5 Elements of fault-tolerant quantum computation with planar surface codes

5.1 Overview

For an architecture to achieve fault-tolerant quantum computation a number of criteria need to be satisfied. There are different ways such criteria can be specified, and different architectural approaches. Here we shall focus on an approach to fault tolerant quantum computing called the “magic state distillation” architecture. Such an architecture needs to include an encoding of quantum bits which:

- allows repeated fault tolerant detection and correction of errors
- allows the distance of the code to be varied and the number of encoded qubits to be varied, including
  - preparing encoded ancilla qubits in the states $|0\rangle$ and $|+\rangle$,
  - and measuring encoded qubits in the $Z$ and $X$ basis
- provides a way to perform perform $CNOT$ and $Hadamard$ on encoded qubits fault-tolerantly - i.e. protected by the error detection and correction process.
- allows one to “inject” or prepare qubits in other states, in particular so-called magic states.

Planar surface codes provide the basis of an architecture for fault tolerant quantum computation which satisfies all of these requirements. In this final part of the lecture course we will give an overview of how this is achieved.
The planar surface code architecture has a number of advantages with respect to rival fault-tolerance approaches. It has a very high error threshold (on the order of 1% error rate for standard error models), it has a planar structure, which is formed of repeated modules each with identical structure, and no “long-distance” quantum gates are required - every quantum gate need be implemented only between spatially neighbouring qubits. These features make it arguably the leading architecture for realising fault tolerant quantum computation at the present time. The principal disadvantage of this architecture is the high overhead (the number of physical qubits needed for every logical qubit).

Due to time constraints, we will not be able to cover, in detail, all aspects of this model, but will focus on certain key ideas - state injection, repeated error correction, and code deformation. For a detailed and clearly written introduction to this architecture I recommend, Austin Fowler et al, *Surface codes: Towards practical large-scale quantum computation*, http://arxiv.org/abs/1208.0928.

5.2 Achieving universal quantum computation

5.2.1 State injection

Fault tolerant architectures are usually limited in the number of quantum gates they can fault tolerantly support. Often, such gates are confined to the *Clifford group* of gates, containing gates such as (and generated by) $H$, the Pauli operators, CNOT and the single qubit phase gate $S = \text{diag}(1, i)$.

\[
S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}
\]  

Fault tolerant encoded qubit preparation is usually limited to a small family of states ($|0\rangle$ and $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$) and small family of measurements $X$, $Z$ basis.

The above operations all belong to the set of stabilizer operations (since they can be described within the stabilizer formalism). Stabilizer operations on their own do not achieve universal quantum computation - indeed it is possible to efficiently simulate them classically (the Gottesman-Knill theorem).

Remarkably, if the Clifford group is augmented by any single extra unitary quantum gate and its inverse, it becomes universal. Usually the most
5.2. Achieving universal quantum computation

convenient gate to choose is the so-called \( T \)-gate (or “\( \pi/8 \)” gate)

\[
T = \begin{bmatrix}
1 & 0 \\
0 & e^{i\pi/4}
\end{bmatrix}
\] (5.2)

Note that \( T^2 = S \) and that \( T^\dagger = TSZ \). The addition of the \( T \)-gate promotes the stabilizer operations to universal quantum computation.

One method of achieving a \( T \)-gate is via state injection, where the gate is achieved via a combination of ancilla preparation and stabilizer operations implements the gate. Consider the ancilla state \( U_z|+\rangle \) where \( U \) is any single qubit unitary which commutes with \( Z \). This state can be “injected” into a circuit which implements \( U \) via the following circuit:

\[
|\psi\rangle \quad X \quad Z \quad (U_zXU_z^\dagger)^m \quad U_z|\psi\rangle
\]

where, here, the dotted box represents a measurement of observable \( Z \), with the output of the measurement \((-1)^m\) encoded in bit value \( m \). The “correction” operator \((U_zXU_z^\dagger)^m\) must be applied if \( m = 1 \). If we only have access to Clifford group unitaries, to make this deterministic \((U_zXU_z^\dagger)^m\) must itself be in the Clifford group. This property is satisfied only in special cases, but, in particular is satisfied by \( T \), due to the identity

\[
TXT^\dagger = e^{-i\pi/4}YS
\] (5.3)

where global phases can be ignored. Thus, provided the corrections \( Y \) and \( S \), which are both Clifford gates, can be implemented, we can implement \( T \) deterministically via injection of the state \( T|+\rangle \).

In the surface code computation that we introduce below, we will not have a full set of Clifford group gates, and not be able to directly implement the \( S \)-gate directly. Fortunately, since \( SXS^\dagger = Y \), \( S \) can itself also be achieved with state injection. Notice that the state injection circuit only requires a CNOT, a measurement in the \( Z \) basis and the implementation of the correction. The state \( S|+\rangle \) can be injected to achieve the desired \( S \) gates.
5.2.2 Magic state distillation

The state injection procedure enables universal quantum computation with this limited set of gates, provided $S|+\rangle$ and $T|+\rangle$ can be prepared. Unfortunately, the preparation of these states is not within our set of fault tolerant operations. The process will lead to the preparation of noisy approximations to these states.

Fortunately, a process called magic state injection can fix this. It is a distillation procedure which distills states very close to $T|+\rangle$ and $S|+\rangle$ given noisy copies of such states. Furthermore it requires only Clifford gates for its implementation (and does not need the $S$ gate). It can therefore be implemented fault tolerantly with the gate set available here.


Magic state distillation is very robust to noise, and thus can tolerate noise far higher than fault tolerant quantum computing. Thus, if we are above the fault tolerance threshold, magic state distillation will succeed.

5.3 Fault tolerance

5.3.1 Repeated error correction with error-prone measurements

How much noise can be tolerated? We have seen that the code error threshold for the planar surface code is approximately 11%. This computation, however, assumed that measurements were error free – a particularly strong assumption when we recall that these measurements are projecting into the code space, and effectively “making” the code. A more pragmatic threshold analysis – which would demonstrate true fault tolerance – would include errors in all components. In particular considering errors in the stabilizer generator measurements.

Remarkably, the surface codes can tolerate errors in those measurements in almost exactly the same way that they tolerate errors on the qubits. Here we will not consider a detailed microscopic error model, but a simple one. We assume that each measurement, independently with probability $p$, returns the opposite value to the correct outcome - e.g. if it should have returned $+1/0$ - it returned $-1/1$. We assume that these errors are “memoryless” - e.g.
that the presence of such an error at time $t$ does not change the probability of such an error at time $t + 1$.

There is a simple way to combat such errors, repeated stabilizer measurement. One would want to repeatedly make stabilizer measurements in any case, to correct errors before they build up too much, so this is easily incorporated. The most naive way to use repeated measurements, would be to then consider the majority vote of such measurements - but this would not be effective if the “correct” error syndrome changed between these measurements, which you might expect if the measurement rate was of similar magnitude to the error rate. Surface codes provide a more elegant and more effective way.

5.3.2 Three-dimensional homology from error-prone measurements

Consider the two-dimensional surface - and for now, let us just consider vertex stabilizer measurements. Represent the outcomes of these measurements at time $t$ as a 2-dimensional surface. Now consider the outcomes of the next round of measurements, at time $(t + 1)$. These also represent a 2-dimensional surface. Now consider a 2-D surface representing the outcome-wise difference between these rounds - place a 0 on every vertex where the outcome is the same as the previous measurement and a 1 when the outcome is different.

We now repeat this for many measurement rounds – for each round creating a surface representing the differences with respect to the previous round – and stack these up into a three-dimensional structure.

Let us simplify our mental image of this structure by assuming 2-D planar surface codes. The structure then has the form of a three-dimensional cubic lattice. On every 2-D plane in this structure, the vertices represent the difference between the outcomes of the stabilizer at time $t$ and time $(t − 1)$.

Now consider the effect of different kinds of errors on this 3-D lattice. A single new qubit $Z$ error at time $t$ will trigger the adjacent vertex operators to flip - these operators will change their value at this point in time, and thus this will be recorded as a pair of 1 values on the associated vertices of the plane associated with time $t$ in the 3-D lattice. We shall assume that the error is not corrected, and thus vertices on other planes for other time-slices do not record this.

What is the effect of a faulty measurement? Consider the case first where the measurement should outcome 0 at each step, but reports a faulty outcome at one time-step e.g. 0,0,1,0,0,0 – the corresponding difference is 2,1
values on adjacent vertices in the time direction 0,0,1,1,0,0. The same would happen if the measurement should be recording 1, e.g. 1,1,0,1,1,1 – would be represented as 0,0,1,1,0,0,0. Thus fault measurements generate pairs of 1 values on our lattice in adjacent vertices in the time direction!

Qubit errors and measurement errors have almost identical effects on the 3-D lattice, qubit errors flip a single pair of vertices adjacent on a plane (so space-like neighbours), while measurement errors flip vertices on adjacent vertices in the time-like direction.

The resulting 3-dimensional structure can be understood (and error corrected!) in homological terms. Both kinds of errors, can be represented as a 1-chain in the 3-d lattice. 1-Cells which lie along a “space-like plane” represent physical errors, while 1-cells which lie in the “time-like” direction, represent measurement errors. The syndrome differences at each time step,
now correspond to the 0-chain boundary of this 1-chain.

Homologically, qubit errors and measurement errors are not distinguished. They can thus be corrected in the same way. A combination of qubit and measurement errors will create a string of 1-cells which is both space-like and time-like, but homologically, it is just a string like any other. The same error correction decoding strategies will work on the 3-D lattice. The optimal decoder will compute the most likely homology class of the error chain (since again, any correction chain homologically equivalent to the error chain will correct it) while the minimum weight perfect matching algorithm will find the most likely error chain.

For an independent noise-model, the threshold for minimum weight perfect matching on this model was found to be 2.9%. More detailed and physically justified error models also report thresholds with error rates on the order of 1%. The fault tolerance threshold for surface codes is thus the one of the highest reported (the only competing architecture with a similarly high threshold is the teleportation-based architecture of Knill [http://www.nature.com/nature/journal/v434/n7029/full/nature03350.html]).

5.4 Code deformation

The surface code can thus protect quantum information in a fault tolerant manner, provided measurements are made repeatedly. To demonstrate universal quantum computation, we also need the ability to initialise qubits, “inject” states and perform a limited set of quantum gates on them. Code deformation is the key technique to many of these.

The planar surface code architecture we are considering encodes quantum information in twin primal hole qubits and also twin dual hole qubits – as introduced in the previous section.

Recall that the distance of these codes depends on the separation of the codes and their circumference. In fault tolerant quantum computing it can be useful to change the distance of a qubit, increasing or decreasing its protection against error. For example, reducing a qubit’s code distance to 1 effectively unencodes it - allowing one to act directly on the qubit in an unencoded, but unprotected way. This is an important ingredient of state injection which we will meet later on. It is thus useful to be able to change a hole’s size.

It will also prove useful to be able to move holes around. This will allow us to implement some encoded logic gates in a robust manner.
Creating holes, enlarging them, deforming them and moving them are all examples of code deformation. In fault tolerant quantum computing, you should imagine every stabilizer generator being measured at every time step. For the time being, let us assume that any errors detected are immediately corrected. The stabilizer measurements (+ corrections) will therefore repeatedly project the system into the code space.

We can now change the code properties by code deformation. Code deformation amounts to manipulating the code by changing the set of stabilizer generators (which are still being repeatedly measured in between these changes). More specifically, we use a combination of techniques:

- Removing a generator from the generating set – i.e. “turning off” the measurement of that operator.
- Changing a generator’s observable – i.e. changing the corresponding measurement circuit.

5.4.1 Creating a hole

We shall focus here solely on primal holes, but everything that follows can be adapted to dual holes by interchanging $X$ and $Z$ operators, plaquette and vertex generators. Let us imagine that we start with a code defined by a single planar surface with a primal boundary. This surface has first Betti number $\beta_1 = 0$ and encodes no qubits - measurement of the stabilizer operators is thus projecting the into a unique pure stabilizer state. (As above we neglect errors at present, later we will show how they can be corrected). Let us call this state the blank surface state.

We wish to create a primal hole in this surface - and hence create a primal hole qubit. How do the stabilizers of the surface code with a primal hole qubit differ from the hole-less surface? The differ in three ways:

- The plaquette operators for plaquettes inside the primal hole are removed.
- The vertex operators for vertices in the interior of the hole are removed.
- The vertex operators which lie along the (primal) boundary of the hole are modified to lower weight operators, with the $X$ operators which would lie inside the hole deleted from the observable.
Imagine, then that we start off in blank surface state and that measurement operators are being repeatedly measured. What happens if the stabilizer measurements are modified in precisely these ways - i.e. the plaquettes and vertices in the hole “switched off” and the vertices at the edges modified. What will happen to the state?

The repeated stabilizer measurements are now projecting the system into the desired primal hole surface code subspace. Thus the primal hole has been created. But what is the state of the encoded qubit? The encoded logical Z operator for this qubit consists of a product of Z’s around the boundary of the hole. But this operator (representing the boundary of the hole) is equal to the product of the plaquette stabilizer operators which have been switched off. The product of the outcome of the measurement of these operators is thus equal to the measured eigenvalue of this boundary operator. Since this operator commutes with both the old set of stabilizer generators and the new set, it retains its eigenvalue after the generator measurements have changed. Thus the qubit is initialised in state \(|0\rangle\) or \(|1\rangle\) depending on this measured eigenvalue. In the error-free case considered here, this preparation will always create the \(|0\rangle\) state.

To summarise, holes can be created by “turning off” stabilizer measurements for the plaquette and vertex operators in the interior, and modification of the plaquette operators around the boundary of the hole. The logical value of the qubit will correspond to the product of the outcomes of the plaquette operators in the hole interior immediately prior to being switched off.

### 5.4.2 Extending a primal hole

Now that we have created a hole, we may wish to enlarge it, e.g. to increase the qubit code distance. We can do this by, as above, modifying the (repeatedly measured) stabilizer generators.

To enlarge the hole by one plaquette, we must turn off that plaquette operator, and modify the vertex operators along the new boundary of the hole as appropriate.

The effect of this is to increase the size of the hole, while maintaining the logical encoded qubit associated with a hole.

One can verify that the remaining encoded logical operators for the qubit commute with the modified vertex operators and the removed plaquette, meaning they remain unchanged (and unmeasured by this process).
5.4.3 Contracting a primal hole

Now suppose we wish to make a hole smaller, e.g. to measure a logical qubit, or to remove it from the surface entirely. This can be achieved by the reverse steps of extending the hole. The plaquette operator which is to be removed from the whole is “switched back on”, and the vertex operators on the part of the hole boundary which has changed are updated accordingly.

It can be verified (similarly to above) that this makes the hole smaller without measuring or affecting the encoded logical qubit.

5.4.4 Moving a hole

Holes can now be moved via a combination of expansion and contraction. Moving the hole may change the qubits code distance but will not measure the qubit. Perhaps surprisingly, then, motion does not always leave the logical state of the qubit unchanged. We will see below, that moving holes relative to one another – in particular braiding holes around each other – can be used to implement certain encoded logical quantum gates.

5.4.5 Preparation of a double primal hole qubit

The fault tolerant scheme we consider encodes qubits as double primal holes and double dual holes. We can use code-deformation to create a double hole qubit and initialise it in the state $|0\rangle$. We switch off the plaquette and vertex generators in the interior of the holes and modify the vertex operators at the boundary. The distance of the qubit is set by the circumference of the holes and their separation.

Double dual hole qubits may be prepared in an analogous way (interchanging plaquette for vertex operators in the above description). If there is no error, their initial state will correspond to $|+\rangle$.

Since the only encoded qubits we will consider in this chapter are double primal hole qubits and double dual hole qubits, we will refer to them as primal qubits and dual qubits for short.

Being able to prepare ancilliary qubits in the $|0\rangle$ state and the $|1\rangle$ state is an important component of fault tolerant quantum computation.
5.4.6 Measurement of a logical qubit

Fault tolerant measurement of qubits is performed in the reverse way to their creation. A primal qubit is measured in $Z$ when its logical $Z$ operator is measured. Switching on the plaquette and vertex stabilizers inside the holes, while modifying the vertex operators on the edge back to their “hole-less” configuration, removes the qubit, but also measures it in the $Z$ basis. The measured eigenvalue will be the product of the “switched on” plaquettes which were previously in the interior of the hole. The product of plaquette outcomes from each hole should give the same outcome (recall our encoding is equivalent to a repetition code). If not, an error has been detected and that qubit measurement outcome lost.

5.4.7 A braided CNOT gate

Another important component of fault tolerant quantum computation is, of course, the execution of encoded logical gates. So far in this course, we have focussed on the error correcting properties of surface codes and not considered any logical gates.

Here, we shall see an example of a gate implemented entirely through code-deformation. A CNOT gate between a primal and a dual qubit is implemented by merely braiding one of the holes of the first qubit between the holes of the second qubit.

To see why this implements a CNOT, we should work in the so-called Heisenberg picture of quantum computation. When a unitary $U$ acts on a quantum register it equivalently transforms a logical Pauli operator $\sigma$ for that register via conjugation, i.e.

$$\sigma \rightarrow U\sigma U^\dagger$$

(5.4)

We can therefore describe the action of a gate via the way it transforms the logical operators. Let $X_1$ and $X_2$, $Z_1$ and $Z_2$ be logical $X$ and $Z$ operators for qubits 1 and 2, and let qubit 1 be control and qubit 2 be target. The CNOT gate transforms these operators as follows:

$$X_1 \rightarrow X_1X_2 \quad Z_1 \rightarrow Z_1$$
$$X_2 \rightarrow X_2 \quad Z_2 \rightarrow Z_1Z_2$$

(5.5)
Figure 5.2: This braiding transformation (achieved via code deformation) implements a CNOT gate between a primal qubit and a dual qubit. The transformation is illustrated here for logical $X_1$, once the braid has occurred this logical operator is now wrapped around and between the dual holes. This “wrapped” operator is homologically equivalent to $X_1 X_2$. A suitable stabilizer operator to “split” the “entangled” $X$-cochain into the “string” representing $X_1$ and the “loop” representing $X_2$ is illustrated as a darker green loop.
This transformation is demonstrated in figure [5.2]. Braiding the primal hole between the two dual holes by code deformation is a continuous deformation which leaves the logical $X_1$ wrapped between the two dual holes. The operator is now homologically equivalent to the $X_1 X_2$ operator - the transformation of a CNOT.

Similarly one can show that the $Z_2$ operator becomes similarly “entangled” by the braid, and transforms to the $Z_1 Z_2$ operator, while the loop-like $Z_1$ and $X_2$ operators are left unchanged by the braid.

Performing quantum gates by braiding is a key method of topological quantum computing with anyons, and this illustrates that the holes in the surface code have anyonic statistics.

The braided CNOT is an elegant fault tolerant realisation of a quantum gate through code deformation. However, it has a disadvantage - it can only be applied between a primal and a dual qubit.

In fault tolerant quantum computing we’d like to be able to perform CNOT gates between any 2 qubits. Fortunately, this can be achieved in a relatively straightforward manner. It requires some ingredients we haven’t seen yet - measurements and preparations of primal qubits in the $|\pm\rangle$ basis and dual qubits in the $|0\rangle, |1\rangle$ basis.

### 5.4.8 X-basis measurement on primal qubits

The braided CNOT demonstrated an element of anyon-like behaviour in the surface holes. If you are familiar with anyons you will know that, in addition to braiding, they can be fused and pairs of anyons created from the vacuum. The measurement of logical $X$ for a primal qubit (a double primal hole qubit) is reminiscent (to me at least!) of anyon fusion.

Recall that the encoded logical $X$ for this qubit consists of stringlike $X$-cochain operators stretching from one hole’s boundary to the other hole. We don’t want to measure that $d + 1$ qubit operator directly, as we wish for our measurements to remain low weight. If we could measure these $d + 1$ qubits individual in $X$ their product would reveal the value of logical $X$. However such individual measurements would conflict with (since they anticommute with) the plaquette measurements on adjacent plaquettes, and be detected as errors.

The solution then is to “merge” or “fuse” the two holes, joining them together by disabling the plaquette measurements on the plaquettes between
them, and at the same time modifying the vertex measurements on the new boundary.

After this “merging” has been done, the qubits inside the hole are no longer entangled with the remaining code qubits, and error detection measurements are no longer being applied to them. These are precisely the qubits we wanted to measure individually in $X$. We can now do this, and the product of the outcomes of the individual $X$ measurement will reveal the measured eigenvalue of the logical $X$ operator we wished to measure.

The hole that is left over can then be shrunk and finally removed altogether using the standard code deformation methods described above.

This method appears to leave the qubits in the code vulnerable – how can they be protected when they are no longer entangled in the code. It is an illustration of the robustness of code deformation that errors on these qubits can still be detected via standard surface code error detection.

We need only consider $Z$ errors since only these can affect the outcome of our logical $X$ measurement. A $Z$ error on one of the qubits which is individually measured will flip its outcome. If this qubit was still part of the code, the error would flip the outcomes adjacent vertex operators and be detected. After code deformation, these vertex operators no longer measure the qubits in question. However, we are measuring the qubits individually in $X$. Multiplying this outcome with the outcomes of the adjacent vertex operators, we can reconstruct the equivalent outcomes of full 4-qubit vertex operators. Thus $Z$ errors on the qubits measured individually $X$ can still be corrected using the standard surface code technique.

Measurement in the $Z$ basis on dual qubits can be achieved in an analogous manner.

5.4.9 Preparation of $|\pm\rangle$ on a primal qubit

When we create primal qubits by “switching off” plaquette measurements, as described above, they are prepared in the state $|0\rangle$. We can prepare logical state $|\pm\rangle$ by reversing the actions in the $X$ measurement described in the previous section. We create a primal hole large enough to contain both of the holes in our primal qubits and the “tube” between them which will represent a logical $X$ for the qubit. We prepare the qubits inside the “tube” in the $|+\rangle$ state. We then remove the “tube” by code deformation. The holes now have the form of a primal qubit and that qubit is in state $|+\rangle$. 
5.4. Code deformation

The reason for this is that measured observable, the logical $X$ commutes with all of the stabilizer measurements made, both before and after the tube is removed. Thus the system remains prepared in the $+1$ eigenstate of this operator - the logical $|+\rangle$. Similarly to measurement, errors in this process will be detected via the standard surface code error correction (which, remember, is going on repeatedly, throughout).

Again, the preparation of state $|0\rangle$ for a dual qubit achieved in an analogous manner.

5.4.10 State injection

We have seen how to fault tolerantly prepare $|0\rangle$ and $|+\rangle$. For state injection we need to prepare states of the form $T|+\rangle$ or $S|+\rangle$. We cannot achieve this in a fault tolerant way, but we know that magic state distillation will allow us to distill states arbitrarily close to these states, if noisy versions can be created, and encoded into the code.

To perform state injection, we thus “turn off” part of the protection of the qubit. Recall that minimum weight of logical $X$ for these qubits is equal to the distance between the holes, plus one. If we bring these holes together, so the edges of the holes touch, the distance is then 1. (Note this is different to merging the holes - see figure 5.4.

In the Heisenberg picture, the effect of $T$ is to transform the $X$ operator to $TXT^\dagger$. Thus, if the logical $X$ is reduced to a single qubit $X$, it can be transformed to $TXT^\dagger$ by a single qubit $T$ gate.

Thus state injection proceeds as follows – create two “touching” holes, apply $T$ to the qubit at the place where the holes “touch”, and then increase the code distance, by moving the holes apart, to protect this qubit. This operation injects a noisy copy of $T|+\rangle$ into the circuit. Once the qubit code distance has been increased to the appropriate level, this qubit is protected by the code, but it may have undergone errors while the distance was low, and hence magic state distillation must be used to create a low noise state suitable for use in generating a low-error-rate $T$-gate.

5.4.11 Teleportation from a primal to a dual qubit

We have seen how a CNOT gate can be implemented between a primal and a dual qubit by braided code deformation. We need, however, to be able to
Figure 5.3: The minimum weight of the logical $X$ for a primal qubit is the distance (plus one) between the holes. If the holes are made to touch, the weight is just 1. Then we can apply a logical $S$ or $T$ gate to a qubit prepared in $|+\rangle$ to “inject” the state $T|+\rangle$. The protection can then be restored to the qubit by moving the holes apart by code deformation. This process is not fault tolerant, so magic state distillation is then required to remove the errors from the injected states.
implement CNOTs between arbitrary qubits, so between primal and primal qubits. A simple solution is to use one-bit teleportation.

One-bit teleportation is a simple, but powerful protocol (which can also be used, for example, to derive the state-injection circuit above, and to derive measurement-based quantum computing on cluster states). The following circuit “teleports” the logical qubit from the upper physical qubit to the lower physical qubit.

\[ | + \rangle \xrightarrow{X} | + \rangle \xrightarrow{Z^m} | \psi \rangle \]

where the controlled gate (control-X) is a CNOT, and where the Z correction is implemented if the outcome of the X-measurement is \(-1\).

We see that this can be implemented using CNOT, \(|+\rangle\) state preparation, logical Z operators and X measurement, all ingredients we have introduced above. We can thus implement a CNOT between two primal (or two dual) qubits by first “teleporting” one qubit onto an ancilliary dual qubit, performing the CNOT, and then teleporting back.

5.5 Completing the set of gates

5.5.1 Hadamard Gate

We have now seen all ingredients of fault tolerant quantum computation on the surface code, except one - the Hadamard gate. We did not need this gate to achieve the operations above, but we need it to have a universal set of quantum gates. Fortunately, Hadamard can be achieved in a straightforward way.

We have seen a clear relationship between X and Z operators in the surface codes, representing Z operators on 1-chains and X operators on 1-cochains. We use this to our advantage here. Consider a planar surface code with a single double primal hole qubit. Apply Hadamard gates to every single qubit. We now have a code which looks very similar to the code we had
before, but now where had X’s we have Zs and vice versa. For example, in the stabilizer generators, we have X operators on the boundaries of plaquettes and Z operators on the co-boundary of vertices. Our logical Z has transformed into a cycle of X’s.

We can, however, manipulate the code back to a standard surface code in a simple way - recall that we can map from primal to dual lattice by shifting the lattice down and left by half a cell. If we move our qubits in this way, we shift our new Z stabilizers back onto the boundaries of plaquettes, and our X stabilizers back onto the coboundaries of vertices.

This shift has also transformed our primal boundary to a dual boundary, and thus our primal holes have become dual holes. The former logical Z has transformed to a dual qubit logical X, while the logical X has transformed to a logical Z.

This combination of Hadamard plus half-cell shift completes the logical Hadamard gate, together with the conversion of the qubit from primal to dual.

If we wish to implement this on a surface with many qubits, we would implement the Hadamard on all the logical qubits - not what we’d want to do. However, areas of surface code containing a single qubit can be “detached” from the rest via code deformation. Thus the sequence to achieve a single qubit Hadamard is somewhat - detach qubit’s section of lattice, perform local Hadamards, shift qubits by half a lattice cell, reattach qubit into full lattice.

At the end of this, the primal qubit has been converted to a dual qubit - and a final teleportation step would be needed to restore the qubit back to primal form.

This method of implementing the Hadamard gate works therefore, but is cumbersome. A number of improved approaches have been suggested, in particular, in A. Fowler [http://arxiv.org/abs/1202.2639](http://arxiv.org/abs/1202.2639) an elegant approach to achieving the Hadamard gate is presented almost entirely in figures.

### 5.6 Summary and outlook

We have seen that the planar surface code with dual hole qubits provides all the ingredients needed for fault tolerant quantum computation. Dual hole qubits provide robust storage of qubits protected by the surface code. Error correction robust to measurement errors is achieved homologically, using the
Figure 5.4: A Hadamard is implemented on a surface code with a single double hole qubit via Hadamards on every qubit, plus a displacement left and down by half a cell. This also changes the qubit-type from primal to dual.
same methods as in the perfect-measurement case. Code deformation provides a suite of tools. Qubits can be fault tolerant prepared and measured in $Z$ and $X$ eigenstates, magic state qubits can be injected and then protected, and CNOT gates can be achieved through braiding.

This architecture has a number of significant advantages for experimental realisation. It’s high fault tolerance threshold for errors is within reach of a number of experimental settings. It’s modular architecture on a 2-dimensional surface well placed for scaled up manufacture. It might seem like the ideal way to build a fault tolerant quantum computer, but, there are disadvantages too. The overhead in physical qubits to logical qubits is high. To increase the code distance linearly, the number of qubits must be increased quadratically. Thus formally, the rate of encoded qubits to physical qubits goes to zero as the distance goes to infinity. In addition, magic state distillation itself comes with a high overhead cost. Putting this all together Raussendorf, Harrington and Goyal estimated an overhead of from $10^5$ - $10^1$ physical qubits to logical qubits. It remains an open research question to determine whether the significant advantages of topological codes for fault tolerant quantum computation can be maintained while significantly reducing this overhead, and this may determine whether the planar surface code remains the front running architecture for scalable and fault-tolerant quantum computing.