TECHNISCHE UNIVERSITÄT DARMSTADT

ANDROID SIGNAL ANALYZER - ANSIAN

STEFFEN KREIS AND MARKUS GRAU

Secure Mobile Networking Lab Exercise

September 4, 2015

Secure Mobile Networking Lab
Department of Computer Science

SECURE MOBILE NETWORKING

Android Signal Analyzer - ANSIAN
Secure Mobile Networking Lab Exercise

Submitted by Steffen Kreis and Markus Grau
Date of submission: September 4, 2015

Supervisor: Jiska Classen

Technische Universität Darmstadt
Department of Computer Science
Secure Mobile Networking Lab

CONTENTS

i PROJECT REPORT 1
1 Introduction . . . 3
2 Related work . . . 3
2.1 State of the art analysis . . . 3
2.2 Fundamentals . . . 4
3 Basic concept . . . 5
4 Project progress . . . 6
4.1 Project management . . . 6
4.2 Planned time . . . 7
4.3 Actual progress . . . 9
4.4 Spent time . . . 11
5 Implementation of AnSiAn . . . 13
5.1 Libraries and foreign code . . . 13
5.2 GUI . . . 14
5.3 Structural changes . . . 18
5.4 Wider band range scanning mod . . . 21
5.5 Morse . . . 21
6 Perspective . . . 23
6.1 New possible features . . . 23
6.2 Github . . . 23
7 Conclusion . . . 23

ii APPENDIX 25
A RELEASE NOTES 27
1 Hardware . . . 27
1.1 Supported devices . . . 27
1.2 Tuners . . . 27
1.3 Power supply . . . 27
2 Contents . . . 28
3 Compilation . . . 28
4 Deployment . . . 28
4.1 Uninstall . . . 28
4.2 Install . . . 28
B ANSIAN MANUAL 31
1 Overview . . . 31
2 Main panel . . . 32
2.1 Morse . . . 32
2.2 Analyzer . . . 33
2.3 Waveform . . . 35
3 Toolbar . . . 35
3.1 Start/Pause/Stop . . . 35
3.2 Demodulation . . . 36

3.3 Recording 37
3.4 Gain 37
3.5 Frequency 38
4 Settings 38
4.1 Misc(ellaneous) 38
4.2 GUI 39
4.3 Morse 40
4.4 Alarm 41
C SPENT TIME BREAKDOWN 43

LIST OF FIGURES

Figure 0.1 Time schedule for the whole project part 1 8
Figure 0.2 Time schedule for the whole project part 2 9
Figure 0.3 Sprint 1 burndown 11
Figure 0.4 Sprint 2 burndown 12
Figure 0.5 Sprint 3 burndown 12
Figure 0.6 Sprint 4 burndown 13
Figure 0.7 WiCat GUI 15
Figure 0.8 WiCat reworked GUI 15
Figure 0.9 RFAnalyzer 16
Figure 0.10 AnSiAn GUI 17
Figure 0.11 GUI classes 18
Figure 0.12 Data input flow 20
Figure B.1 Main screen 31
Figure B.2 Notification 31
Figure B.3 Morse view 32
Figure B.4 Analyzer view 33
Figure B.5 Normal monitoring mode landscape 34
Figure B.6 Scanning mode landscape 34
Figure B.7 Scanning mode landscape 35
Figure B.8 Demodulation selection dialog 36
Figure B.9 Demodulation frequency selector 37
Figure B.10 Recording dialog 37
Figure B.11 Gain dialog 37
Figure B.12 Frequency dialog 38
Figure B.13 Misc settings 38
Figure B.14 GUI settings 39
Figure B.15 Morse settings 40
Figure B.16 Alarm settings 41

ACRONYMS

DFT Discrete Fourier transformation

GUI Graphical user interface

FFT Fast Fourier transformation

SDR Software-defined radio

DVB-T Digital video broadcasting - terrastrial

DAB Digital audio broadcasting

MSPS Million samples per second

ITU International Communication Union

BPSK Binary phase shift keying

AM Amplitude modulation

FM Frequency modulation

Part I

PROJECT REPORT

ANSIAN - ANDROID SIGNAL ANALYZER

1 INTRODUCTION

The basic usage of AnSiAn, the Android Signal Analyzer, is finding, analyzing and demodulating wireless signals on the go. In contrary to regular stationary signal analyzers, the solution of using an Android device together with a Digital video broadcasting - terrastrial (DVB-T) dongle enables this mobile usage at minimum costs. The downside of having lower bandwidth and accuracy in comparison with a regular stationary analyzer can easily be ignored in many use cases, considering the mobility, cost and expansion of possibilities (e.g. demodulation). The initial project definition:
Walking through the city and inconspicuously sniffing wireless protocols - this can be established with our Android signal analyzing software, written in Java. However, some features are still missing...
Your tasks:

- Increase Graphical user interface (GUI) performance
- Decode bits from standard modulation schemes like Binary phase shift keying (BPSK)
- Choose additional features you like depending on group size

2 RELATED WORK

2.1 *State of the art analysis*

There are different approaches for the analysis of signals. For exact measurements nothing compares to stationary analyzers, which, as their name indicates, are not built for mobile use. There are handheld fully-integrated signal analyzers as well. However, for many purposes, lower accuracy is also sufficient. Regarding this, the downside of both dedicated devices is their relatively high acquisition cost. The high computational power of personal computers and embedded devices nowadays enables Software-defined radio (SDR), where many of the hardware elements of a signal analyzer are implemented in software. This reduces the hardware cost and allows signal analysis to be performed at a very low price. There are several implementations of software to perform SDR for desktop and notebook use, but even smartphones and tablets are equipped with enough computational power to realize SDR. The following software listing represents the most popular solutions available for SDR both on mobile and desktop devices.

2.1.1 *Desktop software*

Gqrx and GNU Radio Companion are both based on the GNU Radio toolkit, which is an open source toolkit licensed under the GNU GPL v3. Its purpose is to provide a collection of signal processing tools which can be used by other software with user interfaces to handle SDR. The GNU Radio toolkit is very powerful and so are gqrx and

the GNU Radio companion. Gqrx offers an intuitive GUI with a standard user interface. The GNU Radio companion lets the user build the data flow with modules. This requires some basic knowledge about the process, but the learning curve is much higher. They both support demodulations SDR# (SDRSharp) is a standalone closed source application for MS Windows. It supports several devices.

2.1.2 *Android applications*

The applications SDR Touch and Wavesink Plus are both paid-for solutions with a trial version available. They offer analog radio demodulations and spectrum analysis. Wavesink Plus also supports Digital audio broadcasting (DAB) radio. RFAnalyzer is an open-source project with several demodulations built in. WiCat originated from a lab group in the previous SEEMOO lab/project course (WS14/15). It is very basic and is missing some features like demodulations.

2.2 *Fundamentals*

The replacement of several hardware components by software implementations, the additional features and the graphical representation require some knowledge about specific algorithms, appropriate data representation forms, demodulation/decoding and other background knowledge. Below are some descriptions of the most important issues to give a little overview.

2.2.1 *SDR and the RTL chipset*

As described above, in Software-defined radio most hardware components are replaced by software implementations. There are different SDR devices which differ in functionality and price. Dedicated SDR devices like the HackRF are not only capable of receiving data on a operating range from 10 MHz to 6 GHz, but can also transmit on the same range. The SDR device we used during development is a RTL-SDR DVB-T dongle (Terratec Cinergy TStickRC Rev. 3, RTL2832U chipset and E4000 tuner). RTL means it uses the RTL chipset. Dongles like this one were originally developed for just watching terrestrial television programs. However, some enthusiasts found out that the raw I/Q data could be obtained directly, which extends the possible usage of these dongles to regular SDR receiver devices. The built in tuner is responsible for the overall bandwidth of the device. In our case, the tuner is capable of 52 - 2200 MHz with a gap from 1100 - 1250 MHz.

2.2.2 *Waveform & FFT*

The Waveform is the graphical representation of a signal in its time domain. The Fast Fourier transformation (FFT) is an algorithm for the efficient calculation of the Discrete Fourier transformation (DFT) and is used to transform a signal from the time domain into the frequency domain (and vice versa). This is used to analyze and visualize the frequency components of a signal and thus its spectrum.

2.2.3 *Modulation & demodulation*

Modulation describes the process of manipulating a relatively high carrier frequency with a signal that contains information to be transmitted. This procedure allows the

simultaneous transmission of several signals on neighboring frequency spectra. Broadcasting the low frequency signal unaltered would require a much higher spectrum and huge antennas. Well known modulations are Amplitude modulation (AM) and Frequency modulation (FM) which are used by public radio stations to broadcast their content. Demodulation is the process at the other end, where the lower frequency signal containing useful information is extracted from the received signal.

2.2.4 *Morse code*

Morse code is a technique of transmitting text by encoding letters with on-off keying. This means the carrier wave is simply turned on and off to encode the letters. The corresponding codes for the letters are defined in different Morse alphabets, one is the international Morse alphabet from the International Communication Union (ITU)[1]. The codes consist of dots and dashes. For example "... — ..." is the coded version of SOS. The timings for the on-off keying are as follows:

- a dot "." equals one time unit signal on, also called dit
- a dash "-" equals three time units signal on, also called dah
- one space between word is seven time units signal off, also called medium gap

Further is defined that between each dot and dash for one time unit the signal is off and between each letter three time units the signal is off. Continuing the SOS example the send signal for SOS would be 0101010001110111011100010101010.

3 BASIC CONCEPT

The basic concept is composed by the project requirements, existing implementation and the state-of-the-art analysis. Out of the project topic and the first meeting with the supervisor Jiska Classen the following requirements were identified:

- Increasing the user interface performance of the android application WiCat
- Rework the layout for better user interaction like zoom in waterfall and fixing bugs
- Decoding bits from standard modulation schemes like BPSK
- Enable wider band ranges than the driver allows by splitting the band range and scanning the single parts
- Implement additional features with priority for the basic features defined by the other requirements

So the basic idea was to increase the usability of the application by making it intuitive, with the possibilities the Android API provides. To achieve this, many things in the GUI can be done, like the usage of gestures, following the thumb rule and making it possible to use the whole application with one hand. Keeping the user informed about the system status and the prevention of errors are also important principles to follow.
Another important issue to enable the implementation of new features and functionality

1 http://www.itu.int/rec/R-REC-M.1677-1-200910-I/

is the expandability. WiCat has many dependencies in the main activity which complicate the implementation of new features. Large parts of the code could be separated to decouple the functionalities, to provide expandability and to keep the modules as reusable as possible. The expandability is also important for the future reusability, which is another fundamental goal to achieve.

4 PROJECT PROGRESS

4.1 *Project management*

4.1.1 *Scrum*

In our research for an appropriate development methodology we found agile methods to match best with our adaptive project. We were looking for a flexible and adaptive solution which supports us in keeping focus on which parts of the project need to be done next and keep an overview of all tasks, which then again helps keeping the overall work load rather constant. Scrum fulfills most of our requirements. A well-known downside of agile methods is the lack of documentation because they seem to only focus on implementation. We improved this by including quite a fair amount of project resource into documentation directly in our Scrum project plan.
Regular Scrum project plans for full-time projects recommend a daily scrum and 2-4 week sprints. However, we adapted this to better fit our project needs and decided to run weekly scrums because we only work 13 hours and not in full-time 40 hours a week on the project. For our sprints we chose a cycle of four weeks which results in a total of four sprints for the whole project.

4.1.2 *Agilefant*

There is a vast number of tools that can be used for Scrum-based projects. We decided to use the open-source tool Agilefant for the project management. We will give an overview in this document.
We attempted several project management tools to find out which one best fits our needs. Most of these tools were especially developed for Scrum- or at least agile-based projects. We also tried out generic tools with optional Scrum-functionality like Microsoft Project. This research took a lot of time because of various downsides of the tools like restrictions of free use, lack of functionality or we found them hard to get started with.
We had heard of Agilefant before, the only minor downside was that we needed to set it up on our own server, which needed some time. This lead us to try out alternatives in the first place. On the other hand the own server has its benefit too, because you have the full control over the backend. So you don't have any dependencies from third parties. You have additional work with hosting and maintenance, but won't have problems like a terminated service or support in future. Another advantage of Agilefant in comparison to others is that it is a free open-source tool. It is web-based, so it can be used with a browser, independent of the underlying operating system. It concerns us as we are using different operating systems. Agilefant is provided with basic tutorials which helped us getting started. After working with these, we found Agilefant easy to use.
A downside we discovered later was the restriction of dependencies in the open source

version. Currently dependencies can't be set for the tasks and user stories. But this disadvantage can simply be compensated by a good task distribution and the weekly scrums. In a team of two persons this can be done in quite a proper way without dependencies in the management tool. An additional help is the task queue, which helps to order the tasks to avoid waiting times. Another benefit worth mentioning is that this project management can simply be used for future projects on AnSiAn.

4.2 *Planned time*

The following schedule shows the initial time management for the whole project.

Project week	Sprint	Planned work	Week number
1	Initial project setup	Selection of topic	16
2		Become acquainted with documentation and source code, fetch and setup hardware, setup development environment, compile application, get first experiences with android app, first meeting with supervisor	17
3		Project Definition: selection of an appropiate software development methodology, definition of project requirements, extraction of needed tasks, time planning for the whole project, scheduling and distribution of tasks for the first sprint	18
4	First sprint	20% Become acquainted with existing implementation 40% Implementation 40% Documentation and project management	19
5			20
6			21
7			22
8	Second sprint	80% Implementation 20% Documentation and project management	23
9			24
10			25
11			26
12	Third sprint	80% Implementation 20% Documentation and project management	27
13			28
14			29
15			30

Figure 0.1: Time schedule for the whole project part 1

16	Fourth sprint	50% Implementation 50% Documentation and project management	31
17			32
18			33
19			34
20	Project finalization	Final version of developed code, including all change requests by advisor, draft of documentation, first version of slides	35
21		Final slides, final documentation	36
TBD		Final presentation and code demo	TBD

Figure 0.2: Time schedule for the whole project part 2

4.3 *Actual progress*

The actual progress differed from the planned schedule. Several reasons were responsible for that, provided in the following descriptions.

4.3.1 *First sprint*

During the work on the tasks of the first sprint, we found that some parts of the given Android application WiCat we were working on are taken from the GPL version 2[2] project RFAnalyzer by Dennis Mantz [3]. This lead us to take a closer look and try the RFAnalyzer app. In comparison with WiCat, there are several major advantages of RFAnalyzer, such as:

- waterfall with level diagram
- GUI with zoom and frequency swipe
- better performance
- several demodulation modes
- more supported scanning devices (not just the RTL2832U with the E4000 Tuner)

At the end of the first Scrum, we found that it would be the best solution to integrate more of RFAnalyzer in our App. So we created a new project basis and build up from

2 http://www.gnu.org/licenses/gpl.html
3 http://github.com/demantz/RFAnalyzer

new on the RFAnalyzer. The downside was the missing waveform visualization and a bit loss of time for the completed work on the WiCat app. But the waveform implementation also bases on a GPL project, which can be reused and is not worked out very much. And the most of the completed implementation was the GUI rework, which could be transferred in big parts to the new implementation. However, if we would have continued and implemented everything completely on our own, there would have been less time for developing even more features. The lost time was caught up by the advantages of the RFAnalyzer and succeeded in a better and mightier solution. Simply put, the rebase was the right step. RFAnalyzer has some features which fullfill some of the requirements, it does not have unwanted performance issues and it is just much more melloy.

4.3.2 *Second sprint*

In the second sprint, there were major changes on the project, because a complete project workover on a new basis was started. The project was renamed to AnSiAn - **An**droid **Si**gnal **An**alyzer. The first challenge was the reuse of our previously developed GUI concept for the WiCat application. From this, the tab layout was ported. Because of the data structures in the RFAnalyzer, it was not useful to reuse the Waveform view from the WiCat app. This lead us to use the open source waveform view which the WiCat waveform solution was based on.[4] Things like the thumb rule and general design decisions we already determined and implemented in WiCat had to be reimplemented. The RFAnalyzer could not run in the background, because much of the functionality was implemented directly in the main activity. In order to run in background, these functionalities were outsourced into a service. This service also allows an orientation change without the driver to restart. To achieve this, many structural and data model changes were necessary. These changes supported the intention to split the responsibilities and to get closer to a MVC pattern. This was fundamental for our further approach and the future expandability of the project. As the alpha release deadline was in mid of the second sprint, not all of these improvements were implemented at that point.

4.3.3 *Third sprint*

The plans for the third sprint were the first ones with planned feature implementations, unlike for the first and second sprint which mainly covered reengineering, bugfixes and project management. One of these features is the wider band range scanning. This provides the option to show a wider band range through multiplexing the scanner fft results of neighboring frequency spectra. Another one was the implementation of a demodulation. RFAnalyzer already supported several demodulations like AM and FM radio. In order to provide a useful demodulation, which for example could be used in the CriCom course, the decision came to the implementation of Morse. The third planned feature was the rework of the existing recording function including an alarm trigger, which can also be used to start the recording. All these features and the nearly finished GUI should be delivered with the beta release. As distinguished from the previous plan, which included several feature implementations, the state at the end of the third sprint shows that many other improvements became more important. Several additional structural changes have been made, which were very time-consuming. These changes include the splitting of the main GUI view to make components reusable. Further

4 `https://github.com/google/ringdroid`

improvements to the FFT handling increase the number of calculated and actually used samples. As a consequence, not all the planned features were fully implemented by the end of the third sprint and therefore had to be moved to the fourth sprint.

4.3.4 *Fourth sprint*

The end of the fourth sprint also was the deadline for the final code release. This meant, all the planned features were supposed to be finished at this point. The fourth sprint also contained the finalization of the GUI and debugging. Additionally to the programming part, the plan for the fourth sprint covered a draft of the final documentation and the preparation of the slides for the presentation. As mentioned before, the fourth sprint had to cover some features originally planned for the third sprint. This delayed the preparation of the slides, as they could be postponed into the project finalization and hence had lower priority at that point of time. At the end of the fourth sprint, all planned features were finally built in. However, some details of those features could still be improved.

4.3.5 *Project finalization*

The due date of the final code was one week ahead of the documentation and presentation deadline. The project finalization covered the completion of both the presentation slides and this project documentation. Both were finished in time and include all the planned content.

4.4 *Spent time*

For every single sprint we planned about 4 weeks, so we would have to invest about 104 hours for one sprint. The following figures show a detailed view on how the time was spent per sprint. These figures were exported from the Agilefant project management tool. The scheduled times were manually added, unfortunately the time axis for the second sprint is too small to denote the scheduled time.

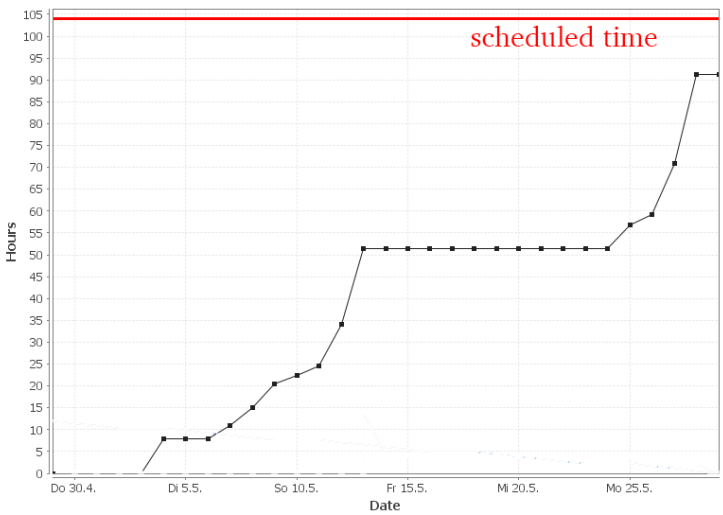


Figure 0.3: Sprint 1 burndown

The first sprint was with 101 hours slightly under the planned time.

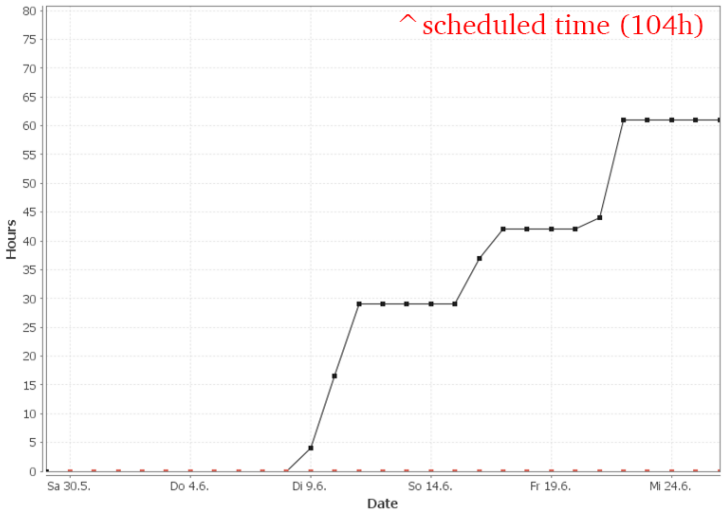


Figure 0.4: Sprint 2 burndown

The second sprint had only 77 hours. Main reason for that lower amount of spent time was other workload at this time, but the sprint goals were achieved.

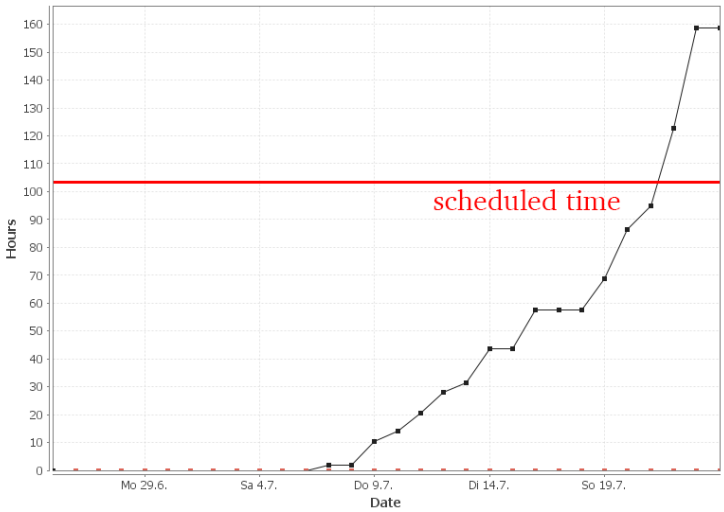


Figure 0.5: Sprint 3 burndown

The third sprint was with over 158 hours overloaded regarding the planned 104 hours per sprint. However, not all goals were achieved. Reflecting that, the needed work for the set goals was hard underestimated in the third sprint.

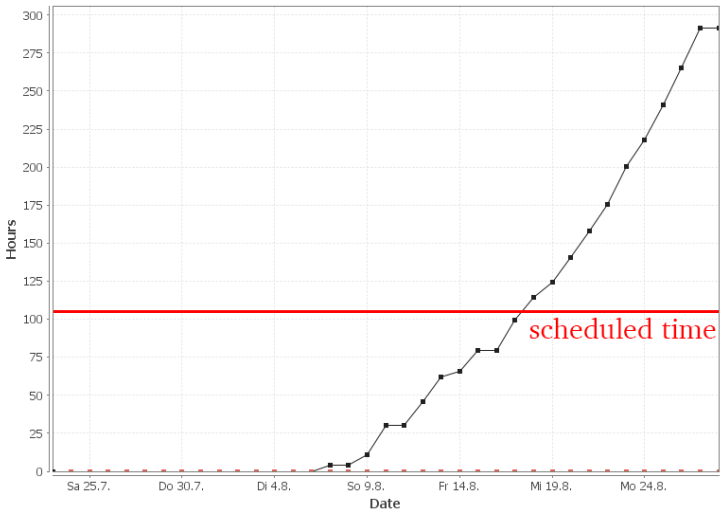


Figure 0.6: Sprint 4 burndown

The leftover tasks from the third sprint had to be done in the fourth sprint, additionally to the already planned tasks. This again lead to a huge amount of time spent in the the fourth sprint, 291 hours to be exact.

With the initial project setup and the project finalization, a total amount of 757 hours were contributed to the project. This is immense, regarding the 540 estimated hours calculated based on the credit points for the project. The motivation for the project helped very much to do those extra hours. An Agilefant breakdown of the total spent time can be seen on the attached document C

5 IMPLEMENTATION OF ANSIAN

5.1 *Libraries and foreign code*

Besides the basic projects WiCat and RFAnalyzer some libraries and other foreign code was adapted and used to enhance the project.

5.1.1 *Waveform*

The WaveformView was roughly taken from the ringdroid[5] project. However, this mostly helped to get a basic idea on how to draw a suitable waveform and due to adaptions, not much of this code is used anymore.

5.1.2 *EventBus*

The EventBus provides a comfortable way to distribute events and data to other modules. The EventBus simplifies communication between these modules. A module that wants to be notified, simply registers itself at the EventBus and implements a method for every event it wants to get notified about. Events are completely customizable and can hold data.

5 `https://github.com/google/ringdroid`

5.1.3 *HackRF USB library*

The default HackRF USB library was also implemented by the developer of the RFAnalyzer. As this library does not support the supposedly HackRF compatible rad10, an improved version with a little fix was taken from a RFAnalyzer fork for rad10[6].

5.1.4 *Morsecoder*

Morsecoder[7] is a simple Java application to encode and decode Morse. It is licensed under the GNU GPL v3[8]. Some classes are adapted and reused for morse decoding and encoding.

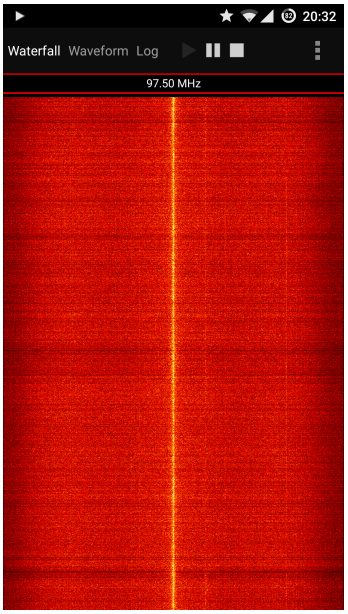
5.2 *GUI*

The GUI was completely reworked and modularized. The additional Screens required new fragments and therefore tabs were needed to offer a proper handling. Some components were reused from the RFAnalyzer, but they were worked over and were adapted to the new structure. Due to the fact that there was a hard project rebasing, at first a short history of the main changes of the GUI should be emphasized.

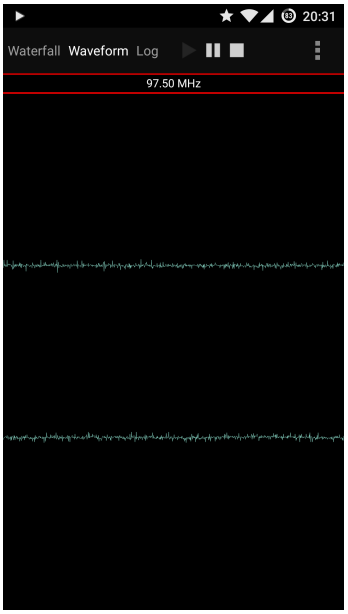
5.2.1 *WiCat*

The implementation started with the given WiCat application from Bernd Conrad and Philipp Roskosch. The user interface consists of three main views0.7. The first is used to show the waterfall, the second for the corresponding waveform and the third for logging. To interact with the application, dialogs and the buttons at the top have to be used. There is no support for gestures, which would enhance the usability. Also there are heavy problems with orientation change. An orientation change brings the app in an unusable state, where nothing can be done any more.

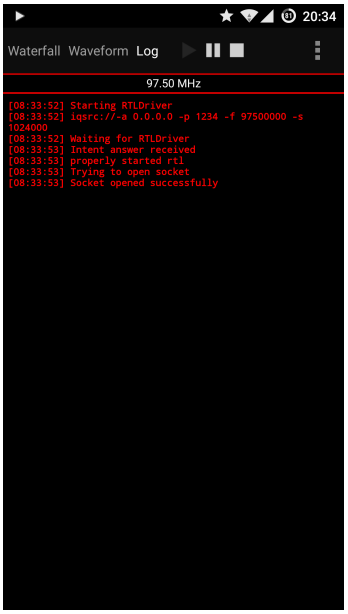
6 https://github.com/rad1o/RFAnalyzer
7 https://code.google.com/p/morsecoder/
8 http://www.gnu.org/licenses/gpl.html



(a) Waterfall



(b) Waveform

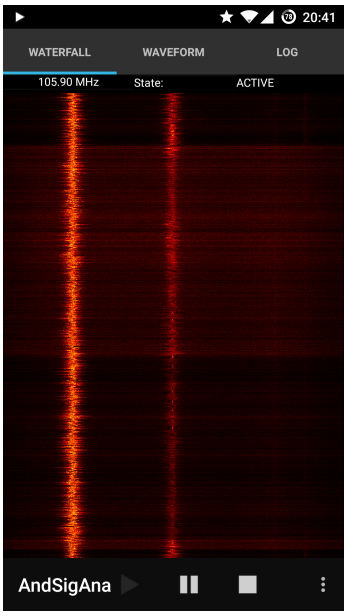


(c) Log

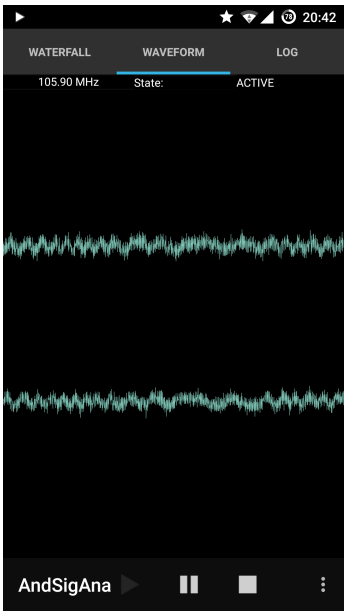
Figure 0.7: WiCat GUI

5.2.2 *WiCat reworked*

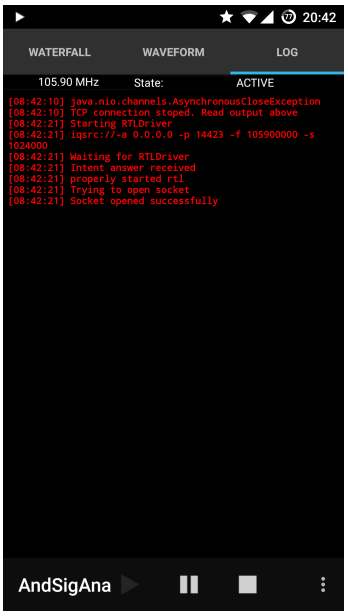
In the reworked WiCat app tabs were added for an easy access to the other screens. Thereby it is possible to swipe to the other screens. The buttons from top were moved in an own toolbar at bottom, so they can reached easy by the thumb.



(a) Waterfall



(b) Waveform



(c) Log

Figure 0.8: WiCat reworked GUI

5.2.3 *RFAnalyzer*

After realizing that the open source RFAnalyzer from Dennis Mantz was worth taking a closer look into, we found out that it was much more melloy than WiCat. It has several demodulations already built in and some other features are integrated as well. The RFAnalyzer has only one main screen which shows the current spectrum and the waterfall view 0.9. RFAnalyzer is missing a waveform view, but it has already some gesture support for better usability. It also has a dedicated activity for the settings. Just as WiCat, RFAnalyzer does not follow the rule of thumb, as all the buttons are at the top of the screen and hardly reachable. Orientation change doesn't kill the app like WiCat, but it shows a black screen with driver restart before going on. Despite the downsides, RFAnalyzer was the better base for the following development.

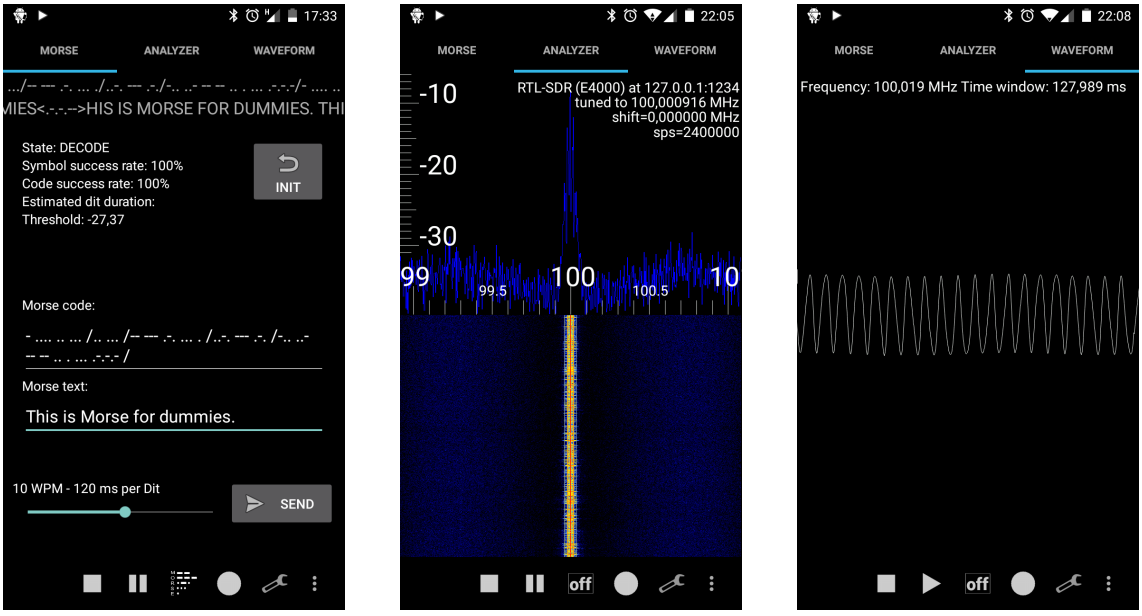
(a) Analyzer

(b) Orientation change

Figure 0.9: RFAnalyzer

5.2.4 *AnSiAn*

For AnSiAn the best parts of the reworked WiCat and RFAnalyzer were composed to a new look. The code written before for the tabs and the toolbar in WiCat was reused. The gestures were reworked, because the swipe support for tabs should not be dropped and new gestures like tapping for frequency changes were added. The toolbar was merged with RFAnalyzers toolbar to support all existing features. Additionally, some extras were added and existing parts completely reworked. The Morse view 0.10 (a) is one of the complete new elements, or an application notification to inform the user, when AnSiAn runs in background. The waveform from WiCat was not reused in cause of complete changed data structures, but it bases on the same open source-project as WiCat did. The settings activity was adapted and there were also tabs added for better seperation. The whole app supports orientation change without loss of information, except the waterfall as this would require a lot more data amount to be permanently calculated and stored without using it most of the time. For more GUI details, please take a look at the manual B.



(a) Morse (b) Analyzer (c) Waveform

Figure 0.10: AnSiAn GUI

5.2.5 *Structure*

The GUI main activity layout consists of three elements. The tab bar using the SlidingTabLayout on top, the MyToolbarView on bottom and a main panel with a ViewPager in the center. The ViewPager is an instance of MyViewPager, which uses the MainActivityPagerAdapter to show the fragments for the different tabs. The figure 0.11 shows a detailed view of the three fragments and their content in the ViewPager. The MorseFragment consists of three views with different tasks. The MorseStatusView displays some information, the MorseSendingView is used for the text input and the playing of Morse sounds and the MorseReceiveView shows the incoming decoded Morse code and symbols. The AnalyzerFragment has an AnalyzerSurface. This surface uses a LayerDrawable to hold all the currently active drawables. The last element in the MyViewPager is the WaveformFragment with its WaveformView.

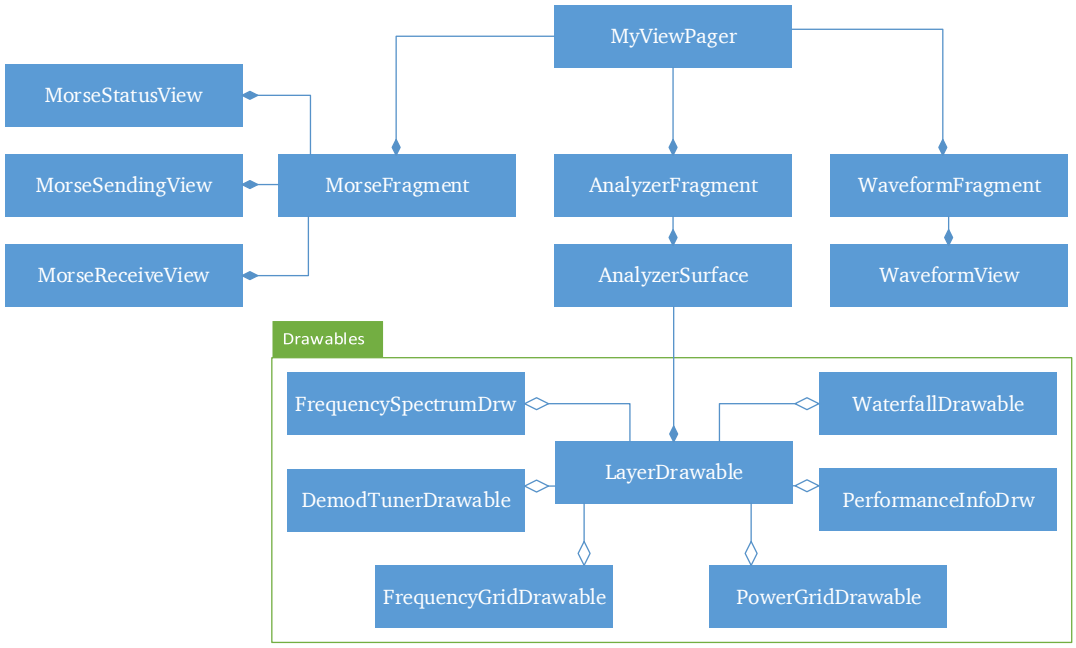


Figure 0.11: GUI classes

5.3 *Structural changes*

To implement all these improvements, many structural changes in the used RFAnalyzer code were necessary.

5.3.1 *MVC*

The base code of RFAnalyzer was not well structured, e.g. many calculations happened in the drawing part. A well-known approach to organize the structure of an application is to build it based on the MVC-pattern. This pattern provides to separate the code into a model, view and controller part. Doing this with the RFAnalyzer code inevitably lead to the division of the classes into smaller ones and the introduction of a lot more packages to organize them.

5.3.2 *StateHandler*

The StateHandler manages the application states. Some tasks are not able to run simultaneously, the StateHandler takes care of the correct handling. The statuses STOPPED, MONITORING, SCANNING and PAUSED are used by the StateHandler to represent the current state. To avoid many substatuses, they are not mutually exclusive, recording and demodulation are handled as simple booleans. All state changes are requested to the StateHandler, which decides if the transition is possible. In most cases, request events are used to accomplish this. If the requesting class needs direct feedback whether a transition was performed or not, the StateHandler announces it by another event.

5.3.3 *Threads & Service*

All threads which should run in background during monitoring are started from the AnsianService, which is managed by the StateHandler. These threads are the Scheduler, FFTCalcThread and Demodulator. Additionally the AnsianService handles the Alarm, which was formerly its own thread, but changed to a normal class after the introduction of the FFTDataEvent.

GUIUPDATETHREAD & FFTCALCTHREAD One essential problem of the RFAnalyzer was that the GUI runs on the same thread as the FFT calculation. So these two completely different tasks were waiting for each other. To solve this, a new GUIUpdateThread was implemented, which is dedicated to draw the surface for the analyzer and the waveform, with a fixed frame rate. The calculation of the FFTs is now done by the FFTCalcThread. With this necessary change now all incoming signal samples are used. With the old single thread only about 25-30% of all incoming signal samples were used for the calculation at a FFT size of 2048.

SOURCECONTROLTHREAD The SourceControlThread is used in the scanner mode to change the tuned center frequency autonomously. The SourceControl starts this thread if the desired bandwidth exceeds the supported maximal bandwidth from the current source. For more information to the scanner mode take a look at 5.4.

5.3.4 *Data structures*

Another key problem was that there were basically no data structures to save both the raw data as also the already calculated data for later reuse. The FFTs was calculated and directly drawn to the surface. For the waterfall only the drawn data was saved to progress it over time. To work on the data later on, complete new models for data handling had to be established. The DataHandler takes care of all new data, holds it and is the interface for all other classes to the data. For a FFT there is a new class FFTSample. These are saved in an ObjectRingBuffer which saves the last calculated ones. The ScannerBuffer is responsible to deliver an organized set of FFTSamples to the GUI in scanner mode. The WaveformBuffer saves a part of the accruing raw or down mixed samples for the waveform. In case of a pause, the DataHandler gets the current data from the buffers and delivers this saved data to the GUI to establish the pause.

5.3.5 *Data input flow*

The IQSource is the interface to the data input source (RTL-SDR, HackRF or file). Figure 0.12 shows the exemplary data input flow for a RTL-SDR device. The RTL-SDR USB driver is is connected to the application via TCP connection. The raw I/Q data are received by the Scheduler, which then again distributes the data to other classes.

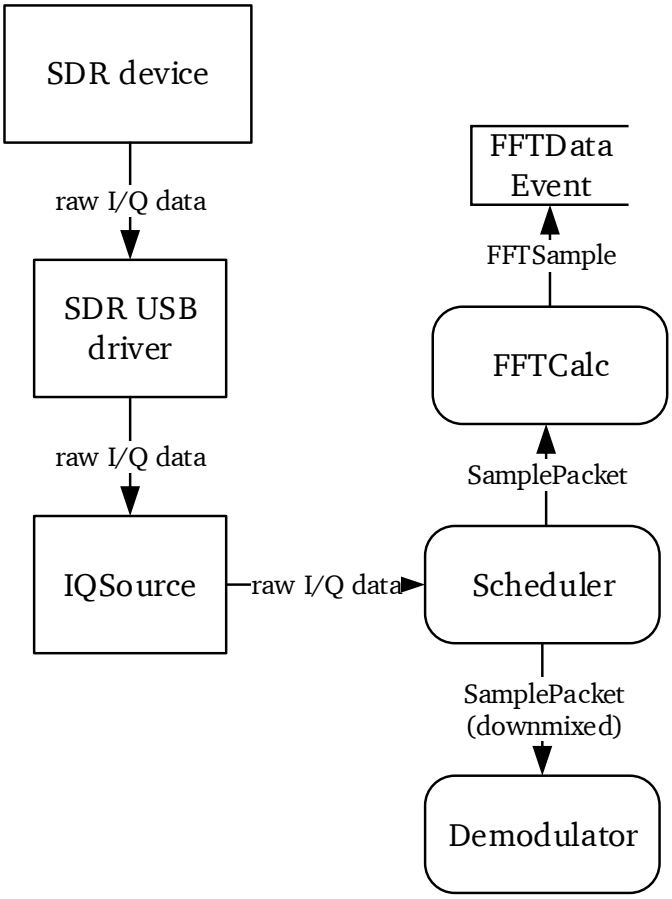


Figure 0.12: Data input flow

5.3.6 *GUI Data adapters*

The data coming from the DataHandler is outsourced from the GUI classes. Different adapters are responsible to calculate a simple data structure to draw. The GUI should remain "stupid" and only draw the given data from the adapters to the surface without further calculation. To achieve this, a GuiFFTDataAdapter to adapt the FFTSample and a WaveformDrawDataAdapter to adapt the samples from the WaveformBuffer for the GUI are used.

5.3.7 *Preferences*

For the preferences, which are displayed by the SettingsActivity, a redesign and separation for better clarity was needed. For the alarm, Morse, colors, GUI and miscellaneous things own preferences were written. These extend MySharedPreferences, which is an extension of the Android SharedPreferences. Every preference has its own preference fragment, which extends MyFragmentPreference. MyFragmentPreference again extends the Android PreferenceFragment, to add some basic functionality for all preference fragments. All these fragments are shown by the SettingsActivity, which has its own SettingsActivityPagerAdapter to arrange the fragments in tabs.

5.4 *Wider band range scanning mod*

The highest practical reliable samplerate to be obtained at once by the RTL2832 chipset is about 2.4 Million samples per second (MSPS), which correlates a bandwidth of 2.4 MHz. The wider band range scanning mode, or shortly called "Scanner" provides the ability to display a wider band range by continuously monitoring neighboring spectra and append them to a greater bandwidth. The more spectra monitored, the longer one pass will take and therefore, the further apart the point of time of the current and the oldest monitored spectra are. One iteration takes about 300 ms. A shorter period resulted in invalid data, as on a frequency change the received packets were falsely denoted with the new center frequency, while they obviously contained data from the previous center frequency. During the pass, old data is simply overwritten by new data. To further avoid unwanted redundant data and maximize the total Scanner bandwidth, neighboring monitored spectra do not overlap. The Scanner is activated automatically when zooming outside the maximum supported at-once-bandwidth and vice versa. Further zooming out and in automatically adjusts the total Scanner bandwidth. Scanning and demodulating are incompatible due to the fact that demodulation requires a permanently tuned demodulation frequency and bandwidth, while the Scanner only remains on a spectrum for a short period of time.

5.5 *Morse*

At first, the Morse recognition algorithm detects the incoming morse codes. Afterwards a morse decoder with the international Morse alphabet (ITU) is used to translate the morse code to plain text. For playing morse sounds a player was implemented and for the receiving sound, the AM demodulation was reused.

5.5.1 *States*

The Morse class uses five states to decode the incoming Morse signal.

STOPPED is the initial state and used when Morse is deactivated.

INIT in this state, after a check which Morse receive mode is used (automatic, semi-automatic or manual), the correct follow state is chosen.

INIT_TIMING is selected if automatic mode is used and the dit duration should be estimated. In this state FFT samples for the statistic are collected.

INIT_STATS follows INIT_TIMING and is active as long as the statistics from the collected FFTs are built.

DECODE semi-automatic and manual mode directly enter decoding, because the dit duration is fixed and MorseStats can be initialized without further information. Or it is entered after the stats are built in automatic mode. In this state the incoming FFT samples are decoded by the MorseStats.

5.5.2 *Automatic determination of dit duration*

This is done in the MorseStats class, which gets a bunch of samples when INIT_STATS is entered, after enough samples are collected during the initialization time. The recognition

of the incoming codes from the signal is done by a simple algorithm which uses the already calculated FFTs. The steps for the automatic recognition are the following:

1. In the initialization time FFTs are collected.
2. From the collected FFTs a threshold is calculated, which is just the middle value from the highest and lowest detected signal strength average over the selected demodulation bandwidth.
3. Using that threshold, the values for each collected FFT can be estimated and saved to a MorseBitSet.
4. Afterwards, the MorseBitSet gets iterated over to build some statistics with the high and low number of consecutive FFT samples. For example three following equal (low or high) samples raise the counter for the entry three.
5. That statistic is used to determine the dit duration. Because high and low dits together have the most occurrence in Morse code, it is just searched for the highest entry in the stats to assume how many samples are one dit.
6. With the information about how many samples one dit are, the number of initialization samples and the initialization time, the dit duration is determined.

5.5.3 *Code recognition and decoding*

In the DECODING state the MorseStats are initialized with a dit duration. Either by the automatic determination or a fixed input value from the user. All incoming FFT samples are processed by the MorseStats, which use the current threshold to decide whether it is a high or low sample and attach this information to the sample. The Morse class now just waits for an ascending or descending edge. If one of them gets detected the time for the low or high duration is calculated by the used timestamps of the samples. With that time the MorseStats can decide for a Morse code through some fixed timing boundaries for dit, dah and word with high or low. Accordingly, as the incoming high or low duration can be classified, the ErrorBitSet gets a new entry. For every incoming new code a MorseCodeEvent is distributed. If the code equals a short gap, the MorseStats uses the Decoder to decode the current collected code since the last gap. With the result a new MorseSymbolEvent is distributed and the symbol ErrorBitSet updated.

5.5.4 *ErrorBitSets*

The ErrorBitSets are used to keep a record about the successful classifications of codes and symbols. These statistics are a simple way to achieve information whether the codes and symbols are recognized at all. The verification can only be done by the user, because parts of the signal can get lost or there may be disturbing signals, which lead to another code than the sent one. But if the codes and symbols can't even be recognized, then the results are wrong. So these ErrorBitSets are used for displaying additional information and for the automatic reinitialization in automatic mode. Currently a fixed threshold of 50% is set for this.

6 PERSPECTIVE

6.1 *New possible features*

As AnSiAn was developed to be expandable, new features are relatively easy to add. The following listing shows some ideas we would find useful.

1. Extending the demodulations. For example, support for DAB radio or ADS-B would be great features.
2. Show information about bandwidths (e.g. highlight and denote if it is known for radio broadcasting).
3. The microphone and the headphone jack could be used as a source.
4. The recording function could be extended to also support the saving of demodulated data instead of only the raw input data.

6.2 *Github*

As the app might be useful for others, we are considering to provide the source code on GitHub. This enables others to extend or improve it independent from the SEEMOO lab/project.

7 CONCLUSION

Because we started the project with developing on base of WiCat, about one month of our development time was spent into improving WiCat. Only few of those changes and experiences were useful for the later base we used, RFAnalyzer. Of course, it would have been much more useful to invest this development time into the improvement of RFAnalyzer directly from the start. Although the RFAnalyzer already had built in some features and functionality we would have had to implement ourselves, this does not mean there was no work left to do. We switched to implement other new features missing in the RFAnalyzer and also, adapting the RFAnalyzer and structural changes meant a lot of work which we would not have thought.
Regarding all this, we definitely underestimated the time consumed by some of the implementations. Although we are glad having accomplished to integrate all the planned features, we know that they are still improvable.

Part II

APPENDIX

A

RELEASE NOTES

The release notes provide some information about the used and tested hardware and compilation/installation instructions.

1 HARDWARE

1.1 *Supported devices*

As the Android device is used as a host, it requires USB-OTG to be supported, which requires at least Android 3.1 (API 12). The project itself is backward-compatible to Android 4.0 (API 14) and thus can be run on about 95% of all Android devices[1]. AnSiAn was tested with the following devices:

- Samsung Galaxy SIII GT-I9300 with Android 5.1.1 (Blisspop custom rom)
- Samsung Galaxy S4 GT-I9505 with Android 5.1.1 (CyanogenMod 12.1)
- Google Nexus 5 with Android 5.1.1 (stock rom)
- Lenovo Yoga Tablet 2-8 with Android 5.0.1 (stock rom)

1.2 *Tuners*

Due to the usage of Martin Marinov's RTL2832U driver which is an Android porting of Osmocom's rtl_tcp support[2], different tuners based on the RTL2832U chipset are supported.[3] The HackRF transceiver and even the new rad1o[4] are supported SDR receivers because an Android HackRF library based on the C-library from Mike Ossmann[5] with a little hack for the rad1o is used.[6] The communication with rad1o worked so far, but it was not further tested.

1.3 *Power supply*

The usage of an DVB-T stick drains a lot of power. We measured for the used CinergyT-Stick RC Rev. 3 a current of 179mA. The usage of the tested Y-OTG cable saved all the power for the stick from the phone, by use of an additional external power supply. If you dont have an Y-OTG cable, a powered usb hub at the phone, can do the same. With this solutions the phone is fully relieved from the stick power drain, because it is only powered by the external power supply or the hub. Charging the phone while using the

1 https://developer.android.com/about/dashboards/index.html#Platform
2 http://sdr.osmocom.org/trac/wiki/rtl-sdr
3 https://www.reddit.com/r/RTLSDR/wiki/compatibility
4 https://events.ccc.de/2015/07/10/rad1o
5 https://github.com/mossmann/hackrf/tree/master/host/libhackrf
6 https://github.com/rad1o/RFAnalyzer

stick, is not always possible. The issue is that it is phone and software dependent, so there can't be an overall solution for all phones. As consequence no further investigation was done on that. But there are some hints already discovered:

1. Try an Y-OTG cable to power your device, while using the stick. Be aware of bad quality OTG cables. Some won't work at all, some restrict the current. It worked out of the box for the Yoga 2-8 tablet with stock rom. The kernel has to support this mode.
2. For phones with induction charging, just try this while using a normal OTG-cable (working for Nexus 5 with 5.1.1 stock rom)
3. If nothing works, use the Y-OTG or a powered USB hub to relieve the phone from the stick power drain.

2 CONTENTS

The complete release will contain the following contents:

- Project AnSiAn
- Android support library project appcompat, needed to compile AnSiAn

3 COMPILATION

We recommend to use Eclipse[7] with the ADT-Plugin[8] and install Android SDK with at least API level 22 to compile the project. First import the projects to compile from the delivered zip file. For AnSiAn also the Android support library package [9] is needed to compile. This package gives is a set of code libraries that provide backward-compatible versions of Android framework APIs. It can also be found in the delivered zip file or in the web with further installation instructions[10].

4 DEPLOYMENT

4.1 *Uninstall*

If necessary, the following command can be issued on the host computer to uninstall AnSiAn: adb uninstall de.tu.darmstadt.seemoo.ansian This was sometimes necessary during development, as the installation of an APK fails when the previously installed version was built on another host.

4.2 *Install*

Three solutions to install the application to the device:

7 https://eclipse.org/
8 http://developer.android.com/tools/sdk/eclipse-adt.html
9 https://developer.android.com/tools/support-library/index.html
10 https://developer.android.com/tools/support-library/setup.html

1. The easiest way to install and run our project is by using Eclipse with the ADT Plugin, if you have already set up everything to compile. Just connect your device per USB and "run" AnSiAn as an Android application. The connected Android device must have debugging enabled in the developer options.

2. Assuming the adb location is stored in the environment variable, the precompiled APK can be installed by using the following command on the host computer: adb install Ansian.apk

3. Just copy the desired apk to your device by your preferred method, for example using and USB cable to hook it up as mass storage device. Then execute it with a file explorer on the device to install it manually. The installation from unknown sources[11] must be allowed.

Additionally the RTL2832U driver[12] has to be installed on the Android device to use the scanner and the DVB-T tuner has to be connected via USB-OTG cable.

11 http://developer.android.com/distribute/tools/open-distribution.html
12 https://play.google.com/store/apps/details?id=marto.rtl_tcp_andro

B

ANSIAN MANUAL

1 OVERVIEW

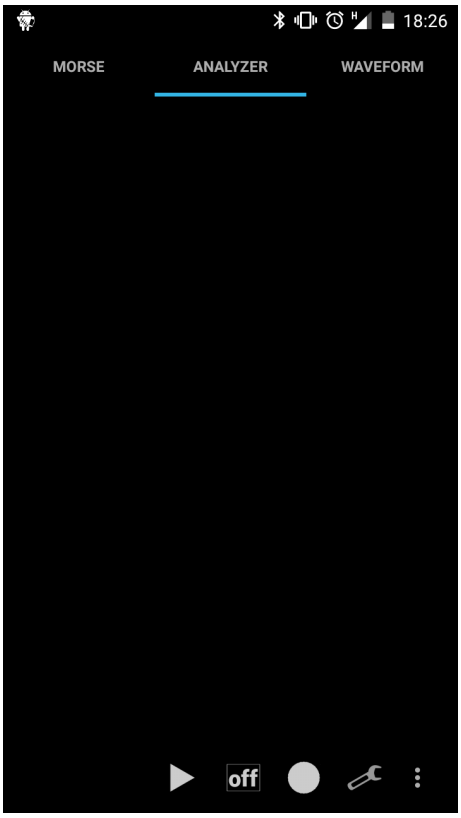


Figure B.1: Main screen

The main screen of AnSiAn consists of three elements, a tab bar at the top, the main screen in the center and a toolbar at the bottom. The main panel has three different views called Morse, Analyzer and Waveform. These views can be swapped by swiping or usage of the tab bar. The toolbar takes care of many functions through different buttons and menus. Note there are several additional settings in the preference area, accessible over associated buttons on the toolbar. AnSiAn also has its own notification B.2 to inform you about the current state and give you the additional possibility to stop the application, while running in background.

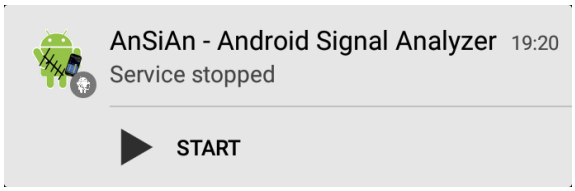
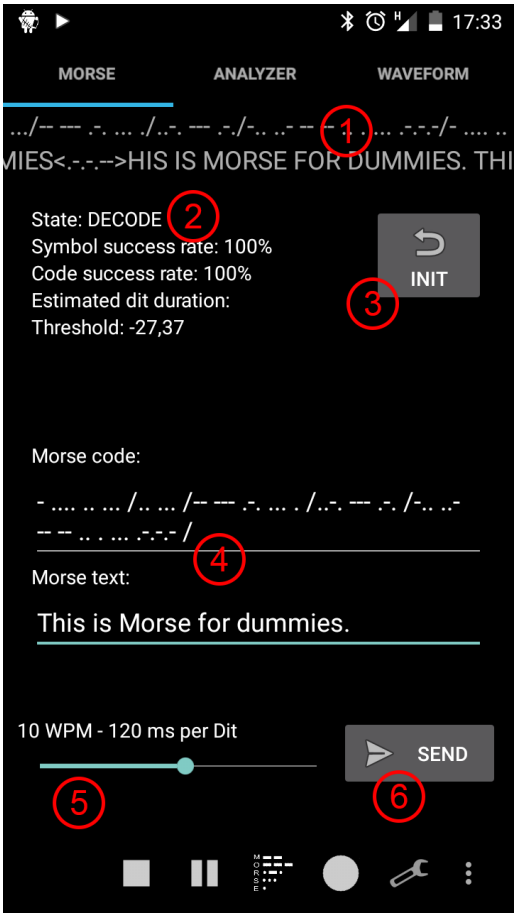


Figure B.2: Notification

2 MAIN PANEL

2.1 *Morse*

This view takes care of all Morse related features. It shows you the received Morse code and decoded text, some information about the current state, it can translate code to text or the other way around and send, respectively play your Morse text.



1. Morse live ticker for the received code
2. Shows you the current state of Morse decoding, the code and symbol recognition rate, what dit duration did the automatic mode estimate and where is the current threshold for a high signal
3. This button starts a new initialization
4. Own Morse code or text can be entered here
5. Drag the bar to the desired sending dit duration and in manual or semi-automatic mode also receiving dit duration
6. Pressing plays the Morse code as audio stream

Figure B.3: Morse view

Note that there are three different Morse receiving modes. For all modes you have to select a proper bandwidth selection of the signal with the demodulation frequency selector B.9. This is used as a base for all modes to calculate the average signal strength over the selected bandwidth of one FFT sample. Be aware that this is a very important step to get good decoding results.

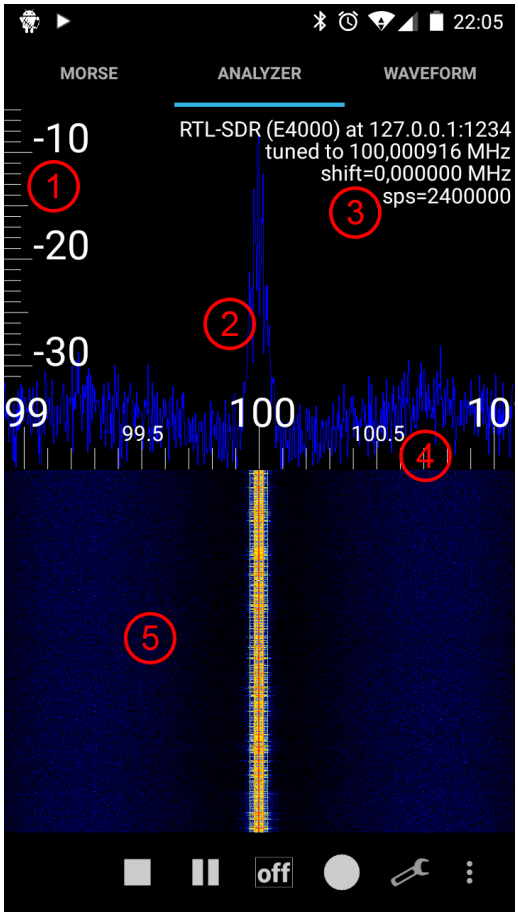
AUTOMATIC does automatic estimation of dit duration and threshold. Therefore an initialization is necessary, which receives some samples and determines the values from it.

SEMI-AUTOMATIC has an fixed dit duration, the threshold is automatically determined. The dit duration is now the same like sending.

MANUAL does nothing on its own. The dit duration for sending and the squelch from the demodulation frequency selector are used.

2.2 *Analyzer*

The Analyzer view shows a spectrum and a waterfall representation of the signal with an interactive GUI.



1. Decibel scale for spectrum and waterfall
2. Spectrum of the incoming signal
3. some additional information, like selected source, tuned center frequency, frequency shift and current samples per second
4. Frequency scale
5. Waterfall, signal over time

Figure B.4: Analyzer view

2.2.1 *Possible interactions*

FREQUENCY The frequency can be easily adjusted by scrolling left and right. Another way is simply to tap on the desired frequency, which becomes the new center frequency.

BANDWIDTH The shown bandwidth can be changed by touching the screen and zoom horizontal in and out. Zooming out more than the source maximum bandwidth, you enter scanner mode automatically. If you are in scanner mode and you tap at a desired frequency, the bandwidth also gets adapted to the maximal source bandwidth.

SIGNAL STRENGTH/GAIN The shown signal strength can be changed by touching the screen and zoom vertical in and out. Note this changes only the shown area of the signal. If you want to change the tuner gain, you have to use the set gain dialog at the toolbar.

2.2.2 *Scanning*

The spectrum has two different modes, scanning B.6 and normal monitoring B.5. As long as the current source can achieve the selected bandwidth it is in normal mode and the tuner is on the fixed center frequency. When the desired bandwidth exceeds the possible bandwidth of the used source, the scanner mode gets active. In this mode the frequency is changed in steps to scan the whole desired bandwidth. In default setting the waterfall is disabled while in scanning mode and the automatic gain is always disabled.

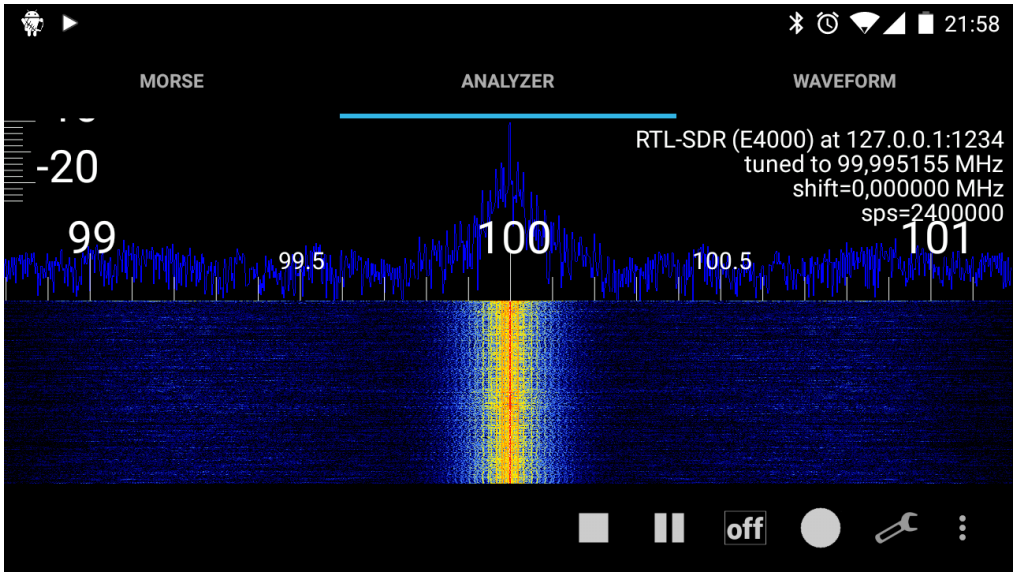


Figure B.5: Normal monitoring mode landscape

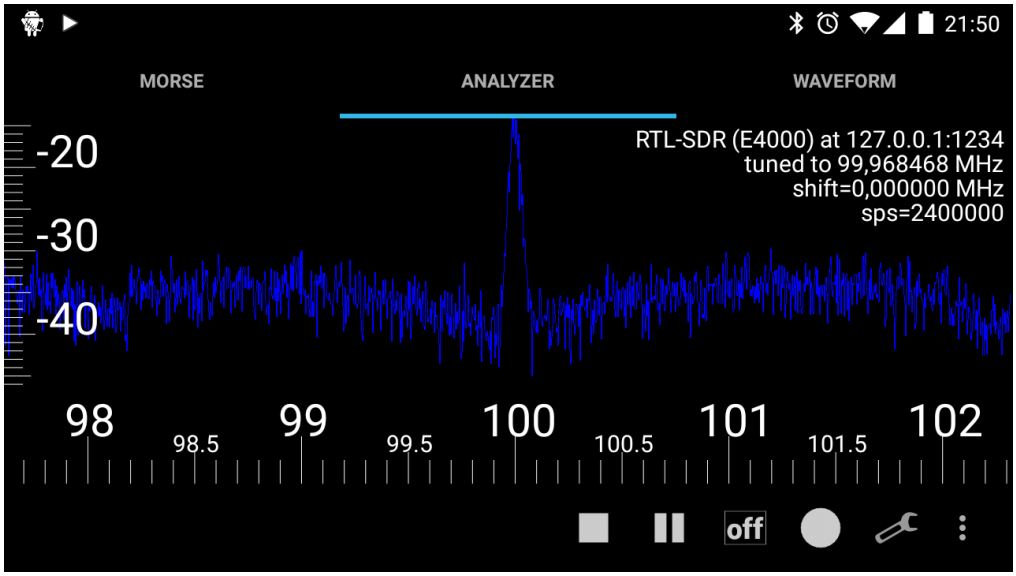


Figure B.6: Scanning mode landscape

2.3 *Waveform*

At the top of the waveform view some information about the signal, like the current frequency and the time window of the currently displayed waveform are shown. If a demodulation is enabled, the down mixed signal is used for the waveform. With no demodulation the incoming unaltered signal (only scaled to display width, but no signal processing) is shown. The waveform can be zoomed in and out. Horizontal zoom increases and decreases the shown time window. Vertical zooming just scales the signal in height.

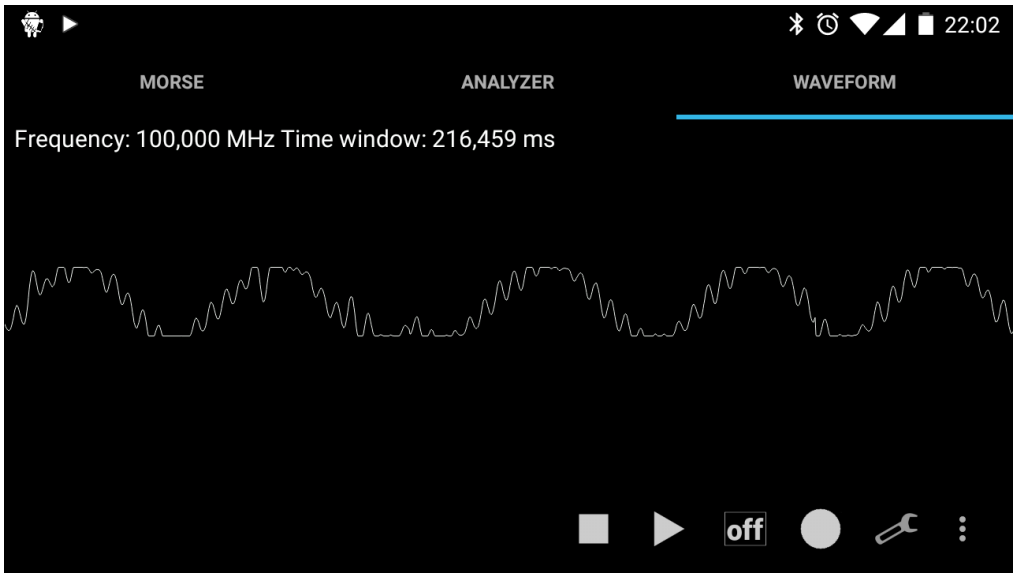


Figure B.7: Scanning mode landscape

3 TOOLBAR

The toolbar holds a bunch of functionality and is accessible in all views, except the settings.

3.1 *Start/Pause/Stop*

3.1.1 *Start*

The start button starts receiving data from the selected source. If you already selected a demodulation, it is also started. In case you are in pause, it will just unpause the GUI.

3.1.2 *Pause*

The pause button only appears while the application appears to the user as running. It will only pause the analyzer and waveform GUI. Things like demodulation and data receiving from source run in background. So the user is still able to zoom in and out of the waveform or spectrum for better signal analysis. In this case the GUI will be updated with the saved raw data, which will be recalculated for the new dimensions appropriately.

3.1.3 *Stop*

The stop button really stops the application and all background processes. If you want to close AnSiAn complete please use it before you leave the application. But if you forget to stop, you will be aware that AnSiAn still runs in background, because you have the notification.

3.2 *Demodulation*

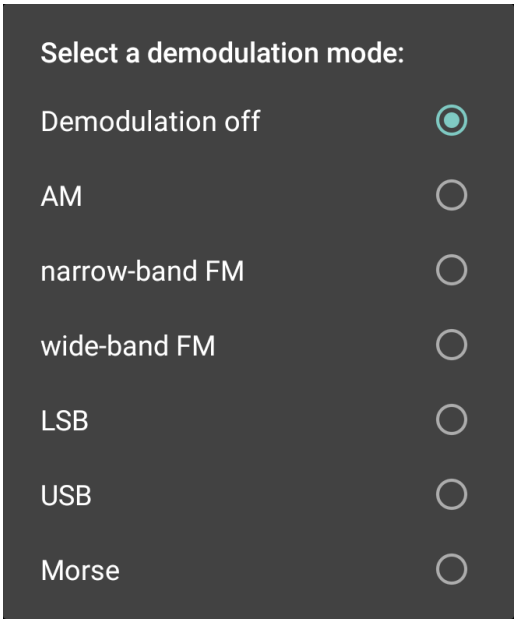


Figure B.8: Demodulation selection dialog

DEMODULATION OFF no demodulation

AM Amplitude demodulation with bandwidth between 3-15 kHz

NARROW-BAND FM Frequency demodulation with bandwidth between 3-15 kHz

WIDE-BAND FM Frequency demodulation with bandwidth between 50-120 kHz

LSB Lower sideband with bandwidth between 1500-5000 Hz

USB Upper sideband with bandwidth between 1500-5000 Hz

MORSE Enables the Morse receiving, decoding and AM demodulation to hear the signal as default (AM can be deactivated in settings)

After selection of a demodulation, an extra frequency selector for the demodulation shows up in the spectrum view. With that selector the demodulation frequency can be adjusted. Note that taps now change the position of the selector and don't affect the tuned source frequency any more. To change the demodulation bandwidth the selector width can be adjusted by horizontal scaling. The squelch defines a threshold for the demodulation. Demodulation is only active as long as the squelch is satisfied. And the squelch is satisfied as long as the average signal strength of the selected demodulation bandwidth is greater than the squelch.

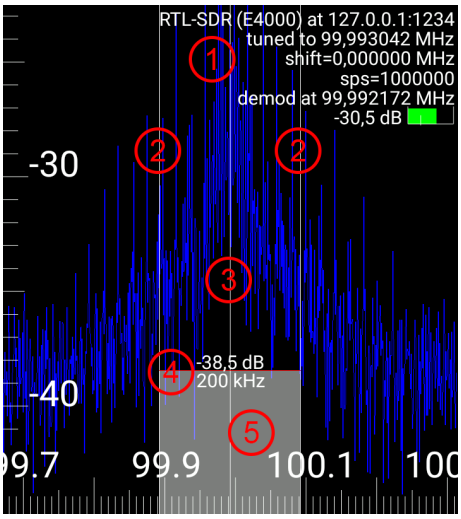


Figure B.9: Demodulation frequency selector

1. Additional demodulation information: center frequency, current average signal strength for demodulation area and squelch bar (green if satisfied, red if not satisfied)
2. Left and right bandwidth border
3. The demodulation center frequency
4. Squelch threshold and demodulation bandwidth
5. The grayed out area is under the squelch threshold

3.3 *Recording*

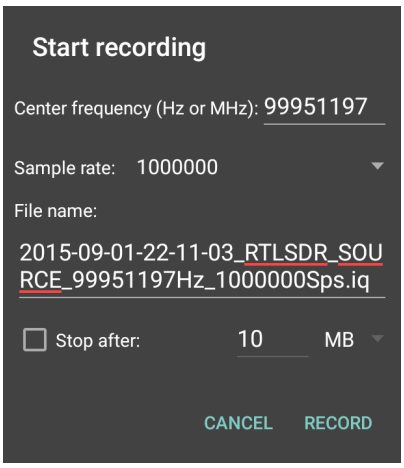


Figure B.10: Recording dialog

In the recording dialog you can adjust the frequency and sample rate to record. You can also change the file name, but it is recommended to use the suggested because if you want to use the file as source for AnSiAn some settings are taken from the file name. If you select stop after the recording will stop automatically after the desired size or time.

3.4 *Gain*

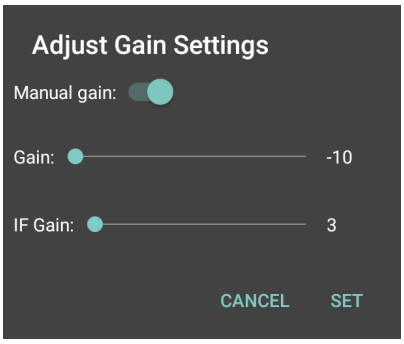


Figure B.11: Gain dialog

The gain dialog gives you control over the source gains. You have to activate the manual gain mode to change the gains, otherwise the automatic gain of the tuner is used. The IF gain influences the intermediate frequency stage in the tuner.

3.5 *Frequency*

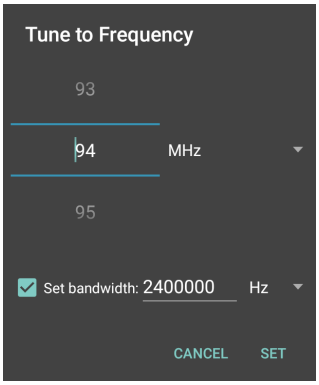


Figure B.12: Frequency dialog

The frequency dialog allows you to change the frequency and bandwidth. The frequency picker is limited to the available frequencies of the source. Pay attention to the used unit.

4 SETTINGS

The settings can be found in an additional activity with some tabs for better structuring and usability.

4.1 *Misc(ellaneous)*

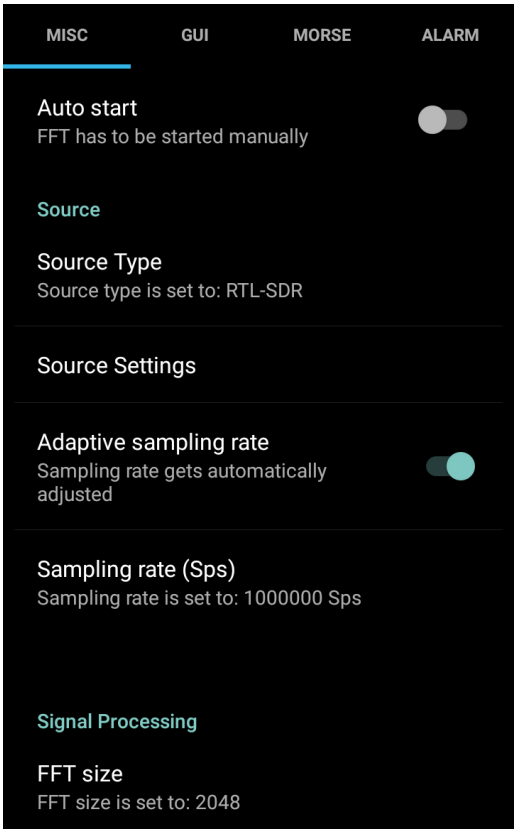


Figure B.13: Misc settings

1. Starts the monitoring and signal processing with application start
2. Choose desired source
3. Additional source settings
4. Automatic sample rate adjustment
5. Fixed sample rate, when adaptive sample rate is deactivated
6. The FFT size defines the resolution used for dividing the window

4.2 *GUI*

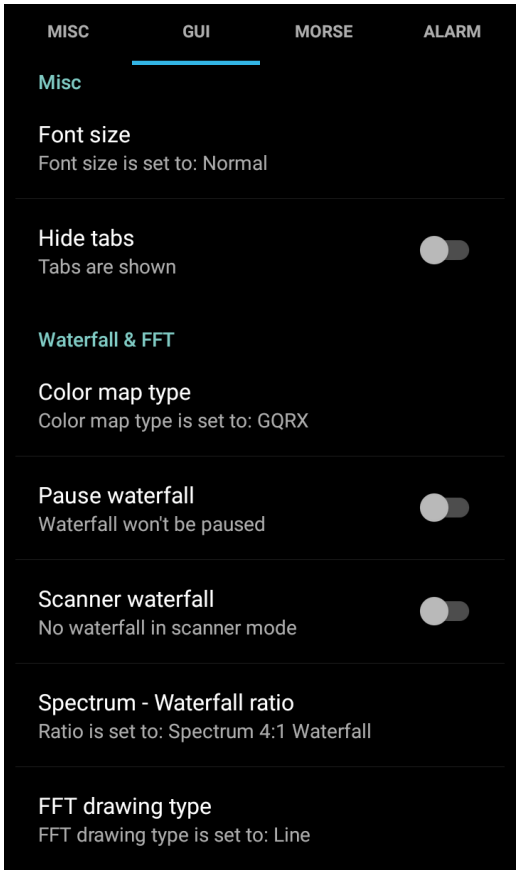


Figure B.14: GUI settings

1. Font size used for all text in analyzer and waveform
2. Can be used to hide the tab bar on top to achieve a bigger screen for analysis
3. Color map for the waterfall
4. Activate to pause the waterfall too, re-drawing needs some time if you scroll or zoom in
5. The waterfall can also be activated in scanner mode
6. Adjust the ratio between waterfall and spectrum to your needs
7. The drawing type for the FFTs in the spectrum view

4.3 *Morse*

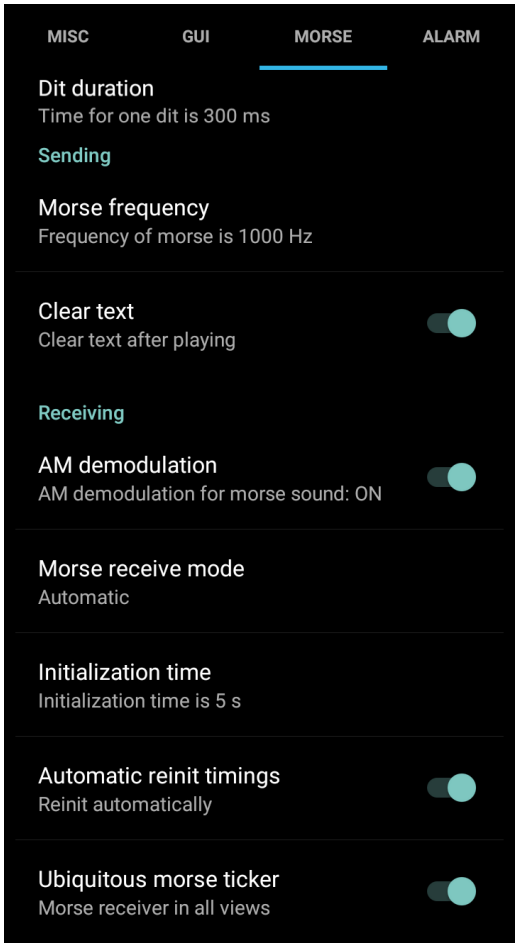


Figure B.15: Morse settings

1. The dit duration used for sending and manual or semi-automatic is also used for receiving
2. Frequency of the Morse sound for playing/sending
3. Should the text and code you entered be cleared after playing the sound
4. Decide whether AM demodulation should be activated while Morse decoding
5. The selected Morse receive mode 2.1
6. Initialization time for the automatic morse mode, if you select the time too short in comparison to the dit length, the result will be a wrong estimation for dit time
7. If selected there will be an automatic reinitialization, when the code and symbol success rate are bad
8. Activate a ubiquitous Morse receiver for all main panels

4.4 *Alarm*

The alarm settings are quite important, because it is the only way to use this feature. It enables the possibility to play a sound or perform a vibration, if the signal strength of the current center frequency of the spectrum exceeds the set alarm threshold.

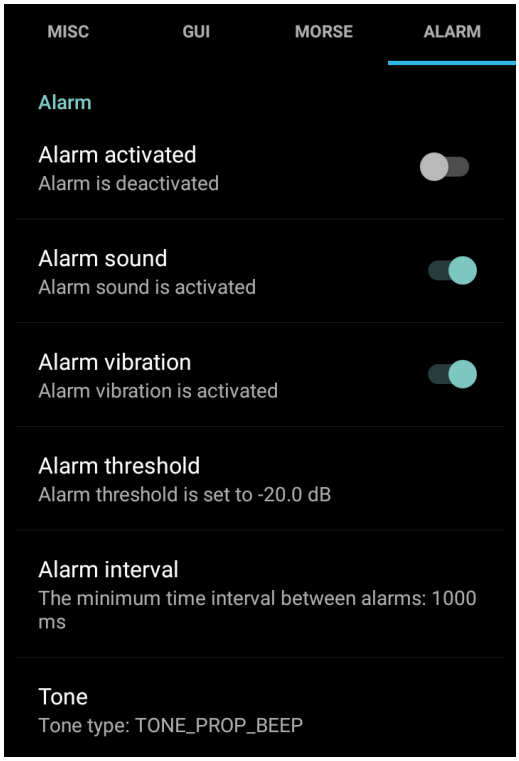


Figure B.16: Alarm settings

1. Activates the alarm
2. Should a sound be played at alert
3. Should the device vibrate at alert
4. Set the threshold for the alert
5. Set a minimum interval between two alarms in ms
6. Choose an alarm tone type, these are some Android default tones

C

SPENT TIME BREAKDOWN

Results

Total spent effort: 757h 27min

AnSiAn - Android Signal Analyzer 757h 27min

Secure Mobile Networking Project SS2015 757h 27min

Initial project setup 38h

Direct spent effort	38h

Project finalization 91h

Stories	91h
Project documentation final	91h

Sprint 1 101h 13min

Stories	101h 13min
Rework of existing WiCat GUI	10h 48min
Setup management tool	11h 30min
Structural changes in WiCat to fit MVC and expandability	21h 14min
Estimate solutions for wider band scanning	17h
Estimate possible performance improvements	5h
Maintain Agilefant Sprint 1	11h 18min
Project documentation	13h
Weekly Scrums Sprint 1	9h 23min
Charging phone while using OTG	2h

Sprint 2

Stories 77h

Rework and integration to RFanalyzer GUI	38h
Project documentation alpha	7h
Maintain Agilefant Sprint 2	2h
Application should also run in background	5h
Structural changes in RFAnalyzer to fit MVC and expandability	9h
Weekly Scrums Sprint 2	6h
Research expansion of frequency range	5h
Wider band range scanning Sprint 2	5h

Sprint 3 158h 44min

Stories 158h 44min

Maintain Agilefant Sprint 3	6h
Weekly Scrums Sprint 3	5h
Project documentation beta	10h
Make wider band range scanning posssible	5h
Further structural work	50h 2min
Demodulate, decode and encode morse	16h
Finish Ansian GUI	52h 15min
Alarm trigger	4h 12min
Increase performance	10h 15min

Sprint 4 291h 30min

Stories 291h 30min

Finalize GUI	72h
Fix Bugs, missing features	187h
Waveform view	20h 30min
Code cleaning & commenting	6h
Maintain Agilefant Sprint 4	2h
Weekly Scrums Sprint 4	4h