1 2 9 0

UNIVERSIDADE D
COIMBRA

Hugo Filipe V. F. E. Costa

OPTIMIZATION OF A POLARIZATION ENCODING MODULE FOR DV-QKD SYSTEMS

Dissertação no âmbito do Mestrado em Engenharia Física
Orientada pelo Professor Doutor Armando Nolasco Pinto
Co-Orientada pela Professora Doutora Maria Helena Almeida Vieira Alberto e
Apresentada ao Departamento de Física da Faculdade de Ciências e
Tecnologias da Universidade de Coimbra.

Julho de 2022

This work is supported by the European Regional Development Fund (FEDER), through the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project Q.DOT with Nr. 039728 (POCI-01-0247-FEDER-039728)];

Cofinanciado por:

COMPETE 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

Acknowledgements

Em primeiro lugar gostaria de agradecer aos meus orientadores. Ao Professor Doutor Armando Nolasco Pinto, por toda a atenção, críticas e orientação objetiva que me deu ao longo do trabalho. Consigo aprendi bastante sobre o papel de um investigador. À professora Doutora Maria Helena Almeida Vieira Alberto, que mostrou sempre disponibilidade para explicar qualquer problema teórico com que me deparei, além de que, sem o seu apoio, nunca teria sido possível chegar ao mesmo nível de detalhe e clareza nos textos apresentados nesta dissertação.

Agradeço ainda a outros membros do grupo de comunicações quânticas do IT. Ao Doutor Nelson Muga e ao Engenheiro João Prata pela sua disponibilidade constante e paciência para esclarecer qualquer questão que tinha, sendo ou não diretamente relacionada com o trabalho desenvolvido. Com este apoio nunca me senti desamparado, tendo ainda obtido muitos conhecimentos extra em áreas de interesse. Agradeço de forma igual ao Investigador Nuno Silva, e aos meus dois colegas no grupo de investigação, Diogo e Sara.

Um agradecimento especial a todos os meus amigos de licenciatura e mestrado por me terem acompanhado e impulsionado nesta jornada, especialmente à Rita Barradas, Maria Melo, Seomara Félix, José Sousa e Bárbara Matos, de entre outros mais.

Em último e não menos importante, agradecer à minha família, principalmente aos meus pais e avós. Sem a sua paciência, apoio incondicional e investimento nunca teria conseguido ultrapassar as pedras no caminho.

Obrigado!

Resumo

São cada vez mais necessários desenvolvimentos em métodos de encriptação que permitam comunicação perfeitamente segura sobre a ameaça de qualquer entidade externa. Essa necessidade surge relacionada com a continua informatização dos dados de qualquer indivíduo. O desenvolvimento de um sistema de encriptação teoricamente infalível está limitado pelo problema de distribuição de chaves, que é passível de intrusão. Certas propriedades quânticas oferecem uma solução possível para a criação de um sistema robusto de distribuição de chaves para encriptação. Exploração de propriedades quânticas, como o principio da incerteza de Heisenberg ou o teorema da não clonagem, permitem a deteção de intrusão durante a distribuição de chaves, pelo efeito que têm no aumento de uma taxa de erro. No entanto, o sistema fica mais sensível e vulnerável a erros em comparação a outros sistemas de encriptação comummente utilizados. Este aumento em dificuldade leva a que o hardware necessário seja mais complexo. O aumento da complexidade do hardware, no entanto, não compensa, atualmente, as perdas em distância e frequência de encriptação. Devido a tal muito trabalho ainda tem de ser feito em otimização e compensação de erros.

Esta dissertação foca-se na apresentação de um método de otimização de um transmissor num sistema distribuição de chaves de encriptação quânticas. Os bits quânticos nesta aplicação são codificados utilizando o estado de polarização de fotões. Este método de codificação é geralmente referido de Distribuição de Chaves Quânticas com Variáveis Discretas (DV-QKD). O transmissor utiliza um Controlador de Polarização Elétrico (EPC) que modifica a estado de polarização da luz consoante a tensão elétrica aplicada aos vários estágios que o constituem. Nesta dissertação procuramos otimizar a escolha de tensões a aplicar ao controlador de polarização elétrico. O objetivo é obter os seis estados de polarização necessários à saída do EPC no menor intervalo de tensões possível. O processo de criação da função custo, escolha e aplicação de um algoritmo de machine learning é discutido detalhadamente. Embora não sabendo todas as caraterísticas do EPC, relacionadas com a calibração, resultados que validam os dados obtidos numericamente pelo algoritmo são demonstrados. A aplicação do algoritmo, Otimização de Enxame de Partículas (PSO), como método de otimização da geração dos estados de polarização, foi relizada com sucesso. Resultados obtidos provam que transições entre os seis estados de polarização, usados em sistemas distribuição de chaves quânticas, são possiveis num intervalo de tensão inferior a 10 V em cada pino de controlo de um estágio do EPC, um valor pequeno quando comparado à gama de total de 140 V do EPC utilizado. A hipótese de utilizar outra versão deste algoritmo como método de compensação a desvios, em relação ao estado de polarização esperado, é também discutida.

Palavras chave: Criptografia Quântica, Transmissor, Estado de polarização, Variáveis discretas, Controlador de polarização elétrico, Machine learning

Abstract

Development of encryption methods that allow for perfectly secure communications under threat of any external entity are increasingly needed with continuously informatization of data from any person. The development of an encryption system infallible in theory is limited by the a key distribution problem. Certain quantum properties offer a possible solution in the creation of a robust system for distribution of encryption keys. The exploitation of quantum properties, like Heisenberg's uncertainty principle or the non cloning theorem, allow for intruder detection during key distribution, due to the error rate increment introduced. However this makes the overall system much more sensible and prone to errors, when compared to the most commonly used encryption methods. This increased difficulty makes it so the hardware necessary is much more complex. This increment in complexity does not yet compensate for loses in distance and communication speeds when compared to other commonly used non quantum encryption methods. Due to that a lot of work still needs to be done in optimization and error compensation.

This dissertation focuses in the presentation of a method to optimize a quantum key distribution transmitter. The quantum bits utilized in this application are codified in the polarization state of photons. This encryption method is commonly referred as Discrete Value Quantum Key Distribution (DV-QKD). The transmitter employs a Electrical Polarization Controller (EPC) that modifies light state of polarization according to the voltage applied to several stages comprising the device. In this dissertation we look to optimize the voltage choices that we will apply to the EPC. The goal is to obtain every required state of polarization at it's output in lowest voltage interval between each state possible. The creation process of the cost function, the choice and application of a machine learning algorithm is discussed in detail. Even though we don't know every characteristic of the EPC, related to it's calibration, results that validate the data obtained numerically by the algorithm are presented. The application of the algorithm, Particle Swarm Optimization (PSO), as a method to optimize the generation of polarization states, was done successfully. Results show that transitions between six states of polarization used in QKD systems can be generated using a voltage range of 10 V for each stage controlling pin, a small value when compared to the standard full range of the drive voltages, which in the employed EPC equals 140 V. The hypothesis of using a variation of this algorithm as a method to compensate deviations, in relation to the expected state of polarization, is also discussed.

Keywords: Quantum cryptography , Transmitter, State of Polarization, Discrete Variable, Electrical Polarization Controller, Machine Learning

Contents

Acronyms **viii**

1 Introduction **1**
1.1 State of the Art . . . 2
1.1.1 Real world implementations and results . . . 4
1.2 Motivation . . . 5
1.3 Objectives . . . 5
1.4 Dissertation Structure . . . 5

2 Cryptography and Quantum Key Distribution Concepts **7**
2.1 Fundamentals of Conventional Cryptography . . . 7
2.2 Quantum Characteristics . . . 7
2.3 Quantum Key Distribution . . . 8
2.3.1 Codification schemes . . . 8
2.3.2 System Types . . . 9
2.4 Introduction to electromagnetic waves and polarization . . . 9
2.4.1 Mathematical representation of polarization . . . 10
2.5 Polarization encoding . . . 12
2.6 DV-QKD protocols . . . 13
2.7 Information Reconciliation and Privacy Amplification . . . 14
2.8 Security Issues of QKD Systems . . . 15
2.9 Polarization encoding advantages and disadvantages . . . 16

3 Optimization of SOP transitions **17**
3.1 Electrical Polarization Controller . . . 17
3.1.1 Working principles . . . 17
3.1.2 Voltage characterization . . . 18
3.2 Applying Machine learning . . . 18
3.2.1 The Cost function . . . 19
3.3 Choosing the algorithm . . . 19
3.4 Particle Swarm Optimization . . . 19
3.5 Choosing the start particle space . . . 21
3.6 Algorithm Implementation . . . 22
3.7 Parameter decisions . . . 23
3.8 Numerical results . . . 24

4 Laboratory Validation **26**
4.1 Laboratory Implementation . . . 27
4.2 Laboratory results . . . 29
4.3 Laboratory results uncertainties . . . 34

5 Conclusions **39**
5.1 Future work . . . 39

A PSO optimization code **44**

B PSO adjustments code **68**

Acronyms

CB	Circular Basis
DB	Diagonal Basis
EPC	Electronic Polarization Controller
IR	Intercept - Resend
LB	Linear Basis
PNS	Photon Number Splitting
PSO	Particle Swarm Optimization
QBER	Quantum Bit Error Rate
QKD	Quantum Key Distribution
QUBIT	Quantum Bit
SOP	State of Polarization
SPD	Single Photon Detector

List of Tables

4.1 α and Δ values for $S_1 = 0.517$, $S_2 = 0.114$, $S_3 = -0,847$ input SOP. 29
4.2 Voltage values to apply to the EPC calculated through PSO for $S_1 = 0.517$, $S_2 = 0.114$, $S_3 = -0,847$ input SOP. 29
4.3 Voltage intervals for the 2 Basis and 3 Basis combinations calculated from Table.4.2. 30
4.4 Measured SOP's, in the lab, after applying the voltage values from Table.4.2 to the EPC pins. 30
4.5 Voltage reference points decided through previous measurement analysis. 31
4.6 Measured SOP's, in the lab, after applying the voltage values from Table.4.5 to the EPC pins. 32
4.7 Combination of voltage values, obtained in the lab, that result in accurate measurements of the desired SOP's 34
4.8 SOP's measured after applying the voltage combinations in Table.4.7 to the EPC pins. 34
4.9 Voltage intervals, obtained in the lab, for the 2 Basis and 3 Basis combinations. 34
4.10 Standard deviation measurements with and without mechanical interactions with the EPC. . 35
4.11 Standard uncertainties as a angle in the Poincaré sphere surrounding the predicted SOP. . . 35
4.12 Standard uncertainty associated with the selected sample of measured values. 36
4.13 Combined uncertainty associated with the measured values and ADC characteristics. 36
4.14 Combined uncertainty associated with the measured values, ADC characteristics and polarimeter characteristics. 37
4.15 Expanded uncertainty values associated with a 95% probability of the next measurement being inside the calculated interval. 37
4.16 Final uncertainty values, of the full laboratory setup, as a angle in the Poincaré sphere surrounding the predicted SOP. 37

List of Figures

1.1 Basic QKD model [52]. 1
1.2 Fiber-coil polarization controllers. 2
1.3 Fiber Squeezer polarization controller. 2
1.4 electro-optical switch example setup. 3
1.5 $LiNbO_3$ Phase Modulator [43]. 4

2.1 MDI-QKD structure [24]. 9
2.2 Representation of the electromagnetic field for linear polarization [54]. 9
2.3 Representation of multiple polarization's [54]. 10
2.4 Polarization ellipse. 11
2.5 Poincaré's sphere with parameters α and β. 11
2.6 Poincaré sphere representation of the polarization states. 12
2.7 Example of polarization based BB84 protocol setup [1]. 13
2.8 Graphical representation of the two SOP's used in the B92 protocol [1]. 14

3.1 Picture of the EPC. 17
3.2 EPC schematic. 17
3.3 Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis. 18
3.4 Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis. 18
3.5 PSO example for a function $y = x^2$: iteration 1. 20
3.6 PSO example for a function $y = x^2$: iteration 2. 20
3.7 PSO example for a function $y = x^2$: iteration 3. 20
3.8 PSO example for a function $y = x^2$: iteration 4. 20
3.9 Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis, red rectangles exemplify areas where the voltage values are fairly constant. 21
3.10 Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis, red rectangles exemplify areas where the voltage values are fairly constant. 21
3.11 Voltage intervals obtained for QKD application using 2 basis, x-axis number of stages used and y-axis maximum voltage interval. 24
3.12 Voltage intervals obtained for QKD application using 3 basis, x-axis number of stages used and y-axis maximum voltage interval. 24

4.1 Laboratory setup (part one). 27
4.2 Laboratory setup (part two). 27
4.3 Close-up of EPC setup. 28
4.4 Difference between the measured SOP and the numerical predicted H SOP in the poincaré sphere, $\gamma = 11.59°$ 30
4.5 Difference between the measured SOP and the numerical predicted V SOP in the poincaré sphere, $\gamma = 70.36°$ 30
4.6 Difference between the measured SOP and the numerical predicted +45 SOP in the poincaré sphere, $\gamma = 107.68°$ 30
4.7 Difference between the measured SOP and the numerical predicted -45 SOP in the poincaré sphere, $\gamma = 146.05°$ 30
4.8 Difference between the measured SOP and the numerical predicted right circ SOP in the poincaré sphere, $\gamma = 132.74°$ 31
4.9 Difference between the measured SOP and the numerical predicted left circ SOP in the poincaré sphere, $\gamma = 39.71°$ 31
4.10 H SOP reference point in the poincaré sphere 32

4.11 V SOP reference point in the poincaré sphere 32
4.12 +45 SOP reference point in the poincaré sphere 32
4.13 -45 SOP reference point in the poincaré sphere 32
4.14 Right circ SOP reference point in the poincaré sphere 33
4.15 Left circ SOP reference point in the poincaré sphere 33
4.16 Polarimeter graphical user interface (GUI). 33
4.17 Example representation of SOP uncertainty angle in the poincaré sphere. 37
4.18 Example projection of the uncertainty angle in the xy-axis. 37
4.19 Maximum polarization error observed in the GUI due to mechanical interactions with the EPC, yellow dot is the current measurement and the red dot is the measurement before the interactions. 38

Chapter 1

Introduction

In today's world data security is increasingly an important subject. Our personal, financial, and health data is transmitted over communications networks. With the growth in computational power we have been experiencing over the years and more recently with the development of quantum computers, traditional encryption methods may be at risk.

The most common encryption methods used currently are based on asymmetric keys/public key cryptography. This method uses two different keys for encryption and decryption, and top of that the decryption key can't be determined from the encryption key, at least in a reasonable amount of time. Generally, deriving the private key would be theoretically possible, but the computation would be too complex to achieve the results in a viable amount of time (millions of years) [51].

Like public-key encryption, quantum communication allows for key establishment through an untrusted network. The security of Quantum Key Distribution (QKD) is based not on the difficulty of mathematical problems but on the laws of quantum mechanics, and can be proven even against an eavesdropper, Eve, who has unbounded computational ability. Fig.1.1 exemplifies the structure of QKD [52]. In the image we see the 3 main intervening parties, Bob as the receiver, Alice as the sender and Eve as an eavesdropper trying to steal the data, by listening to both the quantum and classic channel.

The key is shared through the quantum channel using some communication protocol like BB84 (see more Section 2.6). In most cases de protocol consists on a random selection of basis for codification of the Quantum Bits QUBITS in Alice's side and a random selection of basis for measurement on Bob's side followed by classical post-processing (Section 2.7).

In the classical post-processing the basis for codification used by Alice and measurement used by Bob are compared through the classical channel, and the encryption/decryption key is determined. In this step the Quantum Bit Error Rate (QBER) is also calculated, caused by effects other than the random selection of basis by Alice and Bob. It might be caused by Eve or other some other unknown factor, but it is always assumed that Eve might be listening. If QBER goes over a certain threshold, 11% for the BB84 protocol [7], the communication is considered as insecure, and that key is not used to encrypt data.

Because of the nature of the quantum states there is no way for Eve to measure the QUBIT without increasing the QBER.

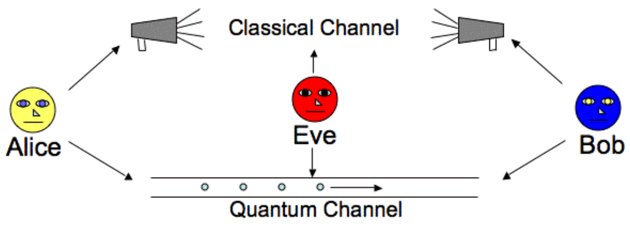


Figure 1.1: Basic QKD model [52].

When the key is prepared with the same length as the message to be sent, and is used only once (one-time pad; OTP), ciphertext cannot be decrypted by any amount of computation, even by the most powerful computers. This kind of security is referred to as *information theoretic security* [1](pp 73-74) [6].

For the purpose of this dissertation, QKD will be applied using the polarization of single photons to codify the QUBIT's.

In conclusion, the underlining issue of current encryption methods is sharing the encryption/decryption key securely, QKD is able to solve that issue.

1.1. State of the Art

As bit rates increase to meet expanding demand, systems have become increasingly sensitive to polarization-related impairments. These include polarization mode dispersion (PMD) in optical fibers, due to different velocities in the two axes of the fiber (slow and fast), polarization-dependent loss (PDL) in passive optical components, differential-group delay (DGD) in electro-optic modulators, and polarization-dependent gain (PDG) in optical amplifiers. Because polarization-induced penalties are time dependent, mitigation of the polarization-related impairments must be dynamic and adaptive to both predictable and random variations.

Polarization controllers are the simplest solution. The most rudimentary ones are **Fiber-Coil polarization controllers**.

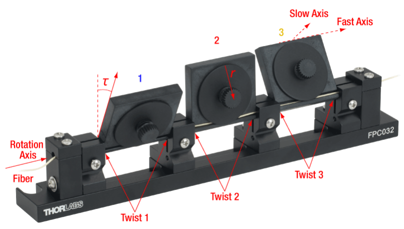


Figure 1.2: Fiber-coil polarization controllers.

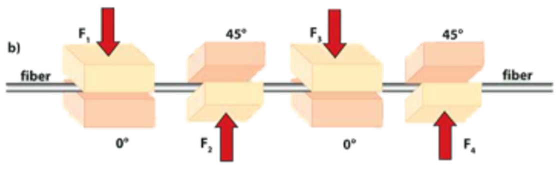


Figure 1.3: Fiber Squeezer polarization controller.

This devices, Fig.1.2, are manually adjusted and usually only serve as passive components or for demonstration purposes in the lab. They are comprised of three waveplates, two quarter-wave plates at the borders and a half-wave plate at the center. It uses the elasto-optic properties of silica fibers to control the output state of polarization [3]. Other devices that also makes use of the elasto-optic effect, variations in the refractive index of optical fiber due to mechanical stress, are the **Electromagnetic Fiber Squeezers**. Their working principles can be seen in Fig.1.3.

Using a complete different scheme are the **Faraday Rotators**, based on the Faraday effect, a magneto-optic effect involving transmission of light through a material when a longitudinal static magnetic field is present. The state of polarization is rotated as the wave traverses the device [19].

Finally like the polarization controller used in this dissertation, they are devices based in **Electro-Optic Crystals**. They work trough the characteristics described by the Pockels effect, a change in the refractive index linearly proportional to the electric field. This effect can only occur in certain crystalline solids that lack inversion symmetry. A very commonly used crystal in such applications is Lithium Niobate ($LiNbO_3$).

From the 4 presented schemes only two of them are usually used in QKD. The choice comes down to the characteristics that are most relevant to the application, QUBIT polarization codification or polarization random drift compensation . Lithium Niobate polarization controllers when compared to fiber-squeezers, like the PolaRITE II from general photonics [49] [50], have much faster response times (100 ns < 30 μ), but lack in most other parameters such as insertion loss (3 dB > 0.05 dB), polarization-dependent loss (0.2 dB > $\approx$ 0 dB) and activation loss (0.15 dB > 0.01 dB), they have similar return loss [49] [50]. The values can still be considered low but certainly add up over distance. The fiber-squeezer is also more wavelength independent 1260 $\rightarrow$ 1650 compared to 1525 $\rightarrow$ 1620 in the Lithim Niobate EPC.

So, the Lithium Niobate EPC comes with the drawback of being more expensive and harder to use due to all its calibration parameters, as we will see in Chapter.3. Nevertheless, there is still a very big advantage, the operation frequency. $LiNbO_3$ EPC can achieve 10MHz, 3 orders higher than the 33kHz of the fiber squeezer.

For **random drift compensation** at the receiver in a QKD protocol, depending on the speed of the system, the use of a fiber-squeezer polarization controller still has it's place since they are easier to implement and the speeds achieved are usually good enough until we reach the Gb/s region. At those speeds a $LiNbO_3$ EPC is certainly necessary, even if it will cause more losses at reception.

For **codifying** the QUBIT's in the SOP of light at the transmitter, the use of a fiber-squeezer polarization controller with a frequency of 33kHz is very far from the from the expected for a real world implementation, even $LiNbO_3$ polarization controllers do not allow GHz state preparation frequencies, common in present-day state of the art QKD setups. Current state-of-the-art reports a BB84 quantum states generation at 5 GHz pulse repetition rate over 151.5 km using a phase modulator, presented further in this section, to encode quantum information on single-photons polarization [12]. Other transmission setups can be used to generate the required SOP's.

The simplest QKD configuration for a transmitter utilizes multiple independent laser sources, one for each SOP required [8]. However, it appears to be hard to guarantee the indistinguishability of pulses emitted from different lasers, resulting in the system's vulnerability. To guarantee the indistinguishability of the pulses emitted by different lasers, the repetition rate is limited to the order of a few hundred mega hertz [10] [2].

Another way to approach this issue would be by using a single photon source and four to five passive linear polarizers. The output polarizer would be chosen randomly trough the use of an electro-optical switch, it would then pass through a passive coupler, connected to an optical fiber [9] [2]. In a setup similar to the one presented bellow, Fig.1.4.

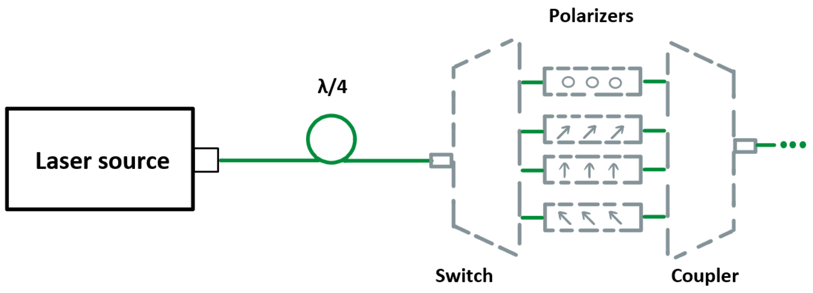


Figure 1.4: electro-optical switch example setup.

Since the light provided by lasers has linear polarization, a quarter waveplate is used to convert to circular, in that way we can use the polarizers to achieve all the necessary SOP's. Limitations in this setup will be connected to the switch speed, and insertion/return loss of the coupler/switch.

The most relevant transmission setups make use of **Phase Modulators** (PM), Fig.1.5. They are not used with the intent to turn any input SOP into any output SOP, they are instead, used to achieve certain SOP's that have direct application in QKD.

Understanding their theoretical application is not complicated, calculations trough Jones notation makes it clear, Section.2.4.1. The pulses injected in the PM have a polarization of defined by $(|H\rangle + |V\rangle)/\sqrt{2}$ [10] [11], in Jones notation this is the +45 SOP. We control the relative phase between $|H\rangle$ and $|V\rangle$ by applying a voltage on the PM to change its birefringence. At the output we will have $\left(|H\rangle + e^{i\varphi}|V\rangle\right)/\sqrt{2}$. Knowing that $|-45\rangle$, $|\text{Right circ}\rangle$, $|\text{Left circ}\rangle$ can be represented in Jones notation as:

$$|-45\rangle = (|H\rangle - |V\rangle)/\sqrt{2} \tag{1.1}$$

$$|\text{Right circ}\rangle = (|H\rangle - i|V\rangle)/\sqrt{2} \tag{1.2}$$

$$|\text{Left circ}\rangle = (|H\rangle + i|V\rangle)/\sqrt{2} \tag{1.3}$$

We see that we can achieve 4 SOP's, usable in a QKD protocol like BB84, Section.2.7, by having φ equal to 0, π, $\pi/2$ and $3\pi/2$ respectively.

Thus far, several polarization modulation schemes have been proposed , the first of which is based on the balanced Mach–Zehnder (MZ) interferometer [10]. In this setup two orthogonal polarization components enter different arms of the interferometer with the help of a polarization beam splitter, after that one of the components experiences a phase shift induced by the modulator. However, fiber Mach-Zendner interferometers are very sensitive to external environmental disturbances.

Another type of polarization modulation scheme is, just like most electro-optic crystals EPC's, designed using state-of-the-art proven lithium niobate (LiNbO_3) technology, Fig.1.5. Each component, $|H\rangle$ and $|V\rangle$ propagates along one axis of the waveguide. By applying a control voltage to the phase modulator, a phase shift is introduced between the two components, resulting in a polarization modulation.

Modulation affects only one axis, phase difference between orthogonal polarization components is in that way produced.

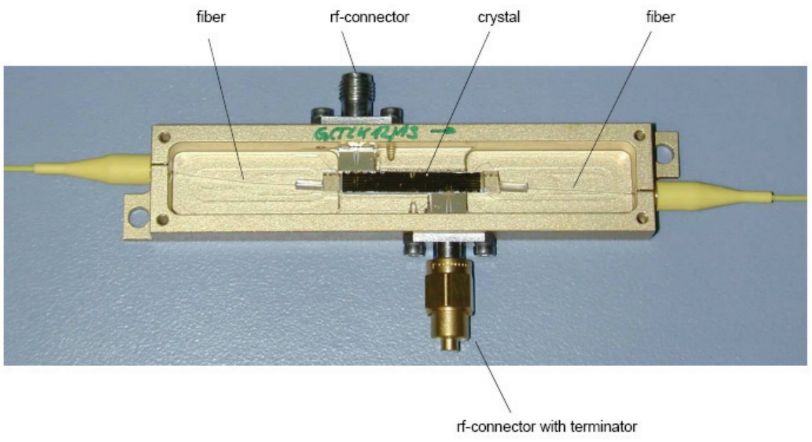


Figure 1.5: $LiNbO_3$ Phase Modulator [43].

These modulators offer the optimum for industrial QKD systems. They can achieve frequencies of up to 40Gb/s, over 3 orders when compared to $LiNbO_3$ EPC's. They also seem to utilize a smaller voltage range compared to $LiNbO_3$ EPC's.

The issue added due to this method, besides the already present PDL in other setups, is differential-groud delay (DGD) caused by the birefringence of the crystal and it being anisotropic [8]. Anisotropy is, in physics, the quality of exhibiting properties with different values when measured along axes in different directions. A delay added to light propagated in one axis of the $LiNbO_3$ crystal is seen when compared to the other axis (≈10 ps for a common $LiNbO_3$ modulator) [10] [14]. This effect is deterministic. Two methods to mitigate it are by adding a highly birefringent fiber [14] after it or, another similar phase modulator [8], the component that passed along the fast axis of $LiNbO_3$ inside the transmitter modulator travels along the slow axis of an identical crystal at the receptor, we do that by applying a 90° rotation to the light field before entering the second PM.

The last scheme is based on interferometers that use either Sagnac loops or Faraday mirrors with single-mode fiber (SMF) for optical input and output [10]. This method, due to its nature, exhibits superior stability compared with the previous two methods. They are much harder to implement.

The scheme used in this dissertation for generating the SOP's will make use of a $LiNbO_3$ Electronic Polarization Controller (EPC). The selected EPC for SOP generation proves to be a viable solution due to its fast response time (<100 ns), high stability, low insertion loss (<3 dB), low polarization dependent loss (PDL), and small size.

1.1.1. Real world implementations and results

Previous QKD demonstrations include: Satellite-to-ground Free Space Optics (FSO) link demonstration, FSO link between two locations in the Canary Islands [37] [38] [39], MDI-QKD over 404 km of ultralow-loss optical fiber and Application of an optimized four-intensity decoy-state over 311 km of a standard optical fiber [40]. Communications at room temperature have already been demonstrated possible over a 100-kilometre distances [41], and even greater distances have been reached, but due to channel losses detectors cannot withstand much further, different solutions are being constantly sought.Given that quantum states cannot be amplified, fiber attenuation limits the distance. Advanced new technology, like a quantum repeaters [42], has been a focal point in the field as professionals aim for more considerable distances.

Field tests of QKD application in fiber networks have been reported in China, Switzerland, South Africa, and so on. Especially, China has built the world's longest quantum secure communication backbone network with a fiber distance of 2000 km [10].

In Portugal some advances have also been made. In 2019, in collaboration with IT, non wire QKD were for the first time demonstrated, being the connection done over a distance of 180 m. There is already a optical fibre connection of over 20 km that connects Gare do Oriente in Lisbon to Pragal, in Almada that allows for transmission of information encrypted using quantum keys. In 2020 a theoretical model for polarisation manipulation using electronic polarisation controllers (EPC's) based on fibre squeezing was published, in which the experimental implementation used a field programmable gate array (FPGA) board to electrically control the four waveplates of the EPC, reaching a rate of 500 qubit/s [2]. Some advancements have also been done in CV-QKD like presenting a novel receiver configuration for CV-QKD that allows for passive polarization drift compensation [44]. Methods for full polarization random drift compensation for DV-QKD have also been published [45]. Experimental demonstrations of polarization-state generation algorithms using off-the-shelf components, have also been recently developed [46].

1.2. Motivation

The current implementation of QKD at Instituto de Telecomunicações in Aveiro (IT) relies on an electromagnetic fiber squeezer polarization controller that, by squeezing/bending the fiber optic in a certain way, induces refractive index changes, making it possible to define the output State of Polarization (SOP) of photons. Due to its mechanical nature, this implementation is not able to achieve key rates that modern QKD systems would require, key rates above 1kHz are far away from being achievable [2].

In my work I will use a different kind of EPC, based on electro-optic crystals (Lithium Niobate Polarization Controler). In IT we have two setups of the previously referred EPC one with 6 stages and one with 8 stages. Each of the stages can be actuated independently, and act like a waveplate. This device also comes with a Low insertion loss (<3 dB), Low polarization dependent loss (PDL) and a response time under 100 ns. Since it applies the Pockels effect, changes or production of birefringence in an optical medium induced by an electric field in the crystals by which the photons pass through, we have a very fast responding device on itself capable of achieving key rates of over 10 MHz, this value is 4 orders of magnitude higher than the fiber-squeezing based counterpart controllers. The EPC will be used as a transmitter in a QKD system. In QKD we need to be able to achieve between 4 and 6 predefined SOP's for the light at the EPC output.

The main goal of the work presented in this dissertation is to find a method that allows us to calculate optimum values for the voltages that should applied to the EPC pins. Finding optimum values means minimizing the voltage differences necessary to apply to the EPC pins while transitioning between each of the necessary SOP's. This reduction in the required range voltages for each waveplate represents a positive realization towards practical and efficient implementation of QKD protocols employing polarization encoding, as it will allow using simpler electronic drivers to control the polarization encoding subsystems.

1.3. Objectives

The goal of this dissertation is to apply, and optimize, the chosen EPC in the current QKD setup at Instituto de Telecomunicações in Aveiro.

The functional requirement for the system is:

- The ability to generate 6 distinct polarization states, that belong to each of the 3 basis, according to a specific quantum protocol, for any given SOP at the input.

The non-functional requirements for the system are:

- The polarization error in the photons should be under 10^{-4} for each of it's Stokes parameters;
- The voltages used to control the EPC should vary between -70 and 70 V with a resolution of around 10 mV;
- The voltage steps at the EPC voltage inputs should be minimized between each SOP change.

To summarize, we have an EPC at IT with 2 distinct configuration 6 and 8 stages/waveplates. The only information known regarding the EPC is the information given by the datasheet, the calibration parameters have not yet been found and that is not a goal of this work. We have information regarding the equations that correlate each stage waveplate characteristics to it's voltage values. The waveplate characteristics are it's phase difference between the fast and slow axis and the angle of the fast axis. There are multiple combinations of this two variables, for each waveplate, that achieve the same output SOP, the number of possible combinations increases with the number of waveplates. It's important to know what combinations make the most sense for a given application. Knowing the waveplate characteristics allows the calculation of the effect it might have on a certain input SOP. Using the information given by the waveplate characteristics, it's connection to the voltage values and the input SOP an optimizing method can be developed.

1.4. Dissertation Structure

The document is divided in 5 chapters containing the following information:

- Chapter.1 - Yields the motivation and goal for this dissertation, after that the state of the art is presented. The chapter is closed by presenting milestones/achievements done outside and inside Portugal.

- Chapter.2 - Some important information regarding quantum characteristic used in QKD. Common implementations are presented and it's security issues analysed. Some theoretical concepts related to a mathematical representation of polarization are also detailed. The chapter ends by discussing advantages and disadvantages of polarization encoding.
- Chapter.3 - Here we go in detail on the inner workings of the EPC and it's voltage characteristics. From there we describe how machine learning was applied to achieve our goals.
- Chapter.4 - The voltage intervals obtained numerically are validated in this chapter, some uncertainty analysis regarding the laboratory setup is also presented.
- Chapter.5 - Discussion regarding the results obtained in this dissertation is presented. Goals for the future of the setup are also talked over.

Chapter 2

Cryptography and Quantum Key Distribution Concepts

This chapter will start by covering the fundamentals of cryptography systems and after that some important quantum concepts relevant to the development of QKD are presented. After that we present the mathematical basis behind electromagnetic waves and polarization and how it can be applied to QKD. We also take a look at some of the more popular QKD protocols, and it's security concerns.

2.1. Fundamentals of Conventional Cryptography

The role of a cryptographic system is to encrypt the message in a way that the eavesdropper cannot obtain the data, and decrypt the cryptogram in a way that the final user can receive the intended message.

Key based algorithms can be broadly categorized in two ways:

1. Symmetric algorithms
2. Asymmetric algorithms

The first method utilizes a symmetric key, for both parties, that is used to encode the information we want to send over a channel. Since both user have access to the encryption key, both of them are able to scramble and unscramble the information they want to trade [15] [16] [17].

The second method uses two different keys for encryption and decryption, on top of that the decryption key can't be determined from the encryption key, at least in a reasonable amount of time. Due to this factor public key systems based in this method have been very popular over the last years [20]. The basic idea of a public key system is that the public key can be easily derived from the private key, but the private key cannot be practically derived from the public key. Generally, deriving the private key would be theoretically possible, but the computation would be so complex that it would take millions of years with **current** computers [51].

All of this paradigms change with the development of quantum computation, as we will begin to see in Section.2.2.

2.2. Quantum Characteristics

Every measurable - observable (position, momentum or angular momentum) - is associated with a Hermitian operator with a complete set of eigenkets. According to P. A. Dirac "A measurement always causes the system to jump into an eigenstate of the dynamical variable that is being measured." [21] Dirac's statement can be formulated as the following postulate: An exact measurement of an observable with operator A always yields as a result one of the eigenvalues $a^{(n)}$ of A. Thus, the measurement changes the state, with the measurement the system is "thrown into" one of its eigenstates. Each one of the eigenstates has an associated probability of being measured. This is the fundamental characteristics of quantum systems, and it's one of the main properties that will allow us to transmit the quantum version of bits, QUBITS, in a public quantum channel and achieve perfect security.

The fundamental characteristics of quantum systems that differ from classic computation can be summarized in 3 concepts [1](pp 178):

1. **Linear superposition** - Contrary to the classical bit, a quantum bit or QUBIT can take not only two discrete values 0 and 1 but also all possible linear combinations of them. This is a consequence of a fundamental property of quantum states: it is possible to construct a linear superposition of a quantum state $|0\rangle$ and quantum state $|1\rangle$.

2. **Quantum parallelism** - The quantum parallelism is a possibility to perform many operations in parallel, which represents the key difference from classical computing. Namely, in classical computing, it is possible to know what is the internal status of the computer. On the other hand, because of no-cloning theorem, it is not possible to know the current state of quantum computer. This property has lead to the development of Shor factorization algorithm, which can be used to crack the **Rivest-Shamir-Adleman (RSA) encryption protocol**. Some other important quantum algorithms include Grover search algorithm, which is used to perform a search for an entry in an unstructured database; the quantum Fourier transform [22], which is a basis for a number of different algorithms; and Simon's algorithm. The quantum computer is able to encode all input strings of length N simultaneously into a single computation step. In other words, the quantum computer is able simultaneously to pursue 2^N classical paths, indicating that quantum computer is significantly more powerful than classical one.

3. **Entanglement** - At a quantum level, it appears that two quantum objects can form a single entity, even when they are well separated from each other. Any attempt to consider this entity as a combination of two independent quantum objects given by tensor product of quantum states fails, unless the possibility of signal propagation at superluminal speed is allowed. These quantum objects that cannot be decomposed into tensor product of individual independent quantum objects are called *entangled* quantum objects. Given the fact that arbitrary quantum states cannot be copied, which is the consequence of no-cloning theorem, the communication at superluminal speed is not possible, and as consequence the entangled quantum states cannot be written as tensor product of independent quantum states. Moreover, it can be shown that the amount of information contained in an entangled state of N-qubits grows exponentially instead of linearly, which is the case for classical bits.

Being that the RSA encryption protocol is one of the most popular public key system protocols [53], we can now clearly understand why we might need quantum based key distribution methods QKD in the near future.

2.3. Quantum Key Distribution

Significant advances have been made recently in quantum computation. There are various companies developing medium scale quantum computers. Given that most of the today's cryptographic systems rely on their computational difficulty to hack, quantum computing present a serious challenge for the modern cybersecurity systems [23]. As an example lets look at the popular RSA protocol. To break this system It's necessary to determine the period r of the function $f(x) = m^x modn = f(x+r) \quad (r = 0, 1, ..., 2^l - 1; m$ is an integer smaller than $n-1$). This period can be found by one of the steps of Shor's factorization algorithm, as referred in Section.2.2. QKD is composed by two main steps, quantum communication succeeded by classical post-processing. The security of systems based on it is not on complex mathematical problems or very secure private channels, but instead on the nature of quantum mechanics. Characteristics such as the described by the non-cloning theorem or Heisenberg's uncertainty principle, are what makes a QKD system unconditionally secure. Heisenberg, who was initially referring to the position and momentum of a particle, described how any conceivable measurement of a particle's position would disturb its conjugate property, the momentum. It is therefore impossible to simultaneously know both properties with certainty. Both of these concepts are directly correlated linear superposition and quantum parallelism discussed in Section.2.2. Non-orthogonal quantum states cannot be cloned.

Different degrees of freedom of the photons can be used to codify the QUBITS used to employ QKD protocols, those can be polarization, time, frequency, phase and orbital angular momentum.

2.3.1. Codification schemes

There are two main general schemes:

1. Continuous variable CV-QKD

2. Discrete variable DV-QKD

The two can be distinguished by the strategy applied on Bob's (receptor) side. In DV-QKD schemes, a single-photon detector (SPD) is applied on Bob's side, while in CV-QKD the field quadratures are measured with the help of homodyne/heterodyne detection.

In DV-QKD the non-cloning theorem is applied as well as the theorem of indistinguishability of arbitrary quantum states. Eve cannot duplicate non-orthogonal quantum states, and orthogonal states cannot be distinguished without introducing some ambiguity. When Eve interacts with the transmitted quantum states, trying to get information on transmitted bits, she will inadvertently disturb the fidelity of the quantum states that will be detected by Bob.

In CV-QKD the uncertainty principle claims that both in-phase and quadrature components of coherent states cannot be simultaneously measured with the complete precision [1] [11].

In the second method different degrees of freedom of light particles (photons) are used to encode information and are then measured using single-photon detectors. Since my work is mainly focused on the (DV)-QKD method, a look over the history of the protocols utilized is presented. We can also classify different QKD schemes as either entanglement-assisted or prepare- and-measure types. We will mostly focus on the second type. By the end of this chapter some real results and advantages of some implementations are presented.

2.3.2. System Types

- **Device-dependent QKD**: The quantum source is placed on Alice side and quantum detector at Bob's side. Popular classes include discrete variable DV-QKD,CV-QKD , entanglement-assisted (EA-QKD), distributed phase reference [1](pp 215).
- **Source-device-independent QKD**: The quantum source is placed at Charlie's (Eve's) side, while the quantum detectors at both Alice and Bob's sides [1](pp 215).
- **Measurement-device-independent QKD(MDI-QKD)**: The quantum detectors are placed at Charlie's (Eve's) side, while the quantum sources are placed at both Alice and Bob's sides [1](pp 215).

The structure of a system including Charlie, like MDI-QKD, is presented in Fig.2.1 [24].

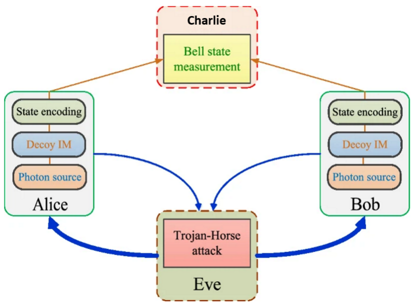


Figure 2.1: MDI-QKD structure [24].

For the purpose of this dissertation Device-dependent QKD is the only one looking to be implemented.

2.4. Introduction to electromagnetic waves and polarization

Since we will be working with light polarization it's important to understand how it can be modeled mathematically, so we can calculate the effects that certain optical devices might have on the SOP. Light can be modeled as an electromagnetic wave. It oscillates perpendicular (transverse) to the propagation direction of the light beam. Such a transverse electromagnetic wave can be divided into unpolarized and polarized, natural light is generally unpolarized, all planes of propagation being equally probable. All field components of polarized light have a fixed phase difference to each other.

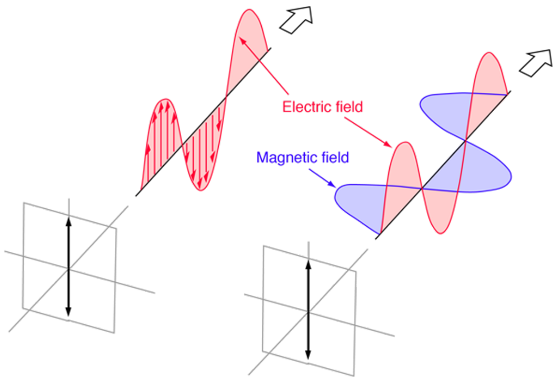


Figure 2.2: Representation of the electromagnetic field for linear polarization [54].

A plane electromagnetic wave is said to be linearly polarized Fig.2.2.The orientation of a linearly polarized electromagnetic wave is defined by the direction of the electric field vector.

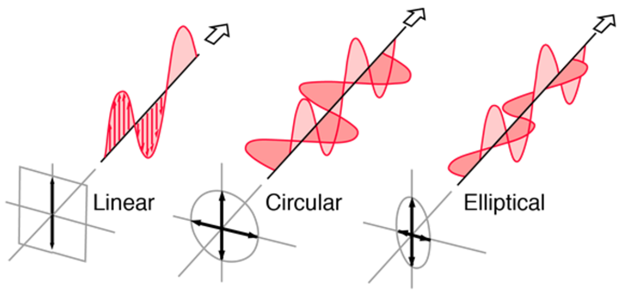


Figure 2.3: Representation of multiple polarization's [54].

If light is composed of two plane waves of equal amplitude but differing in phase by 90°, then the light is said to be circularly polarized, Fig 2.3. If you could see the tip of the electric field vector, it would appear to be moving in a circle as it approached you. The electric field vector makes one complete revolution as the light advances one wavelength toward you. Circularly polarized light may be produced by passing linearly polarized light through a quarter-wave plate at an angle of 45° to the optic axis of the plate.

In a elliptical polarization state, Fig 2.3, both orthogonal waves have a fixed phase between 0 and 90°, the waves could have different amplitudes. The projection results in a ellipse with a right or left direction of rotation. Helium Neon lasers or DFB laser diodes normally feature linear polarization.

A monochromatic plane wave of frequency w traveling in the z direction has its polarization vector in the x-y plane. Any location of the polarization vector at different times t can be written by superposition of the x- and y- vector components:

$$\vec{E} = E_{0x}e^{i(kz-wt+\varphi_x)}\hat{\imath} + E_{0y}e^{i(kz-wt+\varphi_y)}\hat{\jmath} = \vec{E}_x + \vec{E}_y \tag{2.1}$$

E_{0x}, E_{0y}-amplitude of the electric field intensity in x- or y-direction
φ_x, φ_y-phase of the electric field intensity in x- or y-direction
w-frequency
k-propagation vector

2.4.1. Mathematical representation of polarization

Another convenient way to represent light is using the Jones vector formalism. It is an analytical description of the state of polarization.

$$\vec{E} = \left[\hat{\imath}E_{0x}e^{i\varphi_x} + \hat{\jmath}E_{0y}e^{i\varphi_y}\right]e^{i(kz-wt)} = \tilde{E}_0 e^{i(kz-wt)} \tag{2.2}$$

The bracketed quantity, separated into x- and y- components, is now recognized as the complex amplitude $\tilde{E}_0$ for the polarized wave. Since the state of polarization of the light is completely determined by the relative amplitude and phases of these components, we need to concentrate only on the complex amplitude, written as a two element matrix, or Jones Vector:

$$\tilde{E}_0 = \begin{bmatrix}\tilde{E}_{0x}\\ \tilde{E}_{0y}\end{bmatrix} = \begin{bmatrix}E_{0x}e^{i\varphi_x}\\ E_{0y}e^{i\varphi_y}\end{bmatrix} \tag{2.3}$$

Note that only phase differences have physical meaning. An adequate choice of time zero can force $\varphi = 0$ and φ_y becomes the phase difference, φ, between the two components . In addition, if the Jones vector is normalized (by dividing it by E_0) it becomes formally identical to a QUBIT: it is a normalized vector belonging to a two-dimension complex space where a Pauli algebra is defined. Therefore a normalized Jones Vector determined by two parameters (α and β in Figure 2.5) is a QUBIT. Light polarization can also be graphically represented in something called the polarization ellipse Fig.2.4

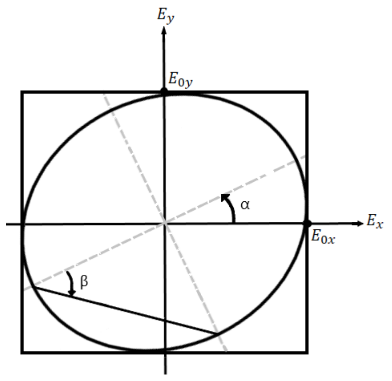


Figure 2.4: Polarization ellipse.

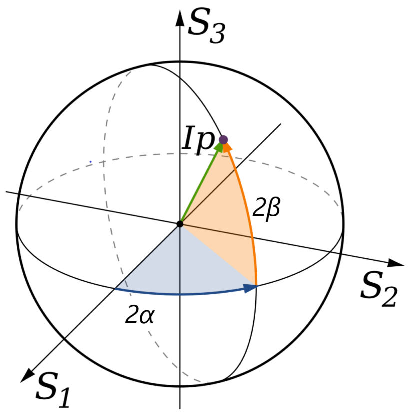


Figure 2.5: Poincaré's sphere with parameters α and β.

The main parameter that defines this ellipse is α and it can be calculated from the electromagnetic wave parameters as follows:

$$\tan(2\alpha) = \frac{2 \cdot E_{0x} \cdot E_{0y} \cdot \cos(\varphi_y - \varphi_x)}{E_{0x}^2 - E_{0y}^2}, \qquad 0 \leq \alpha \leq \pi \tag{2.4}$$

$$\sin(2\beta) = \frac{2 \cdot E_{0x} \cdot E_{0y} \cdot \sin(\varphi_y - \varphi_x)}{E_{0x}^2 + E_{0y}^2}, \qquad -\frac{\pi}{4} \leq \beta \leq -\frac{\pi}{4} \tag{2.5}$$

Jones calculus can be used only when the light is coherent and fully polarized. Light which is randomly polarized, partially polarized, or incoherent must be treated using Mueller calculus. The big difference between this two methods is that Jones calculus works directly with the electric field of the light rather than with its intensity or power, and thereby retains information about the phase of the waves.

In Mueller calculus the SOP can be represented by the Stokes vector ($\vec{S}$) and any optical element represented by a Muller matrix M. The Stokes vector includes 4 parameters, S_0, S_1, S_2 and S_3 that can be directly correlated to a representation in a polarization ellipse Fig.2.4 and the Poincaré sphere Fig.2.5 as follows:

$$S_0 = I \tag{2.6}$$

$$S_1 = I \cdot p \cdot \cos(2\alpha) \cdot \cos(2\beta) \tag{2.7}$$

$$S_2 = I \cdot p \cdot \sin(2\alpha) \cdot \cos(2\beta) \tag{2.8}$$

$$S_3 = I \cdot p \cdot \sin(2\beta) \tag{2.9}$$

I is the total intensity of the beam, and p is the degree of polarization, constrained by $0 \leq p \leq 1$. To obtain the values of the Stokes parameters we can do the following measurements:

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} P_H + P_V \\ P_H - P_V \\ P_{45} - P_{135} \\ P_L - P_R \end{bmatrix} \tag{2.10}$$

P_H, P_V, P$_{45}$, P_{135}, P_R, P_L are the measurements to the intensity of light, done with a ideal polarizers to the horizontal, vertical, 45°, 135°, right-hand circular and left-hand circular polarization directions respectively. The measurements to P_R and P_L cannot be done straight forward and usually require a quarter-wave plate, or a similar configuration, that converts circular polarized light to linear, it can then be passed trough the polarizers. If a beam of light is initially in the state $\vec{S_{in}}$ and then passes through an optical element M and comes out in a state $\vec{S_{out}}$ then it is written:

$$\vec{S_{out}} = \begin{bmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{bmatrix} = M\vec{S_{in}} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{2.11}$$

The optical components we will mostly use in this dissertation behave as general linear retarders that can be described by the following matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos^2(\alpha) + \sin^2(\alpha)\cos(\delta) & \cos(\alpha)\sin(\alpha)(1-\cos(\delta)) & \sin(\alpha)\sin(\delta) \\ 0 & \cos(\alpha)\sin(\alpha)(1-\cos(\delta)) & \cos^2(\alpha)\cos(\delta) + \sin^2(\alpha) & -\cos(\alpha)\sin(\delta) \\ 0 & -\sin(\alpha)\sin(\delta) & \cos(\alpha)\sin(\delta) & \cos(\delta) \end{bmatrix} \tag{2.12}$$

where δ is the phase difference between the fast and slow axis and $\alpha/2$ is the orientation angle of the fast axis.

One of the big differences between this two methods, Jones and Mueller, is the fact that Mueller calculus only works with the intensity or power of the light while Jones calculus works directly with the electric field. Muller calculus is often the most adequate to describe devices that measure power, like polarimeters, radiometers or spectrophotometers. Since describing the SOP as a Stokes vector is common in QKD while also representing them in the Poincaré sphere, this is structure chosen trough the dissertation.

2.5. Polarization encoding

The State of Polarization (SOP) of the photon can be represented/defined using Stockes Vectors. These vectors are characterized by 4 main parameters, S_0, S_1, S_2, S_3. The Poincaré sphere provides a visual method for representing polarization states and calculating the effects of polarizing components. For representation in the Poincaré sphere, Fig.2.6, S_1,S_2 and S_3 are the most relevant since they represent the values in the x,y and z axis respectively. S_0 is related to the intensity of the beam.

In Fig.2.6 we can see a representation of each of the main polarization states, these are also the states that we will use to apply QKD protocols since each pair ($\nearrow$, $\nwarrow$), ($\uparrow$, $\leftarrow$) and ($\circlearrowleft$, $\circlearrowright$) is orthogonal and constitutes a complete basis in a two-dimension qubit space. As mentioned before, Jones vectors can be written in a form equivalent to qubits and exhibit all the qubit properties, namely the no-cloning theorem that states that any two non-orthogonal states cannot be cloned. Only basis states are orthogonal to each other, but are non-orthogonal to any other state in the Pioncaré sphere.

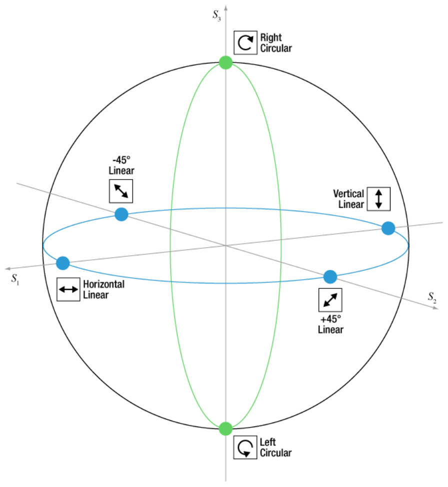
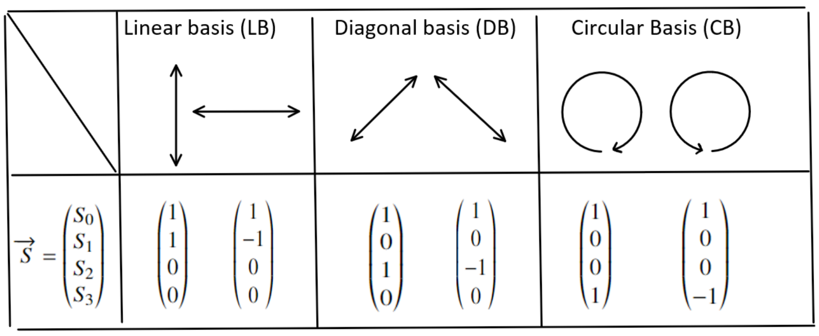


Figure 2.6: Poincaré sphere representation of the polarization states.

For reference, in Section.2.6 I will refer to ($\uparrow$, $\leftarrow$) as the Linear basis (LB), ($\nearrow$, $\nwarrow$) as the diagonal basis (DB) and ($\circlearrowright$, $\circlearrowleft$) as the circular basis. The 3 basis with it's 6 SOP's can be written as Stokes vector's in the following manner:



	Linear basis (LB)		Diagonal basis (DB)		Circular Basis (CB)	
	↕	↔	⤢	⤡	$\circlearrowleft$	$\circlearrowright$
$\vec{S} = \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix}$

To better understand the values presented above and Stokes representation in general see Section.2.4.1. The simplest way to generate the different States of Polarization (SOP's) required to encode the quantum bits would be by using multiple lasers each one representing a different SOP. However, this method suffers from side-channel information leakage. Therefore, the development of efficient methods to generate stable polarization states is now of great importance, ensuring the quality of future polarization QKD systems.

2.6. DV-QKD protocols

Since most of my work is done with photon polarization I will mostly focus on explaining some DV-QKD protocols directly used with photon polarization.

2.6.1. BB84 protocol

Named after Bennett and Brassard, who proposed it in 1984 [1](pp 217-223) [25] [26]. The main structure of this protocol is presented in Fig.2.7 [1](pp 220). The 3 main principles applied in this protocol are the no-cloning theorem, state collapse during measurement, and irreversibility of the measurements [27] . The two bases used in the protocol are the LB and DB base (referred in Section.2.5). The system is prepared in a state belonging to one of the basis, every measurement done with in respect to the other basis as an equally likely result.

In the shifting procedure, Alice and Bob announce the bases being used for each QUBIT and keep only instances when they used the same basis (Section.2.7) Alice selects a base randomly. The logic 0 is represented by $|0\rangle, |+\rangle$ and the logic 1 by $|1\rangle, |-\rangle$. Bob measures every QUBIT selecting randomly the basis at the moment of measurement, LB or DB.

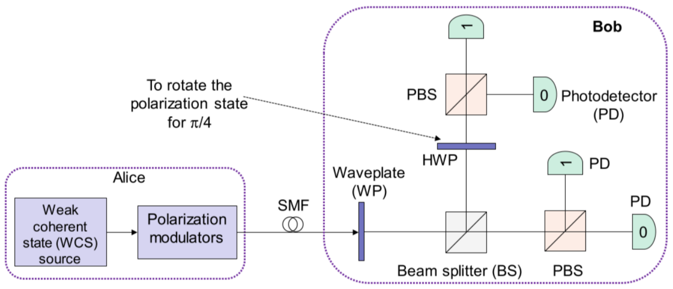


Figure 2.7: Example of polarization based BB84 protocol setup [1].

The output of the laser is known as coherent state. Coherent states are non-orthogonal. When the laser is sufficiently attenuated, the coherent states become weak coherent states, in which we can guarantee we don't have more than a photon with high probability, that probability can be modeled by a Poisson distribution (this a very relevant feature of the implementation as we will see when we take a look at security issues of the implementations).

The BB84 protocol can be applied using different degrees of freedom (DOF) including polarization and photon phase. Experimentally, this protocol as already been demonstrated using fiber optics and free space optics (FSO). BB84 based in polarization trough fiber optics is affected by Polarization Mode Dispersion (PDM), Polarization Dependent Loss (PDL) and fiber losses. In a FSO channel, the polarization effects are minimized. However, the atmospheric turbulence can introduce the wavefront distortion and random phase fluctuations. It can also be applied with phase codification and time-phase encoding. In the second method the temporal base is the LB and the time phase is the DB of the polarization encoding.

2.6.2. B92 protocol

Introduced by Bennet in 1992 [28], it uses only 2 non-orthogonal states, the presented in Fig.2.8 [1](pp 223).

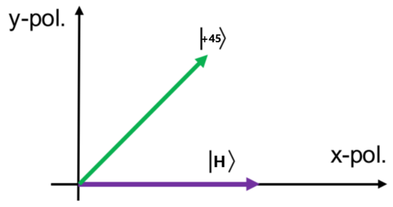


Figure 2.8: Graphical representation of the two SOP's used in the B92 protocol [1].

Alice generates a classic bit $\underline{d}$ and according to its values 0 or 1 it sends the following states to Bob:

$$|\psi\rangle = \begin{cases} |H\rangle & \text{if } d = 0 \\ |+45\rangle & \text{if } d = 1 \end{cases} \tag{2.13}$$

Bob generates a random classical bit d' and measures the received bits in the basis LB if $d' = 0$ and DB if $d' = 1$. With the results of the measurement being $r = 0$ and $r = 1$ corresponding to the eigenstates -1 and +1 of the observable [1](pp 223).Bob publicly announces r and maintains d' secret. Bob and Alice maintain the pairs of {d,d'} to which the measurement result was $r = 1$. The final bit of the key is d for Alice and $1 - d'$ for Bob.

2.6.3. 6 states protocol

The six-state protocol was introduced by Pasquinucci and Nicolas Gisin in 1999 [30]. Instead of using four states, a six state one can better respect the symmetry for a better key generation rate and tolerance to noise. This protocol while very similar to BB84 decreases the possibility of Bob and Alice using the same basis but makes it easier to identify the presence of Eve.

2.6.4. SAGR04

In 2004, Scarani, Acin, Ribordy and Gisin, proposed a new variant of the BB84 at the classical communication channel stage. As we will see further in the document, Section.2.8, Photon Number Splitting (PNS) attacks are a big concern while using non-perfect single photon sources, like attenuated lasers [18].

The protocol is very similar to BB84 as one would expect but when Alice and Bob determine which bits their basis matched, Alice does not directly announce her basis, but instead a pair of non-orthogonal states, one of which being used to encode her bit. Since the two states are non-orthogonal, the PNS attack cannot provide Eve with perfect information on the encoded bit.

2.6.5. Decoy-State-Based Protocol

In decoy-state-based protocols, Alice randomly changes the nature of the quantum signal, such as the intensity of the laser. At the end of transmission process, she reveals which intensities she used so that Eve cannot adapt her attack on the fly. In the post-processing stage, Alice and Bob use this information for parameter estimation. Alice and Bob can maximize the secret fraction by optimizing over both signal levels and probability of occurrence of each level. It allows Alice and Bob to detect if Eve is stealing photons when multiple photons are transmitted [1](pp 269-272) [31].

2.6.6. Others

Ekert (E91) and EPR Protocols, are based in a system where Alice and Bob share n entangled pairs of QUBIT's in the Bell state (EPR pair) [22] [52] [32]. A more complete list of recently developed protocols can be found in [1](ch.9).

2.7. Information Reconciliation and Privacy Amplification

The raw key is imperfect, and we need to perform information reconciliation and amplification of privacy to increase the correlation between the sequence generated by Alice (X) and the ones received by Bob (Y), while reducing eavesdropper Eve's mutual information about the result to a desired level of security [15].

Information reconciliation is nothing more than the error correction performed over a public channel, which reconciles errors between X and Y to obtain a shared bit string K while divulging the information as little as possible to the Eve. They also employ error correction to correct the error introduced by both quantum channel and Eve, and the corresponding key after this stage is completed is commonly referred to as the corrected key [1](pp 216).

Privacy amplification, is used between Alice and Bob to distill from the key a smaller set of bits whose correlation with Eve's string is below a desired threshold. One way to accomplish privacy amplification is through the use of the universal hash functions [1](pp 216).

2.8. Security Issues of QKD Systems

A maximum limit is established for noise/eavesdropping in the quantum communications channel. This limit is dictated by the efficiency of Information reconciliation and Privacy amplification explored previously.

2.8.1. Independent (Individual) or Incoherent Attacks

It's a more constrained family of attacks in which Eve attacks each QUBIT independently, and interacts with each one using the same strategy, The measurements are done after the classic post-processing takes place. The upper,QBER, limit for Incoherent attacks is the same as that for classical Physical Layer Security.

A very important family of Incoherent attacks is the intercept-resend (IR) attack, in which Eve intercepts the quantum signal sent by Alice, and measures it, based on that measurement she prepares a new quantum signal, in the same base that she just measured and sends it to Bob.

In BB84 for example 50% of the basis choices made by Eve are in accordance to Alice's. When Eve uses the wrong base there's 50% of obtaining the correct bit. Same thing goes for Bob, there is a 50% change that he uses the same base as Eve. However, these bits will be correlated to Eve's sequence and not Alice.

Taking all of this into account the total probability of error is 1/4. Given this high fraction Eve's activity is easily identifiable. However, Eve can choose to only apply this attack with probability p_{ir}, in that case the error probability will be $q = p_{ir}/4$.

Another important family of incoherent attacks are Photon Number Splitting (PNS). This kind of attack makes use of one of the weaknesses of the QUBIT generation, non-single photon generators. As referred in Section.2.6.1 the probability of emitting n-photons in a state generated with a mean photon number μ is determined by a Poisson distribution described by the following equation:

$$p_{A,n} = p(n|\mu) = e^{-\mu}\frac{\mu^n}{n!} \tag{2.14}$$

To perform the PNS attack, Eve can employ the beam splitter to take one of the photons from the multi-photon state, and pass the rest to Bob. She can measure the photon by randomly selecting the basis. Eve can even replace the quantum link with ultralow-loss fiber so that Alice and Bob cannot figure that the transmitted signal gets attenuated. To solve the PNS attack, Alice and Bob can employ decoy-state-based protocol (as referred in Section.2.6.5), in which Alice transmits the quantum states with different mean photon numbers, representing signal and decoy states. Eve cannot distinguish between decoy state and signal state, and given that Alice and Bob know the decoy signal level they can identify the PNS attack [1].

The current implementation in the quantum optics lab at Instituto de Telecomunicações of Aveiro expects and average of 1 photon at every 10 photon laser pulses. This makes PNS attacks very hard to execute by Eve, at the cost of lower key rates.

2.8.2. Collective Attacks

The collective attacks represent generalization of the incoherent attacks given that Eve's interaction with each QUBIT is also independent and identically distributed. However, in this attacks Eve stores the QUBIT in quantum memory until the end of the classical post-processing. Given that reconciliation requires the exchange of the parity bits over an authenticated classical channel, Eve can apply the best known classical attacks to learn the content of the parity bits. Based on all information available to her, Eve can perform the best measurement strategy on her ancilla qubits (stored in the quantum memory) [1].

2.8.3. Coherent attacks

The coherent attacks represent the most general and most flexible strategies that Eve can apply on quantum states. She can adapt the eavesdropping strategy on the fly, based on previous and current measurements. She might further entangle as many quantum states as she wants and stores them in the quantum memory. Therefore, the minimization of mutual information between Alice/Bob and Eve is impossible. Nevertheless, the bounds have been determined in many cases, and these are very similar to those obtained for the collective attacks [1].

2.8.4. Quantum Hacking Attacks and/or Side-Channel Attacks

Some Quantum Hacking attacks could be:

- Trojan horse attack - Eve sends the bright laser beam toward Alice's encoder and measures the reflected photons to gain the information about the secret base. This attack can be avoided by using an optical isolator [33].
- It was also noticed that silicon-based photon counters employing the avalanche photodiodes (APDs) emit some light at different wavelengths when they detect a single photon, which can be exploited by Eve [34].
- Time-shifting attack - Eve exploits the efficiency mismatch of Bob's single photon detectors to estimate Bob's basis selection [35].
- Blinding attack - Eve's exploits the APDs-based single-photon counter properties to force Bob to pick up the same basis as Eve does [36].

2.9. Polarization encoding advantages and disadvantages

Polarization encoding presents advantages over the other mentioned encoding methods, namely for free-space optics applications since the atmosphere, unlike telecommunication optical fibers, keeps the polarization stable. When we consider an optical fiber as the quantum channel, maintaining a constant State of Polarization (SOP) of an optical signal has been the main issue for this encoding method. The intrinsic residual birefringence of fiber optics as well as extrinsic mechanisms, in a big part unpredictable, such as temperature and other environmental conditions need to be properly compensated if one expects to achieve low QBER values [4]. Furthermore, DV-QKD systems suffer from dead time related to the time that the SPD remains unresponsive to the arrival of photons due to its recovery time (typically between $10 - 100$ ns), this limits the baud rate and the key rate by relation. CV-QKD systems does not suffer from this issue since they use homodyne/heretodyne detection, no dead time, but their maximum communication distances are usually inferior [1](pp 215). Their compatibility with classical detection hardware also poses a major advantage against current single photon avalanche based detection schemes required for the DV-QKD, which limits on the achievable performance and work at very-low temperatures demanding additional cooling systems [13].

Nevertheless, polarization encoding is simpler to implement when compared with other aforementioned methods. The focus is now on avoiding loopholes, increasing the rate of quantum key trading (GHz), and trying to achieve communication at longer distances (100-300·km). More straightforward implementations in the existing infrastructure for communications, is also a main objective.

Chapter 3

Optimization of SOP transitions

In this chapter the method used to optimize the voltages used to achieve the 6 SOP's used in QKD is explained.

3.1. Electrical Polarization Controller

The Electrical Polarization Controller (EPC) are optical devices that allow changing an arbitrary input SOP of an optical beam into any desired output SOP. They can be controlled using electrical tensions.

The EPC we were looking to optimize is based in Electro-optic crystals, this means it can make use of the Pockels effect. Pockels effect is the change in birefringence of an optical medium induced by an electric field.

Only in certain crystals that lack inversion symmetry, such as lithium niobate, can this effect occur. Lithium niobate is the crystal used in the EPC we are using. The EPC's in our possession come with 6 or 8 stages. Each stage is a section of the $LiNbO_3$ substrate that can be used as a waveplate to modify the SOP.

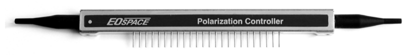


Figure 3.1: Picture of the EPC.

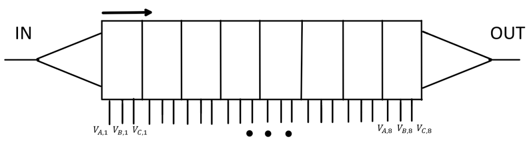


Figure 3.2: EPC schematic.

The EPC in our possession is the one showed in Fig.3.1, the 8 and 6 stages configurations look the same with the only difference being the number of active pins. In Fig.3.2 a simplified schematic of the EPC is presented. Each of the blocks can be considered a stage, each stage has 3 pins to which we apply the voltages to modify the birefrigence of the crystal.

The waveplate can be actuated according to the following equations:

$$V_A = 2 \cdot V_0 \cdot \Delta \cdot \sin(\alpha) - V_\pi \cdot \Delta \cdot \cos(\alpha) + V_{A,bias} \tag{3.1}$$

$$V_B = 0 \quad (Ground) \tag{3.2}$$

$$V_C = 2 \cdot V_0 \cdot \Delta \cdot \sin(\alpha) + V_\pi \cdot \Delta \cdot \cos(\alpha) + V_{C,bias} \tag{3.3}$$

V_A, V_B and V_C are the voltages we need to apply to a certain stage to have a waveplate of Δ retardation and with $\alpha/2$ the orientation angle of the fast axis. V_{pi},V_0 and $V_{A,bias}$,$V_{B,bias}$ are known constants, found during calibration. The variable Δ from equations.3.1 and 3.3 is distinct from δ that characterizes the retardation effect of the Muller Matrix for a general linear retarder in equation.2.12. If we were looking to get a quarter-waveplate at a certain stage, Δ would be $\frac{1}{4}$ and $\delta = \frac{\pi}{2}$, so $\delta = 2\pi \cdot \Delta$. The variable α from equations.3.1, 3.3 and 2.12 represent the same parameter.

3.1.1. Working principles

Has discussed in Section.2.4.1 the effect on the SOP, represented as a Stokes vector, by an optical device, like a waveplate, is done in Mueller calculus by matrix multiplication, equation.2.11. If we want to predict the effect of 6/8 waveplates on the input SOP of light and obtain the combined Muller matrix we need to multiply the 6/8 Muller matrix's , equation.2.12, from the last stage matrix to the first one. The effect on the SOP for a 8 stages EPC can be calculated in the following manner:

$$\begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix}_{OUT} = M_8 M_7 \cdots M_2 M_1 \cdot \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix}_{IN} \tag{3.4}$$

Each stage has 2 variables α and Δ, this means that for a 8 stages EPC we would be working with 18 variables simultaneously, 12 for 6 stages. Once we find the values of α and Δ required for a certain SOP modification we get the voltages necessary for each stage trough equations.3.1,3.2 and3.3. There are multiple combinations of α and Δ for each stage that will result in the same output SOP as we can see. From those combinations we can select those that better fit out purpose.

3.1.2. Voltage characterization

The voltage distributions for a range of $0 \leq \Delta \leq 1$ and $0 \leq \alpha \leq 2\pi$, can be graphed for each stage. The distribuition for each stage should be slightly different because their constant parameters V_{pi}, V_0, $V_{A,bias}$ and $V_{B,bias}$ differ between stages. Stage 1 of the EPC has the following values for it's constant variables according to the datasheet, $V_{pi} = 56.3$, $V_0 = 27.2$, $V_{A,bias} = -10.7$ and $V_{B,bias} = 8.4$. Using this values we can draw the voltage distribution that we expect to see for V_A Fig.3.3 and V_C Fig.3.4.

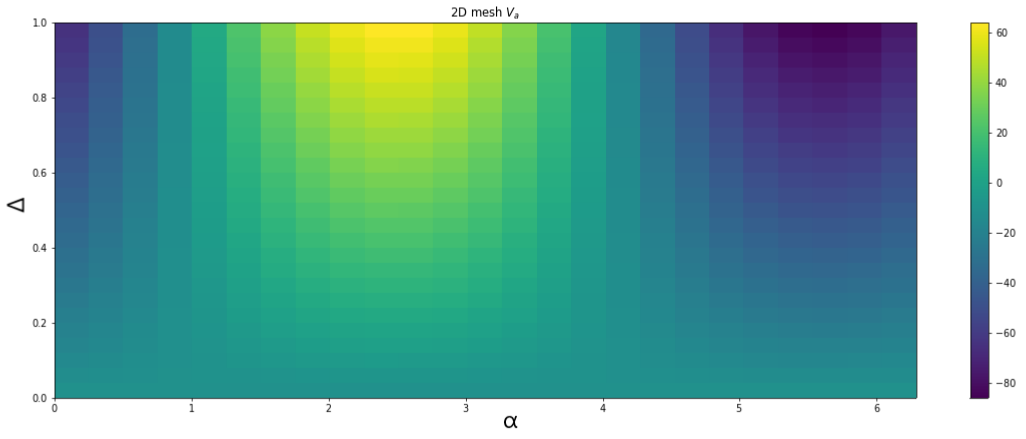


Figure 3.3: Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis.

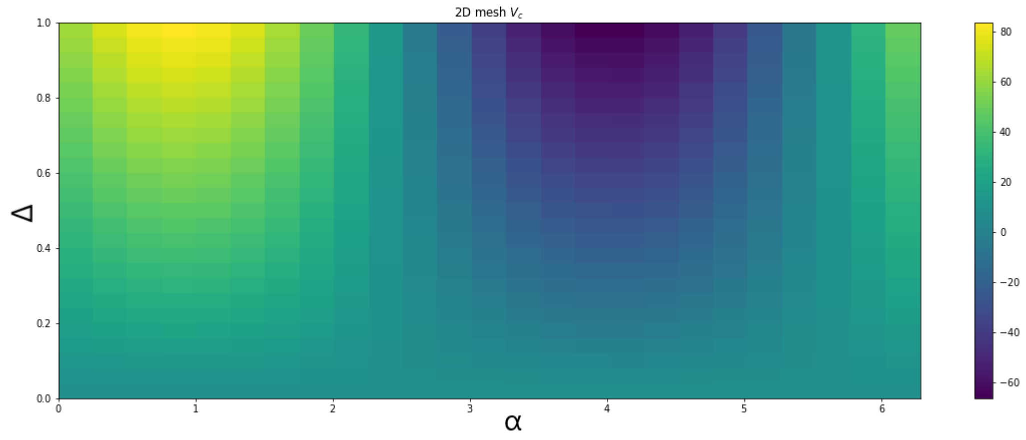


Figure 3.4: Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis.

Variation in the values of $V_{A,bias}$ and $V_{B,bias}$ for a stage don't affect the pattern we see in Fig.3.3 and 3.4, it simply changes all values by the variation amount. On the other hand variations to V_{pi} and V_0 do change the pattern, according to equations.3.1 and 3.3.

3.2. Applying Machine learning

We now know how to connect the voltage equations for each stage, equations.3.1,3.2 and 3.3, with the effect that each stage will have in the input SOP, equations.2.12 and 3.4. Since there could be countless combinations of α's and Δ's that convert a certain input SOP to a certain output SOP we should select from those the ones that allow for transitions in the smallest range/intervals for the 6 SOP's . We want to find a minimum value, optimize a certain function.

3.2.1. The Cost function

If we want to optimize any situation we first need to define it as a function so we can have the ability to compare distinct combinations and make a decision. The cost function we define needs to take in account the output SOP and compare it to the desired SOP while also calculating the interval of voltages required to get that output SOP. The following cost function was designed with a mixture of absolute error and squared error:

$$C = \sum_{i=1}^{3} a \cdot \mid S_i - L_i \mid + \sum_{i=1}^{n} \left(\frac{V_{A,i} - V_{AIN,i}}{b} \right)^2 + \sum_{i=1}^{n} \left(\frac{V_{C,i} - V_{CIN,i}}{c} \right)^2 \tag{3.5}$$

The first sum represents the difference in the values of the Stocks parameters for the current selected combination of $\alpha's$ and $\Delta's$ and the desired Stocks parameters, a is a weight that can be adjusted. The second and third sum take in account all the differences between the voltages for the current combination of $\alpha's$ and $\Delta's$ and the voltages of another SOP selected by the user. We chose to square the difference between the voltages because the largest value from this calculation will be the limiting factor between transitioning SOP's, ideally we would like to have all transitions with similar voltage intervals for the n stages. n is the number of stages we want to utilize/have available. b and c are weights that can be used if needed. This function only takes in account another SOP to compare the differences in voltage values. Ideally we would like to take in account all the SOP voltage values we have found up to that point. What I am saying is, if we calculate the voltages required for the Horizontal and Vertical SOP we could want to use them both when finding an optimal value for the +45 SOP.

$$C = \sum_{i=1}^{3} a \cdot \mid S_i - L_i \mid + \sum_{j=1}^{m} \sum_{i=1}^{n} \left(\frac{V_{A,i} - V_{AIN,i,j}}{b \cdot m} \right)^2 + \sum_{j=1}^{m} \sum_{i=1}^{n} \left(\frac{V_{C,i} - V_{CIN,i,j}}{c \cdot m} \right)^2 \tag{3.6}$$

We add 2 new sums to the cost function and a new variable, m. m is the number of SOP's we want to compare current calculation to. So in the previous described situation, equation.3.5, m would equal 1.

Choosing the values for a, b, c and m, takes some time and analysis and will certainly change according to the chosen algorithm. If we want to modify the algorithm to find more exact SOP when compared to the SOP we desire, we increase a and/or increase b, c, if instead we want to find smaller voltage intervals at the cost of slightly less exact SOP we can decrease a and/or decrease b, c. The variable m is also one we need to take in account because it might not be worth to compare the SOP we are calculating to every other SOP we have calculated up until then, we could be looking at 16 extra sums for each SOP in a worst case scenario of 8 stages.

3.3. Choosing the algorithm

The problem we want to solve is not a classification problem, it can be more easily be compared to a regression problem. Since the SOP's are calculated trough the use of Mueller matrices, a multiple matrix multiplications the choice of the algorithm is very important, because some algorithms like gradient descent might need to do the partial derivatives of multiple variables, 18 in our worst case, after up to 8 matrix calculations, and let's not forget that the variables,α and Δ, are often inside trigonometric entities, equation.2.12. That seems like a lot of calculations when 0.5V or 1V differences from the ideal solution do not matter that much in the bigger scale of QKD. Further more a algorithm like gradient descent might not even offer the better result if the weight parameters in the cost function are not adjusted correctly. Other algorithm's were looked at to apply in this situation, Particle Swarm Optimization (PSO) was one of them as it differed greatly from gradient descent. PSO was found while looking for methods to calibrate the EPC, and it seemed like a great fit in our current problem.

3.4. Particle Swarm Optimization

Particle Swarm Optimization was proposed by Kennedy and Eberhart in 1995 [5]. As mentioned in the original paper, sociobiologists believe a school of fish or a flock of birds that moves in a group "can profit from the experience of all other members". While we can simulate the movement of a flock of birds, we can also imagine each bird is trying to help us find the optimal solution in a high-dimensional solution space and the best solution found by the flock is the best solution in the space. This is a heuristic solution because we can never prove that the real global optimal solution can be found and it is usually not it. However, we often find that the solution found by PSO is quite close to the global optimal. PSO is best used to find the maximum or minimum of a function defined on a multidimensional vector space. Each bird in the explanation above is referred to as a particle. Each particle exists in the variable space, and it's score is given by the cost function.

If we had a very basic one dimensional function like $y = x^2$, and we wanted to find the minimum without calculating the derivative, we could just test the function with multiple values in a range from 1 to -1, for example, and consider the minimum to be lowest result of all of them. The result we get from this method might actually still not be the best and the particles could be closer to zero.

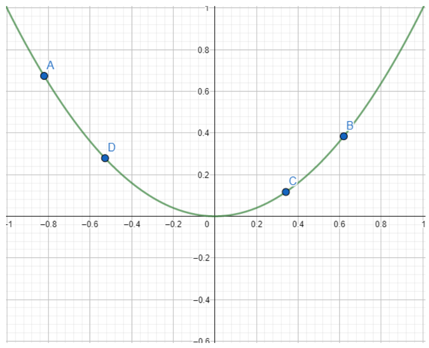


Figure 3.5: PSO example for a function $y = x^2$: iteration 1.

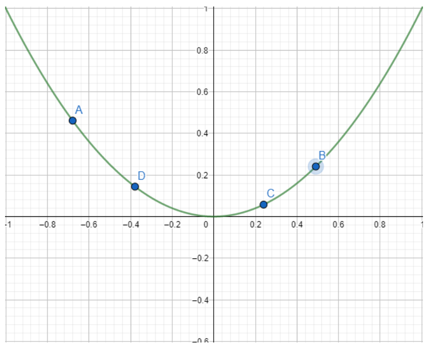


Figure 3.6: PSO example for a function $y = x^2$: iteration 2.

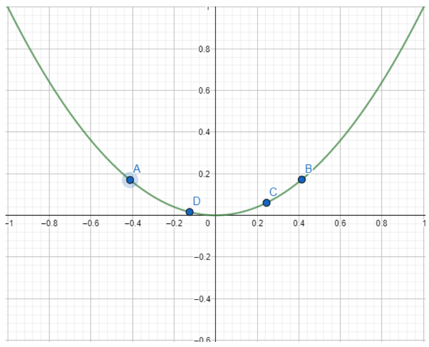


Figure 3.7: PSO example for a function $y = x^2$: iteration 3.

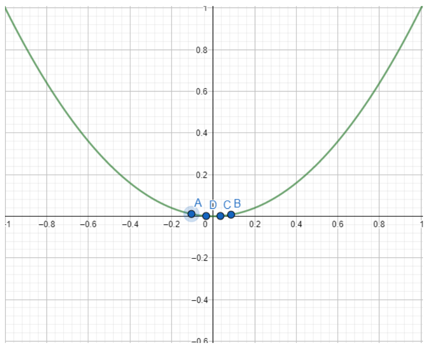


Figure 3.8: PSO example for a function $y = x^2$: iteration 4.

In Figures.3.5,3.6,3.7 and 3.8 we present what could be the effect of 4 iterations for the cost function $y = x^2$ in PSO while using 4 particles, it's a very simple function but the concept remains similar for more complex functions with multiple dimensions.

In PSO, each particle updates it's position after an iteration, and it does so taking in account the best score that it was able to achieve, where that was in the variable space and what was the best score out of all the particles and where that best particle was localized. We can write:

$$X^i(t+1) = X^i(t) + V^i(t+1) \tag{3.7}$$

X^i is a vector of particle i after t iterations of the algorithm, V^i is the velocity of the particle and it denotes the direction and speed at which the particle will modify it's position. The velocity of the particle is modified according to the following equation:

$$V^i(t+1) = w \cdot V^i(t) + c_1 \cdot r_1 \cdot (pbest^i - X^i(t)) + c_2 \cdot r_2 \cdot (gbest - X^i(t)) \tag{3.8}$$

w is a weight that we multiply to the velocity of the particles in the current iteration, $0 \leq w \leq 1$, c_1 and c_2 are constant weights defined by the user, messing with this parameters can influence the exploration that the particles does trough the space. c_1 could be referred as the cognitive coefficient and c_2 as the social coefficient, increasing c_1 causes more exploration by the particles and increasing c_2 causes more exploitation since it will take in account more of what the other particles have achieved. r_1 and r_2 are generated randomly $0 \leq r \leq 1$ with a uniform distribution. $pbest^i$ is a vector with the best position that particle i has achieved until that iteration and $gbest$ is a vector with the position of the best result obtained among all particles. The starting values of V^i are generated following a normal distribution.

3.5. Choosing the start particle space

Has referred in the previous section (Section.3.4) when using PSO, we need to define a starting space for the particles. This means that we need to define a group of values for α and Δ for each each stage of the EPC and for each particle. At max we would need to generate, $\alpha_1, \alpha_2, ..., \alpha_8$ and $\Delta_1, \Delta_2, ..., \Delta_8$ 16 values for each particle. The simplest solution is to generate values randomly, following a uniform distribution, in the entire domain $0 \leq \Delta \leq 1$, $0 \leq \alpha \leq 2\pi$. But as we see in Fig 3.3 and 3.4, that present the voltage distributions for both V_A and V_C in the entire domain we can clearly see areas where the voltage values have less variations.

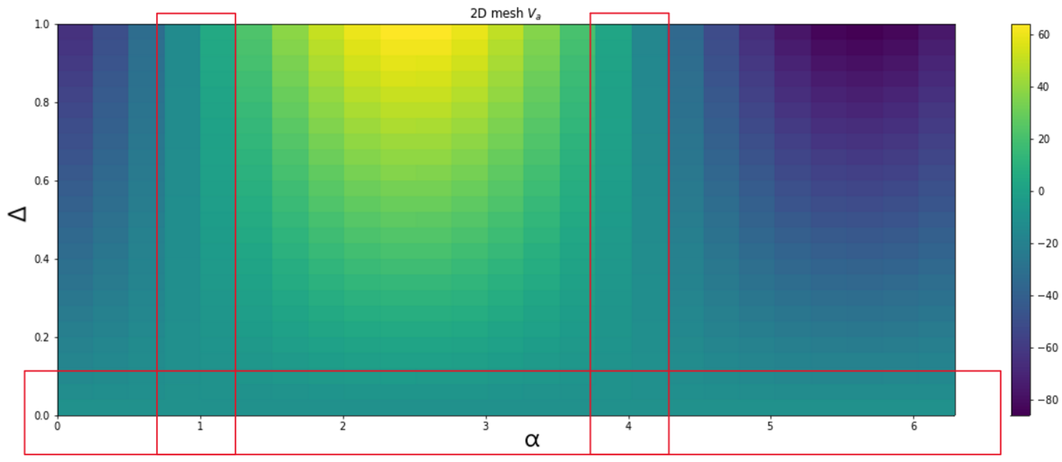


Figure 3.9: Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis, red rectangles exemplify areas where the voltage values are fairly constant.

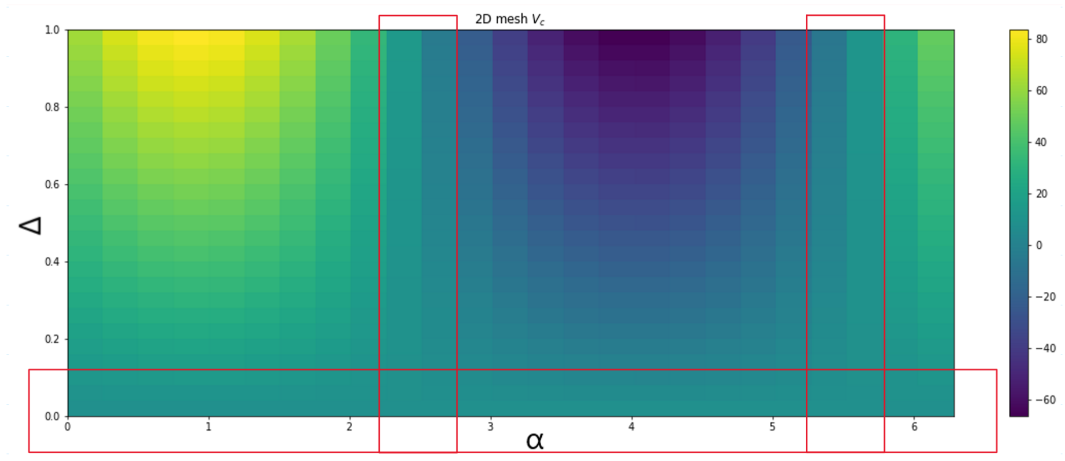


Figure 3.10: Voltage distribution values for V_A for stage 1, α in the horizontal axis and Δ in the vertical axis, red rectangles exemplify areas where the voltage values are fairly constant.

The areas inside the rectangles highlighted in red in the Fig.3.9 and 3.10, represent areas where the voltage values are fairly constant as we can see by the lack of colour variations. The voltage values in this areas are also very close to the values of the constant variables $V_{A,bias}$ and $V_{C,bias}$ in equations.3.1 and 3.3. The idea would be to generate all particles inside those areas. To do that, we calculate the standard deviation of V_A and V_C for multiple α and Δ values in their entire domain. This means that we fix a certain value of δ and then using equations.3.1 and 3.3, we calculate the voltage values for multiple α values in it's entire domain. After that we calculate the standard deviation of the calculated voltage values. The same concept is applied when we test α values. Now that we have an idea of the standard deviation for multiple fixed values of Δ and α, the rows and columns of Figs.3.9 and 3.10, we can select the ones that gave values under the first quantile of calculated values. After defining intervals like the ones in Figs.3.9 and 3.10 we calculate the average of voltage values in those intervals. We will feed this average values to the algorithm, the goal is to get the first SOP calculated the closest possible to them.

3.6. Algorithm Implementation

The implementation of the algorithm was done in python, the only library used was *numpy*. There is nothing that I have done that can't be easily rewritten using another programming language. Furthermore, due the nature of PSO, I do believe it could be implemented in fairly constrained devices like a FPGA. Bellow I present an example of the python code made to calculate the first SOP, Horizontal. We begin the code by defining the constant variables from equations.3.1 and 3.3 that we can find after doing calibration, or in our case obtained through the datasheet. We define the goal SOP and input SOP. After that we provide the algorithm with the number of particles it will use and the number of stages it should take advantage of.

```
V_pi_array=np.array([56.3,56,55.9,56.1,56.1,56,56.1,56])
V_o_array=np.array([27.2,27,27,26.9,26.8,27.2,26.8,27.2])
V_A_bias_array=np.array([-10.7,-9.3,-8.5,-10.9,-7.4,-7.6,-7.4,-7.6])
V_C_bias_array=np.array([8.4,9.6,9.4,11.1,11.6,10.9,11.6,10.9])

''' Important parameters '''
goal_array=np.array([1,0,0])
# input_pol=np.array([ 0.55916189 , 0.12389344 , -0.91504942]) <- Lab SOP
input_pol=np.array([1,0,0])
reference_point=np.sqrt(input_pol[0]**2+input_pol[1]**2+input_pol[2]**2)

n_particles = 80
n_stages=6
```

Thereafter we make the analysis of the voltage distribution and generate the values for the particles. We generate values for all the particles for each stage, each iteration of the while loop is the generation of values for one of the stages. Half of the particles will be created according to the V_a distribution and the other half according to the V_c distribution, since the two voltages don't follow the exact same distributions as we see in Figs.3.9 and 3.10.

After the end of the while loop the arrays are resized to a matrix format. In this matrix format we have the values for each particle in the different columns and each row from this columns has a value for a different stage that was generated according to their voltage distribution.

```
while(i<n_stages):

    start_delta_array_A,start_delta_array_C,end_delta_array_A,end_delta_array_C,start_alpha_array_A
    ,start_alpha_array_C,end_alpha_array_A,end_alpha_array_C,V_A_in,V_C_in=
    interval_selection(V_pi_array[i],V_o_array[i],V_A_bias_array[i],V_C_bias_array[i])

    r_start_delta_array_A.append(start_delta_array_A)
    r_start_delta_array_C.append(start_delta_array_C)
    r_end_delta_array_A.append(end_delta_array_A)
    r_end_delta_array_C.append(end_delta_array_C)
    r_start_alpha_array_A.append(start_alpha_array_A)
    r_start_alpha_array_C.append(start_alpha_array_C)
    r_end_alpha_array_A.append(end_alpha_array_A)
    r_end_alpha_array_C.append(end_alpha_array_C)


    V_A_in_array[i]=V_A_in
    V_C_in_array[i]=V_C_in

    X_delta_add,X_alpha_add=particle_distribution
    (n_particles/2,start_delta_array_A,end_delta_array_A,start_alpha_array_A,end_alpha_array_A)

    X_delta=np.concatenate((X_delta,X_delta_add))
    X_alpha=np.concatenate((X_alpha,X_alpha_add))

    X_delta_add,X_alpha_add=
    particle_distribution(n_particles/2,start_delta_array_C,end_delta_array_C,
    start_alpha_array_C,end_alpha_array_C)

    X_delta=np.concatenate((X_delta,X_delta_add))
    X_alpha=np.concatenate((X_alpha,X_alpha_add))
    i+=1

X_delta=np.reshape(X_delta,(n_stages,n_particles))
X_alpha=np.reshape(X_alpha,(n_stages,n_particles))

X = np.concatenate((X_delta, X_alpha))
V = np.random.randn(n_stages*2, n_particles) * 0.1 #normal distribution
```

We then apply the PSO algorithm. Once the algorithm ends all it's iterations the values obtained for Δ and α can be retrieved as we see bellow.

```
gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,
input_pol,250,2,10,10,0,n_stages)

delta_array=gbest[0:n_stages]
alpha_array=gbest[n_stages:(n_stages*2)]
```

After we have the values of α and Δ we can recalculate the EPC effect on the input SOP. We also redo the measurements of the voltage differences for the calculated SOP compared to a reference point. We are only interested in the maximum differences.

```
output_pol=output_calc(gbest,input_pol,n_stages)

V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time(delta_array,
alpha_array,V_A_in_array,V_C_in_array,n_stages)

V_A_max=np.max(V_A_measurment_abs)
V_C_max=np.max(V_C_measurment_abs)

if(abs(output_pol[0]-reference_point)<1e-4 and V_A_max<1.5 and V_C_max<1.5):
    cond=0
```

The *if* condition in the code above is **very** important, since it serves as way to confirm that the values obtained by the algorithm are inside a certain range we desire. We confirm that the error for the main stokes parameter that defines a certain SOP, like S_1 for the Horizontal state is under a 10^{-4} error, and that the calculated voltage interval is under a certain value. The confirmation that the voltage interval is under a certain value only serves to combat some of the randomness of PSO. PSO has some variability due to it's random nature, if the seed for the random number generator is not fixed, results will vary slightly at each run. Using the *if* condition can also help compensate for situations where the choice of the parameters a, b and c from the cost functions, equations.3.5 and 3.6, was not ideal. The complete code can be found in the following github repository, github.com/Ch0s3n0ne/DV-QKD-Optimizer and in Appendix.A.

3.7. Parameter decisions

In the previous section, Section.3.6, the implementation of the algorithm was presented. In this section I will discuss some of the decisions that were made regarding the values chosen for the variables a, b and c of the cost functions, equations.3.5 and 3.6, and also the conditions by which we might rerun the algorithm for a SOP calculation.

In the current application we have $a = 2$, and $b, c = 10$. The choice of b, c equal to 10 was done knowing that a 10 volts range is a good marking point for drivers available in the market while also taking in account previous runs of the algorithm with other values. The value of a equal to 2 was chosen after analysing the performance of the algorithm for some other values surrounding it.

After the values of α and Δ are obtained by the algorithm for a certain SOP, some final confirmations are done, the main one is related to making sure that the SOP error is under a certain condition, this is done by calculating the difference to the desired value for a Stokes parameter that I consider to be the main one for a certain basis, S_1 for LB, S_2 for DB and S_3 for CB. If the difference between the desired Stokes parameter and the obtained Stokes parameter is under 10^{-4} the result is accepted. For the H SOP, $|S_{1,algorithm} - 1|$ is calculated and for +45 it's $|S_{2,algorithm} - 1|$.

The other condition placed is related to the voltage interval of the current calculation to another SOP, due to the random nature of PSO or imperfect choices for parameters a, b and c, the result obtained by the algorithm might still have some room for improvement, after analysing the results of previous runs a performance bar for the voltage intervals can be placed, the calculations for a certain SOP are rerun if the results are over the performance bar. It's important to understand that this performance bar will need to be changed after certain conditions vary like, the calibration parameters of the EPC, the SOP of the light at the entrance and the number of stages used. Another way to set a performance bar is using the values given by the cost function instead of voltage intervals directly, we could also define a new function that doesn't take in account the calculated SOP precision, this method can be more beneficial when we are calculating a SOP that is being compared to the values of 2 or more SOP's since it's hard to define a expected best result for all simultaneously.

Understanding the way the cost function molds the calculations to get to a certain result is the most important thing, a lot of analysis is required by the user to fine tune it. Beyond that, if a good seed for the random number generator is found it is useful to fix that seed and work on the results given by it, most of the unpredictability associated with PSO is removed.

3.8. Numerical results

The algorithm performance was analysed using the values presented in the datasheet for the calibration constants. The SOP at the input was considered to be a perfect Horizontal SOP, defined by $S_1 = 1$, $S_2 = 0$ and $S_3 = 0$. To implement QKD we will need at least 2 basis of the 3 available ones. To evaluate the performance of a certain basis choice, LB and DB or LB and CB or DB and CB, we will calculate all the voltage intervals between all the SOP's. We select the highest voltage interval for all the basis pairs, this is the limiting factor. It doesn't matter how small most of the SOP transitions between two basis are if a single one of them is much higher than the others, that one will limit the speed at which we will be able to apply QKD.

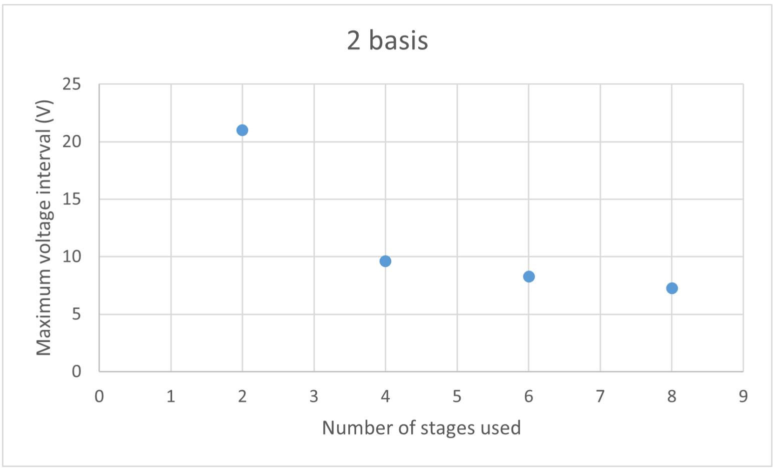


Figure 3.11: Voltage intervals obtained for QKD application using 2 basis, x-axis number of stages used and y-axis maximum voltage interval.

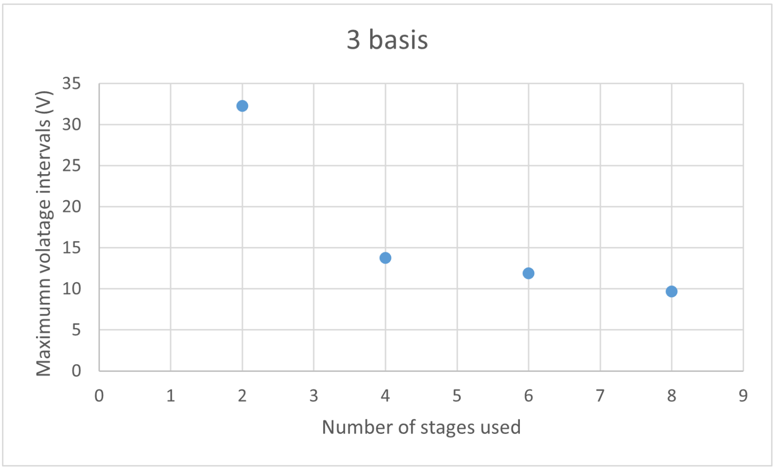


Figure 3.12: Voltage intervals obtained for QKD application using 3 basis, x-axis number of stages used and y-axis maximum voltage interval.

The results we obtain show a clear advantage in voltage intervals when using 2 basis, Fig.3.11, compared to 3 basis, Fig.3.12. The number of stages we use in the EPC will not vary the voltage intervals linearly instead it will follow a negative power trend-line. From the results obtained we expect to be able to apply QKD, using 2 basis, under a 10V range.

Chapter 4

Laboratory Validation

In this chapter what we will be **looking to validate is the ability to achieve all the 6 SOP's in voltages intervals similar to the ones obtained numerically**. It's important to refer again the importance that the calibration parameters, V_0, V_π, $V_{A,bias}$ and $V_{C,bias}$, have on the voltage values that need to be applied to the EPC. We obtain the voltage values that we need to apply to the EPC pins from equations.3.1 and 3.3, this means that if we are not considering the correct values for the calibration parameters we will actually be placing incorrect values of α and Δ in each stage which will result in the incorrect SOP at the output.

A method for calibrating the EPC is not the goal of this dissertation, previous work in this front, described in a dissertation last year [29], was not successful. Some other methods besides the one used in [29], could be used but the tools to apply them have not yet been fully developed. The quickest way to verify if the EPC is not calibrated with the values given by the datasheet is by trying to obtain null birefringence in all stages, the voltage values under those conditions should be equal to V_{bias}. Under those conditions the input SOP should be equal to the output SOP, and that is not verified. One could think that trying to obtain the V_{bias} would be as simple as adjusting the voltage values applied at each pin until the output SOP is similar to the input SOP, but it's very hard to analyse the effects that V_0 and V_π might have on that method and how the stages might compensate each other.

We can try to predict the effect of the calibration parameters V_0 and V_π on the voltage intervals between the SOP's. To do so we tested values withing a range of 10 above and bellow the datasheet values. The results we got showed that, higher values of V_0 and V_π could result in a some SOP transitions having 4.5V higher voltage intervals. Lower values of V_0 and V_π could result in improvements to some transitions of up to 6.3 V. These values are for the α and Δ combinations present in Table.4.1. The values of both V_{bias} wont affect the voltage intervals, they don't affect the voltage distribution patterns like we previously discussed in Section.3.1.2. Nevertheless, whatever the situation might be we should always be able to achieve at least 2 basis under a 10 V range, and **a method to do so even under this restrictions is presented**.

4.1. Laboratory Implementation

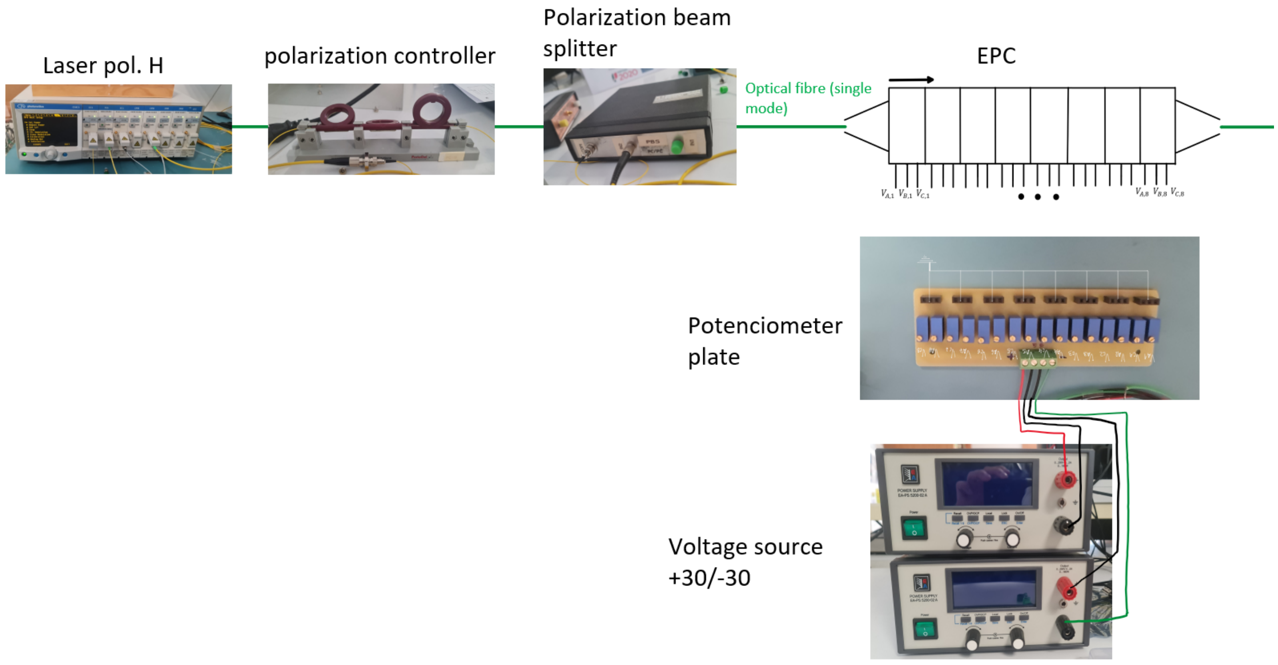

Laser pol. H
polarization controller
Polarization beam splitter
Optical fibre (single mode)
EPC
$V_{A,1}$ $V_{B,1}$ $V_{C,1}$
$V_{A,8}$ $V_{B,8}$ $V_{C,8}$
Potenciometer plate
Voltage source +30/-30

Figure 4.1: Laboratory setup (part one).

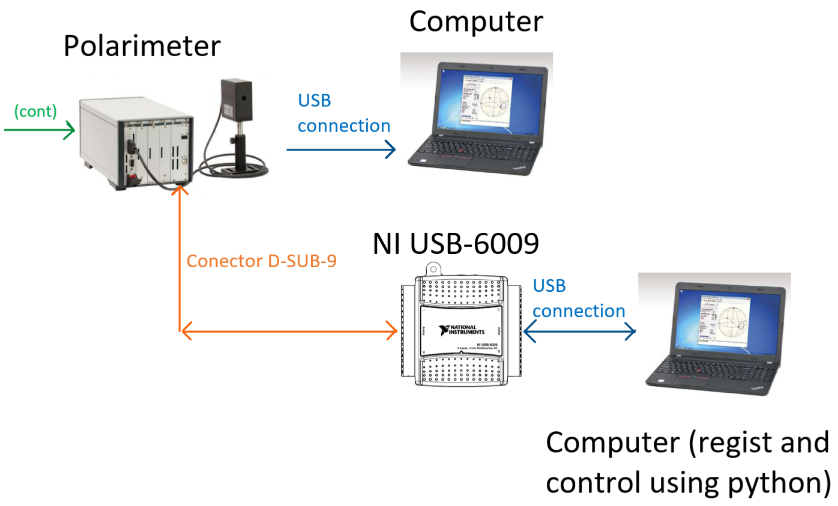


Figure 4.2: Laboratory setup (part two).

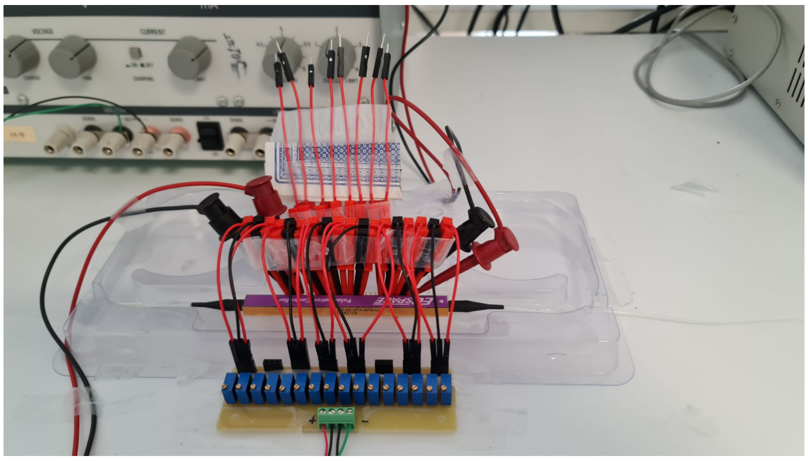


Figure 4.3: Close-up of EPC setup.

In Figs 4.1 and 4.2 we have the complete lab setup used to validate the numerical results. We will now take a closer look at the equipment's involved in our application and their purpose.

Laser

The laser module used is from Photonetics OSICS and it's a High Performance Distributed Feedback laser Diode (DFB) module. Some of the main features of this equipment are it's adjustable Wavelength from 1527.2 to 1610.05 nm, adjustable Power and internal polarization maintaining fiber. It has under 30MHz spectral width. The output polarization is linear and Horizontal in it's own referential.

Polarization Controller and Polarization Beam Splitter

The polarization controller as already described in Section.1.1 is a fiber-coil polarization controller, and it's effect is adjusted manually. For the purpose of our application it was simply used to achieve higher power at the output of the Polarization Beam Splitter. The polarizing beam splitter divides incident unpolarized light into two orthogonally polarized beams. In our case the polarization at it's input is not completely unpolarized, but it's use will guarantee a higher degree polarized light at the EPC input. It will also help maintain a well defined SOP at the EPC input. One of the reasons we don't connect the laser directly to the EPC in our lab setup is because they are at two different heights leading to fiber optic that will bend in the air, this causes some random variations in the EPC input SOP.

Potenciometer Plate and Voltage Source

The potenciomenter plate, Fig 4.3, is a fairly rudimentary device. In it we have 16 potenciometers that can be adjusted using a screw driver, there are 16 potenciometers because we can have up to 8 stages in the EPC. We used the potenciomenters plate to apply the voltage values at the EPC pins, the values at each pin are monitored with a single multimeter. For the voltage source all we need is something capable of going over and under 0V by a certain amount, we expected -30 to +30 to be enough. The current consumption done by the EPC for both V_A and V_C is 76 mA.

Single mode fiber optics

The fiber optics used are single mode, this type of fiber optics do not guarantee that the polarization at the output will be equal to it's input, unlike polarization maintaining fiber optics. Although the input polarization might suffer some changes, they can be minimized and made constant by stretching and fixing the fiber to a plane surface.

Polarimeter and measurement setup

The polarimeter used was the model PAN5710IR3 from Thorlabs. This device provides a lot of useful features like the "real time" representation of the SOP in the Poincaré sphere, using it's graphical user interface software, and it's ability to share measurements trough the use of a 9-Pin D-Sub connector. We took advantage of both this features by using a computer to keep track of the SOP in the Poincaré sphere and another to to register and process measurements. A Bus-Powered USB Multifunction I/O Device (NI USB-6009) was used as a ADC to connect the polarimeter to the processing computer.

Ideally this setup should actuate automatically back on the EPC pins once the measurements are made, but in this tests the actuation's were done by hand. Because they were done by hand the graphical interface made the tests easier to process in "real time". For a more automated setup, only the register and control device would be necessary.

4.2. Laboratory results

The input SOP at the entrance of the EPC was measured using the polarimeter, it was $S_1 = 0.559$, $S_2 = 0.124$ and $S_3 = -0.915$, there are some uncertainties associated with the measurement's due to possible SOP modifications inside the optical fiber while the measurement is being done with the polarimeter since the optical fibre is not the same as the one used by the EPC and neither is the orientation angle. Adding to that due to the need for mechanical interactions with the EPC to measure the voltage levels at the pins, changes in the output SOP can happen. The effect of this mechanical interactions is analyzed further in the document, Section.4.3. The Stokes parameters measured can be normalized so that $\sqrt{S_1^2 + S_2^2 + S_3^2} = 1$, that means the previous measurement could be written as $S_1 = 0.517$, $S_2 = 0.114$ and $S_3 = -0,847$. The input SOP is constant for all measurements presented in this chapter.

For the input SOP described above, and while using 6 stages, the following α and Δ combinations were numerically calculated:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
α_1	4,57279	1,70192	0,46837	2,71855	1,69160	1,80681
α_2	4,44504	1,72987	0,75444	2,66522	1,79984	1,66761
α_3	4,44990	1,59737	0,49861	2,69172	1,59614	1,74955
α_4	4,37001	1,59786	0,60182	2,71202	1,81840	1,77987
α_5	4,47262	1,69933	0,52683	2,83930	1,74015	1,83126
α_6	4,34536	1,73585	0,73136	2,71449	1,62153	1,71214
Δ_1	0,03499	0,03265	0,07272	0,04565	0,10271	0,00702
Δ_2	0,03309	0,03159	0,02442	0,06191	0,09374	0,03849
Δ_3	0,01793	0,07009	0,08767	0,04458	0,10608	0,00081
Δ_4	0,02338	0,08302	0,02697	0,06202	0,07574	0,00945
Δ_5	0,02564	0,07420	0,04727	0,03889	0,10519	0,05780
Δ_6	0,03554	0,04770	0,02653	0,05312	0,10745	-0,02429

Table 4.1: α and Δ values for $S_1 = 0.517$, $S_2 = 0.114$, $S_3 = -0,847$ input SOP.

The SOP error, obtained numerically, for each of the α and δ combinations in Table.4.1 is under 10^{-5} for all stokes parameters. That means that, as an example, for the V SOP the stokes parameters errors are, $|-1 + S_1| \leq 10^{-5}$,$|S_2| \leq 10^{-5}$ and $|S_3| \leq 10^{-5}$.

Using the calibration values from the datasheet it corresponds to the following voltage values:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
V_{A1}	-12,310	-8,699	-12,567	-7,336	-4,456	-10,236
V_{A2}	-10,533	-7,335	-9,393	-4,686	-3,178	-7,023
V_{A3}	-9,175	-4,612	-10,540	-5,209	-2,623	-8,448
V_{A4}	-11,644	-6,309	-11,325	-6,346	-5,907	-10,292
V_{A5}	-8,393	-2,921	-8,418	-4,696	-0,847	-3,571
V_{A6}	-8,690	-4,601	-7,741	-3,695	-1,457	-9,099
V_{C1}	6,241	9,920	13,838	7,075	13,250	8,678
V_{C2}	7,387	11,003	11,499	8,052	13,338	11,460
V_{C3}	8,204	13,079	15,968	8,202	14,976	9,435
V_{C4}	9,474	15,438	13,168	9,326	14,009	11,487
V_{C5}	9,923	15,010	15,166	10,137	16,163	13,758
V_{C6}	8,380	13,020	12,969	9,389	16,432	9,783

Table 4.2: Voltage values to apply to the EPC calculated through PSO for $S_1 = 0.517$, $S_2 = 0.114$, $S_3 = -0,847$ input SOP.

This voltage values present the following maximum voltage intervals between the basis:

	2 Basis			**3 Basis**
	LB $\rightarrow$ DB	LB $\rightarrow$ CB	DB $\rightarrow$ CB	
max voltage interval	7,765	8,051	8,111	8,111

Table 4.3: Voltage intervals for the 2 Basis and 3 Basis combinations calculated from Table.4.2.

The application of the values,Table.4.2, on the EPC pins gave the following normalized measurements:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
S_1	0,979	-0,336	-0,604	-0,440	-0,039	0,319
S_2	0,0369	-0,301	-0,303	0,829	-0,733	0,553
S_3	0,197	-0,892	-0,736	-0,343	-0,678	-0,769

Table 4.4: Measured SOP's, in the lab, after applying the voltage values from Table.4.2 to the EPC pins.

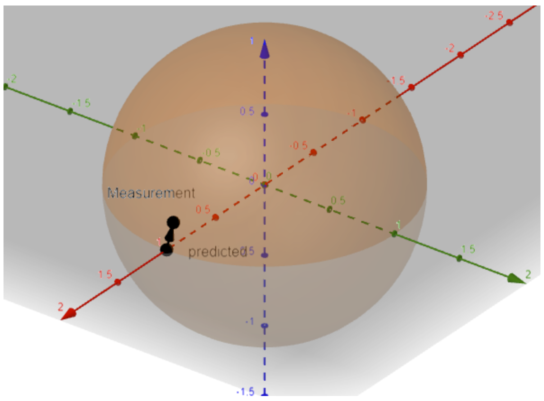


Figure 4.4: Difference between the measured SOP and the numerical predicted H SOP in the poincaré sphere, $\gamma = 11.59°$

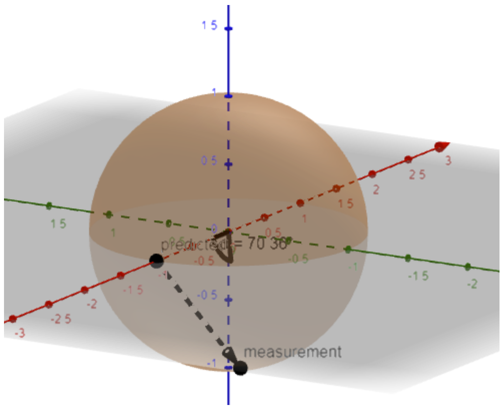


Figure 4.5: Difference between the measured SOP and the numerical predicted V SOP in the poincaré sphere, $\gamma = 70.36°$

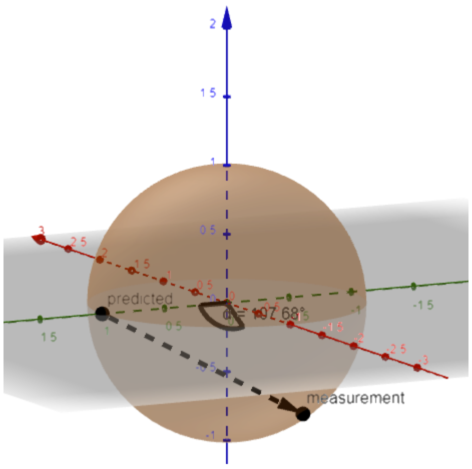


Figure 4.6: Difference between the measured SOP and the numerical predicted +45 SOP in the poincaré sphere, $\gamma = 107.68°$

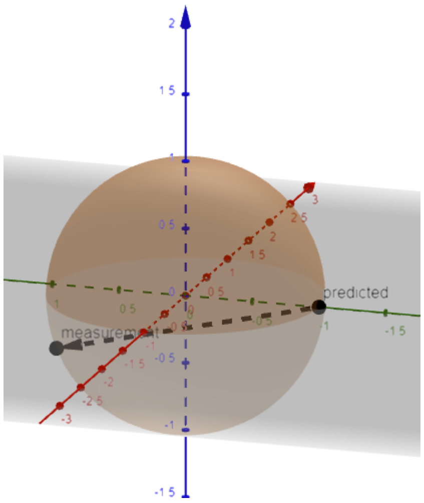


Figure 4.7: Difference between the measured SOP and the numerical predicted -45 SOP in the poincaré sphere, $\gamma = 146.05°$

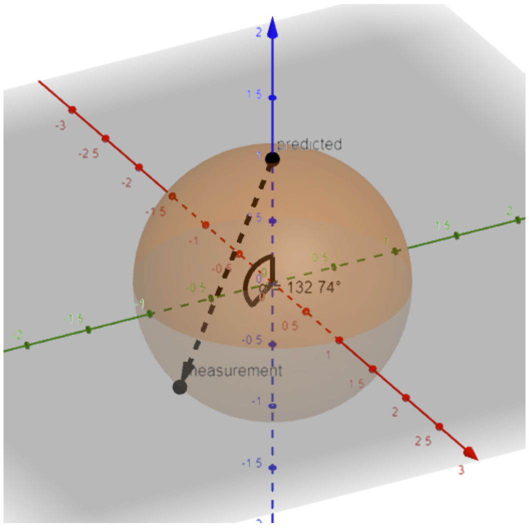


Figure 4.8: Difference between the measured SOP and the numerical predicted right circ SOP in the poincaré sphere, $\gamma = 132.74°$

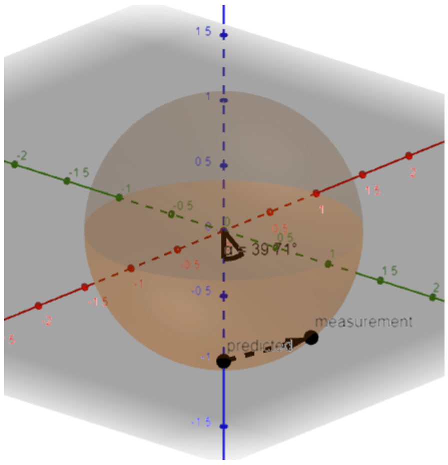


Figure 4.9: Difference between the measured SOP and the numerical predicted left circ SOP in the poincaré sphere, $\gamma = 39.71°$

In figures.4.4 to 4.9 we see a representation, in the poincaré sphere, of the SOP's measured, the values represented are the ones in Table.4.4. The variable γ, presented in the captions, is the angle between the stokes vector for the expect SOP and the measured SOP.

So as expected, due to the lack of knowledge of the real calibration parameters, the voltage combinations applied using the numerical results and the datasheet calibration parameters give measurements as far as 146.05° from the desired result. The values of γ vary considerably. This results do not disprove the numerical results of α and Δ.

After we proved again that the calibration parameters given by the datasheet do not correspond to the ones found in the device, we move on to the concrete goal that we can evaluate with the information and laboratory setup available. We are going to try to obtain voltage combinations that result in each of the 6 SOP's, and we will try do so while limiting ourselves to voltage intervals similar to the ones obtained in Table.4.3. The measurements in Table.4.4 were not useless, from them we can collect reference points. The voltage values for H gave a measurement really close to the supposed one, and even though the others didn't, the measurement's V gave provided a SOP close to left circular, -45 a SOP close to 45, right circular a SOP close to -45. Now we have some reference points that can be adjusted manually or trough the use of an algorithm, like PSO. From previous measurements I also had voltage combinations that corresponded to SOP measurements close to V and right circular. I constructed a table with all the best reference points I obtained, and decided to try to optimize the measurements from there:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
V_{A1}	-12,311	-14,964	-7,337	-16,321	-9,887	-8,472
V_{A2}	-10,534	-11,030	-4,686	-12,661	-9,954	-4,189
V_{A3}	-9,175	-6,773	-5,209	-12,373	-8,000	-7,609
V_{A4}	-11,644	-10,540	-6,347	-14,441	-9,864	-7,674
V_{A5}	-8,394	-7,124	-4,697	-10,358	-6,681	-3,882
V_{A6}	-8,691	-6,105	-3,695	-11,672	-6,277	-3,352
V_{C1}	6,241	-4,536	7,076	2,426	7,599	10,395
V_{C2}	7,387	2,792	8,052	6,697	4,538	13,025
V_{C3}	8,205	-1,667	8,203	5,582	0,225	10,015
V_{C4}	9,475	2,693	9,326	7,815	7,952	14,001
V_{C5}	9,923	7,323	10,138	9,314	13,550	15,777
V_{C6}	8,381	4,615	9,390	8,031	9,116	14,984

Table 4.5: Voltage reference points decided through previous measurement analysis.

That corresponded to the following normalized SOP measurements:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
S_1	0,979	-0,969	-0,440	0,333	0,641	-0,336
S_2	0,036	-0,215	0,829	-0,937	0,474	-0,301
S_3	0,197	-0,114	-0,343	0,098	0,603	-0,892
γ	$11,59°$	$14,11°$	$33,95°$	$20,33°$	$52,89°$	$26,85°$

Table 4.6: Measured SOP's, in the lab, after applying the voltage values from Table.4.5 to the EPC pins.

This reference points can be drawn in the poincaré sphere as follows:

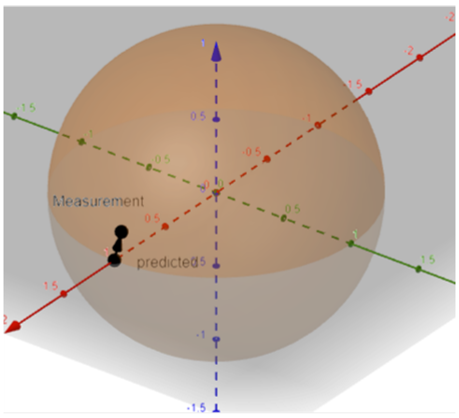


Figure 4.10: H SOP reference point in the poincaré sphere

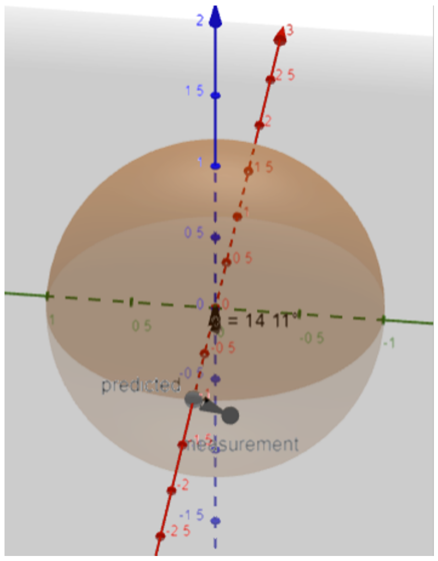


Figure 4.11: V SOP reference point in the poincaré sphere

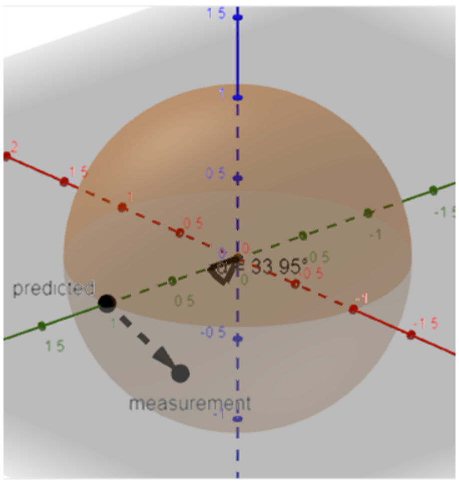


Figure 4.12: +45 SOP reference point in the poincaré sphere

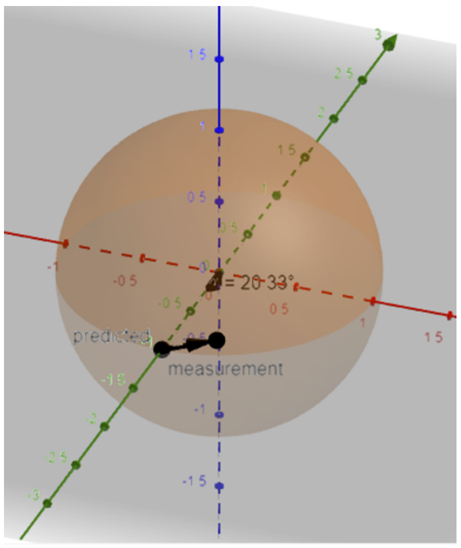


Figure 4.13: -45 SOP reference point in the poincaré sphere

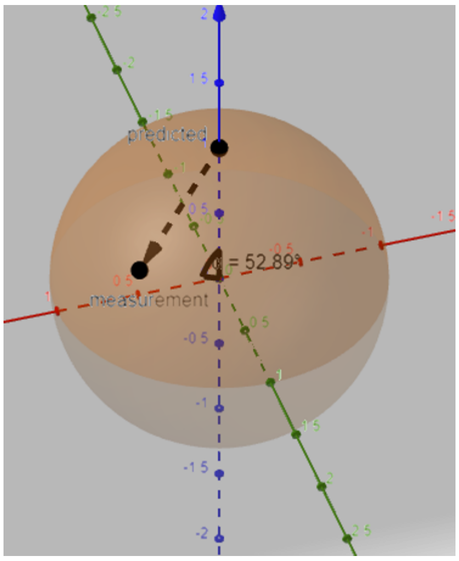


Figure 4.14: Right circ SOP reference point in the poincaré sphere

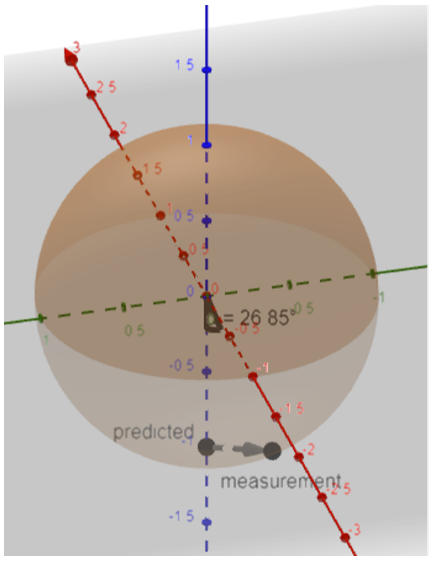
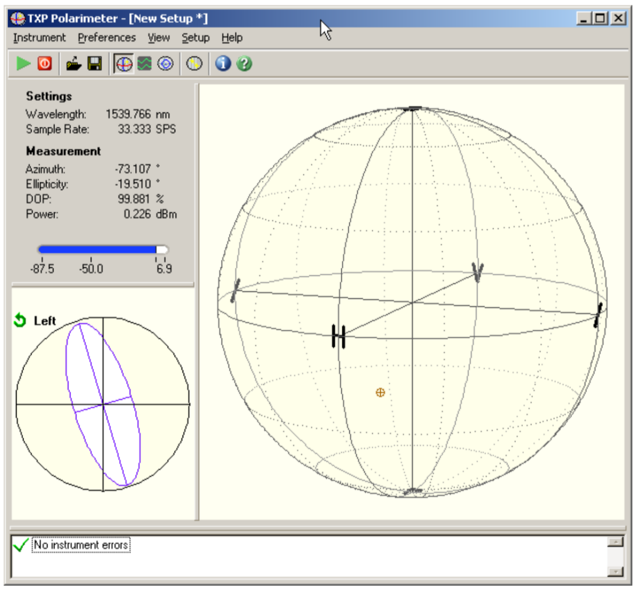


Figure 4.15: Left circ SOP reference point in the poincaré sphere

The process that follows is fairly simple, since the polarimeter is connected to a computer that possesses a visual interface that shows the measured SOP in "real time" drawned in the Poincaré sphere, Fig.4.16, we can see the effect that each voltage value applied to a certain EPC pin has on the SOP measured by the polarimeter.



Settings

Wavelength: 1539.766 nm
Sample Rate: 33.333 SPS

Measurement

Azimuth: -73.107 °
Ellipticity: -19.510 °
DOP: 99.881 %
Power: 0.226 dBm

No instrument errors

Figure 4.16: Polarimeter graphical user interface (GUI).

I corrected the SOP voltage values in the same order that they were calculated using PSO, H followed by V, then +45, -45, right circular and left circular. The first SOP, H, was found without the need to do many adjustments. The voltage values for V were found while keeping in account the voltage intervals between H and V, that means that we decreased the larger voltage intervals and increased the smaller ones to compensate. The same procedure was done for all the other SOP's, in the end when finding the left circular SOP the voltage intervals of it to H, V, +45, -45 and right circular were all being analysed.

While reading my procedure one might think that this method could be done using and algorithm like PSO that takes in the measurements of the polarimeter and the voltage values of the SOP's and calculates the best voltage values based on that information, and that could indeed be done. A version of PSO that took in the measurements of the polarimeter and the voltage values of the SOP's was developed, but since the voltage to each pin was applied by hand trough the potenciometer plate, and the voltage values at each pin were simultaneously being measured using a single multimeter, this whole procedure becomes very time consuming. Due to that reason validation of this application of the algorithm was not finished, even

though some iterations were done. The developed version can be found in, github.com/Ch0s3n0ne/DV-QKD-Optimizer and in Appendix.B.
The voltage value obtained in the lab for the 6 SOP's were:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
V_{A1}	-10,241	-14,830	-10,051	-15,920	-7,735	-10,223
V_{A2}	-9,979	-8,171	-6,216	-12,590	-12,15	-7,496
V_{A3}	-9,071	-7,552	-5,666	-13,220	-10,119	-4,521
V_{A4}	-11,750	-9,671	-7,241	-15,040	-10,398	-8,013
V_{A5}	-10,110	-6,762	-5,023	-11,620	-6,334	-3,822
V_{A6}	-9,881	-4,016	-4,324	-11,240	-6,085	-4,855
V_{C1}	6,249	0,4598	4,385	5,662	3,022	7,843
V_{C2}	7,401	0,501	4,757	5,891	0,97	10,641
V_{C3}	8,208	1,691	6,991	5,889	1,749	9,84
V_{C4}	9,466	2,661	9,063	7,287	7,133	12,86
V_{C5}	9,917	3,744	9,486	4,810	6,373	14,65
V_{C6}	8,382	2,679	8,952	4,858	9,086	13

Table 4.7: Combination of voltage values, obtained in the lab, that result in accurate measurements of the desired SOP's

Corresponding to the following normalized SOP measurements:

	H	**V**	**45**	**-45**	**right circ**	**left circ**
S1	0,999	-0,999	-0,006	0,005	0,004	-0,001
S2	0,029	0,022	0,999	-0,999	0,010	0,011
S3	0,001	0,016	0,009	0,011	0,999	-0,999
γ	$1,676°$	$1,602°$	$0,649°$	$0,747°$	$0,632°$	$0,642°$

Table 4.8: SOP's measured after applying the voltage combinations in Table.4.7 to the EPC pins.

The voltage values found allow for the following voltage intervals between the basis:

	2 Basis			**3 Basis**
	LB $\rightarrow$ DB	LB $\rightarrow$ CB	DB $\rightarrow$ CB	
max voltage interval	7,799	10,906	9,84	10,906

Table 4.9: Voltage intervals, obtained in the lab, for the 2 Basis and 3 Basis combinations.

We were able to obtain a voltage interval similar to the one calculated numerical for LB to DB basis transition, the other transitions gave results a little bit further away, but I believe those could have been improved have them not been found by hand. In another test, done previously,we were also able to validate the voltage intervals, with different voltage values, in which all basis transitions were achieved in under 10V voltage intervals, those results were published in [48].

The fact that we were able to achieve twice with distinct voltage values all SOP's in which the voltage intervals fall within the predicted numerical results, proves that the algorithm can be trusted with those predictions. Unfortunately since the calibration parameters of the EPC are not known further conclusions regarding the precision of the numerical calculations cannot be done. Besides the calibration parameters, knowing the exact input SOP also proves to be quite complicated, it's measurement affects the values obtained from the numerical calculations as well. A method to adjust the output SOP, similar to the one presented here, seems to always be necessary. Something else to keep in mind regarding the results obtained is that since the EPC is a sensible device, attached to the other sensible components used in QKD, the measurements obtained for the same voltage values might vary slightly at each day.

4.3. Laboratory results uncertainties

In this section we will try to present the method used to calculate the uncertainties added due to the laboratory implementation done. We will try take in account the already known uncertainties from the multimeter, polarimeter, ADC and most importantly we will try to analyse the effect that the mechanical iterations done to the EPC, during the work, had on the SOP's measured.

To start I will analyse the collected data and evaluate parameters like the maximum deviation and standard deviation, to compare directly the effect of measurements done with and without mechanical interaction with the EPC.

To have a prediction of the effect that the entire setup has on the measurements, without the mechanically interactions, we collected data during around 1h doing measurements at every 3 min, 34 data points were collected. Since the measurements of the stokes vectors are simply points in the Poincaré sphere it makes sense to convert from cartesian coordinates to spherical coordinates following the logic of the poincaré sphere presented in, Fig.2.5. I will consider $2\alpha = \psi$ and $2\beta = \theta$, it's important to keep in mind how θ is calculated, being in reference to the xy plane. Knowing that:

$$\psi = \arctan\left(\frac{S_2}{S_1}\right) \tag{4.1}$$

$$\theta = \arctan\left(\frac{S_3}{\sqrt{S_1^2 + S_2^2}}\right) \tag{4.2}$$

We can convert every Stokes measurement to the angles ψ and θ. Calculations over the entire dataset give us a average value for $\theta = -43.573°$ and $\psi = -90.249°$ a standard deviation of $\theta = 0.216°$ and $\psi = 0.685$. The maximum variation compared to the average value was, $0.393°$ for θ and $-1.551°$ for ψ. The average SOP as a stokes vector can be calculated with Equations.2.7,2.8 and 2.9, and corresponds to $S_1 = -0.003$, $S_2 = -0.724$ and $S_3 = -0.689$.

To try and analyse the effect of the mechanical interactions with the EPC, measurements were collected during 10min with intervals of 5 seconds, this equated to 121 measurements. We can do the same procedure as before to obtain the following results:

	With interactions	**No interactions**
standard deviation θ	1.749°	0.216°
max deviation θ	5.088°	0.393°
standard deviation ψ	1.658°	0.685°
max deviation ψ	−4.849°	−1.551°

Table 4.10: Standard deviation measurements with and without mechanical interactions with the EPC.

Something else that is important to keep in mind is that, the measurements were not done for the same SOP, the average values for θ and ψ, for the measurements with interactions, were $-56.018°$ and $41.307°$.The stokes vector would be $S_1 = 0.419$, $S_2 = 0.369$ and $S_3 = -0.829$. Just from the results obtained in Table.4.10, we can see a very clear increment in deviations due to mechanical interactions, this will undoubtedly cause an increment in the uncertainty of the results obtained in the lab. Obtaining an angle deviation in the poincaré, let's call it γ, from this calculations is simple, all we need to do is measure the angle between the two SOP, the average and the one with the deviation. I will consider that the γ standard deviation contribution due to θ and ψ is on theory the same but due to lack of measurements or some other unknown factor might end up different.

	With interactions	**No interactions**
γ standard deviation due to θ	1.749°	0.216°
γ standard deviation due to ψ	0.927°	0.496°

Table 4.11: Standard uncertainties as a angle in the Poincaré sphere surrounding the predicted SOP.

Now I will try to do a more in depth analysis, also taking in account the already known uncertainties from the multimeter, polarimeter and ADC. I will use the **G**uide to the Expression of **U**ncertainty in **M**easurement (GUM) standard and it's tools to evaluate as express the uncertainties of our measurements.

The voltage error will be directly correlated to the characteristics of the multimiter so for a voltage values from -10V to 10V the uncertainty would be 0.001V and for values over and under it would be 0.1V. Unfortunately connecting the voltage uncertainties directly to the SOP measurement uncertainties is not possible, with the information we posses. The data is passed from the polarimeter to a computer trough a USB Multifunction I/O Device, that will serve mainly as an ADC. In the selected configuration the ADC is able to utilize 13 bits, in a measuring scale of 20V (-10 to 10V), meaning we have a resolution of 2.44mV. The ADC also has some noise in this configuration that can be estimated with a value of $0.5V_{rms}$.

The SOP measurements will also have some uncertainty added due to the accuracy and resolution of the polarimeter. The polarimeter has an accuracy of $\pm 0.25°$ and a resolution of 0.01° in the poincaré sphere. This uncertanty in the poincaré sphere can be connected to a uncertainty in the measured values of θ and ψ.

For the measurements obtained and discussed previously, I will start by doing do a Type A evaluation. Type A evaluation consists in obtaining the values of mean and standard deviation from the measured values and from there get the standard uncertainty. The standard uncertainty can be calculated by:

$$u(x_i) = \frac{\sigma(x_i)}{\sqrt{n}} \tag{4.3}$$

σ is the standard deviation and n is the number of measurements. Due to the disparity in the number of measurements for the two situations with and without interactions, 121 and 34 respectively, going from here would still give us a bigger uncertainty for the measurements done with interactions, but it's simply not realistic of the real measurements usually done. It's more realistic that I would take 3 measurements, so I will take a sample from the measured values that come close to the σ and average obtained in Table.4.10. Leaving us with:

	With interactions	**No interactions**
standard uncertainty θ	1.010°	0.127°
standard uncertainty ψ	0.957°	0.402°

Table 4.12: Standard uncertainty associated with the selected sample of measured values.

We need to take in account, at this stage, the uncertainty added due ADC characteristics. To find the standard uncertainty due to the ADC, and the polarimeter we can do Type B evaluation in which we consider:

$$u(x_i) = \frac{a_i}{\sqrt{k_d}} \tag{4.4}$$

a_i is the accuracy or resolution and k_d is a constant that is usually equal to 3 when working with accuracy of resolution limits. So the combined uncertainty, precision+noise, of the voltage measurements done by the ADC will be 1.438 mV. Propagating the uncertainties when calculating the values of θ and ψ and combining them with the uncertainties from Table.4.12 gives us.

	With interactions	**No interactions**
combined uncertainty θ	1.011°	0.129°
combined uncertainty ψ	0.961°	0.403°

Table 4.13: Combined uncertainty associated with the measured values and ADC characteristics.

Since the polarimeter accuracy and resolution are both represented as an angle on the poincaré sphere, Fig.4.17. I can say that it affects the uncertainty of θ in the same amount but not ψ, to find the uncertainty added by the polarimeter accuracy and resolution we need to analyse the circumference surrounding the average SOP and project it in the xy-axis, Fig.4.18.

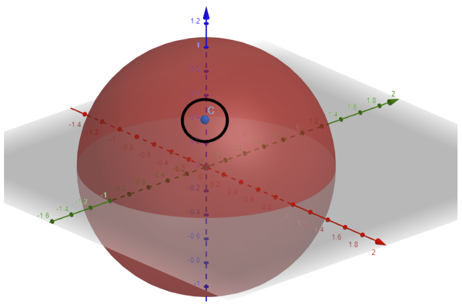


Figure 4.17: Example representation of SOP uncertainty angle in the poincaré sphere.

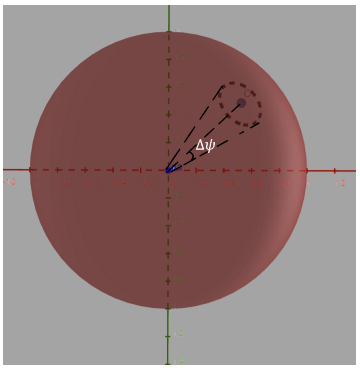


Figure 4.18: Example projection of the uncertainty angle in the xy-axis.

From here we can calculate the corresponding uncertainty added to ψ. As referred previously the accuracy and resolution of the polarimeter will be taken in account using Type B analysis, equation.4.4. Combining all the uncertainties calculated up until now will give us:

	With interactions	**No interactions**
combined uncertainty θ	1.021°	0.193°
combined uncertainty ψ	0.995°	0.449°

Table 4.14: Combined uncertainty associated with the measured values, ADC characteristics and polarimeter characteristics.

We can also calculate the expanded uncertainty, in which we multiply the previous obtained results with a coverage factor, the coverage factor is chosen from the T students table and in our case I will chose one that gives a 95% probability of a 4^{th} measurement being inside this expanded interval:

	With interactions	**No interactions**
expanded uncertainty θ	4.215°	0.427°
expanded uncertainty ψ	3.741°	1.396°

Table 4.15: Expanded uncertainty values associated with a 95% probability of the next measurement being inside the calculated interval.

Calculating the angle uncertainty, I consider that the γ uncertainty contribution due to θ and ψ is on theory the same but due to lack of measurements or some other unknown factor might end up different.

	With interactions	**No interactions**
γ uncertainty due to θ	4.215°	0.427°
γ uncertainty due to ψ	2.090°	1.0124°

Table 4.16: Final uncertainty values, of the full laboratory setup, as a angle in the Poincaré sphere surrounding the predicted SOP.

In conclusion we can observe an actually significant increment in γ uncertainty due to mechanical interactions with the EPC, under this conditions a much higher, 200+, number of measurements needs to be done, in order to reduce the uncertainties to values similar to the ones with no interactions, under the same measurement intervals. In Fig.4.19 I present the γ deviation that I saw once in the GUI due to doing measurements of the voltages on the EPC pins. The high deviation seen in Fig.4.19 would, after some time and no interactions, collapse to a value closer to the expected marked as a red dot. The current implementations has very real limitations for this study that can only be corrected once a electronic driver to control the EPC is implemented.

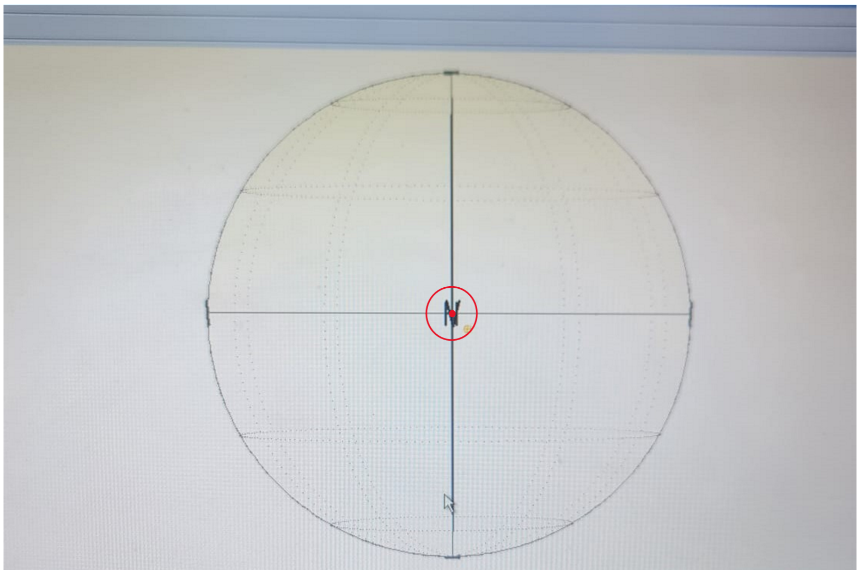


Figure 4.19: Maximum polarization error observed in the GUI due to mechanical interactions with the EPC, yellow dot is the current measurement and the red dot is the measurement before the interactions.

Chapter 5

Conclusions

The main goal of this dissertation was to the find a method that would allows us to calculate optimum values for the voltages that should applied to each of an Electronic Polarisation Controller (EPC) stages. The optimum values that we intended to find were related to the application of the EPC as an transmitter in a QKD. The application in QKD condenses into the ability to obtain 6 predefined SOP's for the light at the EPC output. Each stage of the EPC is an waveplate that as certain characteristics, phase difference between the fast and slow axis and the angle of the fast axis, this characteristics can be related to a voltage value applied to said said waveplate. There are an immense number of combinations of for waveplate characteristics that can achieve the same SOP at the output. With the work presented in this dissertation one should be able find the ideal combinations for the waveplate characteristics. The ideal combinations result in minimal voltage intervals when transitioning between each of the desired SOP's. To find the best solutions a machine learning algorithm and studied and developed, the algorithm is referred as Particle Swarm Optimization (PSO). The development of the algorithm was completed and the results are promising.

The main issues came with the laboratory validation. Due to lack of knowledge of the calibration parameters of the EPC, previous work on this area was not successful [29], exact validation of numerical values obtained by the machine learning algorithm was not possible. This lead the necessity to develop a method to find the required 6 SOP's in the lab using the limited, not automated resources. We did that by taking in account measurements given while testing of the numerical results. The results were adapted by hand while following certain guidelines. All 6 SOP's were successfully found in the lab.

In the end we were able to validate that the voltage intervals predicted by the algorithm were very close to the ones obtained in the laboratory. Only the 6 stages EPC, using all the stages, was tested. Limitations imposed by the laboratory setup didn't allow for much more. The results can be taken with a lot of optimism since they were positive and obtained by hand, there is still room for improvement.

With the work presented in this dissertation a much better better description of the working characteristics of an Electronic Polarisation Controller (EPC) based in electro-optic crystals can be done. Our results show that, the standard full range of the drive voltages, which in the employed EPC equals 140 V, can be reduced to values under of 10 V for each stage controlling pin, while using 6 stages. Reduction in the required range voltages for each waveplate represents a positive realization towards practical and efficient implementation of QKD protocols employing polarization encoding, as it will allow using simpler electronic drivers to control the polarization encoding subsystems. Furthermore, information like the relation between the number of stages used and the maximum voltage intervals can also aid in future investment choices for the EPC's used in the transmission and reception stages of QKD.

The method by which we were able to surpass the lack of knowledge regarding the calibration parameters and find the required 6 SOP's, can also be easily automated using PSO. A variation of the algorithm to surpass the lack of knowledge of the calibration parameters was developed, but due to lack of ability to automate the procedure with the equipment available, it was not fully tested.

Particle Swarm Optimization (PSO) used as the machine learning algorithm, shows promise for future work with the EPC. To conclude, the study was a success and the some of the main questions regarding the kind of device one would need to use to control the EPC were answered.

5.1. Future work

Regarding the equipment's used in the current setup, an improvement that can greatly it is the use of an electronic driver connected directly to the EPC pins. For a calibration stage while using the PSO implementation described in [47], an electronic driver like the one being currently developed capable of achieving voltage values from -70 to 70 V should be still be necessary. For applying QKD a voltage driver with a range of -20 to 20V should be more than enough. Another improvement that can be done is changing the ADC that is currently in use to one that would result in less noise. The ADC currently in use is a very plug and play device with cables hanging in the air, the development of a very simple PCB that converts the voltages from the polarimeter outputs, -2.5 to 2.5 V, to a more usable range with a FPGA or Arduino, 0 to 1V, could be used.

Something else that also makes sense to discuss is what device we should use to control an electronic driver that will eventually be connected to the EPC. As a starting point, I think an arduino like the arduino DUE, with clock frequency of 84 MHz, should work well during a test phase, because it's simple to manipulate, and has a large libraries of code available. It could also be used in a real implementation, depending on the frequencies we actually can achieve. For a permanent implementation, using a FPGA seems like the most sensible decision. The connection with the driver shouldn't be too hard to implement, there is already a wide availability of FPGA's at IT and other implementations have been done using them, which can be useful when replacing previous EPC setups. Some of the more heavy work related to implementing an algorithm like PSO, could be given to a regular computer connected through UART to the FPGA. That would be temporary, in my opinion, because I believe an implementation of PSO could be done in the Arm microprocessor of and SOC FPGA or even in it's fabric.

Changing the current mechanical EPC used to apply QKD in Instituto de Telecomunicaçãoes de Aveiro (IT), to the one analysed in this dissertation, seems very promising and should be straight forward once the issues I presented above are tackled.

Bibliography

[1] Physical-Layer Security and Quantum Key Distribution, Ivan B. Djordjevic, https://doi.org/10.1007/978-3-030-27565-5

[2] Muga, Nelson J., Mariana F., Ramos, Sara T., Mantey, Nuno A., Silva, and Armando N., Pinto. "FPGA-assisted state-of-polarisation generation for polarisation-encoded optical communications".IET Optoelectronics 14, no.6 (2020): 350-355.

[3] N. J. Muga, A. N. Pinto, M. F. S. Ferreira, and José R. Ferreira da Rocha. Uniform polarization scattering with fiber-coil-based polarization controllers. Journal of Lightwave Technology, 24(11):3932–3943, 2006.

[4] Nelson J. Muga Mário F. S. Ferreira and I. Armando N. Pinto Senior Member, "QBER Estimation in QKD Systems With Polarization Encoding," Journal Of Lightwave Technology, vol. 2, no. 3, 2011.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[6] Forney GD (2003) On the role of MMSE estimation in approaching the information theoretic limits of linear Gaussian channels: Shannon meets Wiener. In: Proceedings of the 41st annual allerton conference on communication, control, and computing, Monticello, IL, pp 430–439

[7] Shor, P. & Preskill, J. Simple Proof of Security of the BB84 Quantum Key Distribution Protocol. *Physical Review Letters*. **85**, 441-444 (2000,7), https://doi.org/10.48550/arXiv.quant-ph/0003004

[8] A. Duplinskiy, V. Ustimchik, A. Kanapin, V. Kurochkin, and Y. Kurochkin, "Low loss QKD optical scheme for fast polarization encoding," Opt. Express 25, 28886-28897 (2017)

[9] Almeida, Á., Muga, N., Silva, N., Stojanovic, A., André, P., Pinto, A., Mora, J. & Capmany, J. Enabling quantum communications through accurate photons polarization control. *8th Iberoamerican Optics Meeting And 11th Latin American Meeting On Optics, Lasers, And Applications*. **8785** pp. 1278 - 1285 (2013), https://doi.org/10.1117/12.2026208

[10] Yang Li, Yu-Huai Li, Hong-Bo Xie, Zheng-Ping Li, Xiao Jiang, Wen-Qi Cai, Ji-Gang Ren, Juan Yin, Sheng-Kai Liao, and Cheng-Zhi Peng, "High-speed robust polarization modulation for quantum key distribution," Opt. Lett. 44, 5262-5265 (2019)

[11] Ramos, M.F., Pinto, A.N. Silva, N.A. Polarization based discrete variables quantum key distribution via conjugated homodyne detection. Sci Rep 12, 6135, (2022), https://doi.org/10.1038/s41598-022-10181-4

[12] Grünenfelder, F., Boaron, A., Rusca, D., Martin, A. & Zbinden, H. Performance and security of 5 GHz repetition rate polarization-based quantum key distribution. *Applied Physics Letters*. **117**, 144003 (2020), https://doi.org/10.1063/5.0021468

[13] GreGoire Ribordy, Nicolas Gisin, Olivier Guinnard, Damien Stuck, Mark Wegmuller Hugo Zbinden (2004) Photon counting at telecom wavelengths with commercial InGaAs/InP avalanche photodiodes: Current performance, Journal of Modern Optics, 51:9-10, 1381-1398, DOI: https://doi.org/10.1080/09500340408235280

[14] Grünenfelder, F., Boaron, A., Rusca, D., Martin, A. & Zbinden, H. Simple and high-speed polarization-based QKD. *Applied Physics Letters*. **112**, 051108 (2018), https://doi.org/10.1063/1.5016931

[15] Schneier B (2015) Applied cryptography, second edition: protocols, algorithms, and source code in C. Wiley, Indianapolis, IN

[16] Drajic D, Ivanis P (2009) Introduction to information theory and coding, 3rd edn. Akademska Misao, Belgrade, Serbia (in Serbian)

[17] Katz J, Lindell Y (2015) Introduction to modern cryptography, 2nd edn. CRC Press, Boca Raton, FL

[18] Valerio Scarani, Antonio Acín, Grégoire Ribordy, and Nicolas Gisin. Quantum Cryptography Protocols Robust against Photon Number Splitting Attacks for Weak Laser Pulse Implementations. Physical Review Letters, 92(5):4, 2004.

[19] Okoshi, T.; Cheng, Y.H.; Kikuchi, K.: 'New polarisation-control scheme for optical heterodyne receiver using two Faraday rotators', Electronics Letters, 1985, 21, (18), p. 787-788, DOI: 10.1049/el:19850555 IET Digital Library, https://digital-library.theiet.org/content/journals/10.1049/el_555

[20] Diffie W, Hellman ME (1976) New direction in cryptography. IEEE Trans Inform Theory 22:644–654

[21] Dirac PAM (1958) Quantum mechanics, 4th edn. Oxford University Press, London

[22] Ekert A, Josza R (1996) Quantum computation and Shor's factoring algorithm. Rev Modern Phys 68(3):733–753

[23] Rivest RL, ShamirA, Adleman L (1978)Amethod for obtaining digital signatures and publickey cryptosystems. Commun ACM 21(2):120–126

[24] Wang, W., Tamaki, K. Curty, M. Measurement-device-independent quantum key distribution with leaky sources. Sci Rep 11, 1678 (2021), https://doi.org/10.1038/s41598-021-81003-2

[25] Bennet CH, Brassard G (1984) Quantum cryptography: public key distribution and coin tossing. In: Proceedings of the IEEE international conference on computers, systems, and signal processing, Bangalore, India, pp 175–179

[26] Le Bellac M (2006) An introduction to quantum information and quantum computation. Cambridge University Press

[27] Neilsen MA, Chuang IL (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge

[28] Bennett CH (1992) Quantum cryptography using any two nonorthogonal states. Phys Rev Lett 68(21):3121–3124

[29] Martins, L. Characterization Of Electro-Optic Polarization Controllers For Discrete-Variable Quantum Communications Transmitters. (University of Coimbra,2021,10)

[30] Dagmar Bruß. Optimal eavesdropping in quantum cryptography with six states. Phys. Rev. Lett., 81:3018–3021, Oct 1998.

[31] LoH-K,MaX,ChenK(2005) Decoy state quantum key distribution. Phys RevLett 94:230504

[32] Ekert, A. K., "Quantum cryptography based on Bell's theorem", Physical Review Letters, vol. 67, no. 6, 5 August 1991, pp. 661 - 663.

[33] Lucamarini M, Choi I, Ward MB, Dynes JF, Yuan ZL, Shields AJ (2015) Practical security bounds against the trojan-horse attack in quantum key distribution. Phys Rev X 5:031030

[34] Kurtsiefer C, Zarda P,Mayer S,Weinfurter H (2001) The breakdown flash of silicon avalanche photodiodes—back door for eavesdropper attacks? J Mod Opt 48(13):2039–2047

[35] Qi B, FungC-HF, LoH-K,MaX(2006) Time-shift attack in practical quantum cryptosystems. Quantum Inf Comput 7(1):73–82, https://arxiv.org/abs/quant-ph/0512080

[36] Lydersen L, Wiechers C, Wittmann S, Elser D, Skaar J, Makarov V (2010) Hacking commercial quantum cryptography systems by tailored bright illumination. Nat Photon 4(10):686

[37] Liao S-K et al (2017) Satellite-to-ground quantum key distribution. Nature 549:43–47

[38] Ursin R et al (2007) Entanglement-based quantum communication over 144 km. Nat Phys 3(7):481–486

[39] Boaron A et al (2018) Secure quantum key distribution over 421 km of optical fiber. Phys Rev Lett 121:190502

[40] H.-L. Yin et al., “Measurement-Device-Independent Quantum Key Distribution Over a 404 km Optical Fiber,” Phys. Rev. Lett., vol. 117, no. 19, p. 190501, Nov. 2016, doi: 10.1103/PhysRevLett.117.190501.

[41] Eleni Diamanti, Hoi Kwong Lo, Bing Qi, and Zhiliang Yuan. Practical challenges in quantum key distribution. npj Quantum Information, 2(1):1–12, 201

[42] H. J. Briegel, W. Dür, J. I. Cirac, and P. Zoller. Quantum repeaters: The role of imperfect local operations in quantum communication. Physical Review Letters, 81(26):5932–5935, 1998.

[43] Fernando, Warsha Delurgio, P. & Salvachua, B. & Stanek, R. & Underwood, David Lopez, D.. (2012). Optical Data Links – Technology for Reliability and Free Space Links. Physics Procedia. 37. 1805-1812. 10.1016/j.phpro.2012.02.507.

[44] D. Pereira, , N. A. Silva, and A. N. Pinto. "CA polarization diversity CV-QKD detection scheme for channels with strong polarization drift." . In IEEE International Conference on Quantum Computing and Engineering QCE (pp. -).2021.

[45] M. F. Ramos, , N. A. Silva, N. J. Muga, and A. N. Pinto. "Full polarization random drift compensation method for quantum communication".Optics Express 30, no.5 (2022): 6907-6920.

[46] S. T. Mantey, , M. F. Ramos, N. A. Silva, A. N. Pinto, and N. J. Muga. "Demonstration of a Polarization encoding Quantum Key Distribution System." . In SBRC Workshop de Comunicação e Computação Quântica WQuantum (pp. -).2022.

[47] Xi, L., Zhang, X., Tian, F., Tang, X., Weng, X., Zhang, G., Li, X. & Xiong, Q. Optimizing the Operation of $LiNbO_3$ -Based Multistage Polarization Controllers Through an Adaptive Algorithm. *IEEE Photonics Journal.* **2**, 195-202 (2010)

[48] Costa, H., Muga, N., Silva, N. & Pinto, A. Optimization of a Polarization-Encoding System for Practical Quantum Key Distribution. *SBRC Workshop De Comunicação E Computação Quântica WQuantum.* pp. - (2022,5)

[49] Dynamic Polarization Scrambler/Controller – PolaRITE II. (n.d.). General Photonics Corporation.

[50] Lithium Niobate Polarization Controller datasheet. (n.d.). Eospace.

[51] Breaking RSA Encryption quintessencelabs.com (accessed in 20/06/2022)

[52] A Survey of the Prominent Quantum Key Distribution Protocols cse.wustl.edu (accessed in 20/06/2022)

[53] IBM Documentation Public key algorithms ibm.com (accessed in 20/06/2022)

[54] Classification of polarization hyperphysics.phy (accessed in 20/06/2022)

Appendix A

PSO optimization code

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import time
#Get  the starting seed state
Seed=np.random.get_state()
%store -r
#apply a certain previously stored seed to the algorithm
np.random.set_state(seed_4stages6_t_7_76)

def wave_effect(alpha,delta):
    '''
    Function that represents effect of each wave plate in the SOP
    '''

    wave_plate_matrix=[[np.cos(alpha)**2+(np.sin(alpha)**2)*np.cos(delta), np.cos(alpha)
*np.sin(alpha)*(1-np.cos(delta)),np.sin(alpha)*np.sin(delta)],
                  [np.cos(alpha)*np.sin(alpha)*(1-np.cos(delta)),np.cos(delta)*np.cos(al
pha)**2+np.sin(alpha)**2,-np.cos(alpha)*np.sin(delta)],
                  [-np.sin(alpha)*np.sin(delta),np.cos(alpha)*np.sin(delta),np.cos(delta
)]]

    return wave_plate_matrix

def V_A_value(alpha,delta,V_o,V_pi,V_A_bias):

    '''
    V_A calculation
    '''

    V_A=2*V_o*delta*np.sin(alpha)-V_pi*delta*np.cos(alpha)+V_A_bias

    return V_A

def V_C_value(alpha,delta,V_o,V_pi,V_C_bias):

    '''
    V_C calculation
    '''
    V_C=2*V_o*delta*np.sin(alpha)+V_pi*delta*np.cos(alpha)+V_C_bias

    return V_C

def f_lab(alpha_array,delta_array,V_Ain_matrix,V_Cin_matrix,start_array,goal_array,first
_weight,sec_weight,third_weight,extra_sum,n_stages):
    '''
    alpha_array - array with the values of alpha for the various stages of a PSO particle
    delta_array - array with the values of delta for the various stages of a PSO particle
    V_Ain_matrix - matrix of values of VA for the previous polarization state for each st
age
    V_Cin_matrix - matrix of values of VC for the previous polarization state for each st
age
    start_array - Input State of Polarization as a Stokes Vector
    goal_array - Goal State of Polarization as a Stokes Vector
    first_weight,sec_weight,third_weight - weights of the cost function
    extra_sum - 1 or 0 , 1 if we are comparing to more than one previous calculated SOP
    n_stages - number of stages of the EPC used
    '''


    V_pi_array=np.array([56.3,56,55.9,56.1,56.1,56,56.1,56])
    V_o_array=np.array([27.2,27,27,26.9,26.8,27.2,26.8,27.2])

    #V_A,bias e V_C,bias não têm efeito no processo de otimização
    V_A_bias_array=np.array([-10.7,-9.3,-8.5,-10.9,-7.4,-7.6,-7.4,-7.6])
    V_C_bias_array=np.array([8.4,9.6,9.4,11.1,11.6,10.9,11.6,10.9])

    first_sum=0
    second_sum=0
    third_sum=0

    delta_array_rad=delta_array*np.pi*2

    M=wave_effect(alpha_array[n_stages-1],delta_array_rad[n_stages-1])
    i=n_stages-2
    while(i>=0):
        M=np.dot(M,wave_effect(alpha_array[i],delta_array_rad[i]))
        i-=1

    start_array=np.reshape(start_array,(3,1))

    calc_array=np.dot(M,start_array)

    calc_array=np.reshape(calc_array,(1,3))
    calc_array=calc_array[0]

    for i in range(0,3):
        first_sum+=abs((calc_array[i]-goal_array[i]))*first_weight


    if(extra_sum):


        for j in range(0,len(V_Ain_matrix)):

            for i in range(0,n_stages):

                second_sum+=(abs(V_A_value(alpha_array[i],delta_array[i],V_o_array[i],V_
pi_array[i],V_A_bias_array[i])-V_Ain_matrix[j,i])/(sec_weight*len(V_Ain_matrix)))**2


        for j in range(0,len(V_Cin_matrix)):
            for i in range(0,n_stages):
                third_sum+=(abs(V_C_value(alpha_array[i],delta_array[i],V_o_array[i],V_p
i_array[i],V_C_bias_array[i])-V_Cin_matrix[j,i])/(third_weight*len(V_Cin_matrix)))**2

    else:
        for i in range(0,n_stages):
            temp_calc=(abs((V_A_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_ar
ray[i],V_A_bias_array[i])-V_Ain_matrix[i]))/sec_weight)**2
            second_sum+=temp_calc


        for i in range(0,n_stages):
            temp_calc=(abs((V_C_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_ar
ray[i],V_C_bias_array[i])-V_Cin_matrix[i]))/third_weight)**2
            third_sum+=temp_calc


    z=first_sum+second_sum+third_sum

    return z
```

In [2]:

```
def index_array(lower_Q1_delta,lower_Q1_alpha,V_A_std,alpha_increment,delta_increment):

    '''
    This function is called by the function interval_selection and it
    serves to find the index breraking points
    '''

    index_alpha_array=[]
    for item in range(0,V_A_std[1,:].size):

        if(V_A_std[1,item] <= lower_Q1_alpha):
            index_alpha_array.append(item)

    index_delta_array=[]
    for item in range(0,V_A_std[0,:].size):

        if(V_A_std[0,item] <= lower_Q1_delta):
            index_delta_array.append(item)


    i=0
    sequence_delta_array=[]
    while(i<(len(index_delta_array)-1)):

        if((index_delta_array[i]-index_delta_array[i+1])!=-1):

            sequence_delta_array.append(i)

        i+=1


    i=0
    sequence_alpha_array=[]
    while(i<(len(index_alpha_array)-1)):

        if((index_alpha_array[i]-index_alpha_array[i+1])!=-1):

            sequence_alpha_array.append(i)

        i+=1



    start_delta_array=[]
    end_delta_array=[]
    if(sequence_delta_array==[]):
        start_delta_array.append(delta_increment*index_delta_array[0])
        end_delta_array.append(delta_increment*index_delta_array[-1])
    else:
        start=1

        for item in range(0,len(index_delta_array)):
            if(start==1):
                start_delta_array.append(delta_increment*index_delta_array[item])
                start=0
            if(item in sequence_delta_array):
                end_delta_array.append(delta_increment*index_delta_array[item])
                start=1

        end_delta_array.append(delta_increment*index_delta_array[-1])

    start_alpha_array=[]
    end_alpha_array=[]
    if(sequence_alpha_array==[]):
        start_alpha_array.append(alpha_increment*index_alpha_array[0])
        end_alpha_array.append(alpha_increment*index_alpha_array[-1])
    else:

        start=1

        for item in range(0,len(index_alpha_array)):
            if(start==1):
                start_alpha_array.append(alpha_increment*index_alpha_array[item])
                start=0
            if(item in sequence_alpha_array):
                end_alpha_array.append(alpha_increment*index_alpha_array[item])
                start=1

        end_alpha_array.append(alpha_increment*index_alpha_array[-1])

    return(index_delta_array,index_alpha_array,start_delta_array,end_delta_array,start_a
lpha_array,end_alpha_array)
```

In [3]:

```
def interval_selection(V_pi,V_0,V_a_bias,V_c_bias):

    '''
    Calculates the starting particle space
    '''

    #number of alpha and delta divitions divitons in their range
    size=50

    ###
    alpha_array=np.zeros(size)
    delta_array=np.zeros(size)

    alpha_increment=(2*np.pi)/(alpha_array.size-1)
    delta_increment=1/(delta_array.size-1)


    alpha=0
    i=0
    while i<alpha_array.size:
        alpha_array[i]=alpha
        alpha+=alpha_increment
        i+=1


    delta=0
    i=0
    while i<delta_array.size:
        delta_array[i]=delta
        delta+=delta_increment
        i+=1

    V_a=np.zeros((delta_array.size,alpha_array.size))

    i=0
    while i<delta_array.size:
        j=0
        while j<alpha_array.size:
            V_a[i,j]=2*V_0*delta_array[i]*np.sin(alpha_array[j])-V_pi*delta_array[i]*np.
cos(alpha_array[j])+V_a_bias
            j+=1
        i+=1

    V_c=np.zeros((delta_array.size,alpha_array.size))
    i=0
    while i<delta_array.size:
        j=0
        while j<alpha_array.size:
            V_c[i,j]=2*V_0*delta_array[i]*np.sin(alpha_array[j])+V_pi*delta_array[i]*np.
cos(alpha_array[j])+V_c_bias
            j+=1
        i+=1

    V_C_std=np.zeros((2,size))
    V_A_std=np.zeros((2,size))

    '''
    first row has the standard deviation of the delta rows
    second row has the standard deviation of the alpha columns
    '''

    j=0
    while(j<2):
        i=0
        if(j==0):
            while(i<size):
                V_C_std[j,i]=np.std(V_c[i,:])
                V_A_std[j,i]=np.std(V_a[i,:])
                i+=1
        else:
            while(i<size):
                V_C_std[j,i]=np.std(V_c[:,i])
                V_A_std[j,i]=np.std(V_a[:,i])
                i+=1
        j+=1


    lower_Q1_delta=np.quantile(V_A_std[0,:],0.15)
    lower_Q1_alpha=np.quantile(V_A_std[1,:],0.15)

    index_delta_array_A,index_alpha_array_A,start_delta_array_A,end_delta_array_A,start_a
lpha_array_A,end_alpha_array_A=index_array(lower_Q1_delta,lower_Q1_alpha,V_A_std,alpha_in
crement,delta_increment)
    index_delta_array_C,index_alpha_array_C,start_delta_array_C,end_delta_array_C,start_a
lpha_array_C,end_alpha_array_C=index_array(lower_Q1_delta,lower_Q1_alpha,V_C_std,alpha_in
crement,delta_increment)

    V_C_delta_mean=V_c[index_delta_array_C,:]
    V_A_delta_mean=V_a[index_delta_array_A,:]
    V_C_alpha_mean=V_c[:,index_alpha_array_C]
    V_A_alpha_mean=V_a[:,index_alpha_array_A]

    V_A_mean=(np.sum(V_A_delta_mean)+np.sum(V_A_alpha_mean))/(np.size(V_A_delta_mean[0])
*np.size(V_A_delta_mean[:,0])+(np.size(V_A_alpha_mean[0])*np.size(V_A_alpha_mean[:,0])))

    V_C_mean=(np.sum(V_C_delta_mean)+np.sum(V_C_alpha_mean))/(np.size(V_C_delta_mean[0])
*np.size(V_C_delta_mean[:,0])+(np.size(V_C_alpha_mean[0])*np.size(V_C_alpha_mean[:,0])))


    return(start_delta_array_A,start_delta_array_C,end_delta_array_A,end_delta_array_C,s
tart_alpha_array_A,start_alpha_array_C,end_alpha_array_A,end_alpha_array_C,V_A_mean,V_C_m
ean)
```

In [4]:

```
def particle_distribution(n_particles,start_delta_array,end_delta_array,start_alpha_array
,end_alpha_array):

    ''' Used to calculate random distribuitions used for the particles'''

    val_delta=1
    val_alpha=1
    j=0
    counter=0
    j_delta=n_particles/len(start_delta_array)

    while(j<n_particles):

        if(val_delta==1):
            X_delta=np.random.rand(1, int(j_delta))*(end_delta_array[0]-start_delta_arra
y[0])+start_delta_array[0]
            X_delta=X_delta[0]
            val_delta=0

        else:
            X_delta_add=np.random.rand(1, int(j_delta))*(end_delta_array[counter]-start_
delta_array[counter])+start_delta_array[counter]
            X_delta_add=X_delta_add[0]

            X_delta = np.concatenate((X_delta, X_delta_add))

        counter+=1
        j+=j_delta



    l=0
    counter=0
    l_delta=n_particles/len(start_alpha_array)
    while(l<n_particles):

        if(val_alpha==1):
            X_alpha=np.random.rand(1, int(l_delta))*(end_alpha_array[0]-start_alpha_arra
y[0])+end_alpha_array[0]
            X_alpha=X_alpha[0]
            val_alpha=0
        else:
            X_alpha_add=np.random.rand(1, int(l_delta))*(end_alpha_array[counter]-start_
alpha_array[counter])+start_alpha_array[counter]
            X_alpha_add=X_alpha_add[0]
            X_alpha = np.concatenate((X_alpha, X_alpha_add))


        counter+=1
        l+=l_delta

    return(X_delta,X_alpha)
```

In [5]:

```
def PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,start_array,iteration_m
ax,first_weight,sec_weight,third_weight,extra_sum,n_stages):

    ''' Particle swarm optimization implementation '''
    #since the particles havent started exploring their best position is their current on
e
    pbest=X
    #we apply the coordenates of each particle to the function
    pbest_obj=np.zeros(n_particles)

    for i in range(0,n_particles):
        pbest_obj[i] = f_lab(X[n_stages:(n_stages*2)][:,i], X[0:n_stages][:,i],V_A_in_ar
ray,V_C_in_array,start_array,goal_array,first_weight,sec_weight,third_weight,extra_sum,n
_stages)
    #pick the coordinates of the particle with the lowest values
    gbest = pbest[:, pbest_obj.argmin()]
    #place the value
    gbest_obj = pbest_obj.min()

    iterations=0


    while(iterations<iteration_max):
        #algorithm constants
        c1 = c2 = 0.2
        w = 0.6
        #random factor
        r = np.random.rand(2)
        #particle actualization
        #we subtract both both matrixeses --- we subtract the column of best values to al
l the entire column of X
        V = w * V + c1*r[0]*(pbest - X) + c2*r[1]*(gbest.reshape(-1,1)-X)
        X = X + V


        obj=np.zeros(n_particles)
        for i in range(0,n_particles):
            obj[i] = f_lab(X[n_stages:(n_stages*2)][:,i], X[0:n_stages][:,i],V_A_in_arra
y,V_C_in_array,start_array,goal_array,first_weight,sec_weight,third_weight,extra_sum,n_s
tages)

        #we subtract the coordenate vector with the coordenates of the lowest value obtai
ned up until now
        pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
        #update the best value obtained until now
        pbest_obj = np.array([pbest_obj, obj]).min(axis=0)
        gbest = pbest[:, pbest_obj.argmin()]
        gbest_obj = pbest_obj.min()


        iterations+=1

    return(gbest_obj,gbest)
```

In [6]:

```
def output_calc(gbest,input_pol,n_stages):

    ''' Calculation of the output polarization '''

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]
    delta_array_rad=delta_array*np.pi*2

    input_pol=np.reshape(input_pol,(3,1))

    M=wave_effect(alpha_array[n_stages-1],delta_array_rad[n_stages-1])
    i=n_stages-2
    while(i>=0):
        M=np.dot(M,wave_effect(alpha_array[i],delta_array_rad[i]))
        i-=1

    output_pol=np.dot(M,input_pol)

    output_pol=np.reshape(output_pol,(1,3))
    output_pol=output_pol[0]

    return(output_pol)
```

In [7]:

```
def rise_time(delta_array,alpha_array,V_Ain_array,V_Cin_array,n_stages):
    ''' Voltage intervals calculation '''
    V_A_measurment=np.zeros(n_stages)
    V_C_measurment=np.zeros(n_stages)

    V_pi_array=np.array([56.3,56,55.9,56.1,56.1,56,56.1,56])
    V_o_array=np.array([27.2,27,27,26.9,26.8,27.2,26.8,27.2])


    #V_A,bias e V_C,bias have no effect on the optimization process
    V_A_bias_array=np.array([-10.7,-9.3,-8.5,-10.9,-7.4,-7.6,-7.4,-7.6])
    V_C_bias_array=np.array([8.4,9.6,9.4,11.1,11.6,10.9,11.6,10.9])


    i=0
    while(i<n_stages):
        V_A_measurment[i]=V_A_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_arra
y[i],V_A_bias_array[i])-V_Ain_array[i]
        i+=1
    i=0
    while(i<n_stages):
        V_C_measurment[i]=V_C_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_arra
y[i],V_C_bias_array[i])-V_Cin_array[i]
        i+=1

    V_A_measurment_abs=abs(V_A_measurment)
    V_C_measurment_abs=abs(V_C_measurment)


    V_A_measurment_val=np.zeros(n_stages)
    V_C_measurment_val=np.zeros(n_stages)
    i=0
    while(i<n_stages):
        V_A_measurment_val[i]=V_A_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_
array[i],V_A_bias_array[i])
        i+=1
    i=0
    while(i<n_stages):
        V_C_measurment_val[i]=V_C_value(alpha_array[i],delta_array[i],V_o_array[i],V_pi_
array[i],V_C_bias_array[i])
        i+=1


    return(V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs)
```

In [8]:

```
def particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_delta_array_A,r_end_
delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_array_A,r_end_alpha
_array_C,n_stages,n_particles):

    ''' gives the matrix of new coordinatinates for the particles  '''

    X_delta=np.array([])
    X_alpha=np.array([])
    i=0

    while(i<n_stages):

        start_delta_array_A=r_start_delta_array_A[i]
        start_delta_array_C=r_start_delta_array_C[i]
        end_delta_array_A=r_end_delta_array_A[i]
        end_delta_array_C=r_end_delta_array_C[i]
        start_alpha_array_A=r_start_alpha_array_A[i]
        start_alpha_array_C=r_start_alpha_array_C[i]
        end_alpha_array_A=r_end_alpha_array_A[i]
        end_alpha_array_C=r_end_alpha_array_C[i]

        X_delta_add,X_alpha_add=particle_distribution(n_particles/2,start_delta_array_A,
end_delta_array_A,start_alpha_array_A,end_alpha_array_A)

        X_delta=np.concatenate((X_delta,X_delta_add))
        X_alpha=np.concatenate((X_alpha,X_alpha_add))

        X_delta_add,X_alpha_add=particle_distribution(n_particles/2,start_delta_array_C,
end_delta_array_C,start_alpha_array_C,end_alpha_array_C)

        X_delta=np.concatenate((X_delta,X_delta_add))
        X_alpha=np.concatenate((X_alpha,X_alpha_add))

        i+=1

    X_delta=np.reshape(X_delta,(n_stages,n_particles))
    X_alpha=np.reshape(X_alpha,(n_stages,n_particles))

    return(X_delta,X_alpha)
```

Starting H state

In [9]:

```
start = time.time()
V_pi_array=np.array([56.3,56,55.9,56.1,56.1,56,56.1,56])
V_o_array=np.array([27.2,27,27,26.9,26.8,27.2,26.8,27.2])
V_A_bias_array=np.array([-10.7,-9.3,-8.5,-10.9,-7.4,-7.6,-7.4,-7.6])
V_C_bias_array=np.array([8.4,9.6,9.4,11.1,11.6,10.9,11.6,10.9])

''' important parameters '''
goal_array=np.array([1,0,0])
input_pol=np.array([ 0.55916189 , 0.12389344 , -0.91504942]) #<--lab input SOP
#input_pol=np.array([1,0,0])
reference_point=np.sqrt(input_pol[0]**2+input_pol[1]**2+input_pol[2]**2)

# np.random.seed(seed=0)
n_particles = 80
n_stages=6
'''------------------------------------------------------'''

r_start_delta_array_A=[]
r_start_delta_array_C=[]
r_end_delta_array_A=[]
r_end_delta_array_C=[]
r_start_alpha_array_A=[]
r_start_alpha_array_C=[]
r_end_alpha_array_A=[]
```

```
r_end_alpha_array_C=[]

output_pol=np.zeros(3)
cond=1
while(cond==1):

    V_A_in_array=np.zeros(n_stages)
    V_C_in_array=np.zeros(n_stages)
    X_delta=np.array([])
    X_alpha=np.array([])
    i=0

    while(i<n_stages):

        start_delta_array_A,start_delta_array_C,end_delta_array_A,end_delta_array_C,star
t_alpha_array_A,start_alpha_array_C,end_alpha_array_A,end_alpha_array_C,V_A_in,V_C_in=int
erval_selection(V_pi_array[i],V_o_array[i],V_A_bias_array[i],V_C_bias_array[i])

        r_start_delta_array_A.append(start_delta_array_A)
        r_start_delta_array_C.append(start_delta_array_C)
        r_end_delta_array_A.append(end_delta_array_A)
        r_end_delta_array_C.append(end_delta_array_C)
        r_start_alpha_array_A.append(start_alpha_array_A)
        r_start_alpha_array_C.append(start_alpha_array_C)
        r_end_alpha_array_A.append(end_alpha_array_A)
        r_end_alpha_array_C.append(end_alpha_array_C)


        V_A_in_array[i]=V_A_in
        V_C_in_array[i]=V_C_in

        X_delta_add,X_alpha_add=particle_distribution(n_particles/2,start_delta_array_A,
end_delta_array_A,start_alpha_array_A,end_alpha_array_A)

        X_delta=np.concatenate((X_delta,X_delta_add))
        X_alpha=np.concatenate((X_alpha,X_alpha_add))

        X_delta_add,X_alpha_add=particle_distribution(n_particles/2,start_delta_array_C,
end_delta_array_C,start_alpha_array_C,end_alpha_array_C)

        X_delta=np.concatenate((X_delta,X_delta_add))
        X_alpha=np.concatenate((X_alpha,X_alpha_add))

        i+=1

    '''
    it is possible for errors to occur at reshape if the (number of particles)/len(...) i
s not a whole number,  if the number of particles is divisible by 4 there should be
    no problems.
    '''


    X_delta=np.reshape(X_delta,(n_stages,n_particles))
    X_alpha=np.reshape(X_alpha,(n_stages,n_particles))

    X = np.concatenate((X_delta, X_alpha))
    V = np.random.randn(n_stages*2, n_particles) * 0.1 #normal distribution

    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,250,2,10,10,0,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]


    output_pol=output_calc(gbest,input_pol,n_stages)

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
(delta_array,alpha_array,V_A_in_array,V_C_in_array,n_stages)

    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)
    if(abs(output_pol[0]-reference_point)<1e-4 and V_A_max<3 and V_C_max<3):
        cond=0

    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print(abs(output_pol[0]-reference_point))
    print("gbest ->",gbest_obj)
```

```
V_A diff ->  2.217009424468735
V_C diff ->  1.933878092861498
Output pol [ 1.07950315e+00 -6.82209385e-07  1.66867940e-09]
2.156053113822054e-13
gbest -> 0.44101865110590055
```

In [10]:

```
print("The values of delta for the state H are:")
print(delta_array)
print("The values of delta for the state H are:")
print(alpha_array)
print("The values of V_A for the state H are:")
print(V_A_measurment_val)
print("The values of V_C for the state H are:")
print(V_C_measurment_val)
print("____________________")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))

V_A_measurment_val_H=V_A_measurment_val
V_C_measurment_val_H=V_C_measurment_val

file1 = open("registo_alpha.txt", "w")
file2 = open("registo_delta.txt", "w")
file1.write("H\n")
file2.write("H\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
file2.close()
```

```
The values of delta for the state H are:
[0.03553893 0.03212831 0.01849859 0.02294987 0.02608799 0.03541481]
The values of delta for the state H are:
[4.57289489 4.44343606 4.4497941  4.36837577 4.4705977  4.34265005]
The values of V_A for the state H are:
[-12.33633719 -10.49447405  -9.19624858 -11.6281318   -8.40720797
  -8.67968817]
The values of V_C for the state H are:
[6.20726015 7.44935918 8.16688782 9.50341191 9.89192727 8.38694451]
____________________
Output        polarization        state
S1=1.079503, S2=-0.000001, S3=0.000000
```

H → V

In [45]:

```
output_pol=np.zeros(3)
V_A_max=100
V_C_max=100
cond=1
while(cond==1):

    ''' Important parameters '''
    goal_array=np.array([-1,0,0])
    '''____________________________________________'''
    X_delta,X_alpha=particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_del
ta_array_A,r_end_delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_ar
ray_A,r_end_alpha_array_C,n_stages,n_particles)

    X = np.concatenate((X_delta, X_alpha))

    V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal

    V_A_in_array=V_A_measurment_val_H
    V_C_in_array=V_C_measurment_val_H



    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,150,3,10,10,0,n_stages)
    ''' 2 stages '''
    # gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,20
,10,10,n_stages)

    # gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,15
,5,5)

    output_pol=output_calc(gbest,input_pol,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
(delta_array,alpha_array,V_A_in_array,V_C_in_array,n_stages)

    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)
    ''' 8 stages '''
    # if(abs(output_pol[0]+1)<1e-4 and V_A_max<7 and V_C_max<7):
    #  cond=0
    ''' 6 stages '''
    if(abs(output_pol[0]+reference_point)<1e-4 and V_A_max<7 and V_C_max<7):
        cond=0
    ''' 3 stages '''
    # if(abs(output_pol[0]+1)<1e-4 and V_A_max<13.5 and V_C_max<13.5):
    ''' 2 stages '''
    # if(abs(output_pol[0]+1)<1e-4 and V_A_max<20 and V_C_max<20):
        #cond=0
    # if(abs(output_pol[0]+1)<1e-4):

    #     cond=0

    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print("gbest ->",gbest_obj)
```

```
V_A diff ->  5.636831703526429
V_C diff ->  5.904656350371388
Output pol [-1.07950315e+00 -3.43253843e-11 -1.28050973e-12]
gbest -> 2.768077564269826
```

In [46]:

```
print("The values of delta for the state V are:")
print(delta_array)
print("The values of delta for the state V are:")
print(alpha_array)
print("The values of V_A for the state V are:")
print(V_A_measurment_val)
print("The values of V_C for the state V are:")
print(V_C_measurment_val)
print("__________________")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))
```

```
file1 = open("registo_alpha.txt", "a")
file2 = open("registo_delta.txt", "a")
file1.write("V\n")
file2.write("V\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
file2.close()

V_A_measurment_val_V=V_A_measurment_val
V_C_measurment_val_V=V_C_measurment_val
```

```
The values of delta for the state V are:
[0.0271199  0.03620306 0.06778372 0.08227556 0.07678063 0.04912901]
The values of delta for the state V are:
[1.69913154 1.71313756 1.60363449 1.59613369 1.69826476 1.74512953]
The values of V_A for the state V are:
[-9.04139855 -7.07720138 -4.71724719 -6.35805951 -2.77037627 -4.4906885 ]
The values of V_C for the state V are:
[ 9.66777893 11.2475886  12.93394266 15.40806826 15.13448169 13.05490357]
-------------------------
Output polarization state
S1=-1.079503, S2=-0.000000, S3=-0.000000
```

H → +45

In [13]:

```
output_pol=np.zeros(3)
V_A_max=100
V_C_max=100
cond=1

V_A_in_array=np.concatenate((V_A_measurment_val_H, V_A_measurment_val_V))
V_C_in_array=np.concatenate((V_C_measurment_val_H, V_C_measurment_val_V))

V_A_in_array=np.reshape(V_A_in_array,(2,n_stages))
V_C_in_array=np.reshape(V_C_in_array,(2,n_stages))


while(cond==1):

    ''' important parameters '''
    goal_array=np.array([0,1,0])
    '''_______________________________________________________'''

    X_delta,X_alpha=particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_del
ta_array_A,r_end_delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_ar
ray_A,r_end_alpha_array_C,n_stages,n_particles)

    X = np.concatenate((X_delta, X_alpha))

    V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal


    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,150,3,10,10,1,n_stages)
    ''' 2 stages '''
    # gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,25
,10,10,n_stages)


    output_pol=output_calc(gbest,input_pol,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
```

```
(delta_array,alpha_array,V_A_in_array[0],V_C_in_array[0],n_stages)

    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)
    ''' 8 stages '''
    # if(abs(output_pol[1]-1)<1e-4 and V_A_max<7 and V_C_max<7):
    #  cond=0
    ''' 6 stages '''
    if(abs(output_pol[1]-reference_point)<1e-4 and V_A_max<8.5 and V_C_max<8.5):
        cond=0
    ''' 3 stages '''
    #if(abs(output_pol[1]-1)<1e-4 and V_A_max<13.5 and V_C_max<13.5):
    ''' 2 stages '''
    # if(abs(output_pol[1]-1)<1e-4 and V_A_max<20 and V_C_max<20):

    #     cond=0
    # if(abs(output_pol[1]-1)<1e-4):
    #     cond=0

    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print("gbest ->",gbest_obj)
```

```
V_A diff ->  5.171374195964707
V_C diff ->  9.748881667883609
Output pol [-5.41878263e-13  1.07950315e+00  1.99345001e-11]
gbest -> 3.5635645245469343
V_A diff ->  3.9289279414840284
V_C diff ->  8.787767923273526
Output pol [-6.31859657e-08  1.07950315e+00  4.33961218e-06]
gbest -> 3.314340303578529
V_A diff ->  3.974366818998295
V_C diff ->  16.33893574408344
Output pol [5.63678736e-04 1.07950300e+00 3.80879512e-06]
gbest -> 2.5792539550519242
V_A diff -> 3.288080908780767
V_C diff ->  12.242239767630288
Output pol [-0.01762464  1.07529824  0.09354206]
gbest ->  1.9910118340447405
V_A diff ->  1.7895655763184717
V_C diff ->  12.288442250302849
Output pol [-2.19675675e-08  1.07882642e+00  3.82178345e-02]
gbest -> 1.7158110597101837
V_A diff ->  1.3439017315840918
V_C diff ->  7.801117025810495
Output pol [-1.69975811e-07  1.07950315e+00 -4.47598228e-07]
gbest -> 1.153163006212083
```

In [14]:

```
print("The values of delta for the state +45 are:")
print(delta_array)
print("The values of delta for the state +45 are:")
print(alpha_array)
print("The values of V_A for the state +45 are:")
print(V_A_measurment_val)
print("The values of V_C for the state +45 are:")
print(V_C_measurment_val)
print("__________________")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))

file1 = open("registo_alpha.txt", "a")
file2 = open("registo_delta.txt", "a")
file1.write("+45\n")
file2.write("+45\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
```

```
file2.close()
```

```
The values of delta for the state +45 are:
[0.07271577 0.02441987 0.08767016 0.02696741 0.04726887 0.02653106]
The values of delta for the state +45 are:
[0.46836974 0.75443875 0.49861322 0.60182176 0.52683456 0.73136461]
The values of V_A for the state +45 are:
[-12.56725969  -9.3933139  -10.54015031 -11.32567887  -8.41831067
  -7.74182905]
The values of V_C for the state +45 are:
[13.83875434 11.49957888 15.96800485 13.16845826 15.16610845 12.96973451]
--------------------------
Output polarization state
S1=-0.000000, S2=1.079503, S3=-0.000000
```

In [15]:

```
V_A_measurment_val_p45=V_A_measurment_val
V_C_measurment_val_p45=V_C_measurment_val
```

H → -45

In [16]:

```
output_pol=np.zeros(3)
V_A_max=100
V_C_max=100
cond=1

V_A_in_array=np.concatenate((V_A_measurment_val_H, V_A_measurment_val_V))
V_C_in_array=np.concatenate((V_C_measurment_val_H, V_C_measurment_val_V))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_p45))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_p45))


V_A_in_array=np.reshape(V_A_in_array,(3,n_stages))
V_C_in_array=np.reshape(V_C_in_array,(3,n_stages))

while(cond==1):

    ''' important parameters '''
    goal_array=np.array([0,-1,0])
    '''-------------------------------------------------------'''

    X_delta,X_alpha=particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_del
ta_array_A,r_end_delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_ar
ray_A,r_end_alpha_array_C,n_stages,n_particles)


    X = np.concatenate((X_delta, X_alpha))

    V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal



    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,150,2,10,10,1,n_stages)
    ''' 2 stages '''
    # gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,25
,10,10,n_stages)


    output_pol=output_calc(gbest,input_pol,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
```

```
(delta_array,alpha_array,V_A_in_array[0],V_C_in_array[0],n_stages)

    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)
    ''' 8 stages '''
    # if(abs(output_pol[1]+1)<1e-4 and V_A_max<7 and V_C_max<7):
    #  cond=0
    ''' 6 stages '''
    if(abs(output_pol[1]+reference_point)<1e-4 and V_A_max<6 and V_C_max<6):
        cond=0
    ''' 3 stages '''
    #if(abs(output_pol[1]+1)<1e-4 and V_A_max<13.5 and V_C_max<13.5):
    ''' 2 stages '''
    # if(abs(output_pol[1]+1)<1e-4 and V_A_max<20 and V_C_max<20):

    #     cond=0
    # if(abs(output_pol[1]+1)<1e-4):

    #     cond=0


    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print("gbest ->",gbest_obj)
```

```
V_A diff -> 7.082131957733514
V_C diff ->  1.7947175615231759
Output pol [ 3.55318932e-07 -1.07950315e+00 -8.29929458e-09]
gbest -> 0.8723972036418796
V_A diff ->  6.813557896703144
V_C diff ->  1.190358668650429
Output pol [-1.28279169e-08 -1.07950315e+00  1.34857648e-07]
gbest -> 0.8411295967813073
V_A diff ->  7.430371483853373
V_C diff ->  1.528337155533733
Output pol [-3.79640920e-10 -1.07950315e+00  1.23367170e-09]
gbest -> 0.8590881624686254
V_A diff -> 7.41598037529195
V_C diff ->  1.2809512021957712
Output pol [ 1.35344226e-10 -1.07950315e+00 -3.61875596e-09]
gbest -> 0.8339642975187045
V_A diff -> 8.103470117069604
V_C diff ->  2.0584157533147938
Output pol [-1.40609952e-14 -1.07950315e+00  5.23985079e-09]
gbest -> 0.9546154599952765
V_A diff -> 8.417487856903763
V_C diff ->  1.1856757809714802
Output pol [ 5.18280668e-12 -1.07950315e+00 -9.05402193e-14]
gbest -> 0.8874044285797374
V_A diff -> 6.956707594004174
V_C diff ->  0.8009955828872997
Output pol [ 1.09837797e-11 -1.07950315e+00 -1.06692327e-07]
gbest -> 0.8339014666885745
V_A diff -> 7.727685111784091
V_C diff ->  1.4296117740143295
Output pol [-4.47570023e-09 -1.07950315e+00  1.27522975e-10]
gbest -> 0.8392839542366551
V_A diff -> 5.808361556026234
V_C diff ->  1.0025601173219254
Output pol [-3.28236875e-13 -1.07950315e+00  1.03622334e-13]
gbest -> 0.8111912886008648
```

In [17]:

```
print("The values of delta for the state -45 are:")
print(delta_array)
print("The values of delta for the state -45 are:")
print(alpha_array)
print("The values of V_A for the state -45 are:")
print(V_A_measurment_val)
print("The values of V_C for the state -45 are:")
print(V_C_measurment_val)
print("--------------------")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))

file1 = open("registo_alpha.txt", "a")
file2 = open("registo_delta.txt", "a")
file1.write("-45\n")
file2.write("-45\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
file2.close()
```

```
The values of delta for the state -45 are:
[0.04565401 0.0619087  0.04457535 0.06202406 0.03888721 0.05312143]
The values of delta for the state -45 are:
[2.71854771 2.66522345 2.69171689 2.71201763 2.83929723 2.71448541]
The values of V_A for the state -45 are:
[-7.33666565 -4.68611249 -5.20944241 -6.34682726 -4.69681167 -3.69536026]
The values of V_C for the state -45 are:
[ 7.07587617  8.0520797   8.20288804  9.3263569  10.13788765  9.38950463]
--------------------
Output polarization state
S1=-0.000000, S2=-1.079503, S3=0.000000
```

In [18]:

```
V_A_measurment_val_n45=V_A_measurment_val
V_C_measurment_val_n45=V_C_measurment_val
```

H → ↷

In [37]:

```
output_pol=np.zeros(3)
V_A_max=100
V_C_max=100
cond=1

V_A_in_array=np.concatenate((V_A_measurment_val_H, V_A_measurment_val_V))
V_C_in_array=np.concatenate((V_C_measurment_val_H, V_C_measurment_val_V))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_p45))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_p45))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_n45))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_n45))


V_A_in_array=np.reshape(V_A_in_array,(4,n_stages))
V_C_in_array=np.reshape(V_C_in_array,(4,n_stages))


while(cond==1):

    ''' important parameters '''
    goal_array=np.array([0,0,1])
    '''-----------------------------------------------'''

    X_delta,X_alpha=particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_del
ta_array_A,r_end_delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_ar
ray_A,r_end_alpha_array_C,n_stages,n_particles)


    X = np.concatenate((X_delta, X_alpha))
    V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal



    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,150,2,10,10,1,n_stages)
    ''' 2 stages '''
    # gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,25
,10,10,n_stages)


    output_pol=output_calc(gbest,input_pol,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
(delta_array,alpha_array,V_A_in_array[0],V_C_in_array[0],n_stages)



    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)

    V_A_measurment_val_ex,V_C_measurment_val_ex,V_A_measurment_abs_ex,V_C_measurment_abs_
ex=rise_time(delta_array,alpha_array,V_A_measurment_val_p45,V_C_measurment_val_p45,n_sta
ges)

    V_A_max_ex=np.max(V_A_measurment_abs_ex)
    V_C_max_ex=np.max(V_C_measurment_abs_ex)
    # if(abs(output_pol[2]-1)<1e-4 and V_A_max<7 and V_C_max<7):
    #     cond=0
    if(abs(output_pol[2]-reference_point)<1e-4 and V_A_max<8.06 and V_C_max<8.06):
        cond=0
    #if(abs(output_pol[2]-1)<1e-4 and V_A_max<13.5 and V_C_max<13.5):
    # if(abs(output_pol[2]-1)<1e-4 and V_A_max<20 and V_C_max<20):


    #     cond=0
    # if(abs(output_pol[2]-1)<1e-4):
    #     cond=0
    #print(V_A_max_ex)
    #print(V_C_max_ex)
    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print("gbest ->",gbest_obj)
```

```
V_A diff ->  9.986485520317038
V_C diff ->  8.170318625333177
Output pol [-9.36507210e-10  6.34080134e-12  1.07950315e+00]
gbest -> 0.967821215473l6
V_A diff ->  7.880138246638725
V_C diff ->  8.045667396345204
Output pol [-1.51480448e-08 -1.38853527e-09  1.07950315e+00]
gbest -> 0.9286447434531078
```

In [40]:

```
print("The values of delta for the state +circ are:")
print(delta_array)
print("The values of delta for the state +circ are:")
print(alpha_array)
print("The values of V_A for the state +circ are:")
print(V_A_measurment_val)
print("The values of V_C for the state +circ are:")
print(V_C_measurment_val)
print("_________________")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))

file1 = open("registo_alpha.txt", "a")
```

```
file2 = open("registo_delta.txt", "a")
file1.write("circ+\n")
file2.write("circ+\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
file2.close()
```

```
The values of delta for the state +circ are:
[0.10271409 0.09374465 0.10608228 0.07574369 0.10519349 0.10744969]
The values of delta for the state +circ are:
[1.69159823 1.79984141 1.59613623 1.81840365 1.74014891 1.62152802]
The values of V_A for the state +circ are:
[-4.45619894 -3.17806324 -2.6231463  -5.90785045 -0.84765194 -1.45712623]
The values of V_C for the state +circ are:
[13.25004946 13.33807245 14.97635467 14.00930921 16.1630696  16.43261191]
-------------------------
Output polarization state
S1=-0.000000, S2=-0.000000, S3=1.079503
```

In [41]:

```
V_A_measurment_val_pcirc=V_A_measurment_val
V_C_measurment_val_pcirc=V_C_measurment_val
```

H → ↶

In [22]:

```
output_pol=np.zeros(3)
V_A_max=100
V_C_max=100
cond=1

V_A_in_array=np.concatenate((V_A_measurment_val_H, V_A_measurment_val_V))
V_C_in_array=np.concatenate((V_C_measurment_val_H, V_C_measurment_val_V))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_p45))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_p45))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_n45))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_n45))

V_A_in_array=np.concatenate((V_A_in_array, V_A_measurment_val_pcirc))
V_C_in_array=np.concatenate((V_C_in_array, V_C_measurment_val_pcirc))


V_A_in_array=np.reshape(V_A_in_array,(5,n_stages))
V_C_in_array=np.reshape(V_C_in_array,(5,n_stages))

while(cond==1):

    ''' important parameters '''
    goal_array=np.array([0,0,-1])
    '''------------------------------------------------------------'''

    X_delta,X_alpha=particle_space(r_start_delta_array_A,r_start_delta_array_C,r_end_del
ta_array_A,r_end_delta_array_C,r_start_alpha_array_A,r_start_alpha_array_C,r_end_alpha_ar
ray_A,r_end_alpha_array_C,n_stages,n_particles)


    X = np.concatenate((X_delta, X_alpha))

    V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal


    gbest_obj,gbest=PSO_lab(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,input_p
ol,150,2,10,10,1,n_stages)
    ''' 2 stages '''
    #gbest_obj,gbest=PSO(X,V,n_particles,goal_array,V_A_in_array,V_C_in_array,0,1,150,25,
10,10,n_stages)


    output_pol=output_calc(gbest,input_pol,n_stages)

    delta_array=gbest[0:n_stages]
    alpha_array=gbest[n_stages:(n_stages*2)]

    V_A_measurment_val,V_C_measurment_val,V_A_measurment_abs,V_C_measurment_abs=rise_time
(delta_array,alpha_array,V_A_in_array[0],V_C_in_array[0],n_stages)
    #  V_A_measurment_val_ex,V_C_measurment_val_ex,V_A_measurment_abs_ex,V_C_measurment_ab
s_ex=rise_time(delta_array,alpha_array,V_A_measurment_val_n45,V_C_measurment_val_n45,n_st
ages)

    V_A_max=np.max(V_A_measurment_abs)
    V_C_max=np.max(V_C_measurment_abs)

    # V_A_max_ex=np.max(V_A_measurment_abs_ex)
    # V_C_max_ex=np.max(V_C_measurment_abs_ex)

    # n45 para circular negativa
    ''' 8 stages '''
    # if(abs(output_pol[2]+1)<1e-4 and V_A_max<7.8 and V_C_max<7.8 and V_A_max_ex<7.8 and
V_C_max_ex<7.8):
    #     cond=0
    # if(abs(output_pol[2]+1)<1e-4 and V_A_max<7 and V_C_max<7):
    #     cond=0
    ''' 6 stages '''
    # if(abs(output_pol[2]+reference_point)<1e-4 and V_A_max<8.8 and V_C_max<8.8):
    #     cond=0
    if(abs(output_pol[2]+reference_point)<1e-4 and V_A_max<5 and V_C_max<5 ):
        cond=0
    ''' 3 stages '''
    #if(abs(output_pol[2]+1)<1e-4 and V_A_max<13.5 and V_C_max<13.5 and V_A_max_ex<13.5 a
nd V_C_max_ex<13.5):
    ''' 2 stages '''
    # if(abs(output_pol[2]+1)<1e-4 and V_A_max<20 and V_C_max<20 and V_A_max_ex<20 and V_
C_max_ex<20):

    #     cond=0
    # if(abs(output_pol[2]+1)<1e-4):

    #     cond=0
    # print(V_A_max_ex)
    # print(V_C_max_ex)
    print("V_A diff -> ",V_A_max)
    print("V_C diff -> ",V_C_max)
    print("Output pol", output_pol)
    print("gbest ->",gbest_obj)
```

```
V_A diff ->  5.457539337790931
V_C diff ->  4.860615021301132
Output pol [ 8.23867925e-09 -1.16664824e-06 -1.07950315e+00]
gbest -> 0.4631741553752556
V_A diff ->  4.836078664918034
V_C diff ->  4.010929073354045
Output pol [-1.04984313e-12  6.82458815e-12 -1.07950315e+00]
gbest -> 0.450741619474477
```

In [23]:

```
print("The values of delta for the state -circ are:")
print(delta_array)
print("The values of delta for the state -circ are:")
print(alpha_array)
print("The values of V_A for the state -circ are:")
print(V_A_measurment_val)
print("The values of V_C for the state -circ are:")
print(V_C_measurment_val)
print("------------------------")
print("Output polarization state")
print("S1=%f, S2=%f, S3=%f"%(output_pol[0],output_pol[1],output_pol[2]))

file1 = open("registo_alpha.txt", "a")
file2 = open("registo_delta.txt", "a")
file1.write("circ-\n")
file2.write("circ-\n")
str_alpha=repr(alpha_array)
file1.write(str_alpha+"\n")
str_delta=repr(delta_array)
file2.write(str_delta+"\n")
file1.close()
file2.close()
```

```
The values of delta for the state -circ are:
[ 0.00702162  0.03848801  0.00081359  0.00945274  0.05780356 -0.02428879]
The values of delta for the state -circ are:
[1.80681051 1.66760677 1.74954598 1.77987312 1.83125736 1.71214007]
The values of V_A for the state -circ are:
[-10.23617672  -7.02304666  -8.44867968 -10.2924506   -3.57112931
  -9.09974571]
The values of V_C for the state -circ are:
[ 8.67895    11.46028825  9.43514774 11.48741521 13.7586701   9.78347892]
------------------------
Output polarization state
S1=-0.000000, S2=0.000000, S3=-1.079503
```

In [24]:

```
V_A_measurment_val_ncirc=V_A_measurment_val
V_C_measurment_val_ncirc=V_C_measurment_val
```

V → +45

In [50]:

```
V_A_Calc=V_A_measurment_val_V-V_A_measurment_val_p45
V_C_Calc=V_C_measurment_val_V-V_C_measurment_val_p45

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)


print("TRansition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
TRansition results:
Max voltage interval 5.82290311441572
```

V → -45

In [49]:

```
V_A_Calc=V_A_measurment_val_V-V_A_measurment_val_n45
V_C_Calc=V_C_measurment_val_V-V_C_measurment_val_n45

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))
```

```
else:
    print("Max voltage interva",((np.max(V_C_Calc)*1000)/5876),"ns",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interva 1.0350087403525998 ns 6.0817113583118765
```

V → circ+

In [48]:

```
V_A_Calc=V_A_measurment_val_V-V_A_measurment_val_pcirc
V_C_Calc=V_C_measurment_val_V-V_C_measurment_val_pcirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 4.585199607484955
```

V → circ-

In [47]:

```
V_A_Calc=V_A_measurment_val_V-V_A_measurment_val_ncirc
V_C_Calc=V_C_measurment_val_V-V_C_measurment_val_ncirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 4.60905721190807
```

+45 → -45

In [29]:

```
V_A_Calc=V_A_measurment_val_p45-V_A_measurment_val_n45
V_C_Calc=V_C_measurment_val_p45-V_C_measurment_val_n45

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 7.765116811416753
```

+45 → circ+

In [42]:

```
V_A_Calc=V_A_measurment_val_p45-V_A_measurment_val_pcirc
V_C_Calc=V_C_measurment_val_p45-V_C_measurment_val_pcirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 8.111060747999094
```

+45 → circ-

In [31]:

```
V_A_Calc=V_A_measurment_val_p45-V_A_measurment_val_ncirc
V_C_Calc=V_C_measurment_val_p45-V_C_measurment_val_ncirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 6.532857106583672
```

-45 → circ+

In [44]:

```
V_A_Calc=V_A_measurment_val_n45-V_A_measurment_val_pcirc
V_C_Calc=V_C_measurment_val_n45-V_C_measurment_val_pcirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 7.043107279023278
```

-45 → circ-

In [33]:

```
V_A_Calc=V_A_measurment_val_n45-V_A_measurment_val_ncirc
V_C_Calc=V_C_measurment_val_n45-V_C_measurment_val_ncirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 5.404385453779486
```

circ+ → circ-

In [43]:

```
V_A_Calc=V_A_measurment_val_pcirc-V_A_measurment_val_ncirc
V_C_Calc=V_C_measurment_val_pcirc-V_C_measurment_val_ncirc

V_A_Calc=abs(V_A_Calc)
V_C_Calc=abs(V_C_Calc)

print("Transition results:")
if(np.max(V_A_Calc)>np.max(V_C_Calc)):
    print("Max voltage interval",np.max(V_A_Calc))

else:
    print("Max voltage interval",np.max(V_C_Calc))
```

```
Transition results:
Max voltage interval 7.64261948168911
```

In [35]:

```
end = time.time()
print(end - start)
```

```
127.95106482505798
```

In [36]:

```
''' Store the seed if it's good '''
# seed_4stages6_t_7_76=Seed
# %store seed_4stages6_t_7_76
```

Appendix B

PSO adjustments code

In [29]:

```
def f(measured_array,goal_array,current_volt,last_volt,weight_1,weight_2,n_stages):

    first_sum=0
    second_sum=0

    for i in range(0,3):
        #first_sum+=abs(current_array[i]-goal_array[i])
        first_sum+=abs(measured_array[i]-goal_array[i])*weight_1
        # print(abs(measured_array[i]-goal_array[i]))

    # print("////")

    for i in range(0,n_stages*2):
        #first_sum+=abs(current_array[i]-goal_array[i])
        second_sum+=abs(current_volt[i]-last_volt[i])*weight_2

        # print(abs(current_volt[i]-last_volt[i]))

    z=(first_sum/3)+(second_sum/(n_stages*2))

    #print(first_sum,'~~~',second_sum,'~~',third_sum,'~~>',z)
    return z
```

In [12]:

```
def make_measurement():
    '''
    Setup:
    (Differential mode)
    S1 -> Dev1/ai0
    S2 -> Dev1/ai1
    S3 -> Dev1/ai2
    DOP or Power -> Dev1/ai3

    Ground connected to AI4,AI5,AI6,AI7

    (Rising Edge Mode)
    Trigger output -> Dev1/port0/line0

    '''

    r_analog_input=["Dev1/ai0","Dev1/ai1","Dev1/ai2","Dev1/ai3"]
    o_trigger="Dev1/port0/line0"

    task_1=nidaqmx.Task()
    task_2=nidaqmx.Task()
    task_3=nidaqmx.Task()
    task_4=nidaqmx.Task()
    task_5=nidaqmx.Task()

    task_1.do_channels.add_do_chan("Dev1/port0/line0")
    task_2.ai_channels.add_ai_voltage_chan(r_analog_input[0],terminal_config=TerminalCon
figuration.RSE)
    task_3.ai_channels.add_ai_voltage_chan(r_analog_input[1],terminal_config=TerminalCon
figuration.RSE)
    task_4.ai_channels.add_ai_voltage_chan(r_analog_input[2],terminal_config=TerminalCon
figuration.RSE)
    task_5.ai_channels.add_ai_voltage_chan(r_analog_input[3],terminal_config=TerminalCon
figuration.RSE)

    # task_1.write(True)

    # time.sleep(0.25)

    r_multiple_read=np.zeros((3,8))
    r_measured_values=np.zeros(3)
```

```
    r_multiple_read[0]=task_2.read(number_of_samples_per_channel=8)
    r_multiple_read[1]=task_3.read(number_of_samples_per_channel=8)
    r_multiple_read[2]=task_4.read(number_of_samples_per_channel=8)
    #r_multiple_read[3]=task_5.read(number_of_samples_per_channel=8)

    r_measured_values[0]=np.mean(r_multiple_read[0,:])
    r_measured_values[1]=np.mean(r_multiple_read[1,:])
    r_measured_values[2]=np.mean(r_multiple_read[2,:])
    #r_measured_values[3]=np.mean(r_multiple_read[3,:])


    # task_1.write(False)

    r_output_values=r_measured_values/2.5

    task_1.stop()
    task_1.close()
    task_2.stop()
    task_2.close()
    task_3.stop()
    task_3.close()
    task_4.stop()
    task_4.close()
    task_5.stop()
    task_5.close()

    return(r_output_values)
```

In [ ]:

```
import time
# %matplotlib qt5
import numpy as np
import nidaqmx
from nidaqmx.constants import (
TerminalConfiguration)
import matplotlib.pyplot as plt



n_stages=6
n_particles=6
voltage_range=10 #deve ser necessário 20

goal=np.array([1,0,0])

for i in range(0,3):
    if(goal[i]==1):
        pol_index=i
        signal=1
    if(goal[i]==-1):
        pol_index=i
        signal=-1

# print(pol_index,signal)

V_A_array=np.array([-10.35198091 , -9.11390144  ,-7.22656287 ,-10.3402543 ,  -7.32236055
  ,-7.86467429] )
V_C_array=np.array([ 6.13810597 , 5.11179801 , 3.02560257, 7.31351766 ,11.19369122 ,12.2
8584249])

V_A_bias_array=np.array([-10.7,-9.3,-8.5,-10.9,-7.4,-7.6])
V_C_bias_array=np.array([8.4,9.6,9.4,11.1,11.6,10.9])

V_bias_array=np.concatenate((V_A_bias_array, V_C_bias_array))

V_A_array_m_bias=V_A_array-V_A_bias_array
V_C_array_m_bias=V_C_array-V_C_bias_array
```

```
V_A_bias_array_up=np.zeros(n_stages)
V_A_bias_array_low=np.zeros(n_stages)

V_C_bias_array_up=np.zeros(n_stages)
V_C_bias_array_low=np.zeros(n_stages)

for i in range(0,n_stages):

    V_A_bias_array_up[i]=V_A_bias_array[i]+voltage_range/2
    V_A_bias_array_low[i]=V_A_bias_array[i]-voltage_range/2

    V_C_bias_array_up[i]=V_C_bias_array[i]+voltage_range/2
    V_C_bias_array_low[i]=V_C_bias_array[i]-voltage_range/2

V_A_bias_mod=np.zeros((n_stages,n_particles))
V_C_bias_mod=np.zeros((n_stages,n_particles))

for j in range(0,n_particles):

    for i in range(0,n_stages):
        value_A=np.random.rand(1)*(V_A_bias_array_up[i]-V_A_bias_array_low[i])+V_A_bias_
array_low[i]
        value_C=np.random.rand(1)*(V_C_bias_array_up[i]-V_C_bias_array_low[i])+V_C_bias_
array_low[i]

        V_A_bias_mod[i,j]=value_A[0]
        V_C_bias_mod[i,j]=value_C[0]

X = np.concatenate((V_A_bias_mod, V_C_bias_mod))
V = np.random.randn(n_stages*2, n_particles) * 0.1 #distribuição normal

run=1
run_part=1
next_run=0

while(run):

    pbest=X
    #aplicamos os valores das coordadenadas de cada partícula à função
    pbest_obj=np.zeros(n_particles)


    file = open("record.txt", "w")
    str_pbest=repr(pbest)
    str_pbest_obj=repr(pbest_obj)
    file.write("pbest = " + str_pbest + "\n")
    file.write("#\n")
    file.write("pbest_obj = " + str_pbest_obj + "\n")
    file.write("@\n")
    file.close()


    particle_number=1

    while(particle_number<=n_particles and run_part):

        i=particle_number-1
        print("Particle number:",particle_number)
        print("Voltage values \nVA1",V_A_array_m_bias[0]+X[0,i],"\nVC1",V_C_array_m_bias
[0]+X[6,i],"\nVA2",V_A_array_m_bias[1]+X[1,i],"\nVC2",V_C_array_m_bias[1]+X[7,i],"\nVA3"
,V_A_array_m_bias[2]+X[2,i],"\nVC3",V_C_array_m_bias[2]+X[8,i],
            "\nVA4",V_A_array_m_bias[3]+X[3,i],"\nVC4",V_C_array_m_bias[3]+X[9,i],"\nVA5
",V_A_array_m_bias[4]+X[4,i],"\nVC5",V_C_array_m_bias[4]+X[10,i],"\nVA6",V_A_array_m_bia
s[5]+X[5,i],"\nVC6",V_C_array_m_bias[5]+X[11,i])

        user_input = input("\nMake measurement, type stop to end:\n")

        if user_input=="stop":
            run_part=0
            run=0

        if(run_part==1):
            r_measurment=make_measurement()

            print("The SOP measurement for particle:",particle_number,"-->",r_measurment
,"\n")

            max_value=np.sqrt(r_measurment[0]**2+r_measurment[1]**2+r_measurment[2]**2)

            lab_goal=np.zeros(3)

            lab_goal[pol_index]=max_value*signal

            pbest_obj[particle_number-1]=f(r_measurment,lab_goal,X[:,i],V_bias_array,5,0
.5,n_stages)

            gbest = pbest[:, pbest_obj.argmin()]
            #coloca o valor que a função objetivo deu para essas coordenadas
            gbest_obj = pbest_obj.min()

            if(particle_number==n_particles):
                next_run=1

                print("pbest_obj=",pbest_obj)
                print("gbest=",gbest)

            particle_number+=1

    file = open("record.txt", "w")
    str_pbest=repr(pbest)
    str_pbest_obj=repr(pbest_obj)
    file.write("pbest = " + str_pbest + "\n")
    file.write("#\n")
    file.write("pbest_obj = " + str_pbest_obj + "\n")
    file.write("@\n")
    file.close()

    iteration=0
    while(next_run):

        c1 = c2 = 0.2
        #incial era 0.8
        w = 0.6
        #fator aleatório
        r = np.random.rand(2)
        #atualização das partículas
        #substrainos as duas matrizes --- subtraimos a coluna de melhores valores a toda
a coluna de X
        V = w * V + c1*r[0]*(pbest - X) + c2*r[1]*(gbest.reshape(-1,1)-X)
        X = X + V

        obj=np.zeros(n_particles)

        particle_number=1
        run_part=1

        while(particle_number<=n_particles and run_part==1):

            print("Iteration number:",iteration)
            print("Particle number",particle_number)
            i=particle_number-1
            print("Voltage values \nVA1",V_A_array_m_bias[0]+X[0,i],"\nVC1",V_C_array_m_
bias[0]+X[6,i],"\nVA2",V_A_array_m_bias[1]+X[1,i],"\nVC2",V_C_array_m_bias[1]+X[7,i],"\n
VA3",V_A_array_m_bias[2]+X[2,i],"\nVC3",V_C_array_m_bias[2]+X[8,i],
            "\nVA4",V_A_array_m_bias[3]+X[3,i],"\nVC4",V_C_array_m_bias[3]+X[9,i],"\nVA5
",V_A_array_m_bias[4]+X[4,i],"\nVC5",V_C_array_m_bias[4]+X[10,i],"\nVA6",V_A_array_m_bia
s[5]+X[5,i],"\nVC6",V_C_array_m_bias[5]+X[11,i])

            user_input = input("\nMake measurement, type stop to end:\n ")

            if user_input=="stop":
                run_part=0
                next_run=0
                run=0

            if(run_part==1):

                r_measurment=make_measurement()

                print("The SOP measurement for particle:",particle_number,"-->",r_measur
ment,"\n")

                file = open("record.txt", "a")
                str_r_measurment=repr(r_measurment)
                str_particle_number=repr(particle_number)
                file.write("medição partícula "+ str_particle_number +"=" + str_r_measur
ment + "\n")
                file.write("$\n")
                file.close()

                max_value=np.sqrt(r_measurment[0]**2+r_measurment[1]**2+r_measurment[2]*
*2)

                lab_goal=np.zeros(3)

                lab_goal[pol_index]=max_value*signal

                obj[particle_number-1]=f(r_measurment,lab_goal,X[:,i],V_bias_array,5,0.5
,n_stages)

                #if(particle_number==n_particles):



                particle_number+=1

        if(run_part==1):

            pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
            #atualizar o melhor valor atingido pela particula
            pbest_obj = np.array([pbest_obj, obj]).min(axis=0)
            gbest = pbest[:, pbest_obj.argmin()]
            gbest_obj = pbest_obj.min()

        print("pbest_obj=",pbest_obj)
        print("gbest=",gbest)

        file = open("record.txt", "a")
        str_pbest=repr(pbest)
        str_pbest_obj=repr(pbest_obj)
        file.write("pbest = " + str_pbest + "\n")
        file.write("#\n")
        file.write("pbest_obj = " + str_pbest_obj + "\n")
        file.write("@\n")
        file.close()

        iteration+=1
```