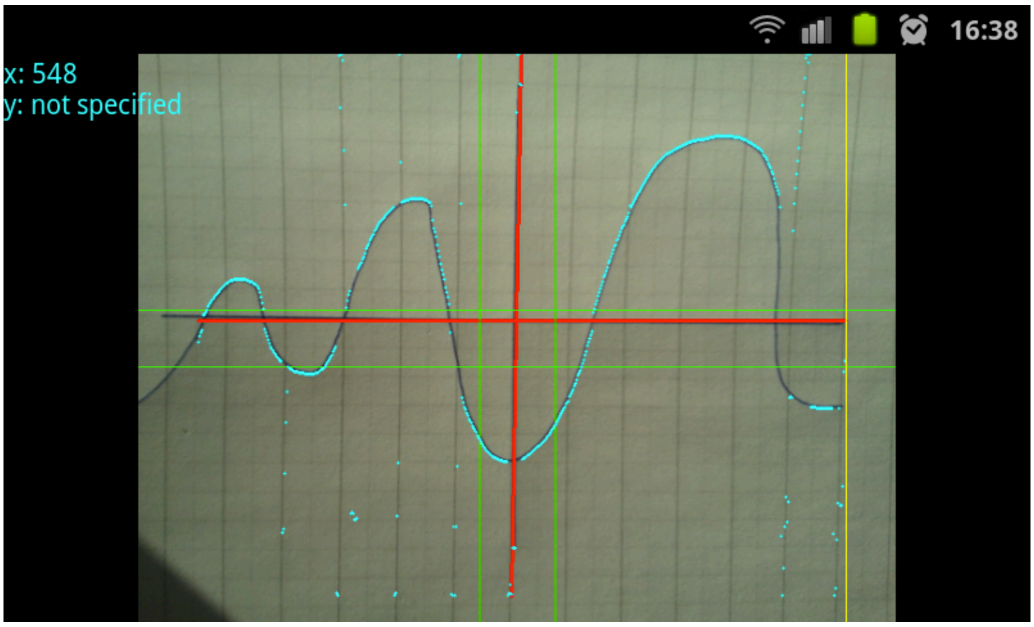
Mid Sweden University

The Department of Information Technology and Media (ITM)

Authors:	Florian Rüter, Timo Schröder
E-mail address:	flru1100@student.miun.se, tisc1101@student.miun.se
Study programme:	Applied Computer Engineering, 7,5hp
Examiner:	Ulf Jennehag, Ulf.Jennehag@miun.se, Mikael Hasselmalm, Mikael.Hasselmalm@miun.se
Scope:	3761 words
Date:	2012-03-18



Project report
within Applied Computer Engineering

Ungraph

Application to determine data in coordinate crosses

Florian Rüter
Timo Schröder

Abstract

The "ungraph" software is an android app for scanning and digitalization of a graph in a printed coordinate system. Using the in-build phone camera or a pre-made image it searches the graph and allows to browse the values by touching the wanted position on the screen. To experience a new development paradigm we decided to use test driven development, which requires the implementation of a test before every module of written code to "drive" the programmer to good -quality-software. This technique was skipped after 1/3 of the development. The discussion of the reasons is part of this project. The app makes heavy us of computer vision, namely the OpenCV library, to analyze the image. One important method used is the Hough transformation to find the x- and y-axis that is described in detail in the report. Finally it is shown that there is huge space for further developments.

Keywords: Android, Test Driven Development, Git, OpenCV

Table of Contents

Terminology

Acronyms

TDD	Test Driven Development
CV	Computer Vision
SVN	Subversion
JNI	Java Native Interface

1. Introduction

1.1. Background and problem motivation

Since the behavior of many technical components is described graphically (i.e. in coordinate-crosses), it is difficult to get detailed values from these documentations. Therefore it would be nice to have a chance to quickly determine detailed values from coordinate-crosses.

1.2. Overall aim

The projects overall aim is to gain knowledge about Android programming, to discover Test Driven Development and Git as a distributed version control system.

1.3. Concrete and verifiable goals

Resulting from the former problem statement and problem motivation the verifiable goal defines itself as follows. The result of this project should be an application, able to run on Android devices, to process a picture of a coordinate cross (with graph) as input and to show exact values relating to the graph.

1.4. Outline

Chapter 2 describes the background behind the image processing that is used in this project. Chapter 3 shows the methods used for the development and Chapter 4 gives a brief description of the projects implementation. Finally Chapter 5 presents the result and Chapter 6 finishes this work with a review and conclusion over the project.

1.5. Work-sharing

The project was split into different blocks where each programmer did certain parts. We met every day to have immediate feedback and idea exchange. Independent of that Timo concentrated on the image processing functionality while Florian focused on the android application layout and framework and the design / interaction of the different activities.

2. Theory

2.1. Image Processing

A bitmap image just consists of a matrix of pixels, which is defined by width, height and color information. To extract information from this matrix, image processing is needed. Possible information is the position or orientation of certain objects in the picture like lines, points or even more complex objects.

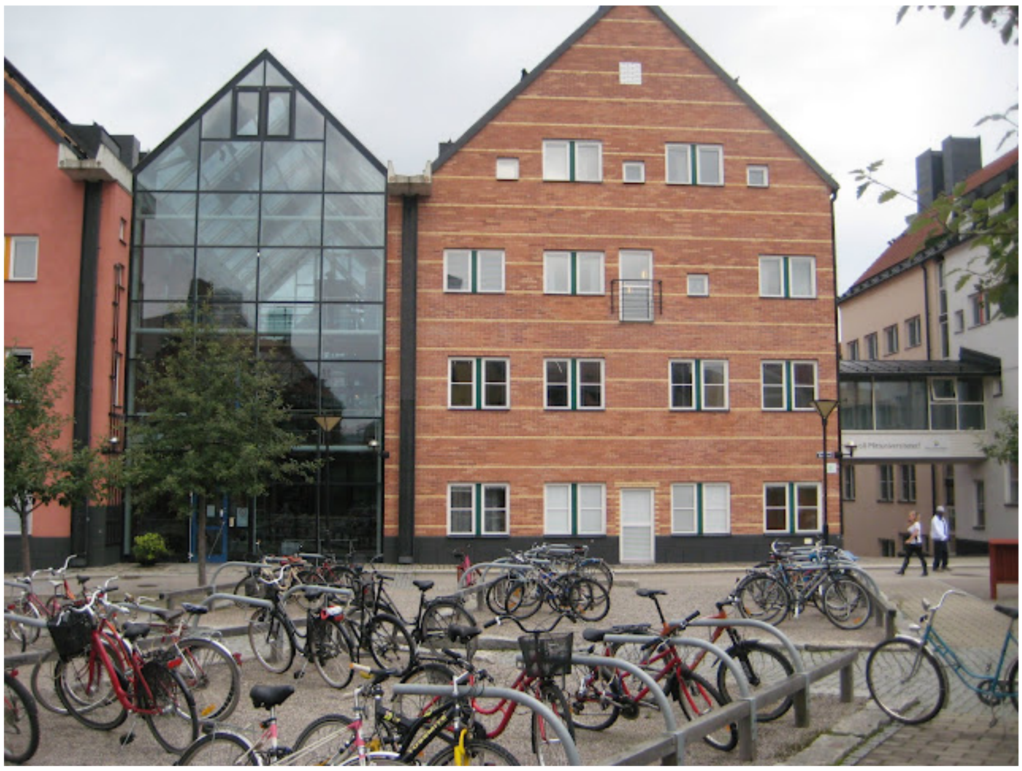


Figure 1: Original picture used for further explanation
source: own picture

2.1.1. Canny edge detector

The canny filter is a filter that can be used to ease the task of line-detection. It transforms a bitmap image to another bitmap image of the same size containing the edges of the source bitmap. The destination bitmap is a black and white image where the white pixels display a detected edge. The following line detection than only has to involve the detected edges and can neglect the black parts of the image. This could drastically increase the efficiency of the algorithm.

The canny edge detector consists of 3 steps that are processed in continuous order. The colours of an image do not add a value to the edge detection quality. For that reason the image is converted to a grayscaled image before processing. This method also speeds up the calculation because only one channel has to be processed.

2.1.2. Noise reduction

To remove noise particles, a Gaussian filter is used to smooth the image. To do that, each pixel is replaced by a combination of the pixel in its neighbourhood including itself. For the canny edge detector the following matrix, as shown in Figure 2, is used to calculate the new pixel:

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

Figure 2: Pixel Pixel B is a composition of the 5x5 pixel matrix from the source image pixels [1]

The following Figure 3 illustrates the image after Gaussian filtering.

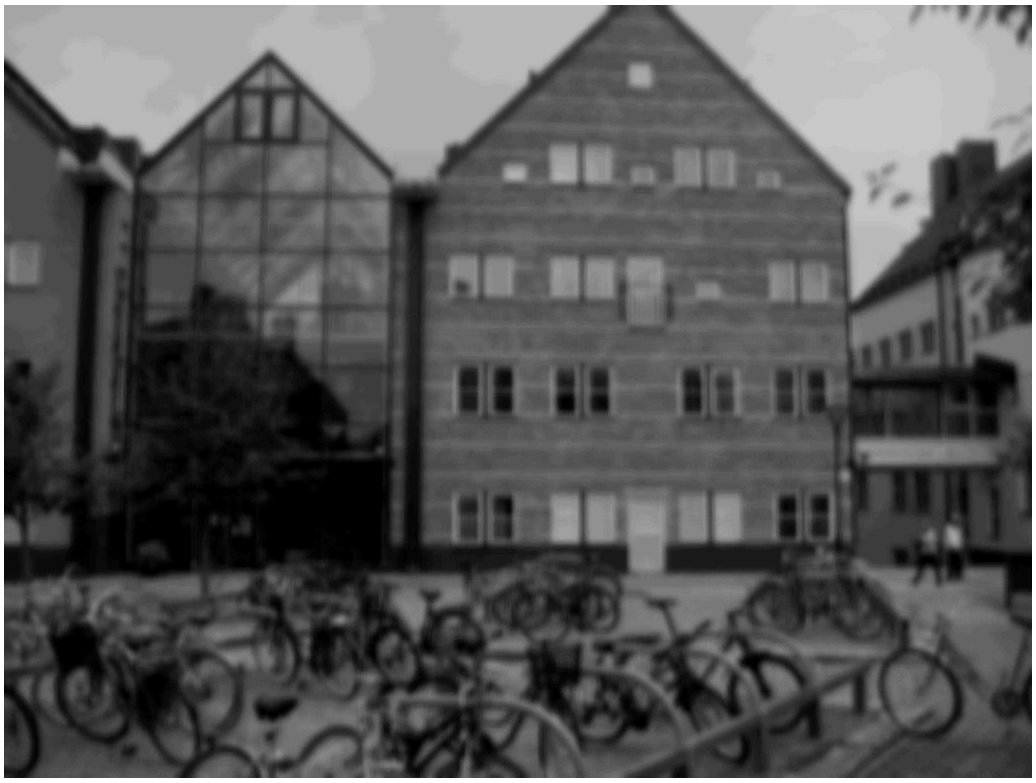


Figure 3: Grayscaled picture after Gaussian filtering
source: own picture

2.1.3. Gradient intensity

The blurred image is than transformed to a gradient image using filters like the Roberts, Prewitt or Sobel [1].

2.1.4. Non-maxima suppression

The last step is to remove all gradients that are below a certain threshold. The threshold declares the intensity that is needed for an edge to make it through the detection process. This step leads to the final black and white image that contains the detected edges as shown in Figure 4.

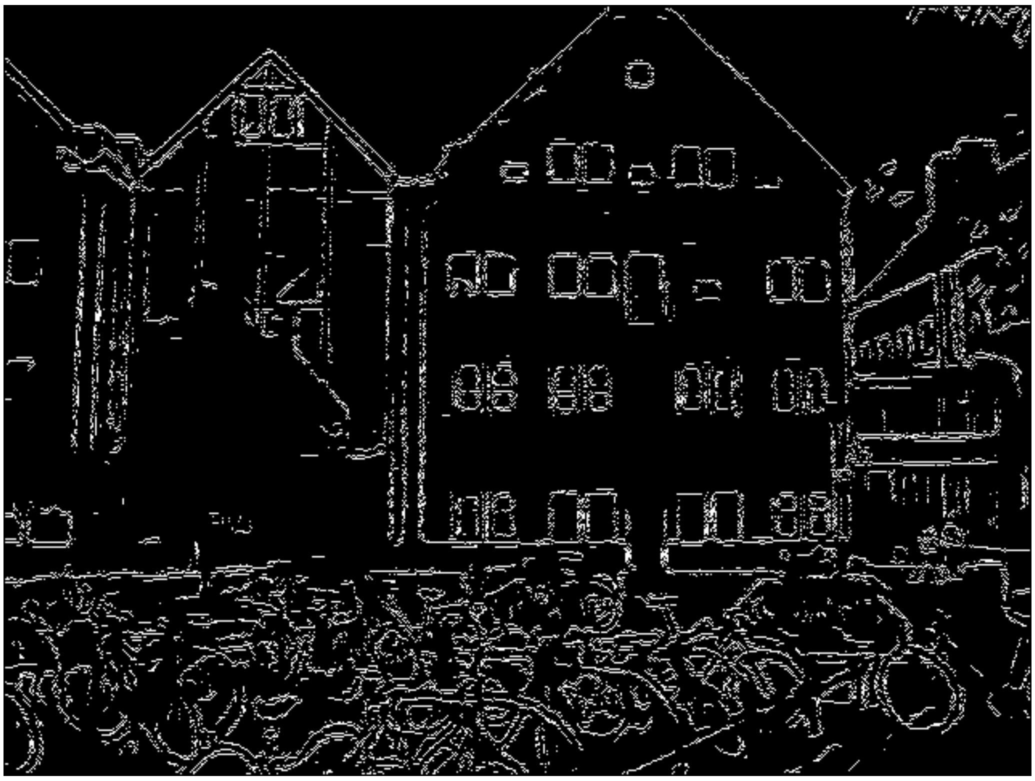


Figure 4: Black/White picture after non-maxima suppression
source: own picture

2.2. Hough Transformation

The Hough transformation is an algorithm that transforms a bitmap image to a parameter matrix that allows for easy detection of lines. The matrix has the same number of dimensions and size of the original image. The first dimension of the parameter matrix represents the distance of the line from the origin of the bitmap (usually the top-left or bottom-left corner of the bitmap). The second dimension represents the rotation of the line. Every pixel on a line in the original image is transformed to the same values in the parameter-matrix.

The transformation works as follows: The value of each pixel on the image is added to every possible line-parameter it could be on in the parameter-matrix. The more pixels lie on a line in the image the more value is added the representing parameter in the Hough-matrix. That means that a strong line will add high value to the corresponding point in the parameter-matrix. Using a threshold, lines which a certain strength can be extracted easily as they are simple points in the transformed matrix. Since the fact that a line is transformed into rotation and distance from origin parameters the position of the begin and end of the line are discarded during transformation. The following Figure 5 shows a bitmap picture containing 2 lines and the corresponding Hough-transformed parameter-matrix.

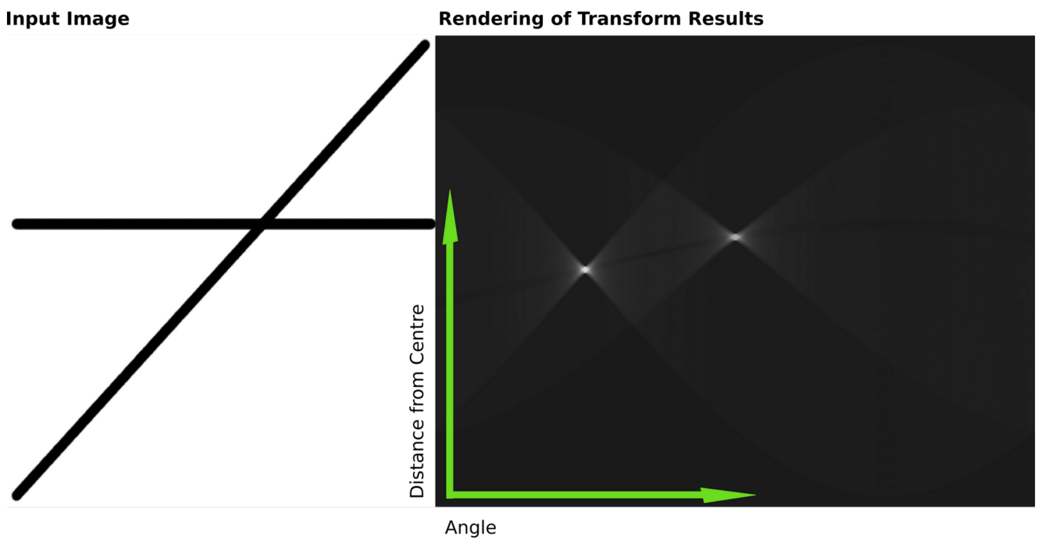


Figure 5: Bitmap image (left) and Hough-parameters (right) [2]

3. Model

3.1. Development models/techniques:

This project will be developed by the use of Test Driven Development (TDD). Since it is a two persons project a method to handle "working together" is necessary. In general sharing the code over a Git repository will do this. Some difficult situations might need more concentration than from one person. These are going to be solved by pair programming. The following sections explain each of this models/techniques in more detail.

3.1.1. Test driven development:

In contrast to many other development models, a testing of the source code is guaranteed with Test Driven Development. This is caused by the fact, that automated tests (most of the time unit-tests, but TDD is also possible with complete program-test) exist before any code is written [3]. Figure 6 shows TDD in comparison to the very common waterfall model. The problem with the waterfall model is, that there is a version of the software that looks and behaves like a ready-to-use software that has bugs that have not been detected because of the lack of testing. That causes quiet often that the testing step gets dropped for different reasons, e.g. that the developers boss (who quiet often is no programmer) sees the ready, untested version and thinks that all work is done while in reality the testing might need the same amount of time as the actual implementation. This cannot happen with the TDD model because the tests are written before a "ready-looking" version of the software is available and nobody can get the idea that the program might be ready to deliver.

On the other hand later changes of the software could implement malfunctions that or not obvious in the first place and require further bug fixing after new software has been delivered. With TDD all tests just need to be re-run to test that all functions of the software are still in order. This is especially useful when a third party is changing the software that is not aware of all details of the software. Obviously the quality of the tests is in direct relationship with the capability to detect software bugs.

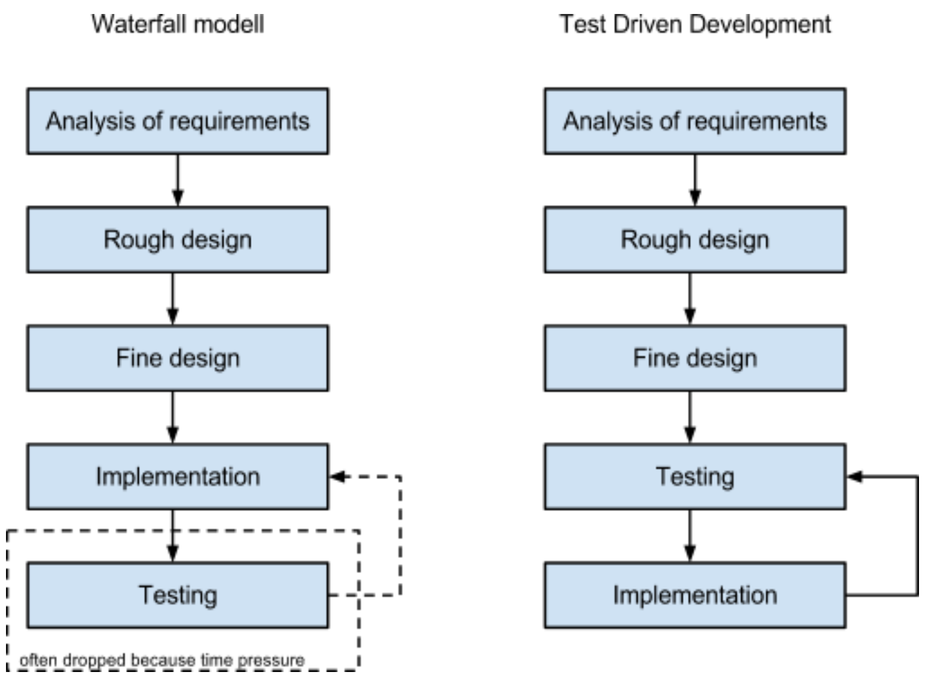


Figure 6: Comparison waterfall model to Test Driven Development
Source: self-drawn

Figure 7 explains the process of TDD with unit testing more in detail. Generally it can be seen as the two last steps from the TDD model from Figure 6. The process starts with writing a test for a new unit and afterwards run this test. Surely in the first iteration of this process the test will fail. Then it is time to write some code. After the implementation it is time to run the test again, but this time together with all tests, which might be already available, to ensure, that already available program parts are not influenced in a negative way. If the test fails, a revision of the code is necessary. Otherwise it is the next step to clean up, refactor, the code. After that it is possible to start the implementation of the next software unit.

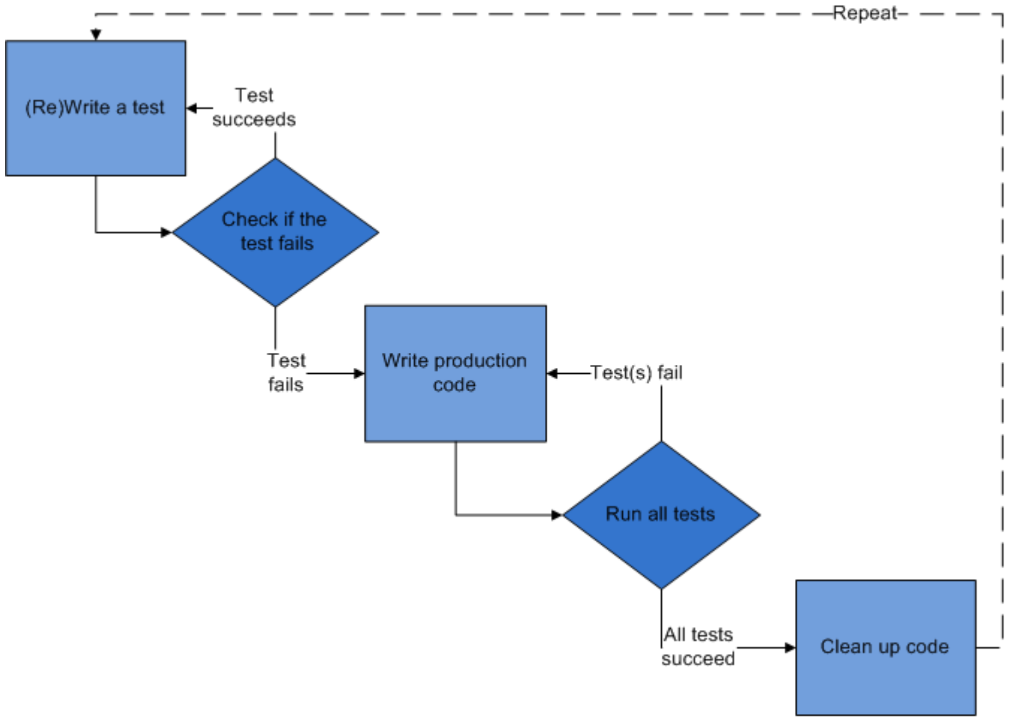


Figure 7: Test Driven Development in detail [3]

3.1.2. Git

Regarding to Wikipedia [4] Git is a distributed revision control system originally designed by Linus Torvalds for the Linux kernel development. In comparison to subversion (SVN), Git is a distributed system, what means that each developer stores a local copy of the complete shared code on his own machine. That ensures that there is no central point of failure (like a server distributing the shared sources). Since Git supports the possibility of creating/merging branches it is possible to work on two solution approaches for one problem at the same time without getting in conflict. Figure 8 explains the idea behind branching and merging with the help of an easy example. This example shows how a project including two different parts (tasks) could be solved with a team of three developers. While *developer A* and *B* focus on solving the first task with different approaches, *developer C* can already start to develop *task 2* on his branch. After a while it comes clear that the approach from *developer B* is better than the one from *developer A*, then it is possible to simply drop branch of *developer A*. When *B's* solution is finished his branch can be merged to the master branch. When C is done with his/her work it will also be merged into the master branch. The master branch contains then a complete copy of the project with the correct implementation of both *task1* and *task2*.

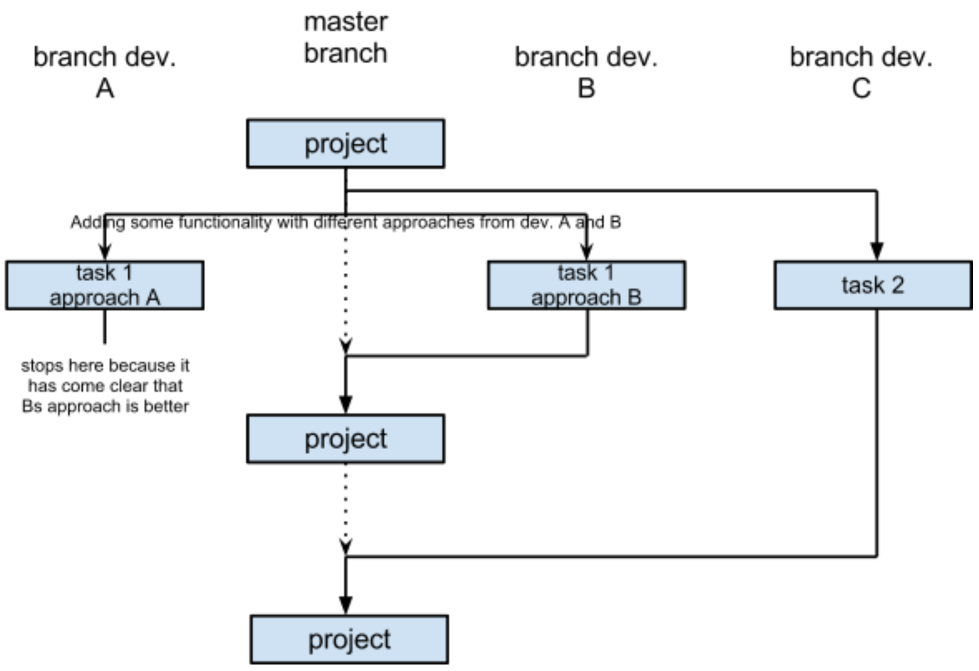

branch dev. A
master branch
branch dev. B
branch dev. C
project
Adding some functionality with different approaches from dev. A and B
task 1 approach A
task 1 approach B
task 2
stops here because it has come clear that Bs approach is better
project
project

Figure 8: Overview about branching and merging in Git
source: self-drawn

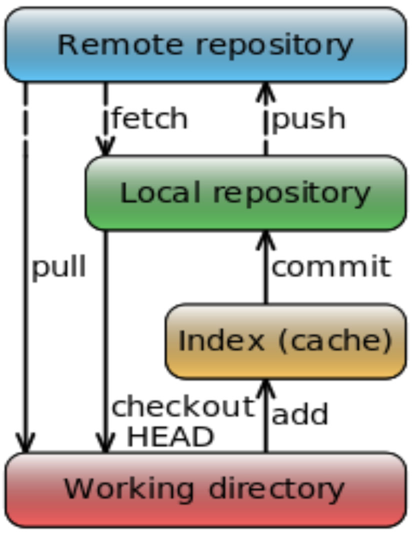


Figure 9: Git workflow [4]

Figure 9 shows the workflow from checking out a Git repository from a remote location to the local working directory and the commit afterwards back to the remote repository. It also shows that a remote repository exists apart from the local one. That makes it possible to work with Git without a network connection by committing the changes to the local repository. When there is an Internet connection available, the work can be pushed to or pulled from the remote repository.

3.1.3. Pair programming:

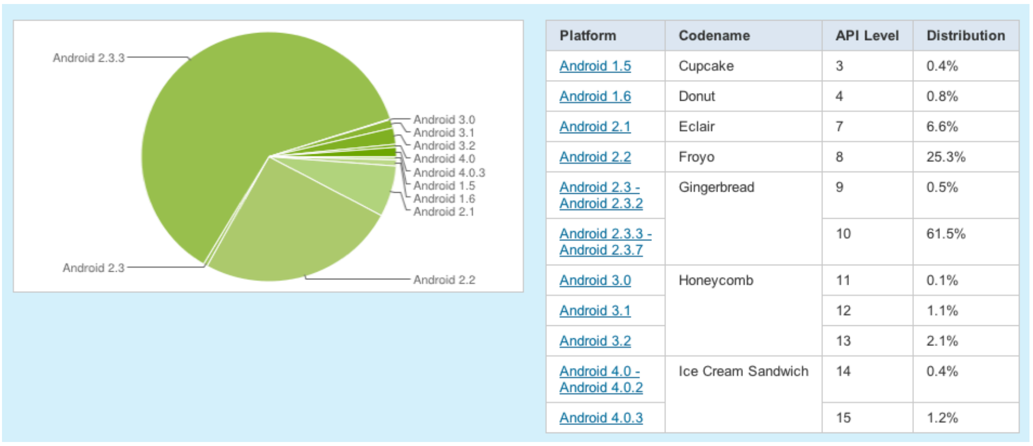
Pair programming is a software development technique that is often used when two programmers work together on the solution for a problem. At pair programming both programmers share one workstation. The programmers are divided in two roles, the driver (who actually writes code) and the observer (who reviews each line of code). The roles are switched frequently, which comes along with the advantage that nobody gets tired of his/her function. The observers' job is to consider the direction of the work and immediately remember the driver when something went wrong. This makes it possible that the driver can completely focus on the current task (e.g. the algorithm to develop) instead of caring too much for the source code syntax. [10]

4. Implementation

4.1. Analysis of requirements

4.1.1. Android

To get the program running on as many devices as possible, it is necessary to get an overview about the distribution of the different versions. Figure 10 shows how the distribution of android versions was on March 5th 2012.



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.4%
Android 1.6	Donut	4	0.8%
Android 2.1	Eclair	7	6.6%
Android 2.2	Froyo	8	25.3%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	61.5%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.1%
Android 3.2		13	2.1%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.4%
Android 4.0.3		15	1.2%

Figure 10: Distribution Android versions March 5th 2012 [5]

Android 2.3.3 and higher covers over 50% of all devices and because of that it seems logical to develop for this version. That means, API level 10 will be used.

4.1.2. Functionality

The program resulting from this project must be able to:

- Process pictures, in detail:
  - Take pictures from camera
  - Load pictures from file system
- Recognize if the picture contains a coordinate-cross including graph
- Present the graph related values in a simple but effective way

4.2. Rough Design

4.2.1. Programming

The analysis of required functionality (Chapter 4.1.2) leads to two different main application steps (three in case of a implemented help-page). The source code for this will be organized in a logical packet structure:

Package

Responsibility

miun.android.ungraph

Main package including classes that will be used in more than one sub-package (e.g. exceptions)

miun.android.ungraph.preview

Including the classes used for showing the preview of the picture. Accessing the camera, recognize coordinate cross, gives user the feedback (preview).

miun.android.ungraph.process

Shows the result. Including the algorithm to detect the graph in the coordinate cross. Gives user the values if available or a feedback if not.

miun.android.ungraph.help

Including the classes that are necessary to show a help screen.

4.2.2. User Interface

In total it is necessary to have at least two screens

- One which offers the possibility to put a picture into the application (from camera or a saved one)
- One which presents the results from the input
  - Additionally a third screen providing the user information how the program is used and to show answers to frequently asked questions might be a good idea.

4.3. Fine Design

4.3.1. Programming

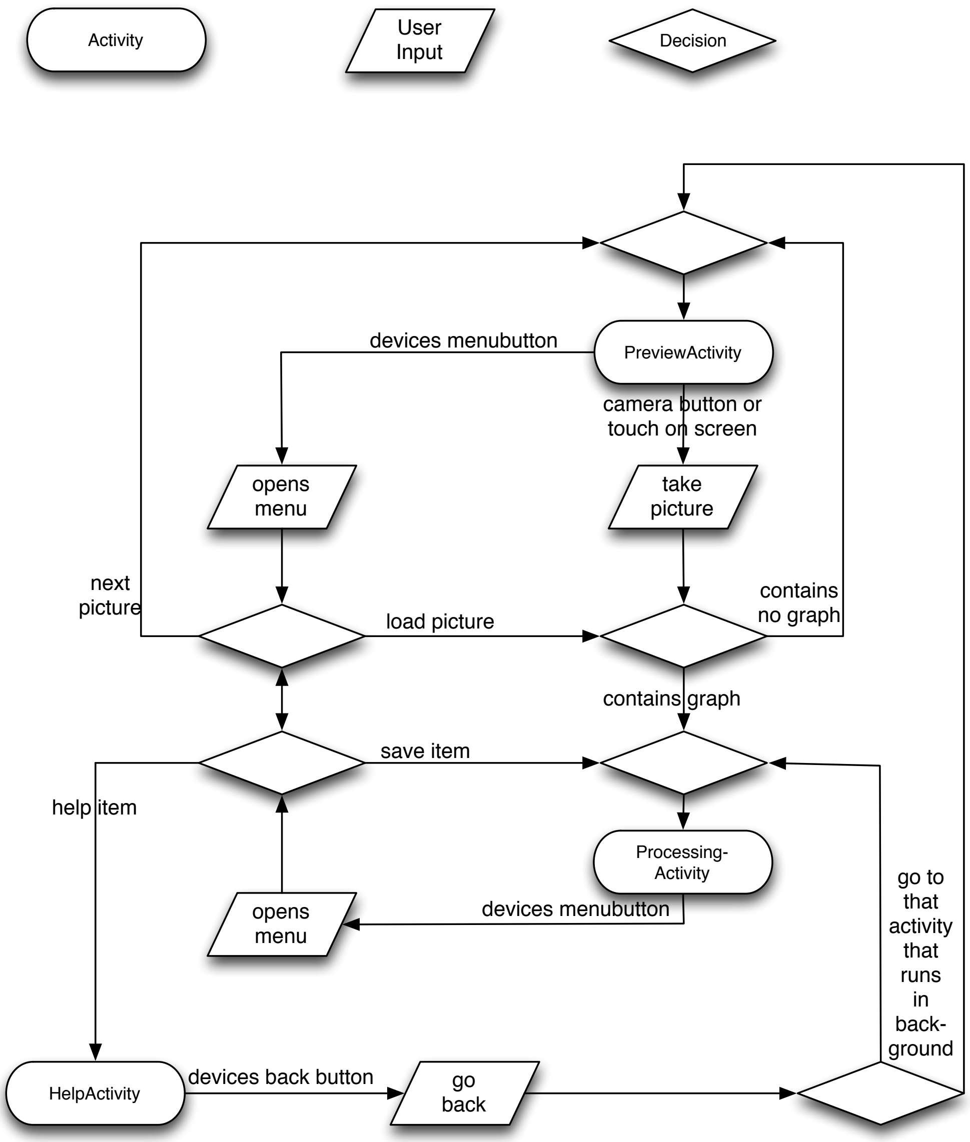


Figure 11: Application cycle plan
source: self-drawn

The activity diagram in Figure 11 shows how the application lifecycle will look like.

4.3.2. User Interface

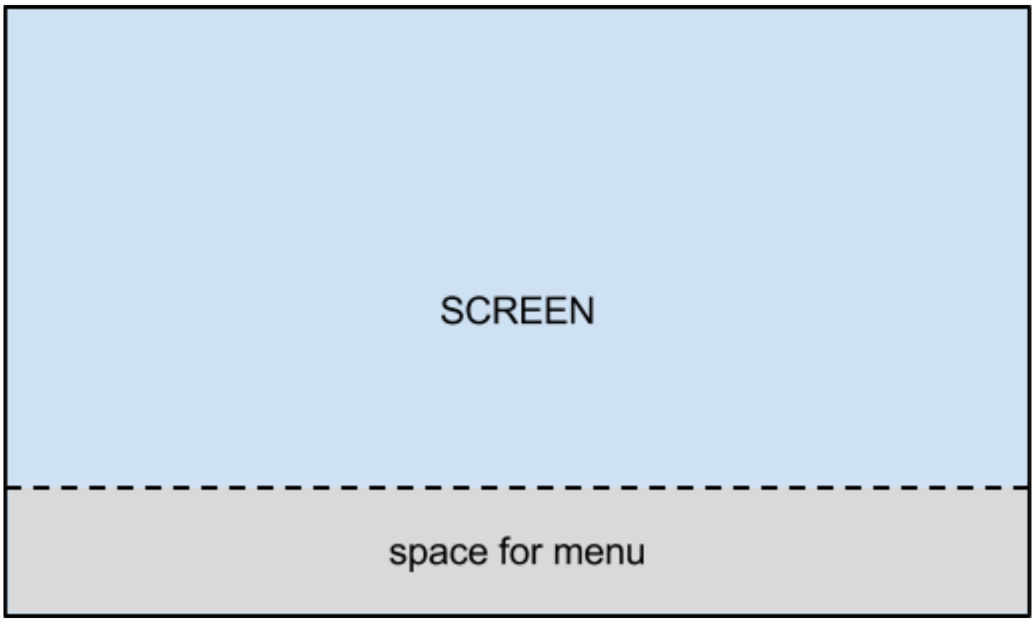


Figure 12: Gui in landscape mode

The GUI will be in landscape mode (Shown in Figure 12). This is because it seems more intuitive to use the camera in that mode than in portrait mode. Additionally the camera button is placed on the right side of the most handsets. For devices not equipped with a camera button, there will be an extra functionality to capture a picture. Namely a touch on the preview picture should also trigger the `takePicture()` function.

4.4. Testing

Testing will be done with the standard Android testing framework [6]. This framework is based on JUNIT and provides different testcase-interfaces specially designed for Androids application/activity testing. In addition to the standard framework Robotium [7] will be used. Robotium provides an easier access to the graphical user interface (GUI) objects and makes it more comfortable to test those and the application flow. Another option to use would have been the Roboletctric [8] extension that is based on JUNIT4 (Robotium on JUNIT3). But because of the fact, that Robolectric is not as well documented in the basic books it is not used in this project.

5. Result

The projects source code can be cloned from [11].
Figure 13 shows the preview screen with the opened menu. The application starts up with this screen. The search area is marked with green lines. The menu allows to load pictures from the file system or to view help.

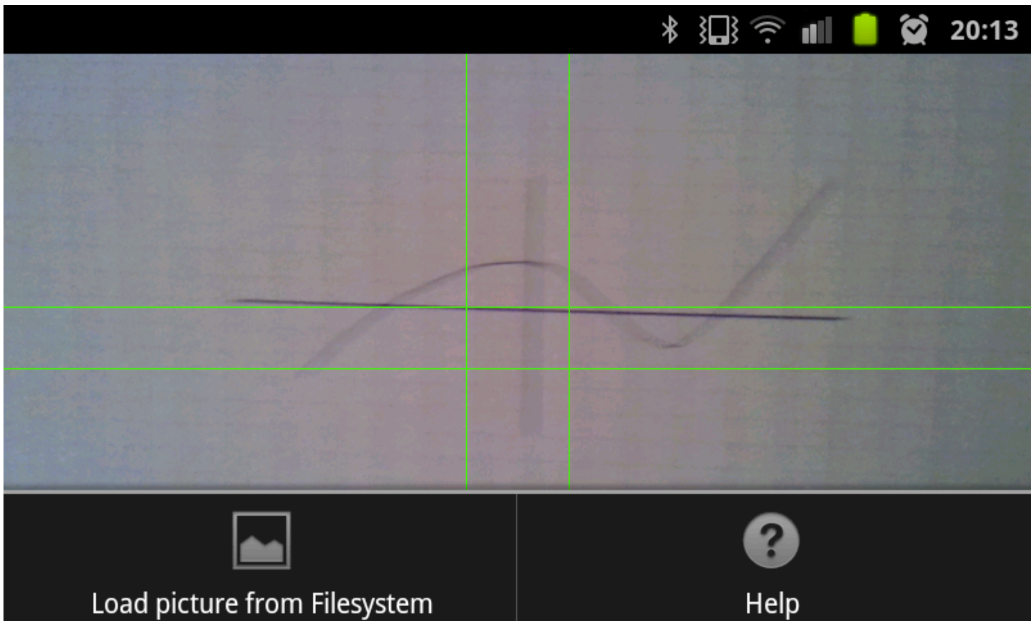


Figure 13: PreviewActivity

On selecting the load item from the menu an intent is issued to load a picture. If more than one app is installed that can handle this intent, the user is able to select this app (Figure 14).

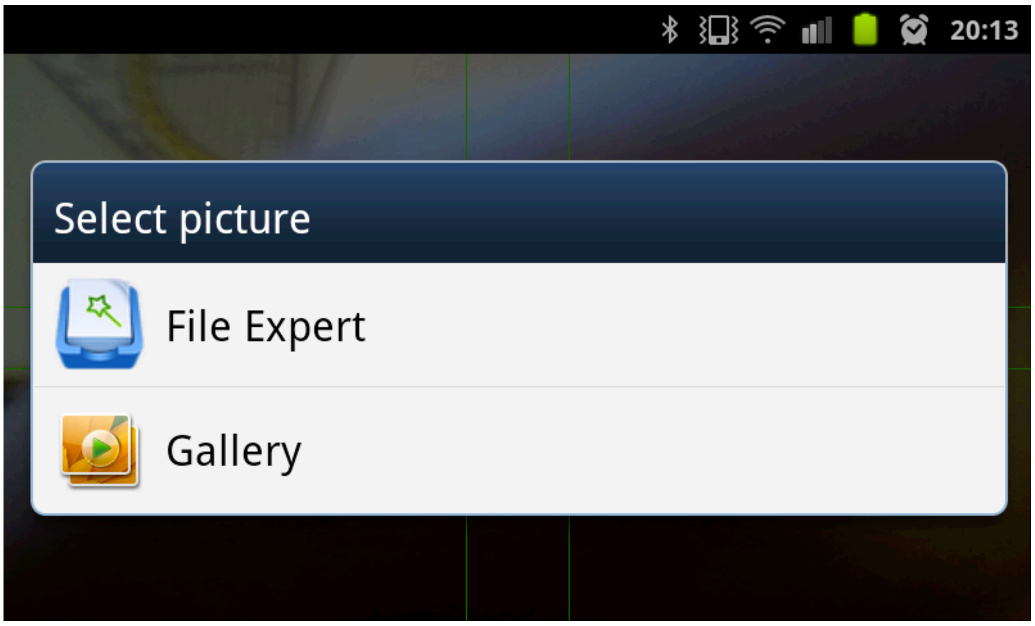


Figure 14: Load image

After a graph has been found and analysed the screen shown in Figure 15 comes up. The red lines are highlighting the axes and the blue graph indicates data-points. Tabbing on the screen the user can select a point in the graph (then represented by the yellow line) and gets the relating values presented in the top left corner.

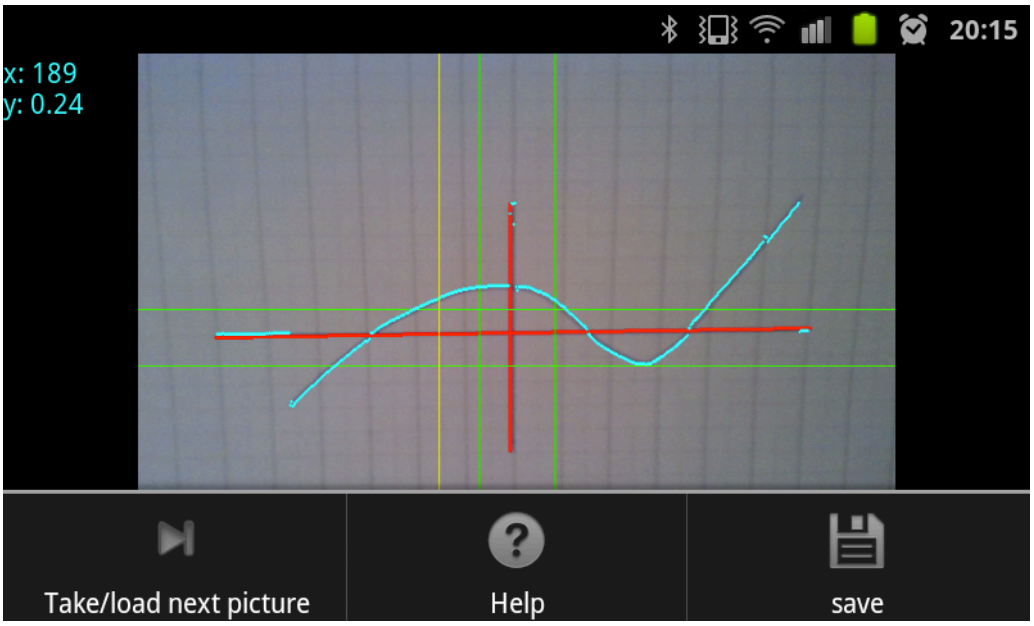
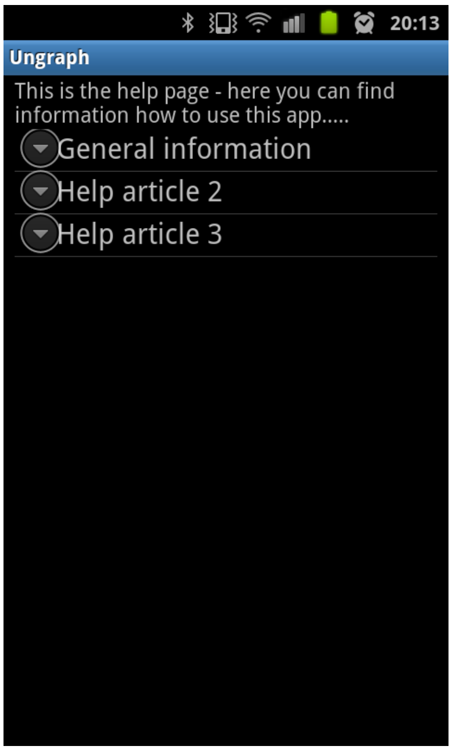


Figure 15: GraphProcessingActivity



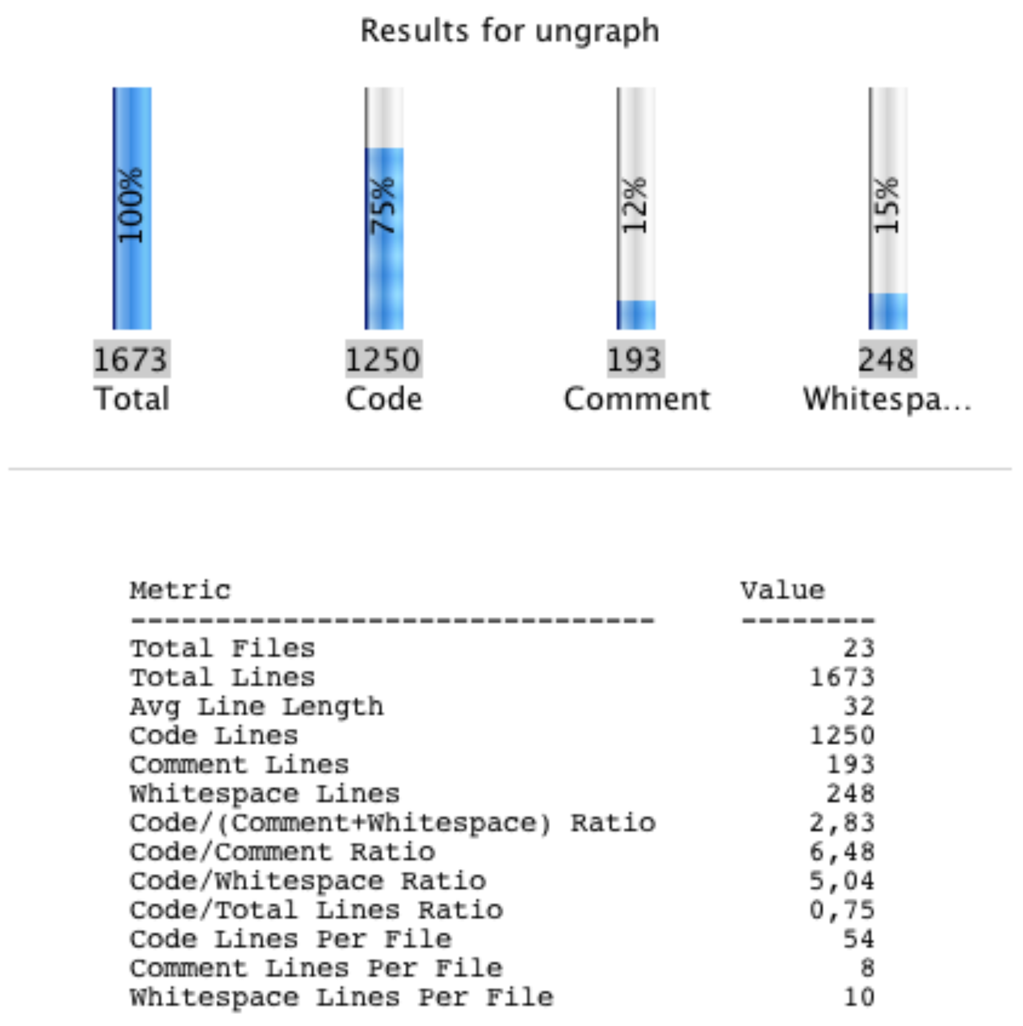
This is the help page - here you can find information how to use this app.....

This is the help page - here you can find information how to use this app.....

Figure 16: HelpActivity

Figure 16 shows the *HelpActivity*. The user is able to select a specific article with a click. The *HelpActivity* is the only screen that can be shown in either, vertical or horizontal direction.

5.1. Code analysis



```
Metric                              Value
-------------------------------     --------
Total Files                               23
Total Lines                             1673
Avg Line Length                           32
Code Lines                              1250
Comment Lines                            193
Whitespace Lines                         248
Code/(Comment+Whitespace) Ratio         2,83
Code/Comment Ratio                      6,48
Code/Whitespace Ratio                   5,04
Code/Total Lines Ratio                  0,75
Code Lines Per File                       54
Comment Lines Per File                     8
Whitespace Lines Per File                 10
```

Figure 17: Analysis of source code

Figure 17 shows that the project consists in total of 1250 lines of code in *.java and *.xml files. 142 of this lines are written in the *.xml files and are describing values and layouts and so on. The other 1108 lines are java code. On average every 7th line of code a comment line comes up, that means the code is easy to understand for future developments. This code analysis was made by the help of Code Analyzer 0.7.0 [9].

6. Discussion

6.1. Test Driven Development

TDD was a promising and interesting approach for software development. The limits that have been detected during the project are discussed below.

First TDD requires a very good knowledge of the used software framework and API. If this is not given it is quite impossible to write a test that "drives" the programmer to the desired functionality because the idea of the wanted code is too vague to produce a precise test.

Following up the last point the process of having ideas slowed down a bit. Programmers often have creative ideas while developing. Because TDD is inserted before the development it is less likely that the programmer stumbles upon interesting new ways. This could also be an advantage because nothing unnecessary is implemented.

Another point is that it is very hard to create a test for a data source that produces unpredictable or random data. The android camera is such a source. It is easy to test that there IS a picture but when it comes to the testing of the algorithms that find information in the picture that are not easily reproducible by hand, testing fails. In other words the algorithm cannot check itself when it is used on unpredictable data.

6.2. Android Camera

At the moment only android operating system below version 3 support our way of camera usage. From version 3 and higher a security feature has been implemented by Android to prevent hidden use of the camera. In detail this means that there must always be an unchanged preview surface attached to the camera driver. As the *ungraph* app uses a data buffer and displays its own version of the preview this results in a black picture in preview mode.

6.3. Efficiency

The efficiency of the *ungraph* app is only crucial when it comes to the image processing part. As most of the image processing is done with the OpenCV library, which is run native on the android device with the use of JNI calls, the speed should be near the maximum possible. The detection of the graph is a self-made algorithm the accesses a lot of pixel in the bitmap picture containing the graph. Each pixel access has the cost of one JNI call which is rather slow because the java virtual machine has to call the native library function. See the next chapter for a possible future development of the graph detection algorithm.

6.4. Future Visions

- Some possible further developments could be:
- The implementation of a faster version of the graph detection algorithm as a native android function
- Detection of different coordinate-cross styles
- Removal of “noisy” points in the detected graph
- User-defined scaling of the axis range (At the moment [-1; 1] is pre-set for both axes)
- Detection of a not centred coordinate-cross
- Detection and compensation of a rotated coordinate-cross
- Zooming function
- Detection of minima / maxima
- Possibility to use logarithmic scales for both axes
- Saving the detected graph as a list of values to a file/database
- Support android OS from version 3 and higher

7. Conclusion

This conclusion should add some more information about the use of Test Driven Development during the project. We stopped the use because we were unable to solve some severe issues with the used libraries and other reasons mentioned in the discussion chapter before.

We nevertheless do not think that TDD is not a helpful tool for development, but the scope where it could be used might not be the whole application. To be efficient with TDD a programmer needs good knowledge of the application framework and it is less suitable for beginners in a certain type of system. The actual decision to stop TDD was made because we would not have a presentable application in the given amount of time.

References

[1] Canny edge detector 07.03.2012
http://en.wikipedia.org/wiki/Canny_edge_detector
[2] Hough transform 07.03.2012
http://en.wikipedia.org/wiki/Hough_transform
[3] Test Driven Development 07.03.2012
http://en.wikipedia.org/wiki/Test-driven_development
[4] Git distributed revision control system 07.03.2012
http://en.wikipedia.org/wiki/Git_(software)
[5] Android platform versions 10.03.2012
http://developer.android.com/resources/dashboard/platform-versions.html
[6] Android testing framework 10.03.2012
http://developer.android.com/guide/topics/testing/testing_android.html
[7] Robotium 10.03.2012
http://code.google.com/p/robotium/
[8] Robolectric 10.03.2012
http://pivotal.github.com/robolectric/
[9] Code Analyzer 0.7.0 18.03.2012
http://sourceforge.net/projects/codeanalyze-gpl/
[10] Pair programming 18.03.2012
http://en.wikipedia.org/wiki/Pair_programming
[11] Public ungraph Git repository
git@github.com:miun/ungraph.git