UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES

University of Novi Sad
Faculty of Technical Sciences
Novi Sad, Serbia
Department for Computing and Automation
Software Engineering and Information Technologies
Section for Computer Science and Computer Communications

BACHELOR THESIS
(Translated from Serbian)

Candidate: **Jelena Banjac**
Index number: **SW 16/2013**

Title: **Developing Expert Tools for the LHC**

Mentor: **Prof. dr Miroslav Popović**
Co-mentor: **PhD Helga Timkó**

October, 2017

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		**Jelena Banjac**	
Mentor, **MN**:		Prof. dr Miroslav Popović (FTN mentor), PhD Helga Timko (CERN supervisor)	
Title, **TI**:		Developing expert tools for the LHC	
Language of text, **LT**:		Serbian (original)	
Language of abstract, **LA**:		English	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2017	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6, Serbia	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/59/24/0/36/12/0	
Scientific field, **SF**:		Electrical Engineering and Computer Science	
Scientific discipline, **SD**:		Computer Science	
Subject/Key words, **S/KW**:		**CERN, RF, LLRF, RF cavity, RF system, RF line, control system, transfer function, LHC, accelerator, algorithms, software architecture, developing, backup, environment, routines, testing, calibration, align phases, python, MATLAB, recommissioning**	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		**This Thesis describes a software tools developed for automated, precision setting-up of low-power level radio frequency (LLRF) loops, which will help expert users to have better control and faster setting-up of the radio-frequency (RF) system in the Large Hadron Collider (LHC) experiment.** **The aim was to completely redesign the software architecture, to add new features, to improve certain algorithms, and to increase the automation.**	
Accepted by the Scientific Board on, **ASB**:		**October, 2017**	
Defended on, **DE**:		October, 2017	
Defended Board, **DB**:	President:	**Doc. dr Ivan Kaštelan**	
	Member:	**Doc. dr Miodrag Đukić**	Menthor's sign
	Member, Mentor:	**Prof. dr Miroslav Popović**	

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my supervisor Helga Timkó, a staff scientist working at CERN, Geneva, Switzerland. The door to Helga's office was always open whenever I ran into a trouble spot or had a question about the project or writing. Sincere thanks to Bastosz Belawski for his motivation and support with the project matter. His input ideas for code improvements as well as code reviews are the of valuable importance to this project. Without their passionate participation and input, the project could not have been successfully finished.

I take this opportunity to express gratitude to all of the Radio-Frequency team members in the CERN's Beam Department.

I would like to express my gratitude to Philippe Baudhgarden, who always had patience to explain how the LHC LLRF system works and new routines implementation ideas.

I place on record, my sincere thank you to my mentor, Prof. dr Miroslav Popović, Professors at Faculty of Technical Sciences, University of Novi Sad in Serbia, for the sincere and valuable encouragement extended to me.

I also thank my parents, my sister and Darko Lukić for the unceasing encouragement, support and attention.

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF GRAPHS

ABBREVIATIONS

CERN - The European Organization for Nuclear Research

LHC - The Large Hadron Collider

RF - Radio-Frequency

LLRF - Low-power Level RF

LINAC2/3 - Linear Accelerator 2 and 3

PSB - Proton Synchrotron Booster

PS - Proton Synchrotron

SPS - Super-Proton Synchrotron

LEIR - Low-Energy Ion Ring

CMW - Control Middleware

RBAC - Role-Based Access Control

RDA - Remote Data Access

FESA - Front-End Software Architecture

CCDB - Controls Configuration Database

SNR - Signal-to-Noise Ratio

SLAC - Stanford Linear Accelerator Center

SoC - Separation of Concerns

IIR - Infinite Impulse Response (filter)

LTI - Linear, Time-Invariant (system)

CIC - Cascaded Integrator-Comb (filter)

MSE - Mean Square Error

1. Introduction

The European Organization for Nuclear Research, known as CERN, is place where physicists and engineers explore the fundamental laws of the universe. The ‘instruments’ used to study these laws are purpose-built particle accelerators and their detectors. The Large Hadron Collider (LHC) is the world’s largest and most powerful proton synchrotron. It consists of a 27 km long ring with thousands of superconducting magnets that bend and focus the beam and 16 radio-frequency (RF) cavities that accelerate the particles so that the trillions of particles circle the colliders tunnel 11245 times per second.

The LHC accelerates the protons to 6.5 TeV [1], corresponding to 99.9999989 % of the speed of light. This would not be possible without the electric field in the cavities alternating at high frequency. To get accelerated, the particles have to cross the cavities at a suitable phase. In a stable situation, the particles are then bunched around this phase [2].

Most of the RF system is located in one of the LHC’s straight sections, 150 m under the surface; part of the electronics is in a surface building. When the beam is present, the underground area cannot be accessed at all. Even without beam, but the RF cavities pulsing, only certain areas can be accessed. Therefore, there is a great need for remote controlling and diagnosing the system. Remote control tools are required both for operators for everyday operation and for system experts for (re-)commissioning and fault tracking if required.

Every year after the year-end shutdown, the low-power level RF (LLRF) control loops of the cavities have to be re-commissioned. This Thesis describes a software package developed for automated, precision setting-up of these LLRF loops, which will help users to have better control and faster setting-up of the system. Starting from an existing MATLAB implementation, the main goal of the work presented here was to develop a Python software package that is not only a translation of the MATLAB package. The aim was to completely redesign the code structure, to add new features, to improve certain algorithms, and to increase the automation.

In particular, we shall address the necessity to provide protection against unexpected program behavior by making a backup of the current system state. This backup is also valuable because it stores the history of device parameters (so-called properties and fields). Another main focus will be on the methodology of the algorithms used in the routines within the context of the overall system. The validation of the software package is presented in form of measurement results.

The structure of the Thesis is as follows.

Chapter 1 introduces in more detail the project challenge, explaining why the project was proposed and what the main goal is.

Chapter 2 introduces the basic theory of beam motion in synchrotrons and defines the concepts and building blocks of the LHC LLRF system.

Chapter 3 outlines a high-level description of how the solution will meet goals and requirements of the project. It describes algorithms and datasets used to determine whether the behavior of the system is well set up and optimized.

Chapter 4 presents the design of the software architecture as well as its implementation of in Python used in each of the routines for LLRF commissioning.

Chapter 5 evaluates the results from a test cavity as well as analyses and validates overall performance in simulated conditions.

Chapter 6 concludes by summarizing the current status of the project, the advantages and disadvantages of the implemented algorithms, closing with recommendations and plans for future work.

1.1 Purpose and Motivation

One of the project challenges was to create a suite of Python-based tools that for the nulling procedure consists of varying the device offsets and observing when the output of the system is the closest to zero. In this way, the system errors or deviations are minimized when offset is calibrated correctly. Another project challenge was to create a phase alignment routine that will align the phases between the analog and digital feedbacks in the LHC LLRF system. The same routine is also used to prepare the data for the overall open-loop measurement of the system, based on the alignment routines.

In addition, a new fine-alignment routine has been developed and implemented with the goal to fine-adjust the same phase on the operational system on a regular basis during technical stops.

All these routines are explained in more detail in the following Chapters.

2. Theoretical Background

In small, low-energy accelerators like tandem accelerators, particles can be accelerated over a static electric field. However, it is difficult to obtain a strong static electric field without sparks occurring. This is why most particle accelerators use oscillating electric fields instead. A suitably-shaped conducting structure can be excited with radio-frequency waves to resonate at its resonant frequency, just like a violin is excited with acoustic waves to make it sound, see Figure 1. Such a device is called an RF cavity, and its exact shape and size determine its resonant frequency. The resonance allows to create a strong electric field amplitude while inputting a relatively small amount of energy in form of radio waves.

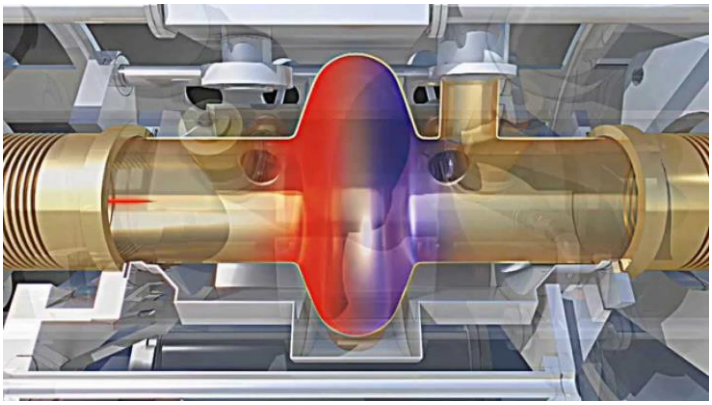


Figure 1: Schematic of the LHC single-cell RF cavity and its oscillating electric field [3].

The consequence of oscillating electric field is that particle beam cannot be continuous. According to the Lorentz force, charged particles are accelerated by the electric field and the direction of their velocity is changed by the magnetic field. In a synchrotron, the two fields have to be synchronized; the oscillation frequency of the electric field is usually an integer multiple of the revolution frequency, which in turn depends on the magnetic field:

$$f_{RF} = h\, f_{rev}\,,$$

where f_{RF} is the RF frequency (around 400 MHz in the LHC [1]), f_{rev} is the revolution frequency and h is the harmonic number. The revolution frequency f_{rev} and period T_{rev} are determined by the path of the particle along the ring and its relativistic velocity $v = \beta\, c$,

$$T_{rev} = \frac{1}{f_{rev}} = \frac{2\pi R}{\beta c},$$

where for the synchronous particle, $2\pi R$ is the circumference of the ring (26 659 m ~ 27 km in the case of the LHC [1]). The harmonic number in the LHC is 35 640.

Therefore, at constant energy, the synchronous particle crosses the cavity at a phase of zero or Pi (depending on the so-called transition energy), does not gain any energy, and returns to the cavity always at the same phase. To get accelerated or decelerated, the particle should see a phase offset in one or another direction around the phase of zero or Pi. Similarly, other particles that can have a slight energy offset w.r.t. the synchronous particle [4], will see a phase offset that decelerates them if their energy is too high, and accelerates them if their energy is too low. Thus, the particles remain concentrated around the "design" or "synchronous" phase and energy; we call this group of particles a bunch.

The phase-space region in phase and energy where particles remain bunched are called buckets, and there are 35 640 of them. However, the LHC design is such that only every 10^{th} bucket is filled with bunches, as the detectors trigger only at 40 MHz and require therefore a 25 ns bunch spacing. The highest number of buckets filled so far was 2556 (out of 3564 possible positions). The ensemble of all particles in a machine are referred to as beam. The beam injection and dump processes require a gap in the filling pattern, so some buckets have to remain empty.

The *beam dump* is a process that switches on the magnets that direct the beam from LHC machine into the dump. There is a beam injection process which requires *injection gap*. The beam injection and dump processes require a gap in the filling pattern, called the *abort gap*, so some buckets have to remain empty.

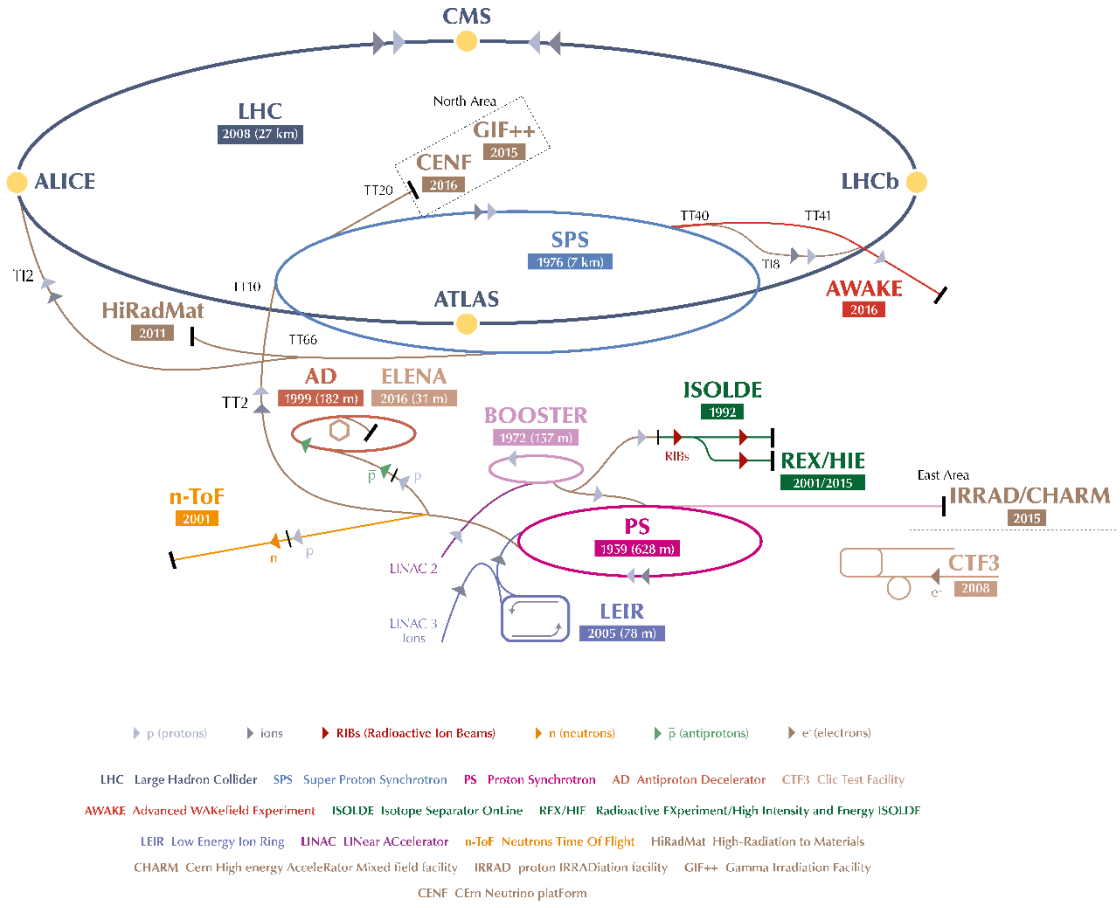


Figure 2: The CERN accelerator complex [5].

The injector chain of the LHC is shown in Figure 2. The LHC is supplied with proton beam through the following chain: LINear ACcelerator 2 (LINAC2) → Proton Synchrotron Booster (PSB) → Proton Synchrotron (PS) → Super Proton Synchrotron (SPS) and with ions from the injector chain LINear ACcelerator 3 (LINAC3) → Low Energy Ion Ring (LEIR) → PS → SPS. They are connected through so-called beam transfer lines [6].

Depending on the beam production schemes used in the injectors, the filling pattern (see Figure 3) and maximum number of bunches in the LHC can vary.

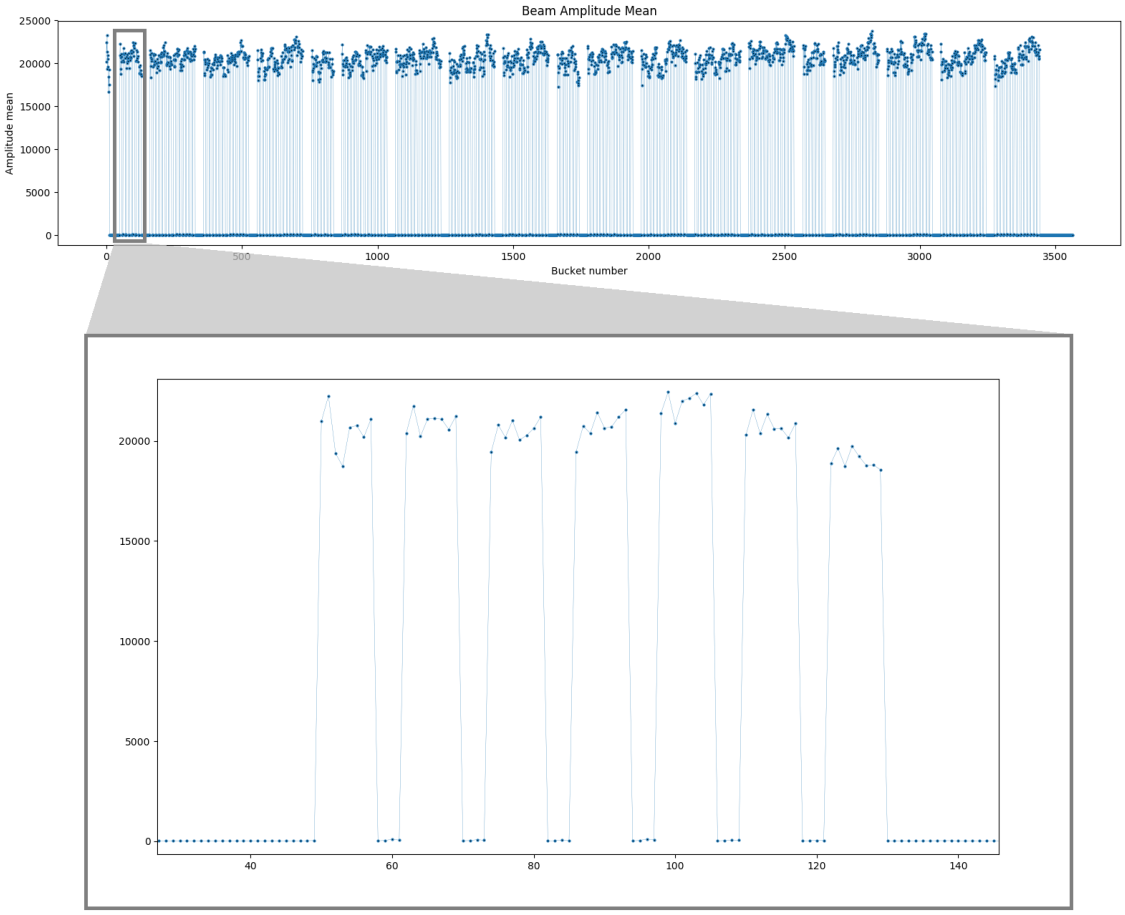


Figure 3: Example of filling pattern in the LHC. Beam voltage amplitude along 3564 buckets (top), zoom of the first batch (bottom).

2.1 LHC RF System

The LHC RF system consists of eight RF stations (or lines, see Sec. 2.1.1) per beam. The LHC has two separate beam pipes, in which the beams circulate in opposite direction. The beams are crossed and collided in four intersection points, where the detectors ALICE, ATLAS, CMS, and LHCb are located. The acceleration is done with eight single-cell, superconducting RF cavities per beam [7]. Four cavities each are put into one cryogenic module (see Figure 4), in which they are cooled down to 4.5 K [8].

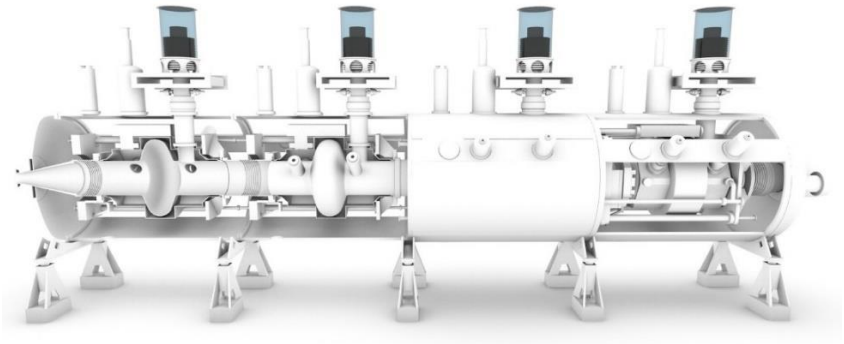


Figure 4: Model of the LHC cryogenic module containing four cavities [9].

The cryogenic modules are installed in the LHC tunnel directly in the beam line. In a cavern behind the cavities (see Figure 5), are the sixteen klystrons (one for each cavity) that produce the RF power. The output was specified in the LHC Design Report to be 300 kW for most klystrons, however, they only deliver about 270 kW [1]. The power is fed into the cavities through so-called wave-guides. The cavern contains furthermore high-voltage bunkers that supply the klystrons and two Faraday cages that contain the electronics controlling the cavities.

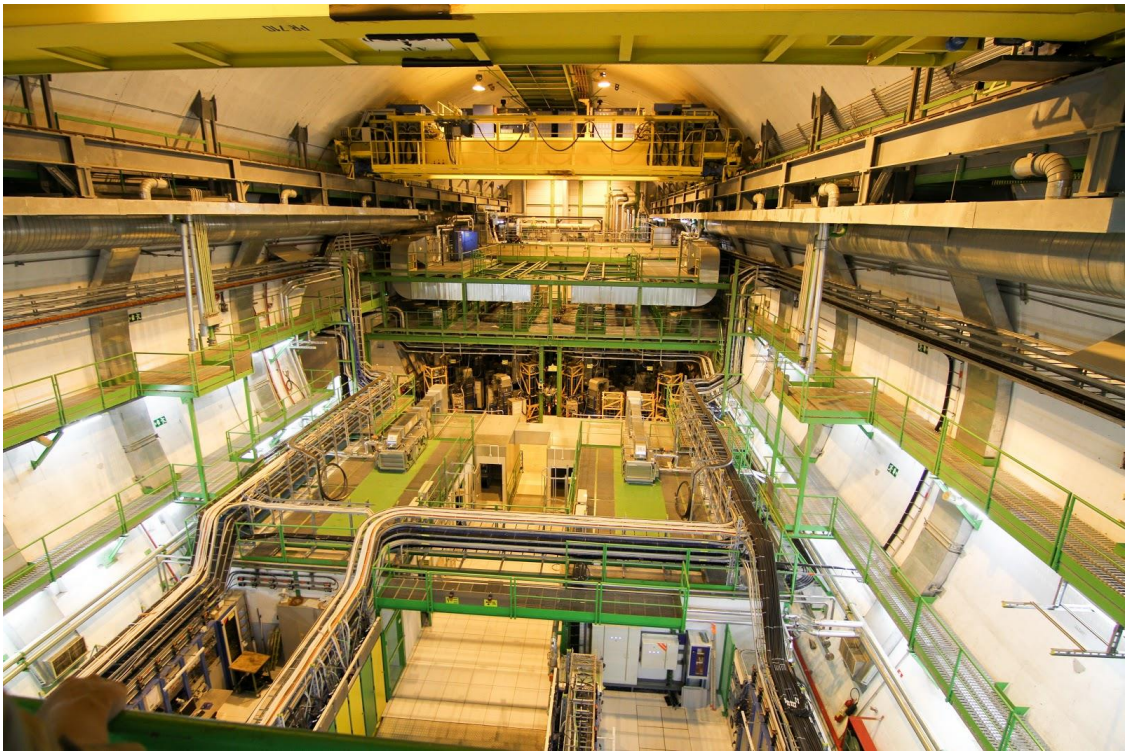


Figure 5: LHC point 4 cavern containing the RF equipment. The cavities are in the tunnel behind the cavern, shielded by concrete blocks. Photo by Bartosz Bielawski, CERN.

2.1.1 The LHC RF Cavity Feedback and Line

The voltage amplitude and phase in each cavity is controlled by an individual cavity controller (see Figure 6). The RF cavity and its high- and low-voltage equipment is referred to as "line". The beam and the cavity controller are two coupled, dynamic systems and both the beam and the feedback can become unstable for different reasons. It is therefore important to commission the cavity controller with stable settings and sufficient stability margin.

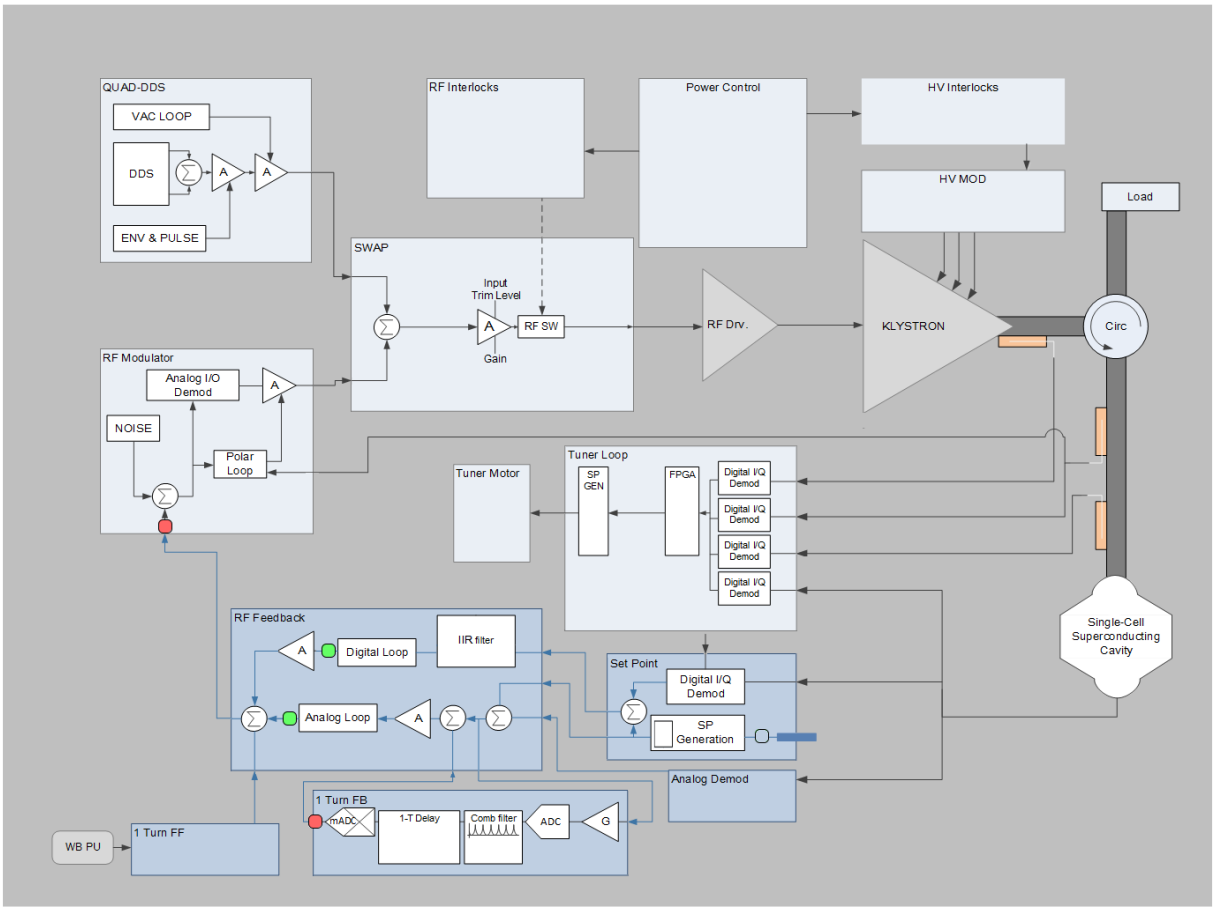


Figure 6: Schematic of cavity controller.

Each line consists of a cavity, klystron, and its LLRF electronics (VME modules) including the RF modulator and klystron polar loop, the switch and protect module, as well as the RF feedback, the one-turn feedback, the set-point and RF tuner modules.

The klystron polar loop acts around the klystron to reduce power supply perturbations and compensate the gain and phase shift of the nonlinear klystron at low frequencies for different operation points.

The switch and protect module clamps the RF power, should the demanded power be beyond the value where the klystron saturates. The digital and analog RF feedbacks reduce the effective cavity impedance at its fundamental frequency, improving the stability of the beam. The one-turn feedback reduces the cavity impedance at the so-called synchrotron frequency side-bands. The set-point module sets the demanded RF voltage and phase, which

can be user-defined or can come from the central distribution of functions, controlled by the machine operators. The tuner module controls the mechanical movement of the cavity tuner that adjusts the resonance frequency and the cavity coupler that regulates the power coupling between the power source and the cavity.

2.2 Development (Control) Environment

In this section, the development environment as well as resources that helped during the tool's development are discussed in greater detail.

The Controls Middleware (CMW) framework provides the communication infrastructure for all CERN accelerators [10], enabling client applications to connect, control, and acquire data from all types of equipment. The CERN controls infrastructure including the CMW and the application layers of Python, MATLAB, and Mathematica applications is shown in Figure 7.

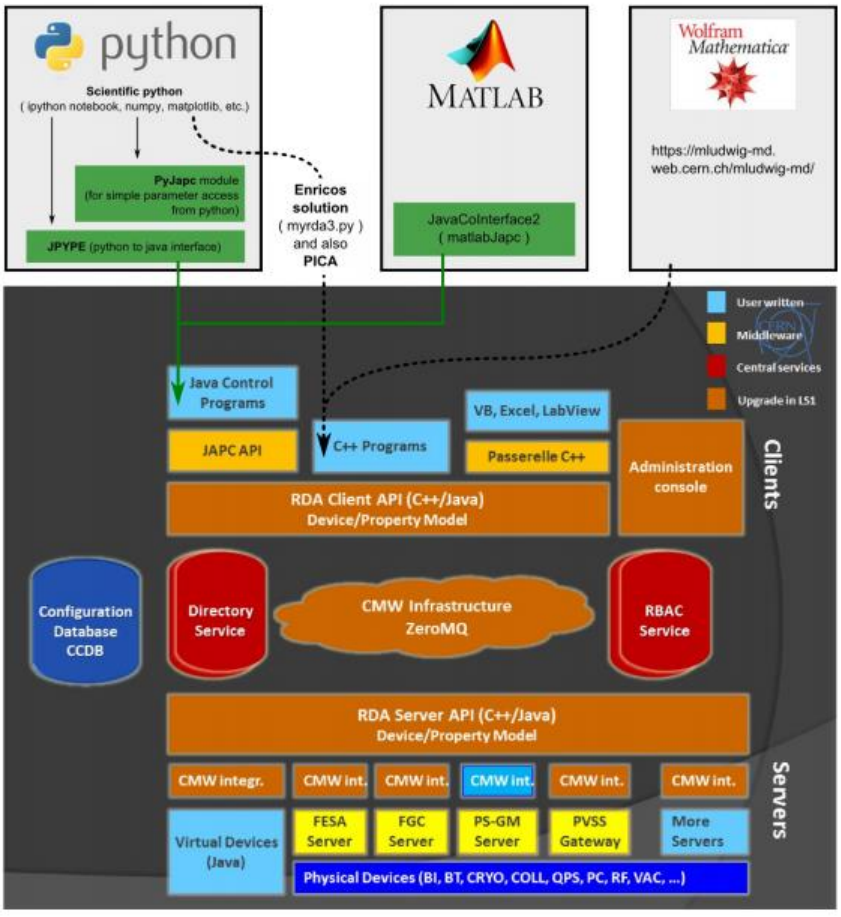


Figure 7: CERN controls infrastructure including the Controls Middleware (CMW) and the application layer [11].

Python application uses Python interpreter and modules that are provided by the Anaconda distribution with extra packages, including CERN-made PyJapc module [12], in order to access the CMW devices. The PyJapc module is used to read and/or write device

parameters from Python. However, since the remote data access (RDA) client side of the communication is implemented in Java and C++, it is necessary to bridge Python and Java applications. This bridging is done using the JPYPE module which is Python to Java interface. Using JPYPE, PyJapc then wraps the Java API for parameter control (JAPC) and role-based access control (RBAC). The JAPC provides parameter control; whereas RBAC is used to restrict system access to authorized users only.

The RDA client as well as RDA server, that are implemented in Java and C++, are using a Device/Property Model to name parameters. The transfer protocol between the RDA client and the RDA server is the ZeroMQ protocol that is implemented in the CMW infrastructure. ZeroMQ is a high-performance asynchronous messaging library developed for distributed or concurrent applications.

On the server side, only the front-end software architecture (FESA), developed at CERN [13], is running to publish data. The FESA server provides a set of actions for the device properties implementing the get, set and/or subscribe services. FESA is used to run soft real-time software[1] on front-end computers to control the accelerator's hardware equipment such as beam instrumentation, cavities, magnets, etc. The control system consists of named devices (e.g. ALLSetPoint5b0t, ALLSetPoint5b0f) that represent the instances of device classes (e.g. ALLSetPoint). Each device class is composed of properties (e.g. phases, status, ...) that can be operated with set, get, and/or subscribe actions, which can be executed from a remote server.

The controls configuration database (CCDB) represents another crucial part of the CERN accelerator complex control system: it collects all the data of the accelerator equipment into one place. It is a database that the users can access whenever they need more details on configuration items and their hierarchy in the control system. Configuration items are ranging from 3000 front-end computers, 75 000 software devices [15], variety of hardware configurations, information on cards, cables, drivers, RBAC, etc. This is allowing remote controlling of the accelerators.

When a software model for an equipment is being implemented, first a device class model has to be created (in form of an XML document), which describes the device. Once the model is saved and processed in the CCDB, the device class instances are deployed on front-end computers. Also, the device instances that can be grouped into device classes are created and updated.

[1] Soft real-time systems are typically used to solve issues of concurrent access and the need to keep a number of connected systems up-to-date through changing situations. Missing the deadline is not total system failure, just the usefulness of results degrades after its deadline [14].

Inspector [16] is a graphical framework implemented in Java that allows to control and to monitor any properties in a FESA class; an example is shown in Figure 8.

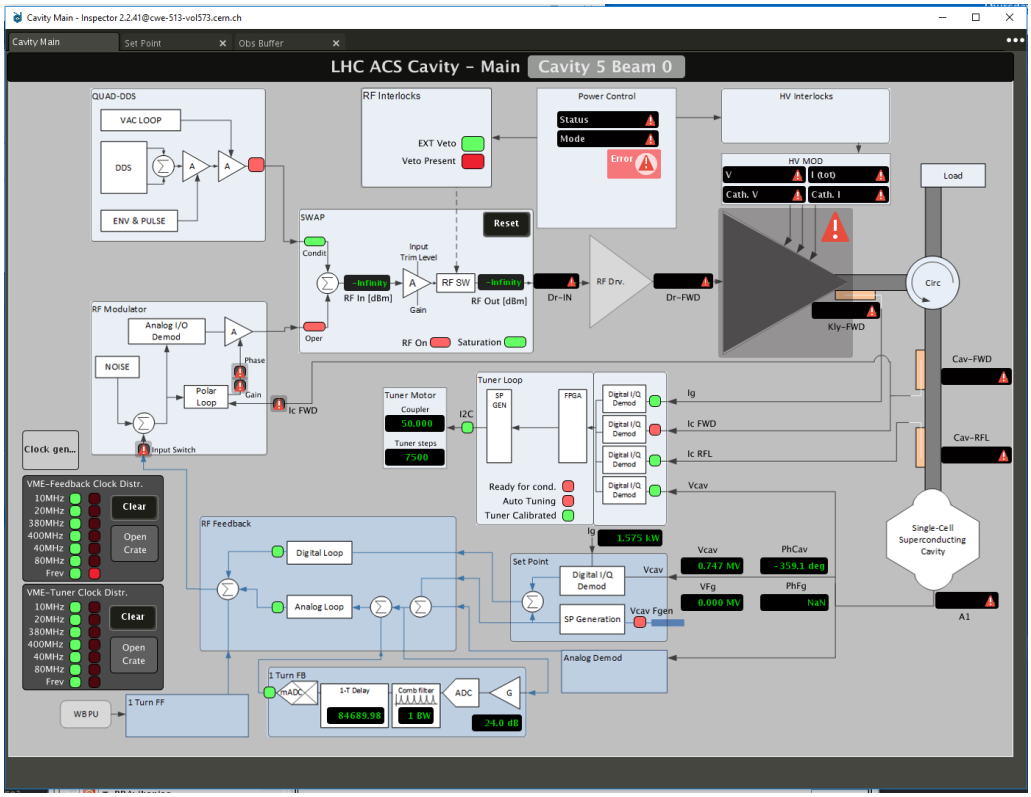


Figure 8: Screenshot of an Inspector panel on the LHC cavity controller for the test stand "Cavity 5 Beam 0"

Before and during the implementation of Python routines, some of their steps were tested with the help of the FESA Navigator[2], Inspector, and the CCDB web browser.

In some cases, for instance, it was necessary to verify whether a given sequence of steps was giving the desired output. Thus, a correct Python routine could be more easily achieved.

[2] The FESA Navigator is a Java application that allows browsing FESA properties.

3. Solution Concept

This Chapter will provide a high-level description of how the solution will meet the goals and requirements. Starting from the planning of a new software architecture, this Chapter will cover the implementation of the backup of all the devices currently available in RF line, as well as the storage of all the environment variables that are calculated during the execution of the routines. Finally, it will cover a description of the planned routine implementation, as well as the improvements and advantages in comparison to the previously used MATLAB tools.

Figure 9 visualizes schematically the idea of using backups and environment variables in the code implementation.

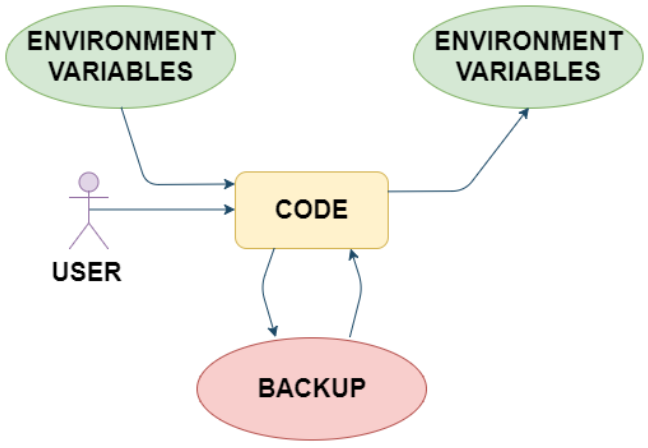


Figure 9: Schematic usage of backup and environment variables.

3.1 Backup

The backup is of fundamental importance to this work, as the initial state of the entire system should be easily restorable to ensure smooth operation. It also protects against failures or unexpected system behavior in the RF line and its sub-units. The main task of the backup is

to store and restore data, and in this way, to create an option for user to roll back to the one of the previous states of the system.

The backup has to be reliable for the user; the user's only concern should be to choose from which moment in time he/she wants to restore the state of the system.

In the entire software package, no routine should be implemented without an automatic backup of all the devices inside the RF line. In addition, the device parameters should be restored in the end of each routine and upload only those few new values to the RF line, which the routine is supposed to optimize.

3.2 Environment Variables

Some of the values obtained in the program execution process are not transmitted to the RF line, but need to be preserved. Part of these values represent residuals or calculation results obtained during routine execution, and part of them need to be used during the execution of future routines. Therefore, there was a need for the implementation of an environment variables container.

The environment variable represents a similar kind of a backup that was described in Sec. 3.1. The main idea behind environment variables is to organize them by hierarchy layers. The elements that have the same parent are considered as a one group of elements. For example, the values (elements) that are calculated during nulling (device calibration) routine will have the same parent. Thus, the container is flexible and the depth of the structure can be increased if needed. In other words, whole environment variable container can be represented as a tree of elements, and the element that is on a highest level of the hierarchy, referred to as root, represents the parent of all elements.

3.3 Routines

The routine is a set of instructions and calculations that are executed towards the same goal (e.g. calibrate the device), and can be grouped into one whole[3]. The previous implementation of this tool written in MATLAB, did not have a concept of a subroutine. The need for a subroutine appeared in order to split huge routines into smaller, logical parts (modules). Planned modularization is shown in Figure 10.

[3] In MATLAB, the routine was implemented as a script and in Python within a class.

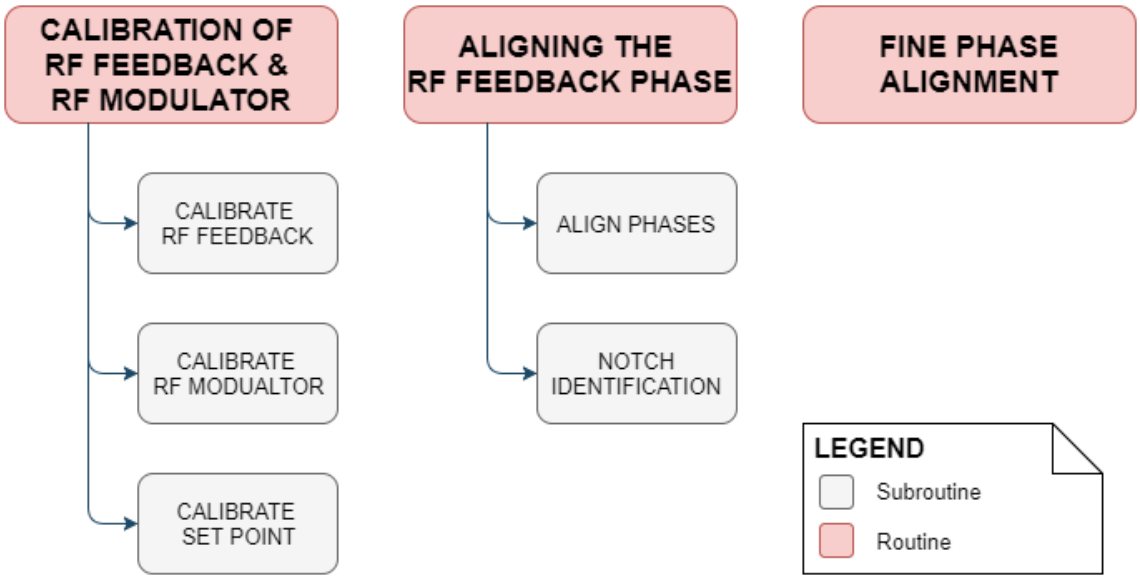


Figure 10: The modules (routines and subroutines) of LLRF commissioning tool.

Below, we provide a detailed description of each routine. It is explained why there is a need of their implementation, how the RF line should be set up before each routine execution and what is the desired outcome after the routine is executed.

3.3.1 Calibration (Nulling) of the RF Feedback and RF Modulator

The nulling procedure aims to minimize the impact of offset errors in the system. Through measuring the offsets as well as other characteristics of the system in closed loop, the corrections that need to be applied can be calculated. After setting the offsets to the optimized values, the system is calibrated and prepared for the routines that follow.

In the beginning of the nulling routine, a re-cabling between some modules is needed in order to bypass part of the system. The signal going out from the Cavity to the Analog Demodulator and Set Point needs to be terminated. This is done by disconnecting the output signal from the Cavity and placing a 50 Ω to the input of the Analog Demodulator and Set Point module. This step is visualized in Figure 11.

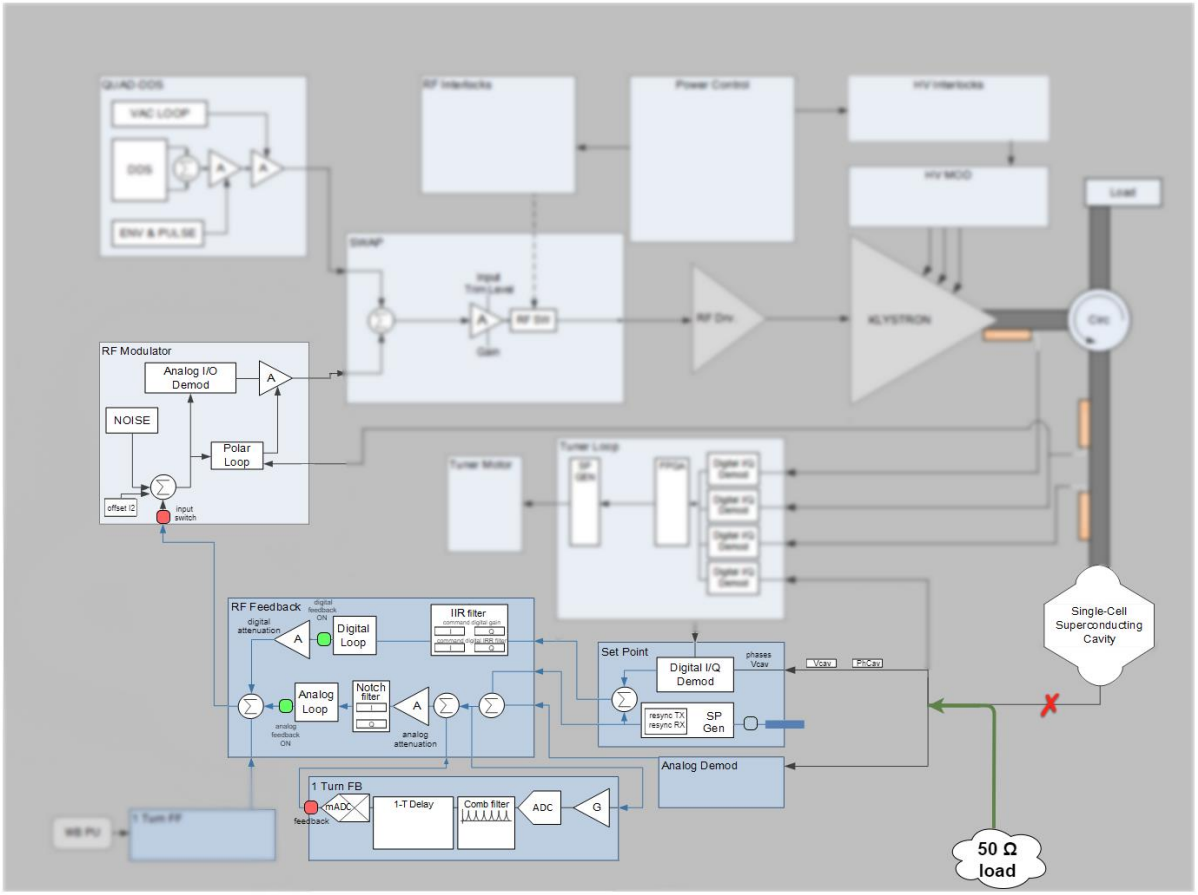


Figure 11: System configuration before the beginning of the calibration. The Cavity signal is disconnected and a 50 Ω load is placed instead to the input of the Analog Demodulator and Set Point.

The nulling routine consists of three subroutines: nulling of the RF Feedback, nulling of the RF Modulator, and a final check of the RF Modulator calibration in which a non-zero input level (offset) is set in the RF Modulator to see if it is possible to null it out. All three subroutines require the system to be in specific initial states, with certain settings applied, see Fig. 3.3. As a result, the cavity controller is prepared for further, more general tests including the overall open and closed loop response measurements.

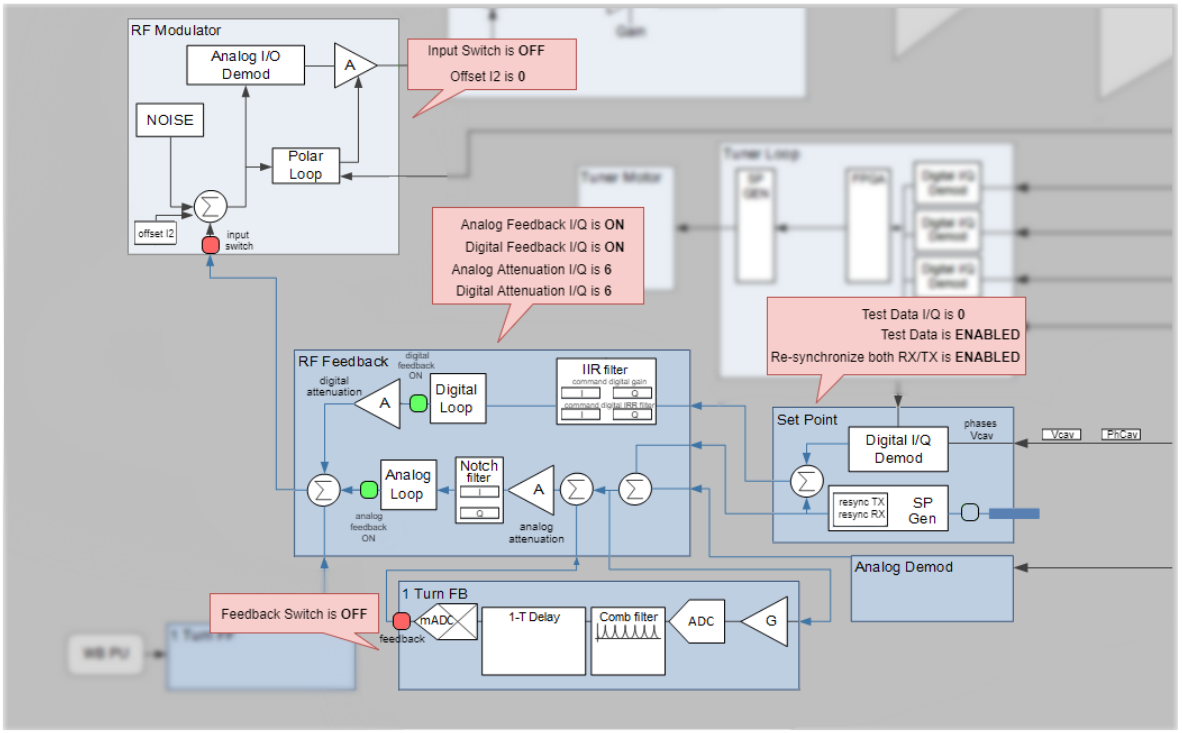


Figure 12: Initial conditions of the system for the nulling routine.

The detailed description of system state for the RF Feedback calibration can be seen in Sec. 4.2.3 part for Calibration of the RF Feedback.

The next step is the calibration of the RF Modulator. As a preparation, part of the system, amongst others the Switch and Protect module, the Klystron, and the Cavity, have to be by-passed. This is done by connecting the RF Modulator output to the input of the Analog Demodulator and the Set Point, see Figure 13.

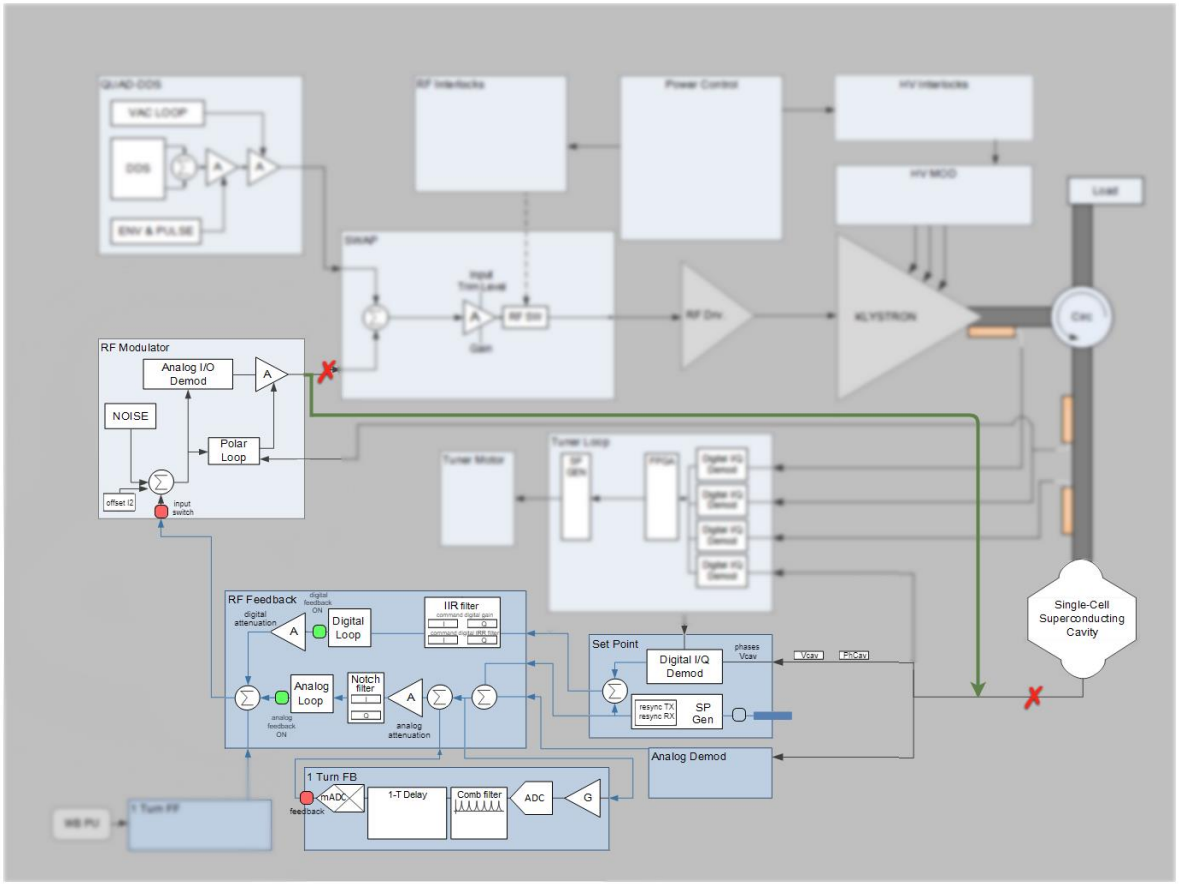


Figure 13: Preparation of the RF Modulator calibration; by-passing the cavity and the klystron by connecting the RF Modulator output to the input of the Analog Demodulator and the Set Point.

The subroutine for nulling the RF Modulator is also scanning the offsets, but this time for channels I (I2) and Q inside the RF Modulator module. For a good signal-to-noise ratio (SNR), the digital gain of the RF Feedback is lowered in the first, coarse measurement of the I and Q offsets. For the second, fine measurement of the offsets, the digital gain is increased. The detailed description of system state for the RF Modulator calibration can be seen in Sec. 4.2.3 part for Calibration of the RF Modulator.

In the third and last subroutine, the calibration is verified for a non-zero power level. The detailed description of system state for the last calibration subroutine can be seen in Sec. 4.2.3 part for Calibration of the Set Point.

3.3.2 Aligning the RF Feedback Phase

The input signal to the Set Point module is the antenna signal of cavity voltage vector ("V_{cav}") and phase of this signal (called "V_{cav} phase") is the property that is adjusted order to align the relative phase of the digital and the analog paths of the RF Feedback. First, white noise is injected in the excitation buffer of the RF Modulator. In order to determine what is the optimal value for the phase of V_{cav}, the I and Q channels of the test data inside Set Point module are varied and the response of the system is observed.

Similarly, as in the second part of the nulling routine (cf. Figure 13), the klystron and the cavity are by-passed also for this routine.

Afterwards, the system is prepared as illustrated in Figure 14.

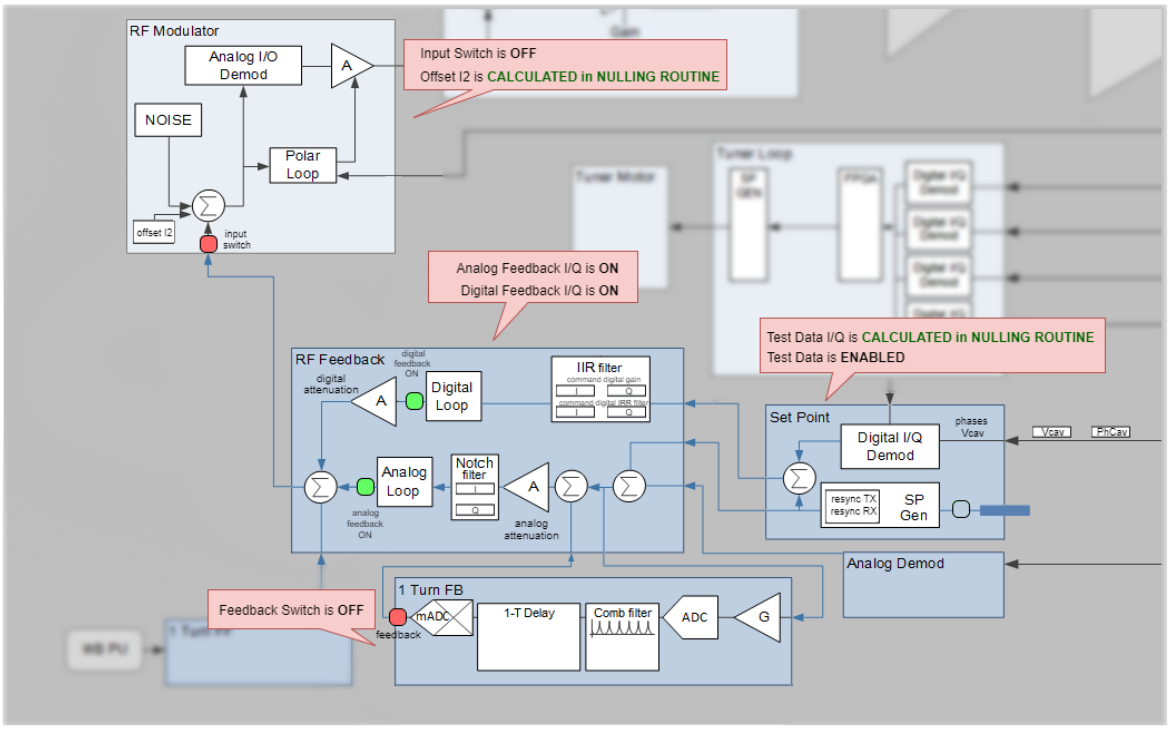


Figure 14: System configuration for the RF Feedback phase alignment.

First the cavity controller loop is opened at the input switch of the RF Modulator. Then the offset I2 in the RF Modulator and the test data I, Q in the Set Point module are set to the values calculated in the nulling routine (third subroutine). The analog and digital loop switches are closed in the RF Feedback module. In addition, if the One-Turn Feedback was closed, it will be temporarily opened. Additional details on phase alignment itself are covered in the Sec. 4.2.4 part for The Align Phases Subroutine.

The phase alignment routine also measures the transfer function of the Notch filter that is part of the RF Feedback module. The Notch filter is used to compensate the klystron resonance and the exact frequency of the notch is tunable via a 32-position variable capacitor. More on how the system is prepared for this routine is covered in the Sec. 4.2.4 part for The Notch Identification Subroutine.

In both subroutines of phase alignment, special steps were implemented in order to collect real data from system in a form of its transfer function (during the network analysis) as well as compare it to the estimated transfer function of the same system that is calculated using the responses of system's building units (during the transfer function fit). More information about these two algorithms could be seen in the Sec. 4.2.4 in the part for helper functions (HELPER FUNCTION: Network Analyzer and HELPER FUNCTION: Transfer Function Fit).

3.3.3 Fine Alignment

The fine alignment routine aims to fine-tune the RF Feedback phase alignment by running the measurement on the operational system. That means that the parts of the system that were by-passed in the phase alignment are now connected with the rest of the system. In addition, in this routine all data measurements and calculations are done only within the Set Point module.

The initial conditions in the Set Point module are summarized in Figure 15.

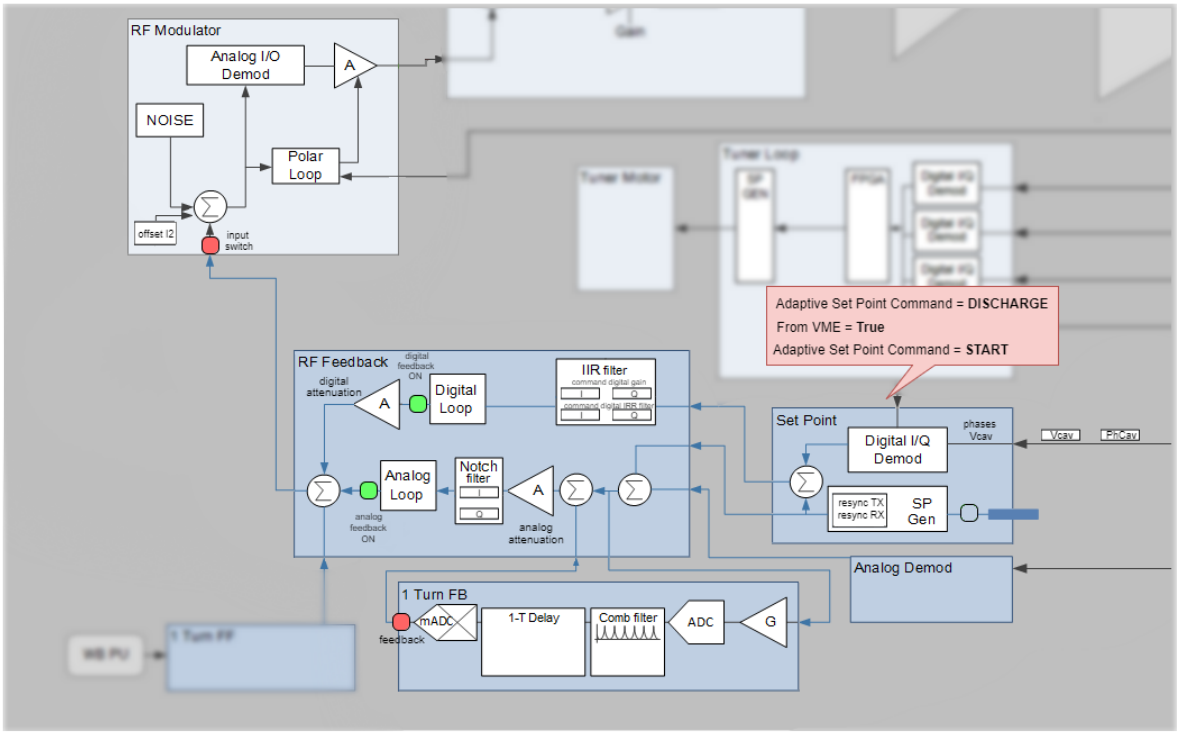


Figure 15: Initial conditions of the fine alignment routine.

Figure 16 summarizes the fine alignment procedure:

1) first, the “adaptive” buffer inside the Set Point module is filled with a phase modulation (step 1); in this Thesis, the default phase modulation (excitation) signal has a trapezoidal shape. This could be changed in the future according to the needs of the user;
2) afterwards, while the V_{cav} phase is varied in the interval of 10° with a step of 1°, starting with a value that is obtained via the phase alignment routine (steps 2 and 4);
3) the measured data is observed (steps 3 and 5) and analyzed in order to determine whether the value that is set for a V_{cav} phase could be the final one. In case the V_{cav} phase value is not yet good enough, another 10° are scanned in a similar fashion;

4) the procedure is repeated until the optimum value is found, which is then set as a final one (step 6).

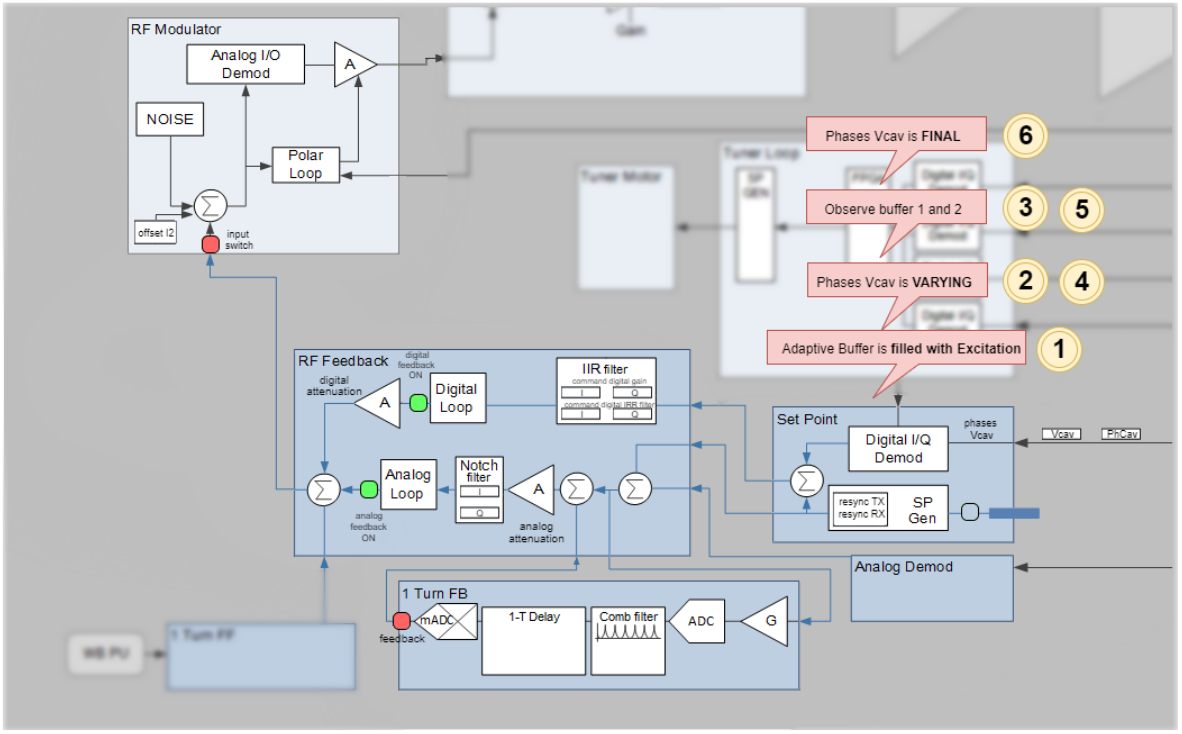


Figure 16: Procedure of the fine alignment.

In the end of the fine alignment routine, the Set Point module is returned to the state it was before running this routine, see Figure 17.

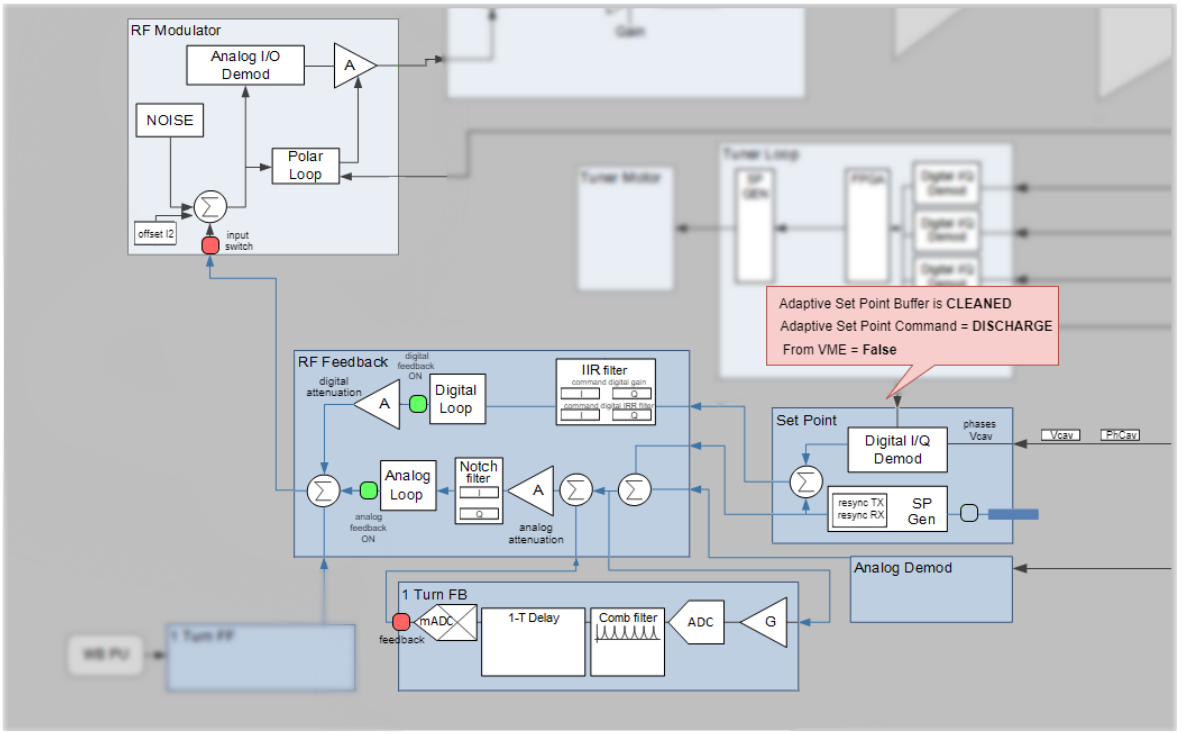


Figure 17: Restoring the Set Point module properties after the fine alignment.

4. Software Solution

The code implementation is described in this Chapter. The code flow of each routine will be illustrated, followed by a short explanation.

In Sec. 4.1, an overview of the general project organization will be shown. Section 4.2 will give a further information on what languages, frameworks and interfaces are used in order to successfully develop this tool. Afterwards, different modules of the code, including backup, environment variables, and routines will be described in greater detail.

4.1 Design

The development name of Python package containing this project is LHC-LLRF and is hosted on the CERN GitLab server under the address: `https://gitlab.cern.ch/htimko/LHC-LLRF/`.

The project structure of the project's root directory can be seen in Figure 18. The description is following:

- The backup directory (`backup/`) contains backup files considering whole RF system (inside `rf_system/` directory), environment variable files (inside `environment/` directory), and data from MATLAB scripts used for testing (inside `matlab_ws/` directory).
- The log directory (`logs/`) stores logging messages from the execution of the routines.
- The miscellaneous directory (`misc/`) containing MATLAB workspace (`.mat`) to JSON data (`.json`) converter; and functions for user interaction used when choosing the backup restoring point.
- The models directory (`models/`) containing responses (transfer function equations) of different control system parts - classes: RF cavity, RF feedback, klystron, notch, phase; parent class: transfer function.

- The modules directory (`modules/`) containing the class representations of devices placed in one RF line. Each class contains list of properties that are characteristic to that device. This would later be used to populate getters and setters of properties (and fields) of a given device.
- The plots directory (`plots/`) contains a custom class for plotting different measurements, since the output should be uniform in all program parts. Due to the need for having more clarity and separation of main functionality from plotting, all plots creation is placed in this directory.
- The modeled PyJapc or PyJapc simulator (`pyjapc_simulation/`) is created for purposes of testing the written code while developing, that is related to the backup's and device's methods (store and restore).
- The RF system directory (`rf_system/`) is the most important directory of this project. In this directory main classes that represent RF station (line) and devices are implemented together with their backup module.
- The rollback directory (`rollback/`) represents rollback of the system to the previous state manually by user, not routine itself. This is planned to be implemented in the future.
- The routines directory (`routines/`) contains the subdirectory for each implemented routine. Every subdirectory contains the main steps that are specific for each of the routine. Also, there is a subdirectory which content is shared between all routines (environment, argument parser, common steps for communication with hardware).
- The test directory (`tests/`) has tests checking the algorithm implementation, as well as checking the backup implementation with the real and modeled PyJapc.

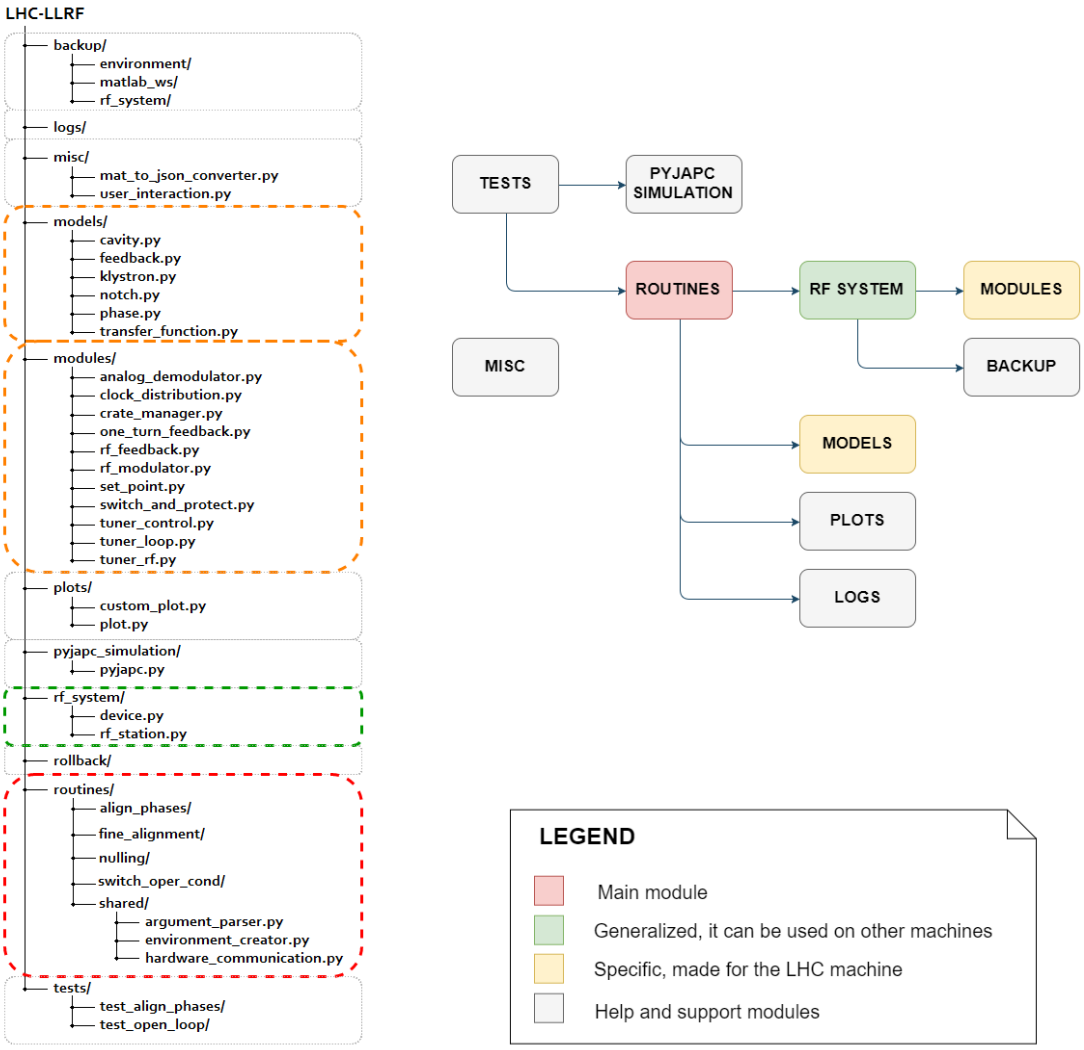


Figure 18: Project structure of the LHC-LLRF python package.

4.2 Implementation

The project is implemented in the programming language Python 3.6. The tools previously used to perform the same task were written in MATLAB in the framework of a collaboration between CERN and SLAC (Stanford Linear Accelerator Center) National Accelerator Laboratory.

First, the development environment as well as resources that helped during the tool's development are discussed in greater detail. Concrete solutions will be analyzed afterwards and the improvements will be highlighted.

4.2.1 Backup

The backup module is implemented both for devices and RF lines. The main goal was to create it as flexible and general as possible, so the backups can be applied also for machines other than the LHC in the future.

Each backup object consists of the device name (in case of device storing) or RF line identification (in case of RF line storing), the timestamp at the moment in which the backup was made, the selector that describes the moment in time an operation (`GET` or `SET`) should act (an example of a valid selector is '`LHC.USER.ALL`'). It stores furthermore the filename under which the backup will be stored and a dictionary of device properties (in case of device storing) or a dictionary of each device backup from a specified RF line (in case of RF line storing).

Methods for storing and restoring devices (or RF lines) are implemented using Python's JSON package in order to store the object in a form of a JSON file. In a first attempt, the Python package pickle was used which serializes Python objects into files in binary format. However, for the user to be able to easily read the file content, this solution was replaced. The solution can be changed back or we can add support for more formats due to the fact that this project embodies separation of concerns (SoC) [17] well which makes this project modular.

The storing and restoring functionalities are also implemented inside the classes that create device instances as well as RF line instances. The main task of these classes is to communicate with the PyJapc API in order to read the system's device properties to store them, and/or set the system's device properties to restore them. The API is used so there is as space to replace it with alternatives if needed (i.e. modeled PyJapc). This communication is accomplished by using a device class attribute called `pyjapc_handle`, which represents an instance of the `PyJapc` class. Since this communication is realized inside the device and RF line classes, future users and developers can automatically profit from it.

The list of properties is specified for each of the device manually by a developer. The devices that have been implemented for this work can be seen in Figure 6. In this way, the writing that the developer has to do is reduced to a minimum; only a list of property instances inside the device have to be specified.

The property object consists of a descriptive name (`nice_name`) that is used for generating the get and set methods, and the real FESA name (`real_name`) that has the following pattern: `PROPERTY#FIELD` or just `PROPERTY`. The object contains also the access right (`access_right`) that can be read-write, read-only, or write-only, the protector (protector) that prevents setting unsupported values to the system, and the filters (filter) that are implemented in order to get certain values from the system. The get and set methods are also

generated (populated) here, by connecting the string name of a device property with the get/set functionality; this getter/setter populator is implemented as a class method.

The real name of a property can be looked up in the CCDB by searching for appropriate device name that contains that property. Any PyJapc query should be made according to the following pattern: `DEVICE_NAME/PROPERTY#FIELD` (e.g. `ALLSetPoint5B0t/Phases#vcav`).

4.2.2 Environment Variable

The environment variable container organizes variables into a hierarchy where the first elements represent the root element and every following element represents a child of the previous one. Its structure is similar to the one of nested dictionaries. On every access level, a new dictionary can be created and increase its tree depth. For convenience, the elements of the container can be accessed using dots ('`.`').

Since the container is a mixture of a dictionary and a tree-type architecture, its data is stored in JSON files that were found to be the most suitable for this purpose.

The environment variable creator has a function that converts MATLAB workspaces containing data from previous years into easily readable JSON files. This helps benchmarking the new Python algorithms against the previously used MATLAB algorithms.

4.2.3 Calibration of the RF Feedback and the RF Modulator

The schematic software implementation of the RF Feedback and Modulator calibration is shown in Figure 19. It consists of three main subroutines out of which two (“Calibration of the RF Modulator” and “Calibration of the Set Point”) are wrapped around their Python’s context managers. These two context managers and the “Calibration of the RF Feedback” subroutine are wrapped then around the main context manager (“Nulling” context manager), creating the overall context for the entire nulling routine.

In Python, the context manager [18] allows controlled resource allocation and release. In this work, context manager methods are written in the following way:

- first, the resources on which the modifications will be performed are acknowledged: `__init__` method defines the environment variables and the RF line instances as some operable resources;
- second, the system environment is prepared for the main functionality of the routine or its subroutines: `__enter__` method sets the RF system’s device properties to their initial values;

- and finally, the system is returned to the state it was before entering this context manager: `__exit__` method for setting the device properties back to their initial values they had before entering the routine (or, in certain cases, clearing the values completely).

We chose to use context managers for this tool because of its simplicity when working with exceptions and in order to make sure that the resources are successfully modified only in the scope of `with` statement. If an exception occurs, the context manager will take care of the resources (device properties, environment variables, backup, ...) affected by the `__exit__` method.

The `__enter__` method of the Nulling routine context manager does not only take care of modifying device properties, but also loads the excitation into the buffers of the RF Modulator. Similarly, the switching off of the excitation is implemented in the context manager's exiting procedure. The details of this procedure can be also seen in Figure 19.

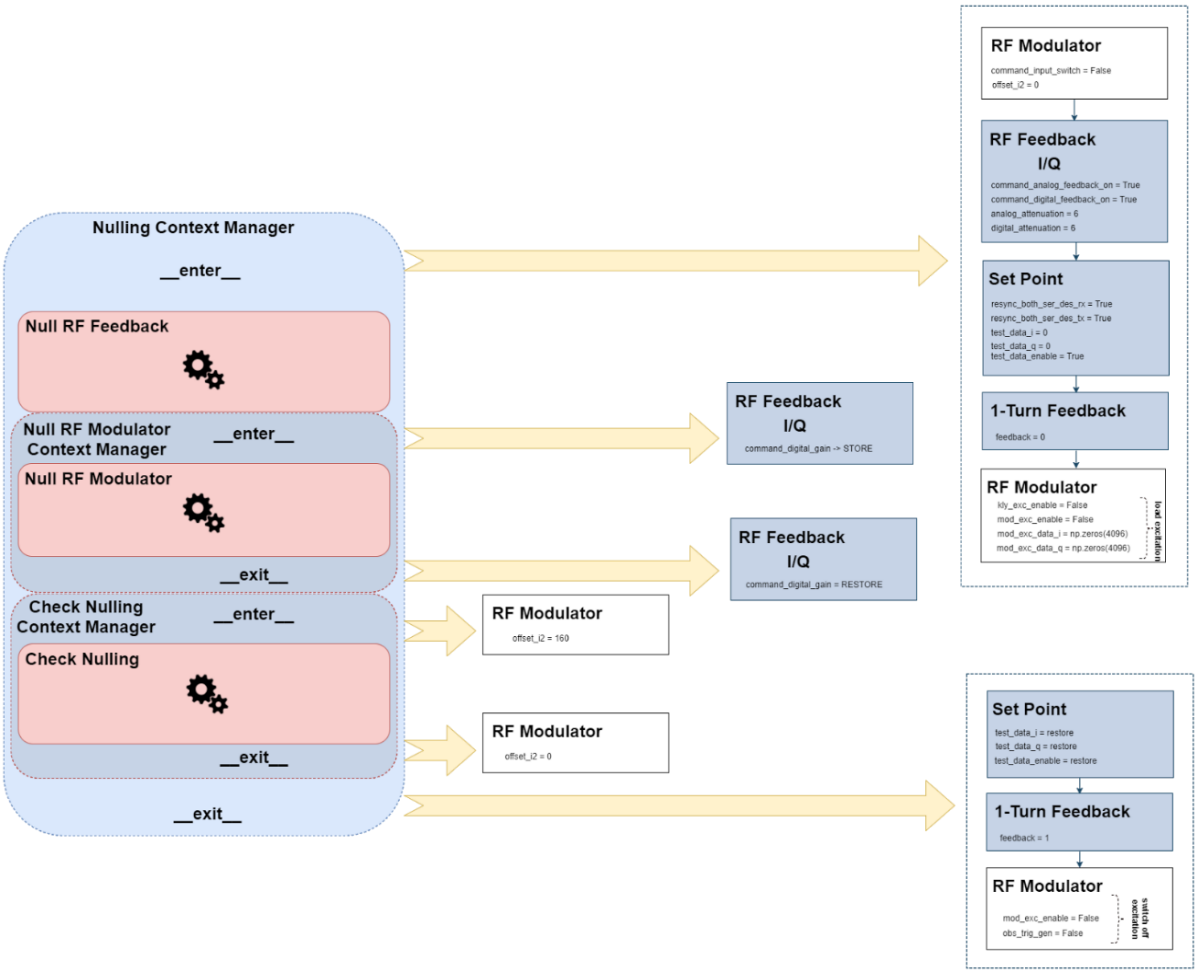


Figure 19: Schematic code organization of the Nulling routine along with the implemented context managers.

Calibration of the RF Feedback

The calibration procedure is shown in Fig 3.4. and steps are as follows:

1) the subroutine for nulling the RF Feedback module starts by varying the I and Q channel offset values in the RF Feedback module (step 1);
2) for every offset value, the data in the RF Modulator excitation buffer is cleared (step 2) and
3) the RF Feedback observation buffer is read out for channels I and Q (step 3); these observation buffers contain the Set Point module's output data for I and Q channels.

The mean value of the data from observational buffer should be close to zero since there is no excitation applied in the RF Modulator. The calibrated RF Feedback offset values are then the ones that result in the Set Point module's average output data being closest to zero.

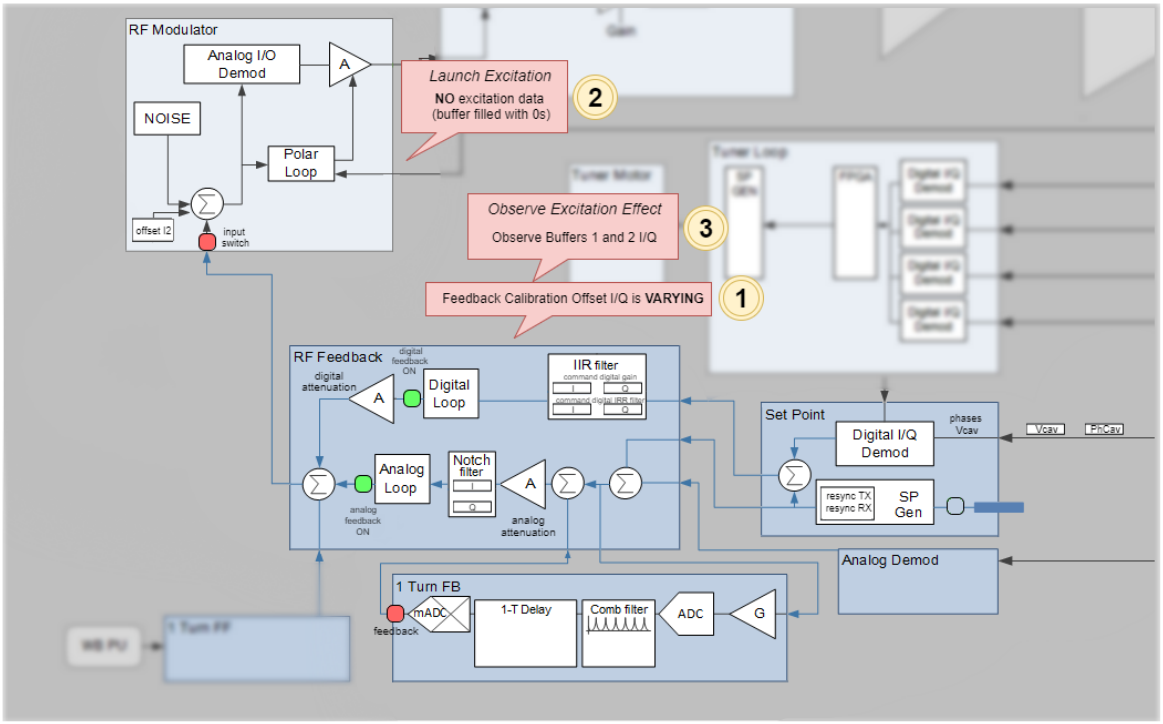


Figure 20: Procedure of the RF Feedback calibration.

Data acquisitions for the calibration of the RF Feedback are done in three main stages (as described above), see Figure 20. These stages for collecting data are repeated twice (step 1 and step 2, see Fig.4.5.). The first step has input values (offsets of channel I and Q) set to -4000 (`offset_I_1=-4000` and `offset_Q_1=-4000`), whereas the second step has input values set to 4000 (`offset_I_2=4000` and `offset_Q_2=4000`). The results of steps executions are the mean values of data collected from the RF Feedback module which depend on the previous offset setting (`mean_I_1`, `mean_Q_1` from the first step; and `mean_I_2`, `mean_Q_2` from the second step).

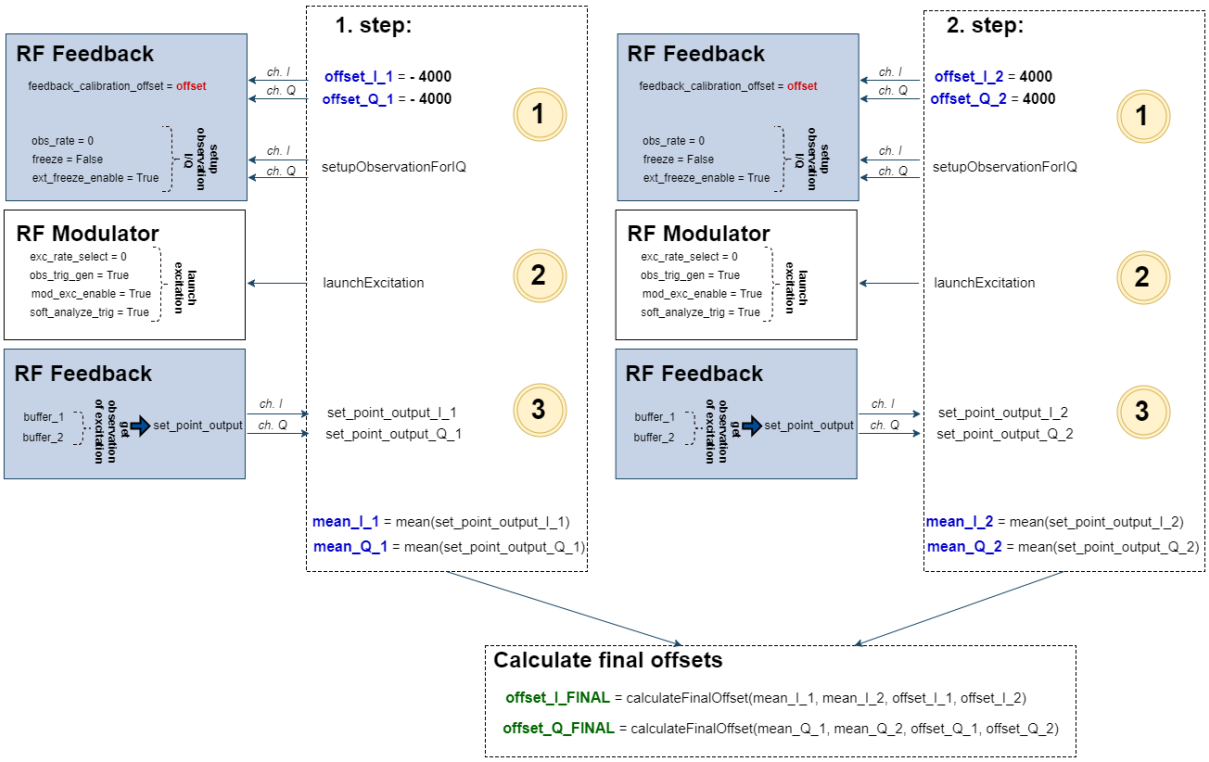


Figure 21: Schematic of the RF Feedback calibration routine.

The actual calculation is performed once the steps for data collection are completed. The final values of offsets on the channel I and channel Q are calculated using the weighted arithmetic average [19]. Since the mean needs to be equal to zero for the calibrated offsets, $m(o_{new}) = 0$, the final equation for calculating final offset value, o_{new}, separately for the channels I or Q is:

$$o_{new} = \frac{m_1 o_2 - m_2 o_1}{m_1 - m_2},$$

where is mean data acquired in the channel (I or Q) for offset values respectively (`mean_I_1` and `mean_I_2` for channel I or `mean_Q_1` and `mean_Q_2` for channel Q, see Figure 21); are offset values -4000 and 4000 (`offset_I_1` and `offset_I_2` for channel I or `offset_Q_1` and `offset_Q_2` for channel Q, see Figure 21).

Calibration of the RF Modulator

The procedure is visualized in Fig. 3.6. Steps are following:

1) During the measurement with a developer-specified digital gain (step 1),
2) first the offset in channel I is scanned (step 2) and
3) the output of the Set Point module is observed in the observation buffer of the RF Feedback (step 3). Then,

4) the procedure is repeated for channel Q (steps 4 and 5). The roughly optimal offset is determined from the minimum signal level at the output. The fine scan of the offsets is done around the rough optimum.
5) When the measurement is completed, the final offsets determined in that measurement are set to RF Modulator (step 6).

These steps are the same for the coarse and fine measurement. The only differences are the value of the digital gain and the range of values in which the offsets are varying.

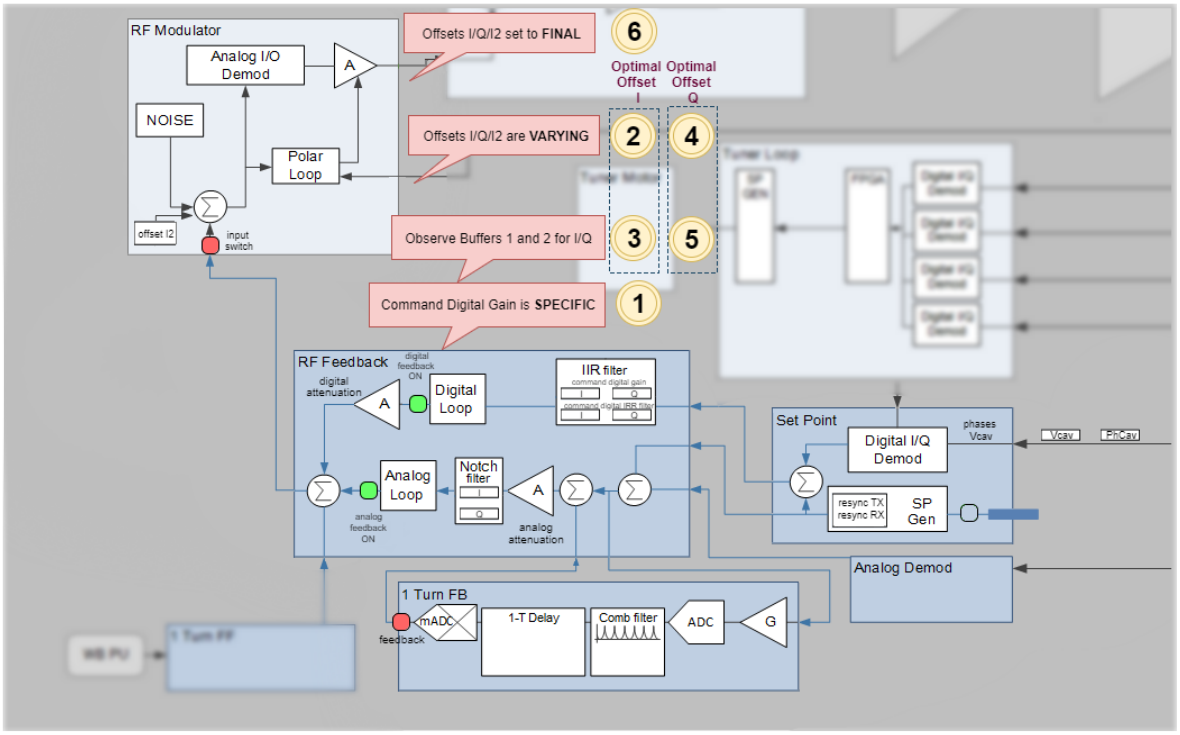


Figure 22: Procedure of the RF Modulator calibration.

The calibration of the RF Modulator is achieved in two steps (see Figure 22), a coarse and a fine measurement as detailed previously.

During the coarse measurement (see part for the *coarse measurement* in Figure 23), the gain of the Infinite Impulse Response (IIR) filter in the RF Feedback (IIR gain, or digital gain) is reduced to 4 dB (the operational value is typically around 9 dB) in order to avoid saturating IIR filter when nulling (step 1 for coarse measurement). Also, the offset range scanned is broader (from around -1600 to 1600 with a step of 160, i.e. 21 iteration) than in the fine measurement. First, the measurement is performed on I channel. The RF Modulator offset of the given channel is set to the iteration value (step 2 for the coarse measurement). Same as already described in the above, data collected from RF Feedback buffers is then used in calculating the amplitude (step 3 for the coarse measurement). When the offset scan for channel

I is finished, the offset that gives the minimal value of the amplitude (`i_min`) is forwarded to the further calculation. Next, the offset range is the same, however, the measurement is performed on the channel Q. Therefore, the steps 4 and 5 for scanning offsets of channel Q, see Figure 23, are similar to steps 2 and 3 for scanning offsets of channel I. Also, when the offset scan for channel Q is finished, the offset that gives the minimal value of the amplitude (`q_min`) is determined and propagated to the final step of the measurement (step 6, see Fig 4.6). Both values, `i_min` and `q_min`, are scaled and uploaded to the RF Modulator as calibrated values of the coarse measurement.

During the fine measurement (see part for the *fine measurement* in Figure 23), the digital gain (IIR gain) in RF Feedback is set to 6 dB and the calibration is performed in a narrower range with a finer resolution (from `C_min-320` to `C_min+320` with a step of 16, i.e. 41 iterations, for the channel `C={i,q}`). The steps itself are the same as in the coarse measurement. Values `i_min` and `q_min` that represent the calibrated I and Q offset values are then uploaded to the system as final values.

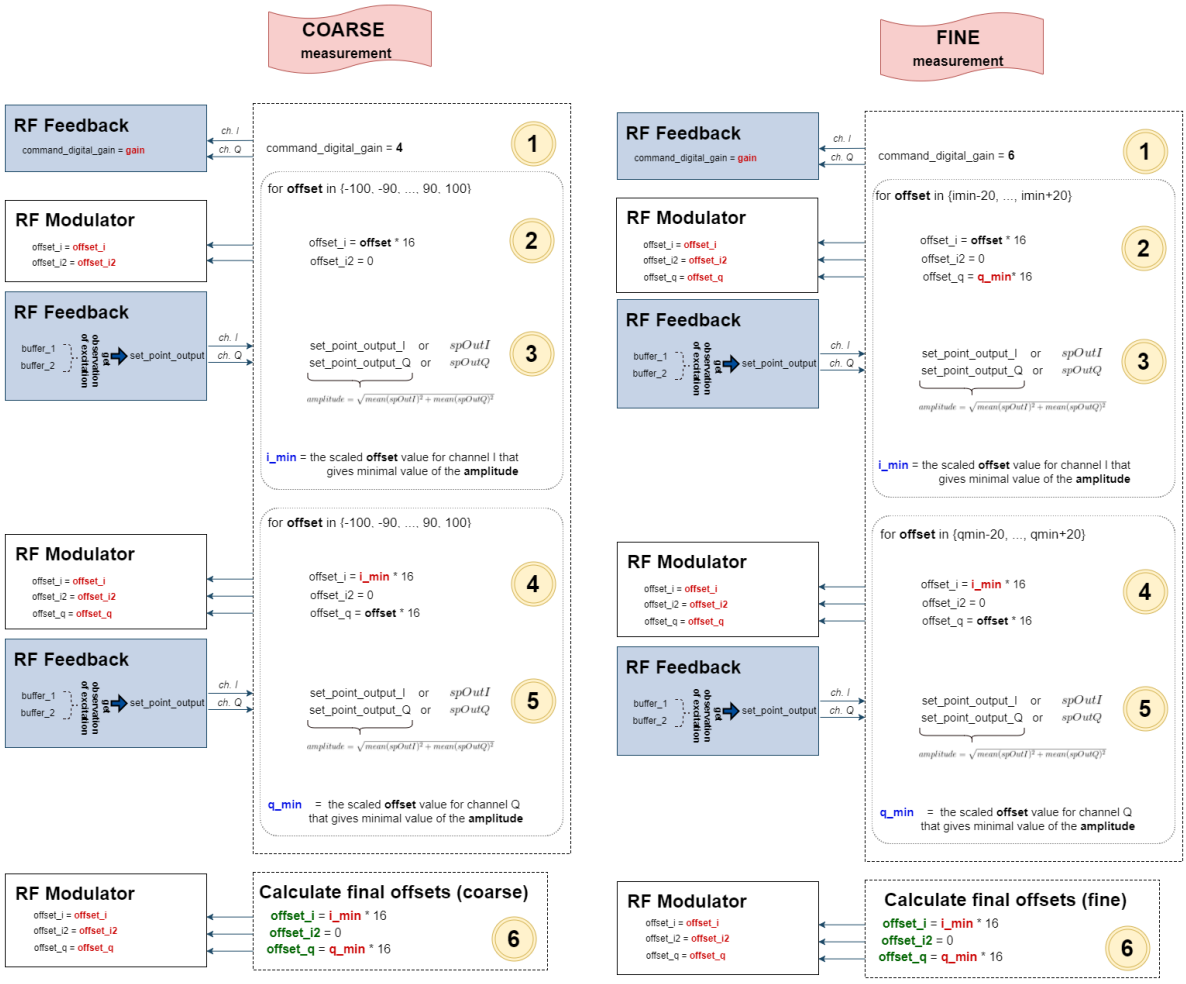


Figure 23: Schematic of the RF Modulator calibration routine.

Calibration of the Set Point

In the third and last subroutine, the calibration is verified for a non-zero power level. To do so, the steps are:

1) a non-zero input level is set in the RF Modulator through the "offset I2" (step 1),
2) which should be counter-acted by setting a corresponding set-point in the Set Point "test data" properties (step 2); for a more accurate calibration of the offsets, measurements are done around zero values of test data property and then they are interpolated in order to get test data values for which the observation buffer data mean will tend to become a zero value;
3) measuring then the observation buffers of the RF Feedback that acts on the difference of the two signals, the mean value of the data should be as close as possible to zero (step 3).

The procedure is illustrated in Fig. 3.7.

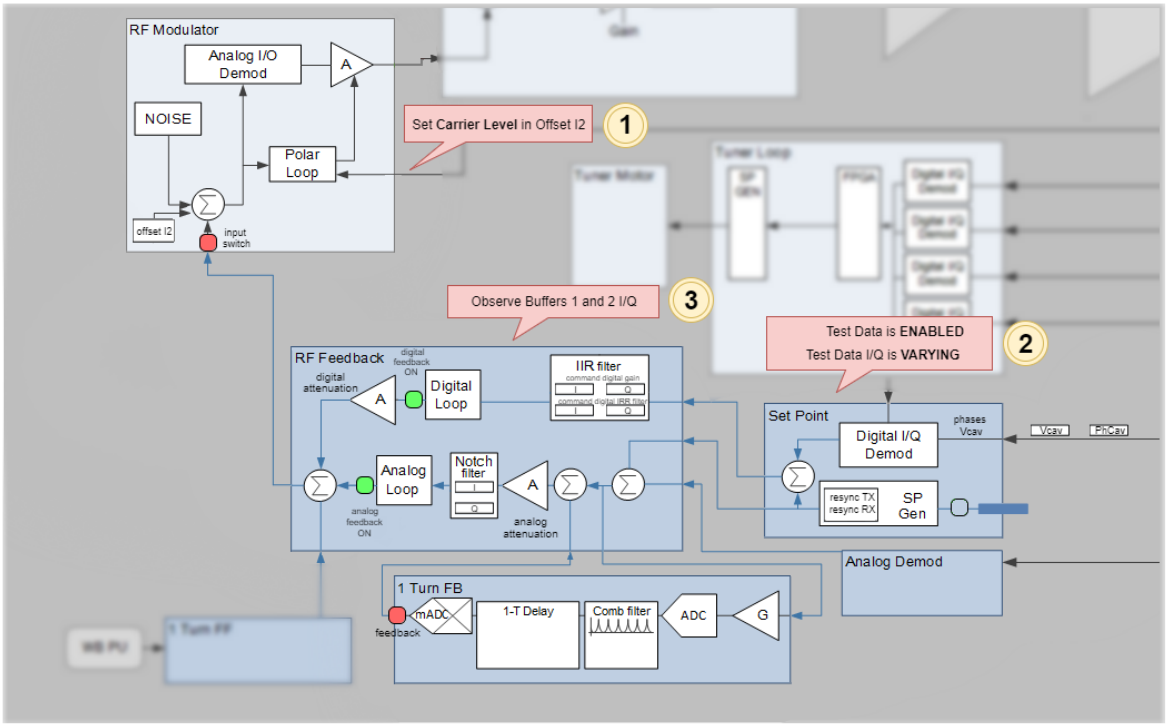


Figure 24: Procedure of the calibration at non-zero power level.

More detailed visualization of pseudo code for Set Point calibration can be seen in Figure 25. The shown algorithm has three main iterations. In the beginning, every iteration sets test data values for I and Q channels to predefined value (step 1). Afterwards, data from RF Feedback is acquired for that measurement/iteration (step 2). When data acquisition is finished, acquired data from observation buffers is forwarded to the module (method) for solving a set of linear equations. The output of this calculation are the calibrated values of I and Q test data

that will generate RF Feedback data which mean is equal to zero. These values will be uploaded to the system as a final result (step 3).

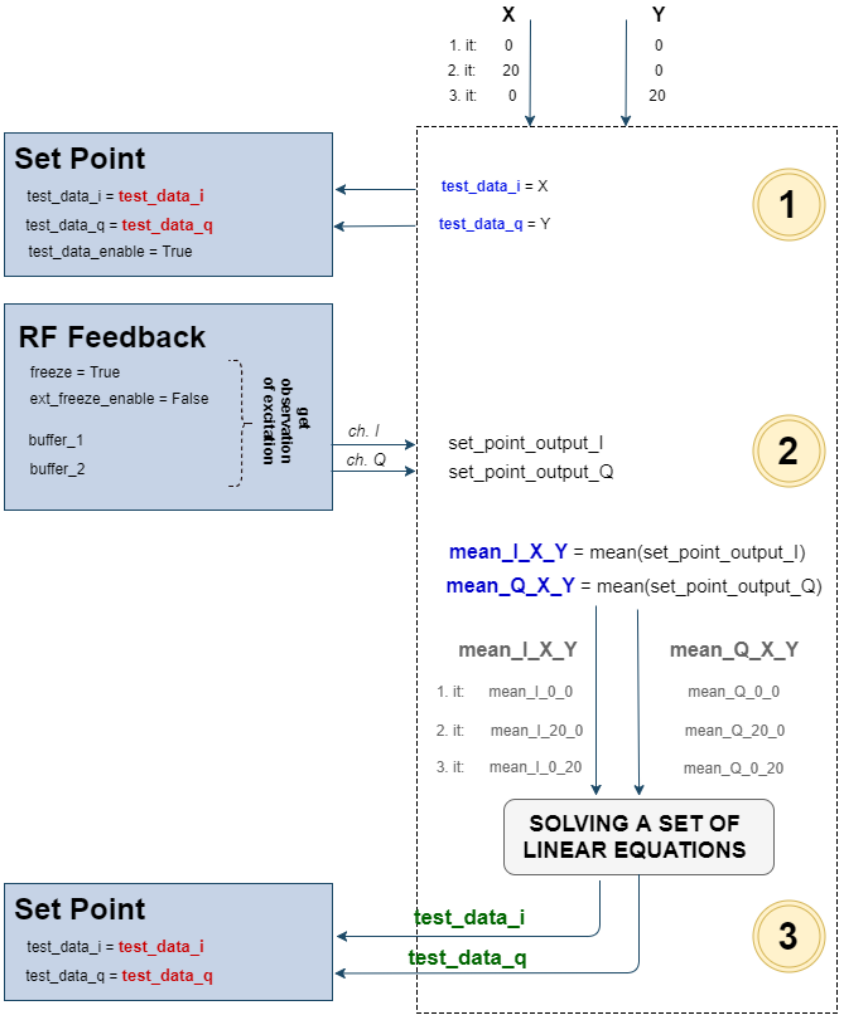


Figure 25: Check Nulling code procedure visualized.

4.2.4 Aligning the RF Feedback Phase

The routine for the RF Feedback phase alignment is summarized schematically in Figure 26. Just like the Nulling routine, also the Phase Alignment routine uses context managers to protect the system, at least partially, from an unexpected or unwanted behavior.

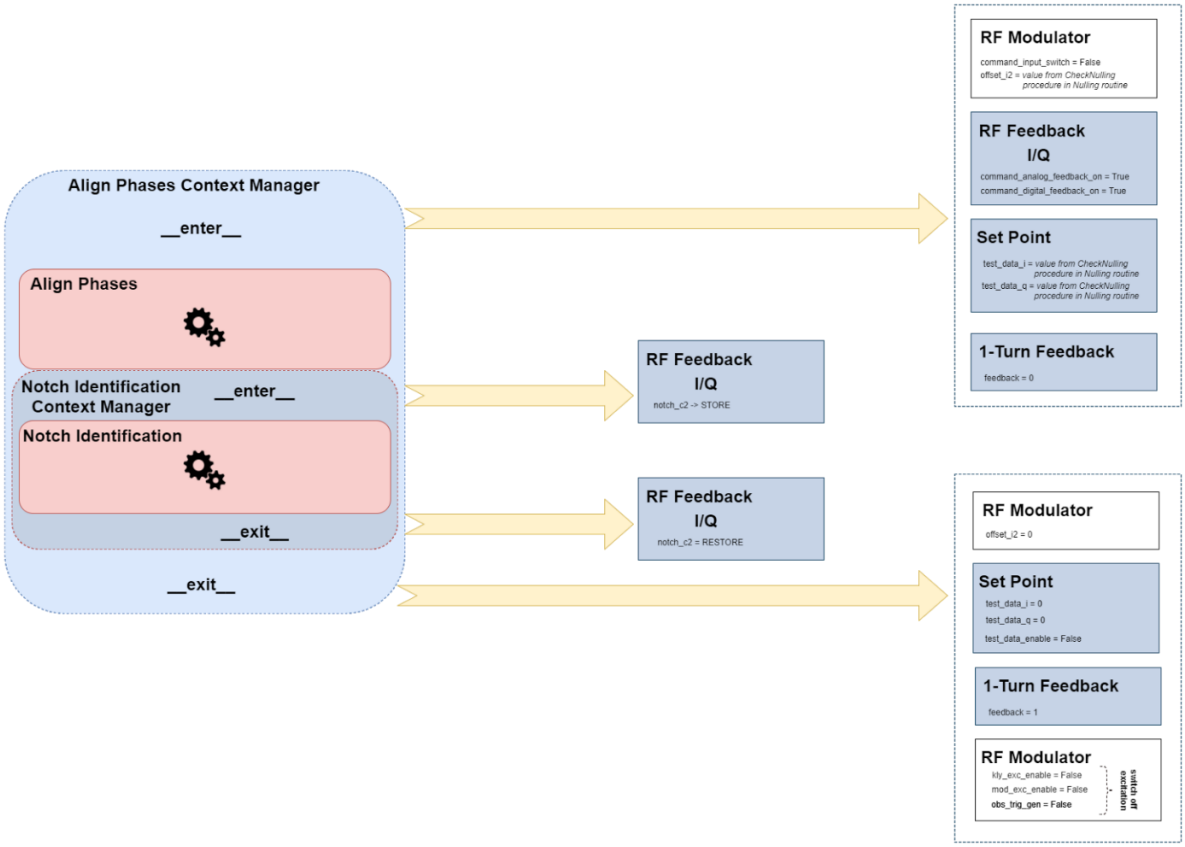


Figure 26: Schematic code organization of the Align Phases routine along with the implemented context managers.

The Align Phases Subroutine

The procedure of the phase alignment routine is shown in Figure 27. After the preparation, the phase alignment routine:

1) starts with setting test data I and Q as previously mentioned (step 1); afterwards,
2) white noise is loaded into the excitation buffer of the RF Modulator (step 2);
3) the following steps are to prepare RF Feedback module for the observation (step 3) and
4) launch the excitation that was previously loaded in the RF Modulator (step 4);
5) the Set Point output is read out from the observation buffers of the RF Feedback (step 5); using this measured data, the system response is obtained via a numerical network analyzer model and the response is fitted with the theoretical transfer function of the system; details of these calculations will be covered in Sec. 4.2.4;
6) finally, the V_{cav} phase is set to one calculated during the transfer function fit procedure (step 6). Steps 1-6 are repeated for a different test data property values.

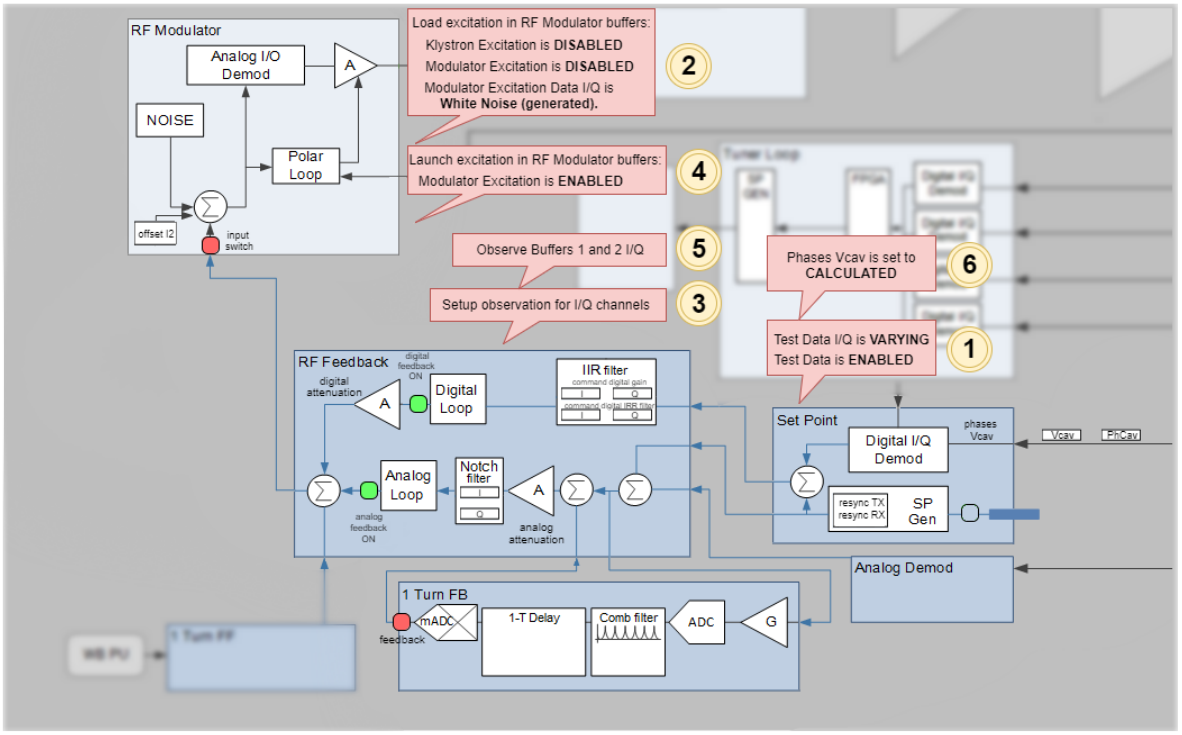


Figure 27: Procedure of the RF Feedback phase alignment.

The detailed implementation of the Phase Alignment routine is shown in Figure 28. The optimal phase between the digital and analog feedback is found in two stages/iterations.

First, test data I and Q values from Set Point module are modified. The test data components I and Q are rotated by the phase of the digital feedback (digital phase),

$$\begin{bmatrix} test\_data\_i_{new} \\ test\_data\_q_{new} \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} test\_data\_i \\ test\_data\_q \end{bmatrix}, \quad (1)$$

such that the new I and Q signal is now described w.r.t. the digital feedback. In the first iteration (1. step), no rotation is applied ($\varphi = 0$) and initial values of I and Q test data are the ones calculated during the calibration of Set Point module. New rotated components are uploaded to the Set Point module. The following steps in this iteration correspond to the steps previously explained in Figure 27.

The response of the system is analyzed by performing a network analysis (see Network Analysis routine, Sec. 4.2.4) on the data and fitting it with the analytical transfer function of the system (see Transfer Function Estimate routine, Sec. 4.2.4). By fitting to the measured data, we can accurately calculate required adjustments to the cavity phase angle (V_{cav} phase) adjustment in the Set Point module. In the end of the iteration, new V_{cav} phase is uploaded to the system and modified environment is updated.

In the second iteration, new angle of rotations and new values of I and Q test data are used. This angle was adjusted during the first iteration in Transfer Function Estimate routine.

The values of I and Q test data are rotated components from the first iteration. The new test data values are calculated using the same equation above. The rest of the steps are the same as explained in the first iteration. Once the test data I and Q components are aligned with the digital feedback, the phase of V_{cav} can be calibrated, as it will now represent the relative phase between the digital and analog feedback. After the second iteration, calibrated value of V_{cav} phase is uploaded to the system.

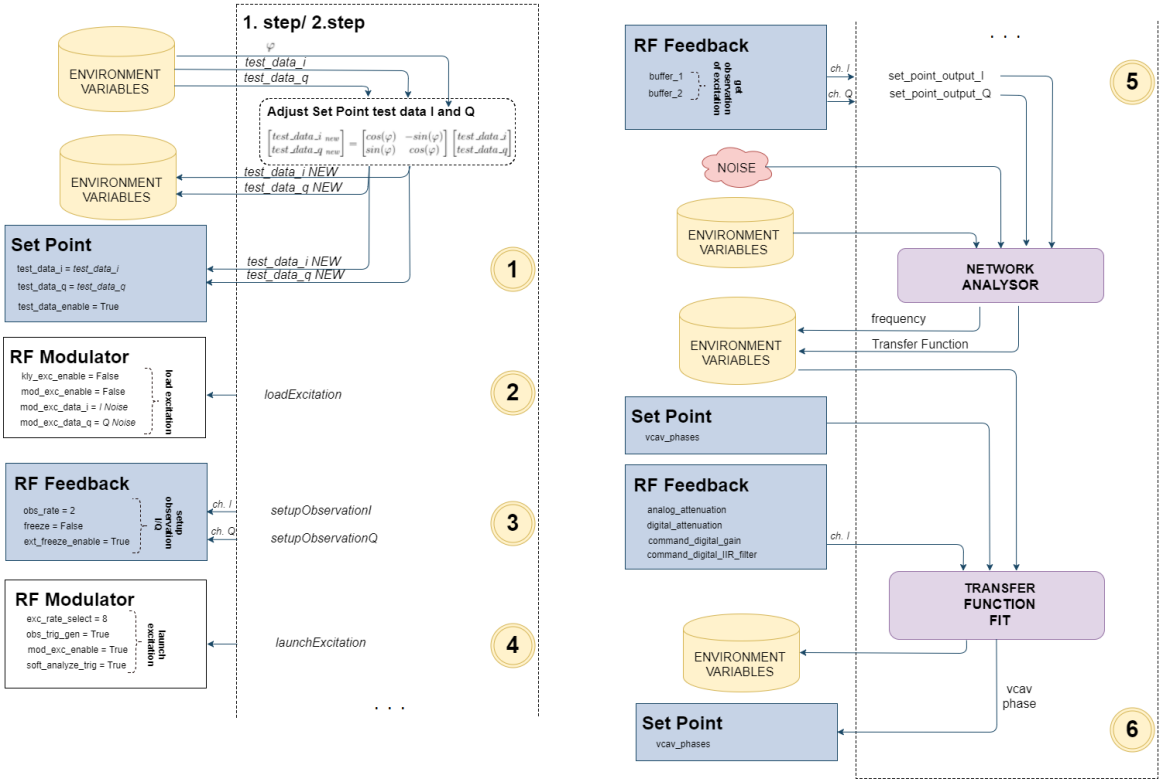


Figure 28: Schematic code organization of the Phase Alignment.

The Notch Identification Subroutine

The notch identification is performed as follows (see also Figure 29):

1) first, the capacitor value inside Notch filter is set to a given value that is scanned from 0 to 31 (step 1);
2) the following steps are similar to those implemented in phase alignment discussed previously: the RF Feedback is prepared for the observation (step 2) and
3) the excitation that was loaded previously in the RF Modulator is launched (step 3);
4) the Set Point output is observed in the buffers of the RF Feedback (step 4); as in the phase alignment, also here the transfer function is obtained from the system via network analysis and a fit with the theoretical transfer function is performed (cf. Chapters ..xx..).

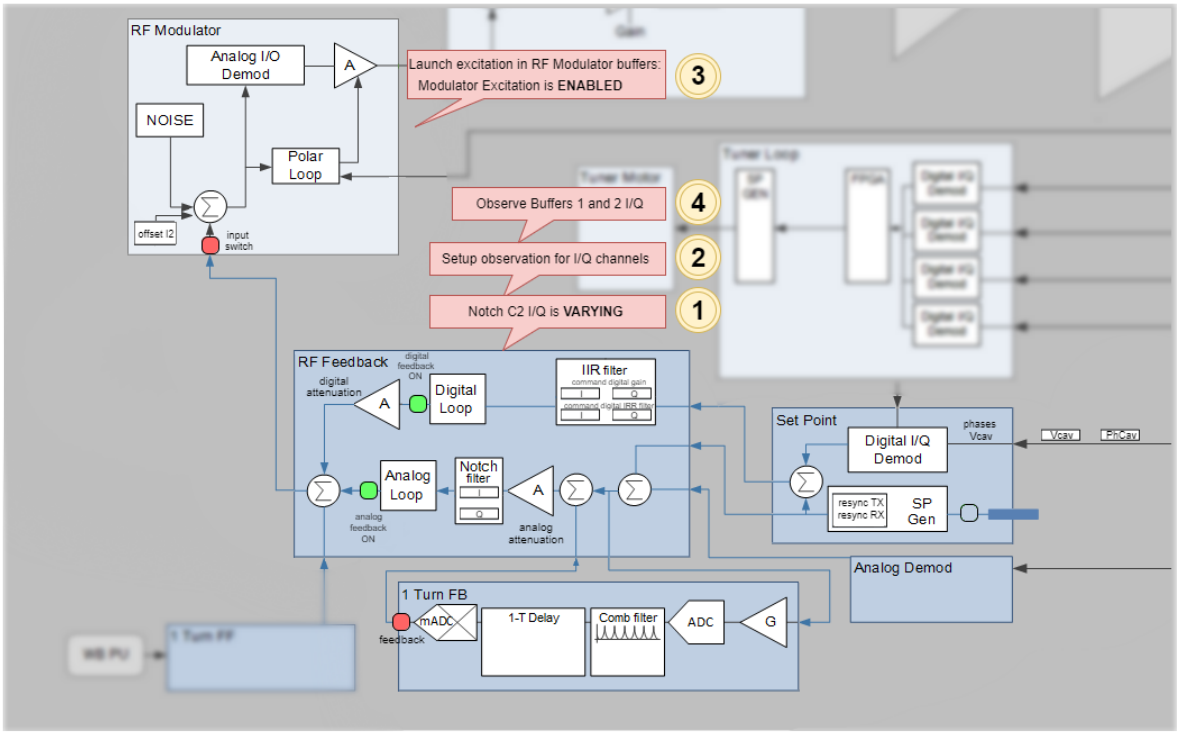


Figure 29: Procedure of the RF Feedback notch filter adjustment.

The Notch Identification, see code structure in Figure 30, is calibrating the frequency of the notch filter that is supposed to cancel the klystron resonance in the system.

The capacitor values in this Notch filter have values in the range from 0 to 31. Since a linear behavior of the notch frequency as a function of notch steps is expected, measurements are performed only at four distinct values (0,10, 21, 31). Afterwards, the RF Feedback is prepared for observation and the white-noise excitation is launched. The response of the system is acquired from the buffers of the RF Feedback module. Just like in the previous routine, the response of the system is analyzed by performing a network analysis on the data (more in Sec. 4.2.4) and fitting it with the analytical transfer function of the system (more in Sec. 4.2.4). In every iteration, data is collected and stored in an environment variable for subsequent Notch filtering done in other routines.

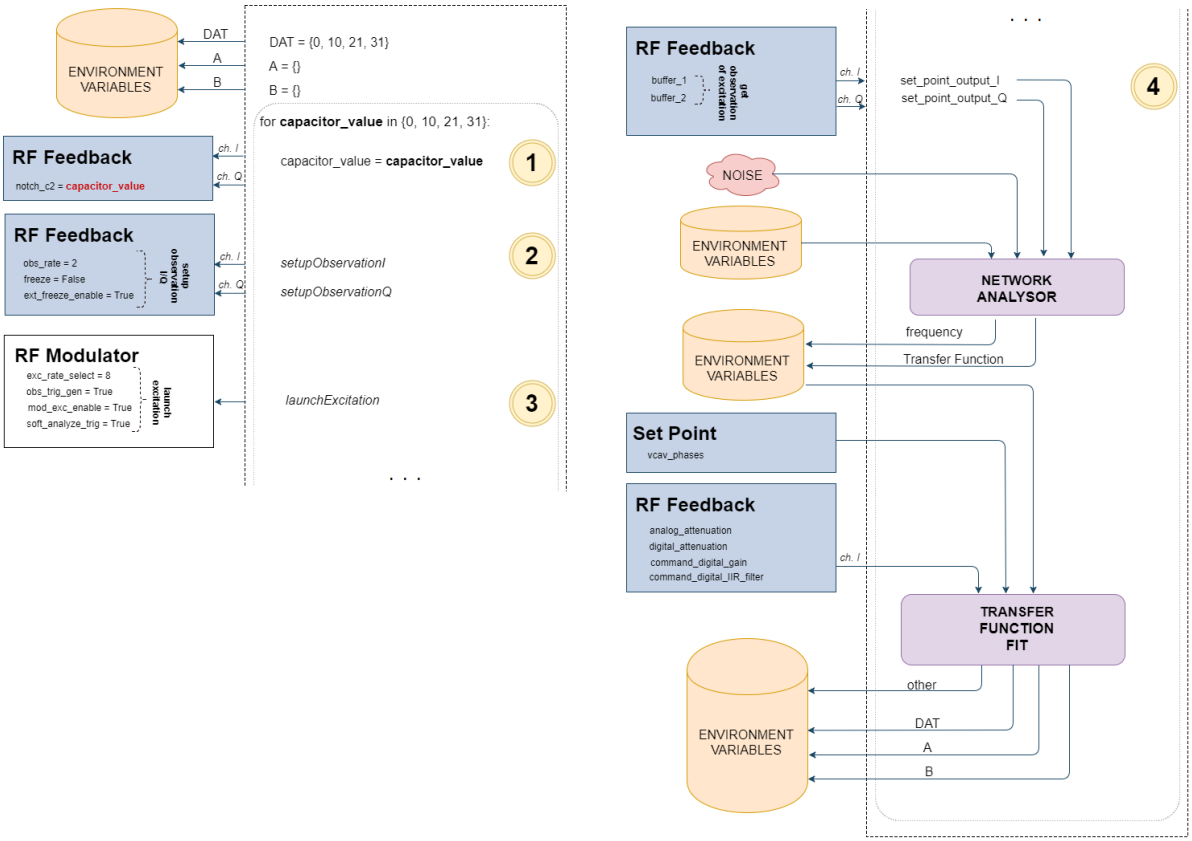


Figure 30: Schematic code organization of the Notch Identification routine.

HELPER FUNCTION: Network Analyzer

Linear, time-invariant (LTI) systems are often described with the aid of the so-called impulse response $h(t)$ in time domain, or it's Laplace transform, the transfer function $H(s)$ in frequency domain, see Figure 31. In such a system, a function $x(t)$ is mapped into $y(t)$ by its convolution with the impulse response:

$$y(t) = \int_0^t \big(h(\tau)\, x(t-\tau)\big)dt\,,$$

or in the frequency domain as:

$$Y(s) = H(s)X(s)\,,$$

where $X(s)$ and $Y(s)$ are the Laplace transforms of $x(t)$ and $y(t)$, respectively [20][21].

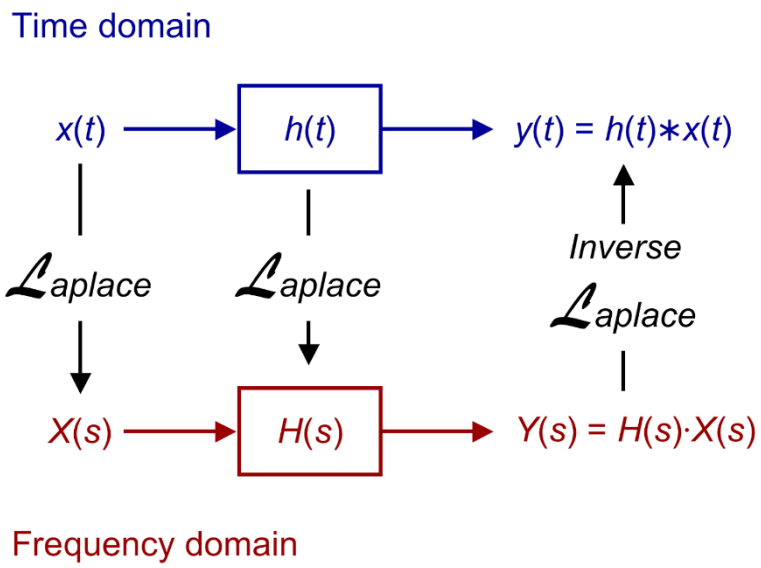


Figure 31: Response of a linear, time-invariant system described via the impulse response and the transfer function [22].

Also, the system response of the LHC RF cavity controller is conveniently described with the help of transfer functions that are known analytically for each of the filters in the system. Depending on the specific parameters used in different parts of the system, we would like to measure the actual, overall transfer function and compare it to what we expect from theory. The Network Analyzer routine extracts this measured transfer function using the input and output signals of the system.

In the case of the RF Feedback Phase Alignment, the transfer function is determined from the (i) generated white noise which is previously injected into the RF Modulator buffers and represent the input signal of the system, and the (ii) data observed in the RF Feedback representing the output signal from the system. From these two signals, the transfer function is determined, see Figure 32.

The Network Analyzer requires to have the same amount of points in the input and output of the system. Since the input signal is much shorter (2^{12} or 4096 points) than the output signal (2^{18} or 262144 points), the input signal is adjusted to have a higher sampling rate and is subsequently interpolated[4].

[4] The interpolation in DSP is the process of up-sampling followed by filtering.

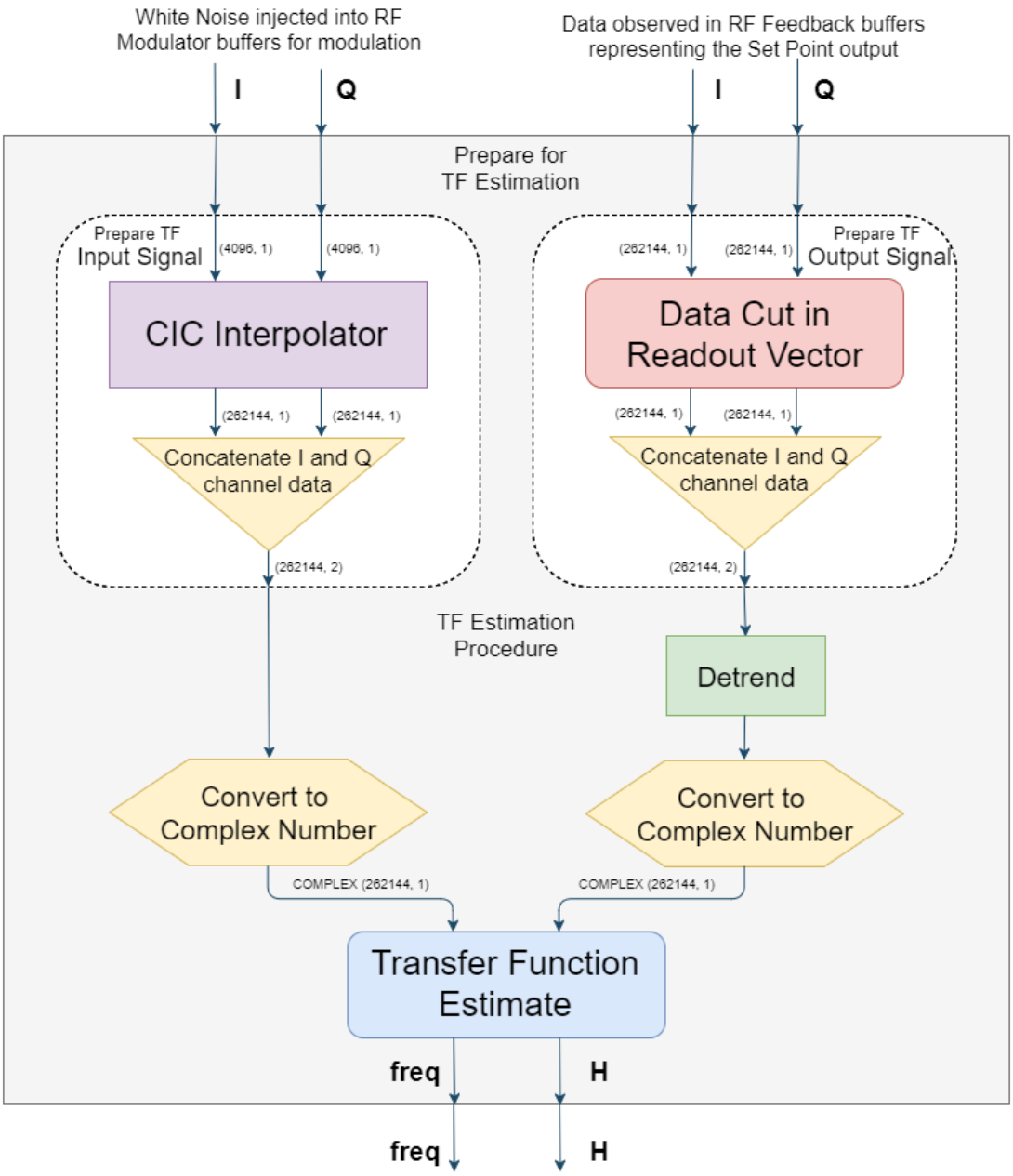


Figure 32: The Network Analyzer algorithm determining the transfer function of the system from its input and output.

The interpolation is done using the so-called Cascaded Integrator Comb (CIC) filter. The CIC interpolation block performs a sampling rate increase (interpolation) on an input signal by an integer factor. As the name suggests, CIC filters consist of a comb part and an integrator part.

The transfer function of CIC interpolator filter is:

$$H(z) = H_I^N(z)\, H_C^N(z) = \frac{1}{(1-z^{-1})^N}(1-z^{-RM})^N \ ,$$

where H_I is transfer function of integrator part of filter, H_C is transfer function of comb part of filter; and N is the number of sections in the CIC filter (defined as the number of sections in

either the comb part or integrator part), R is the interpolation factor, and M is the differential delay. The schematic of the CIC filter is shown in Figure 33.

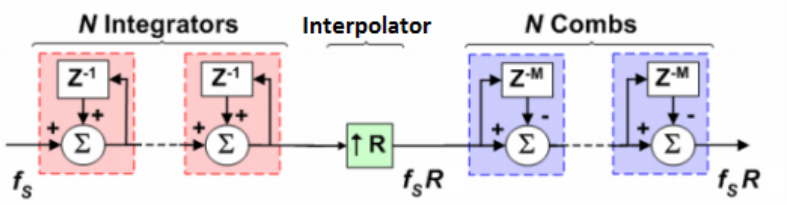


Figure 33: Schematic of the CIC filter.

For the routines described in this Thesis, we use a CIC filter of $N = 1$. To achieve the same length of the input and output, the interpolation rate has to be $R = 2^6$. This transfer function represents the transfer function of integrator and comb part, separately. The output signal of such a single-input single-output (SISO) LTI system, whose input signal and transfer function are known, can be calculated with the "control" Python package developed for control systems. This output signal of the CIC filter is fed to transfer function estimation module.

The other thing that has to be fed into the transfer function estimation module inside the Network Analyzer is adjusted output signal of the system (see Figure 32). This output signal is adjusted by filling in the first few signal samples with zeros, and then reproducing the signal that is collected from the RF Feedback module buffers. Once the output and input signals are obtained, they can be fed into the Network Analyzer algorithm to estimate the transfer function, as a function of frequency.

HELPER FUNCTION: Transfer Function Fit

Once the Network Analyzer gives the measured transfer function, it can be compared to the overall transfer function of the system expected from theory. The Transfer Function Fit algorithm fits the analytical transfer function to the measured data. Ultimately, this is done to extract how much the model deviates from reality and guides the user with an estimate of how much certain parameters have to be changed in order to obtain a given desired system response.

The implementation flow is visualized in the pseudocode in Figure 34.

During the phase alignment of the RF feedback, only the digital and analog feedback, their relative phase and an over phase shift (misalignment of loop) and time delay (cable length) have to be modeled. During the notch identification, also the notch filter is acting and has to be taken into account.

The corresponding transfer functions H as a function of the revolution frequency ω are:

- The overall system delay τ is modeled by the transfer function

$$H = e^{-\omega i \tau} ,$$

and the overall system phase φ w.r.t. the analog feedback is modeled as

$$H = e^{\varphi \frac{\pi}{180} i} .$$

Therefore, the phase response is:

$$H_{ph} = e^{-\omega i \tau} e^{\varphi \frac{\pi}{180} i} .$$

- The overall digital and analog feedback response is:

$$H_{fb} = -G_A \left(\frac{G_D e^{\Delta\varphi \frac{\pi}{180} i}}{1 + \omega \tau_D i} - \frac{\tau_A \omega}{\tau_A \omega - i} \right),$$

where G_A , G_D are analog and digital loop gains, τ_A, τ_D are analog and digital loop delays, $\Delta\varphi$ is a relative phase of the digital feedback w.r.t. to the analog feedback.

- The notch filter transfer function is:

$$H_n = \frac{s^2 + As + {\omega_o}^2}{s^2 + Bs + {\omega_o}^2},$$

where $s = \omega i$, ω_0 is notch frequency, and A, B coefficients uniquely determined.

The tunable system parameters, such as the relative phase of the digital and analog feedback (V_{cav} phase) or the feedback gains, can be adjusted in order to obtain a desired system response. Finding a good fit of the analytic transfer function to the measured one helps to estimate by how much each of these parameters have to be adjusted.

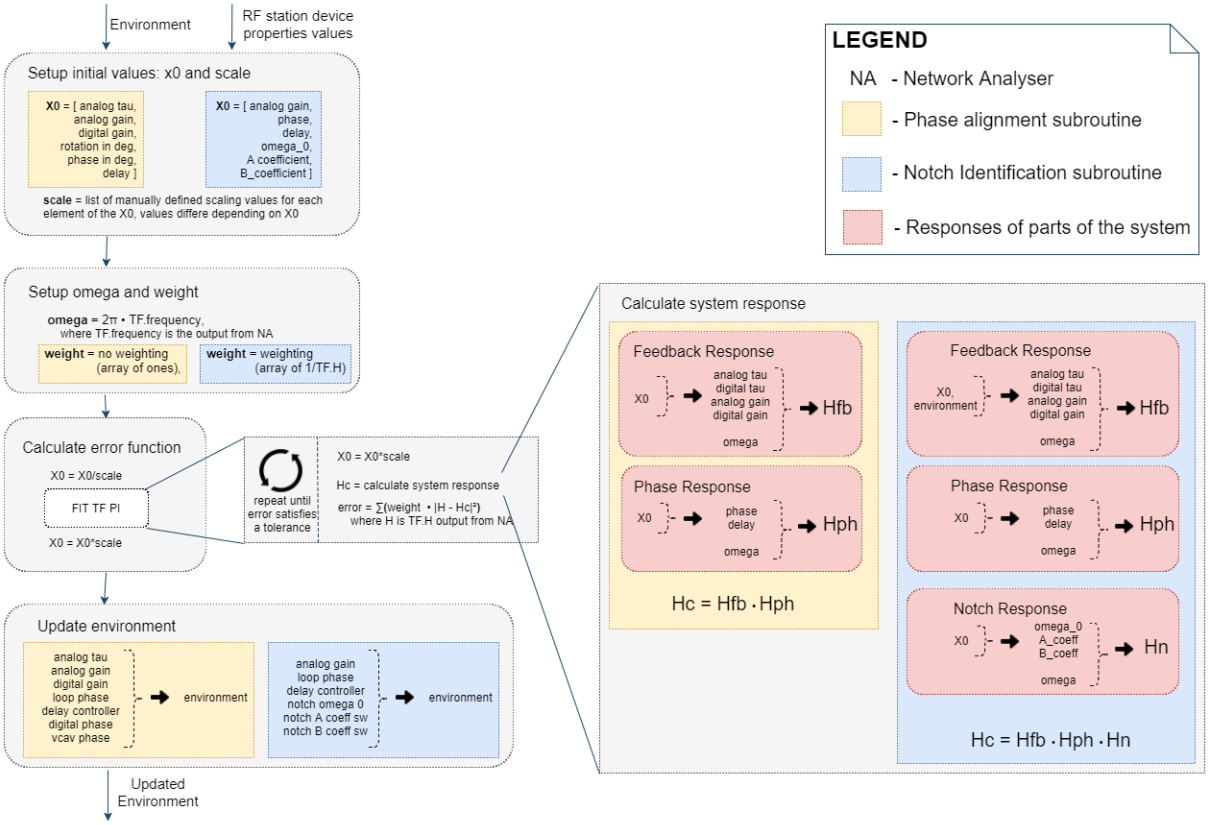


Figure 34: Schematic routine of the Transfer Function Fit algorithm.

4.2.5 The Fine Alignment

The Fine Alignment routine is a completely new development which allows to adjust the phase between the digital and analog feedbacks without re-cabling, directly on the operational system with the cavity connected. Figure 35 shows the schematic outline of the routine.

In this routine, only the Set Point module is involved: all parameter reading and writing happens here.

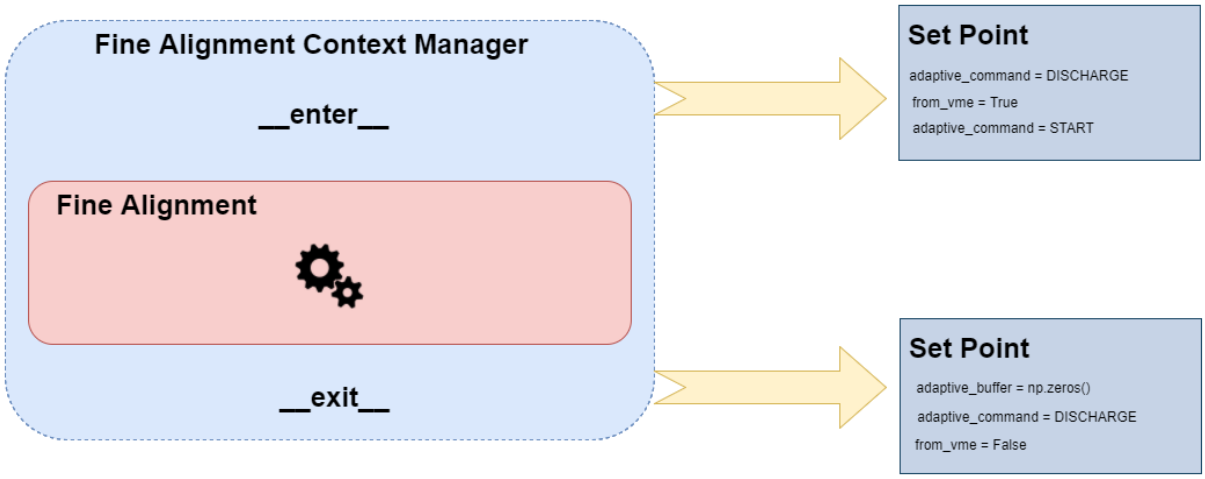


Figure 35: Schematic code organization of the Fine Alignment routine along with the implemented context managers.

The main algorithmic steps of the routine are shown in Figure 36. First, the phase modulation, whose amplitude is trapezoid-shaped for a gentle turn-on and turn-off, is filled in the adaptive buffer. Afterwards, the search for the optimal V_{cav} phase value starts. User can specify the depth of the search (phase step). That means the iterated phase range could be increased with the increase of the specified phase step. By default, the algorithm iterates within the 10° during one phase step. Depending on the output, it continues to iterate in the same direction, it changes the direction of the iteration, or it notifies the user that the optimal V_{cav} phase has been found.

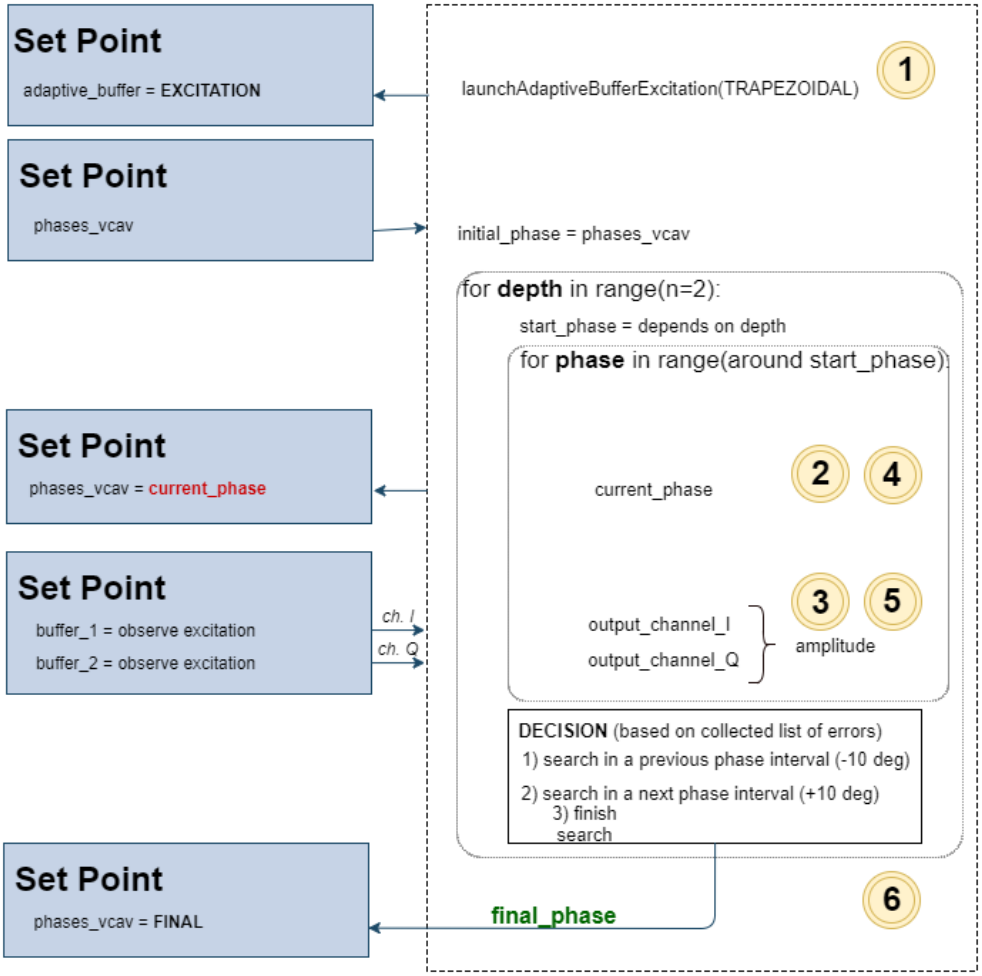


Figure 36: Algorithmic procedure of the Fine Alignment routine.

Since the system becomes unstable for certain V_{cav} phase values, it is not recommended to specify a depth which is greater than 10 phase steps. System stability also depends what is the initial V_{cav} phase value. More about the stability of the system during the Fine Alignment scan of the V_{cav} phase in Sec. 5.1.3.

5. Results

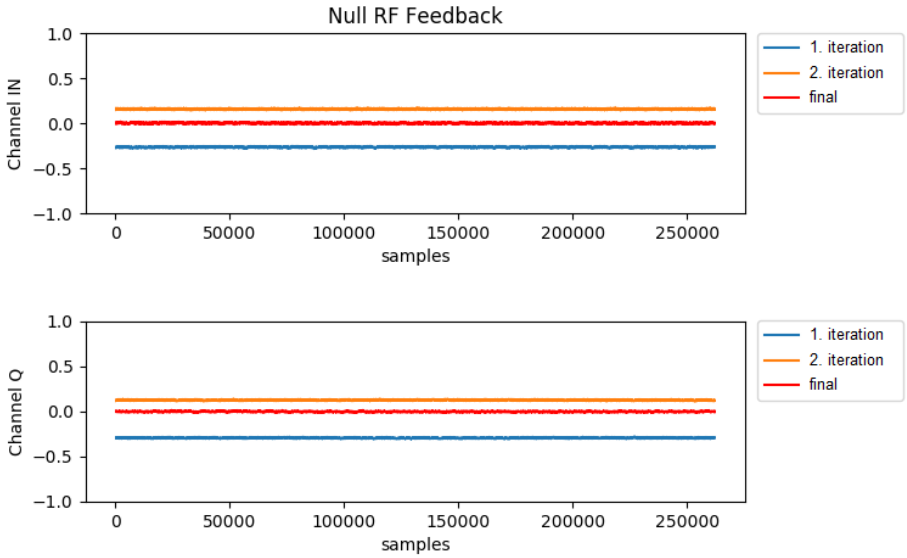
In this Chapter, we present the outcome of each routine and analyze in greater detail the results of each of them. As for the helper routines Network Analyzer and Transfer Function Fit, their implementation as well as the implementation of the backup are tested.

5.1 Routines Outcome

This Section presents and explains the output obtained from the routines. All plots are generated during the execution of the routines.

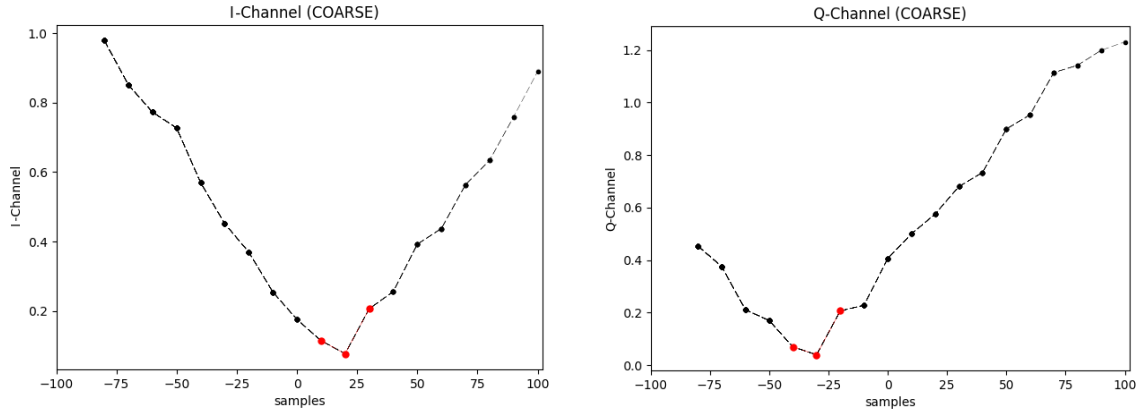
5.1.1 Calibration of the RF Feedback and the RF Modulator

The algorithm used in this routine was described in detail in Sec. 4.2.3. The measurements are done for two discrete offset values, namely -4000 and 4000. Using the data obtained in these two measurements, the calibrated offset value is calculated as explained in Sec. 4.2.3. If the mean data obtained with the calibrated offset is indeed closer to zero than the data measured with the previous two offset values, the calibration is finished. A typical output during a successful calibration of the RF Feedback is shown in Graph 1.



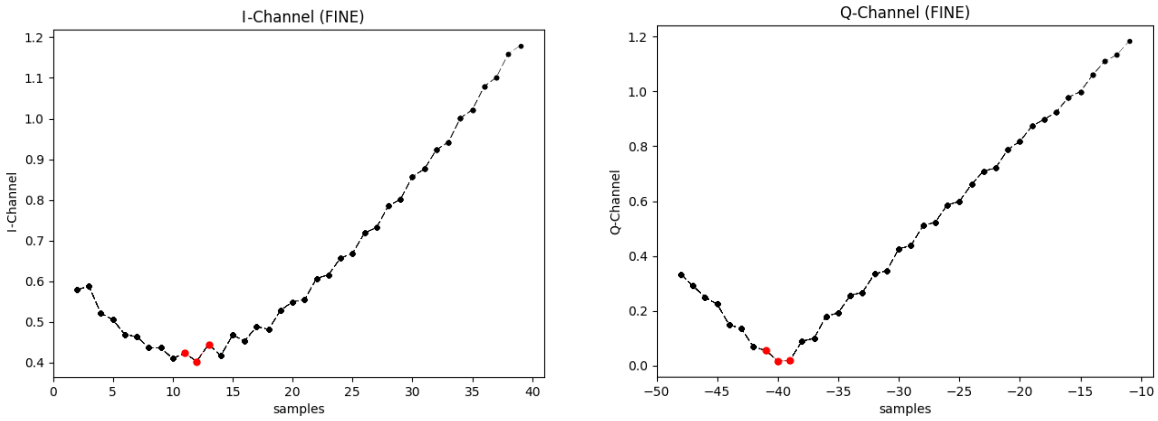
Graph 1: Typical output of the RF Feedback calibration subroutine.

The subsequent subroutine takes care of the calibration of the RF Modulator (cf. Sec. 4.2.3). Graph 2 shows the coarse calibration of the offset, where 19 samples in the range between -100 and 100 are taken (later scaled to be between -1600 and 1600). The red markers on the plots are used to highlight for which offset value the mean of observation buffer is the closest to zero, along with its two neighboring points. On the first plot, for instance, the region of the minimum output in the I-channel with a minimum mean is in range between 10 and 30. Similarly, the minimum in the Q-channel is found for the offset range of -40 to -20.



Graph 2: Coarse calibration of the RF Modulator in the I (left) and Q (right) channels.

As a next step, the region containing the minimum is scanned with a finer resolution for each channel; the outcome is shown in Graph 3. The minima are detected just like for the coarse scan and give, in this example, the offset 12 and -40 for the channels I and Q, respectively.

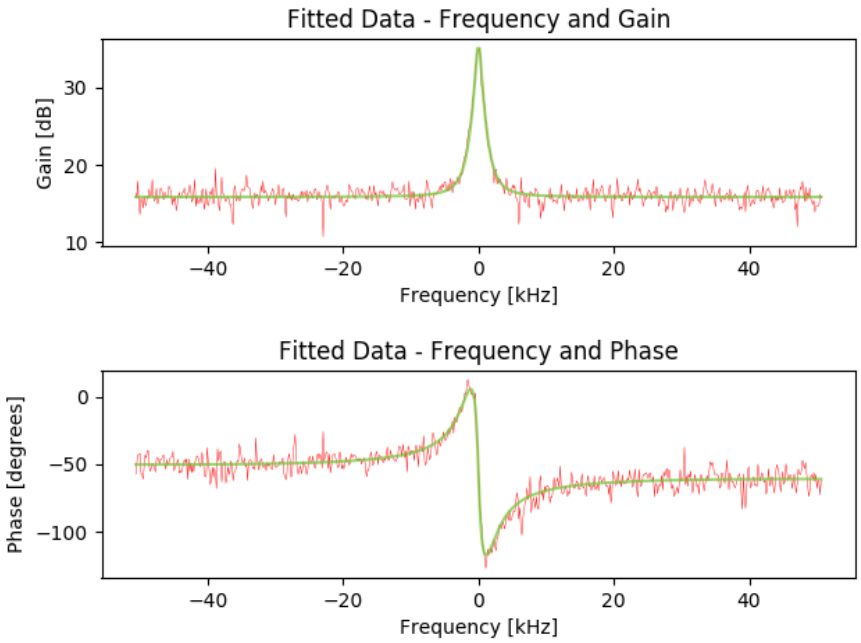


Graph 3: Fine calibration of the RF Modulator I (left) and Q (right) channels.

Finally, the Set Point module is calibrated as described in Sec. 4.2.3. The Set Point devices' test data I and Q values that result in the output that is the closest to zero are, in this specific example, -29 and -64, respectively. The convergence of the subsequent Phase Alignment algorithm (Sec. 5.1.2) depends on the goodness of these test data values.

5.1.2 Aligning the RF Feedback Phase

The algorithm of aligning the digital feedback phase w.r.t. the analog one (V_{cav} phase) was introduced in Sec. 4.2.4. The digital feedback acts in the vicinity of the central frequency while the analog feedback is efficient in the region outside the central frequency. When the two feedbacks are aligned, they act in a constructive manner at the frequencies where they overlap. Thus, the amplitude of the transfer function increases towards the central frequency, see Graph 4.

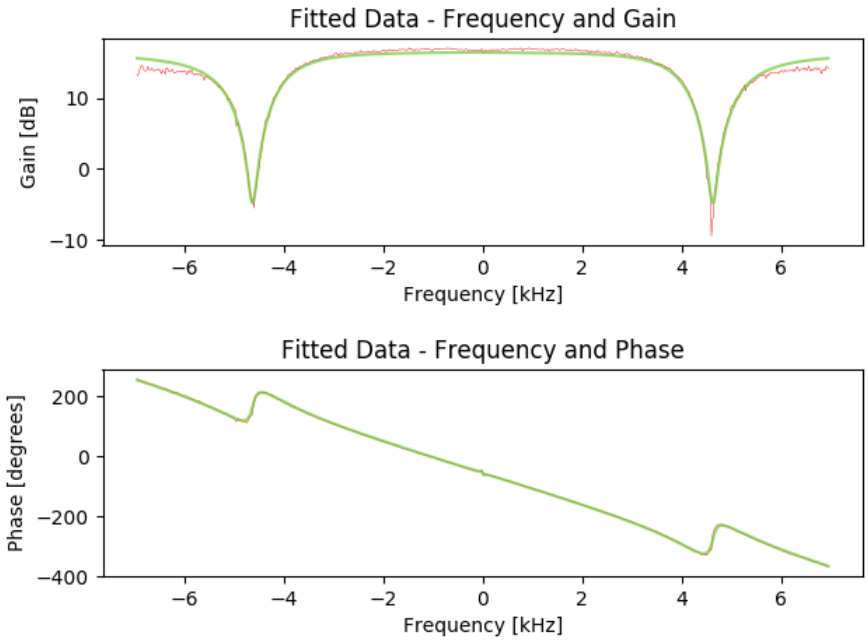


Graph 4: Measured (red) and fitted (green) overall system transfer function (top: gain, bottom: phase) for a well-adjusted phase in the Phase Alignment routine. The central frequency is the RF frequency.

The measured data (red) is obtained from the Network Analyzer (Section HELPER FUNCTION: Network Analyzer), while the fitted data (green) is calculated by the Transfer Function Fit routine (Section HELPER FUNCTION: Transfer Function Fit).

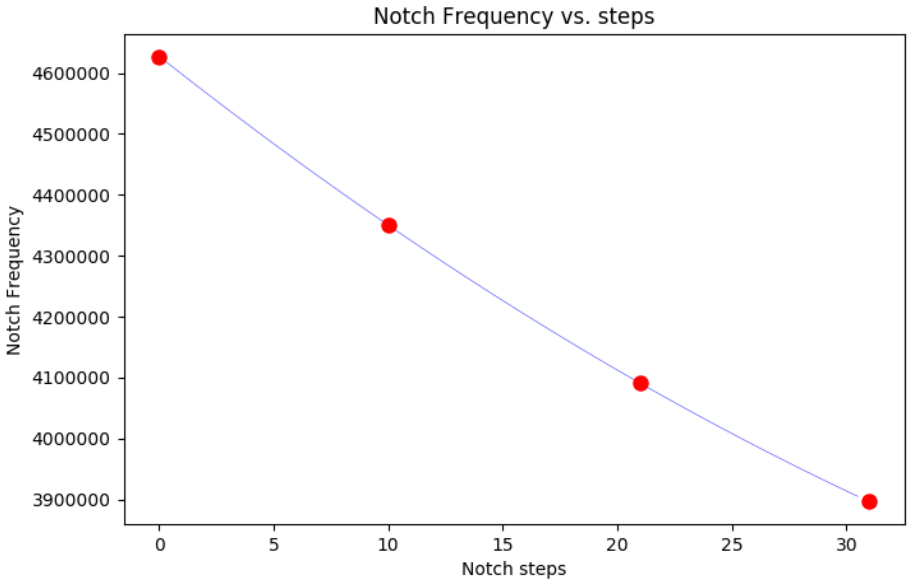
The Notch Identification routine, whose algorithm was provided in Sec. 4.2.4, re-measures the system transfer function, but this time with the notch filter included.

A corresponding typical plot is shown in Graph 5. Just as in the previous case, the measured transfer function is extracted from the observation buffer data via the Network Analyzer and the fit is done using the Transfer Function Fit. From the amplitude (gain) of the measured transfer function, the notch frequency for a given capacitor setting can be extracted.



Graph 5: Measured (red) and fitted (green) overall system transfer function (top: gain, bottom: phase) during the Notch Identification routine. The central frequency is the RF frequency.

Also, the notch frequency is measured as a function of the capacitor value in the notch filter; the result is shown in Graph 6. In general, a more or less linear behavior is expected. Depending on the resonant frequency of the klystron in a given RF line, the resonant frequency can then be adjusted to counteract this resonance.



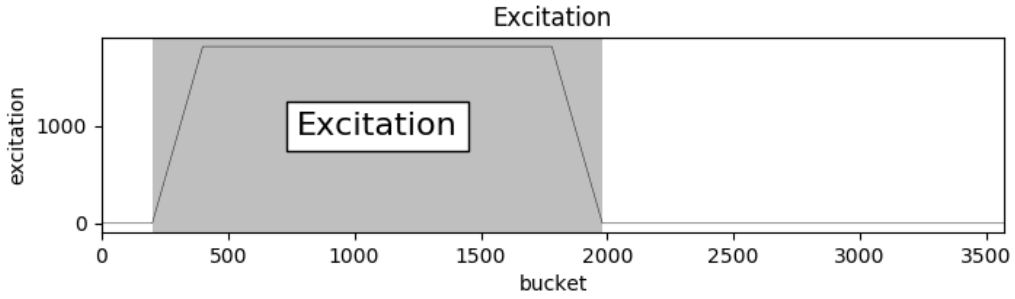
Graph 6: Output of the Notch Identification routine showing the measured notch frequency as a function of the notch filter capacitor value.

5.1.3 The Fine Alignment

In closed loop, the cavity controller is only stable in a limited range of V_{cav} phases; outside of this range, the voltage in the cavity starts to oscillate.

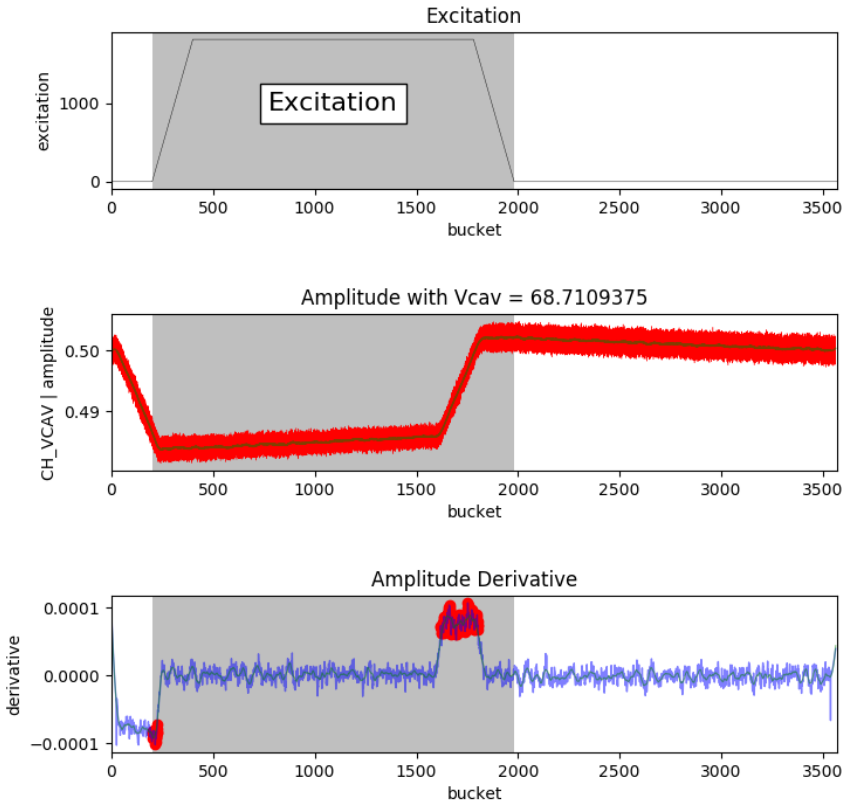
Before beginning the fine alignment routine implementation, it was interesting to perform an entire scan of the V_{cav} phase from -180° to 180°. This is done only on the test stand 5B0 that does not have a real cavity connected to it. In the real system, it should be avoided to drive the loop unstable as hardware components could potentially be damaged, even though the Switch and Protect module clamps at a certain power.

In this measurement, the trapezoidal phase modulation is injected in the Set Point module, see Graph 7. If the digital and analog feedback phases are well aligned, the effect on the cavity voltage amplitude is minimal. To find the phase for which this effect is minimal, the amplitude is first smoothed and then the derivative of the smoothed data is observed. The transients in the derivative, corresponding to the rising and falling edge of the trapezoid, are detected when they are crossing a certain, empirically-found upper and lower limit. The fewer "out-of-range" values are observed, the closer the V_{cav} phase is to the optimized value. The main task is to find for which V_{cav} phase value the number of these points is minimal.



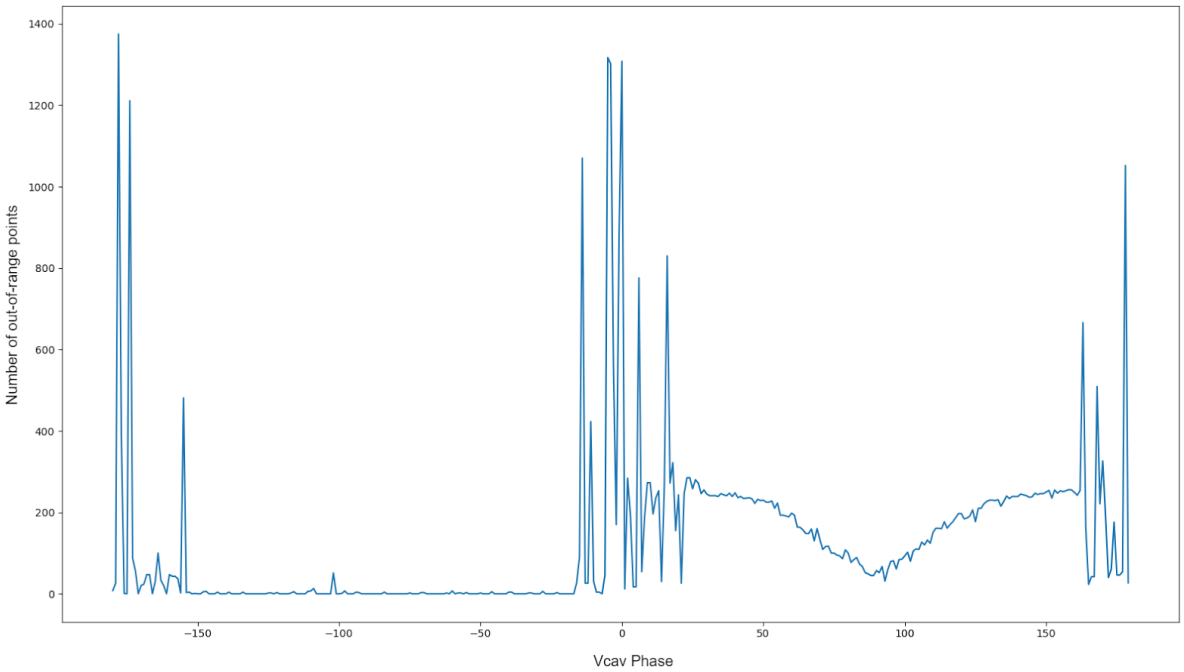
Graph 7: Trapezoid-shape phase modulation (excitation).

An example of the amplitude and derivative signals is shown in Graph 8.

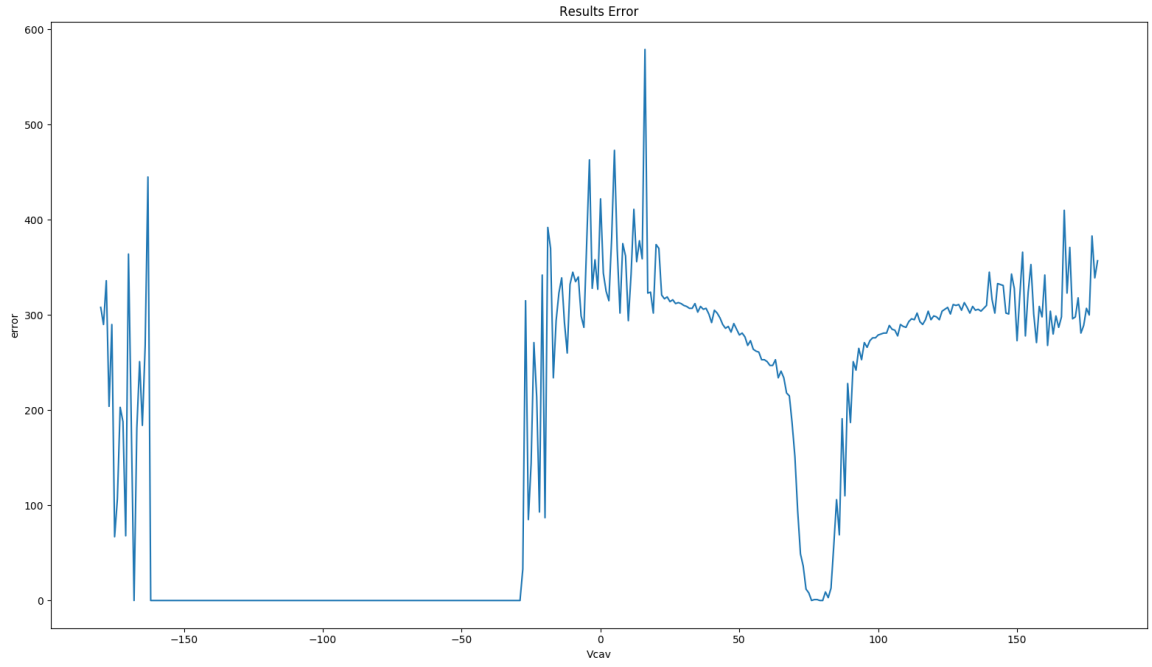


Graph 8: Observations from the Fine Alignment algorithm. The measured amplitude (top red trace) is first smoothed (top green trace) and then its derivative (bottom blue trace) is calculated and smoothed again (bottom green trace). The points on the derivative plot that cross the empirically found lower and upper limit are marked with red dots. The V_{cav} phase was 68.71° in October, 2017.

The number of out-of-range points in a full, 360-degree V_{cav} phase scan is shown in Graph 9 and Graph 10.



Graph 9: Number of out-of-range points in a 360-degree scan of the V_{cav} phase. Minimal error is given with the V_{cav} phase around 88°. Scanned on 5B0, July, 2017.



Graph 10: Number of out-of-range points in a 360-degree scan of the V_{cav} phase. Minimal error is given with the V_{cav} phase around 78°. Scanned on 5B0, October, 2017.

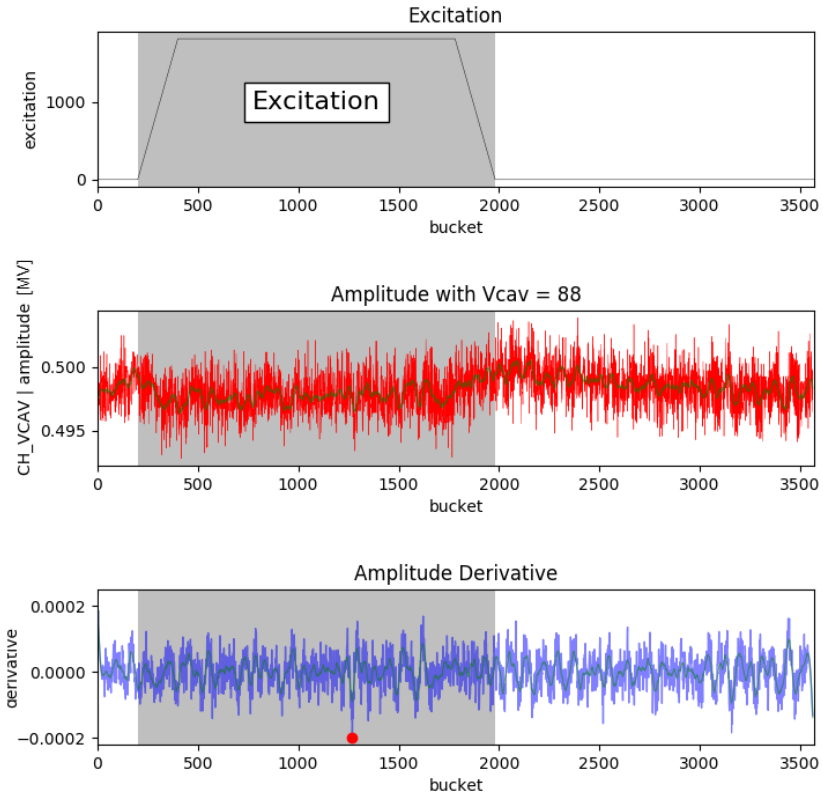
The stable and unstable regions can be nicely distinguished from these plots, Graph 9 and Graph 10. The regions where the number of out-of-range points reaches high values or fluctuates a lot are considered unstable. The only two regions where the number of out-of-range points does not fluctuate are the regions from -150° to -20° (region 1) and from 20° to 160°

(region 2) in Graph 9. Even though region 1 has less out-of-range points than region 2, the system is actually not stable in this region. To realize this, one has to observe also the voltage amplitude in the cavity. For a set point of 0.5 MV, for instance, the voltage is stably maintained in region 2; in region 1, however, the voltage in the cavity is around 1.2 MV, i.e. not correctly maintained at all. In the regions where the number of out-of-range points fluctuates, also the voltage is oscillating.

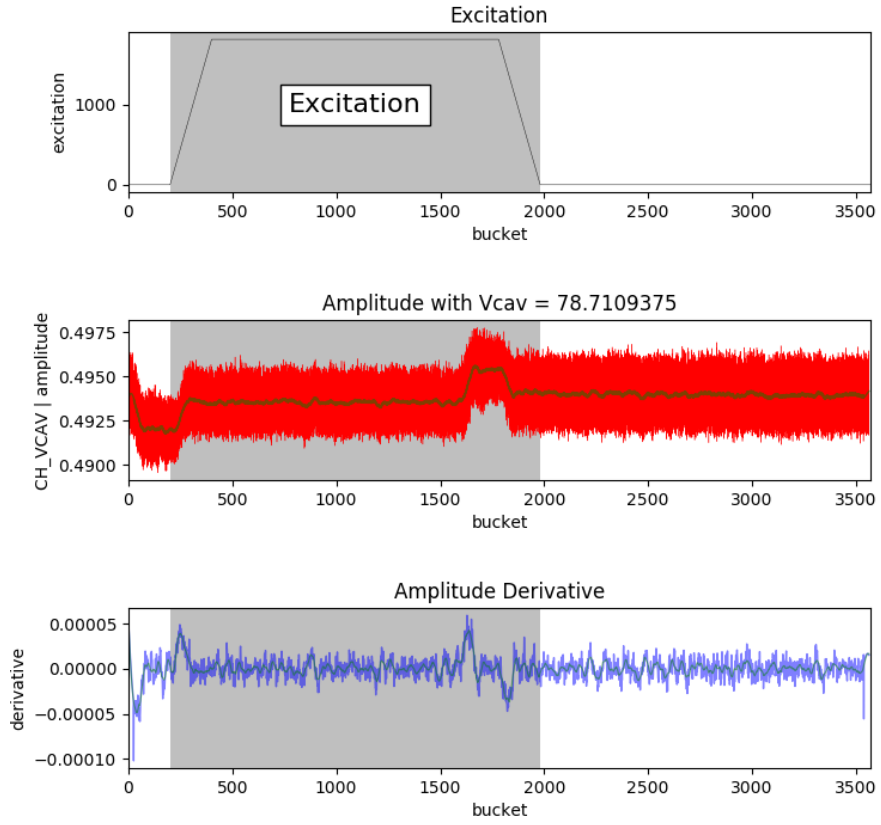
The stable region 2 contains, in addition, a clear local minimum; the location of this minimum is the calibrated V_{cav} phase. In the example of Graph 9, this minimum value is found to be 88°, whereas in the example of Graph 10, the minimum value is found to be 78°.

As already mentioned, a full scan is not desirable on the real system. Instead, on the real system, if nothing has been modified from one year to another, one would scan in the vicinity of the previous year's calibration value. If, for instance, a module was changed, then first a stable voltage region should be found manually.

At the calibrated phase of 88°, the system reacts as presented in Graph 11 (measurement executed on July 2017). In addition, the system reaction on calibrated phase of 78° is shown in Graph 12 (measurement executed on October 2017). The shaded area represents the region in time (buckets) where the phase modulation (excitation) is active. The effect on the cavity voltage amplitude is minimal for this calibrated phase.



Graph 11: Fine Alignment observations at the calibrated phase of 88°: the phase modulation (top) is applied during the shaded time range; the amplitude of the cavity voltage (center) is varies minimally with the phase modulation, which is seen also on its derivative (bottom).



Graph 12: Fine Alignment observations at the calibrated phase of 78°.

5.2 Validation and Verification

The validation is the process of evaluating the final product w.r.t. the needs and specifications of the customer. The verification is the process of evaluating the correct functioning of the same product. The software package presented here was continuously validated by regular customer feedback. For the routines that are to replace the previous MATLAB tools, it was made sure that the same type of input and output are received so that the migration is transparent for the user. The new features that were required by the user, such as roll-back possibility and improved RF Feedback phase alignment were achieved with the Backup module and Fine Alignment routine, in accordance with the customer specifications.

Concerning the verification of the code, different testing methods have been applied. Unit-testing has been used on the lowest layers of the code, and quasi-operational testing was done, whenever possible, by code-to-code comparison between Python and MATLAB using exactly the same inputs. Integration testing was performed using the test cases presented in Sec. 4.2. Because operational testing is not possible during the year when the real system is operational, the test cases were investigating using the test stand "5B0", which is an amazingly realistic simulation stand that provides developers to dynamically (regularly) validate and test their code while writing it. The test stand consists of a real LHC cavity controller: a complete set of the LLRF VME modules that are in operational state, connected to an electronically modeled RF cavity that behaves in terms of its signals just like a real cavity would.

5.2.1 Validation by End User

As already mentioned, the end users have already validated the functionality and the I/O of the Python software package. The final validation of the software performance and functionality will be performed during the next re-commissioning of the LHC RF system in Spring 2018. Should there be any modifications or additions necessary to the software package, these will be done at that stage. First non-invasive tests on the operational system could be performed already before, during the technical stops.

5.2.2 Unit Testing

On the lowest layer of the Python project, unit tests have been performed using the `pytest` framework. Those parts of the code that represent the key algorithms of the code, for instance, have been benchmarked against the outputs that were obtained for the same input from the MATLAB routines.

The unit tests cover also the Backup and Environment Variable implementations, to make sure that the rollback of the system is working reliably. All unit tests can be run from the directory containing the test files using the following command

```
pytest -s -test_*.py
```

The testing of the device backup, for instance, is done by executing the store and restore methods of the Backup class, as well as the store and restore methods of the Device class. It is then verified whether the backup is successfully created and parameters restored in the device. Instead of a single device class, also the entire RF Station class can be backed up in a similar fashion.

In more details, the steps for testing the backup are: (i) store all device properties (or all devices of one RF station in case of testing the RF station), (ii) save the value of the property of a certain device that is planned to be changed, (iii) change that property (e.g. Set Point device's property called `Phases#vcav`) and (iv) upload the changed value to the system (e.g. modified value of property `Phases#vcav`); afterwards, (v) restore all the original values from the backup and finally (vi) compare whether the saved value is the same as the one that is now restored from the backup.

The implementation of a modeled (or simulated) PyJapc package emerged when there was a need to test modifying system parameters that should not actually be uploaded to the system. Thus, instead of using the option of the real PyJapc package that prevents users from setting any value to a real system, with the modeled PyJapc package, this value is set not on the real system but on the stored local file simulating the system state. This allows the developer to more easily test the code and manipulate the state of the system during the algorithm testing.

5.2.3 Integration Testing

The two main algorithms on which integration tests have been performed are the Network Analyzer and the Transfer Function Fit. Since these algorithms have a certain complexity in their implementation, several different tests have been designed to verify their correct functioning. These tests are furthermore useful also for debugging the code should part of it malfunction.

The Network Analyzer algorithm requires the following parameters as an input: the data from Set Point module observation buffers for channels I and Q, the white noise excitation for channels I and Q, and some data-processing parameters like the sampling period. Since the buffers of the Set Point module cannot be triggered in a reproducible way, the resulting data can shift from acquisition to acquisition.

Just to mention one of the performed tests, the transformation of the I,Q data into amplitude and gain was tested separately. To do so, the first step is to test the routine with a reproducible buffer data set. The data (setpoint output from channels I and Q) is collected from the MATLAB code implementation before the point of network analysis. Optionally, data plotting can be enabled.

In case the visual check is not enough, the Python network analyzer output data is compared to MATLAB's network analyzer output data numerically.

To do so, the relative mean-square error (MSE) between the two data sets is investigated. The outcome of the test is considered to be "fail" if the MSE crosses a specified limit (0.46 in this case, determined empirically), and "success" if the relative error remains lower than that.

The integration tests of the Transfer Function Fit are done similarly to the Network Analyzer tests.

In all these tests, different parts and functionalities of the code have been inter-connected and their overall correct functioning was verified.

6. Conclusions

The aim of this Thesis work was to develop a Python software package for the commissioning of the LHC low-power level RF system. It was required to translate some existing MATLAB routines, to design a better code structure, to implement a roll-back option for the operational system, and to write a new routine based on user specifications. All these tasks were successfully accomplished.

The routines for calibrating (see Sec. 4.2.3) the RF Feedback, the RF Modulator and the Set Point module, as well as the routine for the phase alignment (see Sec. 4.2.4) of the RF Feedback have been translated from MATLAB with several improvements, keeping the input and output the same. The structure of the code has been significantly improved by modularizing it. Recurrent operations that have been performed in different MATLAB routines have now been singled out and generalized into separate units that can be used by different routines. This new structure is designed to be easily extendable for new routines that will be implemented in the future.

The Fine Alignment (see Sec. 4.2.5) routine has been implemented based on the users' request for a new type of measurement of the RF Feedback phase; the algorithm and the implementation have been entirely developed to fulfill this request. The advantage of this routine over the Phase Alignment of the RF Feedback is that no re-cabling of the system is required, and thus the adjustment the phase can be performed even on the operational system, during the year.

The Backup and Environment Variable are newly implemented data structures that have not only improved significantly the structure of the software package, but added also new functionalities to it. The Backup successfully stores and restores the system state, so the system can be rolled back at any time to one of its previous states. The Environment Variable container collects the variables during the execution of any routine for future use in another routine, which

is important since the steps of the commissioning have a certain order and depend on each other. Also, future routines will require these two containers.

Unit tests have been creating continuously during the code development in order to check the correct functioning and performance of functions and the Backup container. Integration tests passed successfully on complex routines as well.

In addition, keeping track of log messages is enabled for users to follow the code execution depending on a logging level.

The code documentation as well as some API reference is generated using the Sphinx Python tool [23]. The whole code is written using PEP8 coding standard [24].

At this stage, code implementation of the routines is tested on the LHC RF test stand "5B0". Tests on the operational cavity controllers will be performed in Spring 2018.

The Python software package has clear advantages over the previously used MATLAB package: its modularity allows for easier maintenance and modifications; the Backup and Environment variables make it reliable and safe for the user. By generalizing certain subroutines and functions, similar code can be reused in different routines and the overall complexity of the package is reduced. The well-designed structure and the documentation help furthermore new users to adapt quickly and easily to the tool. Finally, the tool is accessible to be run remotely anytime, from almost anywhere. Remaining disadvantages and potential areas of improvement are that the routines still have to be executed independently, although they follow a certain order and that there is no graphical user interface implemented for the time being.

Now that the structure of the Python package is in place and the first complex routines are in place, the remaining routines can be translated more easily in the near future. By Spring 2018, it is foreseen to complete the Python software package with these remaining routines so the entire re-commissioning can be done with these new tools.

Some improvements, like the enabling/disabling of a device within the RF line and the usage of Python decorators as populators for the get/set methods, could be added to the code to improve its overall flexibility and performance.

7. References

[1] O. Bruning, P.Collier, P. Lebrun, S. Myers, R. Ostojic, J. Poole, P. Proudlock, 2004, *LHC DESIGN REPORT, Volume 1: The LHC Main Ring*, Geneva: CERN-Scientific Information Service, ISBN 92-9083-224-0, pp. 131-154

[2] In Proceedings of the CAS-CERN Accelerator School; RF for accelerators, Edeltoft, Denmark, 8-17 June 2010, edited by R. Bailey, CERN-2011-007, pp. 259

[3] Photo of the RF cavity: http://cds.cern.ch/record/1709737 (entered 30.09.2017)

[4] S.Y.Lee, 2012, *Accelerator Physics, Third edition*, Singapore: World Scientific Publishing, ISBN-10 981-4374-94-6

[5] Photo of CERN accelerator complex together with its accelerators and detectors: https://cds.cern.ch/record/2197559/files/CCC-v2017.png (entered 29.9.2017)

[6] K.Schindl, *CERN Document Server*, The Injector Chain for the LHC, pp. 47

[7] In Proceedings of the CAS-CERN Accelerator School; RF for accelerators, Edeltoft, Denmark, 8-17 June 2010, edited by R. Bailey, CERN-2011-007, pp. 12-14

[8] D. Boussard, T. Linnecar, 1999, *Large Hadron Collider Project*, The LHC superconducting RF system, Geneva: Presented at the 1999 Cryogenic Engineering and International Cryogenic Materials Conference (CEC-ICMC'99), 12-16 July 1999, Montreal, Canada, pp. 1

[9] Photo of the cryogenic module with four RF superconducting cavities: http://anim3d.web.cern.ch/sites/anim3d.web.cern.ch/files/rf%20cavity%20occlu%20website%2009.jpg (entered 30.09.2017)

[10] K. Kostro, J. Andersson, F. Di Maio, S. Jensen, N. Trofimov, 2003, *The Controls Middleware (CMW) at CERN: Status and Usage,* In Proceedings of ICALEPCS2003, Gyeongju, Korea

[11] M. Betz, T. Levens, R. D. Maria, 2016, presentation of *PyJapc – Python tools for machine data analysis and equipment control,* photo of CMW organization together with the application layer, slide 3

[12] T. Levens, R. De Maria, M. Betz, M. Fitterer, C. Hernalsteens, 2017, *Python Tools for Control System Access,* presentation about Python Tools.

[13] M. A. L. Fernandez, S. Jackson, F. Locci, J. L. Nougaret, M. Peryt, A. Radeva, M. Sobczak, M. V. Eynden, 2007, *Front-End Software Architecture,* In Proceedings of ICALEPCS07, Knoxville, Tennessee, USA

[14] Shin, K.G.; Ramanathan, P., 1994, *Real-time computing: a new discipline of computer science and engineering,* in Proceedings of the IEEE. IEEE. 82 pp: 6–24

[15] Z. Zaharieva, M. Martin Marquez, M. Peryt, 2011, *Database Foundation for the Configuration Management of the CERN Accelerator Controls System*, Proceedings of ICALEPCS2011, Grenoble, France

[16] V. Costa, B. Lefort, 2017, *Inspector, IDE Zero Code for development of the management interface for development,* ICALEPCS17, Barcelona, Spain

[17] P. Laplante, 2007, *What Every Engineer Should Know About Software Engineering*. CRC Press.

[18] More Context Managers: http://book.pythontips.com/en/latest/context_managers.html (entered 30.09.2017)

[19] Bronshtein, I.N., Semendyayev, K.A., Musiol, G., Mühlig, H., 2015, *Handbook of Mathematics*, Part for finding weighted arithmetic mean

[20] A. Papoulis, 1977, *Signal analysis*, International Student Edition, Polytechnic Institute of New York, pp. 6-13

[21] Franklin, J. D. Powell, M. L. Workman, 1998, *Digital Control of Dynamic Systems*, 3rd Edition, Stanford, California: Addiso-Wesley, 978-0201820546

[22] Response of a linear, time-invariant system described via the impulse response and the transfer function https://en.wikipedia.org/wiki/Linear_time-invariant_theory

[23] Sphinx documentation: http://vvv.sphink-doc.org/en/stable/ (entered 30.09.2017)

[24] PEP8 standard: https://vvv.Pithon.org/dev/peps/pep-0008/ (entered 30.09.2017)