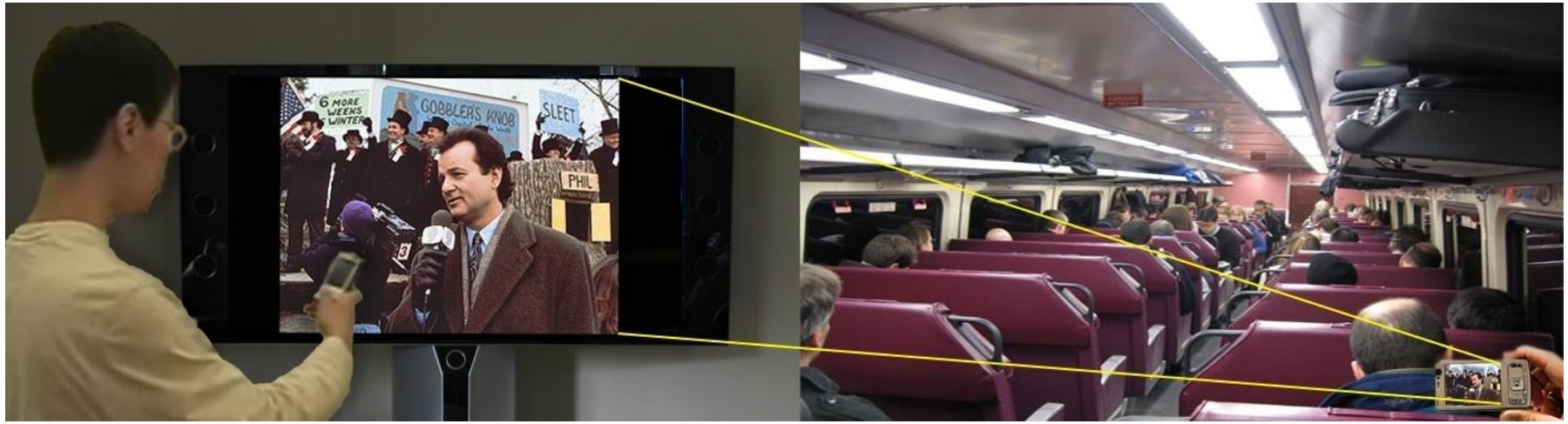
Mobile Image Processing

- Examples of mobile image processing
- Android platform
- Class resources for Android
- Eclipse integrated development environment
- Augmentation of viewfinder frames
- Debugging with DDMS perspective
- Taking a device screenshot
- Creating an Android emulator
- Mixed programming with Android JNI
- OpenCV library for Android
- Past class projects

Recognizing video at a glance



(1) User snaps a photo of screen.

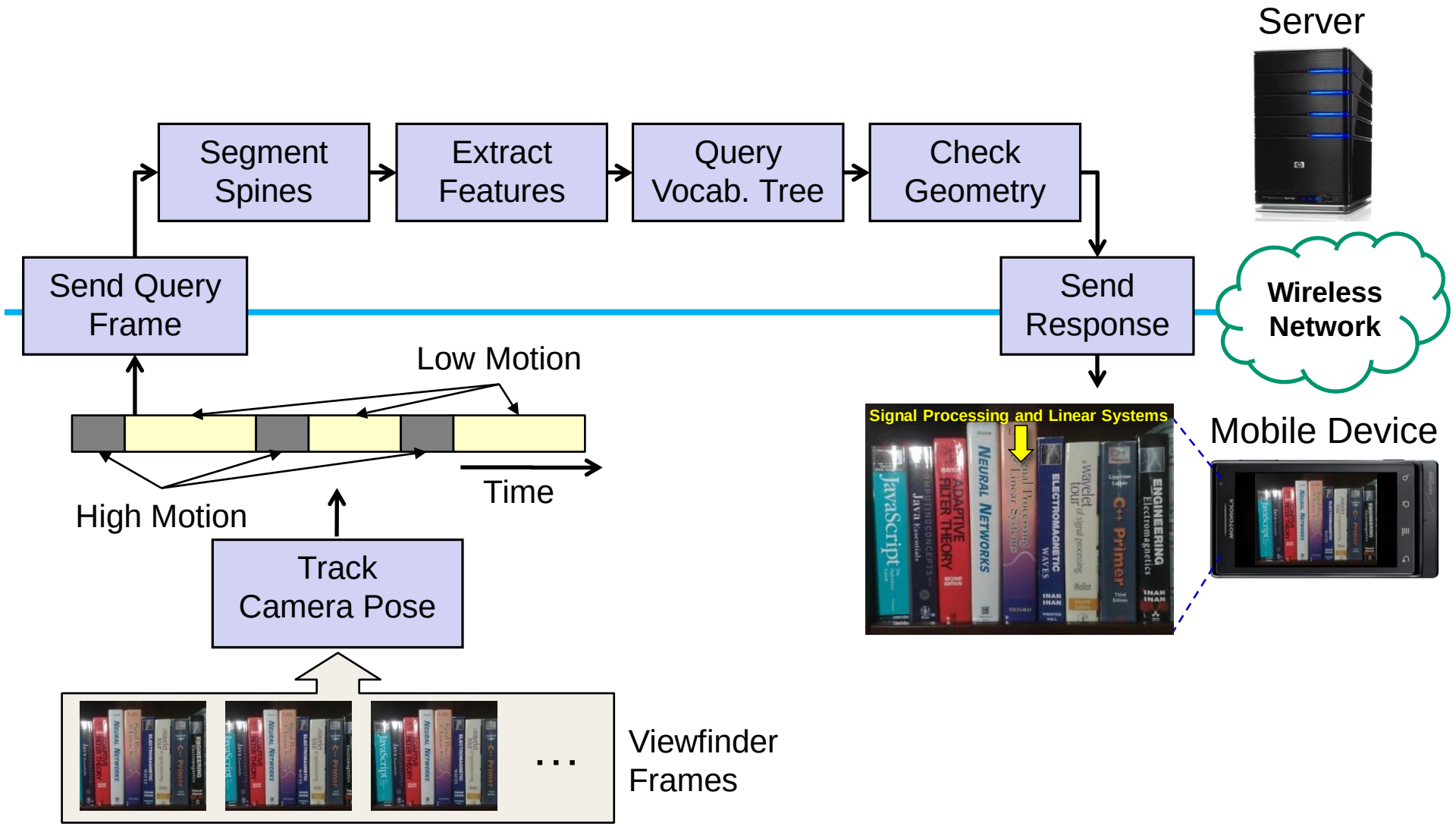
(3) User resumes video on the phone.

(2) Our system identifies video and frame within the video.

Recognizing video at a glance

Mobile book spine recognition

Mobile book spine recognition



On-device image matching

On-device image matching

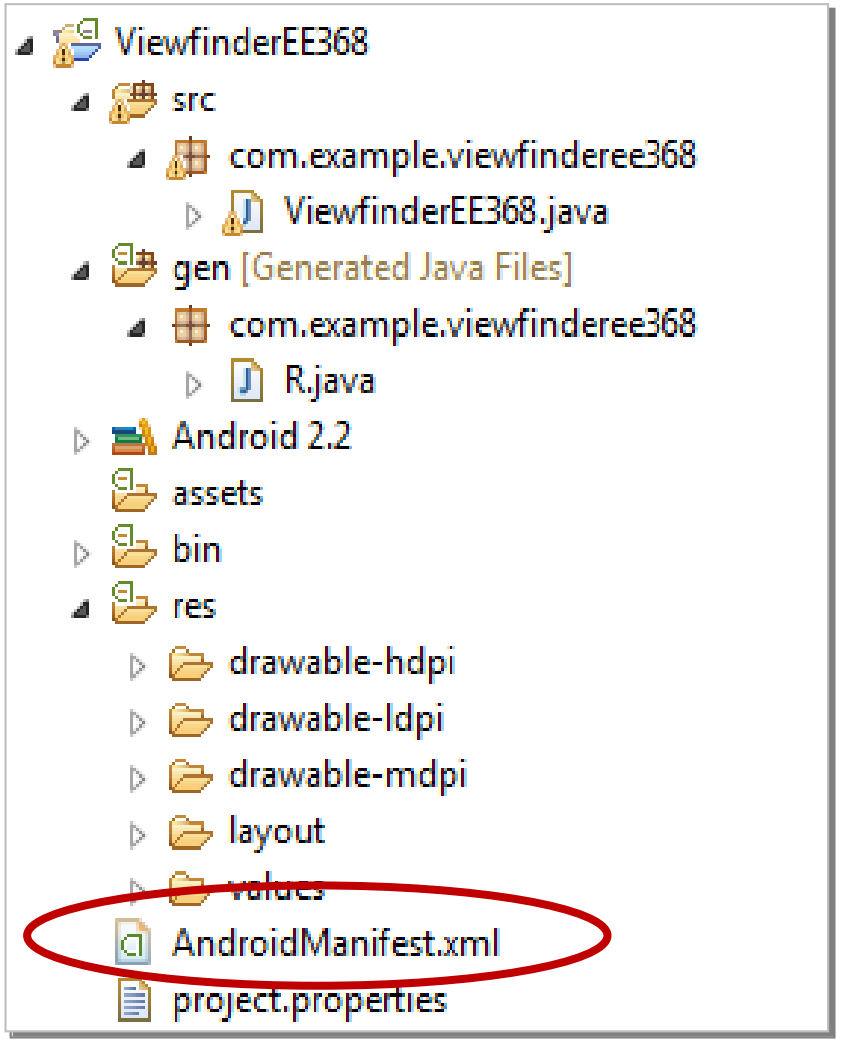
Motion-Adaptive Query Selection

Viewfinder Augmentation

Local Image Database

Local Database Search

Android manifest file



Android manifest file

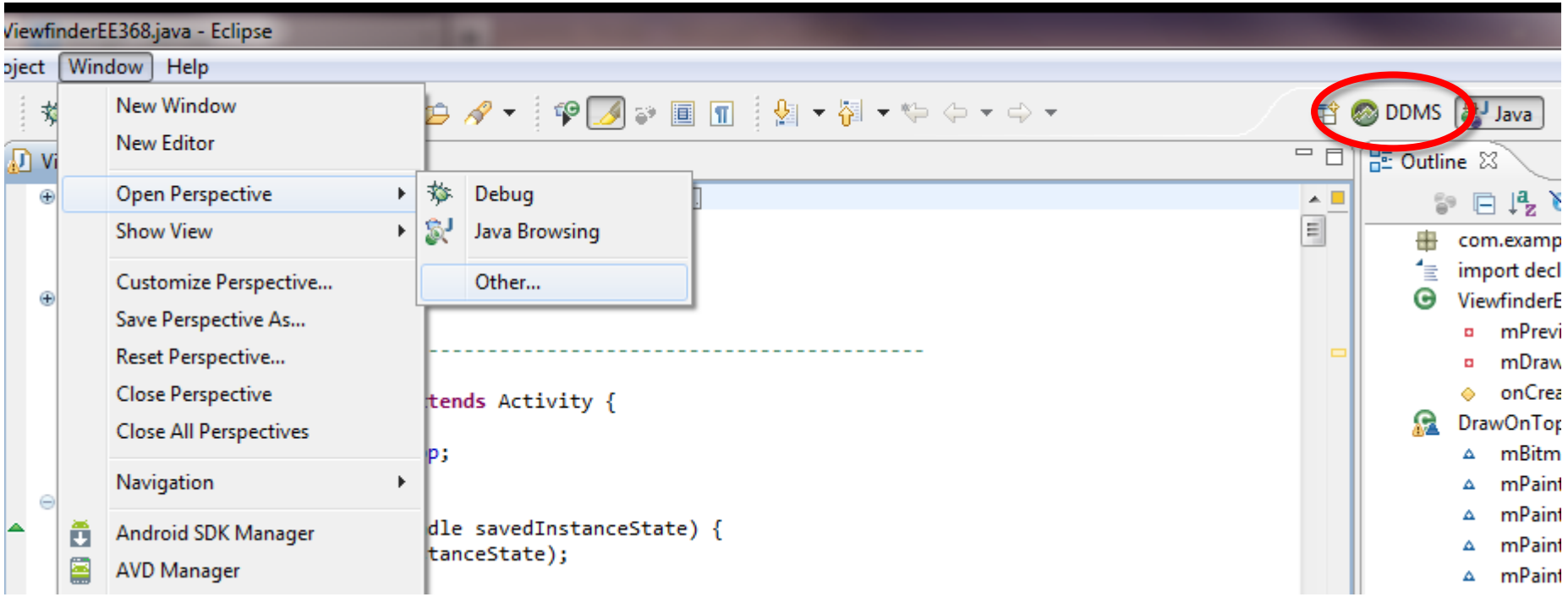
```
...
<application android:icon="@drawable/ic_launcher"
             android:label="@string/app_name">
<activity android:name=".ViewfinderEE368"
          android:label="@string/app_name"
          android:screenOrientation="landscape"
          android:theme="@android:style/Theme.NoTitleBar">
<intent-filter>
     <action android:name="android.intent.action.MAIN" />
     <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

<uses-sdk android:minSdkVersion="8" />
<uses-permission android:name="android.permission.CAMERA"/>

...
```

Set landscape orientation for viewfinder app

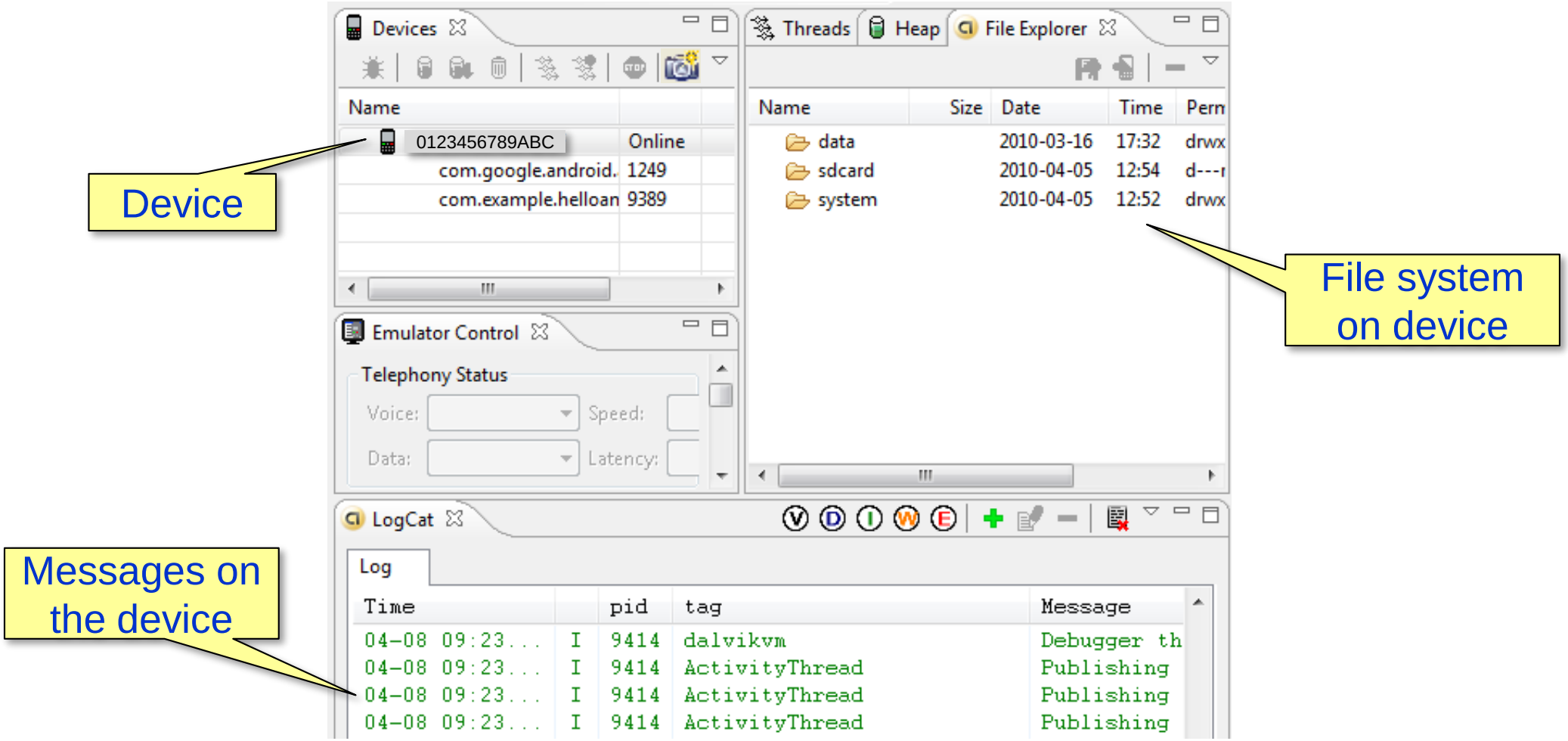
Set this activity as the application's main activity

Declare permission to use the device's camera

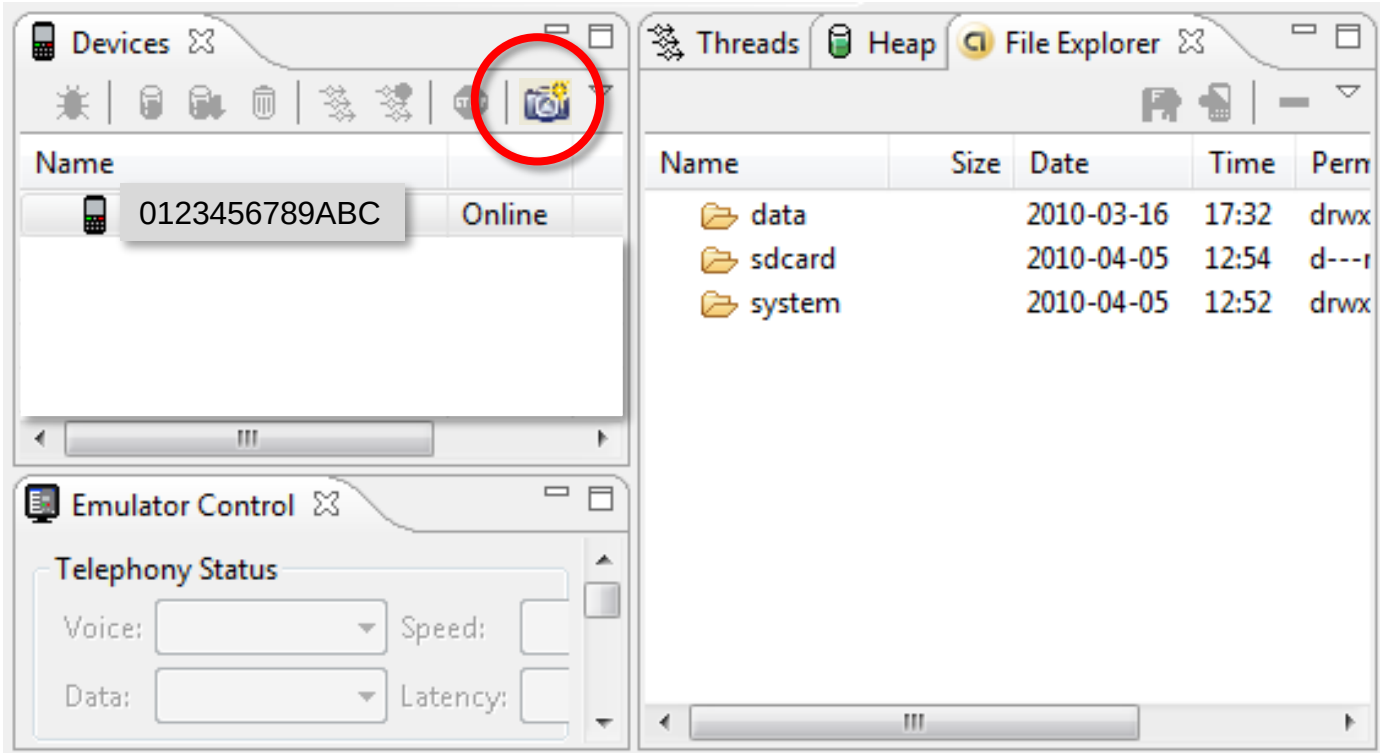
Real-time debugging with DDMS perspective



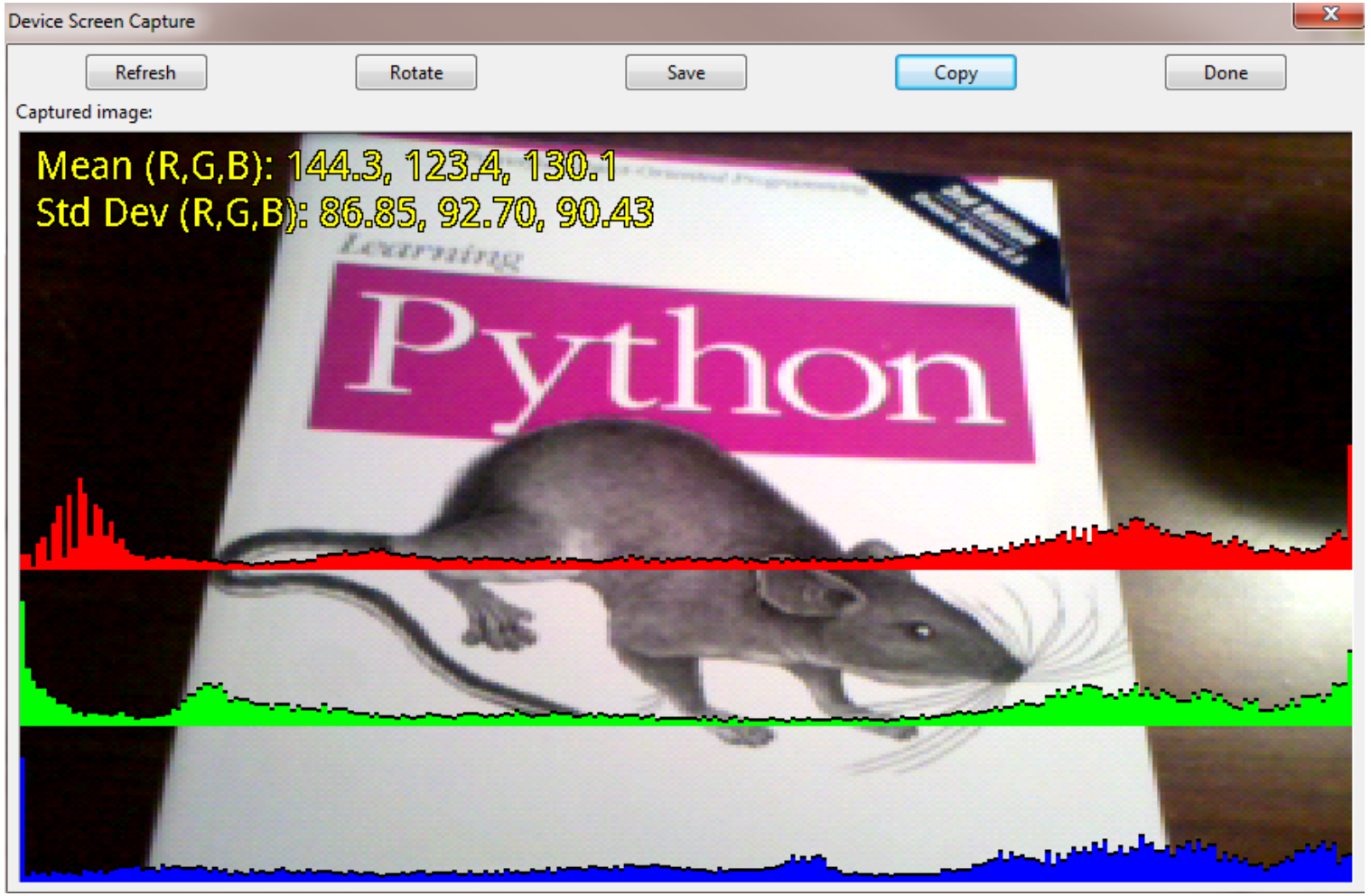
Real-time debugging with DDMS perspective



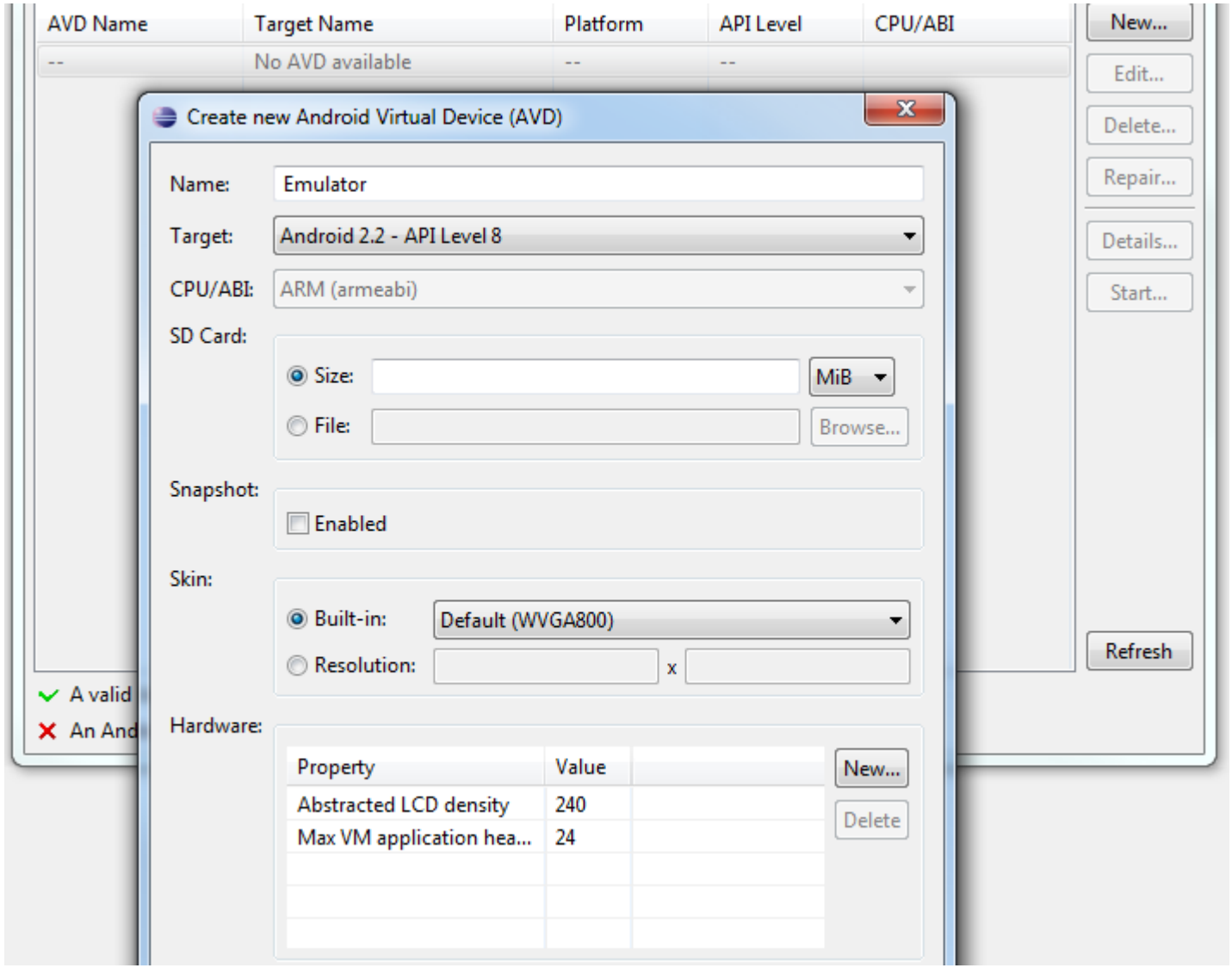
Taking a screenshot of the device

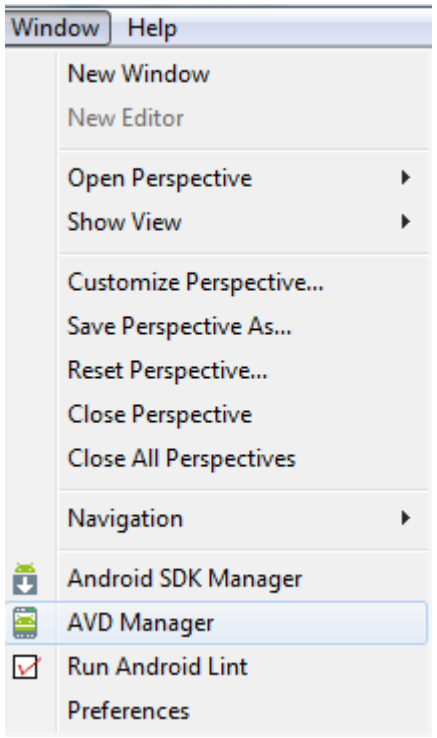


Taking a screenshot of the device

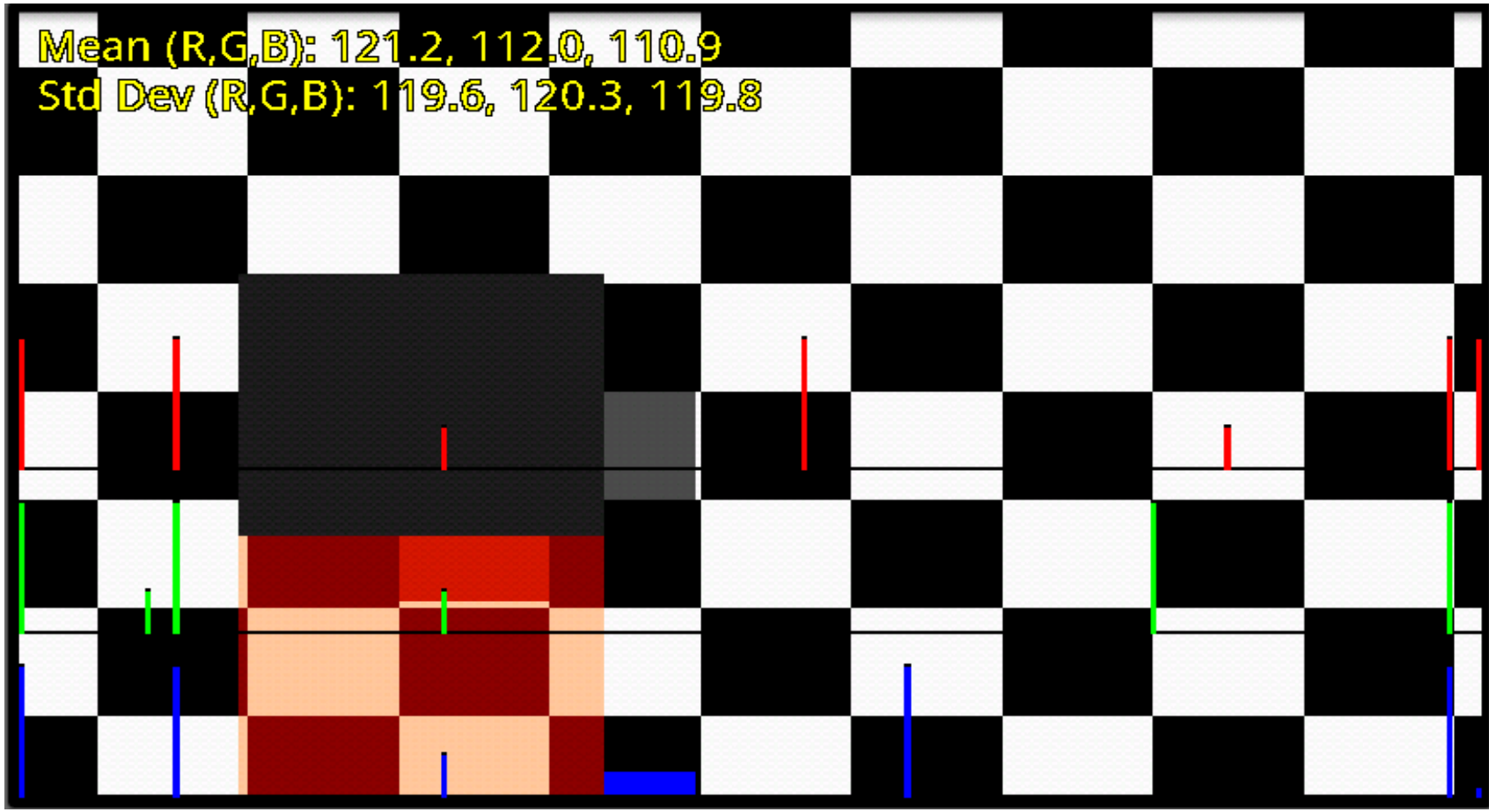


Creating an Android emulator





“Color Histograms” app running on emulator



OpenCV: Open Source Computer Vision

General Image Processing Functions

Geometric descriptors

Segmentation

Features

Transforms

Tracking

Machine Learning:
- Detection,
- Recognition

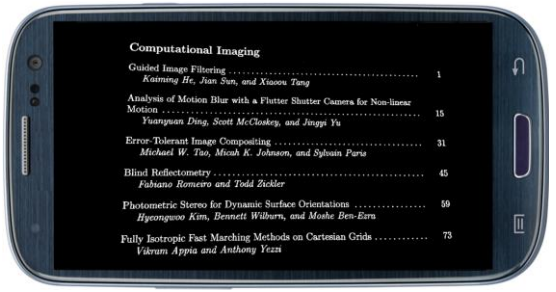
Matrix Math

Image Pyramids

Camera calibration, Stereo, 3D

Utilities and Data Structures

Fitting



http://docs.opencv.org

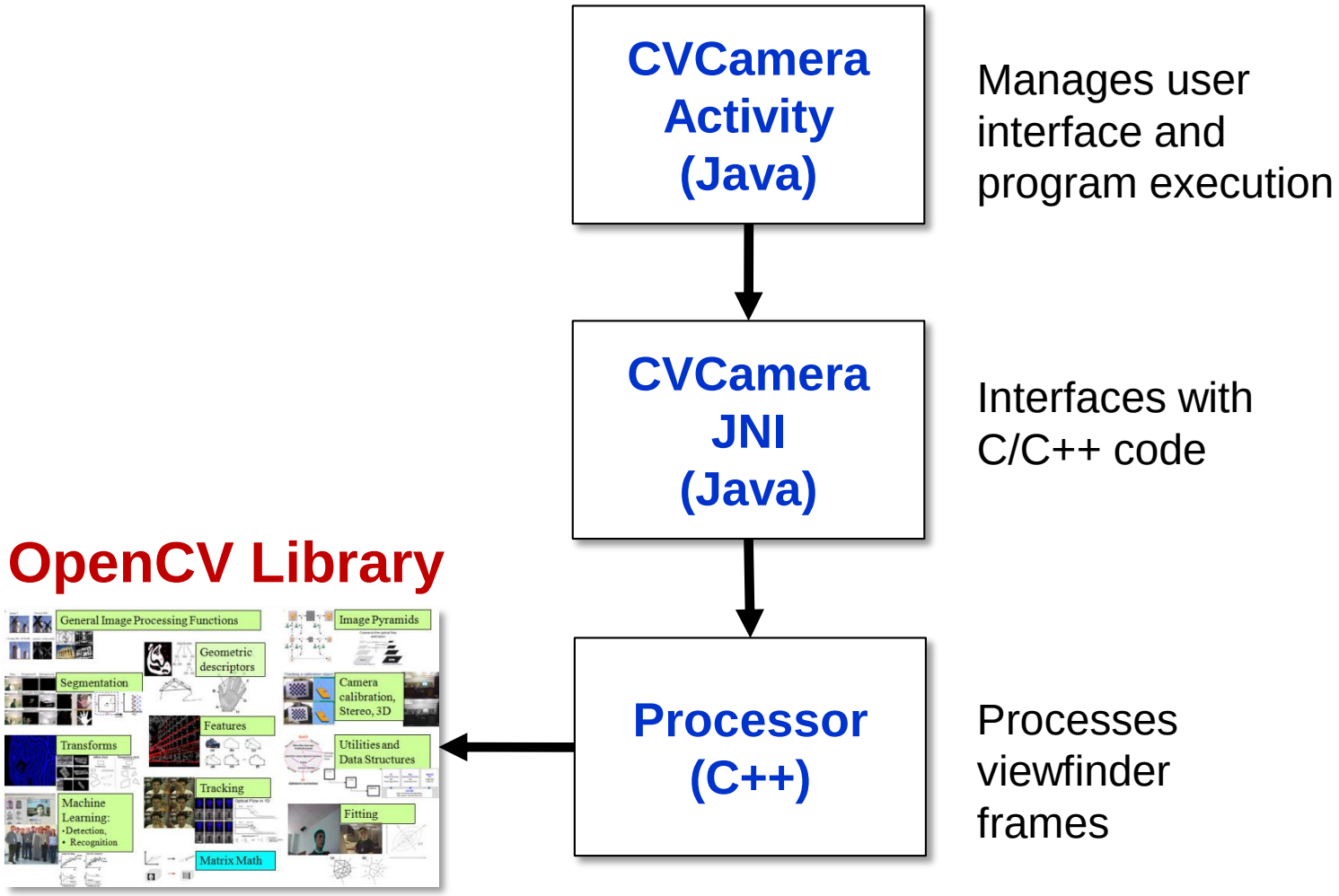
OpenCV for Android

- Port of OpenCV to Android platform
  - Over 2,500 optimized algorithms written in C/C++
  - Compiled with STL-enabled Android NDK
  - Enables popular CV functions to be used on mobile images/videos
- Differences from regular Android programming
  - Requires writing C/C++ code
  - Requires writing Java Native Interface (JNI) wrappers
  - Requires using Android NDK in addition to Android SDK
- Our Tutorial #2 helps to make the transition
  - How to set up OpenCV programming environment
  - How to write Android apps that call OpenCV functions
  - How to integrate NDK compilation into Eclipse IDE

“CVCamera” project

- Goals of this project
  - Learn how to incorporate C/C++ code into an Android project
  - Learn how to utilize OpenCV library functions
  - Learn how to draw feature keypoints on viewfinder frames
- Full source available on class website

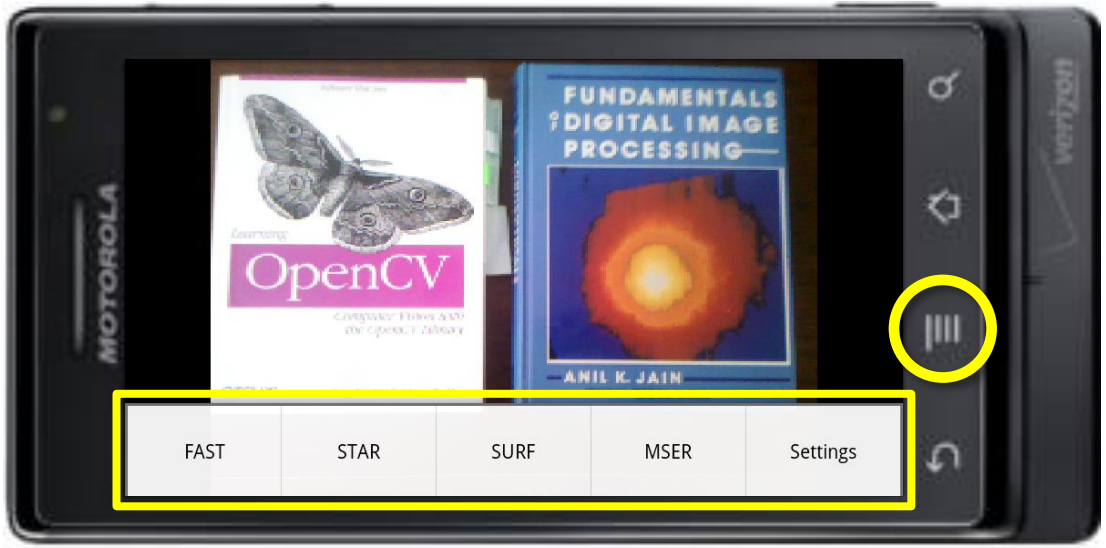
“CVCamera” class hierarchy



CVCamera class: menu options

```
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add("FAST");
    menu.add("STAR");
    menu.add("SURF");
    menu.add("MSER");
    menu.add("Settings");
    return true;
}
```

Java Code



CVCamera class: calling feature extractors

```
public boolean onOptionsItemSelected(MenuItem item) {
   LinkedList<PoolCallback> defaultCallbackStack =
         new LinkedList<PoolCallback>();

   defaultCallbackStack.addFirst(myGLView.getDrawCallback());

   if (item.getTitle().equals("FAST")) {
         defaultCallbackStack.addFirst(new FastProcessor());
   }
   else if (item.getTitle().equals("STAR")) {
         defaultCallbackStack.addFirst(new STARProcessor());
   }
   else if (item.getTitle().equals("SURF")) {
         defaultCallbackStack.addFirst(new SURFProcessor());
   }
   else if (item.getTitle().equals("MSER")) {
         defaultCallbackStack.addFirst(new MSERProcessor());
   }

   myPreview.addCallbackStack(defaultCallbackStack);
   return true;
}
```

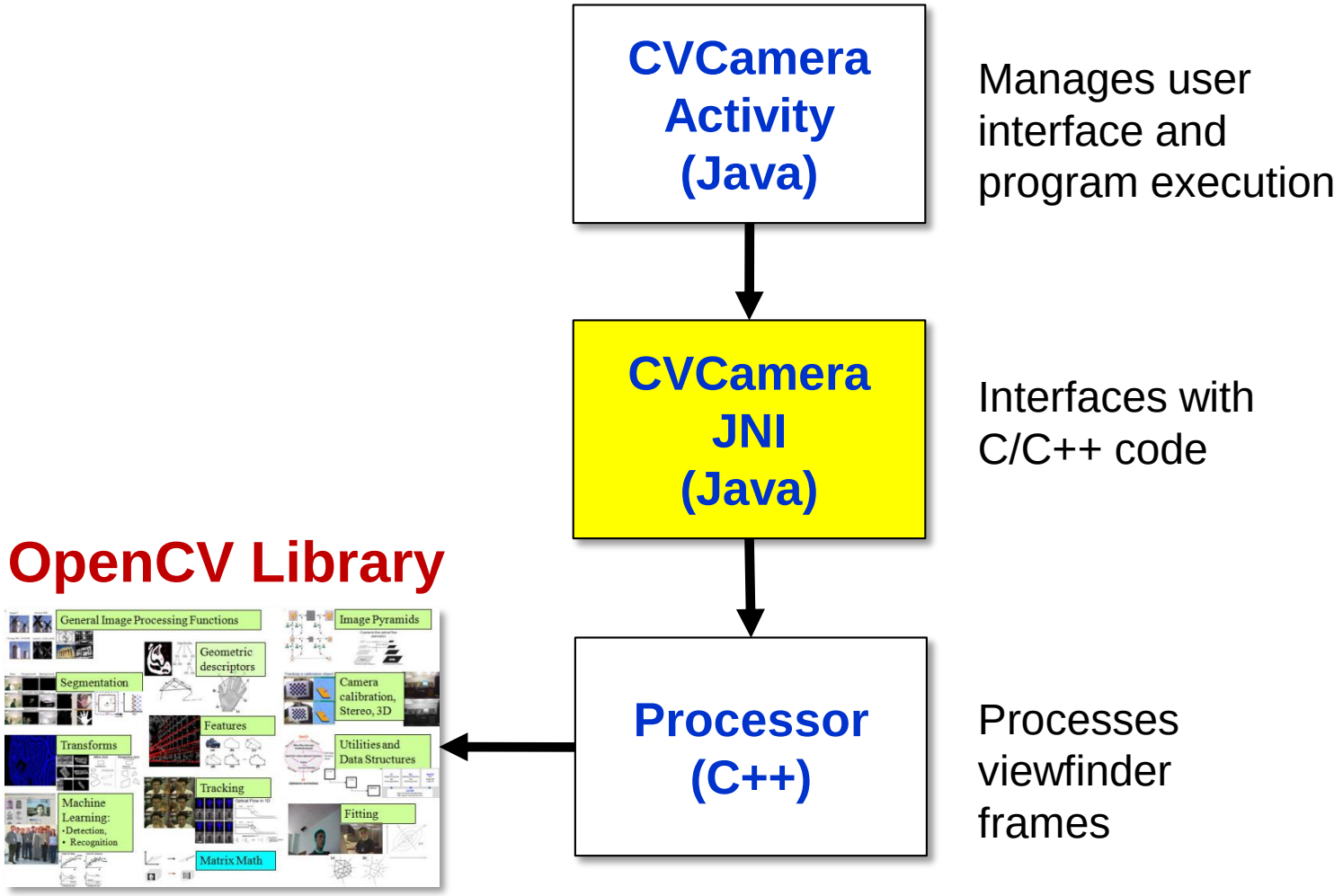
Java Code

Populate a stack of callback functions

First callback function draws the frames

Other callback functions call feature extractors

“CVCamera” class hierarchy



JNI class: interface to C/C++ code

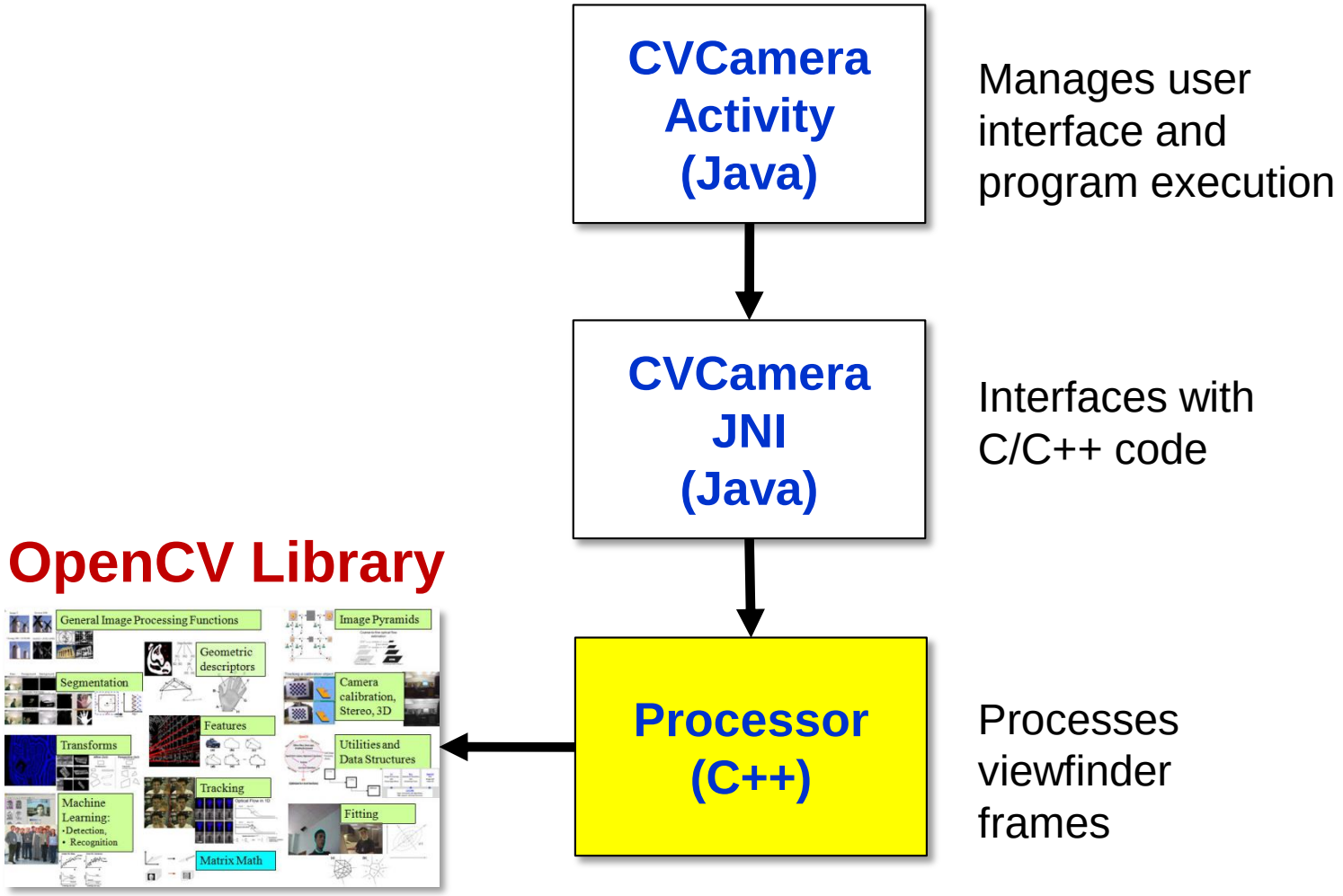
```
static {
   try {
         System.loadLibrary("android-opencv");
         System.loadLibrary("cvcamera");
   } catch (UnsatisfiedLinkError e) {
         throw e;
   }
}

public final static native int DETECT_FAST_get();
public final static native int DETECT_STAR_get();
public final static native int DETECT_SURF_get();
public final static native int DETECT_MSER_get();
public final static native long new_Processor();
public final static native void delete_Processor(long jarg1);
```

Java Code

Load libraries built by the Android NDK

Function prototypes for interface to C/C++ side

“CVCamera” class hierarchy



Processor class: initialize feature detectors

```
class Processor {
private:
  cv::StarFeatureDetector my_stard;
  cv::FastFeatureDetector my_fastd;
  cv::SurfFeatureDetector my_surfd;
  cv::MserFeatureDetector my_mserd;
  std::vector<cv::KeyPoint> my_keypoints;

public:
  Processor():
    my_stard(STAR detector parameters),
    my_fastd(FAST detector parameters),
    my_surfd(SURF detector parameters),
    my_mserd(MSER detector parameters)
    {}
  void detectAndDrawFeatures
    (int idx, image_pool* pool, int feature_type);
};
```

C++ Code

Different feature extractors stored as members

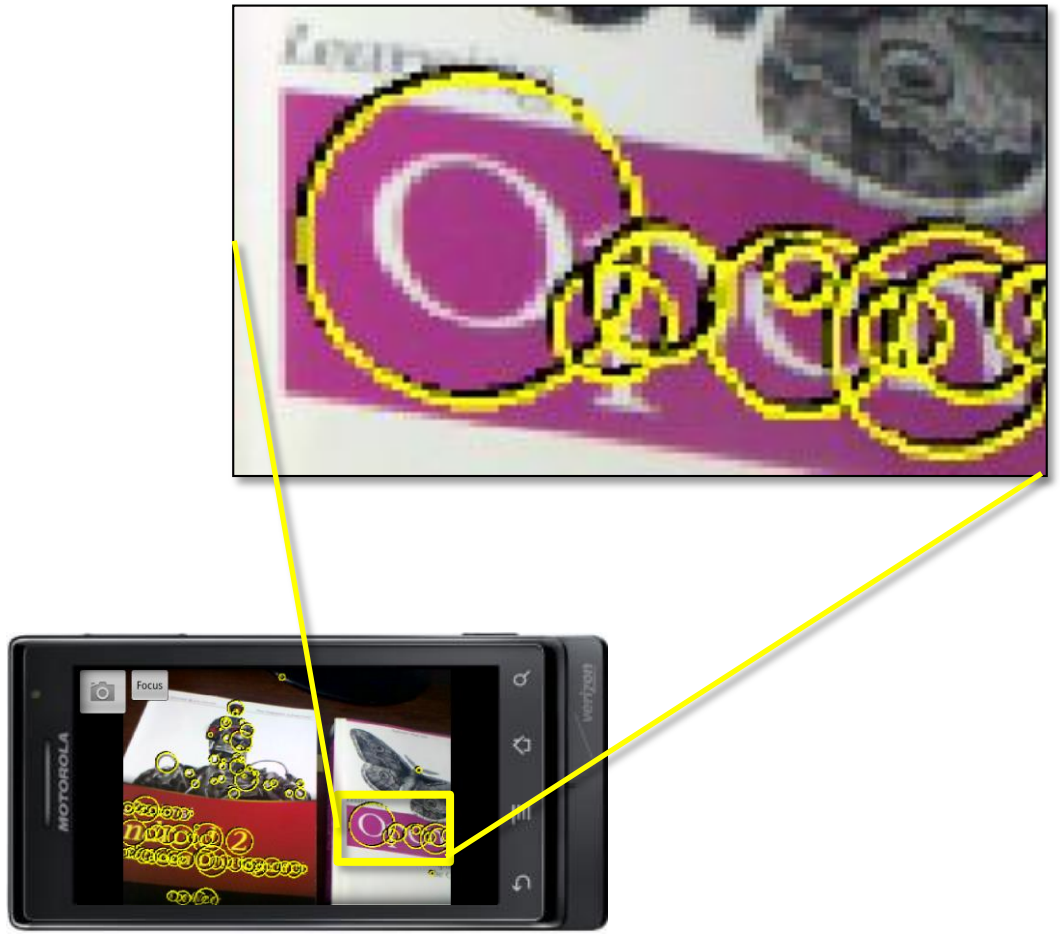
Initialize extractors in the class instantiation list

Function for detecting and drawing keypoints

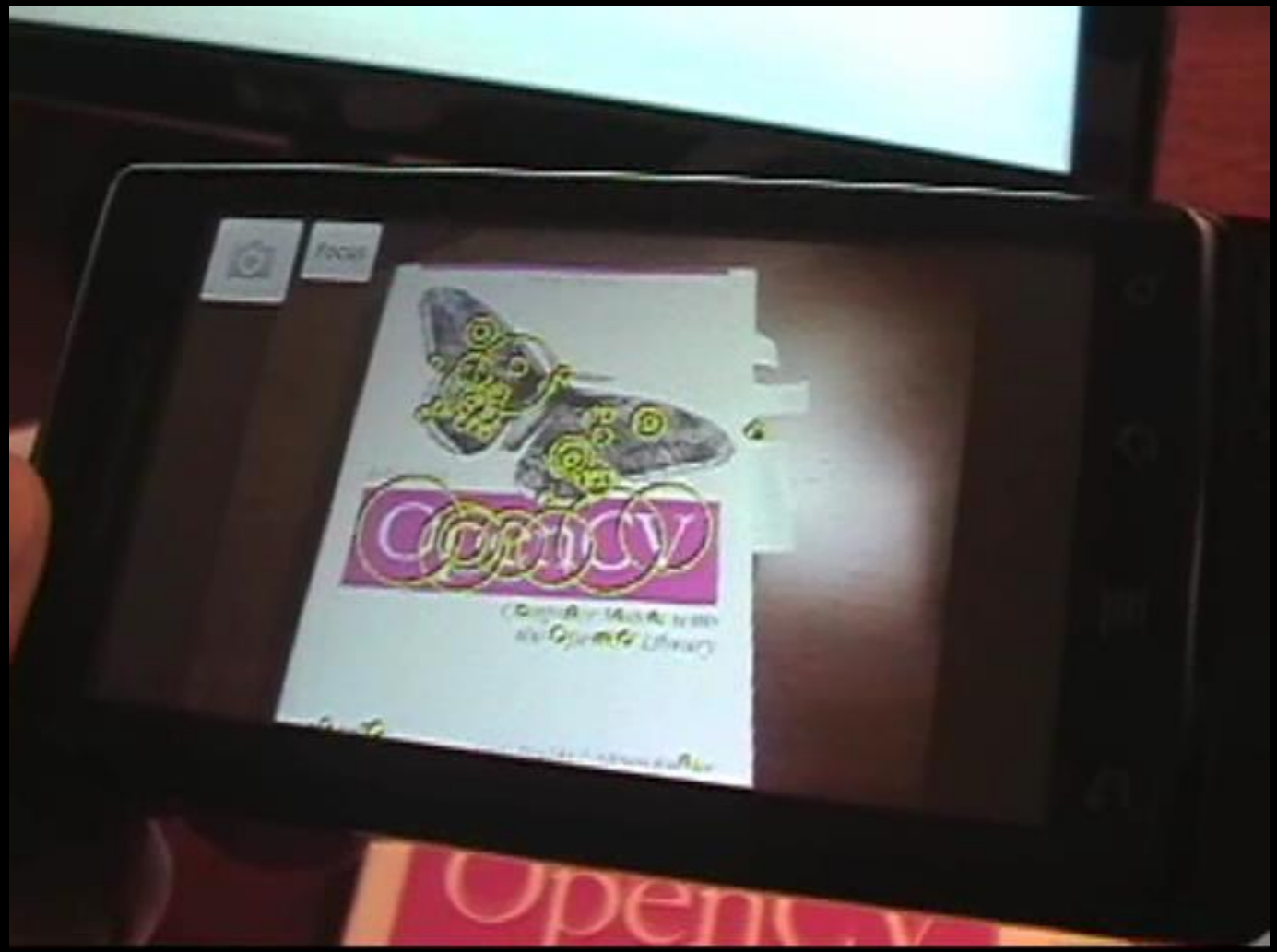
Processor class: detect and draw feature keypoints

```
// Detect feature keypoints
Mat grey_im = pool->getGrey(idx);
Mat color_im = pool->getImage(idx);
my_keypoints.clear();
my_mserd->detect(grey_im, my_keypoints);

// Draw feature keypoints
vector<KeyPoint>::const_iterator it;
for (it = my_keypoints.begin();
     it != my_keypoints.end(); ++it) {

  // Draw black circle
  Point2f pt = it->pt;
  circle(color_im, pt, it->size,
         cvScalar(0,0,0,0), 2);

  // Draw yellow circle
  pt.x += 1;  pt.y += 1;
  circle(color_im, pt, it->size,
         cvScalar(255,255,0,0), 2);
} // iterator
```

C++ Code



Local feature keypoints extraction



Detecting edges, lines, and circles

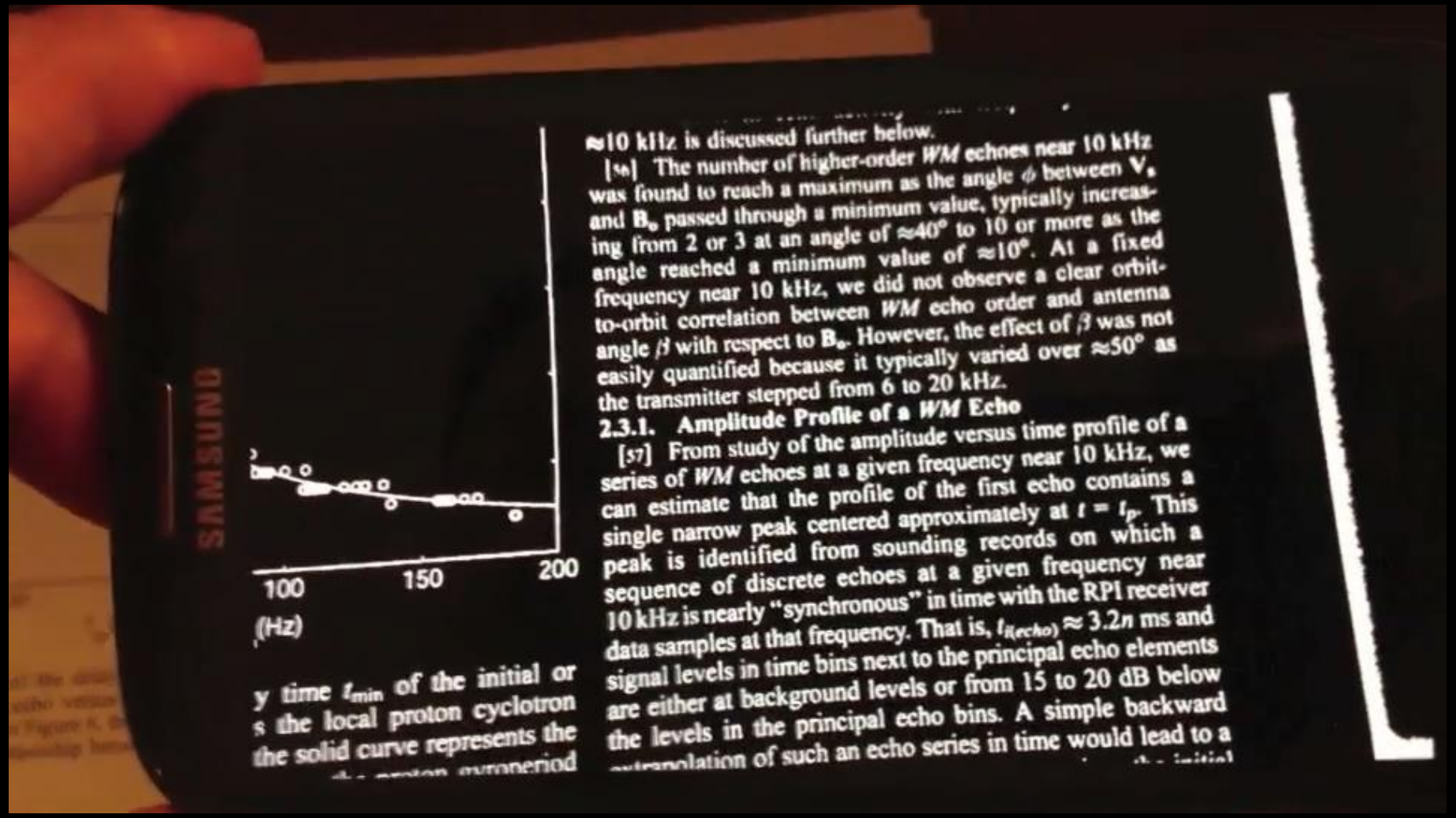
Detecting edges, lines, and circles

```
// Extract Canny edges
C++: Canny(
    InputArray image, OutputArray edges, double threshold1, double threshold2
);
```

```
// Extract lines using Hough transform
C++: HoughLines(
    InputArray image, OutputArray lines, double rho, double theta, int thresh
);
```

```
// Extract circles using Hough transform
C++: HoughCircles(
    InputArray image, OutputArray circles, int method, double accumRatio, double minDist
);
```

Locally adaptive binarization



≈10 kHz is discussed further below.

[56] The number of higher-order *WM* echoes near 10 kHz was found to reach a maximum as the angle ϕ between $\mathbf{V}_e$ and $\mathbf{B}_o$ passed through a minimum value, typically increasing from 2 or 3 at an angle of ≈40° to 10 or more as the angle reached a minimum value of ≈10°. At a fixed frequency near 10 kHz, we did not observe a clear orbit-to-orbit correlation between *WM* echo order and antenna angle β with respect to $\mathbf{B}_o$. However, the effect of β was not easily quantified because it typically varied over ≈50° as the transmitter stepped from 6 to 20 kHz.

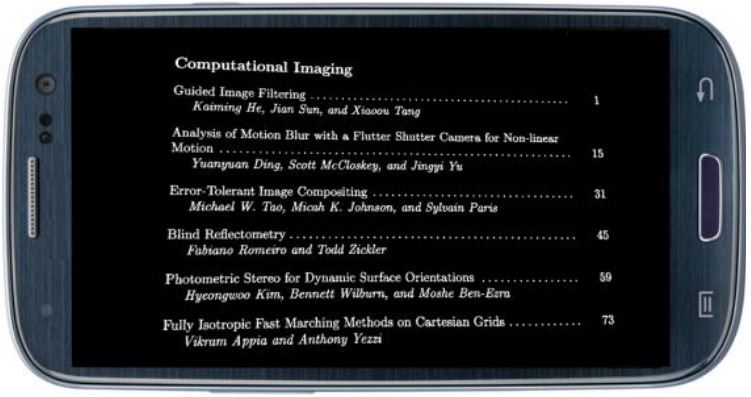
2.3.1. Amplitude Profile of a *WM* Echo

[57] From study of the amplitude versus time profile of a series of *WM* echoes at a given frequency near 10 kHz, we can estimate that the profile of the first echo contains a single narrow peak centered approximately at $t = t_p$. This peak is identified from sounding records on which a sequence of discrete echoes at a given frequency near 10 kHz is nearly "synchronous" in time with the RPI receiver data samples at that frequency. That is, $t_{i(echo)} \approx 3.2n$ ms and signal levels in time bins next to the principal echo elements are either at background levels or from 15 to 20 dB below the levels in the principal echo bins. A simple backward extrapolation of such an echo series in time would lead to a

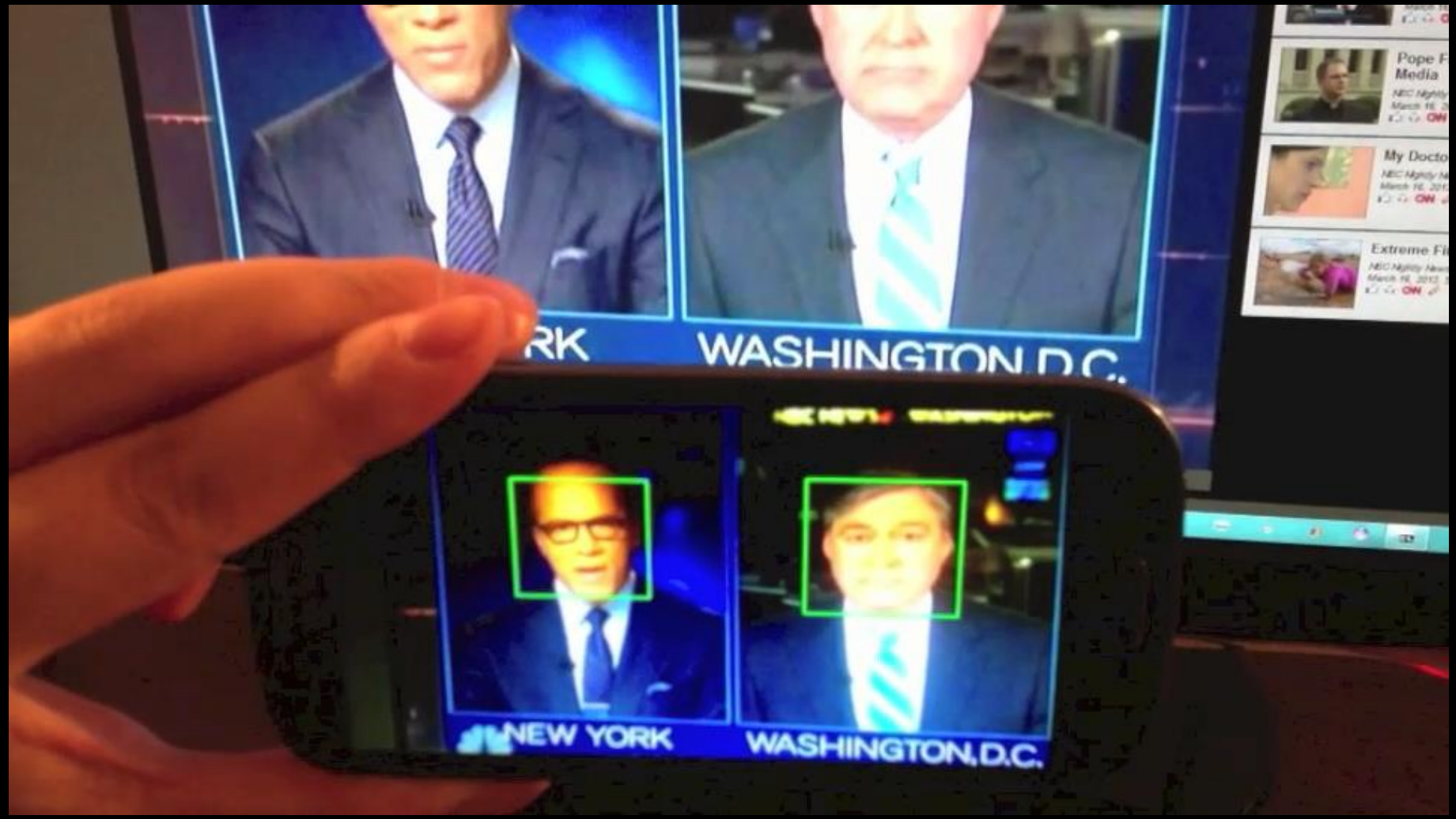
y time t_{min} of the initial or
s the local proton cyclotron
the solid curve represents the

Locally adaptive binarization

```
// Perform locally adaptive binarization
C++: adaptiveThreshold(
    InputArray src, OutputArray dst, double maxValue, int adaptiveMethod,
    int thresholdType, int blockSize, double valOffset
);
```



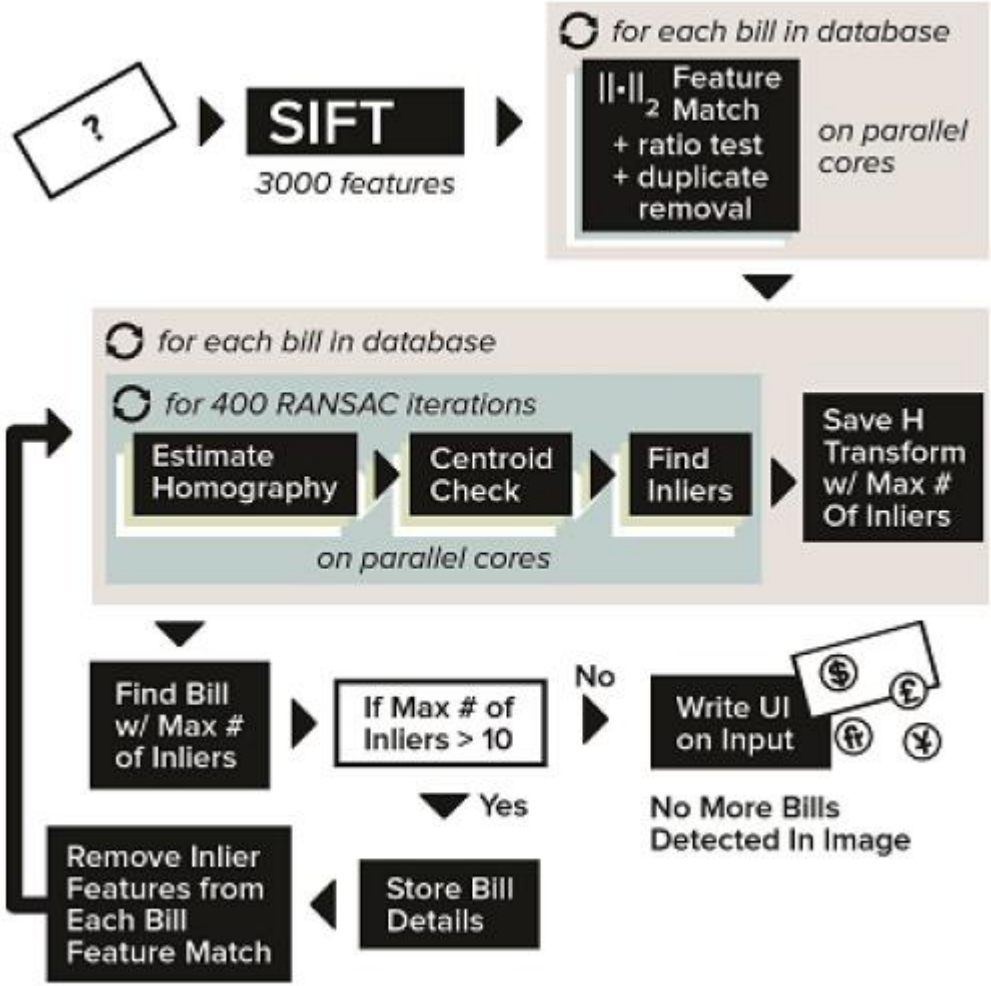
Human face detection



Human face detection

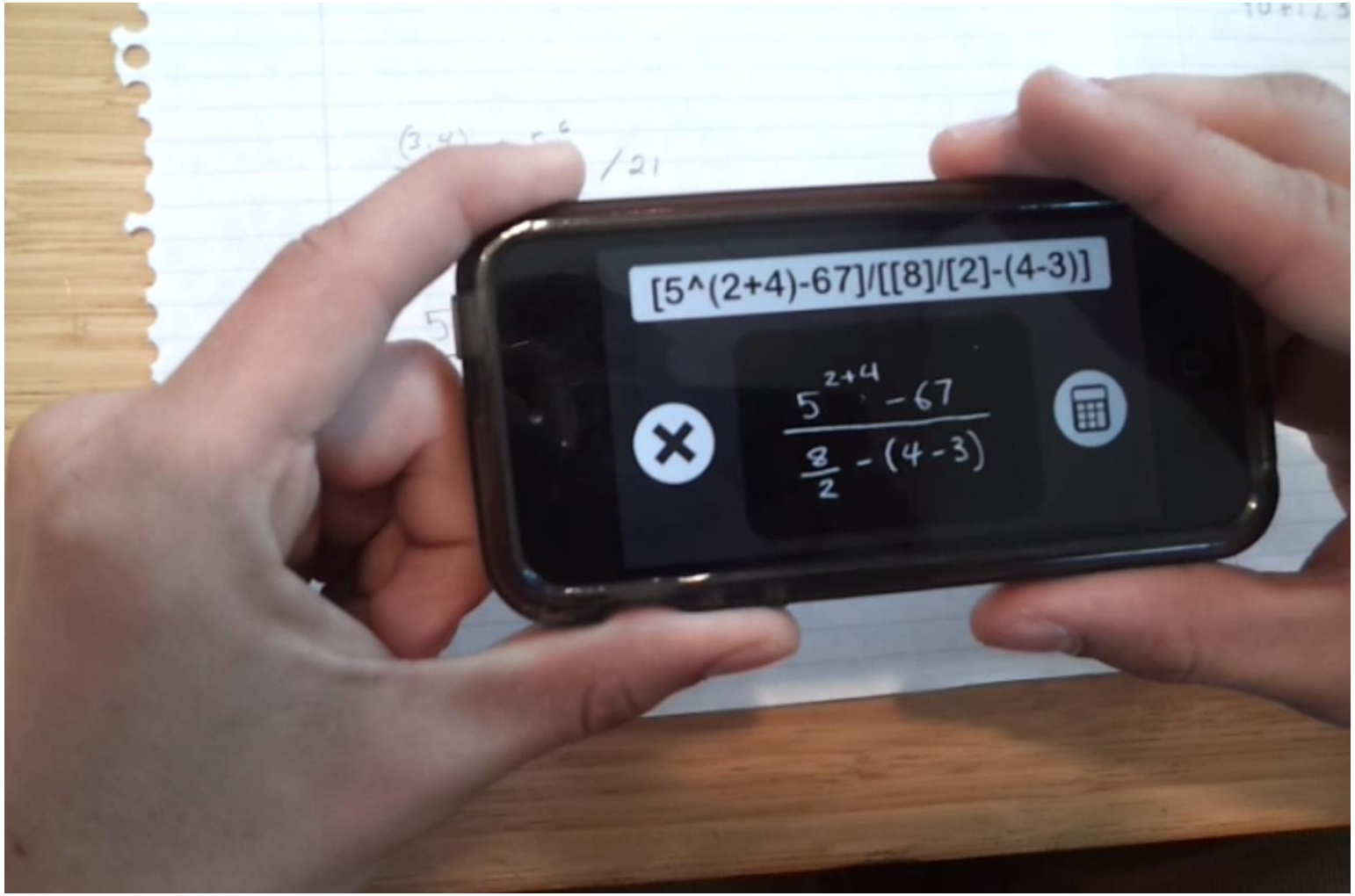
```
// Call Viola-Jones face detector
vector<Rect> RectFaces;
((DetectionBasedTracker*)this)->process(*((Mat*)imageGray));
((DetectionBasedTracker*)this)->getObjects(RectFaces);
for (int nFace = 0; nFace < RectFaces.size(); nFace++) {
       rectangle(imageColor, RectFaces[nFace], cvScalar(255,255,0,0));
} // nFace
```

Class project: Foreign bill detector and converter



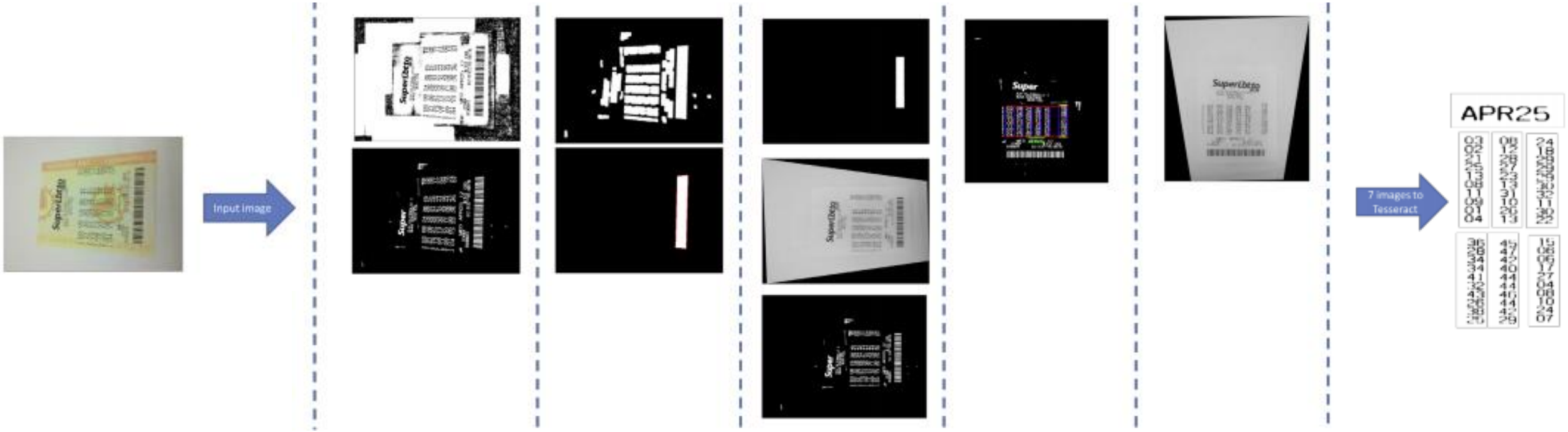
M. Digman and C. Elder, Spring 2013

Class project: Handwritten expression solver



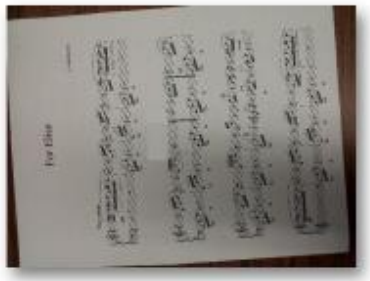
S. Harvey and W. Harvey, Spring 2013

Class project: Lottery ticket checker

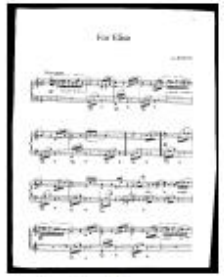


T. Zou and A. Gupta, Spring 2012

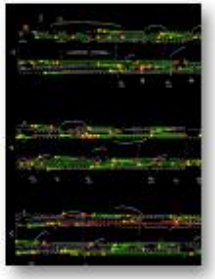
Class project: Optical music recognition and playback



Captured Image of music staff

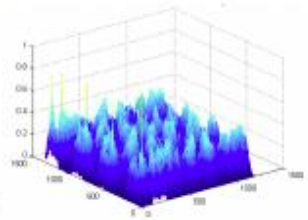


Upright Image after rotation



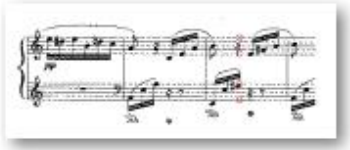
Horizontal Line Detection

Clef Template

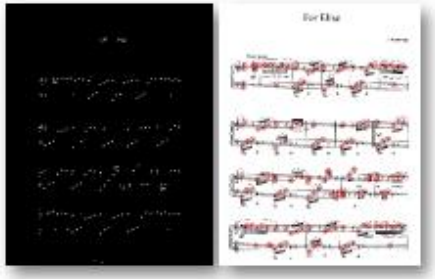


Clef Peak Detection

Matched Clefs



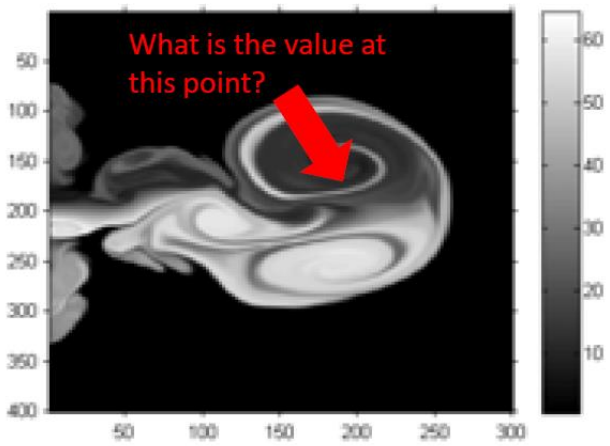
Top & Bottom Stave with red marks

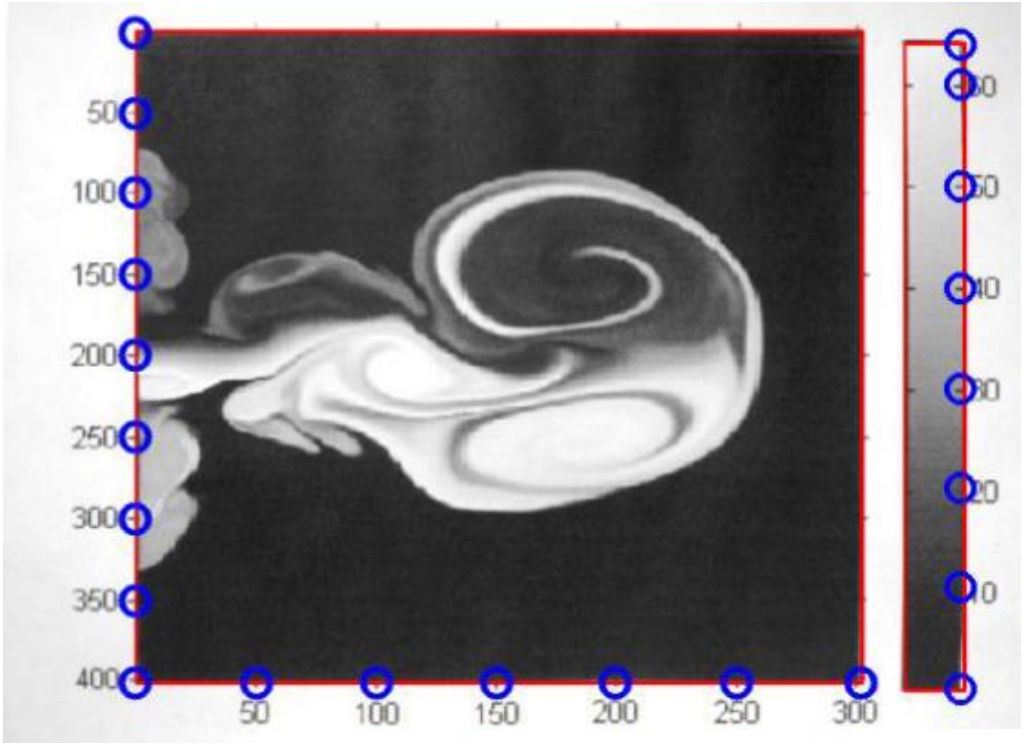


Detected Note Heads

S. Dai, C.-W. Lee, Y. Tian, Spring 2012

Class project: mobile graph reader





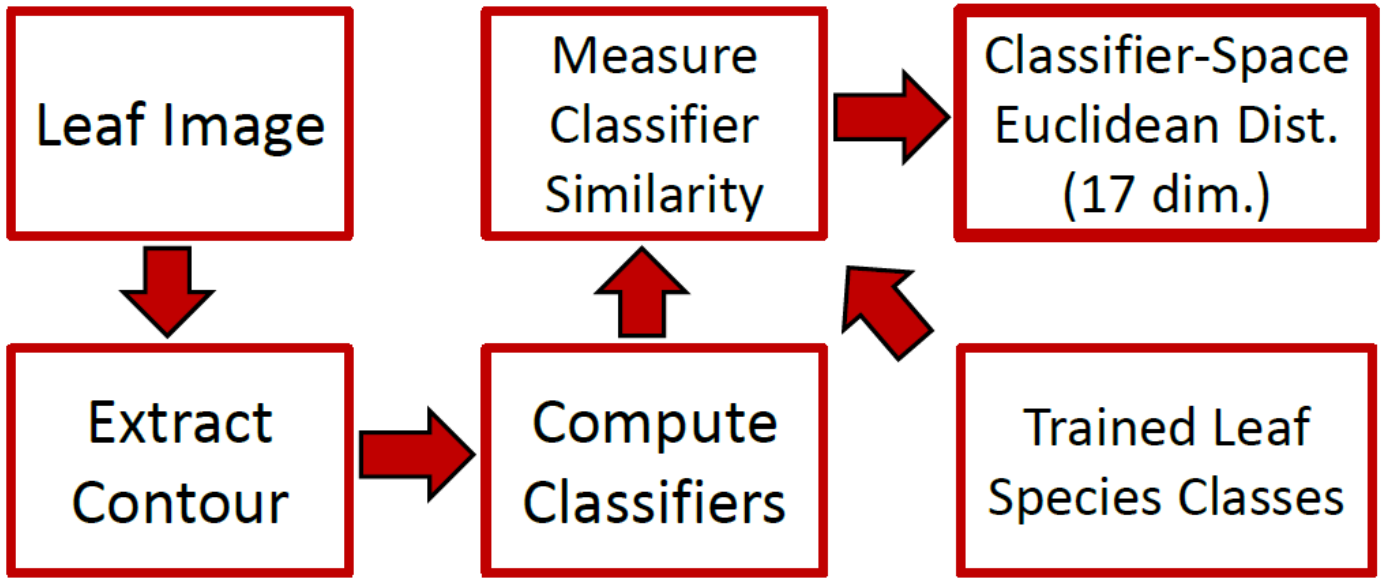
Segmentation of Image

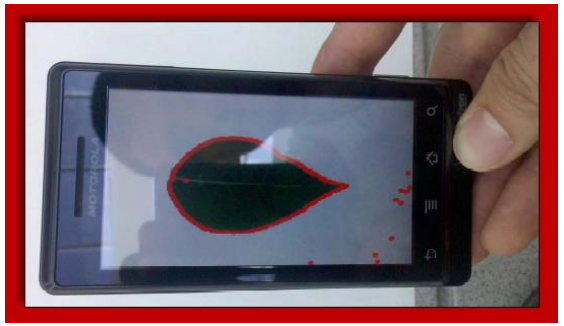
Detection of Tick Marks

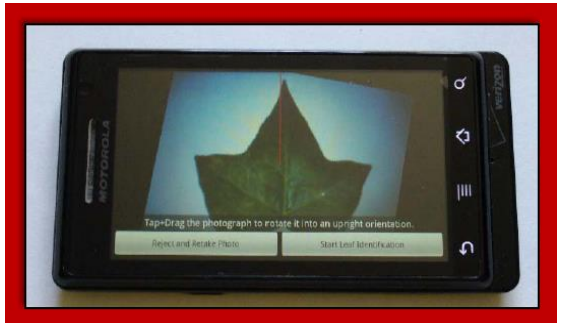
OCR of Text Labels

T. Jou, W. Ni, J. Su, Spring 2011

Class project: plant leaf classification







D. Knight, J. Painter, M. Potter, Spring 2010