# Implementation and analysis of stabilizer codes in pyQuil

Shubham Chandak, Jay Mardia and Meltem Tolunay

**Abstract**

Stabilizer codes form a large family of quantum error correcting codes that includes well-known codes such as Shor code, Steane code, CSS codes and toric codes. In this work, we build a framework for encoding and decoding of general stabilizer codes on pyQuil and test specific single qubit codes with standard quantum noise models. This project is part of CS269Q Spring 2018-19 course project at Stanford.

The code is available at `https://github.com/shubhamchandak94/stabilizer_code/`.

## 1 Introduction

Quantum fault tolerance and error correction are techniques that will need to be used to harness the computational promise of quantum computers. One popular class of quantum error correcting codes that could help us do this are Stabilizer Codes [1]. In this project we implement a general framework to encode and decode arbitrary stabilizer codes in pyQuil on a Quantum Virtual Machine. We then characterize the performance of a few specific codes in this class with real world noise models.

## 2 Methods

The key technical aspect of our implementation is a function to generate the encoding and decoding pyQuil programs given the description of the stabilizer code in terms of the generating elements of the stabilizer group and its error correcting capabilities. For an $[n, k]$ stabilizer code, the encoding program encodes any input state on $k$ qubits to the corresponding encoded state on $n$ qubits. The decoding program takes as input the (noisy) encoded state on $n$ qubits, applies the error correction and decodes the encoded state back to the original state on $k$ qubits [1]. The code for this is available at `stabilizer_code.py`. We have also implemented a framework that takes in a generic stabilizer code specification, a generic noise model (via Kraus operators) and computes the logical error rates for this pair.

We used this to run the following experiments to characterize the performance of codes on various noise models.

1. Basic tests to see if the code can correct one bit flip, one phase flip, one Y gate, one H gate, a combination of an X and Z gate on different qubits. We have tested the framework for the bit flip code, the phase flip code, the Steane code, the Shor code and the five qubit code (optimal code for correcting 1 error). To see this, run `basic_tests.py`.

2. We pick a single qubit stabilizer code and a (parametrized) noise model, and then plot the logical vs physical error rate curve achieved with this pair. Specifically, we use the following codes : bit flip code, phase flip code, 5 qubit code, Steane code. (We have an implementation of the Shor code that also works. however we did not run simulations with it because it was slower than the other codes.) The noise models used are: bit flip noise, phase flip noise, depolarizing noise, amplitude damping, and dephasing noise. Details can be found in the code `noise_models_kraus.py`. To rerun these experiments, simply run `logical_vs_physical_error_rates.py`.

   We run this noise characterization in two ways -

   (a) Our initial state is either $|0\rangle$ or $|1\rangle$ with equal probability. The reason to do this is that since we only measure in the $Z$ basis, it is not possible (without accessing the wavefunction from the simulator or performing state tomography) to accurately measure how many times a error correction has failed if our initial states aren't one the computational basis states. However, this approach has severe flaws. Note that the bit flip code encodes $\alpha|0\rangle + \beta|1\rangle$ as $\alpha|000\rangle + \beta|111\rangle$, and a phase flip error (on all three qubits) would take this to $\alpha|000\rangle - \beta|111\rangle$, an error that can't be corrected by the bit flip code. However, if the initial state was $|0\rangle$ or

[1] As we note in the appendix while describing our encoding and decoding circuits, the procedure provided in [1] had a slight bug. After understanding the logic behind the procedure in complete detail, we made the appropriate corrections to the procedure. We have contacted the author of [1] regarding this.

$|1\rangle$ ($\alpha = 1$ or $\alpha = 0$), the corrupted state is the same as the original state. This would give us artificially low logical error rates (as we observe in our experiments as shown in the Experiments section.) To overcome this, we tried the next point.

(b) Our initial state is a uniformly random point on the Bloch sphere. This has the advantage of not giving artificial results, but has the drawback that we aren't so much measuring whether we recover the initial state as much as how close our recovered state is to our initial state. To elaborate, the drawback is that sometimes our error correction algorithm could fail, but when we try to detect this using a measurement, we may get cases where we think our algorithm has succeeded, simply because the initial and final states are not orthogonal and hence not perfectly distinguishable.

3. We fix a noise model. depolarizing noise with parameter $p = 0.4$. We then plot logical error rates vs "code rate" (number of logical qubits vs number of physical qubits) for various codes to study the trade-off between logical error rates and the amount of redundancy we add to a qubit. To run this experiment, run `code_rate_vs_logical_error_rate.py`.

More instructions about using the code are available in the README at `https://github.com/shubhamchandak94/stabilizer_code/`.

We now describe how we add noise to our qubits. One way to use pyQuil to add noise to qubits is to define a set of Kraus operators, and redefine an existing gate (we choose the I gate) to apply these Kraus operators (which correspond to noise) instead. Thus whenever our program calls for an I gate, this noisy I gets applied[2]. This is the method we use to add noise to our programs [3].

Even though the procedure for encoding and decoding stabilizer codes is described in [1] and [2], the description in these references was quite hard to follow. Thus, we present a systematic description of the circuits we use to encode and decode stabilizer codes in the Appendix along with some examples. We encourage the interested reader to check this out. Now we present the experimental results in the next section.

# 3 Experiments and discussion

| Code | $n$ | $k$ | Error pattern | | | | |
|---|---|---|---|---|---|---|---|
| | | | $X$ | $Z$ | $Y$ | $H$ | $X, Z$ |
| Bit flip | 3 | 1 | ✓ | ✗ | ✗ | ✗ | ✗ |
| Phase flip | 3 | 1 | ✗ | ✓ | ✗ | ✗ | ✗ |
| Shor | 9 | 1 | ✓ | ✓ | ✓ | ✓ | ✓ |
| Steane | 7 | 1 | ✓ | ✓ | ✓ | ✓ | ✓ |
| Five qubit | 5 | 1 | ✓ | ✓ | ✓ | ✓ | ✗ |

Table 1: Basic tests for selected Stabilizer codes showing the ability of each code to correct various single qubit errors and one two qubit error. The last error pattern $X, Z$ means applying an $X$ gate and a $Z$ gate to different qubits.

1. The experiment in Table 1 runs the basic tests to provide a sanity check for our implementation, as we see that the error correction behaves as we expect it to. We can now move on to characterizing the performance under different noise models.

2. We now run the $2^{nd}$ experiment as listed in the Methods section. We use $n$ trials for each data point (code, noise model, noise parameter) and sweep the noise parameter from 0 to 1 with a spacing of 0.1. We show the results below. On the left, the initial state is either $|0\rangle$ or $|1\rangle$, and $n = 500$. On the right, the initial state is a random location on the Bloch sphere, and $n = 2000$.

   **Observations:**

   (a) For initial states $|0\rangle$ or $|1\rangle$, when we have dephasing or phase flip noise, the simulations suggest that all codes except the five qubit code are perfect. However, this is simply a misleading artifact of our choice of initial states, as we remarked in the previous section.

[2] This means that our framework for adding noise (though not the stabilizer encoding and decoding) is contingent on there not being any actual I gates in the program. This isn't a drawback functionally, but one needs to be aware of this while using our code.

[3] One observation that might be of interest to anyone trying to implement noise in such a manner is that if we use the `gate modifiers` 'CONTROLLED' and 'DAGGER' anywhere in our program, we can't define noisy gates this way. We get an error that says "The noisy QVM doesn't support gate modifiers" when we try to add noise in such a manner to a program that has gate modifiers. Our workaround seems to be to stop using the modifiers anywhere in our implementation, though this (slightly) limits the generality of what we are doing.

(b) The five qubit code has much higher logical error rates than other codes for several noise models.

(c) The bit flip code (resp. phase flip code) does very well for bit flip noise (resp. phase flip noise) at reasonably small noise levels, which is what we would expect.

(d) It is interesting to observe that the ranking of "best" code for a given noise model depends not only on the noise model but also on the amount of noise. This can be seen because the logical error rates for different codes cross each other, and is slightly surprising. This effect is most prominent in the plots for bit flip and phase flip noise. In the bit flip noise experiment, we observe that the bit flip code does better than phase flip code, Steane code (both of which have essentially the same performance) up to bit flip probability = 0.5, and then this completely reverses. Note that in practice, if we know that the logical error rate is above 0.5 (as is clearly true in the left bit flip noise plot), it is prudent to apply a logical $X$ gate to our encoded qubits, just like in the classical case for Binary Symmetric Channels with error probability greater than 0.5.

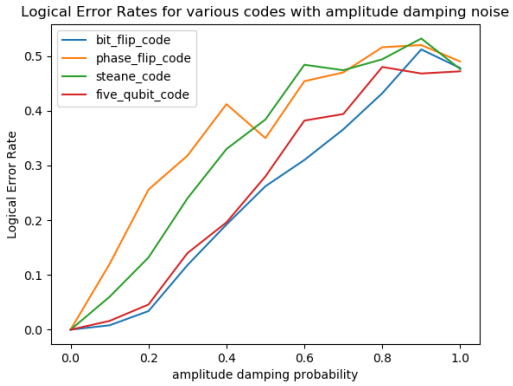


Figure 1: Initial state is either $|0\rangle$ or $|1\rangle$, trials = 500

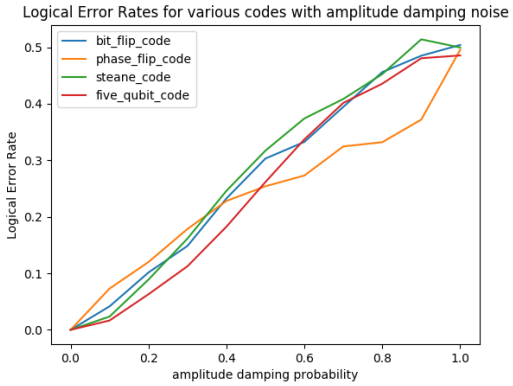


Figure 2: Initial state is a random location on the Bloch sphere, trials = 2000

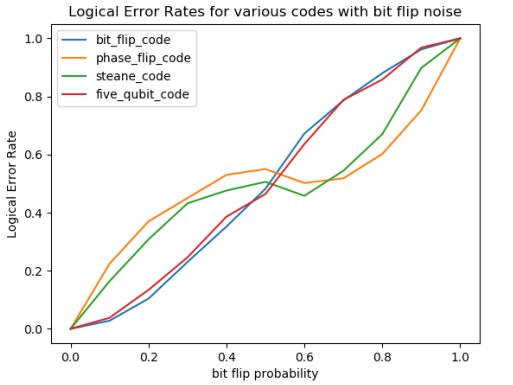


Figure 3: Initial state is either $|0\rangle$ or $|1\rangle$, trials = 500

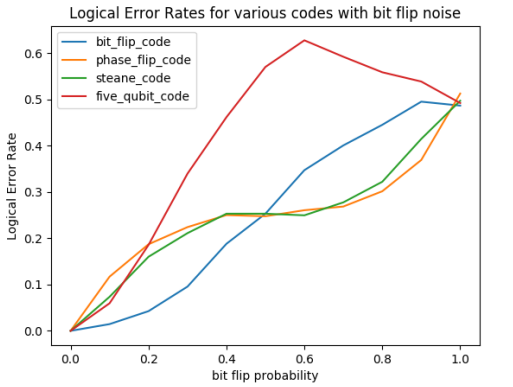


Figure 4: Initial state is a random location on the Bloch sphere, trials = 2000

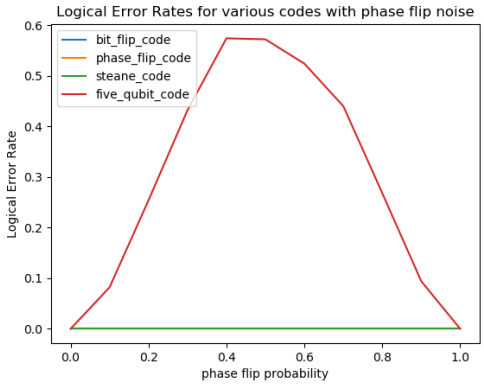


Figure 5: Initial state is either $|0\rangle$ or $|1\rangle$, trials = 500

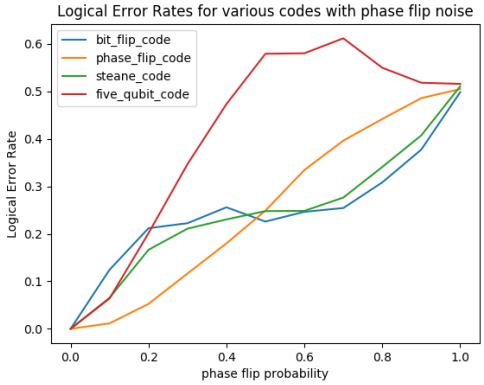


Figure 6: Initial state is a random location on the Bloch sphere, trials = 2000

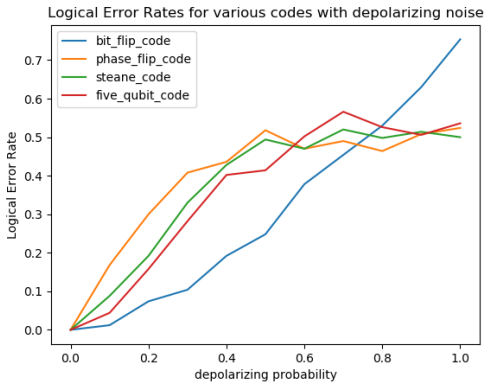


Figure 7: Initial state is either $|0\rangle$ or $|1\rangle$, trials = 500

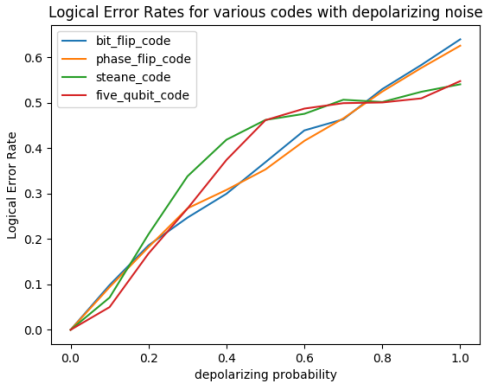


Figure 8: Initial state is a random location on the Bloch sphere, trials = 2000

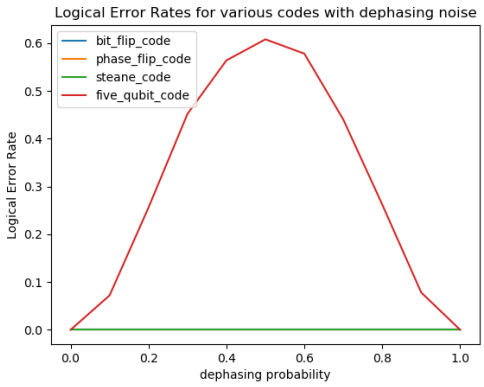


Figure 9: Initial state is either $|0\rangle$ or $|1\rangle$, trials = 500

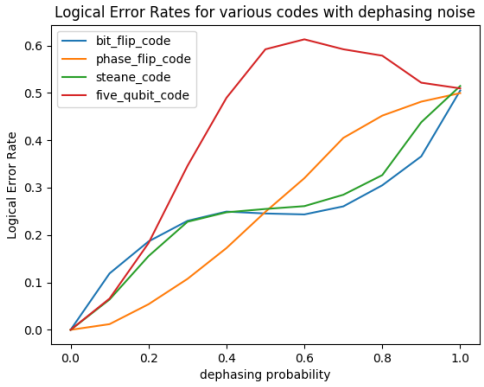


Figure 10: Initial state is a random location on the Bloch sphere, trials = 2000

3. For the third experiment, we plot logical error rates vs code rate. We use depolarizing noise with parameter 0.4 and 1000 trials.

   **Remark:** Naively, one would expect that codes with lower rate (higher redundancy) would have better error correction capabilities, but we can see that for the depolarizing channel this isn't true. The higher rate codes have lower logical error rates. We are not entirely sure why this happens.

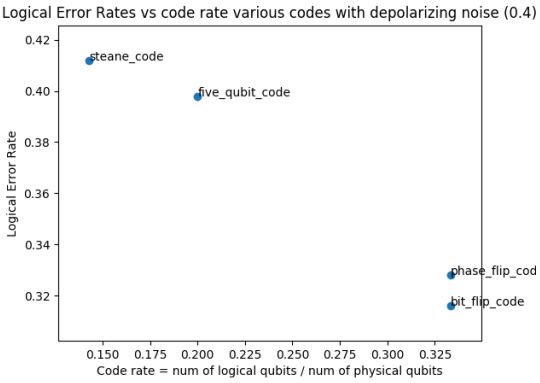


Figure 11: trials = 1000

# 4 Possible Future Directions

1. The logical next step would be to test our implementation of our codes on an actual quantum computer. The challenge here is that the Rigetti quantum computer we have access to does not currently have the functionality to perform gate operations after measurement, which is crucial to error correction after measuring the syndrome.

2. Our current implementation is capable of implementing a generic stabilizer code. The flip side of this generality is that if we wanted to implement specific codes, it is possible to do so in a more optimized manner (fewer gates, fewer ancilla qubits). In particular, CSS codes (a subclass of stabilizer codes) could be implemented more efficiently, and a future direction could be to try and implement this.

3. Currently we are adding noise to our qubits, but our gates are still noise free. It is possible to redefine the entire program to run with noisy gates (appropriately defined). Implementing this sort of noise would be closer to how real quantum computers would behave and hence characterizing code performance in this framework would be a useful thing to do.

4. Since we are simulating these codes on a virtual machine, we can in principle get access to the wavefunction of a state. This would help solve the issues with trying to compute logical error rates that pertain to the choice of initial state. We could start from a random Bloch state, and at the end of the correction procedure compare the output wavefunction to the known initial wavefunction (using perhaps a trace distance or fidelity metric). This would perhaps be a more principled way of measuring the efficacy of a code.

# Acknowledgements

We thank Prof. Will Zeng for helpful discussions and feedback.

# References

[1] D. Gottesman, "Stabilizer codes and quantum error correction," *arXiv preprint quant-ph/9705052*, 1997.

[2] M. A. Nielsen and I. L. Chuang, *Quantum error-correction*, p. 425–499. Cambridge University Press, 2010.

[3] S. J. Devitt, W. J. Munro, and K. Nemoto, "Quantum error correction for beginners," *Reports on Progress in Physics*, vol. 76, no. 7, p. 076001, 2013.

[4] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[5] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, "Perfect quantum error correcting code," *Physical Review Letters*, vol. 77, no. 1, p. 198, 1996.

# Appendix

## A.1 Circuits for encoding and decoding

In this section, we describe the circuits for encoding and decoding stabilizer codes. While the procedure is partially described in the references [1], [2] and [3], the presentation was not very systematic and was hard to follow, perhaps due to lack of a programming environment when these works were published. Therefore, we describe the circuits in the form of a pseudocode in this section. Most of the procedure is based on the discussion in Chapter 4 of [1]. Note that we only provide the algorithm in this section with only a few comments on the reasoning behind the steps. The detailed discussion about the correctness of the various optimizations is provided in [1]. Examples of the procedure for selected codes is provided in Section A.2.

#### Crash course on stabilizer codes

An $[n, k]$ stabilizer code encodes $k$ logical qubits to $n$ physical qubits and is specified by an order $2^{n-k}$ subgroup $\mathcal{S}$ of the Pauli group on $n$ qubits $\mathcal{G}_n$ (up to global phase factors). For a valid code, elements of $\mathcal{S}$ must commute with each other. A state on $n$ qubits $|\phi\rangle$ is a valid codeword state if and only if for every $S \in \mathcal{S}$, $S|\phi\rangle = |\phi\rangle$. It can be shown that the codewords form a subspace of dimension $k$, allowing us to encode $k$ logical qubits. Also, the group $\mathcal{S}$ is generated by a generator set of size $n - k$. We will work with the generator set throughout the section.

**Example:** Consider the bit flip code with $n = 3$, $k = 1$ and the generator set $\{ZZI, ZIZ\}$. The entire stabilizer group is given by $\mathcal{S} = \{ZZI, ZIZ, IZZ, III\}$. It is easy to verify that the span of $\{|000\rangle, |111\rangle\}$ forms the one-dimensional codeword subspace stabilized by the elements of $\mathcal{S}$. One possible encoding function is to map $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$, where the $L$ in the subscript denotes logical.

To understand the error correction capabilities of the stabilizer codes, consider an error operator $E \in \mathcal{G}_n$. If $E \in \mathcal{S}$, then it doesn't impact the state so we have no issues. If $E \notin \mathcal{S}$ but $E$ does not commute with some element in $\mathcal{S}$, then we could detect the error by measuring the stabilizer operator eigenvalue (recall that elements in $\mathcal{G}_n$ either commute or anticommute, so the measurement will give 1 or -1 depending on whether there is an error). If $E \notin \mathcal{S}$ and $E$ commutes with all elements in $\mathcal{S}$, we are in trouble since the state gets transformed to another valid codeword. Thus, it is possible to find the error patterns that a stabilizer code can correct. For further discussion about the error correction capabilities of stabilizer codes, see Section 10.5.5 of [2].

**Example (contd.):** For the bit flip code, consider the following error patterns:

- $E = IZZ$: This is in $\mathcal{S}$ and doesn't modify the codeword.
- $E = XII$: This is not in $\mathcal{S}$ but anticommutes with $ZZI$, so this can be detected by measurements. E.g., for $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$, applying $E$ gives us $E|\psi\rangle_L = \alpha|100\rangle + \beta|011\rangle$. This has eigenvalue -1 for the operator $ZZI$, and hence the error can be detected.
- $E = ZII$: This is not in $\mathcal{S}$ and commutes with all elements of $\mathcal{S}$. For $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$, applying $E$ gives us $E|\psi\rangle_L = \alpha|000\rangle - \beta|111\rangle$ which is a valid codeword corresponding to $|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$. Hence this error cannot be detected.

Finally, we discuss operators acting directly on the encoded qubits, referred to as logical operators. In particular we are interested in logical $X$ and $Z$ operators acting on logical qubit $i$, denoted as $\bar{X}_i$ and $\bar{Z}_i$, respectively. The action of these operators is defined by $\bar{X}_i|\psi\rangle_L = |X_i\psi\rangle_L$ (and similar for $\bar{Z}_i$). These operators need not be unique in general and always commute with the stabilizers and with each other (in particular, multiplying the logical operator with a stabilizer gives another operator performing the same logical operation). The encoded $Y$ operator is given by $\bar{X}_i\bar{Z}_i$ (ignoring global phase).

**Example (contd.):** For the bit flip code, we can take $\bar{X}_1 = XXX$ and $\bar{Z}_1 = ZZZ$. E.g.,

$$\begin{aligned}\bar{X}_1|\psi\rangle_L &= XXX(\alpha|000\rangle + \beta|111\rangle) \\ &= \alpha|111\rangle + \beta|000\rangle \\ &= |X_1\psi\rangle_L\end{aligned}$$

It will convenient to represent the stabilizer generators and the logical operators in a binary matrix format. The idea is to denote $I$ by (0,0), $X$ by (1,0), $Z$ by (0,1) and $Y$ by (1,1). The stabilizer generators are represented as a $(n-k) \times 2n$ matrix with the first $n$ columns representing the first entry in the tuple and the the last $n$ columns representing the second entry in the tuple. Similarly, the logical operators are represented as a $k \times 2n$ matrix (one row per logical qubit).

**Example (contd.):** For the bit flip code, the stabilizer generators $\{ZZI, ZIZ\}$, the logical $X$ operator ($XXX$) and the logical $Z$ operator ($ZZZ$) and are represented as

$$\left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right), \left(\begin{array}{ccc|ccc} 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right), \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right),$$

respectively.

### Preprocessing

Given the stabilizer generator matrix of size $(n-k) \times 2n$, we first apply Gaussian elimination (using row swaps, row additions mod 2 and column swaps) to the left half to obtain

$$\begin{array}{c} \overbrace{\phantom{I}}^{r}\ \overbrace{\phantom{A}}^{n-r}\ \overbrace{\phantom{B}}^{r}\ \overbrace{\phantom{C}}^{n-r} \\ \left(\begin{array}{cc|cc} I & A & B & C \\ 0 & 0 & D & E' \end{array}\right) \begin{array}{l} \}r \\ \}n-k-r \end{array} \end{array}$$

where $r$ is the rank of the left half. Then we perform Gaussian elimination on $E'$ (which must have full rank for valid generator matrices) to get

$$M = \begin{array}{c} \overbrace{\phantom{I}}^{r}\ \overbrace{\phantom{A_1}}^{n-k-r}\ \overbrace{\phantom{A_2}}^{k}\ \overbrace{\phantom{B}}^{r}\ \overbrace{\phantom{C_1}}^{n-k-r}\ \overbrace{\phantom{C_2}}^{k} \\ \left(\begin{array}{ccc|ccc} I & A_1 & A_2 & B & C_1 & C_2 \\ 0 & 0 & 0 & D & I & E \end{array}\right) \begin{array}{l} \}r \\ \}n-k-r \end{array} \end{array}$$

Note that when we perform Gaussian elimination on a submatrix, we need to perform the same row/column operations for the rest of the matrix as well (in particular, column swaps correspond to swapping qubits and when swapping columns in the left half, the corresponding columns in the right half should be swapped as well).

Given this representation, the logical $X$ and $Z$ operators are given by

$$\bar{X} = \begin{array}{c} \overbrace{\phantom{0}}^{r}\ \overbrace{\phantom{E^T}}^{n-k-r}\ \overbrace{\phantom{I}}^{k}\ \ \overbrace{\phantom{E^TC_1^T+C_2^T}}^{r}\ \overbrace{\phantom{0}}^{n-k-r}\ \overbrace{\phantom{0}}^{k} \\ \left(\begin{array}{ccc|ccc} 0 & E^T & I & E^TC_1^T+C_2^T & 0 & 0 \end{array}\right) \}k \end{array}$$

$$\bar{Z} = \begin{array}{c} \overbrace{\phantom{0}}^{r}\ \overbrace{\phantom{0}}^{n-k-r}\ \overbrace{\phantom{0}}^{k}\ \ \overbrace{\phantom{A_2^T}}^{r}\ \overbrace{\phantom{0}}^{n-k-r}\ \overbrace{\phantom{I}}^{k} \\ \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & A_2^T & 0 & I \end{array}\right) \}k \end{array}$$

Next, we generate a map from syndrome measurements to the corresponding correction operators, i.e., a function $f_{cor} : \{0,1\}^{n-k} \to \mathcal{G}_n$. The syndromes correspond to the measurement of the $n-k$ stabilizer generators. We assume that we are provided with a list of errors that the code can correct (such a list can be obtained as discussed in Section 10.5.5 for [2]). Some example error patterns are (i) $X$ on first qubit, (ii) $Y$ on third qubit, (iii) $X$ or first qubit, $Z$ on fifth qubit. For codes that can correct all single qubit errors we would have $3n$ error patterns corresponding to $X$, $Z$ and $Y$ errors on each qubit. If the code can also correct one $X$ and one $Z$ error, we get $n(n-1)$ additional error patterns. Clearly the correction operator is the same as the error pattern since the Pauli matrices are self-inverse. Each error pattern $e \in \mathcal{G}_n$ can be represented as a $1 \times 2n$ binary row vector. The corresponding syndrome is given by $eM^T$ (mod 2). The process for constructing the map $f_{cor}$ is as follows:

- Set $f_{cor}(0^{n-k}) = I$.
- For $e$ in set of error patterns:
  - Set $f_{cor}(eM^T) = e$.
- For any remaining syndrome $s$ in the domain of $f_{cor}$, set $f_{cor}(s) = I$.

The first step sets the error correction for perfect syndrome to identity and the last step takes care of any syndromes that are not produced by the specified error patterns. For degenerate codes such as Shor code [2], multiple error patterns give the same syndrome, but any of them work for the error correction step.

**Encoding**

We next describe the encoding circuit which starts with the state $|\psi\rangle$ in qubits $n-k$ to $n-1$ and $|0\rangle$ in ancilla qubits 0 to $k-1$. After encoding, the $n$ qubits contain the state $|\psi\rangle_L$. The full logic behind the circuit is given in [1], but the core idea is to first prepare the encoded $|0\rangle_L$ state and then use the $\bar{X}_L$ operators controlled by $|\psi\rangle$ to modify it to a general encoded state (although these steps are swapped for optimization reasons).

1. For $i$ in $0 \ldots k-1$:
   - For $j$ in $r \ldots n-k-1$:
     - If $\bar{X}_{i,j} = 1$, apply $CNOT(n-k+i, j)$.
2. For $i$ in $0 \ldots r-1$:
   - Apply $H(i)$.
   - If $M_{i,n+i} = 1$, apply $S(i)$.
   - For $j$ in $0 \ldots n-1$:
     - If $j = i$, continue.
     - Apply controlled Pauli $(M_{i,j}, M_{i,n+j})$ on qubit $j$ controlled by qubit $i$.

**Remark:** In step 2, the procedure provided in [1] first applies $H$ gate on all $r$ qubits and then does the remaining operations in a separate loop (with the $S$ gates replaced by $Z$ gates). Unfortunately that procedure was incorrect and did not produce a codeword in the correct subspace for certain codes. After understanding the logic behind the procedure in complete detail, we made the appropriate corrections to the procedure. We have contacted the author of [1] regarding this.

## Decoding

The decoding consists of three main steps:

1. Perform syndrome measurements: measure the stabilizer generators using the general measurement circuit shown in Figure 12 (also see Section 10.5.8 in [2]).
2. Apply correction according to the syndrome using the map $f_{cor}$.
3. Decode the $n$ encoded qubits $|\psi\rangle_L$ to the $k$ original qubits $|\psi\rangle$. This is described below.

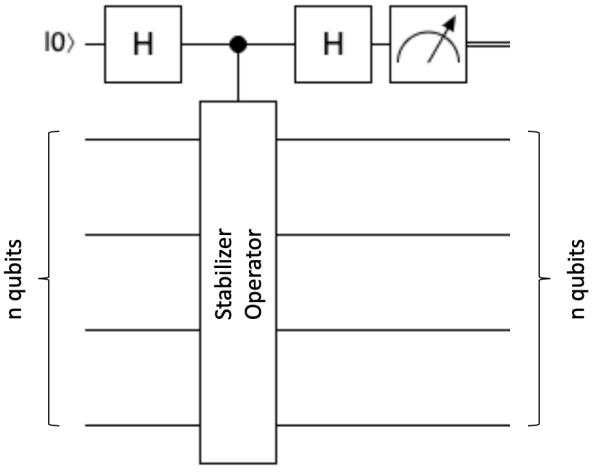


Figure 12: Measurement circuit for stabilizer operator.

For step 3, we start with the state $|\psi\rangle_L$ in qubits 0 to $n-1$ and $|0\rangle$ in ancilla qubits $n$ to $n+k-1$. At the end, we are left with $|0\rangle_L$ in qubits 0 to $n-1$ and the decoded state $|\psi\rangle$ in qubits $n$ to $n+k-1$. The procedure is as follows:

- For $i$ in $0 \dots k-1$:
  - For $j$ in $0 \dots n-1$:
    - If $\bar{Z}_{i,n+j} = 1$, apply $CNOT(j, n+i)$).
  - For $j$ in $0 \dots n-1$:
    - Apply controlled Pauli $(\bar{X}_{i,j}, \bar{X}_{i,n+j})$ on qubit $j$ controlled by qubit $n+i$.

Section A.2 illustrates the process for the Steane code [4] and the five qubit code that corrects one qubit error [5].

## A.2 Examples for encoding/decoding procedure

In this section, we illustrate the encoding and decoding procedure with two examples. These examples were generated using the code at `https://github.com/shubhamchandak94/stabilizer_code/`.

### Steane code

The Steane code is a $[7, 1]$ code defined by the 6 stabilizer generators

$$XXXXIII$$
$$XXIIXXI$$
$$XIXIXIX$$
$$ZZZZIII$$
$$ZZIIZZI$$
$$ZIZIZIZ$$

Note that the each stabilizer generator has either $X$ or $Z$ but not both. Thus, Steane codes belong to the class of CSS codes [2] and have additional properties such as ease of applying several logical operators directly on the encoded data as well simpler decoding procedure [3]. The matrix representation of the generators is

$$\left(\begin{array}{ccccccc|ccccccc} 1&1&1&1&0&0&0&0&0&0&0&0&0&0\\ 1&1&0&0&1&1&0&0&0&0&0&0&0&0\\ 1&0&1&0&1&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&1&1&1&0&0&0\\ 0&0&0&0&0&0&0&1&1&0&0&1&1&0\\ 0&0&0&0&0&0&0&1&0&1&0&1&0&1 \end{array}\right)$$

After applying the Gaussian elimination steps we obtain $r = 3$ and

$$M = \left(\begin{array}{ccc|ccc|c|ccc|ccc|c} 1&0&0&1&0&1&1&0&0&0&0&0&0&0\\ 0&1&0&1&1&0&1&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0&0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0&1&1&1&1&0&0&0\\ 0&0&0&0&0&0&0&1&0&1&0&1&0&1\\ 0&0&0&0&0&0&0&0&1&1&0&0&1&1 \end{array}\right)$$

In this representation, we get

$$\bar{X} = \left(\begin{array}{ccc|ccc|c|ccc|ccc|c} 0&0&0&0&1&1&1&0&0&0&0&0&0&0\end{array}\right)$$

$$\bar{Z} = \left(\begin{array}{ccc|ccc|c|ccc|ccc|c} 0&0&0&0&0&0&0&1&1&0&0&0&0&1\end{array}\right)$$

where we show the block structure of these matrices.

Finally the encoding and decoding circuits for this code are shown in Figures 13 and 14, respectively. Note that the decoding circuit only shows step 3 in the decoding.

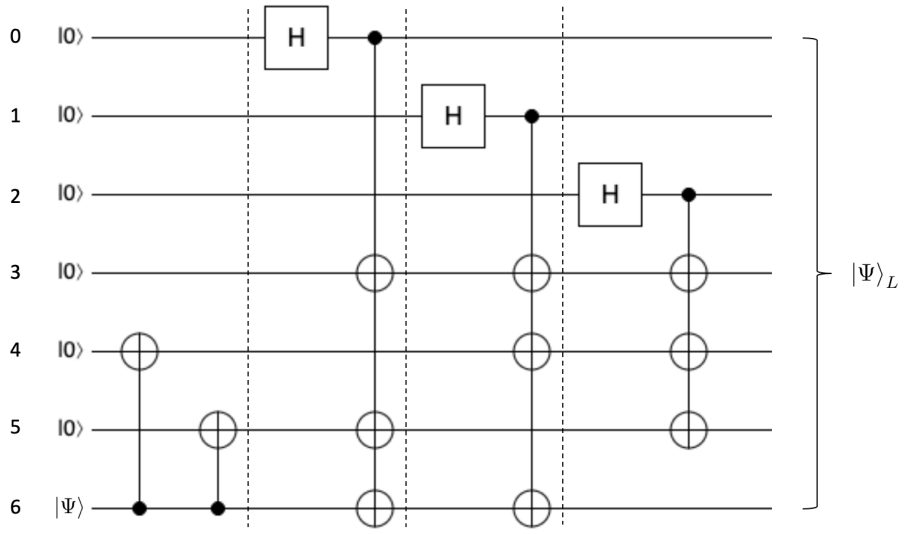


Figure 13: Encoding circuit for Steane code.

**Five qubit code**

The Steane code is a $[5, 1]$ code defined by the 4 stabilizer generators

$$XZZXI$$
$$IXZZX$$
$$XIXZZ$$
$$ZXIXZ$$

The matrix representation of the generators is

$$\left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

After applying the Gaussian elimination steps we obtain $r = 4$ and

$$M = \left(\begin{array}{cccc\|c|cccc\|c} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}\right)$$

In this representation, we get

$$\bar{X} = \left(\begin{array}{cccc\|c|cccc\|c} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right)$$
$$\bar{Z} = \left(\begin{array}{cccc\|c|cccc\|c} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

where we show the block structure of these matrices (note that here $n - k - r = 0$).

Finally the encoding and decoding circuits for this code are shown in Figures 15 and 16, respectively. Note that the decoding circuit only shows step 3 in the decoding.

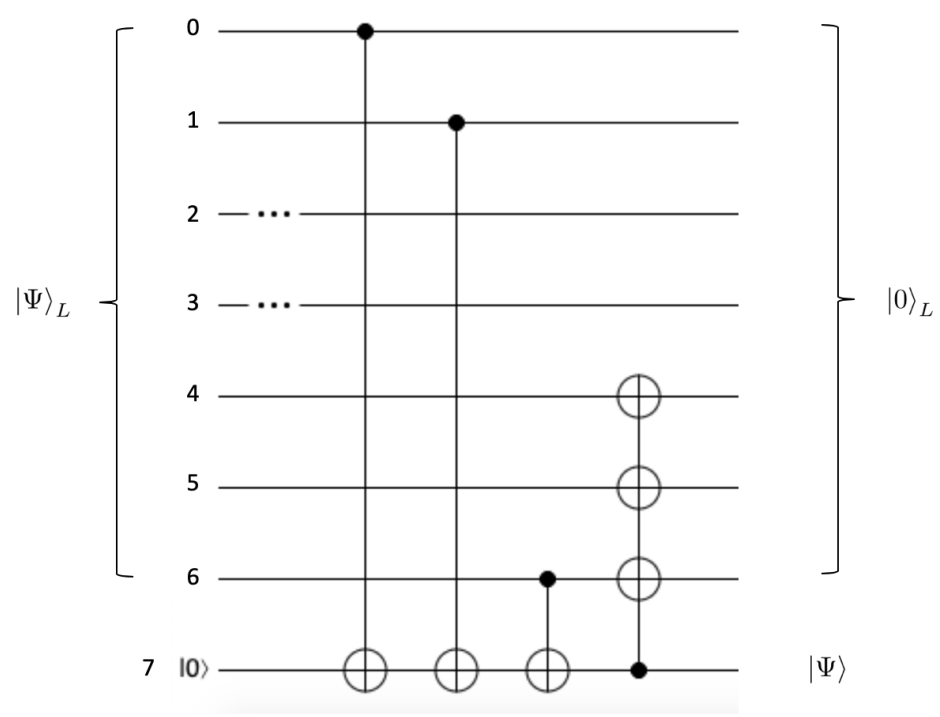


Figure 14: Decoding circuit for Steane code (only step 3).

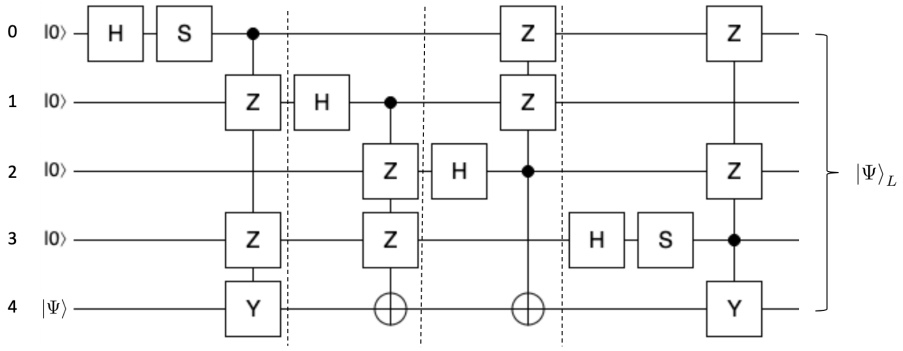


Figure 15: Encoding circuit for the five qubit code.

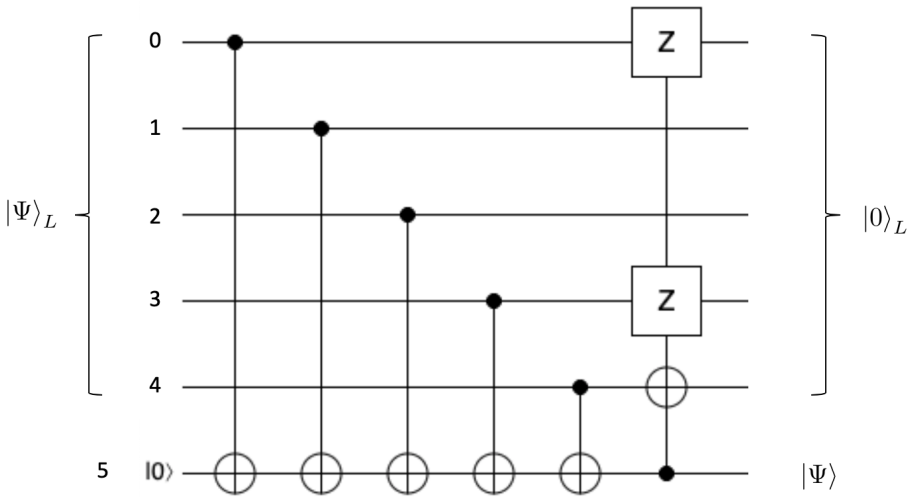


Figure 16: Decoding circuit for the five qubit code (only step 3).