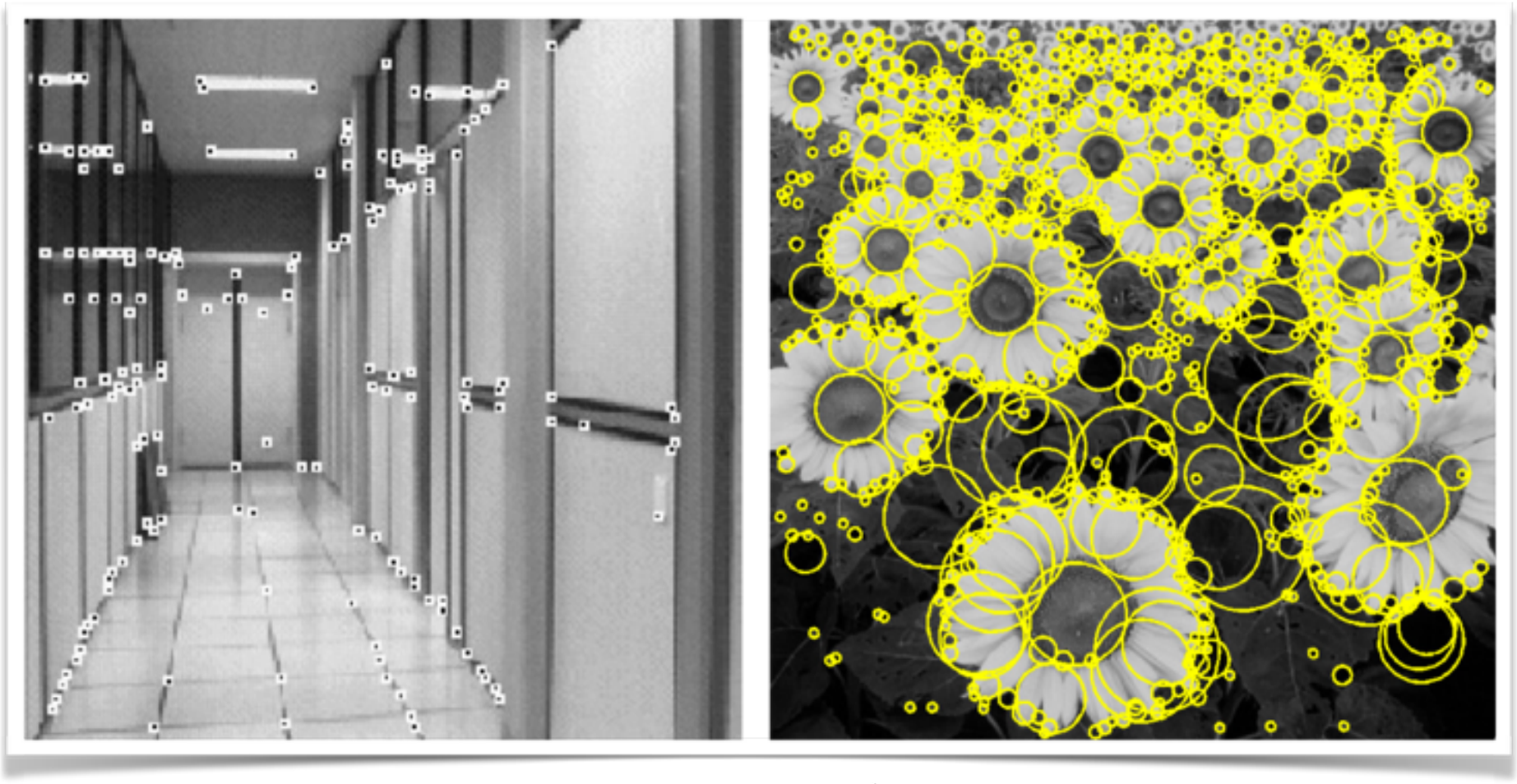
# FEATURES DETECTION

*In computer vision and image processing the concept of feature detection refers to methods that aim at computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not*

# IMAGE FEATURES

- There is no universal or exact definition of what constitutes a feature, and the exact definition often depends on the problem or the type of application
- Image feature can be corners, blobs, edges, etc.

# CORNERS AND BLOBS DETECTION



*Source N. Snavely*

# FEATURES DESCRIPTION

- The second step in the process of using image features is feature description
- The feature descriptors are used to provide more information around the interest points computed over the local region/ neighborhood of the detected feature

# WHAT'S IN A DESCRIPTOR

- A sampling pattern, where to sample points in the region around the descriptor
- Orientation compensation, some mechanism to measure the orientation of the key point and rotate it to compensate for rotation changes
- Sampling pairs, the logic to decide which pairs to compare when building the final descriptor

# BINARY DESCRIPTORS

- These descriptors are floating-point vectors that have a dimension of *64*, *128*, or sometimes even longer
- In order to reduce the memory and computational load it's possible to use binary descriptors composed of a simple sequence of bits *(0s and 1s)*
- With OpenCV using a binary descriptor is no different from using other descriptors

# BRIEF

- Describes an interest point with a description vector of N pairs
- The algorithm chooses N random pairs of pixels in the 31x31 patch region by several randomization methods *(uniform, Gaussian, and others)*
- The algorithm compares them to construct the binary string

# ORB

- The ORB descriptor adds orientation to BRIEF by steering the interest point to the canonical orientation
- For pixels pair sampling method it chooses the pixel pairs in a way that maximizes the variance and reduces the correlation

# BRISK

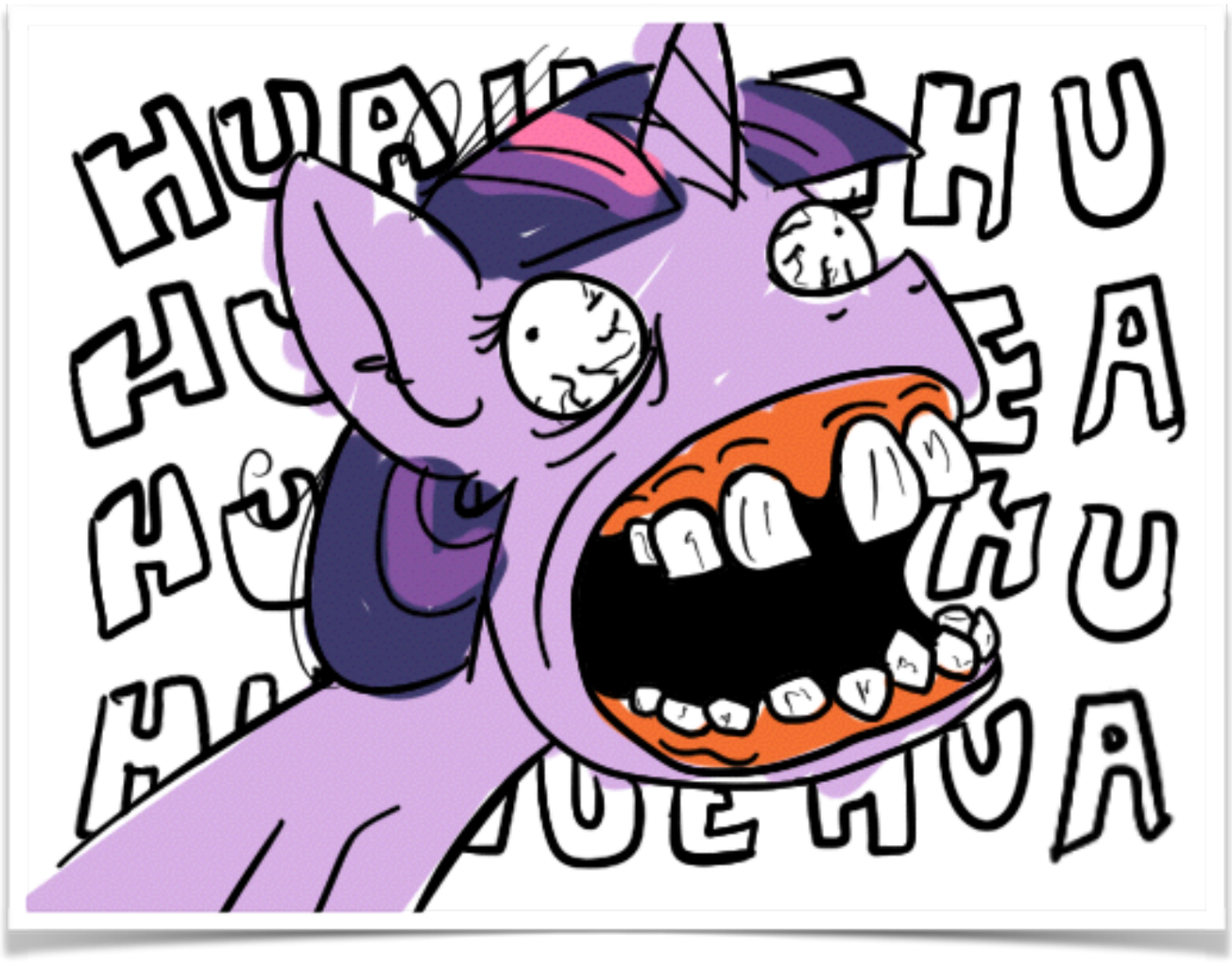
- The Binary Robust Invariant Scalable Key-points *(BRISK)* descriptor is built on 60 points arranged in four concentric rings
- To calculate the orientation, every sampling region is smoothed with a Gaussian filter and local gradients are calculated

# FREAK

- The Fast Retina Key-point *(FREAK)* descriptor's circular shape is based on the human retinal system

- As for the sampling pattern, the best pairs of pixels are learned using an of offline training algorithm to maximize the point-pairs variance and minimize the correlation

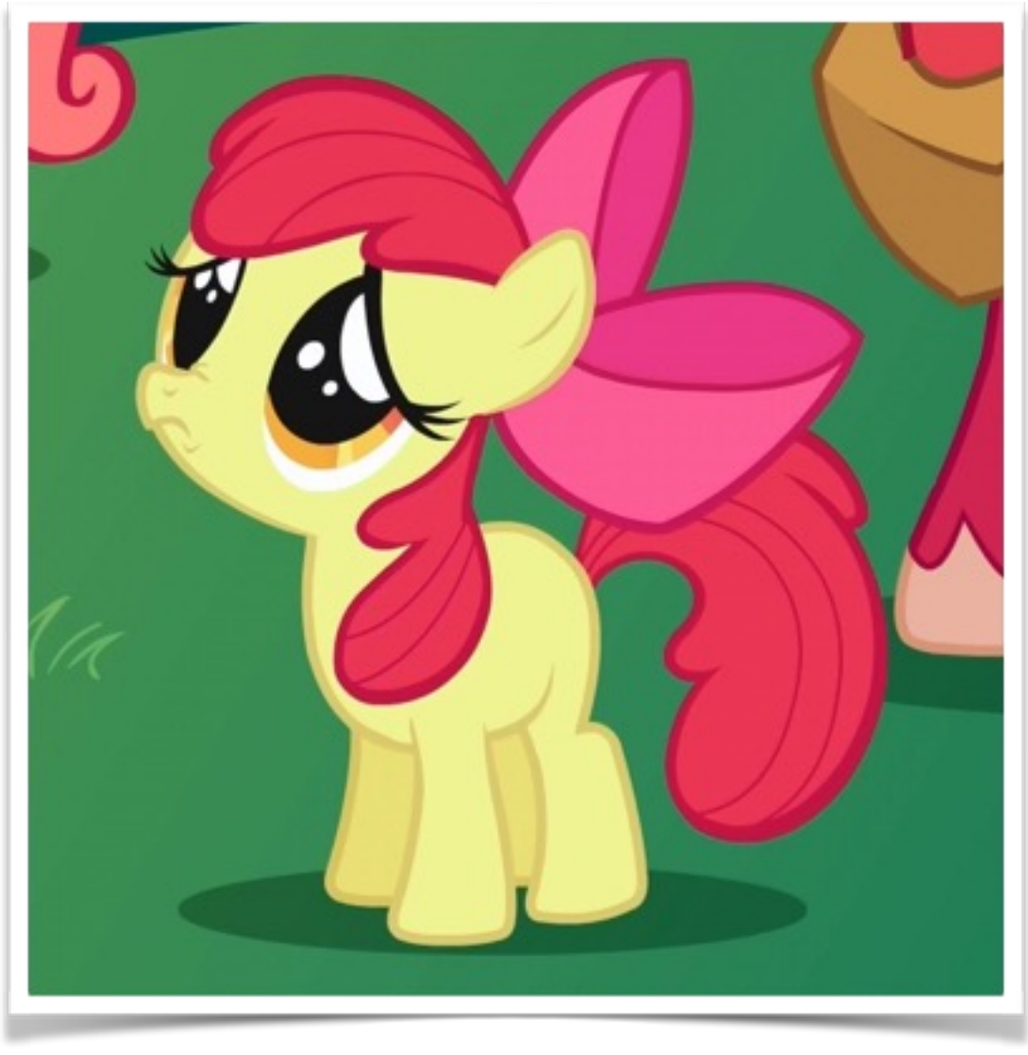
# SHOW ME THE CODE!!!



# FAST FEATURE DETECTOR

```
Mat& mGr  = *(Mat*)addrGray;
Mat& mRgb = *(Mat*)addrRgba;
vector<KeyPoint> v;

Ptr<FeatureDetector> detector = FastFeatureDetector::create(50);
detector->detect(mGr, v);
for( unsigned int i = 0; i < v.size(); i++ )
{
    const KeyPoint& kp = v[i];
    circle(mRgb, Point(kp.pt.x, kp.pt.y), 10, Scalar(255,0,0,255));
}
```

# WHAT'S ABOUT A DEMO?



# AN INTRODUCTION TO OPENCV

*Exploring the building blocks of OpenCV*

# IN A NUTSHELL

- OpenCV is an Image Processing library
- Available for C, C++, and Python
- Open Source and free
- Easy to use and install *(kinda)*

# BASIC OPENCV STRUCTURES

* `Point`, `Point2f` - 2D Point
* `Size` - 2D size structure
* `Rect` - 2D rectangle object
* `RotatedRect` - Rect object with angle
* `Mat` - image object

# 2D POINT

* Constructor - `(int x, y)`
* Functions
  * `Point.dot(<Point>)` - computes dot product
  * `Point.inside(<Rect>)` - returns true if point is inside

# SIZE

* Constructor - `(int width, height)`
* Functions
  * `Point.area()` - returns (width * height)

# MAT

- It stores images information and their components
- Main items:
  - `rows`, cols - length and width(int)
  - `channels` - 1: grayscale, 3: BGR
  - `depth`: CV_<depth>C<num chan>

# OPENCV CAPABILITIES

- Image data manipulation *(allocation, copying, setting, conversion)*
- Image and video I/O *(file/camera based in, image/video file out)*
- Basic image processing *(filtering, edge and corner detection, sampling and interpolation, color conversion, histograms)*
- Motion analysis *(optical flow, motion segmentation, tracking)*
- Object recognition *(eigen-methods, HMM)*

# GEOMETRICAL TRANSFORMATIONS

* `resize()` Resize image
* `getRectSubPix()` Extract an image patch
* `warpAffine()` Warp image affinely
* `convertMaps()` Optimize maps for a faster remap

# VARIOUS IMAGE TRANSFORMATIONS

* `cvtColor()` Convert image from different color spaces
* `threshold()` Convert grayscale image to binary image
* `floodFill()` Find a connected components

# IMAGE PROCESSING FILTERING

* `filter2D()` Non-separable linear filter
* `sepFilter2D()` Separable linear filter
* `boxFilter()` Blurs an image using the box filter
* `GaussianBlur()` Blurs an image using a Gaussian filter

# SUPPORTED PLATFORMS

# WINDOWS

# LINUX

# MACOS

# ANDROID

# IOS

# DESIGN PRINCIPLES

- OpenCV was designed to be cross-platform
- The library was written in C and this makes OpenCV portable to almost any commercial system
- Since version 2.0, OpenCV includes its traditional C interface as well as the new C++ one
- For the most part, new OpenCV algorithms are now developed in C++

# MUCH MORE COMFORTABLE

# OPENCV ANDROID SDK

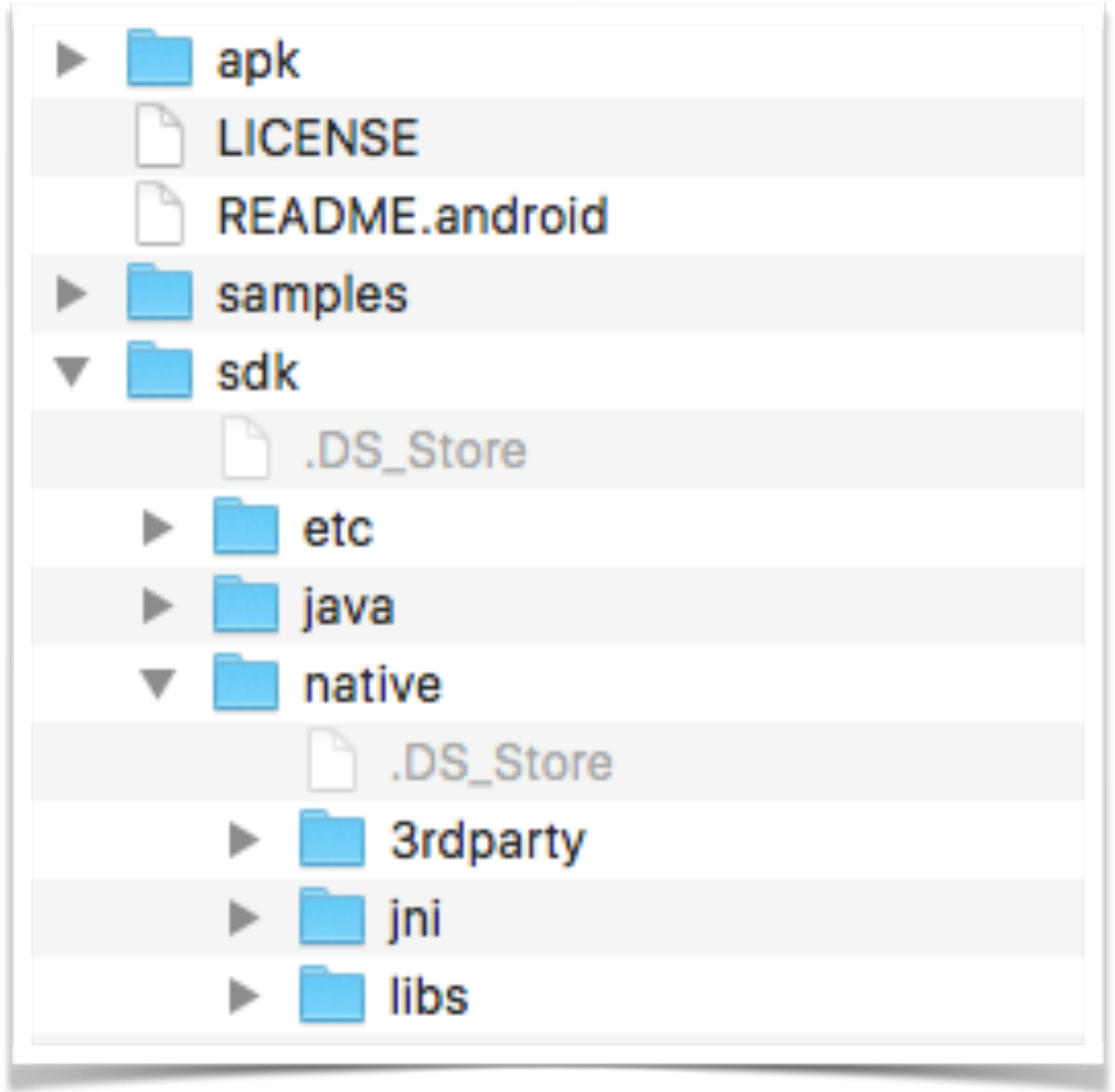
*A practical Java wrapper to OpenCV*

# STEP 1- GET THE IDK

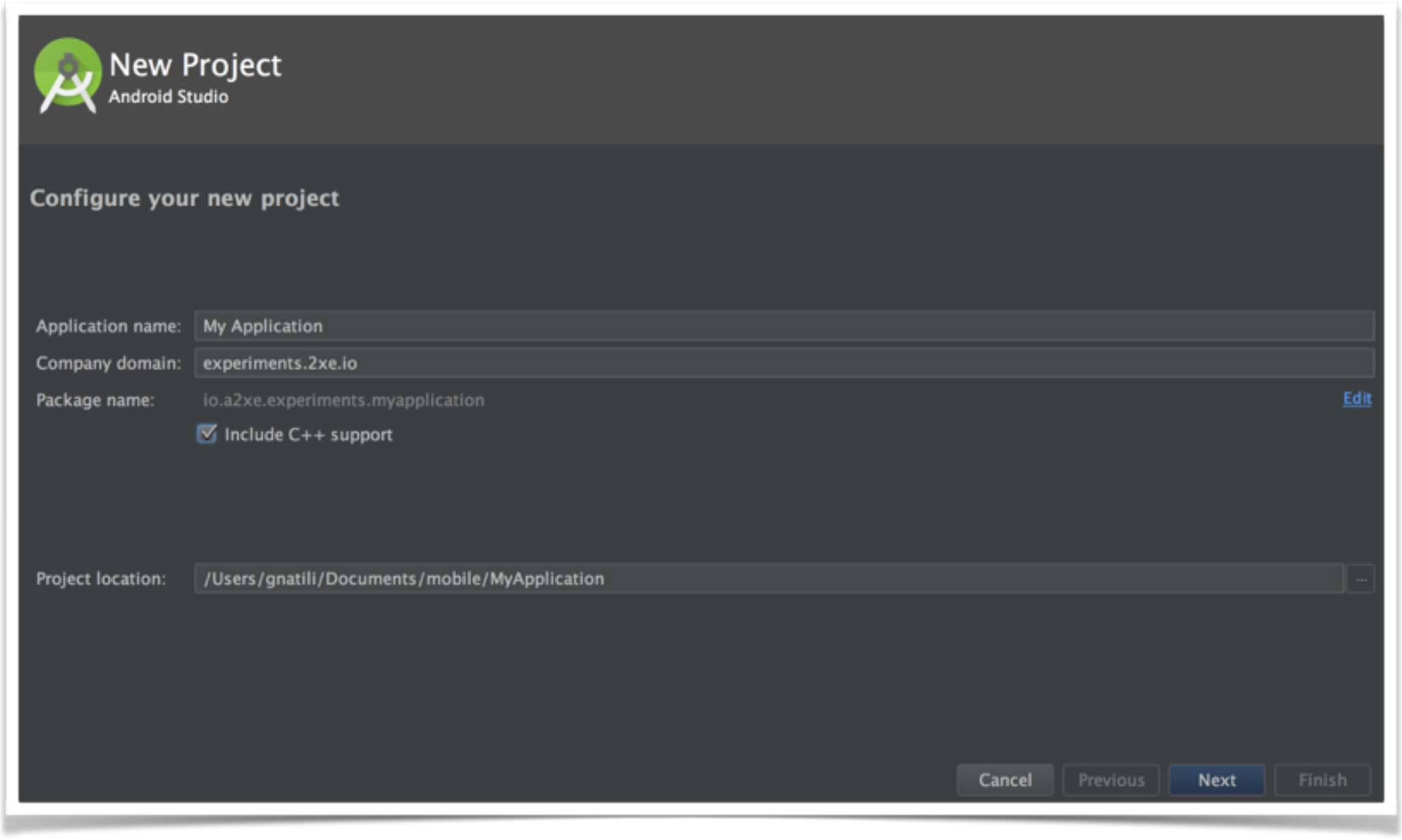
Download and unzip the Android SDK from OpenCV

bit.ly/opencv-330

# STEP 2 – EXPLORE THE FOLDERS



# STEP 3 – CREATE A NEW PROJECT (C++ SUPPORT)

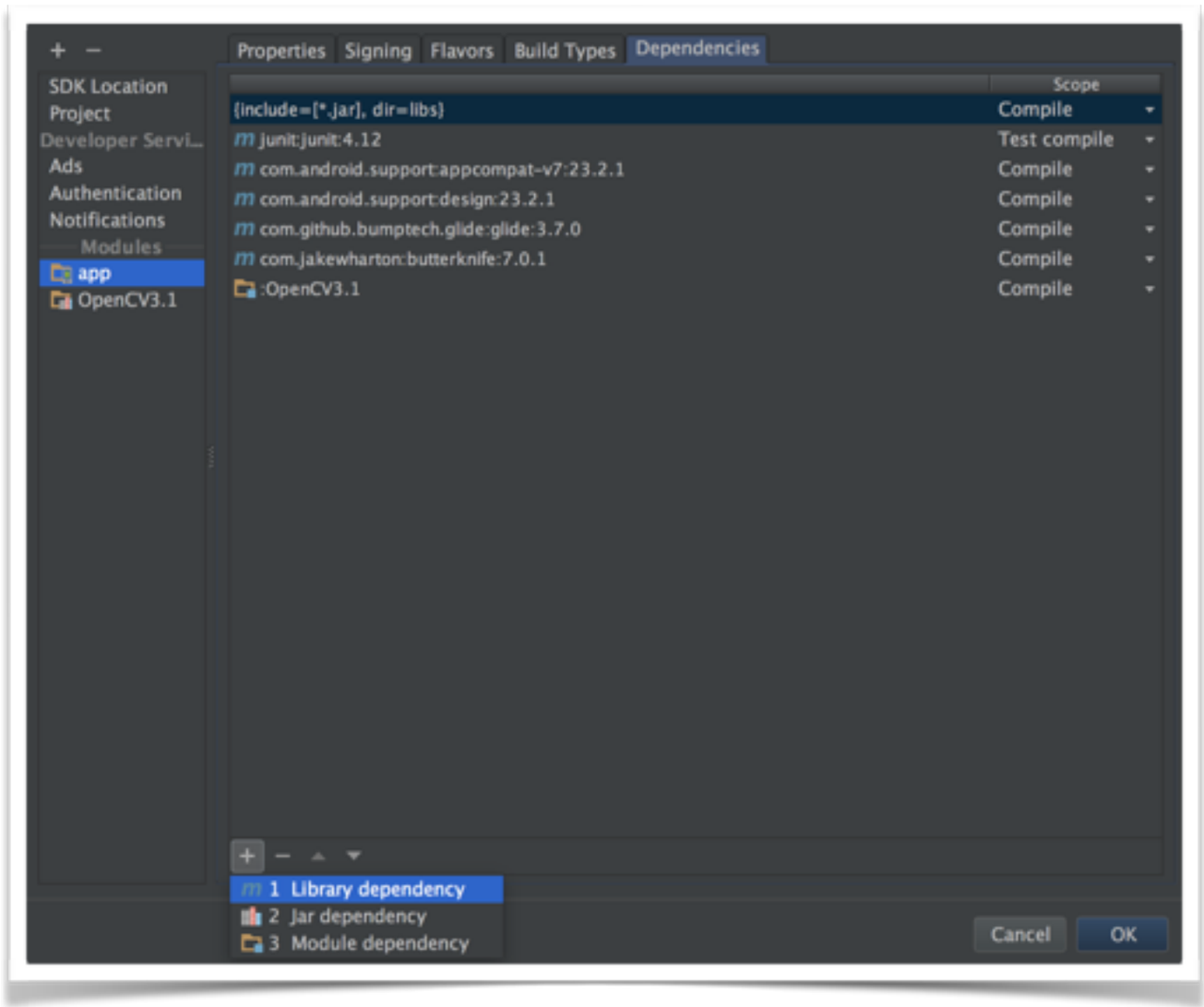


# STEP 4 – IMPORT THE ANDROID SDK

- File > New > Import Module
- Browse to the `./OpenCV-android-sdk/sdk/java` folder
- Accept the default options

# STEP 5 – ADD OPENCV AS A DEPENDENCY

- Select the app module and open the settings
- On the dependencies tab and add the imported module



# STEP 6 – CONFIGURE GRADLE AND THE IDK

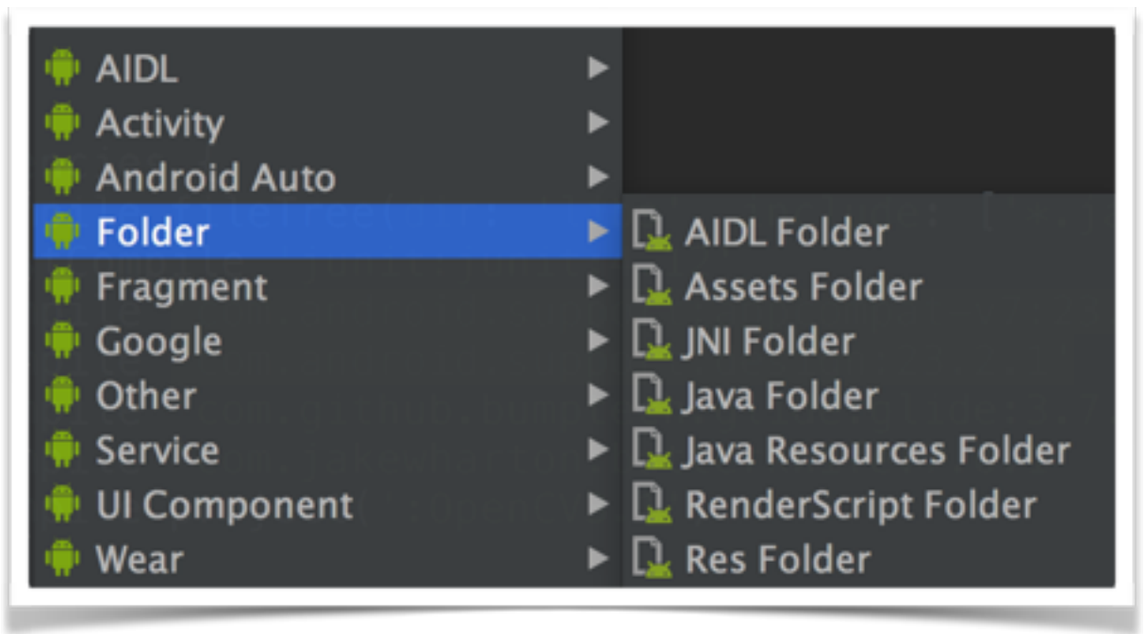
Open the `build.gradle` file into the imported module and update the SDK versions and build tools to match the settings of the app `build.gradle` file

```
compileSdkVersion 23
buildToolsVersion "23.0.2"

defaultConfig {
    applicationId "io.2xe.experiments.imageprocessing"
    minSdkVersion 17
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}
```

```
compileSdkVersion 23
buildToolsVersion "23.0.2"

defaultConfig {
    minSdkVersion 17
    targetSdkVersion 23
    versionCode 3100
    versionName "3.1.0"
}
```

# STEP 7 – ADD JNI REQUIRED FOLDERS

- Switch to the Android view in the project navigation
- Right click on the project and select the option New > Folder
- Select the JNI option and create a folder named `jniLibs`



# STEP 8 – NATIVE LAYER INCLUSION AND CONFIGURATION

- Go back to the `./OpenCV-android-sdk` folder and copy the folders contained in `./OpenCV-android-sdk/sdk/native/libs` into the `jniLibs` folder
- Remove all the `.a` files and keep only the `.so` files
- Locate the `gradle.properties` file and add the option `android.useDeprecatedNdk=true`

# IF YOU DON'T SEE THE FOLDERS COMMENT THE GRADLE FILE


```
externalNativeBuild {
    cmake {
        path "CMakeLists.txt"
    }
}
// sourceSets { main { jni.srcDirs = ['src/main/jni', 'src/main/jniLibs/'] } }
```

# STEP 9 – ADD A REFERENCE TO OPENCV

Open the file `CMakeLists.txt` and add a reference to OpenCV as a target library

```
target_link_libraries( # Specifies the target library.
    images-lib

    # Links the target library to the log library
    # included in the NDK.
    ${log-lib} lib_opencv)
```

# STEP 10 – LOAD OPENCV IN THE APP

```
static {

  if(!OpenCVLoader.initDebug()) {

    Log.d(TAG, "OpenCV Not Loaded");
  } else {

    Log.d(TAG, "OpenCV Loaded!!!!!");
  }
}
```

# RUN THE APP

# ANDROID & C++

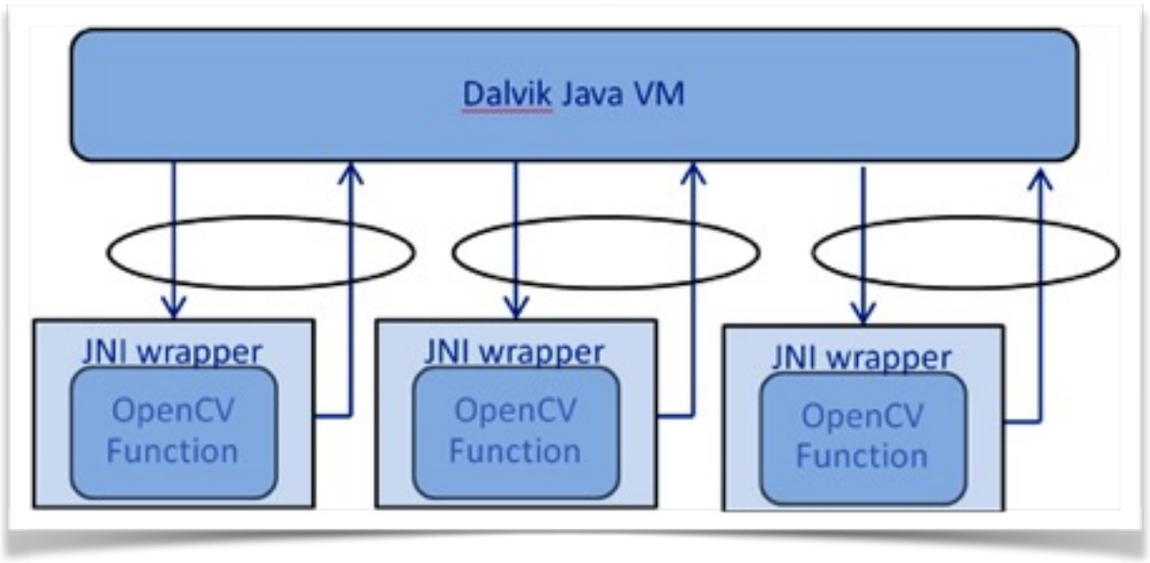
*Never-ending and high-performing extensions of an Android app*

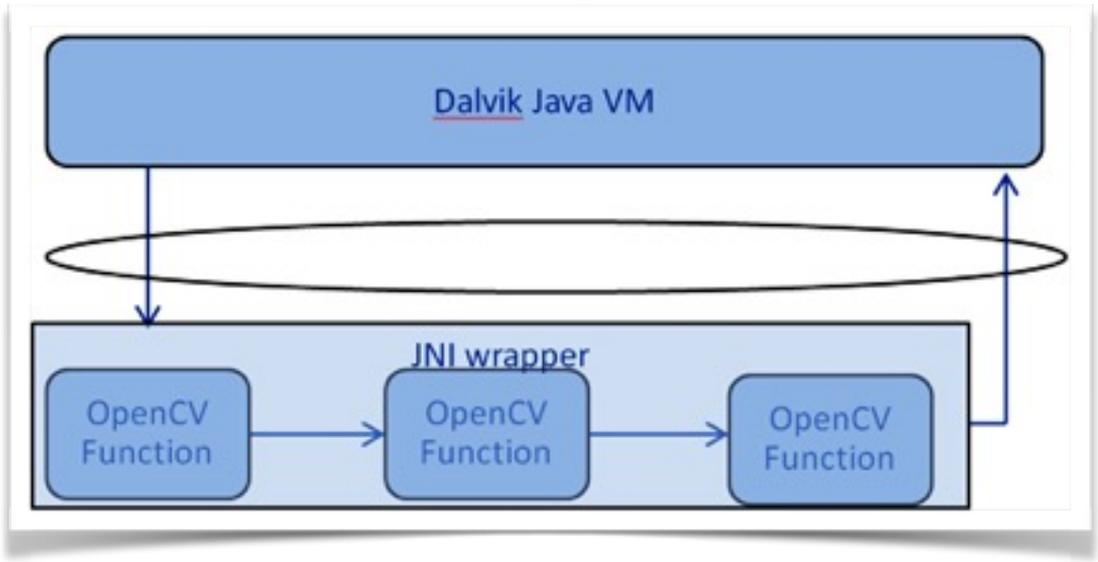
# BOOST YOUR APPS

- Add existing libraries to support advanced features
- Customize existing APIs to match your app workflow
- Write cross platform code *(seriously ?!?)*
- Dramatically improve performances

# PERFORMANCE ANALYSIS

JNI calls using the OpenCV Java API




JNI calls using OpenCV through C++

# ADD C++ CODE

- Go back on the project and open the MainActivity
- Search for a static native function, add another one with a different name but the same signature
- Use Android Studio to create the missing function (`alt + enter`)

# ADD C++ CODE

- Search for a file named `native-lib.cpp`
- The file, originally created by Android Studio, should already contains some code and should have added an empty function named like `Java_io_a2xe_experiments_opencv2integration_MainActivity_helloFromOpenCV`

# REFERENCE THE C++ CODE

```
add_library( # Sets the name of the library.
    images-lib

    # Sets the library as a shared library.
    SHARED

    # Provides a relative path to your source file(s).
    src/main/cpp/images-lib.cpp )

target_link_libraries( # Specifies the target library.
    images-lib

    # Links the target library to the log library
    # included in the NDK.
    ${log-lib} lib_opencv)
```

# LOAD THE C++ CODE

```
static {
  System.loadLibrary("native-lib");
  System.loadLibrary("images-lib");
}
```

# RUN THE APP

# ANDROID AND COMPUTER VISION

*Bringing the magic into a mobile device*

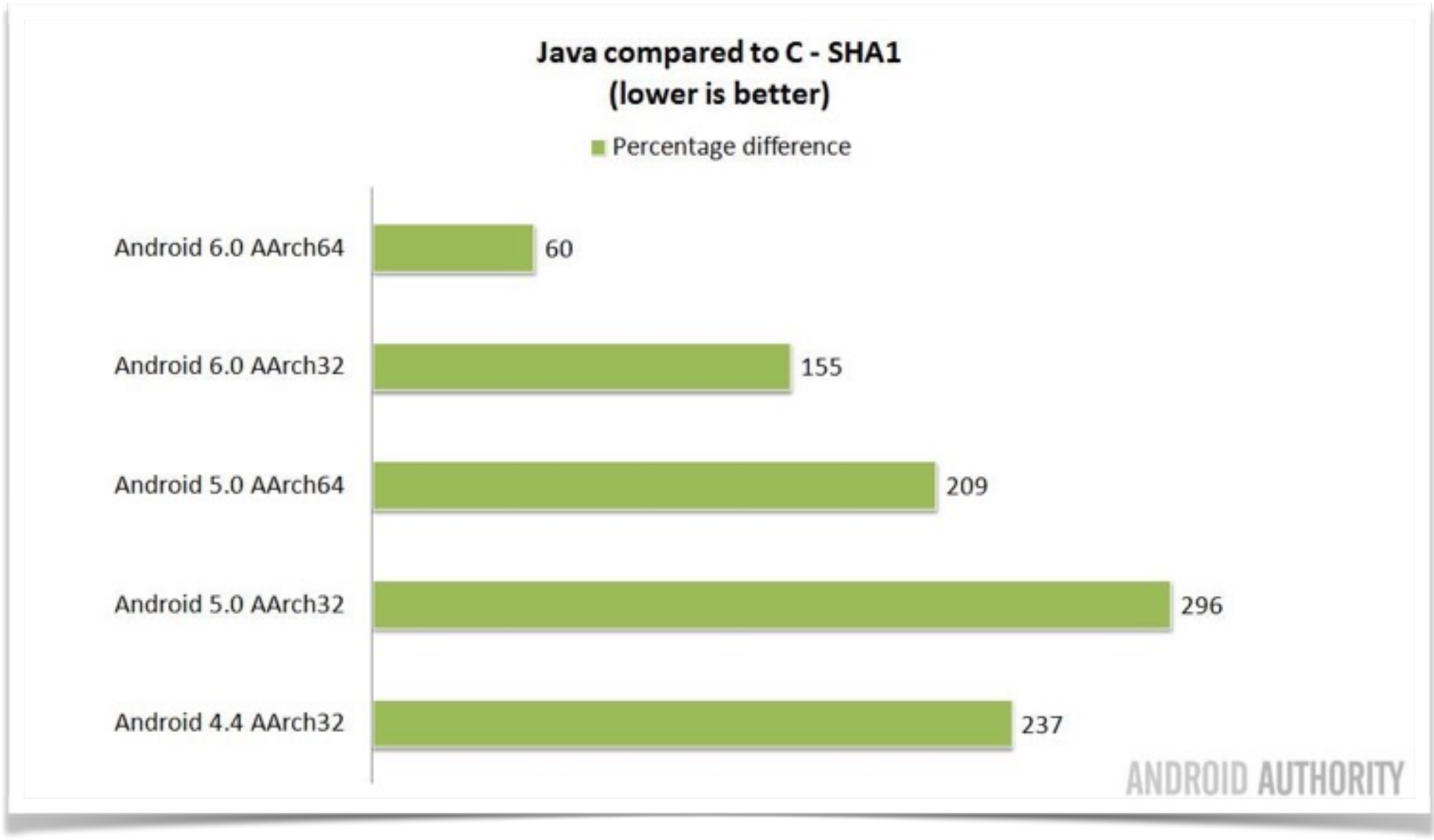
# APP ARCHITECTURE

- The application structure doesn't change too much from whatever clean architecture you prefer
- There are two folders that start to be crucial: the `cpp` and the `jniLibs` ones
- Keeping the file and the logic well organized is crucial
- Separate c++ files and then include them in the main one you reference into the `CMake.txt` file

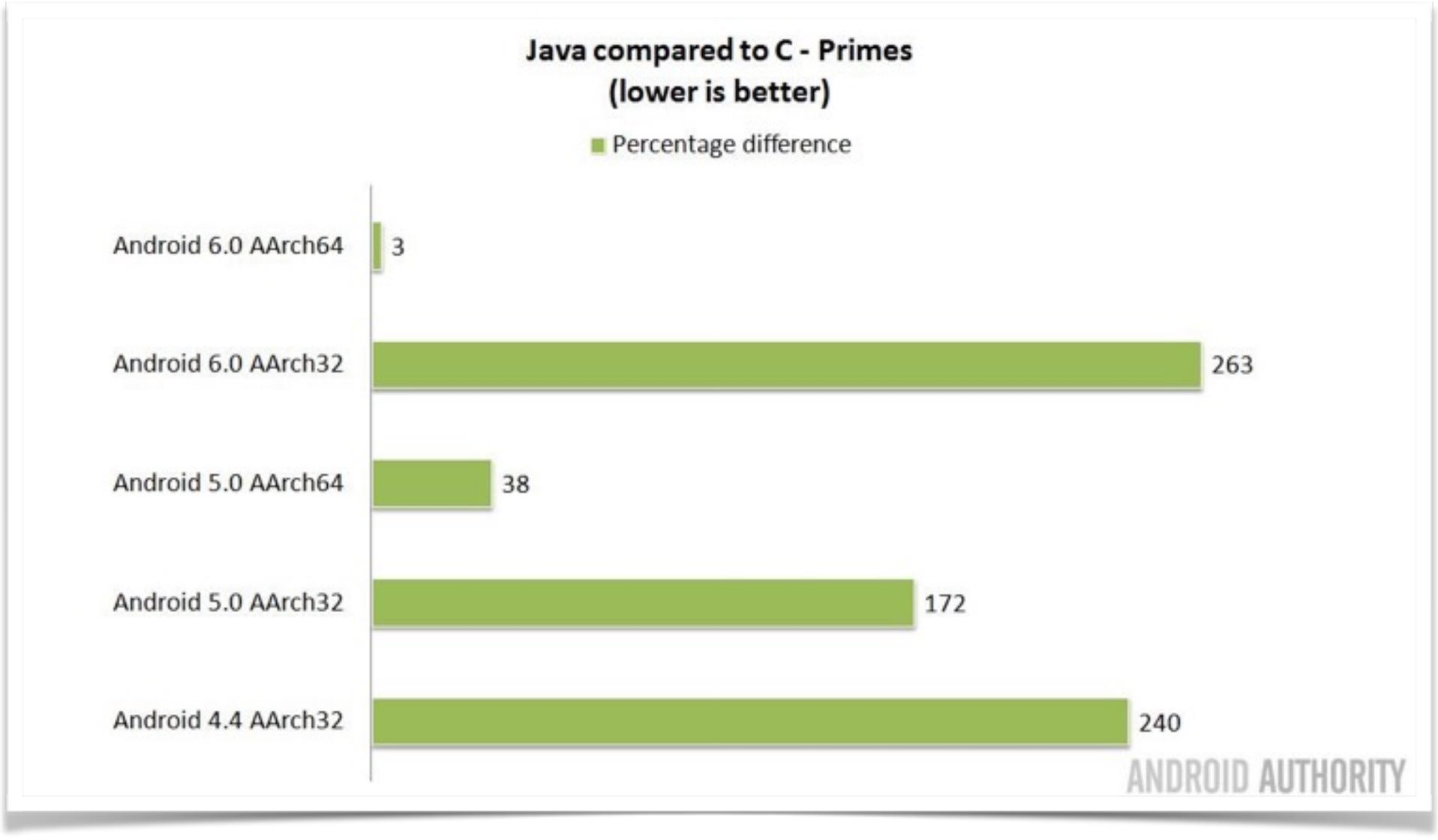
# PERFORMANCES

- As already mentioned, the less you use a wrapper, the faster will be the execution of the c++ functions
- The c++ libraries and files are execute much faster on all the current Android distributions

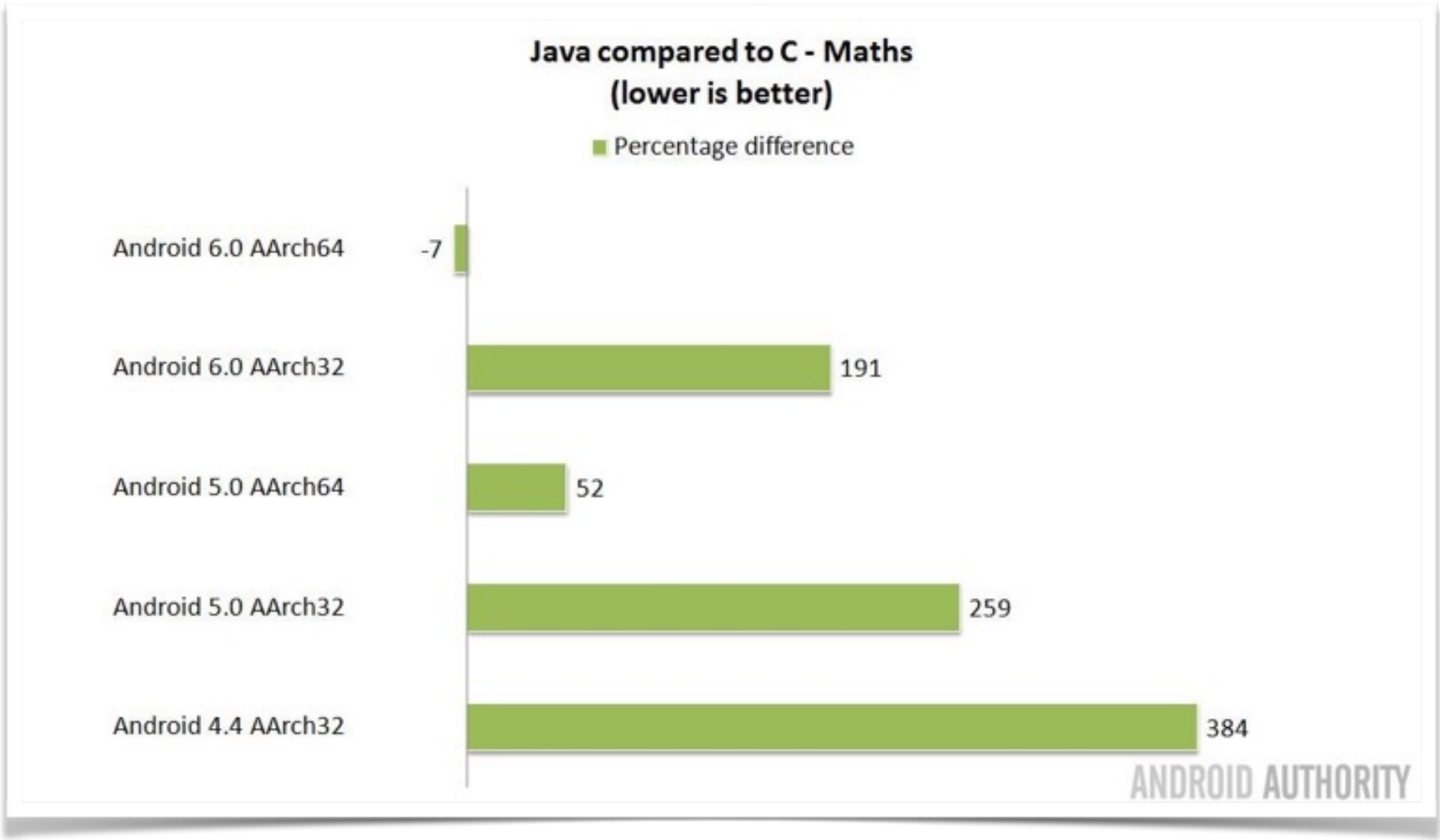
# BENCHMARK


Java compared to C - SHA1
(lower is better)
Percentage difference
Android 6.0 AArch64
60
Android 6.0 AArch32
155
Android 5.0 AArch64
209
Android 5.0 AArch32
296
Android 4.4 AArch32
237
ANDROID AUTHORITY

# BENCHMARK


Java compared to C - Primes
(lower is better)
Percentage difference
Android 6.0 AArch64
3
Android 6.0 AArch32
263
Android 5.0 AArch64
38
Android 5.0 AArch32
172
Android 4.4 AArch32
240
ANDROID AUTHORITY

# BENCHMARK


Java compared to C - Maths
(lower is better)
Percentage difference
Android 6.0 AArch64
-7
Android 6.0 AArch32
191
Android 5.0 AArch64
52
Android 5.0 AArch32
259
Android 4.4 AArch32
384
ANDROID AUTHORITY

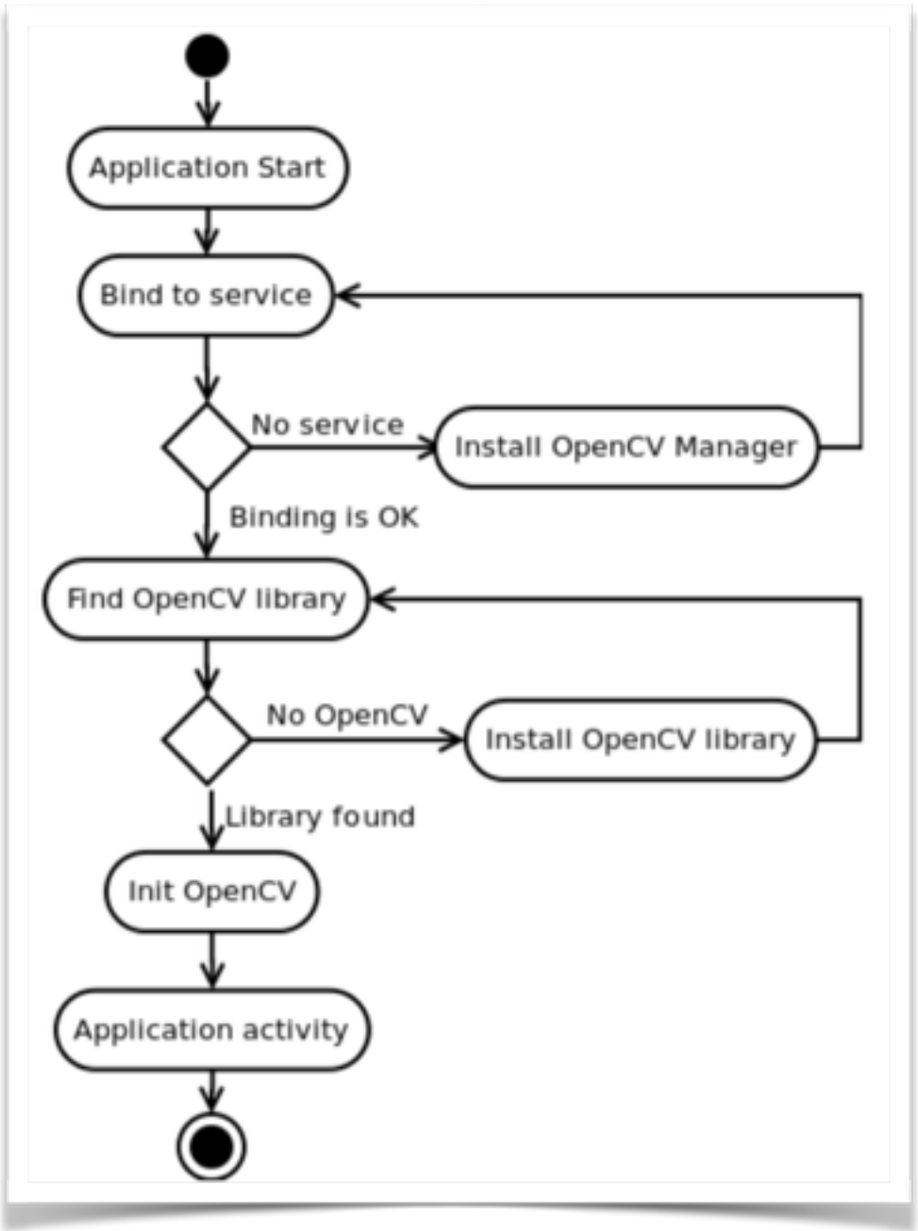
# RECOMMENDATIONS

- Don’t start to use only C++ in your projects
- Prototype fast with a wrapper *(when available)* and then determine if a direct call to C++ is beneficial for the app performances
- Be very precise when versioning your dependencies; embed them *(static initialization)* or use the OpenCV Manager

# ANDROID OPENCV MANAGER

- Less memory usage, all apps use the same binaries from service and do not keep native libs inside themselves
- Hardware specific optimizations for all supported platforms
- Trusted OpenCV library source, all packages with OpenCV are published on Google Play market
- Regular updates and bug fixes distributed on your behalf

# OPENCV MANAGER FLOW



# START CODING

*From theory to practice*

# (TRAFFIC) LIGHTS

# STEPS

- Read the `Bitmap` data from an image
- Convert the `Bitmap` data to a `Mat`
- Convert the `Mat` to gray scale
- Find the minimum and maximum values in the result `Array`
- Add a circle to the result `Array`

# DETECT A TRAFFIC LIGHT

```
Mat rgba = new Mat();
Utils.bitmapToMat(bitmap, rgba);

Mat grayScaleGaussianBlur = new Mat();
Imgproc.cvtColor(rgba, grayScaleGaussianBlur,
Imgproc.COLOR_BGR2GRAY);

Imgproc.GaussianBlur(grayScaleGaussianBlur,
grayScaleGaussianBlur, new Size(gaussianBlurValue,
gaussianBlurValue), 0);

Core.MinMaxLocResult minMaxLocResultBlur =
Core.minMaxLoc(grayScaleGaussianBlur);

Imgproc.circle(rgba, minMaxLocResultBlur.maxLoc, 30, new
Scalar(255), 3);
```

# DRIVING LANES 

# STEPS – DETECT EDGES

- Convert a `Bitmap` to a `Mat` structure
- Convert the `Mat` to gray scale
- Apply the Canny Edges algorithm
- Convert the `Mat` back to a `Bitmap`

# DETECT EDGES

```
Mat rgba = new Mat();
Utils.bitmapToMat(bitmap, rgba);

Mat edges = new Mat(rgba.size(), CvType.CV_8UC1);
Imgproc.cvtColor(rgba, edges, Imgproc.COLOR_RGB2GRAY, 4);
Imgproc.Canny(edges, edges, 70, 100);

Utils.matToBitmap(edges, resultBitmap);
```

# STEPS - RENDER HOUGH TRANSFORM

- Convert a `Bitmap` to a `Mat` structure
- Apply the Canny Edges algorithm
- Determine the lines in the Hough space accordingly to the detected edges
- Iterate the `Array` and draw the lines

# HOUGH TRANSFORM

```
Mat rgba = new Mat();
Utils.bitmapToMat(bitmap, rgba);

Mat edges = new Mat();
Mat mat = new Mat();

Mat lines = new Mat();

Utils.bitmapToMat(bitmap, mat);

Imgproc.Canny(mat, edges, 50, 90);

int threshold = 20;
int minLineSize = 38;
int lineGap = 30;

Imgproc.HoughLinesP(edges, lines, 1, Math.PI / 180, threshold, minLineSize,
lineGap);

for (int x = 0; x < lines.rows(); x++){
  // Iterate the array and draw the lines
}

Bitmap bmp = Bitmap.createBitmap(rgba.cols(), rgba.rows(),
Bitmap.Config.ARGB_8888);
Utils.matToBitmap(rgba, bmp);
```

# FEATURES DETECTION

# STEPS

- Create a feature detector
- Detect the features in the image
- Add a circle for each feature in the RGB `Mat` being processed

# HOUGH TRANSFORM

```
Mat& mGr  = *(Mat*)addrGray;
Mat& mRgb = *(Mat*)addrRgba;
vector<KeyPoint> v;

Ptr<FeatureDetector> detector =
FastFeatureDetector::create(50);
detector->detect(mGr, v);
for( unsigned int i = 0; i < v.size(); i++ )
{
  const KeyPoint& kp = v[i];
  circle(mRgb, Point(kp.pt.x, kp.pt.y), 10,
Scalar(255,0,0,255));
}
```

# RUN THE DEMO

# CHALLENGES

*The main problems still to solve*

# UNKNOWN TRANSFORMATION

- Device camera could have different orientation accordingly to the setup
- Unknown transformations should then be applied to the data source
- Calculate and measure all the possible transformations is something that shouldn’t happen on the device

# DIFFERENT SOURCES

- Every device has a different camera with different specifications
- It’s important to determine which optimizations to apply based on the camera

# MOVEMENT

- The camera is never static, it moves with the car and this creates more issues when comparing objects
- The camera is not supposed to be stable enough to execute straights images comparison

# DATA MASHUP

- Mixing data with Android Automotive hardware abstraction layer API
- Using existing data set to train the model in advance
- When connected to internet gather environmental information to tune the camera and the noise reduction strategies

# QUESTIONS

*and potentially answers :)*

*“If you can count on not having an accident, you can get rid of a huge amount of the crash structure and the airbags.”*

*- Elon Musk*

# THANKS!

*@giorgionatili*

*g@2xe.io*