TAMPERE UNIVERSITY OF TECHNOLOGY

MICHAL JAKUBIAK

CELLULAR NETWORK COVERAGE ANALYSIS USING UAV AND SDR

Master of Science thesis

Examiners: Prof. Yevgeni Koucheryavy,
PhD Dmitri Moltchanov
Examiners and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 8th October 2014

ABSTRACT

MICHAL JAKUBIAK: Cellular network coverage analysis using UAV and SDR
Tampere University of Technology
Master of Science thesis, 48 pages
24 November 2014
Master's Degree Programme in Information Technology
Major: Communications Engineering
Examiners: Prof. Yevgeni Koucheryavy, PhD Dmitri Moltchanov
Keywords: gnuradio sdr uav drone gsm cellular

Drones and Unmanned Aerial Vehicles (UAVs) are being given more and more attention. Drone research focuses on autonomous capabilities, flight dynamics and gathering data with specific sensors. However, there is very little focus on drones carrying radio interfaces for other purposes than communicating with a ground station and it's surprising because of of ubiquitous characteristics of both of these technologies.

This project is about equipping a quadcopter drone with a Software Defined Radio and a small, single-board computer to process the signal and location data. The ultimate goal is to create a modular platform for different, task-specific, UAVs and software capable logging and processing the RF data. As an intermediate point for that goal and the purpose of this thesis, we've created a system that can easily be used to fly around an area and log the power of a GSM pilot carrier. The data is logged together with the precise geographical location so that the data from a logfile can be used to create an easy to read heatmap of the carrier's coverage.

PREFACE

The pages I present here mark as a final milestone on a lengthy journey. A journey that wouldn't be possible without contributions and support from so many people. I would like to express my gratitude to Dmitri Moltchanov and Yevgeny Koucheryavy without whose support none of this would ever see daylight.

Words of appreciation go to the open-source communities behind GNURadio and DIYdrones who not only proved to be a lot of help, but also laid foundations of my work. Special mentions are deserved to Nadir who inspired me to come up with my own topic and Maciek for giving me the shortest and most effective pep talk without even realizing it.

Saving the best for the last - I want thank Tiziana for her unconditional love and support. This thesis would never complete without you putting up with me throughout the ups and downs of creating it.

Michal Jakubiak
Tampere, 20.11.2014

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

UAV	Unmanned Aerial Vehicle
SDR	Software Defined Radio
TUT	Tampere University of Technology
Li-Po	Lithium-Polymer
APM	Ardupilot Mega
PX4	Pixhawk
sps	samples per second
BBB	BeagleBone Black
RTF	Ready-to-Fly
3DR	3D Robotics
WSN	Wireless Sensor Networks
DVB-T	Digital Video Broadcast - Terrestial
UAS	Unmanned Aerial Systems
GSM	Global System for Mobile Communications
WASP	Wireless Aerial Surveillance Platform
USB	Universal Serial Bus
FMU	Flight Management Unit
RTOS	Real Time operating System
OOT Module	Out Of Tree Module
IBSS	Independant Basic Service Set
CLI	Command Line Interface
GPL	General Public License
GNU	GNU is not Unix (recursive acronym)
USRP	Universal Software Radio Peripheral
SWIG	Simplified Wrapper and Interface Generator
GRC	GNURadio Companion
TDMA	Time Division Multiple Access
FDMA	Frequency Division Multiple Access
SDMA	Space division Multiple Access
MS	Mobile Station
ppm	parts per million
GMSK	Gaussian Minimum Shift Keying
NRZ	Non-Return to Zero

1. INTRODUCTION

1.1 Unmanned Aerial Vehicles

Until recently, UAVs (Unmanned Aerial Vehicles) have had a rather fearful reputation for being used as 'sky assassins' and 'spies' of the military. It was a well-deserved reputation because, like with many other technology, the military with it's non-profit oriented budgeting model is a early adopter. For the first part of the 2010s, every time something bad happens around the world, the news would greet us with headlines having words 'drone' and 'air-strikes' next to each other.

However, the drone technology has advanced beyond the domain of multi-million dollar military sponsored projects. Civilian applications are more and more common. A whole new class of drones has evolved over the last decade - the Micro Air Vehicles (MAV). it encompasses a whole range of miniature, flying vehicles that have been around for a while now. However, but there is one section of it that's a fresh blow and is commonly referred as 'drones' - the multi rotors. These devices use 3 or more, vertically aligned brushless engines and movement is achieved by creating a difference in thrust on motors on the opposite sides of the frame resulting the MAV tilting and creating sideways acceleration 1.1. The thrust regulation needs to be very precise and can't be done without a micro-controller. To perform the most basic of operations, the microcontroller needs some basic sensors like a gyroscope and accelerometer.

The first quadcopters were built around Arduino platform, so they already had a surplus of processing power and compatible with many more sensors than just the essentials. From here on, achieving autonomous flight was just a matter of adding additional sensors and programming the software to use them. This trend still continues today and as the sensors grow in numbers, so does the processing power needed to make effective use of them. As of today, widely available drones use dedicated Flight Management Units (FMUs) that come with all the basic sensors integrated on one board and a further support for the most common sensor buses like CAN or I2C.

Figure 1.1 *Quadrotor principle of opertaion*

Drones are already commonly used for crop management, wildlife protection, 3D mapping, search and rescue operations, archaeological sites surveillance, crowd monitoring, police investigations, Amazon prime air, drug delivery in remote regions. This list just keeps on growing. According to a report recently published by Aerospace Industries Association UAVs are projected to contribute 89 billion USD to GDP over the next 10 years in US alone[4].

Drones carry a variety of sensors. Part of them is used to gather data about drone's surroundings and there are sensors that collect useful data. By far, the most popular one is a simple camera. Aerial photography provides a lot of useful information and using a done is very cheap compared to a satellite or a helicopter. There is, however, an area that hadn't been given the attention it deserves - the Radio Frequency domain.

There are very little projects that would make use of RF-enabled drones for communication purposes. We will take a look at the most notable examples in the next chapter.

1.2 Software Defined Radio

Software Defined Radio has been around for a few decades now. Similarly to drones, it started as a large-budget military project and slowly made way into our homes. SDR redefines the way we think of radio. The hardware of an SDR is similar to traditional radio, except at the moment when we get the signal down to baseband, it's being sampled by and ADC converter and the components found further down the datastream are replaced by a programmable Digital Signal Processor (DSP). Tasks traditionally done in hardware like filtering, modulation, coding or equalization are

now performed by a DSP. With the passage of time and increase in processing power, this technology is slowly making entrance to consumer electronics. It's not uncommon to see one DSP GSM/UMTS/LTE baseband processor in mobile phones [17]. However, it's only in recent years when truly open-source PC-based SDRs are outshining obscure, proprietary undocumented DSP chips.

1.3 Task statement: combining UAV and SDR

Omnipresent wireless networks don't surprise anyone these days. If anything - it's the lack of such that causes both surprise and customer dissatisfaction. Maintaining coverage over large areas is a cumbersome task. RF equipped drones can be a valuable tool when dealing with wireless networks. Such device could actively search for a signal or a source of interference. The possible application list extends when we consider not only a receiver, but also a transmitter, thus allowing for two way communications. A drone could act as a mobile sink collecting data from Wireless Sensor Networks (WSNs) or temporarily replace a broken node. As it often happens with new technologies, they also pose whole new layer of security issues as they can be used for less than legal activities like remotely jamming wireless signals or performing untraceable man in the middle attacks. Combining these two technologies opens a world of new, yet untapped research opportunities.

Because this is a relatively new area, in this thesis we tackle the simplest case scenario where we want to gather data about the coverage of a cellular network. Commercially available quadcopters are already designed to have onboard space and the necessary power surplus to carry payloads. In our case, the payload will be a Single Board Computer (SBC) and an SDR hardware. Both SDR and FMU are connected to the SBC via USB. The task of the SBC is to:

- access the positioning data from the FMU
- access the signal data sampled by the SDR radio
- create a log consisting of a series of geographical location and the power of the signal at that location

After performing the measurements, we want to convert the results to a human-friendly form, by display them on a Google maps layer.

2. BACKGROUND INFORMATION

Just as Telecommunication is a multidisciplinary domain encompassing electrical engineering, computer science, signal processing and many more so does this project. With a little bit of robotics, software engineering and lots of different hardware it's a hodgepodge of technologies. In this chapter we want to explore some of them to have a clearer picture of how all the parts of hardware and software operate together.

2.1 GSM

One of the key characteristic why GSM was selected as a target to be measures was the very narrow bandwidth it uses relative to other widely used wireless standards. A GSM frequency channel occupies only 200kHz. Table 2.1 shows a brief comparison of bandwidth used by some wireless technologies. It is noteworthy that the bandwidths tend to increase as new technologies emerge. Large bandwidths pose a problem for SDRs because the hardware required to capture them grows more complex and more expensive.

Higher bandwidths also cause a problem due to the limitations of the interface connecting the SDR to the computer. Following the Nyquist sampling theorem, the sample rate should be twice the highest frequency of the analog signal. Table 2.1 lists common wireless technologies and the minimum throughput required to capture them. Last, but not least - these huge amounts of data require hardware capable of processing it, especially since most radio technologies are time-critical and the processing has to be done in real-time.

More bandwidth directly translates into more cost in terms of transfer rates, processing power and money. But that is not all. Achieving mobility puts constraints on size, weight and power consumption. Having all that in mind, GSM with it's narrow channel bandwidth was an ideal technology to tackle first.

The full GSM specifications with all the extensions cover roughly about 1000 pages. In this chapter we will go through a small section of the air interface which is relevant to the work presented in this thesis.

	Bandwidth	Required throughput @ 12 bits/samp
GSM	200 kHz	4.8 Mbit/s
WCDMA	5 MHz	120 Mbit/s
LTE	up to 20 MHz	480 Mbit/s
802.11a/b/g	20 MHz	480 Mbit/s
802.11ac	up to 160 MHz	3840 Mbit/s
802.11ad	2.16 GHz	51.84 Gbit/s
802.16	up to 20 MHz	480 Mbit/s

Table 2.1 *Comparison of wireless technologies in terms of channel bandwidth*

2.1.1 Multiple Access

GSM utilizes a combination of Time, Space and Frequency Division Multiple Access techniques (TDMA, SDMA and FDMA respectively). First, the 900 or 1800 MHz band is split to 200kHz channels (FDMA) that operate independently of each other. SDMA is achieved by distributing these frequencies among Base Transmission Stations (BTS) in a way that adjacent cells don't use the same frequency in order to avoid interference at the cell's edge. Furthermore, each 200kHz carrier is split in time into 8 slots. This combination of TDMA and FDMA is illustrated in figure 2.1. When initiating a connection, the Mobile Station (MS) will have an uplink and downlink carrier/timeslot assigned to it.

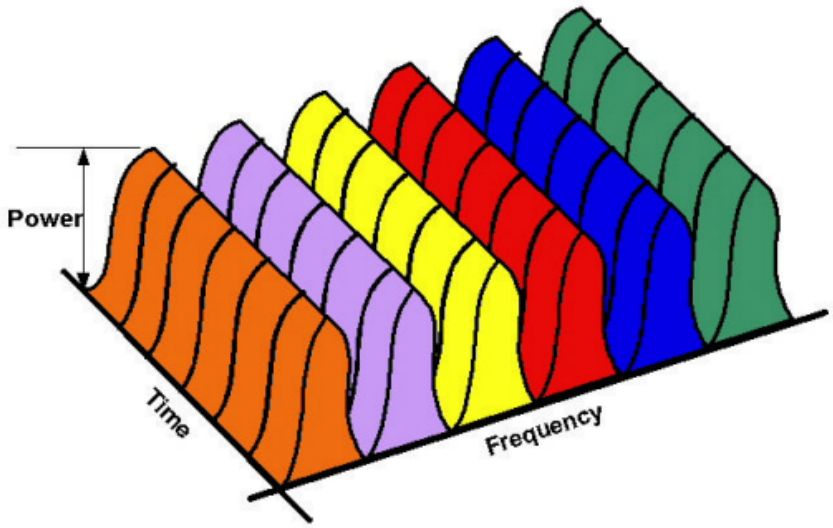


Figure 2.1 *visualization of FDMA and TDMA combination*

2.1.2 Pilot signal

Figure 2.1 depicts a multiple access scheme used in GSM. However to make the illustration clearer, it assumes that all the frequencies and timeslots have a signal present. This is rarely the case as networks are designed to be able to handle traffic with very low blocking probability during rush hours. Outside the rush hours, most of the channels remain unused.

Every BTS has one frequency channel in the downlink portion of the spectrum where the signal is constantly present. This frequency is referred to as 'pilot', 'broadcast' or 'carrier'. When the MS powers on, it's first task is to locate the broadcast frequency and tune into it. The broadcast channel supplies the MS with all the information about the BTS necessary to join and maintain the network connection; starting from synchronization in frequency and time domain, ending at authentication and encryption details.

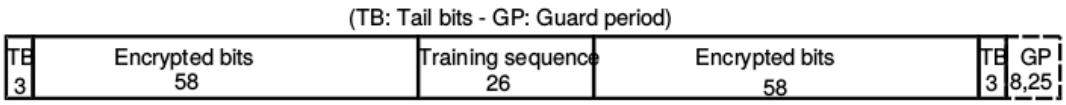
Because the pilot signal is constantly transmitting, it is the frequency channel that our SDR needs to measure to provide us with valuable data to determine the link quality.

2.1.3 Bursts

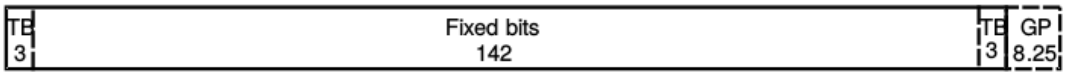
A single carrier frequency is sub-divided into 8 time slots that are shared between users. Each timeslot lasts exactly 3/5200 seconds (roughly 576.7 μs). Timeslots are left unused when there are no users to utilize them. However, when a transmission occurs, the slot is filled with a burst. Bursts are shorter than slots by roughly 8.25 bits of guard period to account for spread of MS in space because singals from MS further away from the BTS take longer to arrive.

GSM is a standard with a long history. The first commercial networks date back to early '90s. Since then, many amendments and improvements have been made to keep it up to date. In practice, the waveforms we observe today vary very much from the ones in the early implementations. As a result, there are many types of bursts. We would like to focus only on the ones relevant to the work done and most commonly present in a broadcast channel. These would be Frequency, Normal and Synchronization bursts. Their structure can be seen on Figure 2.2.

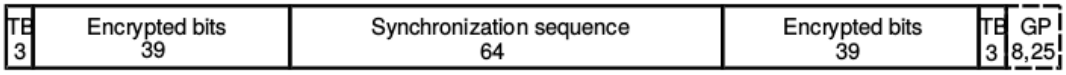
As the name suggest, Normal bursts are the most commonly seen in GSM. They are used to carry user data. Both the Frequency and Synchronization bursts are found only in the downlink of the carrier signal and are used to achieve synchronization in



(a) Normal burst



(b) Frequency burst



(c) Synchronization burst

Figure 2.2 *different types of GSM bursts[2]*

frequency and time, respectively. Figure **??** presents a waveform of a burst found in the broadcast channel captured using RTL-SDR.

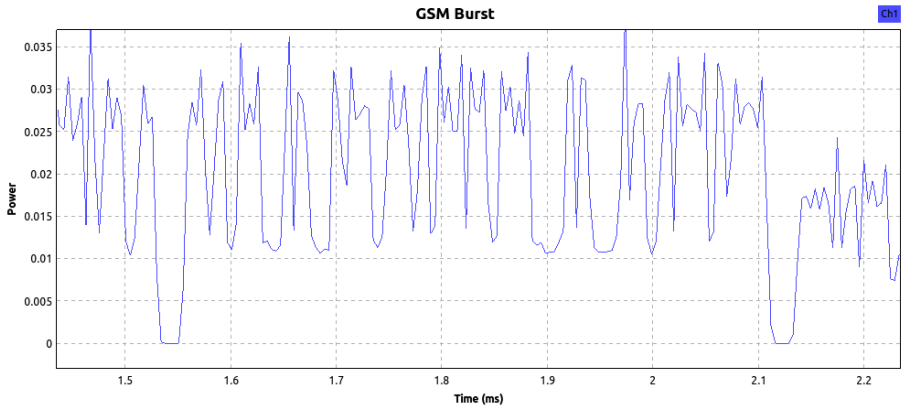


Figure 2.3 *GSM burst seen under an oscilloscope*

2.1.4 Modulation

Contrary to what many sources cite, GMSK (Gaussian Minimum Shift Keying) is not the modulation scheme for GSM. It is the original scheme that GSM standard started with, but a captured burst can be modulated using any of the following: GMSK, AQPSK, QPSK, 8-PSK, 16QAM or 32QAM depending on the type of channel[1]. However, for the purpose of backwards compatibility with the earlier standards, the carrier is GMSK modulated and we will focus on that.

GMSK is a variation of MSK where the NRZ (non-return to zero) encoded signal is passed through a Gaussian filter. The advantage of this technique is improved spectral efficiency at the cost in increasing ISI (Inter-Symbol Interference). This was a very appealing trade-off as frequency licensing is very expensive.

2.2 Software Defined Radio

Since the dawn of wireless communications radios were fully implemented in hardware. There was a good reason for it - for most of that time the term 'software' remained undefined and even after that was overcome we didn't have enough power at our disposal to process even the simplest of signals.

Unsurprisingly, the driving force behind the first working implementation of an SDR was the US military. During the Gulf War, it became apparent that the current communication system had it's shortcomings. Air force, ground forces, the navy and military satellites all operated using different radio technologies. There were a total of 15 mutually exclusive wireless technologies in use and thus making communication between them difficult. At the time, long-range transceivers were bulky and simply carrying 15 different radios wasn't a viable option. This is how the SPEAKeasy program came to life [15]. It's result was a reconfigurable radio that could operate from 2MHz to 2GHz and could communicate with all the wireless technologies employed by the military at the time.

The idea behind SPEAKeasy was to split the traditional all-hardware radio design into two parts. Hardware part was reduced to a minimum and used to transmit/receive the signal and perform A/D, D/A conversions (Digital to Analog and Analog to Digital). Traditionally hardware tasks such as modulation, channel coding and source coding were left to a multi-purpose, easily reconfigurable DSP processor. SPEAKEasy was the first working implementation of an SDR.

More than 2 decades later SDR hardware is commercially available for as low as 20USD for receiver-only solutions.

As of now, smart devices have separate radios for every technology, but only one is in use at a time. This is a waste of resources. Commonly available, cheap SDR hardware opens the doors to the future of wireless communications - Cognitive radio. Cognitive transceivers are SDR radios that are aware of the changing medium properties and can adapt to it by adapting it's parameters such as the modulation scheme, bandwidth or symbol duration to provide the best possible quality of service.

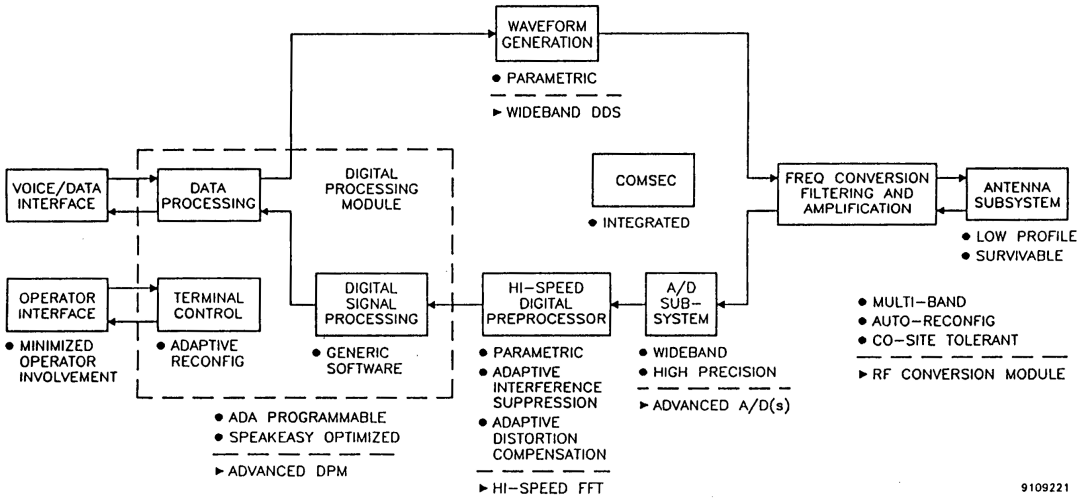


***Figure** 2.4 SPEAKeasy functional block diagram[15]*

2.3 Drones in Telecommunications

Today, drones are at peak of their development. It's the hot, new technology at the 'rapid improvement' phase of the S-curve of technology development cycle (Fig. 2.5). It's the time when giants of the industry like Google or Amazon are heavily investing in research and development to create a viable commercial model. As improvements are made, drones are becoming more agile, autonomous, power efficient and safe. As a consequence of that - also more and more popular. Not so long ago we had to order the drone used for research from the USA. Today, the local electronics store sells lightweight quadcopters as toys.

UAV usage in telecommunications is a huge area for innovation and, as the following pages will show, it's not exclusive to big-budget military or commercial deployments.

In this chapter we would like to have a glimpse at how drones are making a slow, yet steady entry into the world of telecommunications. Even over the period of the work done on this thesis, a lot of innovations have been made.

2.3.1 WASP

WASP stands for Wireless Aerial Surveillance Platform. It was a side hobby for two security engineers: M. Tessy and R. Perkins. The project was showcased at DEFCON 19, a hacking conference in 2011 [14]. They have re-purposed a fuselage of an old military target practice drone to host an SDR and an Single Board Computer

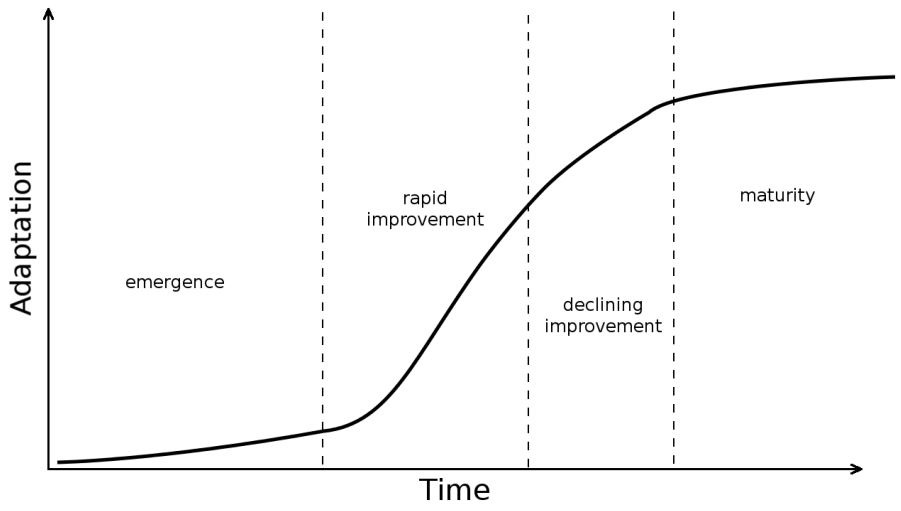


Figure 2.5 *S-curve of technology adaptation*

(SBC) running Backtrack - a security oriented Linux distribution. It is a hacking UAV designed to exploit security vulnerabilities in wireless systems. The drone has also a 4G internet connection to offload some of computation-heavy work such as brute-force password cracking onto a powerful backend computer. WASP can target Wi-Fi, Bluetooth and GSM networks for both cracking and impersonating a trusted internet access network to intercept transferred data.

The project utilizes open-source and off-the-shelf components with a strong focus on system integration rather than component design. Because it's privately funded by two engineers, there was an emphasis on the costs. The components of the entire system cost roughly 6190USD. While it may sound like much on a household budget, it is scant compared to military or corporate funded projects of similar nature.

2.3.2 Colibrex

Colibrex is a subsidiary of a German telecommunications consulting company - LST-elecom. It offers a wide range of services regarding airborne RF measurements using UAVs [7]. The drones carry high-end RF measuring equipment and HD cameras to perform both radiation pattern measurements as well as mast inspections and auditing. The company also provides a consulting background to analyse the gathered data and use it to optimize the antennas for maximum coverage and correct any possible installation faults.

The exact specifications, capabilities and hardware used by the drones are not publicly disclosed.

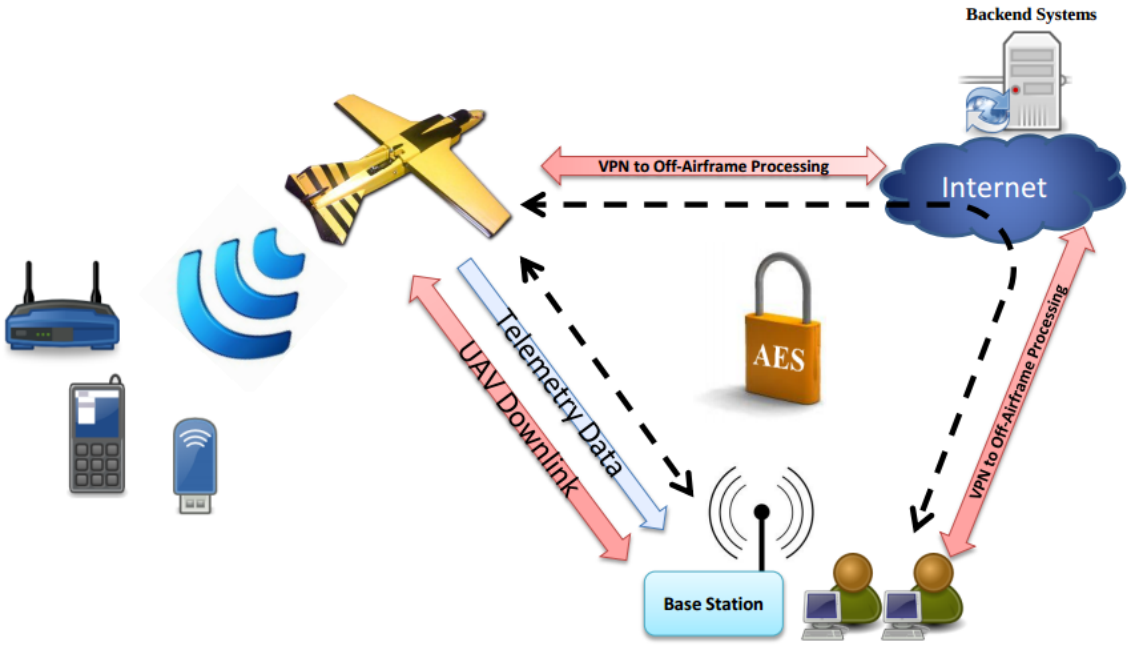


Figure 2.6 *WASP system topology[14]*

Figure 2.7 *Colibrex UAV[7]*

2.3.3 Atmospheric Satellites

Titan Aerospace (recently acquired by Google), a company originating from Moriarty, USA has presented a prototype of an atmospheric satellite at the 2013 AUVSI (Association for Unmanned Vehicle Systems International). It is a high altitude (20km) UAV plane designed to run on solar power and stay airborne for as long

as 5 years at a time. It's called an atmospheric satellite because it's capable of performing many tasks satellites do at a fraction of their costs. However, unlike a satellite they can safely come back to earth for maintenance and upgrades.

Solara 50 is 15m long and has 50 meter wingspan. All of it's surface is covered in photovoltaic cells to produce kilowatts of energy during the daytime, part of which is stored in Li-Ion cells for night time operation. It offers a payload of 32kg. The larger version, Solara 60 is projected to offer a payload of 100kg.

Flying at 20km of altitude gives Solara a line of sight view of an 45000 square kilometers which is roughly an area the size of Estonia and an equivalent of about 100 GSM base stations. Just like satellites a few decades ago, Solara opens up a world of possibilities for wireless communications.

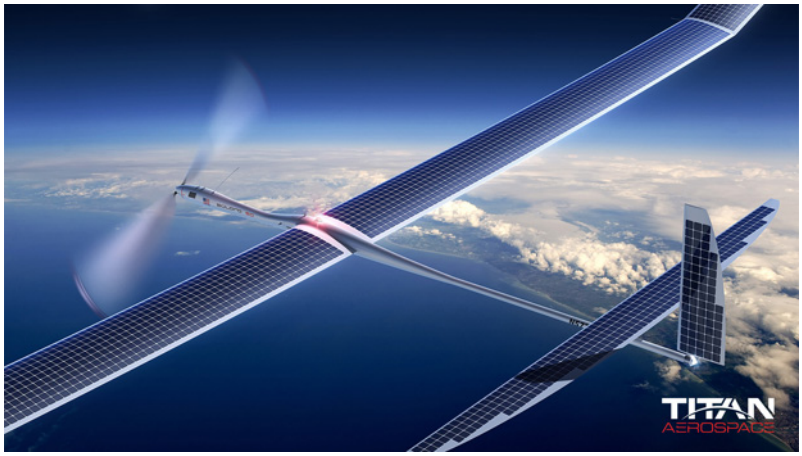


Figure 2.8 *Solara 50, source: IEEE Spectrum*

2.3.4 Phantom Eye

In some ways Phantom Eye similar to Solara atmospheric satellites. However, there are few key differences between them. Phatom eye is a high altitude long endurance craft (HALE) which means it's more of a traditional aircraft design. Unlike Solara, it's powered by liquid hydrogen and weighing at 4,5 tonnes. It comes pre-equipped with 500kg of basic communications equipment and on top of it can carry 200kg of payload. This comes the expense of cruise time being reduced to about 4-5 days. It's a military vehicle designed to provide intelligence and communication capabilities in combat zones and emergency situations. As such, it's also much faster cruising speeds than Solara.[8]

Figure 2.9 *Boeing Phantom Eye UAV[8]*

2.4 GNURadio

In traditional sense, when we talk about radio interfaces, we know that we're talking about hardware. However, in case of SDRs, the hardware described in section 3.2.2 is mainly an Analogue to Digital Converter (ADC) and all the signal processing is done in software.

On GNURadio's website we can find this short description:
GNU Radio is a free and open-source software development tool kit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

As the name suggests, GNURadio is part of the GNU Project. Following the GNU project philosophy, it's free software which means users are free to run it, share it, study it and modify it under the GNU General Public License (GPL). 10 years ago, Eric Blossom, head of the project, wrote this simple definition:
"GNU Radio is a free software toolkit for building software radios"[5]. However, over the past decade, it has evolved to accommodate a much broader spectrum of utility.

GNURadio was originally designed with reconfigurable software radios in mind, but it is also well suitable for simulations, signal processing or protocol engineering. It has been first released in 2001 and is constantly in development to keep up with

the technological advances of our times. The project has also spawned the USRP (Universal Software Radio Peripheral) - the first generation of affordable (sub 10000 EUR) SDR hardware.

2.4.1 GNURadio structure: Python, SWIG and C++

C++ programs run compiled to native machine code. C++ gives the programmer a lot of control and direct access to memory resources. As a consequence - a well written C++ code can be rapidly executed and be very memory efficient. The downside of this approach is that writing code is more complicated and having more control means a higher error chance thus coding takes longer. C++ code can also be convoluted and difficult to understand for third parties.

To contrast all that - Python is on the opposite side of the spectrum. It's a high-level programming language designed with code readability in mind. Many features such as memory management and data types are of no concern to the coder. The trade off for Python is sacrificing performance over simplicity and code readability.

GNURadio's structure takes the best of the two worlds. It has a highly modular design where parts of code are treated as black boxes with clearly defined outputs and inputs and a scheduler manages execution of these blocks. All of the code that performs time-critical signal processing and the scheduler are written in C++. Reconfigurability of an SDR reflects in re-usability of the code. In GNURadio terminology, the reusable 'black boxes' of code are called blocks . There is a block that interfaces the SDR hardware and acts as a signal source or sink, some sink blocks act as GUI, there are blocks that deal with modulation and many more. Before a signal is fed to the transmitter or becomes human readable, it has to pass through many blocks. The way these blocks are connected has a negligible impact on the performance of the system so that part is done using Python to facilitate repid creation of GNURadio applications. Python can also be used to quickly write simple blocks for when performance is not critical. This means that all of the C++ classes have their Python counterparts.

The interconnection of two unrelated programming languages is made possible by SWIG (Simplified Wrapper and Interface Generator). Similarly to the other tools used, SWIG is an open source software. As the name suggest, it generates the 'glue code' that enables calling the C/C++ functions from a large number of scripting languages (Python, JavaScript, Perl...).

GNURadio comes pre-equipped with a lot of blocks and many more are available

from 3rd party sources so for most SDR applications it's enough to connect existing blocks. SWIG greatly simplifies this process while keeping the performance up to par.

2.4.2 The flowgraph

A GNURadio flowgraph is a graph describing the data flow of a GNURadio application. SWIG makes a great job at simplifying the creation of GNURadio applications, but the code is very repetitive and, even with comments, difficult to understand with no programming knowledge. However, as of version 3.2, GNURadio also comes with a GUI (Graphical User Interface) called GNURadio Companion (GRC) that lets the user create SDR applications based on flow graphs. It provides an abstraction layer where blocks are 'black boxes' with input and output. We will have a closer look at the companion in the following pages.

Blocks

Figure 2.10 shows a flowgraph of a simple mono FM radio receiver. As stated before, blocks perform signal processing. For the sake of readability and modularity - ideally one block performs one operation, but the specifics are up to the author of the block and, while it's not recommended, it's possible to write an entire application as one block. Let us take a closer look at the blocks we see in the example.

Sources and Sinks

A source block is any block that has only output ports. It it's not necessarily an interface to a radio. A source block can be a signal generator or a file input. Similarly, a sink is any block that has only input ports. The example shows 2 sink blocks: one interfacing the audio card and one being a GUI element that displays the audio signal in frequency domain. The example graph has one source and two sinks. The source is an RTL-SDR interface. The sinks are an interface to the audio card and a GUI element that displays the signal in the frequency domain as it appears at the output of the RTL-SDR.

GUI elements and programming-specific blocks

Some blocks aren't related to signal processing. These can be subdivided in two categories: programming specific blocks and GUI elements.

Programming specific blocks add a little bit of programming functionality. These include variables for values that are used in many blocks such as sample rate, Python imports and the Options block that sets up some initialization parameters for the GNURadio application.

GUI blocks are used to build a graphical representation of the data like an oscilloscope, FFT or waterfall displays. There re also blocks that create sliders or textboxes that allow to tweak the applications parameters such as tuning the frequency during the runtime.

Data flow

From GR manual: *"GNU Radio was originally a streaming system with no other mechanism to pass data between blocks. Streams of data work well for samples, bits, etc., but can lack for control and meta data."* Streams are a basic data flow mechanism in GR. They can be compared to a water stream - they continuously flow in one direction, down the flow chart. As the GNURadio project evolved to support higher levels of abstraction such as PDUs and error checking, this was not enough and two more data flow mechanisms were added.

Stream tags are a mechanism to pass metadata between blocks. A stream tag can be attached to a single sample in a stream and propagate with it down the flowgraph. A good example of stream tags is tagging a beginning and an end of a GSM burst to avoid processing samples that carry no data and having to locate the bursts in every block.

The last method of passing data is message passing. Unlike streams, messages are asynchronous and event-driven. While streams are limited to carrying just samples, messages can carry all sorts of data or even a lot of different data fields thus making them very good for datagrams and PDUs.

Data types

One particularly important aspect to keep in mind are the data types. Because GR is a versatile tool, it supports many data types commonly found in C/C++ such as int, float, byte and complex of various sizes to fit any SDR needs. To make this possible, most of the blocks are C++ templates and the data types have to be specified upon initialization. It is also important to have matching I/O data types when connecting two blocks.

GNURadio does have one non-standard data type of it's own - polymorphic type (PMT). From GNURadio manual: *Polymorphic Types are opaque data types that are designed as generic containers of data that can be safely passed around between blocks and threads in GNU Radio. They are heavily used in the stream tags and message passing interfaces.*[6]

PMT in GNURadio can represent the following:

- Boolean values of true/false
- Strings (as symbols)
- Integers (long and uint64)
- Floats (as doubles)
- Complex (as two doubles)
- Pairs
- Tuples
- Vectors (of PMTs)
- Uniform vectors (of any standard data type)
- Dictionaries (list of key:value pairs)
- Any (contains a boost::any pointer to hold anything)...

Because these are opaque containers, the programmer needs to make sure that the PMTs are formed correctly between the block that generate them and the blocks that receive them. All these concepts are relatively new to GNURadio and are still under development and standardization.

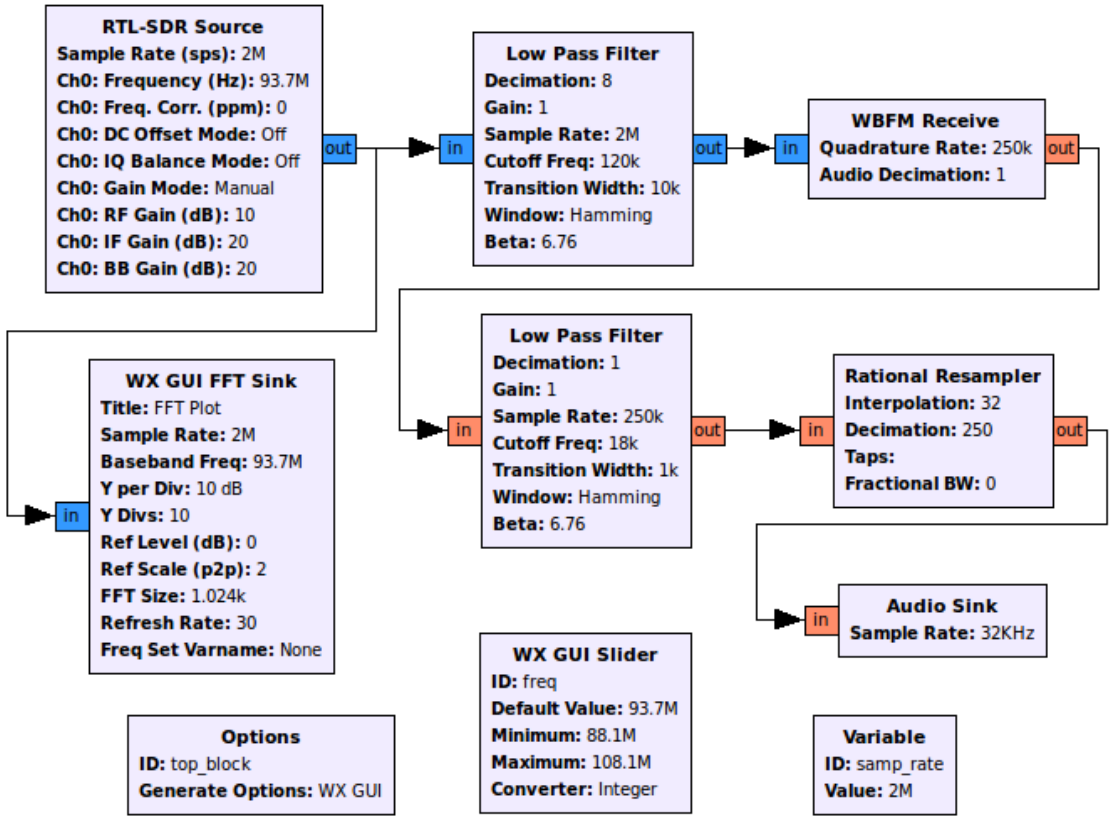


Figure 2.10 *GNURadio flow graph: mono FM radio receiver*

2.4.3 GNURadio Companion

GNURadio Companion (GRC) is GNURadio's GUI for creating SDR aplications. It's a WYSIWYG (what you see is what you get) editor that generates the python code that glues all the blocks together and is the actual GNURadio application. GRC provides a higher layer of abstraction that lets separate GNURadio from programming skills. Some programming paradigms such as utilization of variables or passing Python functions as initialization parameters to blocks can still be used. It doesn't provide the same degree of freedom as writing the code yourself, but it's sufficient for the vast majority of applications. More importantly, it provides a great way of visualizing the flow chart making it much easier to understand than the Python code.

In terms of GRC, certain programming-specific elements such as variables, Python imports or application GUI are also represented as blocks even though they don't take part in the flow of data.

2.4.4 Example GRC flowgraph

Figure 2.10 presents an example GRC flowgraph of a simple mono FM radio receiver. The 'Options' block specifies that this SDR will use Python WX GUI. Other GUI options are QT and CLI (Command Line Interface - no GUI). There are two WX GUI blocks. One is a slider used to tune the radio and the other is an FFT sink showing the spectrum of the signal as it is captured by the hardware. The resulting application can be seen on Figure 2.11. Following the data flow, we can see the source block interfacing the hardware radio. A low pass filter removes out of band signals. The WBFM Receiver block demodulates the FM signal. Another low pass filter removes the stereo component. The Rational Resampler decimates the signal to be fit for playback. Finally, the signal reaches an audio sink that interfaces the sound card.

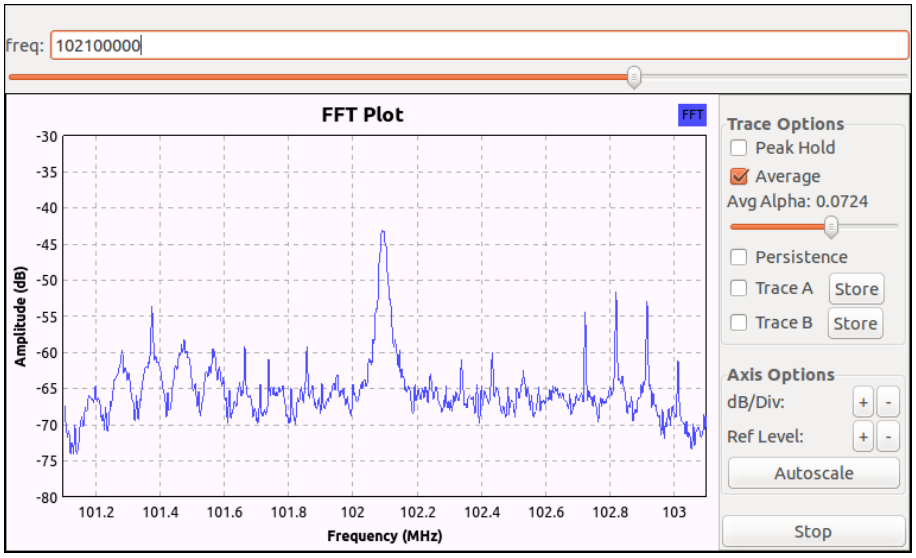


Figure 2.11 *mono FM radio receiver, GUI*

2.5 MAVLink protocol

From MAVLink's website[10]
textit"MAVLink is a very lightweight, header-only message marshalling library for micro air vehicles.
It can pack C-structs over serial channels with high efficiency and send these packets to the ground control station. It is extensively tested on the PX4, PIXHAWK, APM and Parrot AR.Drone platforms and serves there as communication backbone for the MCU/IMU communication as well as for Linux interprocess and ground link communication."

MAVLink it de facto a standard when it comes to Micro Air Vehicles and open source. It was first released in 2009 by Lorenz Meier, the man behind PX4 project. Today it comes in a few different flavours as some autopilots add some messages that are specific for them, but the packet structure (Fiure 2.12) never changes.

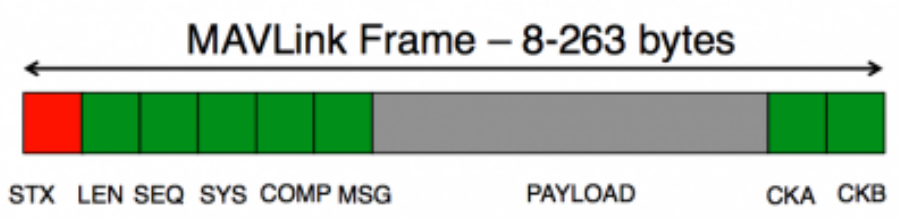


Figure 2.12 *Structure of a MAVLink packet*

The protocol is released under LGPL licence so libraries in C and Python are freely available and can be easily modified to suit any particular use.

2.6 Google Maps API

Google Maps API is written in JavaScript and offers a wide a array of tools to represent geographical data. Unfotunately, Google's heatmap API doesn't take the data values of the data points. It only visualises the concentration of the data point on a map ie. The more data points in an area, the warmer the color. Because of lack of tools, the map had to be constructed from scratch using 'Polygon' object from Google maps api. The script divides the area into squares, calculates the average value of data points within a square and converts that value into decibels. Subsequently, these values are normalized to (0,1) and assigned a color from a red-blue gradient which is displayed on the polygon.

3. THE PROPOSED SYSTEM

To put it in the simplest terms possible - the idea is to strap a computer and a radio onto a small, flying machine. First part of challenge is to find the components that are capable of working together as one mechanism. The second, even bigger part it to make them work with each other and produce a desired output.

Some of the solutions and commercial applications have more capabilities than the project discussed in this thesis. It is worth noting these projects have had extensive budgets and took years of development by teams of skilled professionals. While it's inspired by already existing projects, especially WASP, this project is not meant to compete with them, it's more of a proof of concept intended to showcase that thanks to the advancements in technology, open-source and open-hardware architectures similar projects are possible on a household budget.

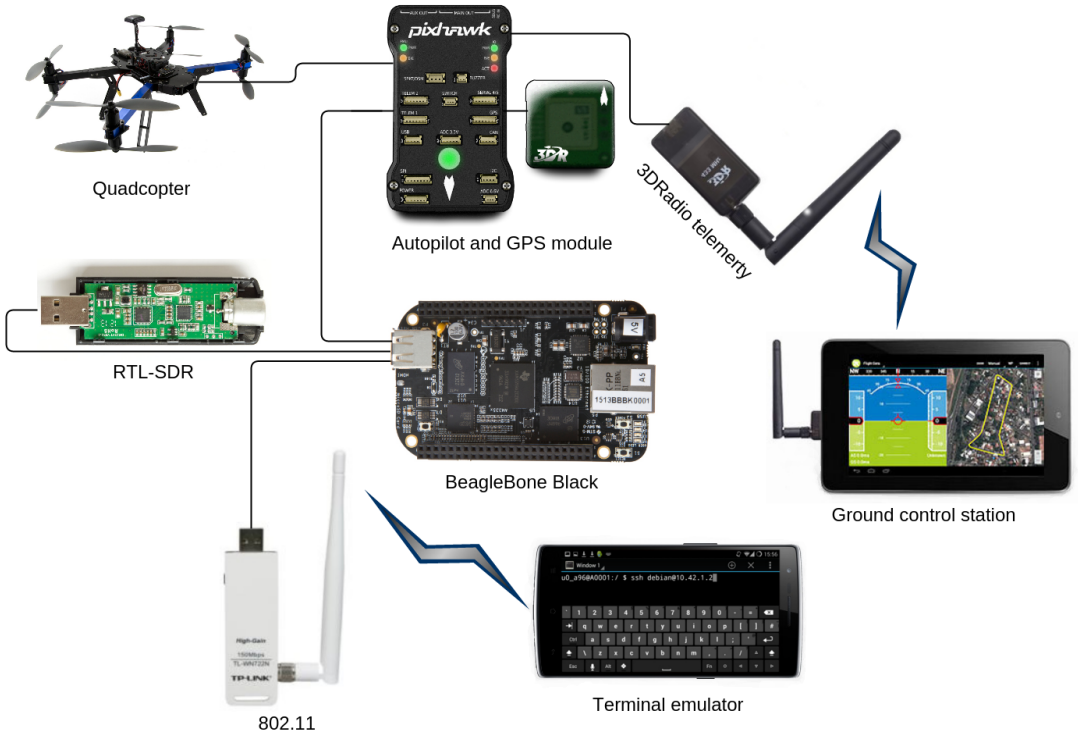


Figure 3.1 *Proposed system topology*

Another aspect that was kept in mind was modularity and compatibility. Because the equipment is lightweight and power efficient, it can be used in conjunction with any type of UAV that supports the MAVlink protocol and is capable of carrying about 100g of payload. For development purposes we used a large, heavy-lift general purpose quadcopter drone, but the equipment can easily be moved onto something more aerodynamic like a plane drone to cover larger areas or a blimp to stay airborne and operational for longer periods of time or simply a smaller quadcopter that would make it possible to manoeuvre in obstacle filled areas without causing damage.

Figure 3.1 showcases the hardware connections between the system's components. We will go through them in more detail in the following pages.

3.1 Open Source

As it was shown in the previous chapters, there are similar projects out there, but they all consist of extensive budgets and years of development and teams of skilled professionals. All of these are beyond the scope of a master thesis and the solution was to tap into the work of other people that has been made freely available for everyone to use. Almost all of the components used are open architectures.

Open source projects come with no price tag, but also with no warranties of any kind and the support is only as good as the community involved in the project. While it may not seem so - the latter is not necessarily a bad thing and an active community will be much more helpful than a person paid to listen to disgruntled customers. A community embraces the aspect of knowledge sharing and it's key members often are experts in their fields.

3.2 Hardware

The hardware part of the project consists of three major components (Fig. 3.2). At the core there is BeagleBone Black - a Single Board Computer (SBC) acting as the data sink for the other components. It also performs the signal processing of the captured samples and logs the results that we are interested in.

Other parts include: a Software Defined Radio and a Flight Management Unit (FMU). The former tunes in to the desired frequencies of the RF spectrum and outputs a sampled signal ready to be processed by the SBC. The latter is a microcontroller equipped with an array of sensors running a Linux-based Real Time Operating System (RTOS) dedicated to control the throttle on the drone's engines.

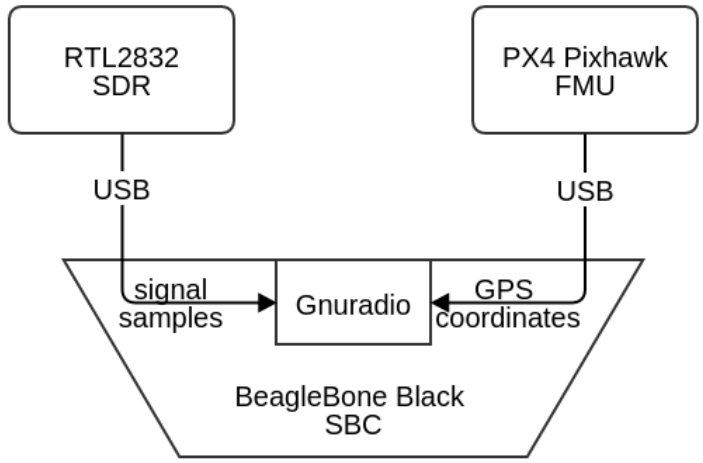


Figure 3.2 *Dataflow between hardware components*

It also feeds the current position to the SBC. Other components visible on Fig. 3.1 don't take part in data acquisition or processing, but are used for controlling and supervising the system. The terminal emulator is an android phone that has a terminal connection to the SBC and the ground control station supervises all the FMU functionalities.

3.2.1 BeagleBone Black

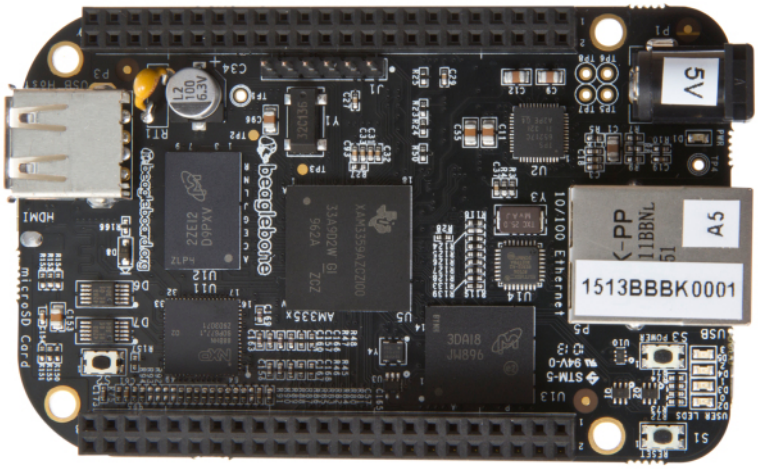


Figure 3.3 *BeagleBone Black SBC*

According to Moore's law [12] of integrated circuits the density of transistors doubles

every two years. The law held true since 1965 and many consider it equivalent to processing power doubling every two years. However, it also means that the performance of a full sized personal computer driven by 300 Watt power supply from 10 years ago can be delivered today by a tiny board powered by a phone charger. Since the success of Raspberry Pi, we've seen SBCs proliferate and become more popular, accessible and affordable.

BeagleBone Black (BBB) is an ARM based Single Board Computer (SBC) similar to Raspberry Pi. Table 3.1 shows technical specifications of the BBB. This board is particularly suitable for signal processing because it's processor includes a NEON Floating Point Unit accelerator that speeds up floating point operations. Because we want the SBC to operate on board of a flying, battery powered machinery, the weight and the energy consumption are of significant importance. BeagleBone, weighing at 37g and opertaing at only 2W of power is a great board for this application. Another strong point of this SBC is it's popularity which results with better support, software compatibility and a large selection of accessories. The last-mentioned may seem of little importance, but having a ready-made case that can be easily attached to the drone's body saves time.

BeagleBone Black specifications	
CPU	AM335x 1GHz ARM Cortex A8 NEON floating-point accelerator 2x PRU 32-bit microcontrollers
GPU	PowerVR SGX530
RAM	512 DDR3
Storage	4GB flash + microSD

Table 3.1 *BeagleBone Black technical specifications*

The board requires a 5V DC power supply. The UAV battery is a 4 cell Li-Po with nominal voltage of 14.8V. The voltage for this type of battery can range from as low as 12V when discharged to 16.8V when fully charged. To assure a constant power supply for the BBB, a step down voltage regulator (a.k.a. a buck converter) has been used (Fig. 3.4). SBC's power consumption is negligible compared to the craft (2W vs. up to 2.5kW) so it can be safely powered from the same source.

3.2.2 SDR: RTL-SDR

RTL-SDR is a general term referring to a whole family of devices based on the inexpensive RTL2832 chip manufactured by Realtek [13]. It is actually intended to ba a part of a DVB-T (Digital Video Broadcast - Terrestial) receiver and was not

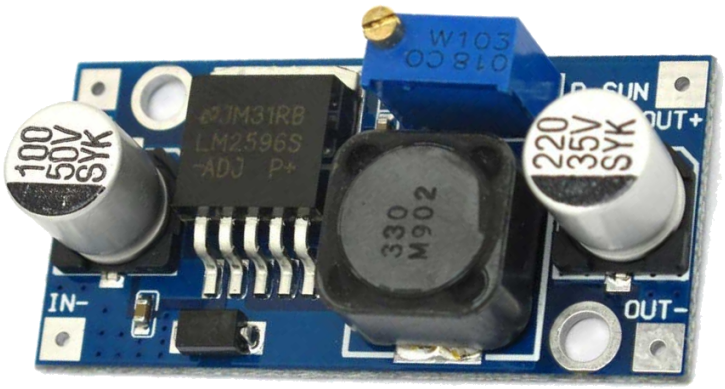


Figure 3.4 *Step down voltage converter*

intended to be used as an SDR. The is chip responsible for demodulating the signal and feeding it to USB interface. However, it has a hidden feature that makes it possible to access raw I/Q samples. Any DVB-T tuner based on that chip can be used as an SDR after unloading the default kernel driver and replacing it with one that supports access to the raw I/Q data.

These receivers can cost as little as 20USD. It's not surprising that they can't deliver the same performance as their much more expansive counterparts but for many applications they are more than enough. Because it's affordability, it became quite popular. The popularity caused the proliferation of SDR software and growth of projects such as GNURadio. None of these would see daylight if the access barrier for this technology was still within the range of a couple thousand EUR. Examples of existing applications include: tracking maritime boat positions, high quality entropy source, satellite imagery receiver, triangulating source of a signal and many, many more.

The specific receiver used for this project is Terratec Cinergy T-Stick RC (Rev.3) 3.5. It comes with Elonics E4000 tuner that lets it access frequencies of up to 2.3GHz without an additional downconverter. Most tuners coupled with RTL2832 tune only up to 1.7GHz making it impossible to peek at 1.8GHz GSM channels.

It's also worth noting that this is just the hardware part of the SDR. As the name suggests much of the radio is done in software. Next section will bring a more in-depth description of the software part of the SDR - GNURadio.

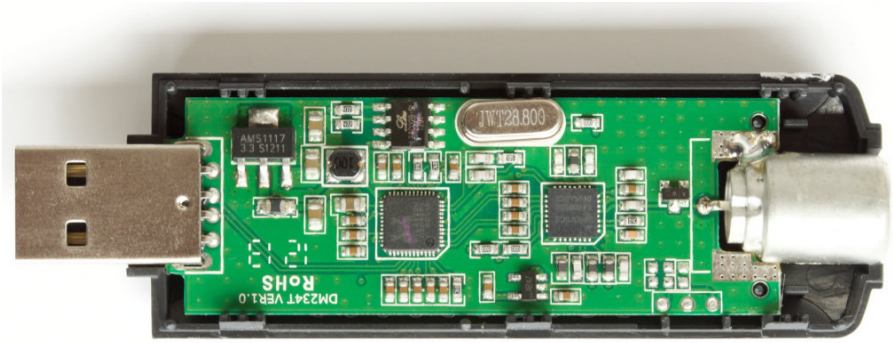


Figure 3.5 *Terratec Cinergy T-Stick RC (Rev.3)*

3.2.3 3DRobotics X8 2014

3DRobotics X8 (Fig. 3.6) is an equivalent of a truck in the world of drones. It's a heavy-lift, high redundancy, high-stability quadcopter capable of fulfilling many different applications. It has a nominal payload of 800g and 8 electric brushless motors in an 'X' configuration. The combined power of the motors is roughly 2.5kW. 8 engines in 'X' configuration gives additional redundancy as the quad can easily function with just 4 or even 3 motors. It also translates into higher stability against wind gusts. The frame and the tall landing gear have plenty of room for additional components. It's downsides are high power consumption, weight and difficulty to navigate in tight spaces. Such a large and heavy drone also has more potential for causing damage.

3.2.4 Pixhawk autopilot

PX4 or Pixhawk is a Flight Management Unit. Similarly to BeagleBone it is an SBC, but it's custom made for the specific purpose of flight management. Because it is crucial that all of the operations regarding flight management are performed in real time, it is impossible for BeagleBone to handle it as taking just a split second to write a log file can result with a crash. To prevent that from happening, Pixhawk runs NuttX, which is a POSIX compliant Real time Operating System (RTOS).

Pixhawk is both open-source and open-hardware project developed and supported by the PIXHAWK Project of the Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology), the Autonomous Systems Lab and the Automatic Control Laboratory.

It incorporates a number of sensors like gyroscope, accelerometer or barometer. It

Figure 3.6 *3DRobotics X8 drone, source: 3DRobotics*

also has a number of drone specific input and output ports like a radio receiver for remote control, a GPS connector or the telemetry port.

3.2.5 Telemetry and Ground control Station

Figure 3.7 *3DRobotics telemetry set*

The key advantage of drones is their autonomous mode of operation. However, it's greatest strength is also it's greatest weakness. The larger the drone, the greater

potential for damage. A drone should never be left unmonitored. Current US regulations dictate that autonomous operation is allowed as long as the drone stays within the line of sight (LOS) of the operator. From the authors experience, even this can be problematic without an onboard camera. Even while maintaining LOS, the drone is so small it's impossible to tell it's orientation at a distance exceeding 100m.

In spite of the technology's rapid development, autonomous obstacle avoidance is still in research and far from a commercial deployment. While manual steering offers a way to pass controls to the drone, telemetry offers a bidirectional link to the drone providing the pilot with much of essential flight data.

3.2.6 Remote terminal

A problem encountered during the testing phase was the inability to have a real-time peek at the data being logged. Because the drone had to be detached from any device that could logon to BBB's console, it was only possible to check the logs back in the office once the device could be connected to a computer.

To tackle this, we used a spare TP-LINK TL-WN722N USB wireless card and an android phone. The Debian distribution on the BBB comes with the wireless driver already pre-installed. All that was needed to be done was configuring it to create/join an ad-hoc network.

Android has no native support for ad-hoc networking. However, since Android is an open source project, it is possible to find 3rd party firmware packages that provide IBSS support and Cyanogenmod is one of them. The device used to emulate a terminal was Oneplus One which comes with the only commercial version of Cyanogenmod and were lucky to have one of these at our disposal. Cyanogenmod can easily be installed on any rooted Android device.

3.3 LINUX environment

Linux comes in many different flavours. However, this is the case when we're talking about well-developed platforms such as x86 or x64. The matters complicate when we're going after ARM architecture as they are not as popular and even if ports are available, quite often they are still under development and are subject to bugs. Another important aspect was the availability of libraries. Missing libraries can be downloaded and installed manually, but GNURadio depends on 47 different ones,

and making sure that every single one of them is working fine and dealing with potential problems is too much work. Luckily, Debian Wheezy 7.6 turned out to satisfy all the requirements. It's a reliable, mature distribution and it has all the required libraries in it's repositories.

3.3.1 Debian on BBB

BeagleBone comes with 2Gb internal eMMC RAM storage. It's enough to have a functioning Debian installation, but after that there is not much space left and it's nowhere near enough to compile and run GNURadio. To bypass that, the system was installed on an external microSD card. We've used a 16Gb Class 10 card. There is a publicly available image containing a precompiled, BeagleBone tailored version of the system. We need to download and block copy it to an external microSD card. To do that, the following commands need to be executed on an external machine (in this case, it was Ubuntu x64 14.04.1 LTS).

```
~$ wget https://rcn-ee.net/deb/microsd/wheezy/bone-debian-7.6-console-armhf-
~$ unxz bone-debian-7.6-console-armhf-2014-08-13-2gb.img
~$ sudo dd if=./bone-debian-7.6-console-armhf-2014-08-13-2gb.img of=/dev/mmc
```

where /dev/mmcblk0 is the desired microSD card.

Once the operation is complete, the card needs to be inserted into BeagleBone's microSD slot. BBB boots from eMMC by default. To force booting from external microSD, the 'boot' button has to be held down upon powering up until all 4 LEDs start flashing. Once Debian boots up and we log in (default login:password is debian:temppwd), the partition needs to be expanded to encompass the entire card. The precompiled images comes with a script to support that. The partition can be grown with the following commands:

```
$ cd /opt/scripts/tools
$ git pull
$ ./grow_partition.sh
$ sudo reboot
```

After that procedure, the partition takes up the entire microSD card and we have space to proceed with the GNURadio installation.

3.3.2 GNURadio installation

The ARM architecture also poses some problems to the default installation of GNURadio. It is possible to install it from the repositories, but they usually lag behind

the main branch and message passing mechanism wasn't yet implemented in the version currently found in the repositories. In this case we've built it from source to get the latest version. Fortunately there is a shell script automating the downloading all the components, meeting their dependencies and compiling them:

```
~$ wget http://www.sbrac.org/files/build-gnuradio
```

The downside of the script is that it wasn't created with ARM architecture in mind and requires some additional editing. As of GNURadio version 7.3.4, additianl ASM flags have to be passed to every cmake call:

```
-DCMAKE_ASM_FLAGS="-march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=n
```

Without it, there are critical errors during compilation and the build fails. The script needs to be edited to include these before running it.

3.3.3 Distcc

Compilation of a large project like GNURadio requires a lot of computing power and can produce critical errors that make it impossible to get it right the first time. Spending hours on an unsuccessful compilation is a waste of time so some technique of speeding the process up is desirable. Distcc distributes compiling task across networked computers, leaving only the linking to be done by the thin client. It's open-source and freely available from the repositories in both x64 and armhf architectures so it can be installed on both machines using apt-get. It's a versatile tool, but here we will just go through a very basic configuration.

BeagleBone thin client

The trick is to use distcc instead of default compilers. The easiest way is to substitute CMake variables CMAKE_CXX_COMPILER and CMAKE_C_COMPILER. However, these are defined only after running cmake in the CMakeLists.txt file. A workaround is needed if we are to automate the process with the build-gnuradio script. One way of achieving it is to masquerade distcc to be called instead of the compilers by using symbolic links and adding the path to the links to the beginning of PATH environment variable.

```
~$ mkdir .links
~$ cd .links
~$ sudo ln -s /usr/local/bin/distcc gcc
```

```
~$ sudo ln -s /usr/local/bin/distcc c++
~$ export PATH=~/.links/:$PATH
```

The export command works only for the current session so after relogging the PATH will be restored to it's original value.

Now, we need to configure the IP's of the computers that will perform the compilation.

```
~$ export DISTCC_HOSTS="10.42.0.1"
```

Distcc server

The server used in this work was 4th generation i7 machine running Ubuntu 14.04.1. Because it's a different architecture, an ARM cross compiler had to be used and masqueraded with symbolic links to replace the default compilers in order to produce ARM machine code. After that, distcc can be run and configured command line arguments:

```
~$ distccd --daemon --jobs 8 --allow 10.42.0.80
```

This command starts a daemon process in the background with 8 child processes that listen for incoming compilation requests from 10.42.0.80 (the BeagleBone's IP). The '–jobs' argument specifies the number of concurrent processes to run. There should be one process per CPU core.

3.3.4 Default RTL kernel driver

RTL chips aren't meant to be used as SDR devices and Linux kernels come with driver that enable DVB-T and FM reception using these devices. That driver needs to be removed.

```
~$ sudo modprobe -r dvb_usb_rtl28xxu
```

unloads the default driver. To make sure that the correct driver is in use, we should test the reception of samples from RTL-SDR:

```
~$ rtl_test -t
```

rtl_test is one of the executables created during the installation of GNURadio.

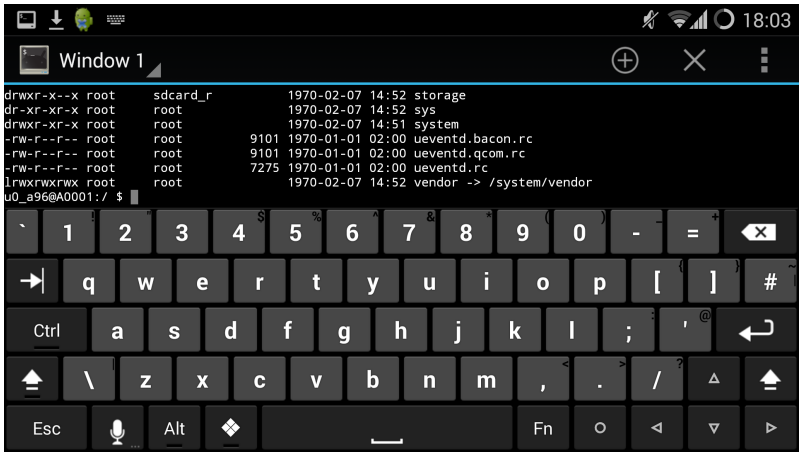


Figure 3.8 *Android emulating a terminal*

To avoid unloading the default driver after every login, it can be blacklisted in modprobe:

```
~$ sudo nano /etc/modprobe.d/ban-rtl.conf
blacklist dvb_usb_rtl28xxu
```

3.3.5 Remote terminal access

To have access to BeagleBone's CLI during tests, an ad-hoc wireless network needs to be configured:

```
debian@arm:~$ sudo ifconfig wlan0 10.42.1.1
debian@arm:~$ sudo ifconfig wlan0 broadcast 10.42.1.255
debian@arm:~$ sudo ifconfig wlan0 netmask 255.255.255.0
debian@arm:~$ sudo ip link set wlan0 down
debian@arm:~$ sudo iw wlan0 set type ibss
debian@arm:~$ sudo ip link set wlan0 up
debian@arm:~$ sudo iw wlan0 ibss join copternet 2452
```

Where 'ibss' stands for Independent Basic Service Set, which is another name for an ad-hoc network and 'copternet' is the BSSID of the network we're creating.

To achieve connectivity on the terminal side, some additional software was installed. Terminal Emulator for Android by Jack Palevich gives access to UNIX-like Command Line Interface (CLI and Hacker's Keyboard by Klaus Weidner gives an on-screen keyboard that includes the keys missing in Android keyboard needed for navigating the CLI such as Ctrl, arrows or Tab. A screenshot of a working Android terminal is presented on Fig. 3.8.

3.4 Frequency correction algorithm

Accurate clocks are expensive. For that reason it's not possible to have a high-accuracy clock built into an affordable SDR hardware. Furthermore, the frequency error is not a constant and it is highly dependable on environmental factors. The more expensive radios will have more accurate clocks or bypass that problem by synchronising to a GPS signal. Mobile stations in a cellular network, for which affordability is all the more of an important factor, are suffering from the same problems. Fortunately, GSM has a built in solution for this problem.

GSM has very tight requirements for frequency synchronization. This article [16] suggests a maximum acceptable frequency mismatch for reliable timing acquisition of 500Hz. The technical specification[3] requires that *"the MS carrier frequency shall be accurate to within 0.1 ppm."* This means that at 900MHz band the frequency mismatch should be less than 90Hz (or 180Hz for GSM1800). Common ppm values for low-cost crystal oscillators are about 20 - 50 ppm which is 3 ranges higher than GSM specifications meaning an error of 19-40kHz. More specifically , the mismatch for RTL-SDR in an office environment was measured to be roughly 27kHz and for the more expensive board, USRP B200, the deviation was about 2.7kHz. Both of them are falling short of the requirements.

GSM has a logical channel dedicated to solving that problem. It's called the Frequency Correction Channel or FCCH for short. Each base station broadcasts a frequency correction burst on it's pilot frequency. This burst consists of all '0' bits which, because of differential encoding becomes a sequence of alternating '0' and '1' symbols. After GMSK modulation, these symbols create a sine wave oscillating precisely at 67.7033kHz. Figure 3.9 depicts the spectrum of a GSM signal at base-band captured with RTL-SDR. The green line are the peak recorded values. There is a clear peak at around 40kHz meaning that the frequency correction is roughly 27kHz. When the receiver is out of sync, this burst will have a different frequency upon receiving it and a frequency correction needs to be applied to shift it to the original 67.7033kHz.

While the GSM specifications clearly describe the FCCH, they leave the detection method of frequency burst up to the manufacturer of the equipment. For the purpose of this thesis, we've implemented an algorithm described in this paper[16] with a few modifications. The detection method uses an adaptive line enhancer (ALE) to identify the frequency burst.

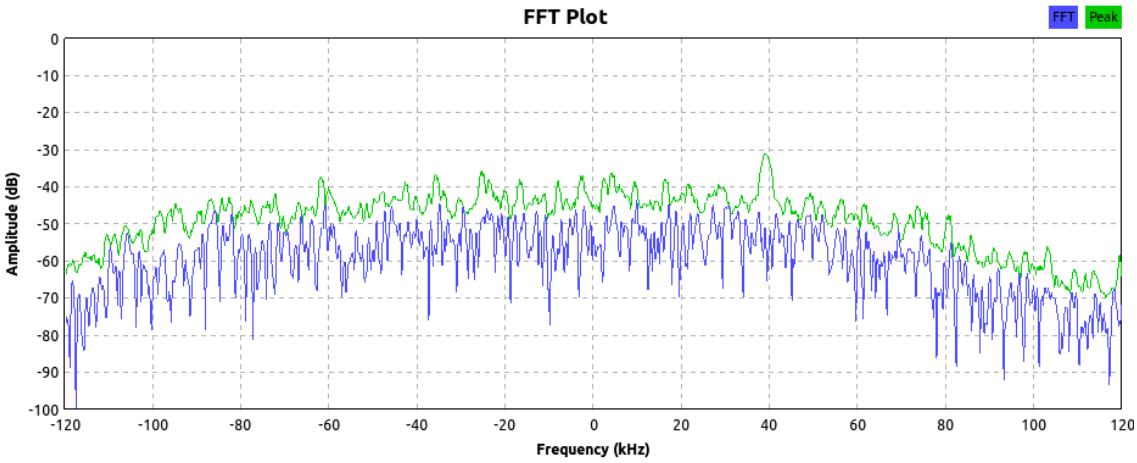


Figure 3.9 *Spectrum of a GSM channel before applying the frequency correction, captured with RTL-SDR*

3.4.1 Adaptive Line Enhancer

ALE (3.10) is a type of adaptive filter which means it's coefficients are updated as the signal passes through it. More specifically, line enhancement is removing noise from a signal consisting of one or more sinusoidal waves corrupted by broadband noise[9]. ALE is a filer that uses a delayed version of input signal to try to predict the next samples of the signal. The predicted output is subtracted from the signal to compute the error function which, in turn is used as feedback to the ALE.

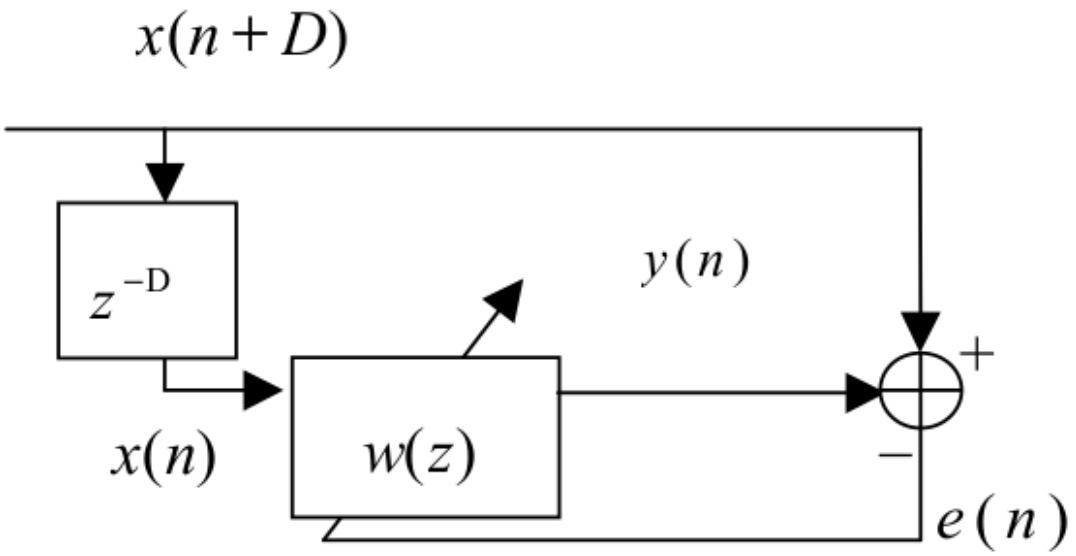


Figure 3.10 *Adaptive Line Enhancer[16]*

3.4.2 Detection principle

ALEs are meant to remove wideband noise from a narrowband signal, but we're not trying to do that. Instead we build on the convergence property of the filter. When the signal fed to the predictor is a pure tone, the value of the error function will significantly drop thus indicating the presence of a tone.

After every iteration, the prediction error e(n) is calculated:

$$e(n) = x(n + D) - y^*(n) \tag{3.1}$$

Where * denotes the complex conjugate

and filter coefficients w(n) are updated accordingly:

$$w(n + 1) = w(n) + G * e^*(n) * x(n) \tag{3.2}$$

where G is the convergence factor or the step size.

Because the values of samples vary significantly (there is a minimum of 30dB difference between burst and guard period), we can't effectively use the computed prediction error as a measure of convergence. Instead, we normalize average error with respect to received power. Average error power is defined as:

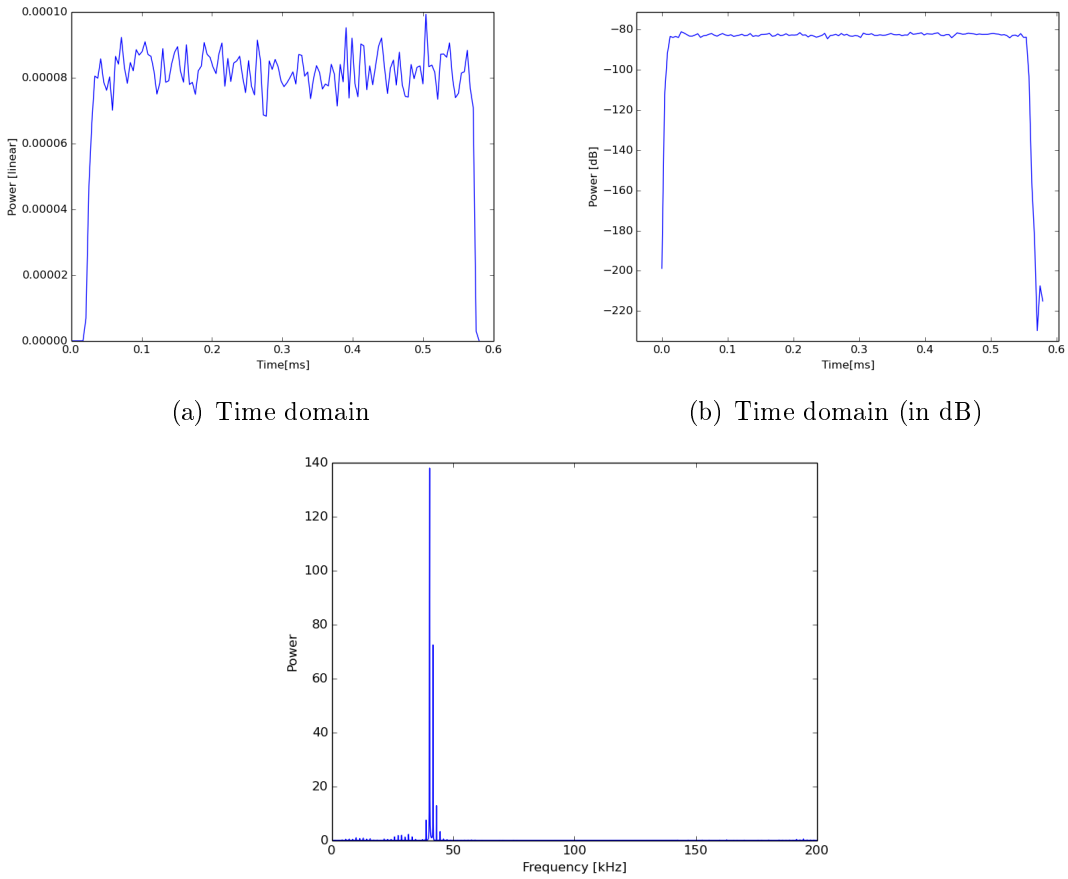
$$\overline{e}(n) = (1 - \rho) * \overline{e}(n - 1) + \rho * |e(n)|^2 \tag{3.3}$$

The function we use to as a measure of the filter's convergence is defined as:

$$c(n) = \frac{\overline{e}(n)}{\sum_{i=0}^{F} x(n - i)} \tag{3.4}$$

Where F is the number of filter coefficients used by the ALE filter.

The convergence factor G 3.2 is a important parameter of the filter and needs to be chosen carefully[9]. If it's value is too high, the filter will diverge. If G is too low, the filter will converge too slowly making it difficult to spot frequency bursts. According to[9], in order for the filter to converge, G should not be positive and smaller than twice the inverse average power of input signal:



(a) Time domain

(b) Time domain (in dB)

(c) Frequency domain

Figure 3.11 *Frequency correction burst in time and frequency*

$$0 < G < 2/\overline{P} \tag{3.5}$$

3.4.3 Boundry detection

Just as the ALE converges in presence of a pure tone, it also rapidly diverges when the tone is gone. The end of a frequency burst will always be marked by a steep increase in the convergence function. Finding that makes it easy to determine the boundaries of the burst.

3.4.4 Fasle positives

According to the publication[16], FCH detection could be achieved by determining a threshold level of the convergence function. If the function falls below a certain level,

it indicate the presence of a frequency burst. However, the convergence function periodically produces 'false positives' when the function rapidly spikes down towards '0' just to return to it's average values a few samples later. ALE converging in presence of a FCH burst and a false positive of the convergence function can be seen on the figure 3.12. One possible solution was to tweak the convergence factor. However, this would smooth the shape of the function and it would make the edge detection more difficult. To avoid detecting a false positive, another criterion has been added.

The algorithm not only checks if d(n) falls below a certain threshold, but also stays under it for 2/3 of a burst duration in samples. An arbitrary threshold has been chosen Once a burst has been successfully detected, the iterator moves the equivalent of 79 time slots.

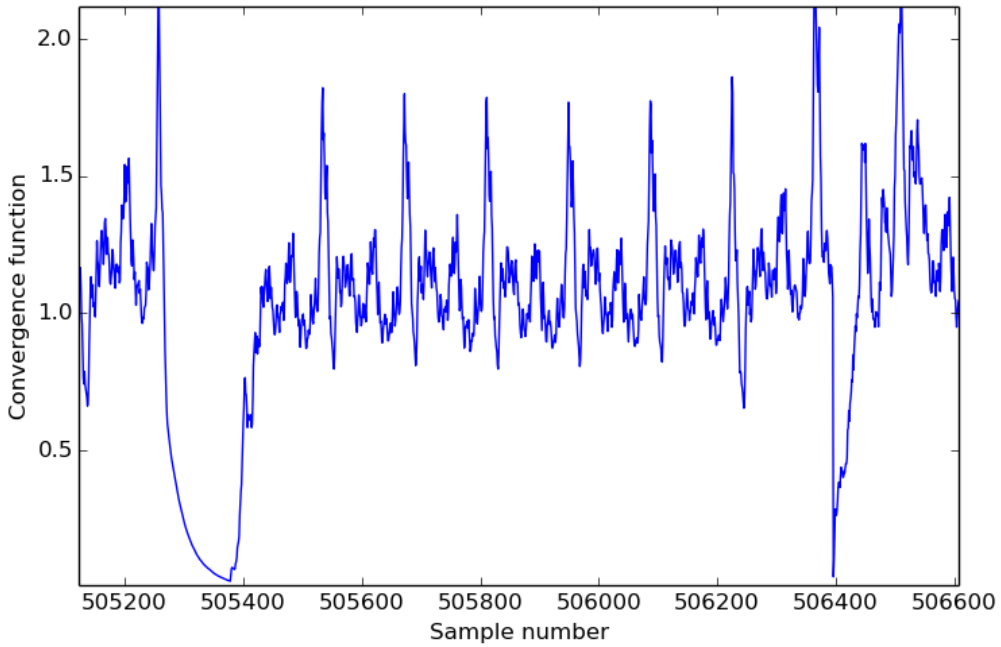


Figure 3.12 *Convergence function, FCH burst detected on the left, false positive on the right*

3.5 The power of a GSM burst

Fig. 3.13 shows the power levels of a GSM burst at the transmitter. We want our device to be able to distinguish between what is and what isn't a valid GSM burst. However, while GSM bust is strictly defined at a transmitter, there are no standards on defining it's shape at the receiver. Because of the nature of the wireless medium,

the signals at the receiver differ significantly from what is transmitted and a lot of work is put into reconstructing the original signal. To reflect the signal arriving at the receiver we had to readjust the mask from the technical specification. In the detection algorithm, we're retained the 30Db difference between and active burst and a guard period, but 2dB tolerance during the burst period had to be omitted.

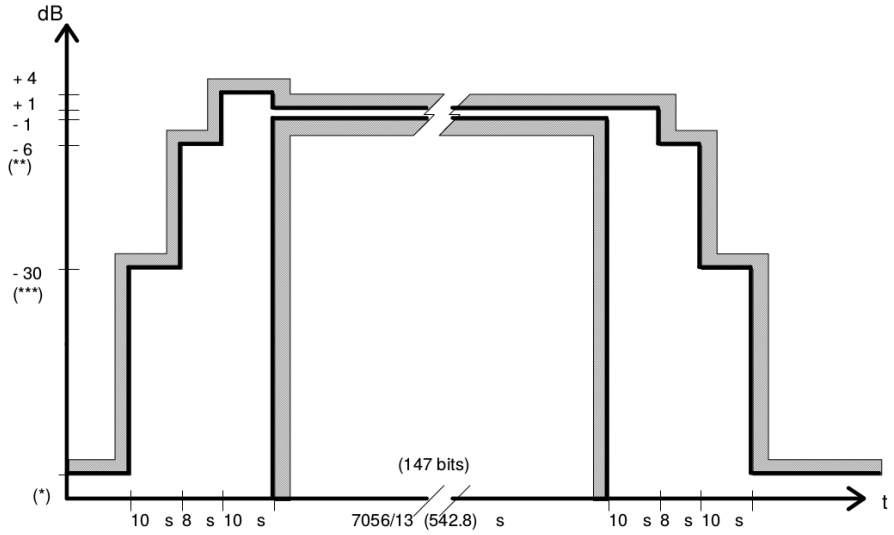


Figure 3.13 *Power mask of a GSM burst at the transmitter*

3.6 The GNURadio application

Fig. 3.14 show the GRC flowchart for the final application. Just as presented on the chart from Fig. 3.2, the blocks can be split into two categories - blocks handling the sampled signal and the blocks handling the GPS data. To reflect that, they have been coded as two, separate GNURadio modules that we will look into.

Two of the blocks present are built-in standard GNURadio blocks. One of them, the RTL-SDR source, we've covered in previous chapters. The other - Frequency Xlating FIR filter serves a few purposes in the flowgraph. First of all it removes the DC offset. Because we're using cheap SDR hardware, it introduces a DC component around the in the center frequency. To remove that, we are receiving a wider band that is shifter in frequency by 500kHz, the Xlating filter shifts it back to center frequency, narrows the spectrum to just 240kHz by decimating the samples and filters out the unwanted frequencies from our signal. It also has an input message port. through which the frequency correction is applied.

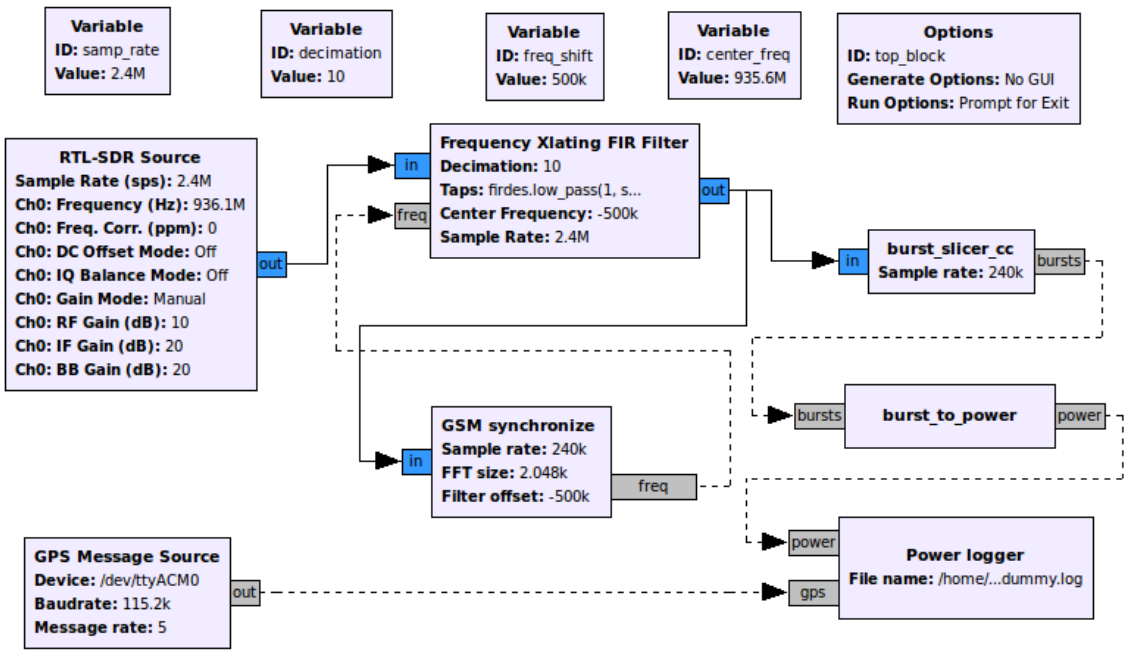


Figure 3.14 *GRC application*

3.6.1 gr-mavlink

The GPS Message Source and Power Logger blocks are part of this module. As their names indicate, the former connects to the FMU and acquired the positioning data and passes it down the flowchart as a message. We can specify the address of the serial port, it's baudrate and the frequency of messages.

The Power Logger block listens for incoming messages on the 'gps' port while simultaneously keeping track of all the values arriving on the other port. Once a message arrives on the 'gps' port, the block writes the coordinates and an arithmetic average of all the values that arrives on the 'power' port since the last 'gps' message.

3.6.2 gr-gsm_measure

This module contains the blocks used in analysing the GSM bursts. By far the most complex of all - the GSM Synchronise block locates the frequency bursts, computes it's FFT and the frequency correction, which is sent to the Frequency Xltaing FIR filter.

The burst_slicer_cc block locates burst boundaries and packs samples of a single burst into a message. Lastly, the burst_to_power block calculates the average power of a received burst and passes it in a message as a float to the logging block.

3.6.3 Addidtional Python modules

Libraries in terms of python Python are referred to as 'modules'.

Since accessing MAVLink data is not performance critical, we've used the pymavlink Python module providing an API to access all the information that is made available by the autopilot. Pymavlink is openly available on github[11]. Pymavlink module has dependencies of it's own and requires pyserial module to function properly.

These two modules need to be present and their location has to be added to the PYTHONPATH environment variable for Python to make use of them.

4. TESTING SCENARIOS AND NUMERICAL RESULTS

The results are logged in a file for further analysis. Each lines consists of 2 integers and a float and corresponds to a single measurement. The integers represent latitude and longitude. One can notice that the values extend way beyond (-90, 90) and (-180, 180). That is because they would otherwise have to be converted to floats and Pixhawk natively represents them as integers to avoid all of the problems commonly associated with floats. To get a more readable version of the coordinates, we need to divide them by 10000000.

```
614503225,238564281,1.23053
614503218,238564281,1.23053
614503212,238564280,1.23659
614503207,238564279,1.21532
614503199,238564277,1.24606
614503194,238564276,1.26538
```

Power is represented in linear scale. It's important to remember that these numbers are not Watts. Since the measuring equipment is low-cost and not designed for measurements, the units are arbitrary. The device can be calibrated using a more precise and professional equipment. However, we convert these units to Decibels. Because what we're really interested to see is the difference of signal levels, even an uncalibrated device still provides us with meaningful data.

4.1 Graphical representation

A logfile with thousands of lines of numbers is less than an optimum representation of the gathered data. Not only it is impossible to see all of it at once, but a series of numbers is meaningless. We are not good at reading data from large strings of numbers. The logfile isn't worth much until we have a way of representing it as an human-readable. Because this it geographical data, the best way of representing it is to put it on a map. Google maps API has been used to create a JavaScript to

present the dataset in a user-friendly way.

4.2 X8 and university's surroundings

Appealing as it may seem, using the drone for testing purposes was less than convenient. There is a lot of downtime between flights to charge the batteries. The drone and the RC transmitter are bulky and heavy to carry in and out of the building. X8 is a big, heavy lift drone and navigating it through closed spaces is dangerous and illegal without proper training. The university and it's surroundings are rather crowded areas. In fact, the as the lawmakers are gaining on the technology and regulations are enforced, many countries require a license to fly a drone of this size. Another downside of a general purpose drone is the battery lifetime which is about 15-20 min, the large scale test took roughly 30 minutes. Having all that in mind, the tests were performed with the engines off.

4.3 Small scale test

For the initial tests, we've wanted a small scale test that was easy to perform, in a space that was relatively easily accessible and could be easily repeated. We also wanted to do them in a where we already know what kind of results to expect.

The first tests were performed around one of the university's buildings - Sahkotalo. Because it is a tall, 4-floor building , we expected to see the carrier being strong on the side of the building facing the BTS and weak on the opposite one and in the inner courtyards. We've picked the strongest carrier we could find to measure.

The results can be found on Figure 4.1. It can be clearly seen how the building shadows the signal. The picture even gives us a pretty good idea about the origin of the signal. Because it's a small area, the GPS inaccuracies start becoming apparent at this scale. To keep the map easy to read, the area around the building has been divided into squares and each square represents the average power within it's area. The map represents little under 3000 measurement points.

4.4 Large scale test

Once the small scale test was successfully completed, we could proceed to gather data over larger areas. We've put the drone inside a car, under the rear window so it could have a clear view of the sky. The antenna has a magnetic base so it was

Figure 4.1 *Graphical representation of the results, AFRCN 93: 956,3MHz*

very easy to install on the roof of the car. Unlike the previous test case, at this scale the GPS is pretty accurate so there was no need to divide the test area. The results can be seen on the Figure 4.2. The figure represents around 8500 measurements.

Because we've covered a much larger area, we've also reached a point where we could no longer distinguish the signal from noise. To be more specific this is defined as when the power gap between the active burst and the guard period is less than 30dB. This is represented by black color on the map. Similarly to the previous case, we've selected the carrier signal that was the strongest at out starting location. It's the small area where the signal is marked red. To the left of that place, there there is a residential area with a lot of tall buildings. We can see that the range of the signal is much smaller than to the right, where there is only single-family 1 or 2

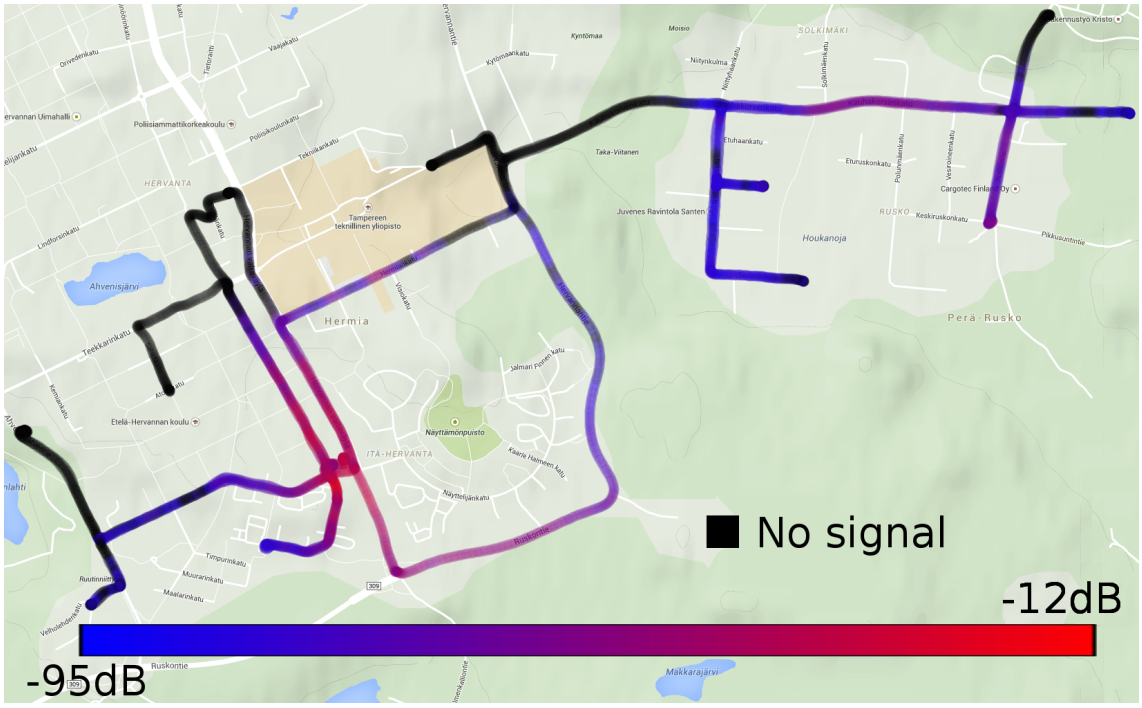


***Figure** 4.2 Graphical representation of the results, AFRCN 75: 950MHz*

floor houses or forest. The place to the far right is an industrial area with a few medium sized buildings and a few warehouses spread around it - As expected - not much attenuation can be seen from there.

5. CONCLUSIONS

In this thesis we have presented a device capable of performing remote, airborne RF measurements. It consists of a quadcopter with a GPS enabled FMU, BeagleBone Black SBC and an RTL-SDR based radio. In spite of minor drone issues, the project had reached it's goals.

The original idea of this project was to create a system working as a 'black box' where we have a set of geographical coordinates at the input and a map showcasing the strength of an RF signal of our interest at the output.

While we have to admit that we are not quite there yet, not all is lost. We are close. Obstacle avoidance is still a one of the primary areas of drone research. Unfortunately, improvements in that aspect are way beyond a scope of a master thesis nor were they the focus of this project and we couldn't make any improvements on that. Perhaps in a year or two we will start seeing systems that are not just autonomous, but autonomous and safe.

The drone we've had at our hands made it difficult to use the project in a useful scenario. X8 is designed for wide, open space areas. RF data collected from such areas isn't very useful because we already have reliable mathematical models for that and data collected from above the treetops isn't of much interest. It is also problematic to navigate without proper training and an onboard camera. However, the drone itself is just a small part of this project. All of the components can be removed and easily attached onto another drone and will work without any additional setup as long as the FMU supports the MAVLink protocol.

This, relatively simple project is just the tip of the iceberg for both SDR and UAV applications in telecommunications. Both of these technologies are under development and are yet to evolve into more capable versions of what we have at our hands now - Cognitive radio and environment aware drones. Both of them present many interesting challenges for development.

Possible future research in the direction set by this thesis would include expanding the measurement capabilities to include other wireless technologies, decoding the

GSM signals to gain access to the data it carries. Only with the hardware set-up presented over here, the possibilities seem endless. The list grows exponentially when we consider the possibility of installing a transceiver to be able to send signals instead of just receiving it. There are many SBC boards available on the market. Some of them even carry multi-core x86 or x64 CPUs so with enough payload we can also have a plentiful increase in processing power.

BIBLIOGRAPHY

[1] *3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Modulation (Release 11).*

[2] *3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Physical layer on the radio path; General description (Release 12).*

[3] *3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; Radio subsystem synchronization (Release 11).*

[4] Aerospace Industry Association. *Unmanned Aircraft Systems: Perceptions & Potential.* 2013.

[5] E. Blossom. "GNU Radio: Tools for Exploring the Radio Frequency Spectrum". In: *Linux Journal* (2004).

[6] *GNU Radio Manual and C++ API Reference Documentation.*

[7] L. Haeberle. *Airborne measurement for State-of-the-Art Acceptance Testing and Verification of Broadcasting Sites, Annual Meeting of Regional Broadcasting Organizations and Trasmitting Equipment Manufacturers.* 2014. URL: http://www.romkatel.ro/mbt/presentation/09.Luc_Haeberle-Colibrex.pdf.

[8] G. Kasper. *Phantom Eye backgrounder.* 2014. URL: http://www.boeing.com/assets/pdf/bds/phantom_works/docs/bkgd_phantom_eye.pdf.

[9] V. J. Mathews and S. C. Douglas. *Adaptive Filters.* 2003.

[10] L. Meier. *MAVLink Micro Air Vehicle Communication Protocol.* 2009. URL: http://qgroundcontrol.org/mavlink/start.

[11] L. Meier and A. Tridgell. *Python MAVLink implementation.* 2012. URL: https://github.com/mavlink/pymavlink.

[12] G. Moore. "Cramming more components on integrated cicuits". In: *Electronics* (1965).

[13] M. B. Sruthi et al. "Low cost digital transceiver design for Software Defined Radio using RTL-SDR". In: *Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on.*

[14] M Tessy and R Perkins. *Wireless Aerial Surveillance platform.* 2011. URL: http://www.defcon.org/images/defcon-19/dc-19-presentations/Tassey-Perkins/DEFCON-19-Tassey-Perkins-Wireless-Aerial-Surveillance-Platform.pdf.

[15] M. F. Torre. "Speakeasy - a new direction in tactical communications for 21st century". In: *Tactical Communications Conference, 1992. Vol. 1 Tactical Communications: Technology in Transition.*

[16] G. N. Varma, U. Sahu, and G. P. Charan. "Robust Frequency Burst Detection Algorithm for GSM/ GPRS". In: *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th.*

[17] H. Welte. *rtl-sdr Turning USD 20 Realtek DVB-T receiver into a SDR*. 2012. URL: `http://taipei.freedomhec.org/dlfile/rtl-sdr.pdf`.