COLOR BASED TRACKING

Karl Greuter, Mathias Mitterdorfer, Markus Leitner, Helmut Wolf

Table of Content

Table of Figures

1 History

Date	Who	Version	Description
18.03.2013	Helmut Wolf	V0.9	Initial draft, text
18.03.2013	Markus Leitner	V0.91	Proof reading, minor corrections
18.03.2013	Helmut Wolf	V0.92	Added source code section and Team member roles
07.04.2013	Helmut Wolf	V1.00	Minor improvements, added screenshots of app

2 Group Members

- Karl Greuter
- Mathias Mitterdorfer
- Markus Leitner
- Helmut Wolf

3 Basic Considerations

3.1 Frame Rate

Basic explorations have shown that working with the *whole frame* image on the pixel size, using *java*, is inefficient leading to a frame rate around 0.5 fps.

There are possible ways to circumvent this; possible options are:

- Reduce frame rate directly in hardware
  the full resolution is actually never needed, e.g. CameraBridgeViewBase provides a method setMaxFrameSize[i] that handles this.
- Downsample the frame in software
  openCV provides special functions which handles downsampling already, the predestined function for this is the static pyrDown-Method[ii] in Imgproc.
- Prevent iterating over all pixels of the frame in a java-for-loop and instead making use of the predefined openCV-framework functions

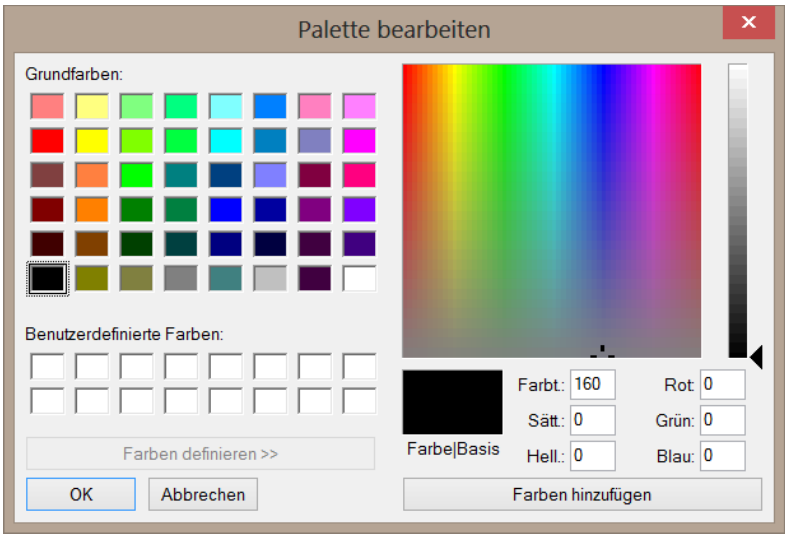
We have decided to use the third option; this decision still allows us to sample approx. 9.6 fps on a LG Nexus 4 device.

3.2 Color space

Though the RGB(A)-color space is natural for the human eye to see (and thus to locate) things on the picture, the HSV-color-space is much better suited to color-comparisons. As we do color-based-detection we do every step involved in the HSV-color-space.

3.2.1 The HSV Color Space

HSV stands for hue, saturation and value, respectively.



Farbt.: 160
Sätt.: 0
Hell.: 0
Rot: 0
Grün: 0
Blau: 0

Figure 1: HSV-color picker in MSPAINT

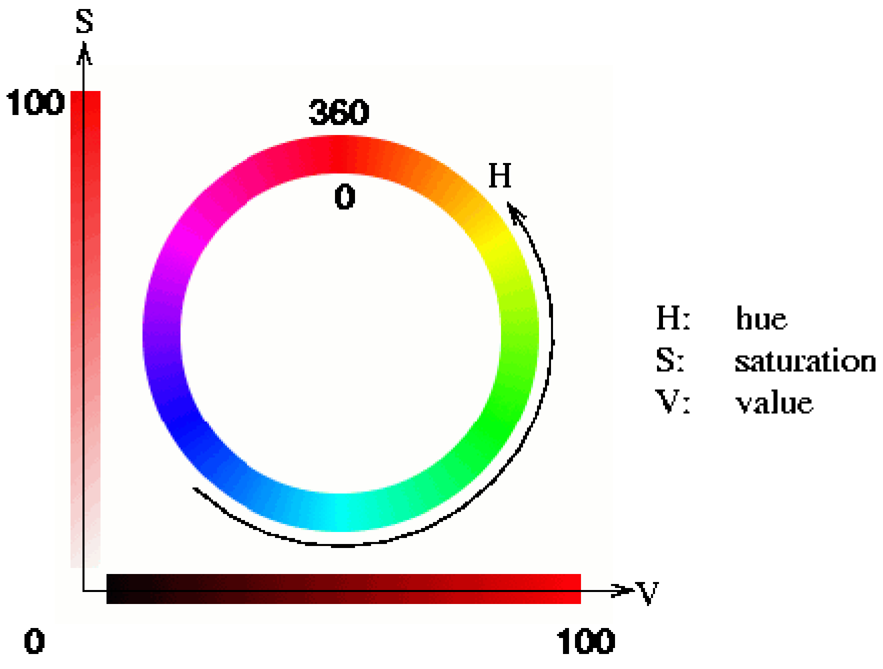


Figure 2: Illustration on the HSV-color space[1]

[1] As found on: http://ie.technion.ac.il/CC/Gimp/node51.html

4 Algorithm Description

4.1 Preconditions

4.1.1 Choosing color of interest

For being able to track an object the user first of all has to choose which kind of object (i.e. color) he is interested in. In our application we have decided to track the middle of the center (as indicated by the rectangle). If the user clicks at the frame this color is tracked and used for further comparison.

4.1.1.1.1 Digging Deeper

The 'color of interest' is detected using the arithmetic average of a square of 32x32 pixels (default). Taken a submat of the current frame when the user first clicked the screen, the pixels are summed up using Core.sumElems[iii] and each scalar is the divided by the amount of pixels used.

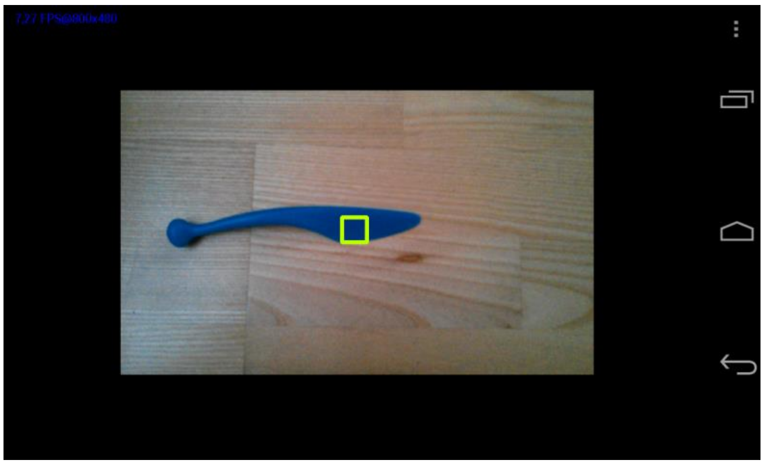


Figure 3: Choosing color of interest

4.2 Basic Steps Involved

In general for being able to track a BLOB[2] of color, there are three cases necessary.

4.2.1 Converting to HSV-Color Space

As described above (see 3.2.1) the HSV-color space is better suited for color comparison, thus first of all the current frame – which by the matter is not down sampled at all – is converted into HSV.

4.2.2 Calculate Distance In Color Space And Threshold Image

As we are in a well-suited color-space we are now able to check the whole frame on how far away a pixel is from our whished color. For accomplishing this, we first of all we need to have the user a color chosen (see 4.1.1). Given this color we take a (empiric) algorithm, which transforms the user-chosen-color into a range (actually subtracting a HSV-color-value to gain the lower bound for comparison and vice-versa for the upper bound; these values are described in the variable 'hsvColorRadius'):

$$hsvColorRadius \coloneqq \begin{matrix} 25 & [hue] \\ 25 & [saturation] \\ 25 & [value] \end{matrix}$$

Though comparison just in the hue-space should be enough, we also do a little bit of comparison in the saturation- and value-space.

[2] BLOB… binary large object

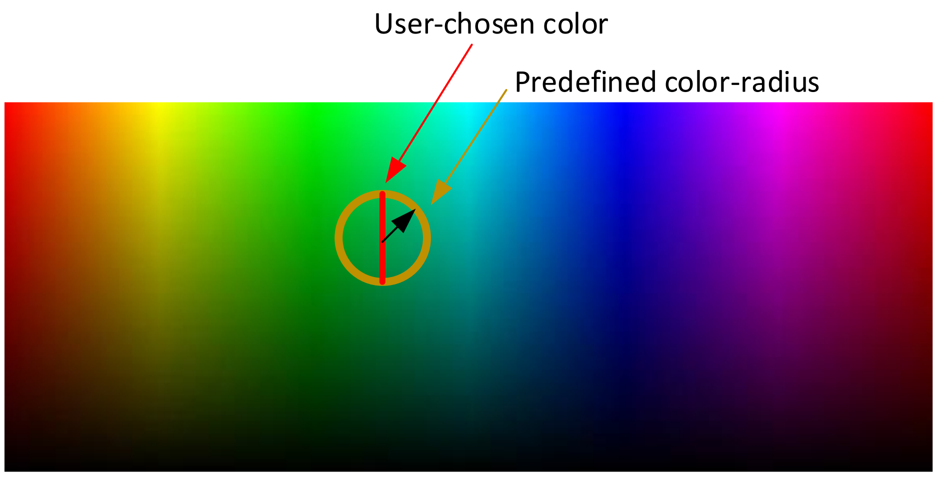


Figure 4: Illustration on the color radius used

In openCV the function inRange of the module Core[iv] realizes all the above functionality. Furthermore this function also does thresholding: every pixel which lies in the supplied range is given a value of 255, all the others 0, thus resulting in a grayscale image where just the ROI[3] gets displayed.

4.2.3 Contour Tracking

As our current mask just contains the ROI, we now can track the basic contours of our blob. Of course openCV also supplies a function for this case, namely the findContours-method in the ImgProc-class[v]. This class returns a List of Matrixes (MatOfPoints) indicating the contours of all the BLOBs given. From this list the MatOfPoints with the biggest area is extracted as this is most probably the object we want to track.

4.2.4 Computing The Center Of Mass

As suggested on http://stackoverflow.com/questions/10942742/opencv-c-usage-of-cvmoments one could use moments[vi], however we have used a simpler method, namely the bounding-rectangle (aka bounding box) –approach.

[3] ROI… region of interest

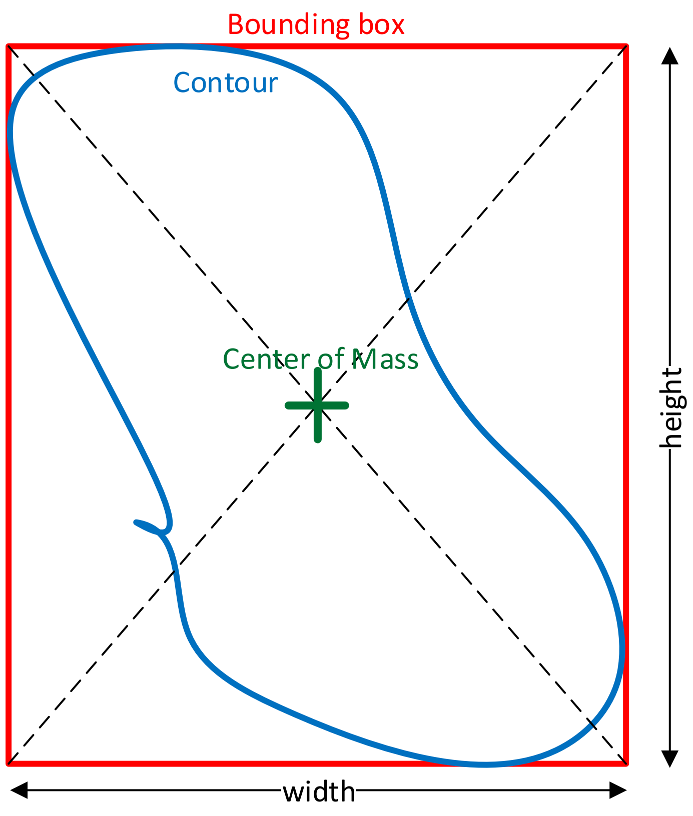


Figure 5: Bounding box approach for Center of Mass-detection

5 Source Code

Beside the basic xml-File used as a layout in a traditional Android-apk, there are 3 classes of interest.

5.1 ColorBasedTrackingActivity

The main activity used for creating the look&fell of the app. The menu items are created in here and based on the choice the user makes, the respective Picture is shown on the screen.

5.2 TrackerHelper

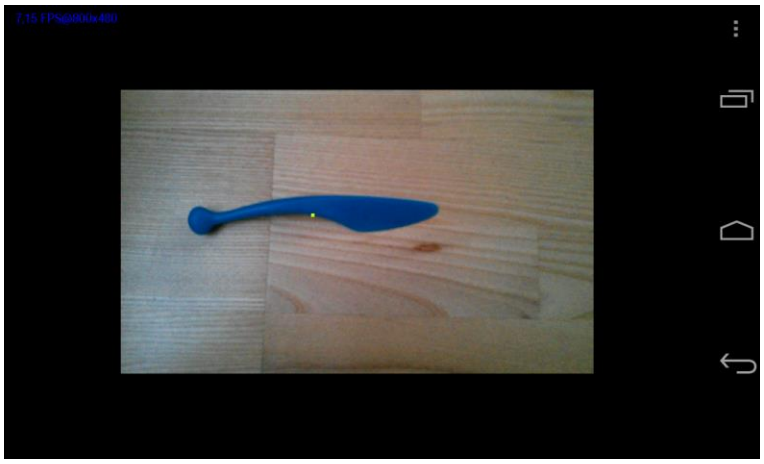
This class is just used to calculate the average color of the above described 32x32 square.

5.3 ColorBasedTracker

This is the class where the magic happens, it does all the processing from the incoming hsv-image to the resulting center-of-mass (and all immediate steps included)

6 Screenshots

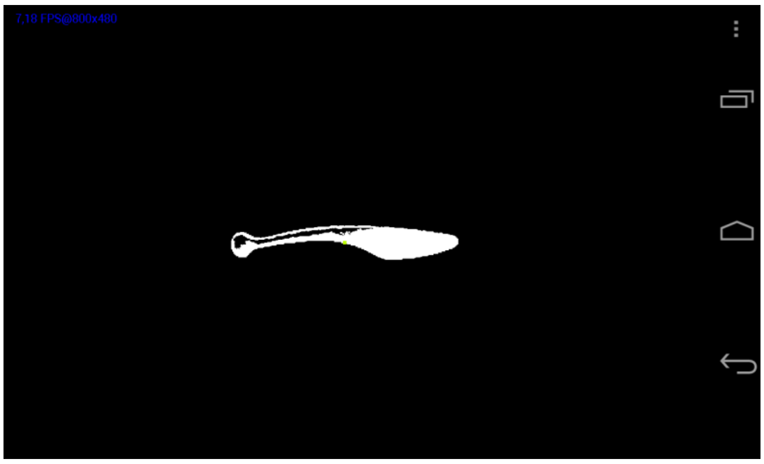
6.1 CenterOfMassRgbMode



```
7,15 FPS@800x480
```

Figure 6: CenterOfMassRgbMode

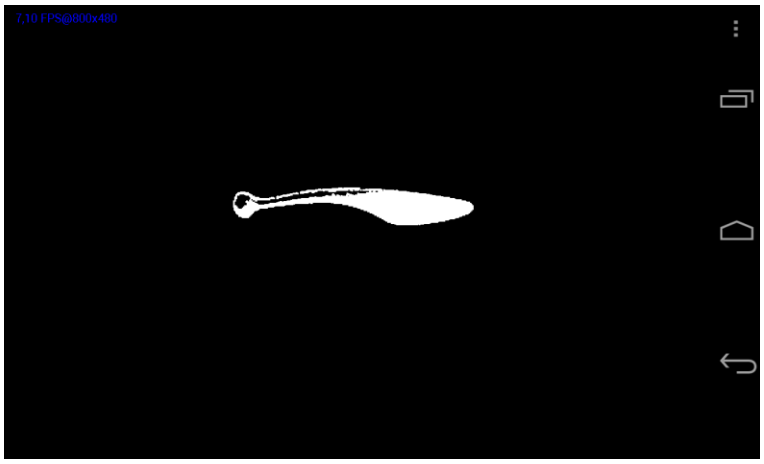
6.2 BlackWhiteMode



```
7,18 FPS@800x480
```

Figure 7: BlackWhiteMode

6.3 DilateMode



```
7,10 FPS@800x480
```

Figure 8: DilateMode

6.4 BoundingRectangleMode

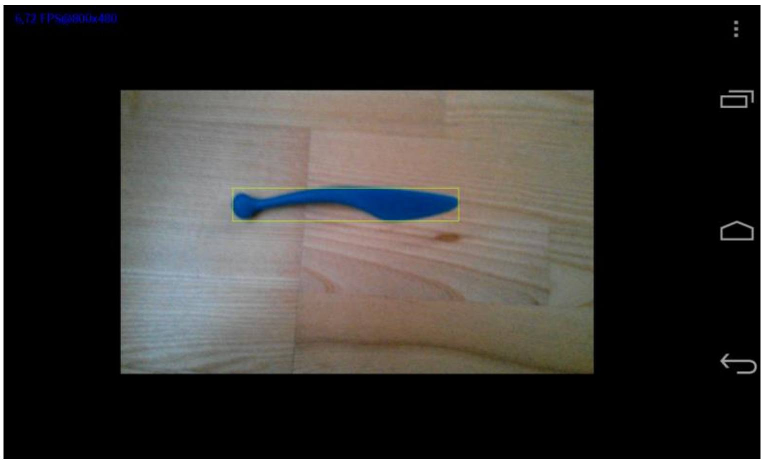


Figure 9: BoundingRectangleMode

6.5 ContoursMode

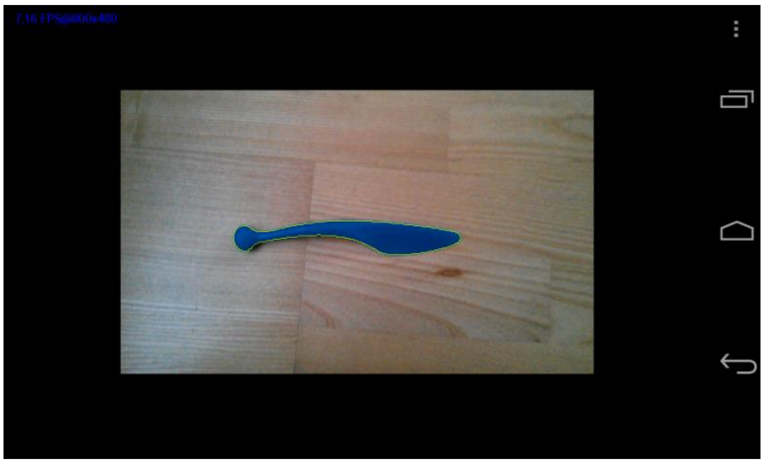


Figure 10: ContoursMode

6.6 CenterOfMassModeBlackWhiteMode

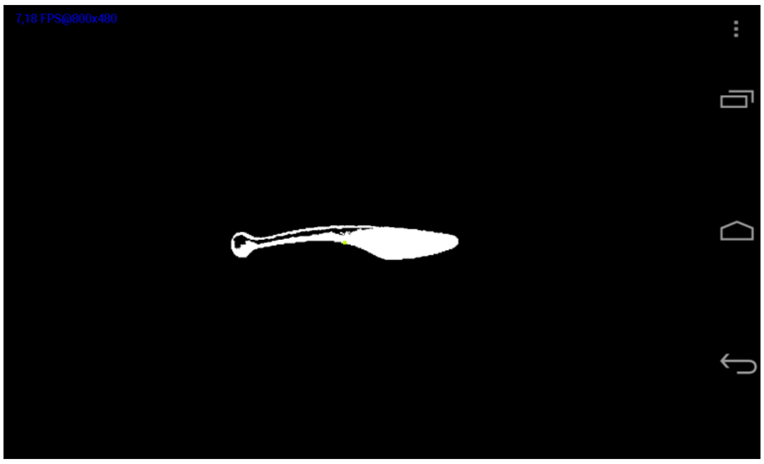


Figure 11: CenterOfMassModeBlackWhiteMode

6.7 TrackingPathMode

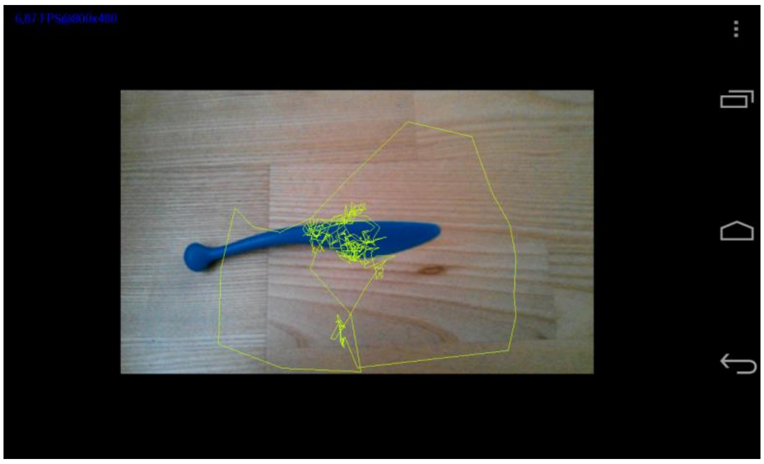


Figure 12: TrackingPathMode

7 Team Member Roles

The basic workload has been splitted equally to group members; the main points on this assignment definitely were the completely new framework OpenCV, for some of us the android application development as well as this documentation itself.

What?	Who?	Time taken?
Getting to know OpenCV	Everybody	4*1-2h
Getting to know Android-apk-Development	Markus Leitner, Karl Greuter, Mathias Mitterdorfer	3*2h
First steps: frame-rate problems, reading pixel per pixel etc.	Helmut Wolf	3h
Main application development, debugging – 1st version	Mathias Mitterdorfer	13h
Main application development, debugging – 2nd version – deprecated	Helmut Wolf	8h
Code reviewer, quality assurance, Refactoring	Helmut Wolf	5h
Documentation	Helmut Wolf	1*5h+1*4h
Documentation review	Markus Leitner	1h
Total		***53h***

8 References

[i] setMaxFrameSize-documentation:
http://docs.opencv.org/java/org/opencv/android/CameraBridgeViewBase.html#setMaxFrameSize(int, int)
[ii] Imgproc.pyrDown-documentation:
http://docs.opencv.org/modules/imgproc/doc/filtering.html?highlight=pyrdown#pyrdown
[iii] Core.sumElems-documentation:
http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#sum
[iv] Core.inRange-documentation:
http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#inrange
[v] Imgproc.findContours-documentation:
http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours
[vi] Imgproc.moments-documentation:
http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#moments