Quantum and Classical Channels

Amir Mohammad Moghaddam 9823144

June 2024

Introduction

This report is divided into two main sections: the Theory section and the Results section. The Theory section provides an overview of the foundational concepts and methodologies, while the Results section presents the outcomes of the simulations described in the Theory section.

In the Theory section, we begin by examining quantum channels, with a particular focus on photon states used for quantum communication. Although we will briefly discuss coherent state quantum communication, it is not our primary focus. Instead, we will shift our attention to Orbital Angular Momentum (OAM) quantum channels, detailing the methods used to modulate and detect these states. Additionally, we will address the challenges associated with these channels.

Following the Theory section, the Results section will outline the simulations conducted and the specific objectives we aimed to achieve. We will then analyze the results to determine whether the objectives were met.

Contents

Chapter 1

Theory

1.1 Channels

In general, our objective is to transmit data from one point to another. This process involves using a medium to carry the data, and the physical degrees of freedom (DOFs) inherent to this medium will dictate how we modulate and detect information at both ends. This is where we seek to revolutionize traditional information and communication paradigms by leveraging the unique advantages provided by the quantum realm. Within this realm, we can utilize any degree of freedom for communication. As we will explore, certain DOFs offer specific advantages over others in various aspects of communication.

1.1.1 General Quantum Channel

As discussed in previous reports, quantum channels are defined as trace-preserving completely positive linear maps. These maps, denoted as Φ, are represented as $\Phi : \rho \to \rho'$, where ρ and ρ' are the quantum states that are sent and received, respectively, in a quantum communication system, such as a photon. There are various ways to describe a photon depending on the chosen degree of freedom, and this choice influences the states that can be used and the corresponding quantum channels. Now we will look at the photon states from the article [3]

Photon degrees of freedom (DOFs) commonly utilized in communication include photon frequency and photon number, but these are not the only available DOFs. As solutions to Maxwell's equations, photons exhibit a vectorial nature characterized by polarization, which describes the oscillation of electric and magnetic fields in space and time during propagation. Photons are transverse entities, meaning their electric and magnetic fields lie in a plane orthogonal to the propagation direction. Consequently, only two independent vectors are required to define the polarization state of a single photon, such as $|L\rangle_\pi$ (left-handed) and $|R\rangle_\pi$ (right-handed) in the circular polarization basis. Furthermore, a photon's spatial mode, which satisfies the wave equation, is also quantized. Depending on the chosen coordinates—considering symmetry and boundary conditions—the quantization parameters may be continuous or discrete, and they can be bounded or unbounded.

For example if we use the Cartesian coordinates with the boundaries set at infinity the general photon state is given by:

$$|\psi\rangle = \int d\omega \int dk_x \int dk_y \sum_{N,\pi} c^{N,\pi}(\omega, k_x, k_y)|N, \omega, \pi, k_x, k_y\rangle \tag{1.1}$$

where $c^{N,\pi}(\omega, k_x, k_y) = C^{N,\pi}(\omega, k_x, k_y)e^{iz\sqrt{\frac{\omega^2}{c^2}-k_x^2-k_y^2}}$ where $C^{N,\pi}(\omega, k_x, k_y)$ is the expansion coefficient of the expansion to plane wave modes. Here by $|N, \omega, \pi, k_x, k_y\rangle$ we mean N photons at frequency ω in a polarization state of π, and transverse wave-vector of k_x and k_y. We also have to know that this is not the only way to describe the state of photons and the description changes by the modes we use to expand our general wave as we will see.

1.1.2 Orbital Angular Momentum(OAM) quantum states

In the context of Maxwell's equations, cylindrical symmetry is frequently employed, particularly in optical laboratory devices such as optical fibers, lenses, and mirrors. This symmetry makes it advantageous to use power-normalized and complete modes that also exhibit cylindrical symmetry, such as the Laguerre–Gauss modes, denoted as $LG_{p,l}(\mathbf{r})$. The integer indices p and l dictate the photon probability and phase distributions along the radial ρ and azimuthal ϕ coordinates, respectively. The photon's state remains invariant under a 2π rotation, necessitating that the azimuthal index l be an integer. Consequently, the position representation of the state is given by $exp(il\phi)$. Photons with this helical wavefront, represented by $exp(il\phi)$, possess $l\hbar$ units of angular momentum per photon in the direction of propagation.

The photon state in cylindrical coordinates (ρ, ϕ, z) is given by :

$$|\psi\rangle = \int d\omega \sum_{N,\pi,p,l} c_{p,l}^{N,\pi}(\omega)|N, \omega, \pi, p, l\rangle \tag{1.2}$$

In the equation 1.2 exactly like the in the state 1.1 we can obtain the the time and position representation as :

$$\begin{aligned} \langle t|\omega\rangle &= e^{-i\omega t} \\ \langle \mathbf{r}|p, l\rangle &= LG(p, l) \end{aligned} \tag{1.3}$$

Photons in the aforementioned superposition state are known as structured photons, and they can be categorized into several different classes. Understanding how to generate, manipulate, and detect these photonic degrees of freedom—including photon number, frequency, polarization, and spatial modes—enables us to encode more information onto a single information carrier (photon). Besides increasing the communication channel capacity, utilizing these degrees of freedom enhances communication security.

1.2 OAM Quantum Communication

n this section, we introduce a communication channel based on the orbital angular momentum (OAM) feature of a single photon. We emphasize the use of a single photon to avoid the complications associated with the coherent state of a laser, such as photon counting errors and dark counts. Specifically, we will consider the OAM states of the photon to transmit our data, using the general model depicted in Figure 1.1. As discussed in the previous report, the OAM states illustrated in the figure correspond to the mapped states of a four-dimensional qudit, with the actual OAM states utilized being those with $l = \pm 3, \pm 1$. In the subsequent subsections, we will explore a method for modulating digital information using these states, as well as a technique for detecting these states.

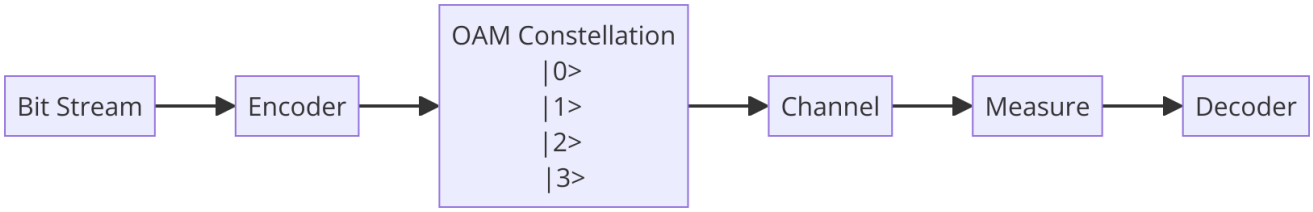

Figure 1.1: Digital Transmission using a OAM based channel

1.2.1 Modulation and Detection

In this subsection, we discuss the implementation of a quantum OAM channel as used in the article by Mirhosseini et al. (2015) [2]. This channel facilitates various quantum key distribution (QKD) protocols using OAM-based photon states. Our primary focus is on the methods employed for modulating and detecting OAM states. The

model, illustrated in Figure 1.2, broadly consists of a laser and attenuator setup, along with an acousto-optic modulator (AOM), which produces the desired photon pulse. This pulse is directed to a spatial light modulator (SLM) where the OAM state is encoded via a computer-generated hologram. The photon is then transmitted through the communication channel, which can be either an optical fiber or free space. After transmission, an OAM mode sorter and a beam splitter are used to detect the two different bases employed in the protocol (OAM and angular momentum). The choice of measurement basis is determined by the protocol requirements, enabling the execution of a QKD protocol. However, within the context of this section, we focus on the use of a single basis for channel operation. For this channel, SLMs are utilized for modulation and a mode sorter for detection. Detailed descriptions of these methods and others can be found in the article by [4].

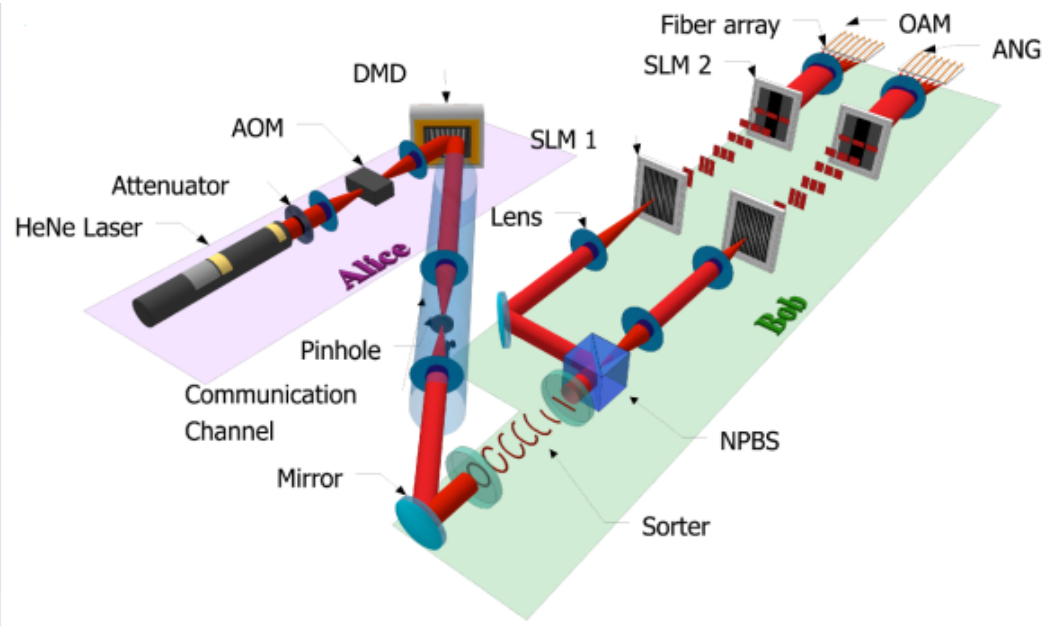

Figure 1.2: The experimental setup. Alice prepares the modes by carving out pulses from a highly attenuated He-Ne laser using an AOM. The spatial mode information is impressed on these pulses with a DMD. Bob's mode sorter and fan-out elements map the OAM modes and the ANG modes into separated spots that are collected by an array of fibers.From article [2]

1.2.2 Challenges in each channel

As mentioned earlier, the channel used to send photons can be either an optical fiber or a free-space link. However, each of these channels presents specific challenges. In this subsection, we will briefly discuss some of these challenges and list them for potential further implementation in the next steps of our project. This section draws on insights from the article by Zhou et al. (2021) [4].

For free-space links, atmospheric turbulence is a major issue that can cause random phase distortions in the transverse beam profile. These distortions are time-variant, potentially inducing dynamic intermodal power coupling. Consequently, this could lead to modal coupling and crosstalk in both classical and quantum communication links. Additionally, in an OAM-based communication link, the receiver must be able to distinguish the transmitted spatial modes. When the receiver aperture size is limited, or there is misalignment between the transmitter and the receiver, significant power leakage into undesired modes can occur, potentially causing the receiver to fail to correctly identify the transmitted modes. Moreover, the divergence of higher-order OAM beams in free space is more pronounced than that of lower-order beams. For a receiver with a limited aperture size, capturing the entire higher-order OAM beam is challenging, leading to power loss. Beam truncation due to a circular aperture can

also induce modal power coupling into different LG modes, resulting in inter-channel crosstalk in LG-mode-based communication links.

For fiber-based links, quantum encoding systems can also utilize OAM modes. However, the fiber channel differs from free space in several key aspects. Since the beam is guided within the fiber, it does not diverge. Nevertheless, the fiber can cause cross-modal power coupling within the same mode group or between different mode groups due to temperature gradients, various inhomogeneities, bends, and other imperfections. Typically, intramodal-group power coupling is relatively stronger. A summary of these challenges is illustrated in Figure 1.3.

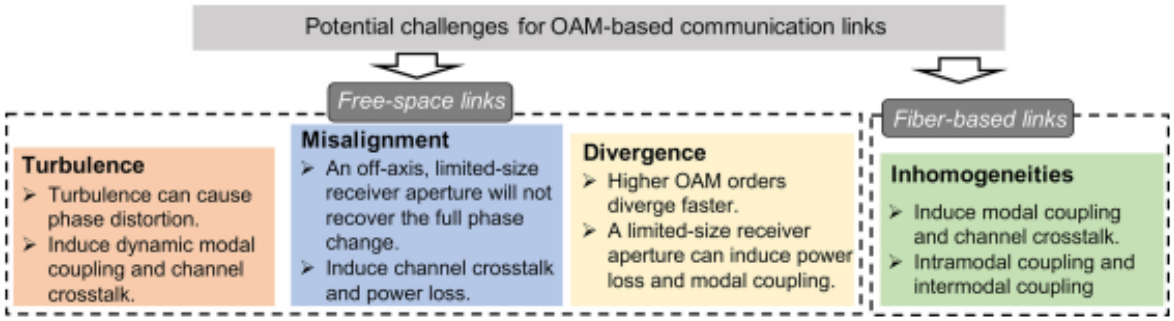

Figure 1.3: Potential challenges for OAM-based communications systems.article [4]

1.3 Simulation Goals

In this section, we will first a discussion of the objectives we aim to achieve through these simulations, followed by an examination of the models used for our simulations. The final subsection will focus on analyzing the required number of simulation runs. Here, we will explore the theoretical basis for the protocol and determine a safe and effective number of iterations for our simulation.

1.3.1 Objectives

In the BB84 protocol, Alice and Bob must compare their measurement bases to ensure the integrity of their key. Given our primary objective of exploring the use of quantum OAM as a communication tool, this aspect of the protocol provides an excellent opportunity to analyze both classical and quantum channels. To this end, we have incorporated two distinct models into our simulation code. The objective was to compare the parameters of these two models to identify any similarities or differences. Furthermore, after this comparative analysis, we executed two versions of the BB84 protocol (using 2-dimensional and 4-dimensional states) with these channels to observe the differences in the simulation outcomes.

1.3.2 Models

As we mentioned earlier we introduced two channel models into our simulation code, a Classical and a Quantum channel. First lets explore the classical channel:

```
def encode_MPSK(bits, M):
    # Calculate the number of bits per symbol
    bits_per_symbol = int(np.log2(M))
    # Ensure the bit stream length is a multiple of bits_per_symbol
    while len(bits) % bits_per_symbol != 0:
        bits = np.append(bits, 0)

    # Create the mapping from bit groups to symbols
    mapping = {}
    for i in range(M):
```

```
        # Calculate the phase angle for the i-th symbol
        phase = 2 * np.pi * i / M
        symbol = np.exp(1j * phase)  # Complex symbol on the unit circle
        # Get the corresponding bit group as a tuple
        bit_group = tuple(int(x) for x in format(i, f'0{bits_per_symbol}b'))
        mapping[bit_group] = symbol

    # Encode the bit stream
    symbols = []
    for i in range(0, len(bits), bits_per_symbol):
        bit_group = tuple(bits[i:i+bits_per_symbol])
        symbol = mapping[bit_group]
        symbols.append(symbol)

    return np.array(symbols)
# AWGN Channel
def add_awgn_noise(signal, snr_db):
    snr_linear = 10**(snr_db / 20)
    power_signal = np.mean(np.abs(signal)**2)
    noise_power = power_signal / snr_linear
    noise = np.sqrt(noise_power / 2) * (np.random.randn(len(signal)) + 1j * np.random.randn(len(
    signal)))
    return signal + noise
def decision_maker(symbols, M):
    # Generate the constellation for M-PSK
    constellation = np.array([np.exp(1j * 2 * np.pi * i / M) for i in range(M)])

    # Decide the nearest constellation point for each received symbol
    decided_symbols = []
    for symbol in symbols:
        distances = np.abs(symbol - constellation)
        decided_symbols.append(constellation[np.argmin(distances)])

    return np.array(decided_symbols)
def decode_MPSK(symbols, M):
    # Calculate the number of bits per symbol
    bits_per_symbol = int(np.log2(M))

    # Create the mapping from symbols to bit groups
    mapping = {}
    for i in range(M):
        # Calculate the phase angle for the i-th symbol
        phase = 2 * np.pi * i / M
        symbol = np.exp(1j * phase)  # Complex symbol on the unit circle
        # Get the corresponding bit group as a tuple
        bit_group = tuple(int(x) for x in format(i, f'0{bits_per_symbol}b'))
        mapping[symbol] = bit_group

    # Decode the symbols
    bits = []
    for symbol in symbols:
        # Find the closest symbol in the mapping (nearest neighbor)
        closest_symbol = min(mapping.keys(), key=lambda k: np.abs(k - symbol))
        bits.extend(mapping[closest_symbol])

    return np.array(bits)
def SendClassicalChannel(SendingList,SNR,M,EC):
  length = len(SendingList)
  if EC :
     Parity_added = CEC.add_parity_bits(convert_to_np_array(SendingList),length)
  else:
     Parity_added = convert_to_np_array(SendingList)
  Encoded_SendingData = encode_MPSK(Parity_added,M)
  Noisy_Encoded_SendigData = add_awgn_noise(Encoded_SendingData,SNR)
  Decided_Symbols = decision_maker(Noisy_Encoded_SendigData,M)
```

```
  Received_SendingData = decode_MPSK(Decided_Symbols,M)
  if EC:
    RData = convert_from_np_array(CEC.EC_with_parity(Received_SendingData,length),False)
  else:
    RData = convert_from_np_array(Received_Sendin
}
```

Listing 1.1: Classical Channel

This code simulates a classical communication channel using M-ary Phase Shift Keying (M-PSK) modulation and an Additive White Gaussian Noise (AWGN) channel model. The "encode_MPSK" function converts a stream of bits into M-PSK symbols, where M denotes the order of the PSK modulation scheme (e.g., M=4 for QPSK). This function first ensures the bit stream length is a multiple of the number of bits per symbol, then maps each group of bits to a unique phase symbol on the unit circle. The "add_awgn_noise" function simulates the effect of AWGN by adding complex Gaussian noise to the encoded symbols, controlled by the signal-to-noise ratio (SNR) in decibels. This simulates the degradation of the signal due to noise.

In the decoding process, the "decision_maker" function takes the noisy received symbols and maps them to the nearest ideal constellation point, effectively reducing the noise influence. The "decode_MPSK" function reverses the encoding process by mapping the decided symbols back to their corresponding bit groups. The overall process, encapsulated in the "SendClassicalChannel" function, involves optional error correction (EC), encoding the bit stream into M-PSK symbols, adding noise, and then decoding the received symbols back into the bit stream. This function can optionally add and correct parity bits for error correction before and after the transmission over the noisy channel, ensuring data integrity to some extent.

Now lets explore the quantum model of a channel explained in the section 1.2 :

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.visualization import *
from numpy import pi
import numpy as np
import math
from qiskit_aer import Aer
from qiskit import transpile
import qiskit.quantum_info as qi
import random
from qiskit.circuit.library import SdgGate
import matplotlib.pyplot as plt
import QuditObj
import CEC

def OAM_Modulation(Qudit,Dit,Basis):
  if Basis == 0:
    #print("Basis 0")
    if Dit == 1:
      Qudit.circuit.x(0)
    if Dit == 2:
      Qudit.circuit.x(1)
    if Dit == 3:
      Qudit.circuit.x(0)
      Qudit.circuit.x(1)
  else:
     #print("Basis 1")
     if Dit == 0:
      #print("Entered 0")
      Qudit.circuit.h(0)
      Qudit.circuit.h(1)
     elif Dit == 1:
      #print("Entered 1")
      Qudit.circuit.x(1)
      Qudit.circuit.h(1)
      Qudit.circuit.h(0)
      Qudit.circuit.s(0)
     elif Dit == 2:
```

```
      #print("Entered 2")
      Qudit.circuit.x(0)
      Qudit.circuit.h(0)
      Qudit.circuit.h(1)
     elif Dit == 3:
      #print("Entered 3")
      Qudit.circuit.x(0)
      Qudit.circuit.h(0)
      Qudit.circuit.s(0)
      Qudit.circuit.x(1)
      Qudit.circuit.h(1)
  return Qudit

def OAM_ReciviengSinglePhoton(Qudit,Basis,Shot):
  csgate = SdgGate().control(1) # the parameter is the amount of control points you want
  if Basis == 1:
     #print("Measure Basis 1")
     Qudit.circuit.h(1)
     Qudit.circuit.append(csgate, [1, 0])
     Qudit.circuit.h(0)
     Qudit.circuit.cx(1,0)
     Qudit.circuit.cx(0,1)
     Qudit.circuit.cx(1,0)
  Qudit.measure()
  Qudit.run(Shot)
  #print(Qudit.Counts)
def OAM_Getdits(QList):
    output_string = ""
    for d in QList:
      key = list(d.keys())[0]  # Extract the key from the dictionary
      output_string += key
    return  output_string
def OAM_CrossTalk(Qudit,EP):
  EA = math.sqrt(EP) * 1j
  NEA = math.sqrt(1-EP)
  Gate = [[NEA,EA,0,0],
          [EA,NEA,0,0],
          [0,0,NEA,EA],
          [0,0,EA,NEA]]
  Qudit.CT(Gate)
def OAM_Turbulance(Qudit):
    pass
def OAM_Misallignment(Qudit):
    pass
def OAM_Divergence(Qudit):
    pass
def OAM_FreeSpace(Qudit):
    pass
def OAM_Fiber(Qudit):
    pass

def POL_Modulation(cir,bit,basis,theta_radians):
#This is gettingready but with Alligment error
  cir.ry(theta_radians,0)
  if bit == 1:
    cir.x(0)
  if basis == 1:
    cir.h(0)
  return cir
def POL_ReciviengSinglePhoton(Qubit,Basis,Shots):
    #print("Bob Basis is : ", BobBasis)
    if Basis == 1:
        #print("Bob Basis is +-45")
        Qubit.h(0)
    Qubit.barrier()
```

```
    Qubit.measure(0,0)
     # Use Aer's qasm_simulator
    simulator = Aer.get_backend('qasm_simulator')
    # Execute the circuit on the qasm simulator
    new_circuit = transpile(Qubit, simulator)
    job = simulator.run(new_circuit, shots = Shots)
    # Grab results from the job
    result = job.result()
    # Return counts
    counts = result.get_counts(Qubit)
    return counts
def POL_Getbits(QList):
    output_string = ""
    for d in QList:
      key = list(d.keys())[0]  # Extract the key from the dictionary
      output_string += key
    return  output_string
def SendQuantumOAMChannel(SendingList,EP,Base,EC):
  length = len(SendingList)
  EC = True
  if EC :
      Parity_added = convert_from_np_array(CEC.add_parity_bits(convert_to_np_array(SendingList),
    length),False)
  else:
      Parity_added = SendingList
  DitSequence = map_bits_to_numbers(Parity_added)
  length2 = len(Parity_added)
  ListofCounts = []
  for dit in DitSequence:
    Qudit = QuditObj.QuditCircuit(4,1)
    Qudit = OAM_Modulation(Qudit,dit,Base)
    OAM_CrossTalk(Qudit,EP)
    OAM_ReciviengSinglePhoton(Qudit,Base,Shot=1)
    ListofCounts.append(Qudit.Counts2)
  RDits = OAM_Getdits(ListofCounts)
  RBits = map_numbers_to_bits(RDits,length2)
  if EC:
    CorrectedBits = convert_from_np_array(CEC.EC_with_parity(convert_to_np_array(RBits),length),
    False)
  else:
    CorrectedBits = RBits
    CorrectedBits = [int(num) for num in CorrectedBits]
  return CorrectedBits
def convert_to_np_array(data):
    """
    Converts a list or string of numbers into a numpy array of integers.

    Parameters:
    data (list or str): Input list or string of numbers.

    Returns:
    np.ndarray: Numpy array of integers from the input data.
    """
    if isinstance(data, str):
        data = [int(num) for num in data]
    elif isinstance(data, list):
        data = np.array(data, dtype=int)
    else:
        raise ValueError("Input must be a list or a string of numbers")
    return np.array(data, dtype=int)

def convert_from_np_array(np_array, to_string):
    """
    Converts a numpy array into a list or a string.

```

```
    Parameters:
    np_array (np.ndarray): Input numpy array.
    to_string (bool): If True, convert to string; if False, convert to list.

    Returns:
    list or str: Converted list or string from the numpy array.
    """
    if not isinstance(np_array, np.ndarray):
        raise ValueError("Input must be a numpy array")

    if to_string:
        return ''.join(map(str, np_array))
    else:
        return np_array.tolist()
def map_bits_to_numbers(bit_list):
    # Work on a copy of the input list to avoid modifying the original list
    modified_bit_list = bit_list.copy()
    # Ensure the list length is even by appending a 0 if necessary
    if len(modified_bit_list) % 2 != 0:
        modified_bit_list.append(0)

    # Map pairs of bits to numbers
    number_list = []
    for i in range(0, len(modified_bit_list), 2):
        pair = modified_bit_list[i:i+2]
        number = pair[0] * 2 + pair[1]
        number_list.append(number)
    return number_list # Return the original length as well

def map_numbers_to_bits(number_string, original_length):
    bit_string = ""
    for char in number_string:
        num = int(char)
        bits = f"{num:02b}"  # Convert number to its 2-bit binary representation
        bit_string += bits

    # If the original bit list length was odd, remove the last bit

    if original_length % 2 != 0:
        bit_string = bit_string[:-1]
}
```

Listing 1.2: Quantum Channel

This code simulates a high-dimensional quantum communication channel using Orbital Angular Momentum (OAM) quantum states, facilitated by a custom qudit object representing a higher-dimensional quantum system. The OAM_Modulation function modulates the qudit state based on the given basis and digital value (dit). This modulation involves applying various quantum gates to the qudit's circuit to encode the dit into the quantum state. For instance, in basis 0, the modulation is straightforward with x gates, while in basis 1, the modulation involves more complex operations like Hadamard (h) and phase (s) gates.

The OAM_ReceivingSinglePhoton function handles the measurement of the received qudit state. Depending on the basis, it applies a series of gates to prepare the state for measurement, including controlled SdgGate and Hadamard gates. The function then measures the state and runs the quantum circuit for a specified number of shots to obtain measurement outcomes. These outcomes are stored in the qudit object's counts, which represent the measured state of the qudit after transmission through the channel.

To simulate realistic communication scenarios, the code includes functions to introduce various channel impairments. For example, OAM_CrossTalk simulates crosstalk between different qudit states by applying a custom gate that introduces a controlled error probability. This gate mixes the amplitudes of the qudit states, reflecting the physical phenomenon where different OAM modes can interfere with each other. Placeholder functions like OAM_Turbulance, OAM_Misallignment, and OAM_Divergence suggest future extensions to model other types of channel impairments, such as atmospheric turbulence or misalignment in the optical setup.

The SendQuantumOAMChannel function orchestrates the entire transmission process. It first optionally adds parity bits for error correction, maps the bits to dits, and modulates the qudits. After introducing channel impairments like crosstalk, it measures the received qudits and decodes the results back into bits. If error correction is enabled, it applies error correction algorithms to retrieve the original bit sequence. This function simulates the complete process of sending and receiving information through a quantum OAM channel, highlighting the complex interplay between quantum state preparation, channel impairments, and error correction.

Additionally, the code provides functions for polarization modulation and measurement (POL_Modulation and POL_ReciviengSinglePhoton), allowing for hybrid communication schemes that utilize both OAM and polarization states. These functions prepare and measure the polarization state of a single qubit, which can complement the OAM-based communication by adding another layer of information encoding. The functions convert_to_np_array and convert_from_np_array facilitate data handling by converting between different data formats, ensuring compatibility with the quantum circuits and classical processing steps. The map_bits_to_numbers and map_numbers_to_bits functions handle the conversion between bit sequences and numerical representations required for qudit modulation and demodulation.

1.3.3 Simulation's iterations

Problem Statement

In addressing the challenge of estimating the error probability within a bit stream with a given confidence level, we turn to the concept of confidence intervals. This methodical approach allows us to delineate the necessary steps with clarity. Problem Statement

Understanding the Problem

In our study, we consider various scenarios involving a mixture of errors and eavesdropping attacks (Eve) in the BB84 protocol implemented in two-dimensional (d=2) and four-dimensional (d=4) quantum systems. These scenarios are characterized by different error rates and require varying numbers of iterations for simulations. For each case, we calculate the total probability of error per bit (P_e) and determine the necessary number of iterations based on this probability. We then analyze the worst-case scenario to ensure robust security measures.

Bit streams, consisting of N bits, experience errors that follow a binomial distribution with a given probability P_e. The mean number of errors, μ, inherent to this distribution, is computed as $\mu = NP_e$. Understanding the variance, σ^2, which is crucial for evaluating the spread of errors, is also essential. The variance is expressed as $\sigma^2 = NP_e(1 - P_e)$. This detailed analysis of error characteristics is vital for optimizing the performance and security of quantum communication protocols under different operational conditions.

Confidence Interval

A pivotal step in our approach is the definition of confidence intervals. These intervals provide a structured framework within which we anticipate the true parameter (error probability) to reside, encapsulating the inherent uncertainty of our estimation. The confidence interval formula for a proportion P_e is fundamental:

$$CI = P_e \pm Z_{\alpha/2}\sqrt{\frac{P_e(1 - P_e)}{n}} \tag{1.4}$$

where $Z_{\alpha/2}$ represents the Z-value corresponding to our desired confidence level, and n denotes the number of simulations.

To achieve a for example a 90% confidence level, we ascertain the appropriate Z-value $Z_0.05 \approx= 1.645$ This value serves as a critical determinant in our calculations, ensuring robust statistical inference.

To determine the requisite number of simulations n, we rearrange our confidence interval formula. This strategic adjustment yields,

$$n = \frac{Z^2_{\alpha/2}P_e(1 - P_e)}{E^2} \tag{1.5}$$

where E denotes the desired margin of error. This calculation underscores the importance of precision in simulation planning, ensuring reliable estimation outcomes.

1.3.4 Values for Simulation

To compute the values of n for our simulation, we must determine the theoretical error probabilities P_e for various scenarios. This is achieved using a probability tree, as illustrated in Figure 1.4. In this tree, the detected parts represent cases where an error is introduced into the bit stream. The probability tree for more complex cases, such as a 4-dimensional (4D) BB84 protocol with both errors and eavesdropping (Eve), is significantly more intricate due to the non-homogeneous nature of errors across different bases.

The theoretical values are summarized in Table 1.1, where P_q denotes the error probability for a single qubit or qudit. It is important to note that for the 4D BB84 case, the error in one bit is calculated by:

$$P_b = 1 - \sqrt{1 - P_e} \tag{1.6}$$

The last two columns in Table 1.1 represent values computed with $P_q = 0.11$. For the sake of the simulation we will use n = 300.

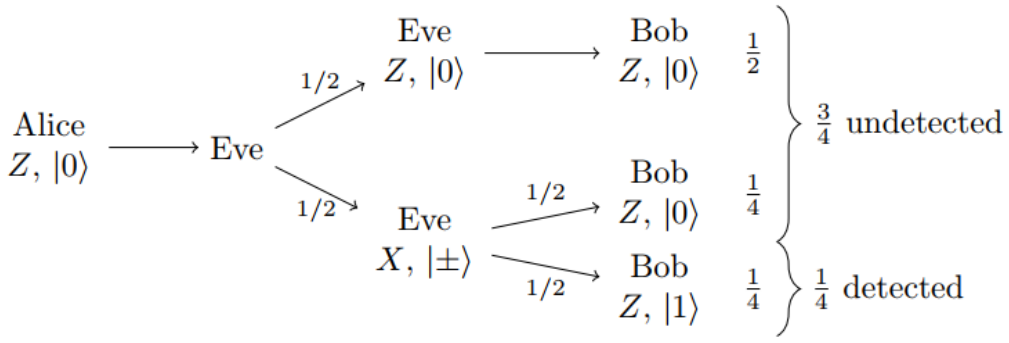

Figure 1.4: The Probability tree for 2D BB84 with Eve and No error.From Book [5]

Description	P_e	P_e (based on P_q	n
2D No Eve No Error	0	0	0
2D No Eve Error	P_q	0.11	106
2D Eve No Error	0.25	0.25	203
2D Eve Error	$P_q(1-P_q)+0.25$	0.3479	246
4D No Eve No Error	0	0	0
4D No Eve Error	$0.75P_q$	0.0825	44
4D Eve No Error	0.375	0.375	179
4D Eve Error	$0.75P_e(1-P_e)+0.375$	0.448425	207

Table 1.1: Error Rates and number of iterations in Different Scenarios for a 90% confidence level and 5% error margin

Chapter 2

Results

2.1 Channel Results

First, we examine how the models defined in Chapter 1 perform under different parameters. Let us begin with the classical channel. For this channel, we consider two main scenarios: one with error correction and one without. The error correction scheme employed here is a basic method using parity bits, as introduced in reference [1]. The error percentage versus Signal-to-Noise Ratio (SNR) for these two cases is illustrated in Figure 2.1. The line indicating an

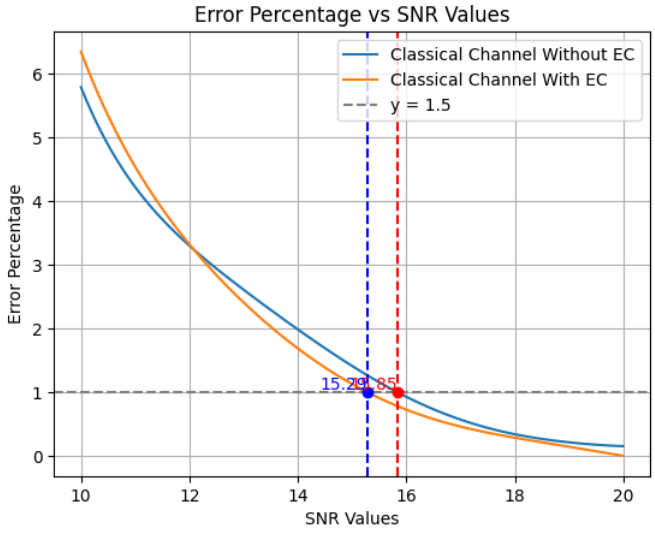

Figure 2.1: Error percentage versus SNR values

error percentage of 1.5% is shown in the figure for later comparison with the quantum channel. Next, we examine the quantum channel model. In this model, we consider four different possibilities. First, we can use an Orbital Angular Momentum (OAM) based link (4-Dimension) with the OAM basis. Second, we can use an OAM based link (4-Dimension) with the angular (ANG) basis. Additionally, in both scenarios, we can apply the classical error correction scheme mentioned earlier. Consequently, we have four distinct cases. These cases are compared in Figure 2.2.

We see that in the quantum channel the error percentage line of 1.5% is for different values for different cases so we will use the ANG basis with EC because it allows us to have a worse error probability.

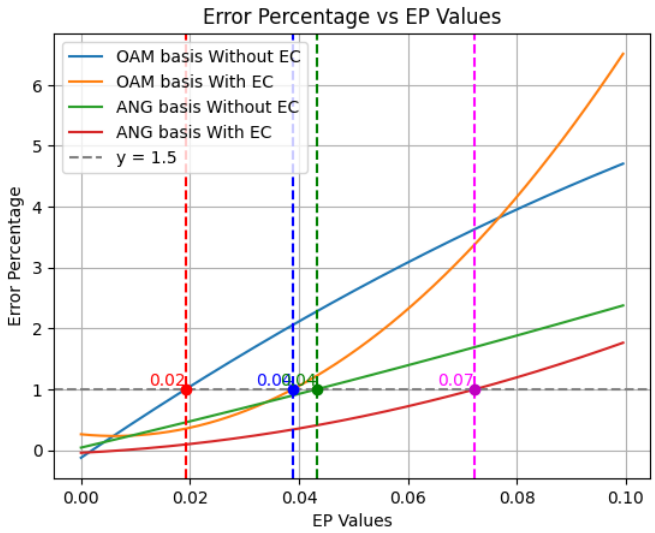

Figure 2.2: Error percentage versus Error Probability(EP) values

2.2 BB84 Results

First, we examine the BB84 protocol results under an ideal channel condition to assess how closely our simulation aligns with the theoretical values derived in Chapter 1. Table 2.1presents the outcomes from 300 iterations of the algorithm, providing a comprehensive comparison between our simulated data and the expected theoretical performance.

Description	Max Error (%)	Min Error (%)	Avg Error (%)	Abortions	Detection (%)
2D No Eve No Error Bob	0	0	0	0	0
2D No Eve Error Bob	46.67	0	19.04	0	96.33
2D Eve No Error Bob	52.94	0	24.04	0	97.33
2D Eve No Error Eve	58.33	0	24.53	0	0
2D Eve Error Bob	69.23	5.26	34.92	0	100
2D Eve Error Eve	66.67	0	34.10	0	0
4D No Eve No Error Bob	0	0	0	0	0
4D No Eve Error Bob	21.43	0	4.08	0	46.67
4D Eve No Error Bob	83.33	0	24.30	0	95.33
4D Eve No Error Eve	66.67	0	24.12	0	0
4D Eve Error Bob	83.33	0	27.51	0	98.33
4D Eve Error Eve	66.67	0	27.03	0	0

Table 2.1: Error Rates and Detection in Different Scenarios

Next, we analyze the BB84 protocol's performance in both classical and quantum channels. Table 2.2 details the results for a quantum channel with an error probability (EP) of 0.07. Conversely, Table 2.3shows the results for a classical channel with a signal-to-noise ratio (SNR) of 15.28 dB. These tables allow us to compare the effects of different channel conditions on the BB84 protocol's efficacy.

Description	Max Error (%)	Min Error (%)	Avg Error (%)	Abortions	Detection (%)
2D No Eve No Error Bob	50	0	0.83	24	3.26
2D No Eve Error Bob	46.15	0	19.33	19	80.43
2D Eve No Error Bob	60	0	26.28	29	92.25
2D Eve No Error Eve	71.43	0	24.69	29	0
2D Eve Error Bob	66.67	0	34.99	19	96.80
2D Eve Error Eve	73.33	5.26	34.58	19	0
4D No Eve No Error Bob	38.89	0	0.24	10	0.69
4D No Eve Error Bob	37.50	0	4.12	10	46.90
4D Eve No Error Bob	83.33	0	24.01	7	96.25
4D Eve No Error Eve	62.50	0	24.55	7	0
4D Eve Error Bob	66.67	0	27.05	13	96.52
4D Eve Error Eve	70	0	26.08	13	0

Table 2.2: Error Rates and Detection in Various Scenarios with a quantum channel

Description	Max Error (%)	Min Error (%)	Avg Error (%)	Abortions	Detection (%)
2D No Eve No Error Bob	33.33	0	0.53	27	1.83
2D No Eve Error Bob	50	0	19.08	26	78.83
2D Eve No Error Bob	60	0	26.33	29	94.46
2D Eve No Error Eve	61.54	0	25.75	29	0
2D Eve Error Bob	77.78	0	34.39	27	97.07
2D Eve Error Eve	72.73	5.26	34.26	27	0
4D No Eve No Error Bob	11.11	0	0.15	10	2.07
4D No Eve Error Bob	25	0	4.06	18	47.16
4D Eve No Error Bob	70	0	24.09	19	95.02
4D Eve No Error Eve	62.5	0	24.48	19	0
4D Eve Error Bob	66.67	0	26.86	15	97.89
4D Eve Error Eve	75	0	26.95	15	0

Table 2.3: Error Rates and Detection in Various Scenarios with a classical channel

Bibliography

[1] A.B. Carlson and P.B. Crilly. *Communication Systems*. McGraw-Hill Higher Education, 2010.

[2] Mohammad Mirhosseini, Omar S Magaña-Loaiza, Malcolm N O'Sullivan, Brandon Rodenburg, Mehul Malik, Martin P J Lavery, Miles J Padgett, Daniel J Gauthier, and Robert W Boyd. High-dimensional quantum cryptography with twisted light. *New Journal of Physics*, 17(3):033033, mar 2015.

[3] Alicia Sit, Felix Hufnagel, and Ebrahim Karimi. Chapter 6 - quantum cryptography with structured photons. In Mohammad D. Al-Amri, David L. Andrews, and Mohamed Babiker, editors, *Structured Light for Optical Communication*, Nanophotonics, pages 139–176. Elsevier, 2021.

[4] Alan E. Willner, Kai Pang, Hao Song, Kaiheng Zou, and Huibin Zhou. Orbital angular momentum of light for communications. *Applied Physics Reviews*, 8(4):041312, 10 2021.

[5] T.G. Wong. *Introduction to Classical and Quantum Computing*. Rooted Grove, 2022.