RF Handover

Axel Cerin
Julia Blomgren
Felix Ljungkvist
Noah Wassberg

August 2024

Contents

1 Introduction

The aim of the 2024 RF summer project has been to develop a direction finding system, in order to identify the angle of arrival for incoming radio signals. To achieve this, a KrakenSDR, with 5 antenna mounts, was utilized. Furthermore, the MUSIC algorithm was implemented, which preforms the DOA approximation by comparing the phase differences of the signal for each antenna. It does this by creating a set of scanning vectors containing phase delays corresponding to different angles of arrival and then determining the correspondence between these vectors and the measured samples. A FIR filter was also implemented in order to reduce unwanted noise. Both linear and circular antenna configurations were implemented successfully. The testing was mainly conducted with signals within the 433MHz-band (equivalent to a car key), but other frequencies were used as well, yielding similar results. Many different antenna configurations were tested, where the linear and and especially the circular array were deemed the most successful. Another SDR, in this case the PlutoSDR, was also utilized to give a wider view of the signal environment for a wider frequency spectrum, which was visualized in a waterfall plot. Both the waterfall plot and the DOA result was the streamed to the Wara PS platform. Moreover, agents were used to be able to change the frequency of the waterfall plot while running and start and stop the video streamin from WARA PS.

2 Linear Antenna Configuration

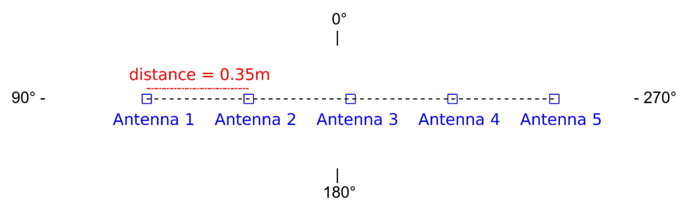

Figure 1: Linear Antenna Setup

The linear antenna configuration consisted of five equally spaced antenna elements. For optimal phase shift readings, the spacing between the antennas was set to either half or a fourth of the wavelength of the measured signal. Here, the center frequency was set to correspond with the frequency of the measured signal and antenna positions were specified as arrays of x- and y-coordinates. The MUSIC algorithm was then implemented and preformed on all antennas simultaneously. The resulting DOA approximation was then plotted on a polar graph.

This method was proved to be very precise at determining the angles of arrival for signals originating orthogonally to the antenna array, i.e. angles close to 0° or 180° in Figure 1. For signals origination at angles close to the endpoints of the array, the DOA approximation became much less clear and sometimes even unintelligible. Another limitation of this configuration was that the DOA would always return two possible angles of arrival, one corresponding with the actual signal and one mirrored signal at the other side of the array. This phenomenon stems from the configuration's one dimensional geometry, which makes it impossible to differentiate between signals originating from in front or behind the array.

3 Circular Antenna Configuration

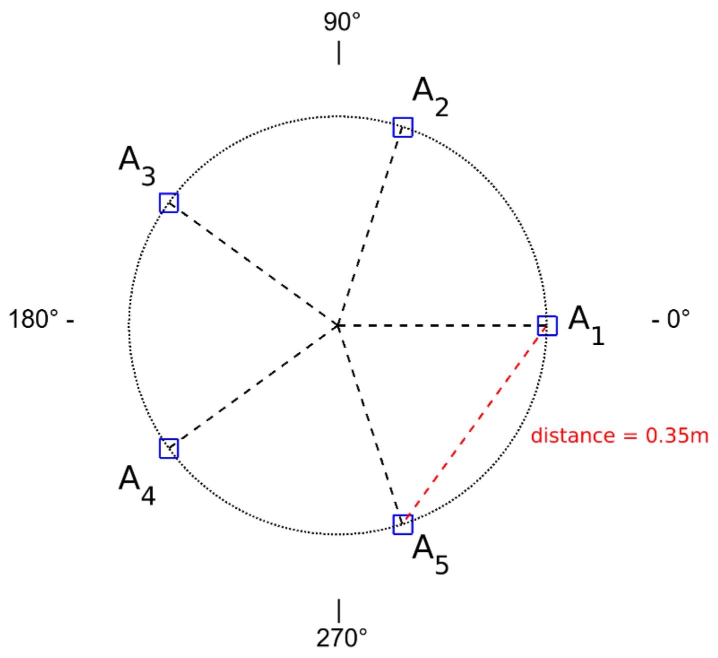

Figure 2: Circular Antenna Setup (distance adjusted for 433MHz)

The circular antenna configuration was implemented in order to overcome the limitations of the linear antenna configuration and thus give accurate DOA approximations for the entire 0° to 360° range. In this configuration, five antennas were placed in a circle so that the distance between each antenna and the next was half of a wave length for the received signal (see Figure 2). The antenna coordinates were specified and the MUSIC algorithm was implemented in a similar way to linear antenna configuration. The scanning vector for this configuration had to be generated in a slightly different manner to account for its circular (multi-dimensional) geometry. The resulting DOA approximation gave full range coverage and acceptable precision for all angles.

4 Unfinished Concepts

4.1 Circular Configuration With Divided Sectors

The circular antenna configuration was difficult to implement and while was still under development, it showed lack of precision. To increase this precision another method was implemented that drew from the strengths of both the circular and linear antenna setups. This method used the same circular antenna configuration and utilized it's direction finding implementation as a broad approximation. The DOA-angle was then used to determine which pair of antennas within the circular configuration had the most optimal position in relation to the signal. In other words, it determined which antenna pair was the most orthogonal to the signal's DOA.

The MUSIC algorithm was then used once more, but this time only considering the most optimal pair of antennas, which gave a much more precise measurement. This effectively divided the 360° range into ten different 36°-wide sectors were different pairs of antennas operate. The information from the broad approximation was also used to eliminate any mirrored angles in the precise measurement, leaving only one angle remaining. This method proved to be rather successful at giving precise approximations for the entire 360°- range and switching between sections also happened rather seamlessly.

When the MUSIC-implementation and the generation of the scanning-vectors were properly adjusted for circular antenna configuration, this makeshift solution was no longer needed. However this concept was interesting enough that it would be a shame not to mention it here and it is possible that similar methods may be of use in the future. The corresponding program has been named "kraken_heimdall_star.py" and can be found in the "srckrakenmisc" -folder. To run this program, run the start_doa_star.sh" -file, while in the "/misc" -folder.

4.2 Cross-Shaped Configuration

In order to develop a direction finding system with 360°- range, another method was briefly tested in parallel to the development of the circular antenna configuration. This method involved placing the the five antennas in a cross- or X-shape, and then preforming two separate DOA approximations, one with each line of antennas. The idea was then to combine these approximations to cover a wider range of angles by letting the two linear arrays cover each others weak spots. This worked alright for most angles, but for angles directly in between the two arrays the approximations were still unsatisfactory.

This concept was abandoned when the circular antenna configuration started showing promising results, but it is possible that this method of using two separate DOA approximations simultaneously, could be of use in the future, for example when developing DOA approximations in multiple dimensions. The script made for this configuration has been named "kraken_heimdall_cross.py" and can be found in the "src/kraken/misc" -folder. To run this program, run the "start_doa_cross.sh" -file, while in the "/misc" -folder.

4.3 Linear Configuration With Triangulation

In order to determine not just the angle of arrival, but the position of the transmitter, triangulation could be used. Such a solution was briefly attempted by using a linear array and letting the left and right halves of the antennas preform DOA approximations individually. These two angles would then differ from each other based on the distance between the two antenna groups. The idea was to then find the intersection between the corresponding DOA vectors and with this information derive the distance and position of the transmitter. Unfortunately none of the methods tried to calculate this were successfully and the transmitter distances were incorrect.

The cause of this was somewhat unclear. It might have been because the measurements were to imprecise to derive proper triangulation, maybe because the distance used while measuring were very short, or maybe there was a mistake in the calculations. Either way this side project was abandoned to make time for functions of higher priority. The script made for this configuration has been named "kraken_heimdall_tri.py" and can be found in the "src/kraken/misc" -folder. To run this program, run the start_doa_tri.sh" -file, while in the "/misc" -folder.

5 Repo Structure

The root of the repo contains a src folder, files related to docker and git and a start scripts. Within the src folder there are more folders corresponding to each of the software defined radios that has been used. Among these the KrakenSDR has been the main focus and is further explained in the KrakenSDR section (7). In the kraken folder another folder named "heimdall_daq_fw" can be found. This folder contains data acquisition firmware that is explained in the heimdall daq subsection (7.1).

6 Docker

For the KrakenSDR and PlutoSDR code docker is used. In the root of the repo there exists a docker compose file defining the different services.

The KrakenSDR Dockerfile uses an official miniconda base image as well as mamba for faster environment creation. The docker file downloads all required linux packages and builds KFR (DSP library for x86_64) from source. If a ARM processor is used Ne10 should be used instead but this has not been tried or added to the Dockerfile. The environment.yml file handles the needed python dependencies. After the repo and submodule has been cloned and built (see build guide) the KrakenSDR direction finding is started by running "./start_kraken.sh" in the root of the repo.

The PlutoSDR Dockerfile is very similar to the KrakenSDR DockerFile. It also uses an official miniconda base image and mamba for faster environment creation. The environment.yml file handles the python dependencies. The overall setup is simpler since there is no need for synchronization. To run the PlutoSDR spectrogram there is a start script in the root of the repo named "start_pluto.sh".

7 KrakenSDR

The KrakenSDR is a software defined radio with 5 RX-channels based on RTL-SDR. "kraken_heimdall.py" is the main file for the KrakenSDR and performs the data acquisition, signal processing and plotting. PyQtGraph which is a plotting library that works particularly well for real-time applications is used to create the graphs[10].

To change the parameters of the KrakenSDR the config.py file is used. The reason for this not being in the "kraken_heimdall.py" file that it has to be ran before the data acquisition firmware is started and the firmware has to be started before running "kraken_heimdall.py".

The MUSIC algorithm and other related functions can be found in the "direction_estimation.py" file. These functions are mostly PyArgus[5] code optimized with Numba[9]. Numba translates Python functions to optimized machine code at runtime using the LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN. Numba supports a subset of Python and Numpy which means that there are restrictions to what methods, functions and classes that can be used.

The "iq_header.py" and shmemIface.py files are modified from the official kraken repo[8]. These are used to initialize and read from the shared memory that the data acquisition firmware uses.

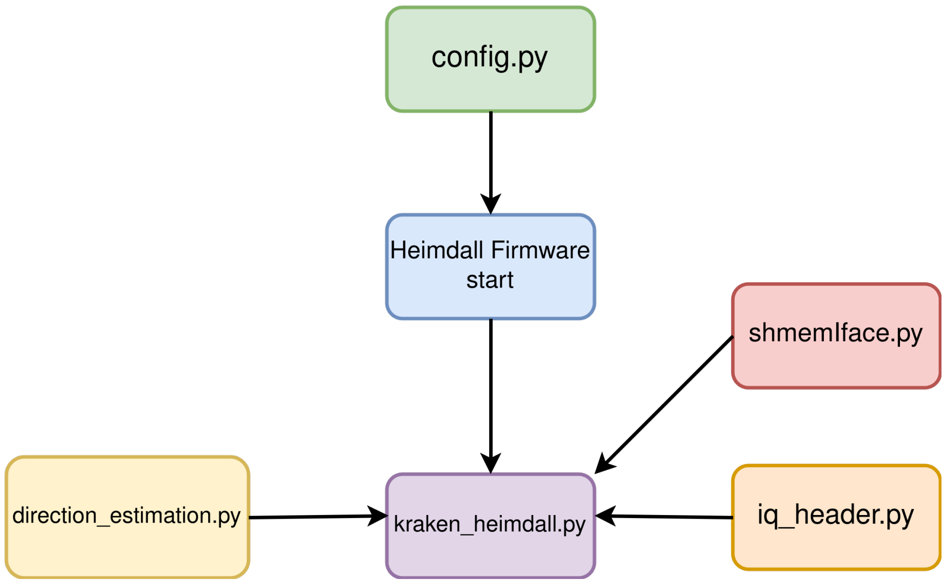

Figure 3: Graph illustrating workflow of KrakenSDR code

7.1 Heimdall Data Acquisition Firmware

The KrakenSDR makes use of five custom RTL-SDR circuits consisting of R820T2 and RTL2832U chips. The Heimdall Data Acquisition Firmware is developed by the Kraken team in order to synchronize the five input channels of the SDR[6]. Without the synchronization it is not possible to make accurate direction of arrival estimations. Heimdall uses the ports 5000 and 5001 to control the hardware e.g changing center frequency or gain during runtime. The RF repo makes use of a fork of Heimdall as a submodule.

8 PlutoSDR

The PlutoSDR is a SDR released by Analog Devices. The PlutoSDR used in this project uses a AD9361 Agile Transceiver. It is capable of receiving signals from frequency 70 MHz to 6.0 GHz and transmit from 47 MHz to 6.0 GHz. The bandwidth can be between 200 kHz to 56 MHz The revision C version that has been used in this project is capable of having two receive channels and two transmit channels.

To get started with the PlutoSDR there is a quick guide on analog devices page[3]. The docker file should also contain all necessary components to be able to run the program.

The PlutoSDR has been used to create a waterfall plot, send it to WARA PS and in the 2022 Arena Map/Atlas change the center frequency of the waterfall plot. All files related to how to do it is placed in the plutosdr-folder in the RF repo. The code is written in python and uses adi for the PlutoSDR. The library can be installed through "'pip install pyiio-adi"'. Down below is the main components of the waterfall explained.

8.1 Waterfall Plot

The Waterfall plot is created by performing Fast Fourier Transformation(FFT) on large arrays of data. In our code each array consisted of 65536 (2^{16}) samples. After a segment has been transformed from the time domain to the frequency domain through the FFT, the segment is shifted to place the zero frequency component in the middle of the spectrum. Then the magnitude of the complex values from the FFT and the magnitude's power is calculated. This way the power(or energy) at each frequency component is obtained. To be able to visualize the spectrum the base-10 logarithm of the power spectrum and then the multiplication by 10 is performed. The get the real life plotting of the plot pyqtgraph and pyqt is used. A new array of samples is collected through the PlutoSDR everytime the update_plot()-function is called and then the procedure above is performed.

8.2 Branch Pluto

The pluto branch contains some code for when the Plutosdr has been used for direction finding. You can look at it for inspiration, however it is not done yet. It also contains implementation using the soapy library. The soapy library is a "generic" library that can be used for a lot of different SDRs, the PlutoSDR and Kraken included. However, it did not work for the Kraken as intended and Pluto's own library adi was easier to use so we did not use the Soapy library. The branch also contains information about how to write the received signals from the SDR:s to file. You can check out the file write_to_file for an example on how to do that.

9 Introduction to WARA PS

WASP Research Area of Public Safety[11] or *WARA PS* for short, provides a suite of tools for research and development within the public safety domain. Their focus lies primarily in machine learning as well as autonomous systems interacting together with humans, adapting to and learning from their environment, forming intelligent systems-of-systems. WARA PS works by providing building scenarios where

9.1 WARA PS Agents

WARA PS offer agents that can do a variety of different tasks. These agents connect to WARA PS through a mqtt-broker[1]. MQTT is a lightweight open source messaging protocol used for machine-to-machine communication. It uses a publish and subscribe pattern. [4].

By combining an agent's code with the waterfall's and the fact that the PlutoSDR can change center frequency while it is running it is possible to change the center frequency of the waterfall plot using the WARA PS 2022 ARENA or the Atlas platform. The agent we have implemented is called PlutoSDR and have a task called change-frequency. However, the task is only executable on the atlas-summer.waraps.dev platform(this may have changed if you are reading this in summer of 2025). If you want to perform the functionality on the WARA PS Arena you have to "cheat" a bit. If you want to implement your own task an agent on the WARA PS platform, WARA PS need to add this task to their data model, so we cannot create new tasks soulely in our agent. We need some support from the WARA PS platform. The new tasks have been added to the atlas-summer.waraps.dev but not in the 2022 Arena. However, we can use already existing tasks in their data model and edit how our agents perform the tasks so we get wished behaviour. For example, we can use the task "move-to" and the pretend that he taks's altitude parameter is our frequency in MHz. The task start-stream is also implemented in the atlas-summer.waraps.dev and will start a stream to WARA PS. However, the task stop-stream is not implemented because it does not exist in WARA PS data model yet. In the future WARA PS will implement a task making it possible to stop the stream.

WARA-PS also have a lot of good examples of how to implement agents in different languages, such as python, C++ etc., on github: `https://github.com/wara-ps/waraps-agent-examples/tree/master`

To connect an agent to the WARA PS platform you need to connect to WARA PS broker. In the file .env in the plutoSDR-folder there is both the possibility to connect to a local broker and to the WARA PS broker. To connect to the WARA PS broker you need an username and a password. Check with your supervisor or WARA PS to obtain these.

9.2 Streaming to WARA PS

WARA PS holds an *Real Time Messaging Protocol (RTMP)* server where it is possible to stream video to. They provide the address "*rtmp://ome.waraps.org/app/your-stream-name*" where *your-stream-name* should be replaced by a name to give the stream. This streaming capability has been used to send our visualizations to WARA PS.

In order to provide this functionality to our application, a library called

RtmpStreamer[7] was created. This library makes use of the GStreamer[2] library where it generates a pipeline to take input frames from the program and propagate them to the RTMP-Server.

One of the main functionalities is the ability to turn the stream on and off at runtime. In order to accomplish this, the streamer generates whats called **bins** where multiple **elements** of the pipeline are stored. Each element performs some form of operation on the data in order to prepare, send or display the sent frames. By dividing the related elements up into bins, we are able to decouple the bins that should not be active at any giving moment. We are then able to integrate this functionality into the waraps agents capabilities, making us able to turn on and of the stream via the WARA PS toolset.

The library has been written in C++ in order to perserve the application speed while streaming. In order to run the library with the python application, python bindings have been created. The streamer is automatically installed in the docker image for both the Direction finding and waterfall plotting.

All the library functions of the *RtmpStreamer* library are documented and available in the library repo[7] header file "*include/rtmp.hpp*". There are also example programs for both *python* and *c++* under "*examples/*".

10 Future Work

- Use multiple direction finding approximation at different position and triangulate to determine the distance to and position of the signal transmitter. Triangulation experiments were conducted and corresponding programs can be found in the src/kraken/misc -folder. Here the linear array was divided into two sections, conducting individual DOA approximations. However the distance calculation was not successful despite many attempts and different methods being tried.
- Implement direction finding for several dimensions. This could be done through, for example, by running DOA approximations in multiple planes simultaneously. Finding the optimal antenna configuration for this task and combining the data from the different planes will likely be an interesting challenge.
- Further explore direction finding algorithms and preprocessing tech-

niques to improve performance. For example through decorrelation, methods, useful transforms, or faster DOA algorithms like Root-MUSIC. Increasing the number of samples per read or the resolution of the scanning vectors does not seem to lead to noticable improvements.

- Sync multiple SDRs in order to utilize even more antennas. The more elements in the antenna array the better the performance of direction finding algorithms. Synching multiple SDRs would allow for a bigger antenna array.
- Explore other antenna array setups which have desirable characteristics. For example Uniform planar, uniform rectangular or hexagonal array setups. It is likely that the "gen_scanning_vectors_linear()" function works for these array setups but it should be double checked.
- Automatically detect and set tune the center frequency to the most significant signal. However it may be hard to achieve good performance with the direction of arrival algorithms since optimal antenna spacing is related to the center frequency.
- At the time of writing this there is only build support for x86_64 cpu architectures. To get the project to run on AMD the build process would have to be modified.
- Build filters through firmware instead of using scipy. The heimdall data acquisition firmware directly supports filters. This has not been explored and could provide better performance.
- Create a RADAR. It is possible to use music together with a transmitter to create a Radar (see wikipedia)
- Evaluate the ability to detect multiple signals at once (MUSIC should be able to do this. The resolution of the direction finding of both linear and circular arrays as well as difference in amplitude among the singals can be evaluated. At the moment there is a naive approach to estimating the number of signals, improving this could help. The theoretical max number of signals that can be detected at once is 4 (for an array with 5 elements).
- See what happens when sending more signals than music could theoretically detect. For an antenna array with 5 elements the theoretical max is 4 signals.

- Mount transmitter on a drone. Drones have very complicated congestion control protocols leading to frequent switching in transmitted frequency. This makes it very hard to detect and track them. A first step to detecting drones could be to mount a transmitter sending on one frequency on a drone.

- Explore the possibility to create agents that can work together. For example mounting two AntSDR:s on a drone each and have them perform direction finding.

- Connect the waterfall graph with the direction finding algorithm. By using the waterfall graph to figure out which frequency to tell the Kraken to look at when direction finding.

References

[1] *Agent Communication*. URL: https://api.docs.waraps.org/#/agent_communication/agent_communication.

[2] *Free Desktop GStreamer Home Page*. 2024. URL: https://gstreamer.freedesktop.org/documentation/index.html?gi-language=c.

[3] Robin Getz. *PlutoSDR Quick Start*. 2021. URL: https://wiki.analog.com/university/tools/pluto/users/quick_start.

[4] *MQTT protocol main page*. Accessed: 2024-08-07. 2022. URL: https://mqtt.org/.

[5] Tamás Pető. *pyArgus: Antenna array signal processing library implemented in python*. https://github.com/petotamas/pyArgus. Accessed: 2024-07-29. 2024.

[6] Tamás Pető and Carl Laufer. *Heimdall DAQ Firmware: Coherent data acquisition signal processing chain for multichannel SDRs*. https://github.com/krakenrf/heimdall_daq_fw. Accessed: 2024-07-29. 2024.

[7] *Rtmp Streamer Library for dynamic streaming to RTMP server*. 2024. URL: https://github.com/acoustic-warfare/RtmpStreamer.

[8] KrakenRF Development Team. *KrakenSDR DOA Receiver*. https://github.com/krakenrf/krakensdr_doa/tree/main/_sdr/_receiver. Accessed: 2024-07-29. 2024.

[9] Numba Development Team. *Numba: A Just-In-Time Compiler for Python*. Version 0.54.0. 2024. URL: https://numba.readthedocs.io/en/stable/index.html.

[10] PyQtGraph Development Team. *PyQtGraph: Scientific Graphics and GUI Library for Python*. Accessed: 2024-07-29. 2024. URL: https://www.pyqtgraph.org/.

[11] WARA Public Safety. *WARA Public Safety*. Accessed: 2024-08-07. 2024. URL: https://wasp-sweden.org/industrial-cooperation/research-arenas/wara-ps-public-safety/.