Tailoring 3D topological codes and decoders to biased noise

Arthur Pesah

Supervisor: Dan Browne

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Research

of

University College London.

Department of Physics and Astronomy

University College London

September 1, 2021

I, Arthur Pesah, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

In this thesis, we use the belief propagation decoder to evaluate the threshold of several types of 3D topological codes under a biased noise model. More precisely, we consider both a cubic and a rhombic lattice, and introduce deformations of their stabilizers with the objective of improving the performance of those codes in the biased regime. We show that combining this choice of stabilizers with the belief propagation decoder increases the threshold of the 3D toric code with a cubic lattice from 22% to more than 36% under a highly biased noise model. For the rhombic lattice, we show that the standard deformation procedure does not improve its threshold. However, simply using a highly biased noise model on the original rhombic code increases the threshold from 1.6% to 28%, even for a modest bias ratios of 30.

Contents

Chapter 1

Introduction

1.1 Motivation

When the concept of a quantum computer was first proposed and formalized in the 1980s and 1990s [1, 2, 3], many physicists were skeptical that those devices would one day see the light of day. As an example, Haroche and Raimond famously said in an article that appeared in 1996 in Physics Today that *"the large-scale quantum machine, though it may be the computer scientist's dream, is the experimenter's nightmare."*. The reason for their skepticism is the inherent fragility of quantum states: any noise present in a quantum system, due for instance to unwanted interactions with the environment, can irreversibly destroy the state. The invention of the first quantum error-correcting codes [4, 5], and with them of *threshold theorem*, changed the game: there exist some families of quantum codes that can correct arbitrary errors by increasing the number of redundant qubits, as long as the noise level of the system is below a certain threshold.

While those first quantum codes served as a useful proof of concept, their threshold was estimated to be around 10^{-6} [6], which was far below the experimental capabilities of the time. The introduction of the *stabilizer formalism* in 1998 [7] revolutionized quantum error-correction and led to the invention of the *toric code* [8, 9], whose two-dimensional instance, the *surface code*, has a threshold over 1% in realistic settings [10].

While the surface code remains today one of the most promising architectures for a first generation of fault-tolerant devices, it has the downside to require a large qubit overhead to perform universal quantum computation with it. On the other hand, its three-dimensional generalization, the *3D toric code*, has recently been shown to be better suited for universal quantum computing: it has a natural implementation of non-Clifford gates such as the Toffoli gate [11]. However, constructing a 3D architecture in practice is experimentally challenging, and recent work has shown that taking all the characteristics of the code into account, building a 3D toric code might be more costly than its 2D version [12].

Meanwhile, recent experimental advances have allowed to gain new insights on the nature of quantum noise. While most quantum error-correcting codes had been designed and evaluated with the assumption that noise was symmetric, i.e. that bit-flip errors occur as frequently as phase-flip errors, experiments on superconducting qubits [13, 14, 15], trapped ions [16] and quantum dots [17] have shown that noise is actually often *biased* towards one type of errors. For instance, in Kerr-cat cubits, phase-flip errors can occur 100 times more frequently than bit-flip errors [18].

Under this new assumption, small modifications of the surface code have been proposed that have boosted the threshold of the code with realistic bias levels [19, 20, 21, 22]. The latest of those new codes tailored for biased noise, the XZZX surface code, has improved the threshold from around 11% to around 40% for the realistic bias ratio of 100 (reached by Kerr-cat qubits) [22]. Those codes are constructed by *deforming the stabilizers* of the original surface code, meaning that the operators used to detect errors, called *stabilizers*, are rotated in a certain way to form new stabilizers.

In this work, we aim to answer the following question: can the 3D toric code be improved under biased noise by deforming its stabilizers? If a significant improvement is discovered, new comparisons with the surface code must be performed, similar to the one described in Ref. [23], in order to re-evaluate

the 3D code in realistic settings, with the potential to rehabilitate it as a practically relevant error-correcting code.

1.2 Objectives

The objective of this project is to propose a new variant of the 3D toric code with deformed stabilizers and evaluate it under a biased noise model, hoping to improve the performance. Similarly to the 2D case with the recently proposed XZZX code, the new stabilizers must provide an advantage in the biased noise regime, such as reducing the dimensionality of the decoding problem. Since the 3D toric code can come in many different flavors, we will restrict our attention to two lattice types—cubic and rhombic—and to periodic boundary conditions. The project can be divided into three main steps:

1. Designing the new stabilizers

2. Numerically evaluating the threshold of the new code under several biased noise models, for both the cubic and the rhombic lattice, using a decoder with high performance.

3. Theoretically computing the threshold—using statistical mechanical models—whenever it is possible

This project is part of an international collaboration comprising four other researchers: Eric Huang (University of Maryland), Michael Vasmer (Perimeter Institute), Christopher Chubb (Université de Sherbrooke) and Arpit Dua (Yale University). Before I joined, preliminary work had already been done: some deformed stabilizers were designed for the cubic lattice and evaluated using a specific decoder called the SweepMatch decoder, showing improved performance compared to the original toric code: from 15% to 21% for highly biased noise with a ratio over 100. However, recent work has shown that another decoder—the BP-OSD decoder (Belief Propagation with Ordered Statistics Decoding)—performs better than SweepMatch for the 3D homological product code [24]. I was therefore given the first task to implement and evaluate this

decoder on the cubic lattice in the regimes of interest, with both the regular and the deformed stabilizers. A second task would then be to design stabilizer deformations for the rhombic code, and evaluate the new code using the best-performing decoder. Finally, we would all look at statistical mechanical models to try to derive the threshold of the deformed code in theory.

1.3 Contributions

During the three months of this project, I achieved the following tasks:

1. I implemented a fast BP-OSD decoder in Python and integrated it into the library `bn3d` developed by the collaboration

2. I evaluated the threshold of the BP decoder for the cubic lattice, and showed a significant performance improvement compared to the Sweep-Match decoder, e.g. from 15% to 22% for the undeformed code and from 22% to more than 36% for the deformed code, under a noise model fully biased towards phase-flip errors.

3. I implemented the rhombic lattice and its deformation, and integrated it in `bn3d`. I evaluated its threshold under biased noise, showing a threshold gain from 1.5% with no bias to 28% with maximal bias. However, the proposed deformation did not improve the code.

4. I developed a graphical interface to visualize and manipulate 3D codes, noise models and decoders, gaining more intuition on those codes as a result and simplifying the debugging process.

The evaluation of the theoretical threshold using analogies from statistical physics is left as future work.

1.4 Outline

In the following part of this thesis, Section 2, we will give some background on quantum error correction, biased noise and 3D toric codes, including a review of the relevant literature. We will then present our method and results in Section

3, including a presentation of the stabilizer deformations we introduced, the threshold results we obtained and a tour of the graphical interface. Finally, we will conclude this thesis and discussed future work in Section 4.

Chapter 2

Background

2.1 Classical error correction

Before discussing quantum error correction, we will first review the basics of classical error correction. After having defined some general concepts of error correction, we will dive into the theory of linear codes.

2.1.1 General setting

Let us consider the following setting: we would like to send a N-bit message $\mathbf{x}$ across a noisy channel. If we choose to send $\mathbf{x}$ directly without any pre-processing, a different result $\tilde{\mathbf{x}}$ will arrive with a certain number of errors, i.e. flipped bits. To protect the message against bit-flip errors, we can choose to add redundancy to it, for instance by sending each of its bits three times: 0 becomes 000 and 1 becomes 111. Assuming that at most one error can occur on each triplet, the original message can be decoded by taking a majority vote, e.g. 010 is decoded as 0. The message $\mathbf{y} = \mathbf{xxx}$ that we send across the channel is called an encoding of $\mathbf{x}$. More generally, an error-correction process can be summarized with the following diagram:

$$\mathbf{x} \xrightarrow{\text{encoding}} \mathbf{y} \xrightarrow{\text{noise}} \tilde{\mathbf{y}} \xrightarrow{\text{decoding}} \tilde{\mathbf{x}}$$

Let us now introduce some important jargon. A **codeword** is an element in the image of the encoding process. For instance, in the code discussed above—called the 3-repetition code—, we have two codewords: 000 and 111.

The **distance** of a code is the minimum number of bit-flips required to pass from one codeword to another, or in other words, the minimum number of errors that would be undetectable with our code. For instance, the distance of the 3-repetition code is 3 (we need to flip all 3 bits to have an undetectable error). A code that encodes k bits with n bits and has a distance d is called an $[n, k, d]$**-code**. The 3-repetition code is an example of $[3, 1, 3]$-code. More generally, an n-repetition code (consisting of n repetition of each bit) is a $[n, 1, n]$-code. Finally, the **rate** of an $[n, k, d]$-code is defined as $R = k/n$. While the n-repetition code family has a high distance, we see that it has a low rate $R = 1/n$, which asymptotically goes to 0 when $n \to \infty$, making those codes impractical as they require a high number of redundant bits to encode a single bit.

2.1.2 Parity-checks and Hamming code

In 1950, Richard Hamming discovered a more intelligent way to introduce redundancy in a message than having to repeat it several times [25]. To illustrate his method, let us consider a 4-bit message $\mathbf{x} = abcd$, as well as the three variables

$$z_1 = a \oplus b \oplus d$$
$$z_2 = a \oplus c \oplus d$$
$$z_3 = b \oplus c \oplus d$$

where $\oplus$ denotes a XOR operation (addition modulo 2). Those are called parity-check bits, as they indicate the parity of the sum (0 for even and 1 for odd). If we now send the 7-bit message $\mathbf{y} = abcdz_1z_2z_3$, any single-bit error will be correctable. Indeed, let's see what happens if an error occurs only on bit a. In this case, both $a \oplus b \oplus d$ and $a \oplus c \oplus d$ will be different from z_1 and z_2, while the value of z_3 will remain at $b \oplus c \oplus d$. This can only happen if a is flipped, which allows us to correct the error. A similar reasoning can be performed for the other bits. This code, called the Hamming code, is a

$[7,4,3]$-code with a rate $R = 4/7 \approx 0.57$, which is already better than the rate $R \approx 0.33$ of the 3-repetition code, for a similar distance.

2.1.3 Linear codes

This idea of transmitting both the message and some parity-check bits can be generalized with the notion of linear code. A linear code consists in using a matrix $\mathbf{G}$—called **generator matrix**—as our code, i.e.

$$\mathbf{y} = \mathbf{G}\mathbf{x}. \tag{2.1}$$

If our message $\mathbf{x}$ has length k and is complemented by m parity checks, such that $n = k + m$ is the size of $\mathbf{y}$, we can often write $\mathbf{G}$ as

$$\mathbf{G} = \begin{pmatrix} \mathrm{I}_k \\ \hline \mathbf{A} \end{pmatrix} \tag{2.2}$$

with I_k the $k \times k$ identity matrix (used to reproduce the message in the code) and $\mathbf{A}$ an $m \times k$ matrix that performs the parity checks. In this notation, all the matrix operations are performed modulo 2, i.e. in the field $\mathbb{Z}_2$. For instance, the generator matrix of the Hamming code can be written

$$\mathbf{G} = \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right) \tag{2.3}$$

A visual way to construct the generator matrix is through the **Tanner graph** of the code. The Tanner graph is a bipartite graph containing two types of nodes, the data nodes (one for each bit of the original message) and the check nodes (one for each parity check). A check node i and a data node

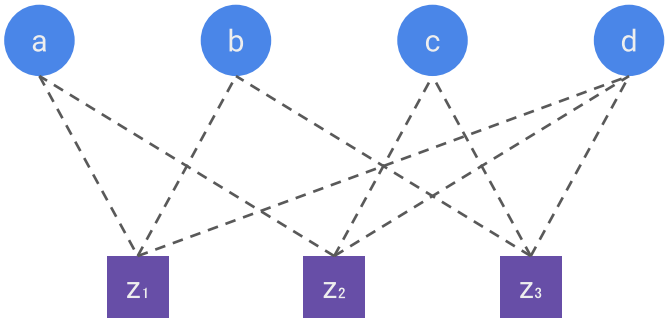


Figure 2.1: Tanner graph of the Hamming code. The top nodes represent the bits of the message and the bottom nodes the parity checks, with an edge whenever a bit is involved in a parity check.

j are connected if the parity check z_i depends on x_j. Figure 2.1 represents the Tanner graph of the Hamming code. The generator matrix can then be constructed by setting $\mathbf{A}$ (its non-identity part) to the adjacency matrix of the Tanner graph, i.e. it has a 1 at row i and column j if the check node i is connected to the data node j, and 0 otherwise.

An equivalent picture to describe linear codes is through the **parity-check matrix**, defined as an $m \times n$ matrix $\mathbf{H}$ such that

$$\mathbf{H}\mathbf{y} = 0 \tag{2.4}$$

if and only if $\mathbf{y}$ is a codeword. If the generator matrix is given by Eq. (2.2), the corresponding parity-check matrix can then be constructed as

$$\mathbf{H} = \left(\mathbf{A} \;\middle|\; \mathrm{I}_m\right) \tag{2.5}$$

The parity-check matrix gives a convenient method to detect and correct errors. Indeed, if $\tilde{\mathbf{y}} = \mathbf{y} + \mathbf{e}$ is the received message disturbed by an error vector $\mathbf{e}$, applying the parity-check matrix to $\tilde{\mathbf{y}}$ gives

$$\begin{aligned} \mathbf{H}\tilde{\mathbf{y}} &= \mathbf{H}(\mathbf{y} + \mathbf{e}) \\ &= \mathbf{H}\mathbf{e} \end{aligned} \tag{2.6}$$

The new vector $\mathbf{s} = \mathbf{H}\mathbf{e}$ has dimension m and is called the **syndrome**. Each component s_i of the syndrome is equal to 1 if the parity-check equation i is

violated. Decoding a message then consists in finding the most probable error $\mathbf{e}$ that has yielded to the syndrome $\mathbf{s}$.

An important class of linear codes are the low-density parity-check (LDPC) codes, introduced by Gallager in 1962 [26], but forgotten until a renewed wave of interest since the 1990s due to the improvement of decoding methods [27]. LDPC codes are defined as linear codes with a sparse parity-check matrix: each parity check only involves a constant number of data bits, and each data bit is only involved in a constant number of parity checks. The quantum version of LDPC codes has also gained traction recently [28] and we will consider those codes when discussing about belief propagation in Section 2.5.

2.2 Quantum error correction

Generalizing classical error correction to the quantum domain is not an easy task. Indeed, let us consider a 1-qubit state $|\psi\rangle = a|0\rangle + b|1\rangle$ going through a noisy quantum channel. Contrary to classical channels where errors can be modelled by a series of bit-flips, our quantum state can in principle be deformed in infinitely many ways. Moreover, measuring the state changes it, so one must be careful about what exactly to measure in order to gain information on some potential errors. To solve those challenges, a clever formalism for quantum error correction was proposed at the end of the 1990s: the stabilizer formalism. With this formalism in hand, it became possible to construct an important family of quantum error-correcting codes, known as topological codes, and which includes the 2D and 3D toric codes studied in this thesis.

After having reviewed some general notions of quantum error correction and noise models, we will introduce the stabilizer formalism and show how it can help to design error-correcting codes. We will then define topological codes, with an accent on the 2D toric code. Finally, we will give a short review on the existing decoding techniques for topological codes.

2.2.1 General setting

In any quantum algorithm, a state $|\psi\rangle$ is generated at each cycle of the quantum device. However, noise can alter this state with a certain probability, giving a state $\left|\tilde{\psi}\right\rangle$ instead. Similarly to the classical case, any quantum error-correction process can be summarized by four main steps:

1. **Encode** the logical state $|\psi\rangle$ as a physical state $|\psi\rangle_E$ defined on more qubits.

2. **Detect** whether an error has occurred by performing measurements on the physical state $\left|\tilde{\psi}\right\rangle_E$ without altering the state.

3. **Decode** the measurement results and propose a correction operator

4. **Compute** gates on the logical state by redefining a universal gate set on the code.

In this thesis, we will mostly focus on the first three steps. Those can be described by the following process:

$$|\psi\rangle \xrightarrow{\text{encoding}} |\psi\rangle_E \xrightarrow{\text{noise}} \left|\tilde{\psi}\right\rangle_E \xrightarrow{\text{decoding}} |\psi\rangle$$

The vector space of all encoded states $|\psi\rangle_E$ is called the **code space**. The **distance** of a code is the minimum number of qubits that need to be acted on (e.g. with an error) to go from one state of the code space to another. A code that encodes k logical qubits with n physical qubits and has a distance d is called an $[[n, k, d]]$-code. Finally, we define a **logical operator** as an operator that can change logical state of the code.

Before looking at different encoding and decoding methods, we first need to review how quantum noise works.

2.2.2 Noise in a quantum computer

The first challenge with quantum error-corrections is that noise can take a large variety of shapes. In the most general way, a noise process can be modelled by a quantum channel $\mathcal{N}$ acting on n qubits. However, quantum error-correcting codes are usually studied with two more assumptions on this channel:

1. It is **separable**: noise acts independently on each qubit:

$$\mathcal{N}(\rho) = \mathcal{N}_1(\rho) \otimes ... \otimes \mathcal{N}_n(\rho) \tag{2.7}$$

2. It is **incoherent** (or stochastic): an operator is applied randomly to each qubit with a certain probability. In general, single-qubit incoherent errors can be modelled by a Pauli channel:

$$\mathcal{N}_i(\rho) = (1-p)\rho + p(r_X \mathrm{X}\rho\mathrm{X} + r_Y \mathrm{Y}\rho\mathrm{Y} + r_Z \mathrm{Z}\rho\mathrm{Z}) \tag{2.8}$$

where $r_X + r_Y + r_Z = 1$. It means that at each cycle, a Pauli error $P \in \{\mathrm{X}, \mathrm{Y}, \mathrm{Z}\}$ acts on our qubit with a probability $p \cdot r_P$.

Since $Y = iXZ$ consists in applying both X and Z consecutively, all our quantum errors can in fact be decomposed into only two types of errors:

$$\text{Bit-flips: } |\psi\rangle \longrightarrow \mathrm{X}|\psi\rangle$$
$$\text{Phase-flips: } |\psi\rangle \longrightarrow \mathrm{Z}|\psi\rangle$$

A common example of Pauli channel is the depolarizing channel, which assigns equal probability $p/3$ to all three Pauli operators:

$$r_X = r_Y = r_Z = \frac{1}{3}$$

In this thesis, we will mostly be interested in biased noise models, in which one of the three Pauli operators occur with a higher probability than the other two. For instance, in the pure Z noise model, $r_X = r_Y = 0$ and $r_Z = 1$. We will discuss biased noise models in more details in Section 2.3

2.2.3 Illustrative example: the repetition code

Let us start with an illustrative example: the generalization of the 3-repetition code to the quantum domain. Let us encode the state $|0\rangle$ with $|000\rangle_E$ and the state $|1\rangle$ with $|111\rangle_E$. Therefore, a general 1-qubit state $|\psi\rangle = a|0\rangle + b|1\rangle$ is encoded as

$$|\psi\rangle_E = a|000\rangle + b|111\rangle, \tag{2.9}$$

and $\mathcal{C} = \{a|000\rangle + b|111\rangle : a, b \in \mathbb{C}, |a|^2 + |b|^2 = 1\}$ defines the code space of our repetition code. For simplicity, let us first consider the effect of X errors on the code. We can notice that in this case, the distance of the code is 3: applying the operator $\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3$ turns the state $a|000\rangle + b|111\rangle$ into $a|111\rangle + b|000\rangle$ which is still in the codespace. Conversely if a bit-flip occurs on only one or two qubits, the state will leave the code space and the error can be in principle detected.

In order to detect and correct those bit-flip errors, it is not possible anymore to simply look at the state and apply a majority vote. However, we can measure the parity of each pair of qubits and verify that there is always an even number of 1. This can be done using the three operators $S_1 = \mathrm{Z}_1\mathrm{Z}_2$, $S_2 = \mathrm{Z}_2\mathrm{Z}_3$ and $S_3 = \mathrm{Z}_1\mathrm{Z}_3$. Indeed, those operators commute and can therefore be measured simultaneously. Moreover, if we measure one of the stabilizers $\mathrm{Z}_i\mathrm{Z}_j$, we will obtain $+1$ if the qubits i and j are the same, and -1 otherwise. Measuring those three operators hence tells us if the three physical qubits are the same, or in other words, if the state is in the codespace. For instance, those three operators applied to the state $|\psi\rangle = |001\rangle$ gives us $\{s_1 = 1, s_2 = -1, s_3 = -1\}$. This indicates an error probably occurred on the third qubit, since only the first two qubits have an even parity and are therefore identical.

2.2.4 Stabilizer formalism

The stabilizer formalism generalizes the intuition developed in the previous example, by giving a general framework to detect errors without disturbing the state, generalizing at the same time the parity-check picture that we saw in the previous section.

Let us start with some defintions. The n-qubit **Pauli group** $\mathcal{P}_n$ is the (multiplicative) group generated by all tensor products of Pauli elements:

$$\mathcal{P}_n = \{aP_1 \otimes ... \otimes P_n | P_i \in \{\mathrm{I}, X, Y, Z\}, a \in \{1, -1, i, -i\}\} \tag{2.10}$$

An element of the Pauli group is called a **Pauli operator**. The **weight** of a Pauli operator is the number of non-identity elements in it. For instance, $\mathrm{XIIZ} = \mathrm{X}_1\mathrm{Z}_4 \in \mathcal{P}_4$ has weight 2.

The **stabilizer group** $\mathcal{S}$ of an $[[n,k,d]]$-code with a code space $\mathcal{C}$ is a subgroup of $\mathcal{P}_n$ such that

1. All the elements of $\mathcal{S}$ commute

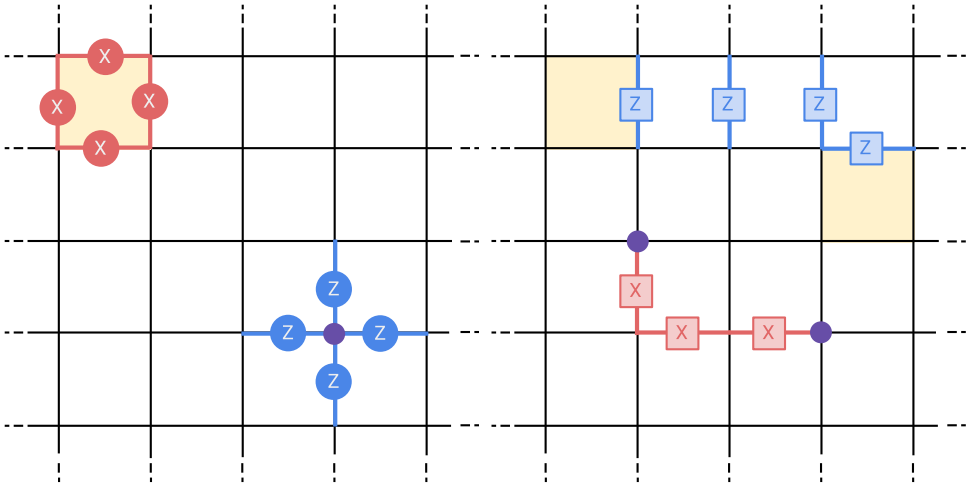
2. For each n-qubit state $|\psi\rangle$, $|\psi\rangle \in \mathcal{C} \iff \forall S \in \mathcal{S}, S|\psi\rangle = |\psi\rangle$.

The first property means that we can measure all the stabilizers simultaneously. The second property means that measuring a state with a stabilizer does not disturb it, and returns 1 if we are still in the codespace. On the other hand, if an error $P \in \mathcal{P}_n$ of weight below the distance d occurs, some of stabilizer measurements will return -1. In the repetition code introduced above, the stabilizer group is given by $\mathcal{S} = \{\mathrm{I}, \mathrm{Z}_1\mathrm{Z}_2, \mathrm{Z}_2\mathrm{Z}_3, \mathrm{Z}_1\mathrm{Z}_3\}$.

Finally, we call the result of all the stabilizer measurements is called the **syndrome**. For instance, measuring the stabilizers for the state $|\psi\rangle = |001\rangle$ in the repetition code gives the syndrome $(1,-1,-1)$. Moreover, each stabilizer equal to -1 is called an **excitation**.

A convenient way to describe a code and its stabilizers is through the **quantum parity-check matrix**, a straightforward generalization of its classical counterpart. To write the parity-check matrix from a stabilizer code, we start by constructing its Tanner graph. Like the classical Tanner graph, the quantum one is still a bipartite graph with two types of node, stabilizers and qubits, but qubit nodes now appear twice: once to represent X errors and once for Z errors. There is an edge between a stabilizer node S_i and a qubit node $q_j^{(\mathrm{X})}$ (resp. $q_j^{(\mathrm{Z})}$) if the stabilizer S_i acts non-trivially on qubit q_j with the operator X (resp. Z). The parity-check matrix is then the $m \times 2n$ adjacency matrix of the Tanner graph, with stabilizers in rows and qubits in columns. By convention, the first n columns represent qubits of type Z and the last n columns qubits of type X .

A particularly useful family of stabilizer codes is the class of **CSS codes** (named after Calderbank, Shor and Steane). Those are codes which contain two types of stabilizers: the X stabilizers (made entirely of X and I operators)



(a) Vertex and face stabilizers **(b)** Detection of X and Z errors

Figure 2.2: **(a)** Original surface code. Edges are qubits, and we define two types of stabilizers: plaquette stabilizers, made of Z operators around each face, and vertex stabilizers, made of X operators around each vertex. **(b)** Chains of errors excite the stabilizers at their boundary

and the Z stabilizers (made entirely of Z and I operators). In other word, no stabilizer mixes both X and Z operators. CSS codes are characterized by a parity-check matrix of the form

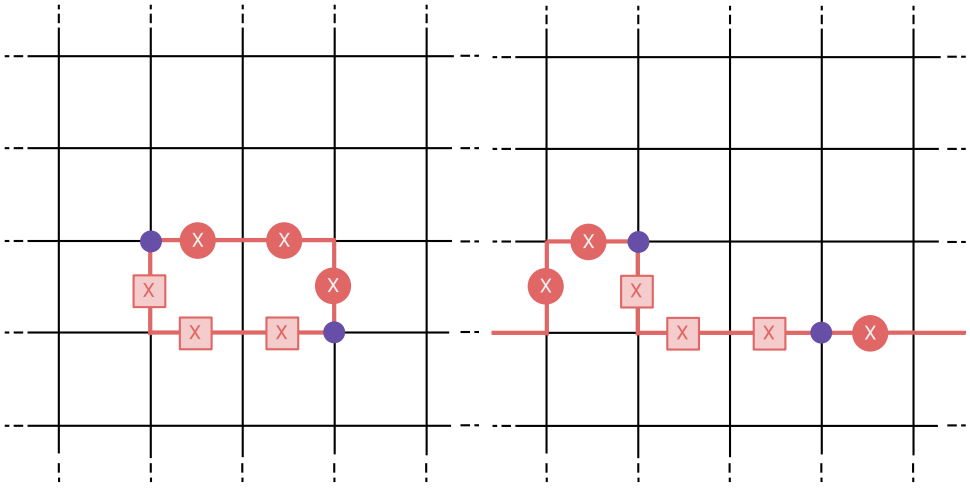
$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_Z & 0 \\ 0 & \mathbf{H}_X \end{pmatrix} \tag{2.11}$$

with $\mathbf{H}_X\mathbf{H}_Z^T = \mathbf{H}_Z\mathbf{H}_X^T = \mathbf{0}$ to ensure that the commutation conditions are fulfilled.

2.2.5 Topological codes

With the stabilizer formalism at hand, we are now ready to introduce the family of quantum error-correcting codes considered in this thesis: the topological codes. The most common example of topological code is the 2D toric code, also called surface code, and we will use it as an illustrative example.

In the surface code, we place the physical qubits on the edges of a 2D lattice with periodic boundary conditions, as illustrated in Figure 2.2a. Topological codes are constructed differently than before: rather than starting with



(a) Trivial loop **(b)** Non-trivial loop

Figure 2.3: Decoding problem: a chain of errors (red squares) produces a given syndrome (purple circles). The goal of decoding is to propose a correction (red circles) that fit the syndrome without creating a logical operator. **(a)** Correction that results in a stabilizer: the original state is recovered **(b)** Correction that results in a logical operator: the logical state is modified

a code space and constructing its stabilizer group, we start by constructing the stabilizers. The surface code is a CSS code, and has therefore two types of stabilizers: vertex stabilizers—defined on each vertex as the product of X operators on the adjacent qubits—, and plaquette stabilizers—defined on each face as the product of Z operators on the bordering qubits. Those operators commute, as either zero or two X and Z operators intersect between the different stabilizers.

What happens when the surface code is subjected to Z errors? We can consider all Z errors as a set of strings on the lattice. Since plaquette operators are made only of Z operators, they will all commute with our errors. On the other hand, if we measure vertex operators, we can observe that only the two ends of each string will be detected by vertex operators, since boundary vertices are the only ones that intersect with the errors on an odd number of qubits (exactly one). The picture for X errors is very similar, with strings of parallel

qubits instead of adjacent qubits, and plaquette instead of vertex operators. An illustration of this process is given in Figure 2.2a.

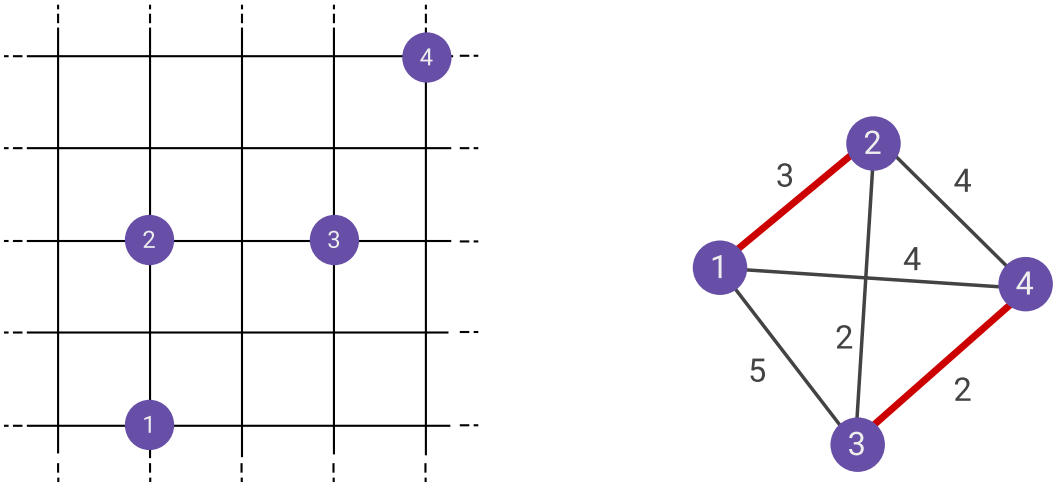
We can now wonder what happens when a string of errors form a loop? There are two possibilities: either the loop does not go around the torus—we say that the loop is *trivial*—or it does. In the first case, the physical error will not result in a logical error, since trivial loops of Z or X errors can be constructed as products of plaquette or vertex operators, and are therefore stabilizers themselves. Therefore, strings of errors that go around the torus will be the only undetectable errors. If we denote L the size of the lattice, the shortest non-trivial loop has length L, and the distance of the code is therefore $d = L$. As non-detectable errors, non-trivial loop can also be seen as logical operators.

The surface code can be generalized to higher dimensions, including the 3D toric code that we will discuss in Section 2.4, and to different boundary conditions [29]. More generally, topological codes can be defined by the cellulation of any n-dimensional manifold, not only a torus. For instance, topological codes have been defined on hyperbolic manifolds [30], Möbius strips [31], and more. Techniques from algebraic topology then allow to deduce the property of a given topological code from the topological properties of its underlying manifold [32].

2.2.6 Decoding the surface code

As we have seen, errors in the surface code come in chains, creating a pair of excitations at the two ends of each chain. The decoding problem consists in taking all those pairs as input, and returning a correction operator that matches them. As illustrated in Figure 2.3, applying such a correction has two possible consequences: either the combination of the errors and the correction forms a trivial loop, or it goes around the torus. In the first case, we obtain a stabilizer and the original state is recovered. In the second case, we obtain a logical operator, thereby introducing a logical error.

For a given noise model, the optimal decoding strategy—called maximum



(a) Error syndrome **(b)** Equivalent graph problem

Figure 2.4: Illustration of the minimum-weight perfect matching decoder. (a) Observed syndrome, with excitations in purple. (b) Corresponding graph, where each edge is weighted by the minimum distance between two excitation. The solution, which has a cost of 5, is highlighted in red.

likelihood decoding—consists in finding the coset of errors $\mathbf{e} \cdot \mathcal{S}$ (modulo the stabilizer set $\mathcal{S}$) that fits the syndrome and has the highest probability [8]:

$$\max_{\mathbf{e}} P(\mathbf{e} \cdot \mathcal{S} | \mathbf{He} = \mathbf{s}) \tag{2.12}$$

where $\mathbf{s}$ is the syndrome and $\mathbf{H}$ the parity-check matrix. Using this decoder on the surface code with a depolarizing noise model gives an optimal threshold of 18.9% [33]. However, the maximum-likelihood decoder is known to be NP-Hard, as the number of possible errors increases exponentially with the size of the code, and calculating the maximum involves in general to go through all of them.

Many alternative decoders with a lower computational complexity have been proposed in the literature. One of the most commonly used, known as the minimum-weight perfect matching (MWPM) decoder, solves the problem of pairing excitations [8]. If we define the weight of an error as the number of qubits it affects, the MWPM decoder works under the assumption that the most likely pairing is the one that minimizes the total error weight. With

this assumption, we can see the decoding problem as a graph problem. Let us construct a complete graph, where nodes are excitations and an edge between two excitations s_i and s_j is given by the L_1 distance between s_i and s_j. The objective is then to match all the nodes such that the sum of the selected edges is minimal, as illustrated in Figure 2.4. This problem is well-known in the graph theory community and can be solved in polynomial time using for instance the Edmond's algorithm [34]. Despite this efficiency gain compared to the maximum-likelihood decoder, this algorithm has multiple drawbacks: its complexity scales as $\mathcal{O}(N^3)$, which is still too slow to be used in practice, and it has a relatively low threshold compared to other decoders [35].

Apart from the MWPM decoder, several other decoders compete as fast and accurate decoders for the surface code. Some decoders are directly trying to approximate the likelihood function, using for instance Monte-Carlo algorithms [36, 37, 38] or belief propagation [39, 40, 35, 41, 42]. Some are exploiting the locality and symmetries of the surface code by using an algorithm based on the renormalization group [43, 44]. Some are using artificial neural networks as a way to learn the mapping syndrome $\rightarrow$ errors from data [45, 46, 47, 48]. Finally, other graph formulations of the decoding problem have been proposed, and in particular one based on the union-find data structure, which allows fast decoding of the surface code [49, 50].

2.3 Biased noise and stabilizer deformations

2.3.1 Biased noise

Until recently, most quantum error-correcting codes were evaluated under simple noise models where the probability of X and Z errors are the same. However, experiments have been showing over the years that those two types of error can have a highly different probability in practice, or in other words, noise is biased. For instance, phase-flip errors occur with a higher probability than bit-flip errors in superconducting qubits [13, 14, 15], trapped ions [16] and quantum dots [17].

As discussed in Section 2.2.2, any Pauli noise model can be parametrized by a vector (r_X, r_Y, r_Z) where $r_X, r_Y, r_Z \geq 0$ and $r_X + r_Y + r_Z = 1$. Those configurations form a simplex, represented in Figure 2.5. For example, the point $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ at the center represents a depolarizing noise model (no bias), while the three corners $(1,0,0)$, $(0,1,0)$, $(0,0,1)$ represent fully-biased noise models (toward X, Y or Z respectively).

Following the convention of [20], we parametrize biased noise on each axis by a single parameter η, called the **bias ratio**. For instance, for Z-biased noise, we define

$$\eta_Z = \frac{r_Z}{r_X + r_Y} \tag{2.13}$$

such that $\eta_Z = 0.5$ when there is no bias, and $\eta_Z \to \infty$ for full Z-bias.

Many modifications of existing error-correction procedures have been proposed to tackle biased noise, often leading to a significant increase of the threshold in this regime [22, 51, 52, 21, 19, 20, 53]. For instance, in Ref. [52], the authors propose to adapt the frequency rates of X and Z corrections to the rates of X and Z errors, correcting more frequently the most common error type. In Refs. [54, 51, 55], asymmetric codes are proposed to correct X and Z errors in a different way, often with a larger distance for the most common error type.

More recently, simple changes to the surface code stabilizers have been proposed to improve its performance under biased noise. For instance, Ref. [19] proposed to change the vertex stabilizers in the Z-biased regime, replacing Z by Y operators, such that Z errors activate both types of stabilizers. The idea behind this transformation is that, when Z errors are predominant, more information becomes available for decoding them. We will call such modifications of the surface code **stabilizer deformations**. In Ref. [20], the authors show that this new surface code has a threshold of 50% for pure Z noise, and that taking the two dimensions of the lattice to be coprime lead to large improvement of the logical failure rate. In a more realistic regime where Z errors are 100 times more prominent than X errors, Ref. [21] shows that the

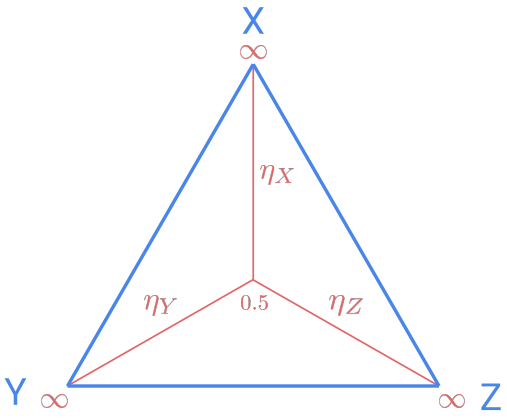


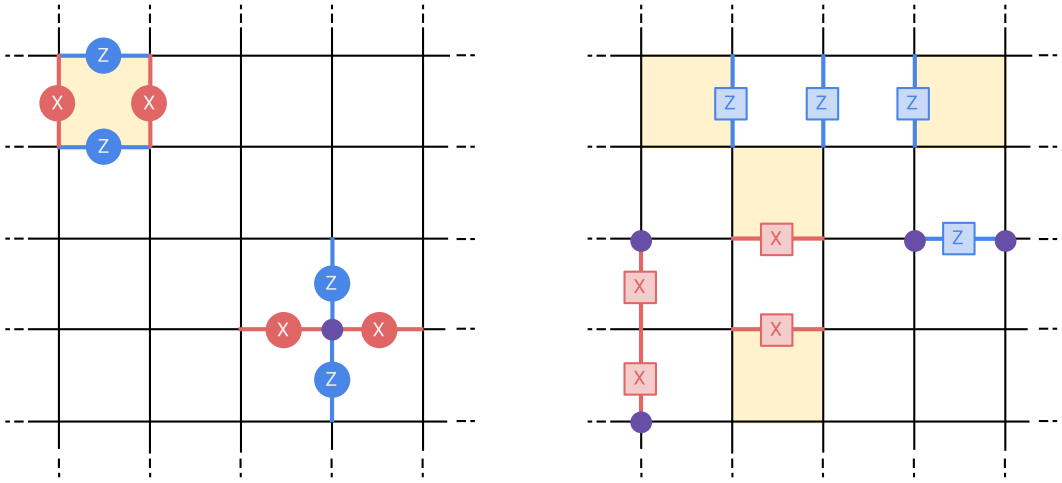
Figure 2.5: Simplex of Pauli noise models. The center corresponds to depolarizing noise, while the three corners correspond to fully-biased noise models. Each red line parametrizes biased noise in one of the three directions, using a scalar η varying from 0.5 to ∞.

threshold can be improved by 5% compared to the undeformed surface code. The authors obtain those results using a new decoder tailored to this new code. Finally, a different deformation of the surface code, called the XZZX surface code, has recently been proposed, leading to an even higher gain in performance [22]. Since the work of this thesis is highly inspired by this new deformation, we will talk about it in detail in the next section.

2.3.2 The XZZX deformation

The XZZX surface code consists in applying a Hadamard operation on a chosen axis of all the stabilizers. The resulting stabilizers, represented in Figure 2.6a, are each made of two X and two Z operators. As a consequence, errors are detected in different way as the original surface code, as shown in Figure 2.6b. Plaquette stabilizers detect horizontal chains of Z errors and vertical chains of X errors, while vertex stabilizers detect horizontal chains of X errors and vertical chains of Z errors. Therefore, what used to be arbitrary chains in the 2D surface code are now one-dimensional chains.

Let us see the consequences for pure Z noise: vertical and horizontal errors are now detected with different types of stabilizers, and are therefore decoded



(a) Vertex and face stabilizers **(b)** Detection of X and Z errors

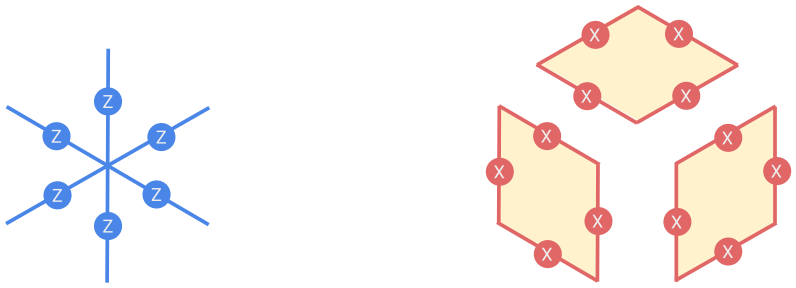
Figure 2.6: **(a)** XZZX surface code. Plaquette and vertex stabilizers are defined with a combination of X and Z. **(b)** As a result, X errors only lead to vertical chains and Z errors to horizontal chains, making the decoding process easier under biased noise

independently. Moreover, pairs of excitations now lie on a straight line, considerably simplifying the decoding problem. In other words, for pure Z noise, decoding the XZZX surface code is equivalent to decoding $j+k$ repetition codes (where j and k are the dimensions of the lattice). Since repetition codes have a 50% threshold [20], it follows that the XZZX code has a threshold of 50% under pure Z noise.

2.3.3 Deformed noise models

There is an alternative way to look at deformations, that tends to simplify both their theoretical analysis and their numerical implementation. Instead of applying deformations at the stabilizer level, we can apply them at the noise level. For example, in the XZZX surface code, applying a Hadamard to each stabilizer on the vertical axis is equivalent to applying a Hadamard to each vertical error. More formally, a Pauli noise channel given by

$$\mathcal{N}(\rho) = (1-p)\rho + p(r_X X\rho X + r_Y Y\rho Y + r_Z Z\rho Z) \tag{2.14}$$



(a) Vertex stabilizer **(b)** Face stabilizers

Figure 2.7: Stabilizers of the 3D toric code with a cubic lattice

on each qubit, stay the same on every horizontal qubits but becomes

$$\mathcal{N}(\rho) = (1-p)\rho + p(r_X Z\rho Z + r_Y Y\rho Y + r_Z X\rho X) \tag{2.15}$$

on every vertical qubits.

Theoretically, it means that we can analyze the XZZX surface code just like the original surface code, but with a modified noise model. In practice, it means that we can reuse existing numerical implementations of the surface code, simply changing the noise model when evaluating the threshold.

2.4 3D toric codes

The 3D toric code is the generalization of the surface code on a three-dimensional torus. Many types of lattices have been proposed for the 3D toric code, each coming with a different set of stabilizers [56, 57]. In this work, we consider the two most common lattice types for the 3D toric code: the cubic and the rhombic lattices.

2.4.1 Cubic lattice

The construction of the 3D toric code with a cubic lattice follows a similar pattern as the surface code: we place qubits on the edges of a 3D grid with period boundary conditions. We then define two types of stabilizers: vertex stabilizers, defined at each vertex by six Z operators around that vertex, and face stabilizers, defined at each face by four X operators around that face [56]. The resulting stabilizers are represented in Figure 2.7.

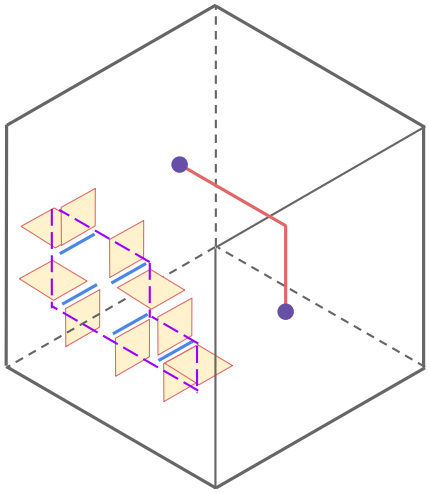
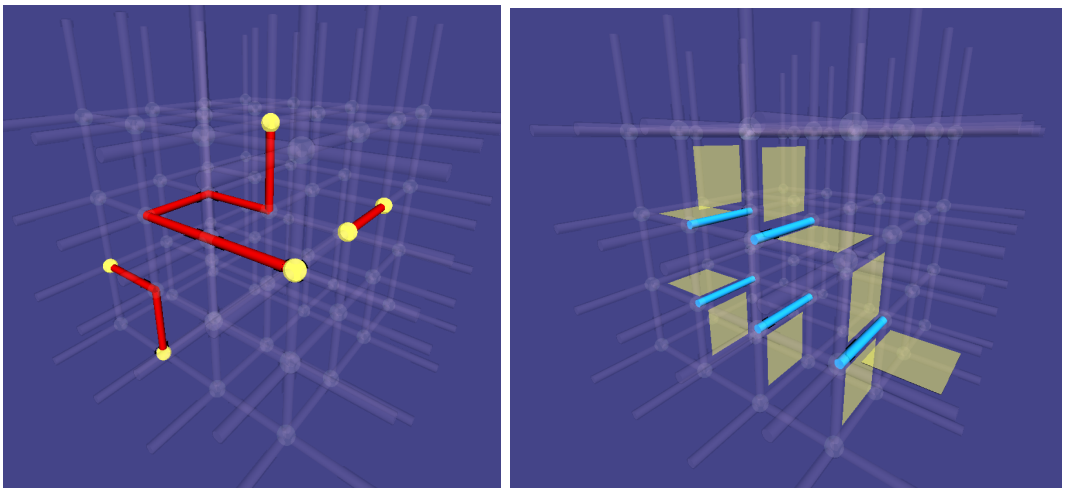


Figure 2.8: Error detection in the 3D toric code with a cubic lattice: chains of X errors still activate stabilizers at the boundary of each chain, but membranes of parallel Z errors activate a loop of stabilizers around the chain. See also Figure 2.9 for a different graphical representation of those errors.

Let us observe what happens when errors occur. Similarly to the surface code, chains of X errors are detected by vertex stabilizers at their two extremities: they form the so-called **point sector**. On the other hand, Z errors are detected differently: membranes of parallel Z errors are detected by face stabilizers forming a loop around that membrane, as represented in Figures 2.8 and 2.9: they form the so-called **loop sector**.

The logical operators of the cubic code are represented in Figure 2.10. There are three logical X and three logical Z, one for each axis of the cube. The logical X are the chains going around the torus, while the logical Z are membranes forming a cross-section of the torus. Therefore, if all the dimensions of the lattice are equal to L, the cubic code is a $[3L^3, 3, L]$-code. However, contrary to the surface code, the Z-distance and the X-distance are this time different: while the X-distance is L, the Z-distance is higher and equal to L^2. For this reasons and the fact that Z errors are detected by four faces (instead of two vertices for X errors), we can expect the 3D toric code to perform better under Z-biased noise than X-biased noise.



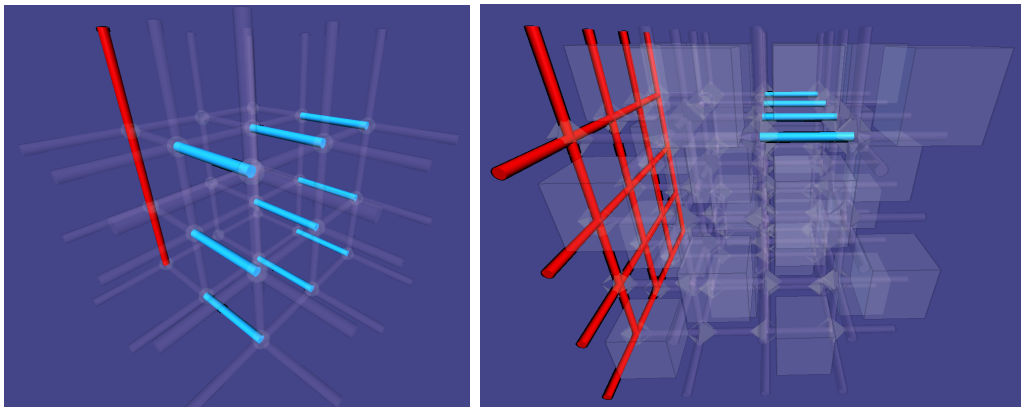
(a) Vertex stabilizer **(b)** Face stabilizers

Figure 2.9: Stabilizers of the 3D toric code with a cubic lattice: chains of X errors (red) are detected at the boundary vertices, and chains of Z errors (blue) are detected by a loop of faces

The reader might wonder what makes the 3D toric code attractive. Indeed, 3D quantum architectures are notoriously hard to build, and the global distance of the code has a worst scaling than the surface code. However, its main advantage can be seen when trying to design a fault-tolerant gate set for the code [11]. Contrary to the surface code which requires resource-intensive techniques such as magic state distillation to obtain a universal gate set, universality can be reached at a lower cost (using transversal gates) with the 3D code. There is therefore a trade-off between universality and the other characteristics of the two codes. This trade-off has been studied in detail in Ref [12], showing that 3D codes do not have a clear advantage compared to 2D codes when taking everything into account. The hope of our project is to lead to a reevaluation of this cost when the noise is biased.

2.4.2 Rhombic lattice

The rhombic lattice can be constructed by placing qubits on the edges of a 3D grid with even dimensions. As usual, we build two types of stabilizers: the cube stabilizers, defined on every other cubes of the lattice by placing Z



(a) Cubic code **(b)** Rhombic code

Figure 2.10: Logical operators in cubic and rhombic lattices. **(a)** In the cubic lattice, the three X logical operators are formed by chains of X operators going around the 3D torus along one of the three axis. The three Z logical operators consist in membranes of parallel Z operators. **(b)** In the rhombic lattice, the three X logical operators are formed by membranes of adjacent X operators. The three Z logical operators are made of chains of parallel Z operators.

operators on the twelve edges around that cube, and the triangle stabilizers, defined on each vertex of the remaining cubes as the three X operators around that vertex inside the cube. Those stabilizers are represented in Figures 2.11 and 2.12. The name *rhombic* comes from an equivalent construction of this code, in the so-called rectified picture, where qubits are placed on vertices, forming an actual rhombic dodecahedron [56].

The detection of errors in the rhombic lattice is illustrated in Figure 2.13. Chains of parallel X errors are detected by cubes at their extremities (they form the **point sector**), while membranes of adjacent X errors are detected by triangles around the membrane (they form the **loop sector**). As for the cubic lattice, there are three Z and three X logical operators, represented in Figure 2.10b. Z logical operators are made of 2D membranes crossing the torus and X logical operators are made of chains of parallel operators. Therefore, if L is the size of the lattice, the rhombic code is a $[3L^3, 3, L]$-code, similarly to

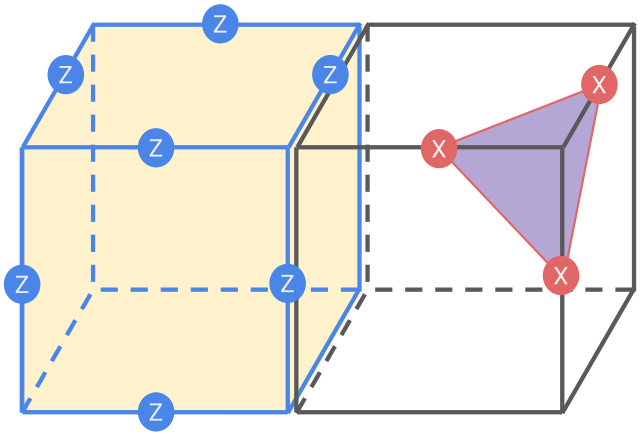


Figure 2.11: Stabilizers of the rhombic lattice. One of every two cubes (yellow) represents a cube stabilizer, with Z operators on every edge. The remaining cubes (white) have a triangle stabilizer at each vertex, made of X operators on the three adjacent edges. See Figure 2.12 for a graphical representation of the whole lattice with the stabilizers

the cubic case but with a higher Z-distance of $2L^2$.

The rhombic lattice is attractive in the highly Z-biased case, where a lot of information is available for decoding and the distance is high. However, it comes at the cost of having stabilizers made of twelve operators, which can be hard to build in practice.

2.5 Decoders for 3D toric codes

While X and Z errors in the 2D surface code are dual to each other, errors come in two different fashions in 3D. Z errors, detected by vertices, lead to a point-like syndrome—two excitations at the ends of each chain of errors—similarly to the 2D case. However, X errors, detected by faces, lead to a loop-like syndrome: membranes of parallel errors are surrounded by excited faces, as illustrated in Figure 2.9. Therefore, while 2D decoders can often readily be generalized for decoding the point-sector in 3D, new decoders need to be considered for the loop sector. In this thesis, we considered two particular decoders for the loop sector: the sweep decoder (combined with matching in the point sector) [23, 56] and the BP-OSD decoder [41].

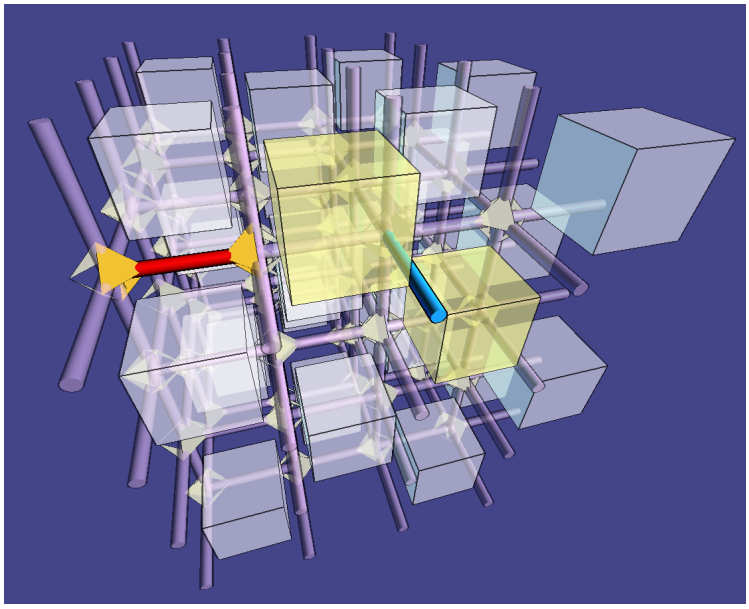
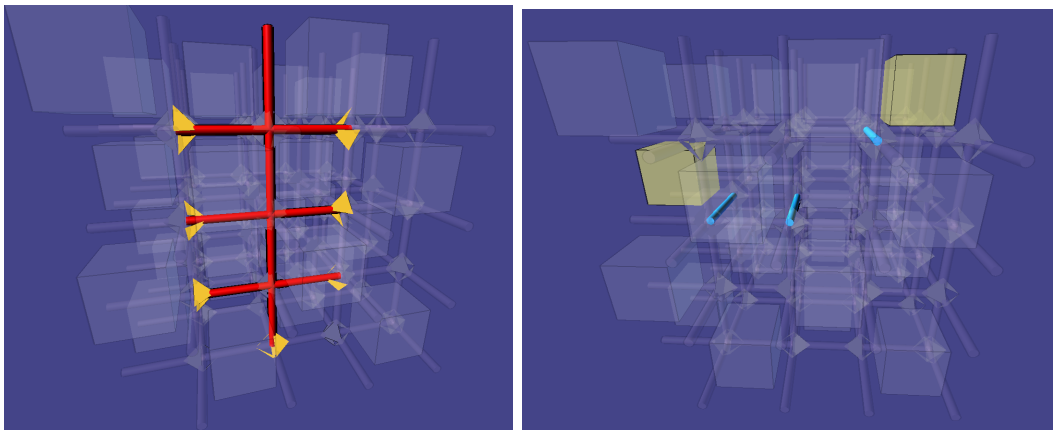


Figure 2.12: 3D graphical representation of the rhombic lattice



(a) Detection of X errors with triangles **(b)** Detection of Z errors with cubes

Figure 2.13: Error detection with the rhombic lattice

2.5.1 SweepMatch decoder

The sweep decoder is a special type of cellular automaton decoder, meaning that is an iterative algorithm in which a set of operators is applied at each step based on the current syndrome and a particular rule, with the objective of eliminating every excitations after a finite number of steps. The sweep decoder is based on a variant of Toom's rule called sweep rule [23]. We start by choosing an arbitrary spatial direction, defined by a 3D vector $\vec{\mathbf{v}}$, called the sweep direction, with the only condition that it is not parallel to an edge of the lattice. We then apply the following rule at each iteration, and for all the vertices simultaneously:

1. Find the three oriented edges $\vec{\mathbf{e}}_1, \vec{\mathbf{e}}_2, \vec{\mathbf{e}}_3$ pointing away from the vertex and in the same direction as $\vec{\mathbf{v}}$, i.e. such that $\vec{\mathbf{e}}_i \cdot \vec{\mathbf{v}} > 0$.

2. Find the three faces f_1, f_2, f_3 defined by the three pairs of edges we can form.

3. If two of the faces are excited, apply a Z operator to the intersecting edge. If three faces are excited, apply a Z operator to a random edge among $\vec{\mathbf{e}}_1, \vec{\mathbf{e}}_2, \vec{\mathbf{e}}_3$. Otherwise, do nothing

4. Stop if no excitation remains or if a maximum number of iterations has been reached (in which case the decoder has failed).

When combined with a minimum-weight perfect-matching decoder (described in Section 2.2.6) for the point sector, we call the resulting decoder SweepMatch. We will now describe an alternative to SweepMatch that can be used to decode the 3D toric code, and more generally any quantum LDPC code: the BP-OSD decoder [41].

2.5.2 BP-OSD decoder

The belief propagation (BP) decoder, also called *message-passing decoder*, is one of the most commonly used decoders for classical LDPC codes, its high performance having contributed to a renewed interest for LDPC codes in the

1990s [27]. It consists in iteratively calculating the probability that each bit has an error, by locally passing messages between parity-checks and data bits. While there is a straightforward generalization of the BP decoder to quantum LDPC codes, the fact it is not more widely used is due to several quantum-specific issues. In particular, the degeneracy problem—the idea that many errors of equal weight can yield a given syndrome—and the presence of short loops in the Tanner graph of quantum codes can cause a failure of the BP algorithm. To circumvent those problems, Ref. [41] considered the combination of the BP algorithm with a second decoder, the *ordered-statistics decoder* (OSD), in case of failure of the BP procedure. The OSD takes the probabilities calculated by the BP algorithm as input, and returns a valid correction that fits the syndrome. The BP-OSD decoder has recently been used to successfully decode a 3D homological product code, exhibiting a much higher threshold than all the other decoders tested in their paper (including SweepMatch) [24].

In this section, we will start by describing the belief propagation algorithm and how it can be used to decode quantum codes. We will then study its issues, and in particular the degeneracy problem, and finally see how the OSD procedure can make use of the BP output to give a valid decoding solution.

2.5.2.1 Belief propagation algorithm

Belief propagation is a special type of *inference* algorithm: given a syndrome $\mathbf{s}$, we want to infer the probability $P(\mathbf{e}|\mathbf{s})$ of each error $\mathbf{e}$. Since an N-bit message can have 2^N possible errors $\mathbf{e}$, just storing this probability would require an exponential space. Therefore, we simplify this problem by slightly altering our goal. Instead of inferring the full probability, we will aim to compute each marginal probability

$$P(e_n|s) = \sum_{e_k, k\neq n} P(e_1, ..., e_N|\mathbf{s}) \tag{2.16}$$

Applying Baye's rule to each term of the sum gives

$$P(\mathbf{e}|\mathbf{s}) = \frac{P(\mathbf{s},\mathbf{e})}{P(\mathbf{s})} \tag{2.17}$$

$$\propto P(\mathbf{s},\mathbf{e}) \tag{2.18}$$

The joint probability $P(\mathbf{s},\mathbf{e})$ can be factored as

$$P(\mathbf{s},\mathbf{e}) = P(\mathbf{s}|\mathbf{e})P(\mathbf{e}) \tag{2.19}$$

$$= \mathbb{1}\left[\mathbf{H}\mathbf{e} = \mathbf{s}\right] P(\mathbf{e}) \tag{2.20}$$

$$= \prod_m \mathbb{1}\left[s_m = \sum_{n\in\mathcal{N}(m)} e_n\right] \prod_n P(e_n) \tag{2.21}$$

Therefore, we obtain for the marginal probability:

$$P(e_n|s) \propto \sum_{e_k:k\neq n} \prod_m \mathbb{1}\left[s_m = \sum_{n\in\mathcal{N}(m)} e_n\right] \prod_n P(e_n) \tag{2.22}$$

The normalization constant is often not so important in practice, as it can either be calculated at the end or eliminated completely using probability ratios.

The idea of belief propagation is to calculate the probability $P(e_n|\mathbf{s})$ by exploiting the sum-product structure of Eq. (2.22). The intuition is that a sum of products can be factorized into a product of sums, which can often be calculated using a reduced number of operations. For instance, in

$$ac + ad + bc + bd = (a+b)(a+c), \tag{2.23}$$

calculating the LHS involves seven operations (four products and three additions), while the RHS involves only three operations (one product and two additions).

To understand how the BP algorithm works, we first need to introduce the notion of *factor graph*. Let us consider a factorizable function over N variables

$$f(x_1,...,x_N) = \prod_j f_j(\mathbf{x}_j), \tag{2.24}$$

where each $\mathbf{x}_j$ is a subset of the variables $\{x_i\}$. Its factor graph consists in N data nodes, representing the N variables x_i, and M factor nodes, representing

the M functions f_j. There is an edge between node i and node j if f_j depends on x_i. For instance, the factor graph of $P(\mathbf{e},\mathbf{s})$, corresponding to Eq. 2.21, has a data node for each bit of the message and a factor node for each parity check, with an edge between s_j and e_i when s_j depends on e_i. In other words, the factor graph of the decoding problem is the same as the Tanner graph of the code.

Belief propagation then works by iteratively propagating messages between check nodes and data nodes. It works as follow [27]:

- **Initialization**: the messages from data nodes e_n to check nodes s_m, that we denote $m^x_{n\to m}$, are initialized to the prior probabilities $P(e_n = x)$, for $x \in \{0,1\}$.

- **Check to data**: we send the following messages from check nodes to data nodes

$$m^x_{m\to n} = \sum_{e_{n'}:n'\in\mathcal{N}(m)\setminus n} \mathbb{1}\left[s_m = x + \sum_{n'\in\mathcal{N}(m)\setminus n} e_{n'}\right] \prod_{n'\in\mathcal{N}(m)\setminus n} m^{e_{n'}}_{n'\to m} \tag{2.25}$$

  where $\mathcal{N}(m)$ denotes all the data nodes that are connected to s_m in the Tanner graph.

- **Data to check**: we send the following messages from data nodes to check nodes:

$$m^x_{n\to m} = \alpha_{nm} P(e_n = x) \prod_{m'\in\mathcal{N}(n)\setminus m} m^x_{m'\to n} \tag{2.26}$$

  where α_{nm} is a normalization constant and $\mathcal{N}(n)$ denotes all the check nodes that are connected to e_n in the Tanner graph.

- **Calculation of the probabilities**: we then compute an approximation of the probability that each error is equal to 1 as

$$p_n = \alpha_n P(e_n = x) \prod_{m\in\mathcal{N}(n)} m^1_{m\to n} \tag{2.27}$$

where α_n is a normalization constant. From those probabilities, we can compute a correction as

$$\tilde{e}_n = \mathbb{1}\left[p_n > \frac{1}{2}\right] \tag{2.28}$$

- **Stop**: when the syndrome equation $\mathbf{H}\tilde{\mathbf{e}} = \mathbf{s}$ is verified, or after a maximal number of iterations has been reached (in this case, we return that the BP decoder has failed).

While convergence is guaranteed when the factor graph is a tree, the presence of loops can hinder those guarantee [27]. In those cases, the BP algorithm can however still be used as a powerful heuristic.

The algorithm described above is sometimes called *sum-product algorithm* in contrast with one of its variant called *min-sum algorithm* and widely used in the context of decoding. In the min-sum version of the BP algorithm, we want to directly compute the error that has the highest probability

$$\max_{\mathbf{e}} P(\mathbf{e}|\mathbf{s}). \tag{2.29}$$

This maximization can be decomposed as

$$\max_{e_n} \max_{e_k, k\neq n} P(e_1, ..., e_N|\mathbf{s}). \tag{2.30}$$

The goal of the min-sum algorithm is to evaluate the first part:

$$\max_{e_k, k\neq n} P(e_1, ..., e_N|\mathbf{s}). \tag{2.31}$$

Replacing $P(\mathbf{e}|\mathbf{s})$ by $-\log(P(\mathbf{e}|\mathbf{s}))$, we get the following new problem:

$$\min_{e_k, k\neq n} \sum_m -\log\left(\mathbb{1}\left[s_m = \sum_{n\in\mathcal{N}(m)} e_n\right]\right) + \sum_n -\log\left((P(e_n)\right). \tag{2.32}$$

In order words, we have replaced the *sum* by a *min* and the *product* by a *sum*. Since the BP algorithm only uses the distributivity property of the product over the sum, it can be generalized to any field. In particular, $(\mathbb{R}, +, \min)$ is a field since we have the distributivity property

$$a + \min(b, c) = \min(a + b, a + c) \tag{2.33}$$

and the sum-product algorithm described above can therefore directly be transformed into a min-sum algorithm with those new operations.

A more detailed account of belief propagation decoding is given in Ref. [27], and the exact version of the min-sum algorithm we implemented is given in Ref. [41]. While the generalization of the BP algorithm to the quantum case is straightforward [39, 40], care needs to be taken to avoid a quantum-specific issues, as we will see next.

2.5.2.2 The degeneracy problem

Applying the BP algorithm to quantum codes causes two important problems:

1. The Tanner graph of quantum LDPC codes inevitably contains 4-loops, as shown in Ref. [40], while the BP algorithm is only guaranteed to converge to the correct probability distribution for trees. However, this issue is not too problematic in practice, as our goal is not to infer the exact probability distribution but only a valid correction that has a high probability of being correct.

2. Toric codes contain many degeneracies: some syndromes can be caused by many errors of equal weights. Whenever a degenerate syndrome occurs, the BP algorithm might assign an equally high probability to all the possibilities, leading to a meaningless correction where all the possible low-weight errors are applied at the same time. Simple examples of such phenomenon are given in Ref. [40].

Several solutions to the degeneracy problem have been proposed in the literature, such as breaking the degeneracy with random noise [40], using a neural network to learn the BP procedure, with a loss function tailored to avoid degeneracies [58], adding memory effects [42], or complementing the BP decoder with a second decoder such as the Ordered-Statistics Decoding (OSD) [41]. In this work, we used this last solution, that we will describe now.

2.5.2.3 Ordered-Statistics Decoding

The decoding problem can be formulated as an inverse problem: for a syndrome $\mathbf{s}$, we want to find an error $\mathbf{e}$ that solves the equation

$$\mathbf{He} = \mathbf{s} \tag{2.34}$$

However, this system of equations is highly underdetermined: many combinations of errors can fit a given syndrome. The idea of the Ordered-Statistics Decoder (OSD) is to fix some values of $\mathbf{e}$ at a given value (for instance 0), such as the system applied to the remaining variables is full-rank. To choose the subset of $\mathbf{e}$ that is kept in the system, OSD takes as input a prior probability on each error e_i. In the BP-OSD decoder, this prior probability corresponds to the output of the Belief Propagation algorithm.

The OSD algorithm then works as follow [41]:

1. Order the columns of $\mathbf{H}$ by decreasing probability of error.
2. Select the r independent columns of $\mathbf{H}$ with the highest probability, where r is the rank of $\mathbf{H}$.
3. Select r independent rows from the new matrix, obtaining a full-rank square matrix $\mathbf{H}_\mathrm{r}$.
4. Solve the reduced system

$$\mathbf{H}_\mathrm{r}\mathbf{e}_\mathrm{r} = \mathbf{s}_\mathrm{r} \tag{2.35}$$

   where $\mathbf{e}_\mathrm{r}$ contains the r selected components of the error (ordered by probability), and $\mathbf{s}_\mathrm{r}$ the r selected components of the syndrome.
5. Set the remaining components of $\mathbf{e}$ (not contained in $\mathbf{e}_\mathrm{r}$) at 0.

Higher-order versions of OSD that set the remaining components in Step 5 to values other than zero have been proposed [41], but we have not implemented them in this thesis.

Chapter 3

Methods and Results

3.1 Visualizing a 3D code

In order to gain intuition on 3D codes and their deformations, I developed a 3D graphical interface that allows to easily manipulate different 3D codes, noise models and decoders. This interface is represented in Figure 3.1. It is developed in Javascript using the library Three.js and can run on any modern browser. It communicates with a Python backend—to construct the different parity-check matrices and perform decoding—written with the library Flask. We outline here the main features of the interface:

- The user can select a lattice type (cubic or rhombic) and lattice size. A 3D representation of the code is then displayed on screen. The user can zoom in/out, rotate and move around the code.

- Left-clicking (resp. right-clicking) on an edge creates a Z (resp. X) error, colored differently depending on the error type (X, Y or Z). The excited stabilizers are then colored in yellow.

- The user can choose between different noise models (currently X/Z biased noise and depolarizing), error probability, and whether it is deformed or not. Pressing ”R” on the keyboard then generates random errors.

- The user can choose between different decoders (currently BP-OSD and

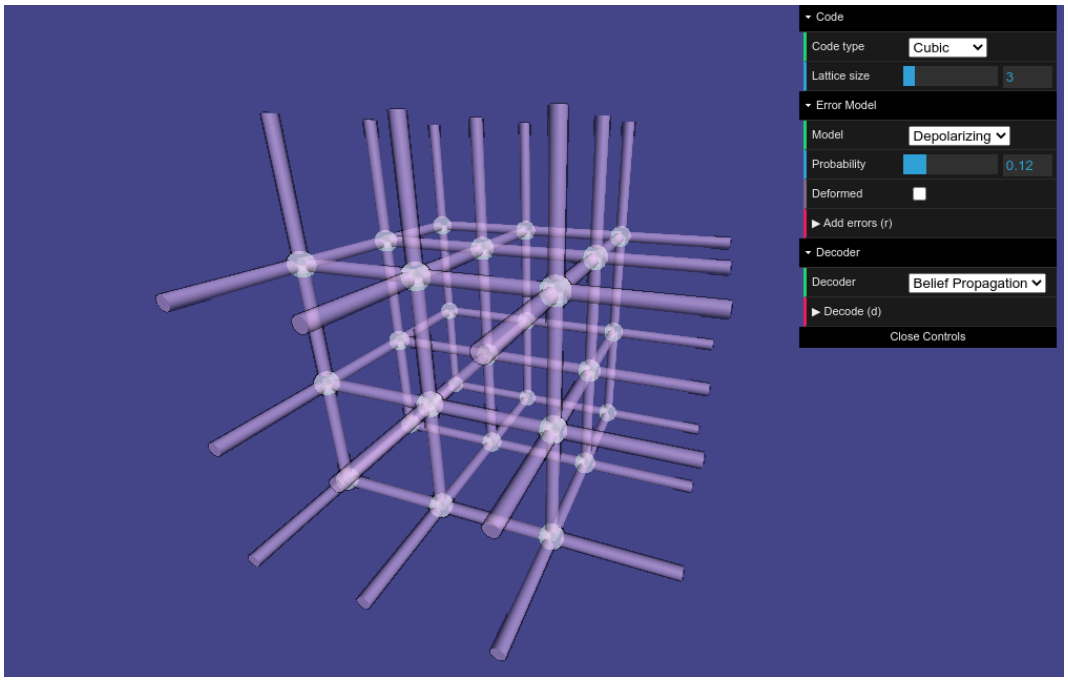
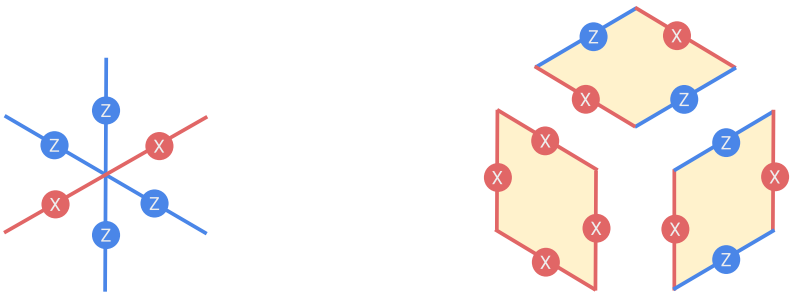


Figure 3.1: Graphical interface developed in the context of this thesis to manipulate the different 3D toric codes

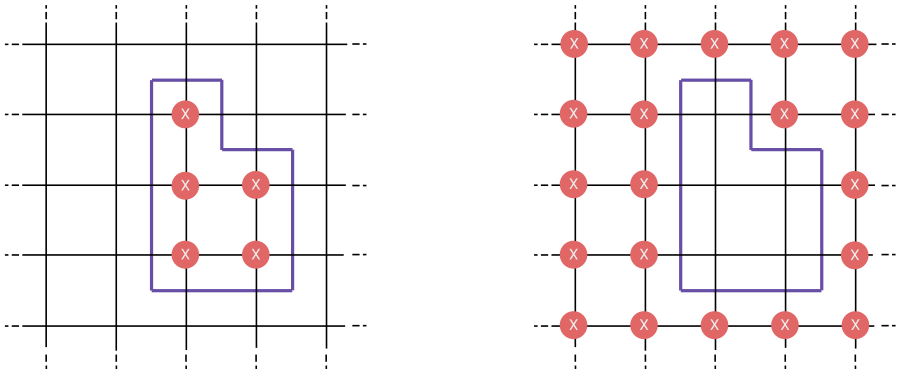
SweepMatch) and run the decoding process by pressing ”D” on the keyboard.

During the project, this interface helped in several ways. First, it significantly simplified the debugging process for the implementation of both the lattices and the decoders. Then, it allowed us to gain intuition on the proposed deformations, by visually showing the stratification of 3D errors into lines and planes. In some future work, it could potentially help develop new ideas of deformations. Finally, it served as a tool to explain my project to other people. We consider putting this interface online as a supplementary material to help understand the paper, or more generally as an educational tool to learn about quantum error correction.



(a) Vertex stabilizer **(b)** Face stabilizers

Figure 3.2: Stabilizers of the deformed 3D toric code with a cubic lattice, obtained by applying a Hadamard operator to the usual stabilizers on a chosen axis.



(a) First decoding solution **(b)** Second decoding solution

Figure 3.3: Loop decoding in 2D. The plane represents a cross-section of the 3D cubic code, and each vertex represents a qubit normal to the plane. Membranes of X errors normal to the plane form a loop syndrome (purple), with two decoding possibilities: **(a)** correction inside the loop ; **(b)** correction outside the loop

3.2 Deformations of the 3D toric code

3.2.1 Cubic lattice

Similarly to the XZZX code, the deformations we designed for the 3D toric code consist in applying a Hadamard operation to all the stabilizers on a chosen axis, as can be seen in Figure 3.2. The idea behind this construction is again to reduce the dimensionality of the decoding problem in fully-biased noise models. Indeed, let us observe what happens in case of pure Z noise: vertex stabilizers are activated at the extremities of straight chains of errors along the deformed axis, while face stabilizers are activated around errors on the two other axes. Therefore, one axis can be decoded independently of the two others, thereby simplifying the decoding problem. We can also notice that the decoding problem on the deformed axis becomes equivalent to a repetition code. We can therefore expect the point sector to have a threshold of 50%, while the difficulty of the decoding problem in the loop sector is unknown a priori. We will determine the loop sector threshold numerically in Section 3.4.

For pure X noise, vertex stabilizers are activated at the end of chains lying on the 2D planes spanned by the undeformed axes, making the decoding problem equivalent to that of a 2D toric code. On the remaining axis, membrane of errors normal to each plane are surrounded by loops. The decoding problem therefore consists in decoding loops in 2D, as represented in Figure 3.3. This can be done with a threshold of 50%.

3.2.2 Rhombic lattice

For the rhombic lattice, we apply the same deformation: a Hadamard operation on a chosen axis. The cube stabilizers are now made of eight X and four Z operators, while the triangle stabilizers each have one X and two Z operators. The new decoding problem has a reduced dimensionality in exactly the same way as for the cubic code.

3.3 Simulation method

3.3.1 Experimental setup

The goal of this work is to evaluate the threshold of the two types of 3D codes (cubic and rhombic) for different biased noise models, and compare the original code with its deformed version, as well as the BP-OSD decoder with the SweepMatch decoder. We parametrize the bias with the bias ratios η_X, η_Y, and η_Z described in Section 2.3, and take $\eta \in \{0.5, 1, 5, 10, 30, 100, \infty\}$ for each direction.

To evaluate the threshold, we simulate both codes for lattice sizes $L \in \{4, 6, 8, 10\}$ and physical error rates between $p = 0.05$ and $p = 0.5$, with a step size of 0.05. From those data, we build the so-called **crossover plots**, which give the logical error rate in function of the physical error rate for each lattice size. The threshold can then be read as the intersection point between the curves. To extract this crossover point from the figure, we use a technique called finite-size scaling analysis, which consists in fitting a curve parametrized by the threshold to the data. In this project, we use the method described in [21] where

$$y = A + Bx + Cx^2 \tag{3.1}$$

with y the logical error rate and

$$x = (p - p_{\text{th}})L^{1/\nu} \tag{3.2}$$

with p the physical error rate and p_{th} the threshold probability we are trying to evaluate. We then run an optimization procedure to get the parameters $(p_{\text{th}}, \nu, A, B, C)$ that fit the data the best, as measured by the mean-squared error. While this technique worked in most of our experiments, it failed to give a coherent result in a few crossover plots. In those cases, we reported the threshold as observed by manually looking at the curves. Before publishing the presented results, we will make sure that the finite-size scaling analysis work for all our data.

We ran all the simulations on the high-performance computing cluster Myriad available at UCL. We used one job per code type, noise model and physical error rate, allowing us to massively parallelize the computation. The results are reported in Sections 3.4 and 3.5. Apart from reporting the general thresholds and crossover plots, we also split each crossover plot into two plots—one for the loop sector and one for the point sector—since those can have very different thresholds. Finally, we report the evolution of the threshold with the bias ratio η, for different bias directions, on codes with and without deformations.

3.3.2 BP-OSD decoder: implementation details

As discussed before, the OSD algorithm has a scalability issue. In particular, two of its steps induce a high cost: the selection of r independent columns and the inversion of the resulting matrix. A naive implementation of the first step would consist in iterating over each column of the matrix, computing the rank (mod 2) of the selected matrix (all previously selected columns, plus the new column), and keeping the column if and only if this new matrix if full-rank. The inversion step could then be implemented using any variant of Gaussian elimination (such as LU decomposition).

However, we managed to gain a factor $\times 10$ in performance by noticing the presence of repeated calculations. Indeed, calculating the rank at each iteration (to select independent columns) involves repeating a Gaussian elimination process. Instead, we chose to first compute the reduced row echelon form of the full matrix $\mathbf{H}$ (sorted by most probable column). Independent columns can then be extracted directly from it: columns with their last non-zero element in the same row are dependent and one of them can be removed. Finally, the reduced row echelon form allows a fast inversion of the resulting matrix.

3.4 Results for the cubic lattice

We present in this section the results obtained for the 3D toric code with a cubic lattice, for different noise models, stabilizer deformations and decoders. Table

3.1 summarizes the different thresholds obtained in each configuration. Before diving into the detail of those results, we can notice three general phenomena from Table 3.1:

1. Going from no bias (depolarizing noise) to full Z bias results in a large increase of the threshold, regardless of the decoder and the deformation used in the simulation. On the other hand, going from no bias to full X bias only increases the threshold when the deformed code is used.

2. Deforming the stabilizers significantly improves the threshold in the biased noise regime: from **21.7%** to **36%** for pure Z noise decoded with the BP-OSD decoder, and from **2.9%** to **7.4%** for pure X noise

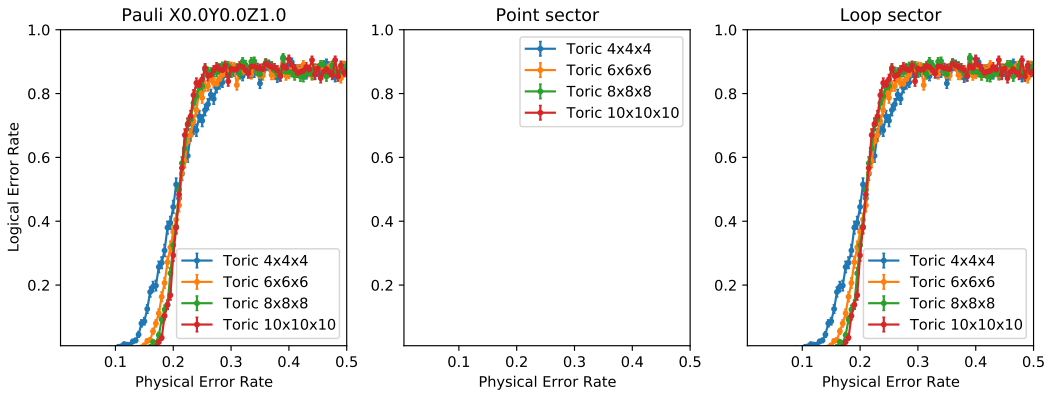
3. Changing from the SweepMatch decoder to the BP-OSD decoder yields a large increase of threshold in the pure Z noise model: from **14.7%** to **21.7%** for the undeformed code, and from **22.2%** to **36%** for the deformed code.

Therefore, combining the deformation with the BP-OSD decoder seems to maximize the performance in the biased noise regime. Let us now look at the crossover plots that allowed us to obtain those thresholds for those different regimes, before studying more precisely the effect of bias on the threshold. The remaining figures and thresholds presented in this section are produced using the BP-OSD decoder.

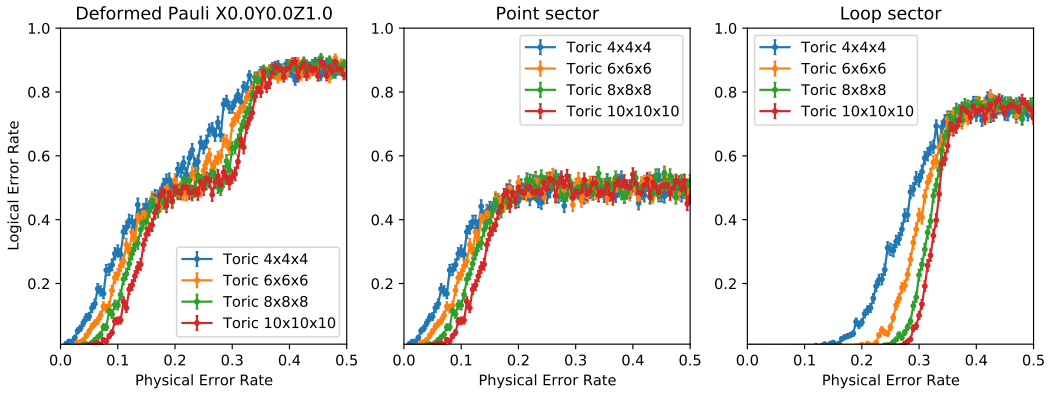
Noise model	Pure Z	Pure X	Depolarizing
Undeformed SweepMatch	14.7	2.9	4.4
Undeformed BP-OSD	21.7	2.9	4.3
Deformed SweepMatch	22.2	7.1	**4.7**
Deformed BP-OSD	$> \mathbf{36}$	**7.4**	4.2

Table 3.1: Thresholds (in %) for the cubic lattice, using different noise models, code deformations and decoders. The best result for each noise model is highlighted in bold. The case of pure Z noise with a deformed BP-OSD decoder could not be determined precisely with our data, but is probably more than 36%, as discussed in the main text.

3.4.1 Pure Z noise



(a) Undeformed code



(b) Deformed code

Figure 3.4: Crossover plot of the cubic code under pure Z noise, decoded with a BP-OSD decoder, for both the undeformed (a) and deformed (b) stabilizers.

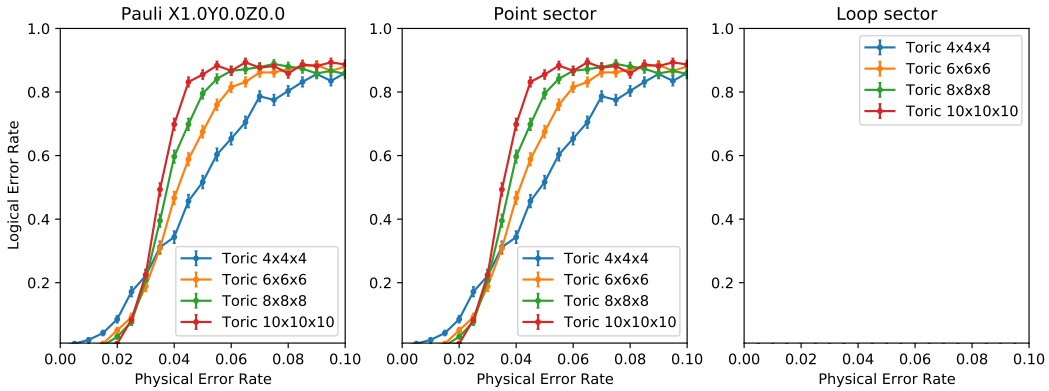
Figure 3.4 shows the crossover plot obtained for the 3D toric code with a cubic lattice, for both deformed and undeformed stabilizers, in the pure Z noise regime. We decomposed the logical error rates into the loop sector (errors triggering a face stabilizer) and the point sector (errors triggering a vertex stabilizer).

For the undeformed code, only the loop sector has errors due to the absence of X errors (which would trigger the point sector). The crossover point is clearly visible and occurs at around 21.7%.

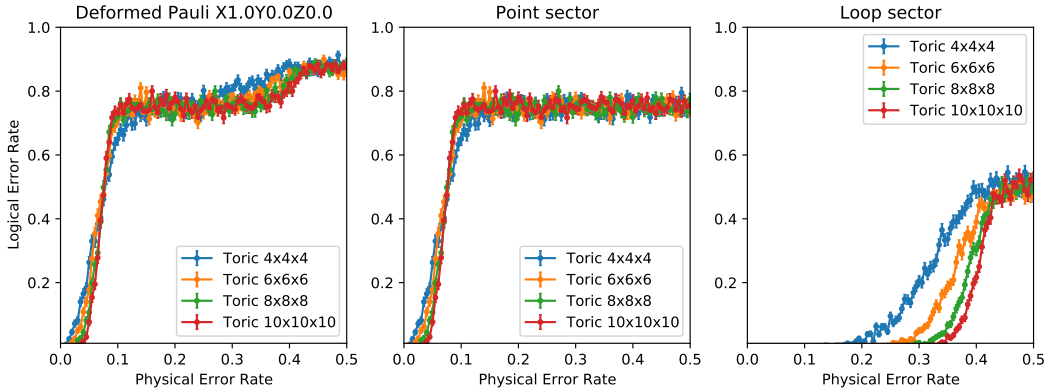
For the deformed code, both sectors are contributing to the total error.

As discussed in Section 3.2.1, the threshold for the point sector is in theory 50%, since the decoding problem is equivalent to that of a repetition code. We cannot resolve it in the plot due to a finite-size effect where the logical error saturates. Similarly, the threshold for the loop sector cannot be resolved precisely with the current data and is an example of failure of the finite-size scaling technique to determine the threshold. From the current data, the threshold seems to be between 36% and 50%, and we can hope that adding more lattice sizes to the simulation would help resolve it. The threshold could also be 50% and more theory will be needed to understand why. In any case, we observe that deforming the code leads to a dramatic increase of the threshold, even though its exact magnitude still needs to be determined.

3.4.2 Pure X noise



(a) Undeformed code



(b) Deformed code

Figure 3.5: Crossover plot of the cubic code under pure X noise, decoded with a BP-OSD decoder, for both the undeformed (a) and deformed (b) stabilizers.

Figure 3.5 shows the crossover plots for pure X noise. As expected, the loop sector in the undeformed case has no error, as X errors only lead to point excitations. On the other hand, the loop sector contributes to the logical error rate in the deformed case. As discussed in Section 3.2.1, the theoretical threshold for the loop sector is 50%. This cannot be resolved precisely on the curves, due to a finite-size effect, but we are expecting to recover this 50% as we add more lattice sizes to the system.

The actual thresholds of the code under pure X noise can be read from the point sector, and we find 2.9% for the undeformed case and 7.4% for the

deformed case.

3.4.3 Threshold and bias

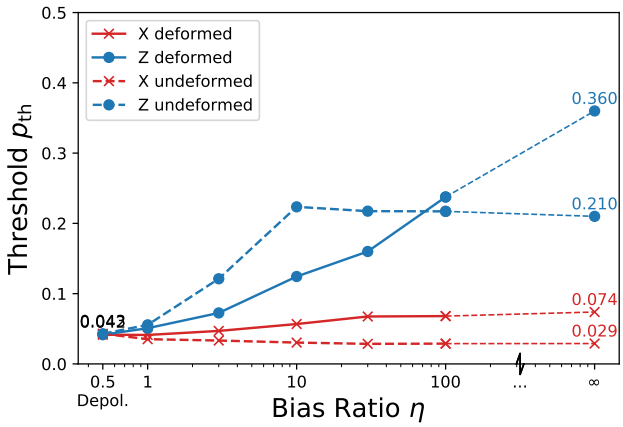


Figure 3.6: Threshold vs bias plot for the cubic lattice, for both X and Z bias, deformed and undeformed codes. We notice that for Z-biased noise, there is a regime for which the undeformed code performs better than the deformed one

Finally, Figure 3.6 shows the evolution of the threshold with the bias ratio η. For Z-biased noise, we observe the presence of two regimes: one for η_Z slightly below 100 where the deformation is worst than the original code, and one for higher bias ratios where the deformation quickly provides a significant advantage. Knowing the exact transition point will be useful in choosing which code to use depending on the experimental parameters of a given quantum device. On the other hand, for pure X noise, the deformed code provides a constant advantage.

3.5 Results for the rhombic lattice

We present in this section the results obtained for the 3D toric code with a rhombic lattice, for different noise models and stabilizer deformations. Since the sweep decoder is not directly generalizable to the rhombic lattice, we only used the BP-OSD decoder. Table 3.2 summarizes the different thresholds obtained in each configuration. As for the cubic results, let us start by noting

Noise model	Pure Z	Pure X	Depolarizing
Undeformed BP-OSD	**28.4**	1.0	1.5
Deformed BP-OSD	6.5	**1.7**	**1.7**

Table 3.2: Thresholds (in %) for the cubic lattice, using different noise models and code deformations. The best result for each noise model is highlighted in bold.

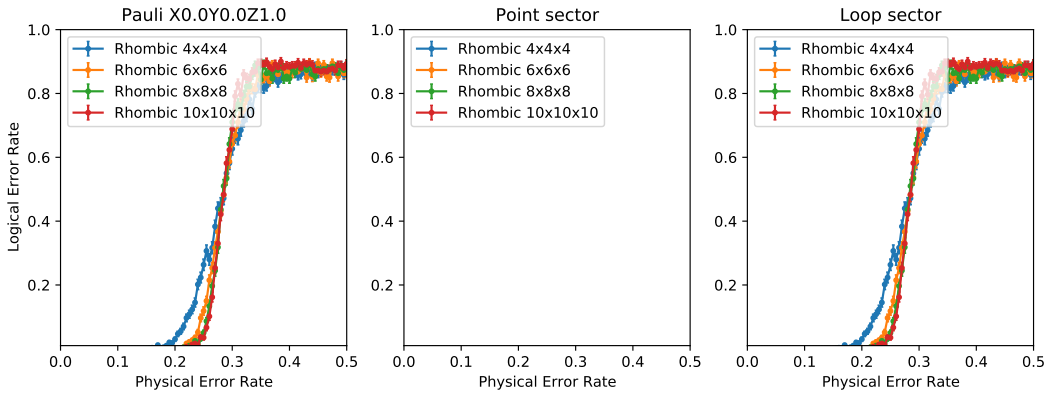
a two general trends from Table 3.2:

1. Going from no bias to pure Z bias has a dramatic effect on the original rhombic code: we pass from a threshold of 1.5% to 28.4%. This can easily be explained by the fact that pure Z noise only involves triangle stabilizers, which give a lot of information on the error.

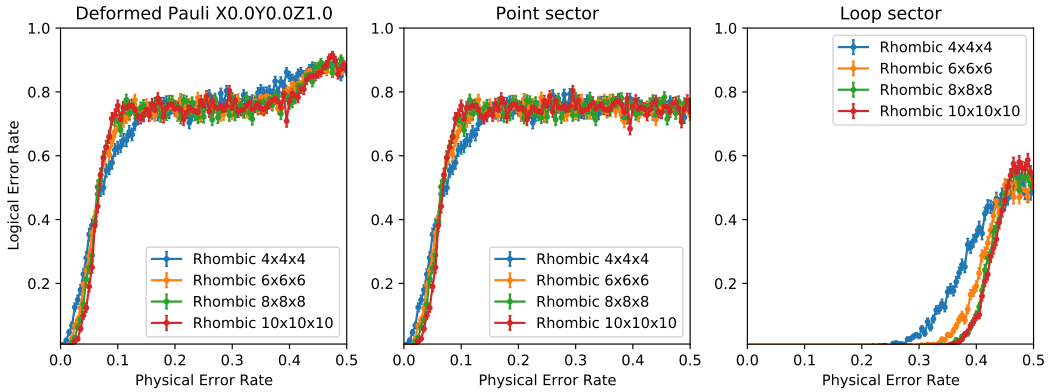
2. Deforming the code does not seem to help, at least in the pure Z case (which is the most relevant case as it leads to best overall thresholds). It can be explained by the introduction of a few cube excitations, which are particularly hard to decode, apparently even with the dimensionality reduction resulting from the deformation.

Therefore, the rhombic lattice could be a relevant lattice for a device with highly biased noise, but more work needs to be done to find better deformations. Let us now look at the detailed crossover plots and threshold vs bias curves.

3.5.1 Pure Z noise



(a) Undeformed code

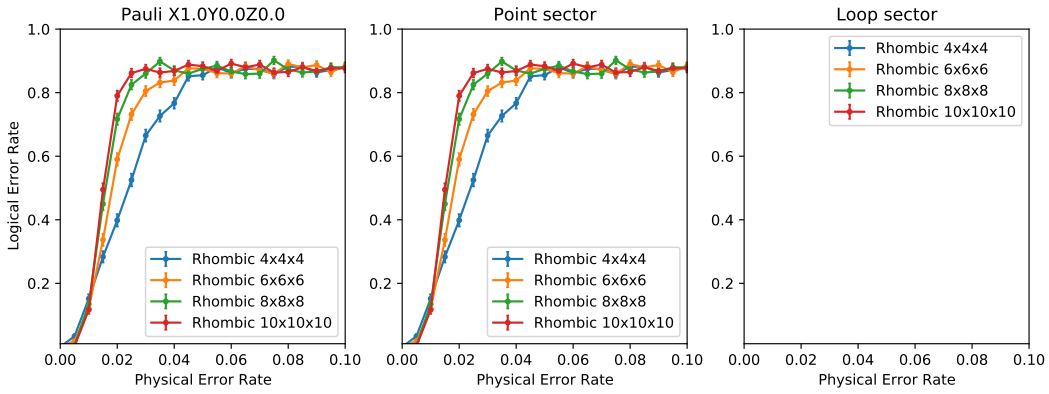


(b) Deformed code

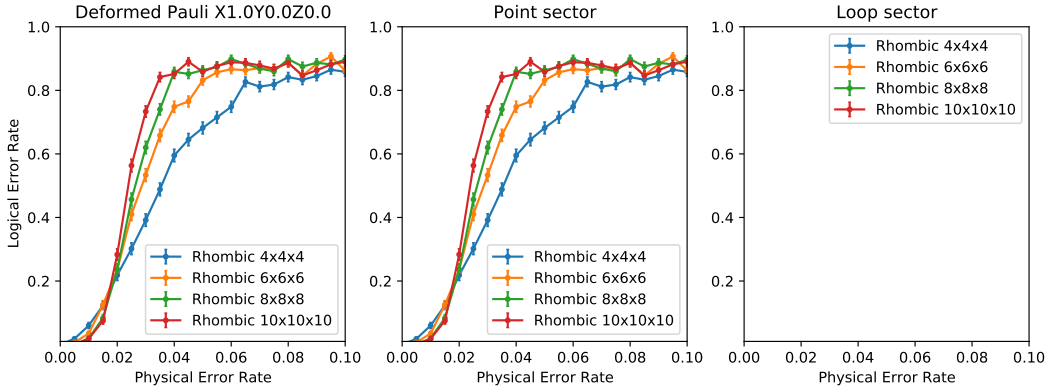
Figure 3.7: Crossover plot of the rhombic code under pure Z noise, decoded with a BP-OSD decoder, for both the undeformed (a) and deformed (b) stabilizers.

The results for pure Z noise are represented in Figure 3.7. We observe that the undeformed code only contains loop errors as expected, and has distinct crossover at 28.4%. The deformed code has some contributions from both sectors, with a loop threshold below 50%, which is first difference compared to cubic lattice.

3.5.2 Pure X noise



(a) Undeformed code



(b) Deformed code

Figure 3.8: Crossover plot of the rhombic code under pure X noise, decoded with a BP-OSD decoder, for both the undeformed (a) and deformed (b) stabilizers. Note: we rescaled the x-axis between 0 and 0.1 for convenience, since no loop error was detected between 0 and 0.5.

The results for pure X noise are represented in Figure 3.8. We observe that for both the deformed and the undeformed code, loop errors make no contribution to the total logical error. For the undeformed code, it is expected as only cube stabilizers are activated. For the deformed code, it is less obvious but it can also be explained theoretically: the deformed loop sector detects straight lines of X errors on a single axis. Since triangle stabilizers form a loop around those chains, they give sufficient information to fully reconstruct and correct the error (without any degeneracy). Therefore, the decoder is able to correct loop

errors all the time.

3.5.3 Threshold and bias

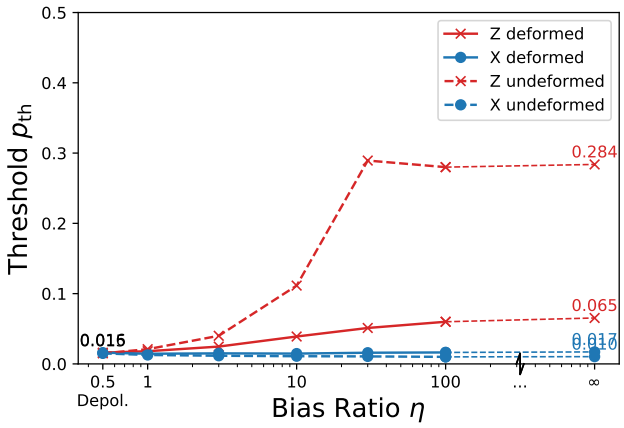


Figure 3.9: Threshold vs bias plot for the rhombic lattice, for both X and Z bias, deformed and undeformed codes.

Finally, Figure 3.9 shows the evolution of the threshold with the bias ratio. The main observation we can make from this plot is that threshold for undeformed Z noise quickly stabilizes to its final value of 28%. It means that the rhombic code could give good performance even for bias ratios as low as 30. For comparison, realistic bias ratios in current experiments can be of the order of 100 [18]. Comparing this to the deformed cubic lattice, which reaches 28% for bias ratios above 100 (see Figure 3.6), we can see that for a bias ratio between 30 and 100, the undeformed rhombic lattice gives the best performance of all the codes studied in this thesis.

Chapter 4

Conclusion

In this thesis, we have introduced a new version of the 3D toric code, with deformed stabilizers inspired by the recent XZZX surface code. We have shown that it outperforms the original 3D code when noise is highly biased towards X or Z errors. Moreover, we have compared two common decoders for 3D codes, SweepMatch and BP-OSD, and shown that BP-OSD systematically outperforms SweepMatch in the biased noise regime. Finally, we have studied the rhombic code under biased noise and found a very high threshold for bias ratios above 30 . However, the deformations proposed for the rhombic lattice did not lead to any relevant threshold improvement.

This project was part of a collaboration and my own contribution has been to implement both the BP-OSD decoder and the rhombic code in our library, and to evaluate the corresponding thresholds. Moreover, I have developed a 3D interface that has helped the integration and debugging of new codes in our library, in addition to helping gain more intuition on those codes and serving an educational purpose.

The results presented in this manuscript are still preliminary and more work needs to be done before publication. In particular, we plan to simulate more lattice sizes in order to have a better resolution of the threshold. We also plan to improve the finite-size scaling heuristic to obtain a correct threshold estimation for all our data, as well as some error bars to evaluate the uncertainty of the estimated threshold. Apart from those essential items, there are

many avenues for future work, including:

1. Improving and accelerating the BP-OSD decoder. In particular, many improvements of the BP decoder have been proposed in the literature, such as adding random noise to the messages [40] or using memory effects [42] in order to solve the degeneracy problem. Having a more accurate BP decoder would result in less calls to the costly OSD procedure, accelerating the overall decoding process. Moreover, implementing higher-order versions of OSD, as proposed in Ref. [41], could improve the threshold of our decoder.

2. Evaluating the threshold of the 3D code with coprime dimensions, generalizing the idea of Ref. [20] that 2D surface codes with coprime dimensions tend to have the highest threshold.

3. Incorporating non-periodic boundary conditions to our 3D codes, such as the ones proposed in Ref. [56], which are more experimentally relevant than the periodic ones.

4. Incorporating measurement errors to our threshold estimation, in order to evaluate our deformations in a more realistic setting.

5. Developing a statistical physics model to evaluate the threshold of the deformed code theoretically. This work has already been started by members of the collaboration.

6. Redoing the comparison between 2D and 3D codes made in Ref. [12], this time taking deformations and biased noise into account, to see if the 3D code has some chance of being relevant alternative to the surface code in the near future.

Overall, the recent idea of deforming stabilizers to improve the performance of a code under biased noise has been one of the most insightful ideas in quantum error-correction, as it has led to tremendous threshold improvements in both

the surface code and the 3D toric code. Developing this theory and applying it to more code families and realistic noise models has the potential to make fault-tolerance a little closer to reality.

Bibliography

[1] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982.

[2] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[3] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[4] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[5] Andrew M Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793, 1996.

[6] Emanuel Knill, Raymond Laflamme, and Wojciech Zurek. Threshold accuracy for quantum computation. *arXiv preprint quant-ph/9610011*, 1996.

[7] Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD Thesis, California Institute of Technology, 1997.

[8] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[9] A Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[10] David S Wang, Austin G Fowler, and Lloyd CL Hollenberg. Surface code quantum computing with error rates over 1%. *Physical Review A*, 83(2):020302, 2011.

[11] Michael Vasmer and Dan E Browne. Three-dimensional surface codes: Transversal gates and fault-tolerant architectures. *Physical Review A*, 100(1):012312, 2019.

[12] Michael E Beverland, Aleksander Kubica, and Krysta M Svore. Cost of universality: A comparative study of the overhead of state distillation and code switching with color codes. *PRX Quantum*, 2(2):020341, 2021.

[13] Panos Aliferis, Frederico Brito, David P DiVincenzo, John Preskill, Matthias Steffen, and Barbara M Terhal. Fault-tolerant computing with biased-noise superconducting qubits: a case study. *New Journal of Physics*, 11(1):013061, 2009.

[14] Alexander Grimm, Nicholas E Frattini, Shruti Puri, Shantanu O Mundhada, Steven Touzard, Mazyar Mirrahimi, Steven M Girvin, Shyam Shankar, and Michel H Devoret. Stabilization and operation of a kerr-cat qubit. *Nature*, 584(7820):205–209, 2020.

[15] Shruti Puri, Lucas St-Jean, Jonathan A Gross, Alexander Grimm, Nicholas E Frattini, Pavithran S Iyer, Anirudh Krishna, Steven Touzard, Liang Jiang, Alexandre Blais, et al. Bias-preserving gates with stabilized cat qubits. *Science advances*, 6(34):eaay5901, 2020.

[16] Daniel Nigg, Markus Mueller, Esteban A Martinez, Philipp Schindler, Markus Hennrich, Thomas Monz, Miguel A Martin-Delgado, and Rainer Blatt. Quantum computations on a topologically encoded qubit. *Science*, 345(6194):302–305, 2014.

[17] Michael D Shulman, Oliver E Dial, Shannon P Harvey, Hendrik Bluhm, Vladimir Umansky, and Amir Yacoby. Demonstration of entanglement of electrostatically coupled singlet-triplet qubits. *Science*, 336(6078):202–205, 2012.

[18] Andrew S Darmawan, Benjamin J Brown, Arne L Grimsmo, David K Tuckett, and Shruti Puri. Practical quantum error correction with the xzzx code and kerr-cat qubits. *arXiv preprint arXiv:2104.09539*, 2021.

[19] David K Tuckett, Stephen D Bartlett, and Steven T Flammia. Ultrahigh error threshold for surface codes with biased noise. *Physical review letters*, 120(5):050505, 2018.

[20] David K Tuckett, Andrew S Darmawan, Christopher T Chubb, Sergey Bravyi, Stephen D Bartlett, and Steven T Flammia. Tailoring surface codes for highly biased noise. *Physical Review X*, 9(4):041031, 2019.

[21] David K Tuckett, Stephen D Bartlett, Steven T Flammia, and Benjamin J Brown. Fault-tolerant thresholds for the surface code in excess of 5% under biased noise. *Physical review letters*, 124(13):130501, 2020.

[22] J Pablo Bonilla Ataides, David K Tuckett, Stephen D Bartlett, Steven T Flammia, and Benjamin J Brown. The xzzx surface code. *Nature communications*, 12(1):1–12, 2021.

[23] Aleksander Kubica and John Preskill. Cellular-automaton decoders with provable thresholds for topological codes. *Physical review letters*, 123(2):020501, 2019.

[24] Armanda O Quintavalle, Michael Vasmer, Joschka Roffe, and Earl T Campbell. Single-shot error correction of three-dimensional homological product codes. *PRX Quantum*, 2(2):020340, 2021.

[25] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

[26] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[27] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[28] Nikolas P Breuckmann and Jens Niklas Eberhardt. Ldpc quantum codes. *arXiv preprint arXiv:2103.06309*, 2021.

[29] Michael H Freedman and David A Meyer. Projective plane and planar quantum codes. *Foundations of Computational Mathematics*, 1(3):325–332, 2001.

[30] Nikolas P Breuckmann, Christophe Vuillot, Earl Campbell, Anirudh Krishna, and Barbara M Terhal. Hyperbolic and semi-hyperbolic surface codes for quantum storage. *Quantum Science and Technology*, 2(3):035007, 2017.

[31] Theodore J Yoder and Isaac H Kim. The surface code with a twist. *Quantum*, 1:2, 2017.

[32] Dan Browne. Lectures on topological codes and quantum computation. *retrieved from bit. do/topological*, 2014.

[33] Héctor Bombin, Ruben S Andrist, Masayuki Ohzeki, Helmut G Katzgraber, and Miguel A Martín-Delgado. Strong resilience of topological codes to depolarization. *Physical Review X*, 2(2):021004, 2012.

[34] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[35] Ben Criger and Imran Ashraf. Multi-path summation for decoding 2d topological codes. *Quantum*, 2:102, 2018.

[36] James R Wootton and Daniel Loss. High threshold error correction for the surface code. *Physical review letters*, 109(16):160503, 2012.

[37] Adrian Hutter, James R Wootton, and Daniel Loss. Efficient markov chain monte carlo algorithm for the surface code. *Physical Review A*, 89(2):022326, 2014.

[38] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3):032326, 2014.

[39] David Poulin. Optimal and efficient decoding of concatenated quantum block codes. *Physical Review A*, 74(5):052333, 2006.

[40] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *Quantum Info. Comput.*, 8(10):987–1000, November 2008.

[41] Joschka Roffe, David R White, Simon Burton, and Earl T Campbell. Decoding across the quantum ldpc code landscape. *arXiv preprint arXiv:2005.07016*, 2020.

[42] Kao-Yueh Kuo and Ching-Yi Lai. Exploiting degeneracy in belief propagation decoding of quantum codes. *arXiv preprint arXiv:2104.13659*, 2021.

[43] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Physical review letters*, 104(5):050504, 2010.

[44] Guillaume Duclos-Cianci and David Poulin. Fault-tolerant renormalization group decoder for abelian topological codes. *arXiv preprint arXiv:1304.6100*, 2013.

[45] Giacomo Torlai and Roger G Melko. Neural decoder for topological codes. *Physical review letters*, 119(3):030501, 2017.

[46] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7(1):1–7, 2017.

[47] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network–based decoder. *Quantum Machine Intelligence*, 2(1):1–12, 2020.

[48] Nikolas P Breuckmann and Xiaotong Ni. Scalable neural network decoders for higher dimensional quantum codes. *Quantum*, 2:68, 2018.

[49] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218*, 2017.

[50] Shilin Huang, Michael Newman, and Kenneth R Brown. Fault-tolerant weighted union-find decoding on the toric code. *arXiv preprint arXiv:2004.04693*, 2020.

[51] Ashley M Stephens, Zachary WE Evans, Simon J Devitt, and Lloyd CL Hollenberg. Asymmetric quantum error correction via code conversion. *Physical Review A*, 77(6):062335, 2008.

[52] ZWE Evans, AM Stephens, JH Cole, and LCL Hollenberg. Error correction optimisation in the presence of x/z asymmetry. *arXiv preprint arXiv:0709.3875*, 2007.

[53] Andrew S Darmawan and David Poulin. Linear-time general decoding algorithm for the surface code. *Physical Review E*, 97(5):051302, 2018.

[54] Panos Aliferis and John Preskill. Fault-tolerant quantum computation against biased noise. *Physical Review A*, 78(5):052331, 2008.

[55] John Napp and John Preskill. Optimal bacon-shor codes. *arXiv preprint arXiv:1209.0794*, 2012.

[56] Michael John George Vasmer. *Fault-tolerant quantum computing with three-dimensional surface codes*. PhD thesis, UCL (University College London), 2019.

[57] TR Scruby, DE Browne, P Webster, and M Vasmer. Numerical implementation of just-in-time decoding in novel lattice slices through the three-dimensional surface code. *arXiv preprint arXiv:2012.08536*, 2020.

[58] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *Physical review letters*, 122(20):200501, 2019.