Controlling a Superconducting Quantum Computer

Daniel Cohen Hillel

Contents

1 Introduction

1.1 What is a Quantum Computer?

We'll assume the reader understands the basics of computing and quantum mechanics, here's a brief overview.
A classical computer is, essentially, a calculator, not of "Regular" numbers but of *binary numbers* [1]. A *binary digit*("*bits*" from now on) can be in one of two states, usually represented by 0 and 1. We can use *logic gates* to control and manipulate bits to do all kinds of calculations[2]. This is the building blocks of the classical computer, with the ability to do calculation with bits, and the ability to store bits in the memory we are able to construct a computer.

So what is a quantum computer then? Well, if the classical computer uses bits to do calculations, a quantum computer uses *quantum bits*("*qubits*" from now on) for it's calculations. A qubit, much the same as a bit, has 2 states, a 0 state and a 1 state(notated $|0\rangle$ and $|1\rangle$ for reasons we'll see later), the difference is that a qubit can be in a *superposition* of the 2 states, so the qubit has essentially and infinite amount of possible states

1.2 Qubits and Quantum Gates

1.3 Algorithms and Further motivation

1.4 Superconducting Quantum Computers

[1] add further reading about binary numbers
[2] Additional information about bit calculation

2 Our System

2.1 The cavity

2.2 The Transmon

2.3 The Oscillator

2.4 Describing the System Mathematically

To describe the system mathematically we are going to calculate it's Hamiltonian, that way we can run simulations of the system with the Schrodinger equation.

We can partition the system into different parts and analyse each part individually, so the total system Hamiltonian will be:

$$H(t) = H_{Oscillator} + H_{Transmon} + H_{Interaction} + H_{Drive}(t)$$

We'll begin by the easy to characterize, Transmon And Oscillator, they are a simple quantum system that can be described by the uppering and lowering operators [3]. We can write:

$$H_{Oscillator} = \omega_C a^\dagger a$$

$$H_{Transmon} = \omega_T b^\dagger b$$

We can find ω_C and ω_T experementally.

The next part to characterize is the interaction(See appendix A), again by the Jaynes–Cummings model, we can write the interaction hamiltonian as:

$$H_{Interaction} = \chi a^\dagger a b^\dagger b$$

Again, we can measure χ experimentally.

Finally, we can characterize the driven and most important part of the system. It can be written as:

$$H_{Drive} = \epsilon_C(t)a + \epsilon_T(t)b + h.c.$$

Or as(expanding the hermitian conjurgate):

$$H_{Drive} = \epsilon_{I_C}(t)(a + a^\dagger) + \epsilon_{Q_C}(t)(a - a^\dagger)i + \epsilon_{I_T}(t)(b + b^\dagger) + \epsilon_{Q_T}(t)(a - a^\dagger)i$$

[3]See appendix A

3 Optimal Control

3.1 What's GRAPE

As explained in previous sections, to control our qubit and manipulate it, we need to send microwave pulses in the cavity. The problem is, what pulse do we send? How does it look? sin? cos? In what frequency? or maybe even some arbitrary wave.

To answer this question, we can model our system on a normal computer and simulate what happens when you send a pulse, then, we can try to change the pulse(in a smart way) until we get the desired effect. So for example, let's say we want to find the wave pulse that corresponds to the NOT gate, we can start by guessing some random wave(constant zero, sin and so on), the random wave probably won't act as a NOT gate, then, we change the wave a little bit many times and on each iteration the pulse acts more and more as a NOT gate.

So what's GRAPE than? The *GR*adient *A*scent *P*ulse *E*ngineering was first proposed in [2]. When we model our system, we treat the wave as a step-wise constant function, so the wave is just an array with many variables and we want to find the best values that give the result that we want. then we set a cost function [4] that tells us how are the values of the wave to give the wanted result. This cost function is a many dimensional function(each step of the wave is a dimension of the cost function), and we can find its gradient. Using the cost function and it's gradient we can use some optimization algorithm(mainly, the L-BFGS-B method) to find the maximum of the cost function that gives us the optimal wave to send to the cavity.

3.2 The Cost Function

So what is this cost function really? ***Fidelity***. It measures the "closeness" of two quantum states and it's a value from 0 to 1. So for example, the fidelity between state $\langle 0|$ and state $\langle 1|$ is equal to 0, because they are the most different two quantum states can be, while for example the fidelity between state $\langle 0|$ and state $\langle 0|$ is equal to 1, because they are the closest two states can be to each other(for a matter of fact, every state has fidelity 1 with itself and end fidelity 0 with an opposite state). But still, how do we calculate the fidelity between two states? well, turns out its very simple, it's just their product [5]. We can write:

$$F(\psi_1, \psi_2) = |\langle \psi_1 | \psi_2 \rangle|^2$$

We want to maximize the fidelity with GRAPE.

In a previous chapter we characterized the Hamiltonian of the system(equation (num)), so now we can use the good old *time-dependent Schrodinger equation*:

$$i\hbar \frac{d}{dt} |\Psi(t)\rangle = \hat{H} |\Psi(t)\rangle$$

[4]discussed in details in the next section

[5]Assuming both states are pure states

And also, the Hamiltonian of the system is in the form[6]:

$$H(t) = H_0 + \sum_k \epsilon_k(t) H_k$$

Because of the way this Hamiltonian is built, on each constant step of the wave function, the Hamiltonian is constant, and luckily for us, the solution of the Schrodinger equation for a constant Hamiltonian is pretty simple and given by[7]

$$U(t) = e^{-\frac{i}{\hbar}\int_{T_0}^{T_1} H(t)dt}$$

And because we chose T_0 and T_1 as the end points of a step of the functions, the total Hamiltonian of the system is constant so the integral is just a simple multiplication by $T_1 - T_0$ which we'll write as δt. So the solution is

$$U(t) = e^{-\frac{i\cdot\delta t}{\hbar} H(t)}$$

More then that, to solution over all time is the product of all the solutions for each constant piece. So

$$U(\epsilon(t)) = \prod_{k=1}^{N} U_k$$

Where N is the number of time steps in the simulation(larger N is a more precise simulation).

This way the state of the qubit after the drive is given by:

$$|\Psi_{final}\rangle = U\,|\Psi_{initial}\rangle$$

This way if we want to calculate the fidelity after applying the drives we can simply calculate the fidelity between the wanted state and the final state,

$$F(\epsilon(\vec{t})) = F(\Psi_{target}, \Psi_{final}) = |\langle\Psi_{target}|\,U\,|\Psi_{initial}\rangle|^2$$

Now, theoretically we can use an algorithm to try different waves until we find a wave that does what we want(brute force for example), but this will take to much time and the computation won't finish in any reasonable amount of time. Because of this, we'll want to use a smart search algorithm(such as L-BFGS-B) but to do so we need the gradient of the cost function(the variables of the cost function are the values of the steps of the drive pulses). We can obviously use the finite difference method to calculate the gradient but this method is heavy on the computation and has a lot of over head. We'll take a smarter approach to calculating the gradient.

We can look at the expression,

$$c = \langle\Psi_{target}|\Psi_{final}\rangle = \langle\Psi_{target}|\,U\,|\Psi_{initial}\rangle$$

[6]Further details were given in section 2.4

[7]Add a simple justification

We want to differentiate this expression by each control parameter. U is defined as:

$$U = U_N U_{n-1} ... U_2 U_1$$

And when differentiating by a control parameter only one U_k is affected, so we can write,

$$\frac{dc}{d\epsilon_k} = \langle \Psi_{target} | \, U_N U_{N-1} ... \frac{dU}{d\epsilon_k} ... U_2 U_1 \, | \Psi_{initial} \rangle$$

Assuming very small time steps we can derive U_k over ϵ_k,

$$U_k = e^{-\frac{i \cdot \delta t}{\hbar} H(t)}$$
$$\frac{dU_k}{d\epsilon_k} = \frac{i\delta t}{\hbar} \frac{dH}{d\epsilon_k} U_k$$

We can use this expression as the gradient values but it's still rather complex computationally(N^2 complexity).

We can use a bit different method to calculate the gradient to save on the computation by reducing the overhead.

3.3 Constraints

4 Controlling the FPGA(wave generator)

5 Conclusion