Bachelor Project for The Open University of Israel
Done at The Weizmann Institute of Science

Controlling a Superconducting Quantum Computer

Daniel Cohen Hillel

Supervisor: Dr. Serge Rosenblum

Advanced Project in Physics A (20382)

The Open University of Israel

The Weizmann Institute

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, we can use that property to our advantage to create new type of computation.

1.2 Qubits and Quantum Gates

Mathematically, we think of qubits as 2-dimensional vectors, where the first term corresponds to the $|0\rangle$ state and the second term corresponds to the $|1\rangle$ state, so a qubit in a state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ can be represented as

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

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

In this world of qubits as vectors, we think of logic gates, known as *quantum gates*, as matrices[3], and when the qubit goes through the logic gate, the result is multiplying the matrix by the qubit. Let's see an example for one of the simplest logic gates we have in classical computing, the NOT gate, a quantum implementation of this gate will take $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$), the matrix that achieves this is

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Also called σ_1[4]. We can see that if we, for example, input a $|0\rangle$ into that quantum gate, we get as a result

$$NOT\,|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

As we expected. There are many more 1-qubit quantum gates we can think of(infinite amount actually). The interesting thing is that in classical computing, we only have 4 possible logic gates on a single bit(compared to infinite amount for qubit), these are the identity(do nothing), the NOT, output always 1, and output always 0. The last thing we need to know to understand the basic of quantum computing, is how to treat multiple qubits. If we have sevral of qubits in our system, we think of all the qubits together as one vector that is the *tensor product*[5] of all the qubits, so let's say we have a $|0\rangle$ and $|1\rangle$ qubits in our system, we represent that by $|01\rangle$ and it is equal to

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

And a quantum gate on multiple qubits is simply a larger matrix. The thing about the tensor product is that for N qubits, it has 2^N coefficients(!)

Now that we have the basic tools of quantum computing, we can use them to get motivation for the amazing things quantum computers can do

[3]There is a requirement that the matrices will be unitary
[4]known as the Pauli matrix 1
[5]Kronecker product to be precise

1.3 Algorithms and Further motivation

...

1.4 Superconducting Quantum Computers

The physical implementation of the qubit itself isn't to subject of this project but we can look a bit on how you would implement such a thing. A problem we face when making a quantum computer is would physical phenomena would actually be the qubit. For classical computer we already have this figured out for years, the bit is the voltage on a wire, 1 is one there is voltage on the wire and 0 is if there's none, simple. For a quantum computer this is much more complicated, there are many quantum phenomena we can use as our qubit, such as the energy level of an atom, the spin of an electron, the polarization of photons and so on. This project is about a *superconducting* quantum computer, with superconducting qubits.

Superconducting qubits are microwave circuits in which the cooper-pair condensate effectively behaves as a one-dimensional quantized particle. By inserting Josephson junctions, the circuit can be made nonlinear, allowing us to isolate the qubit from higher-energy levels and treat it as a two level system(instead of the many level system that it really is).

2 Our System

2.1 The cavity

2.2 The Transmon

2.3 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:

$$H(t) = H_{Oscillator} + H_{Transmon} + H_{Interaction} + H_{Drive}(t) \tag{2.1}$$

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 [6]. We can write:

$$H_{Oscillator} = \omega_C a^\dagger a \tag{2.2}$$

$$H_{Transmon} = \omega_T b^\dagger b \tag{2.3}$$

We can find ω_C and ω_T experimentally.

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

$$H_{Interaction} = \chi a^\dagger a b^\dagger b \tag{2.4}$$

Again, we can measure χ experimentally.

[6] See appendix A

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

$$H_{Drive} = \epsilon_C(t)a + \epsilon_T(t)b + h.c. \tag{2.5}$$

Or as(expanding the hermitian conjurgate):

$$H_{Drive} = \epsilon_{I_C}(t)(a + a^\dagger) + \epsilon_{Q_C}(t)(a - a^\dagger)i + \epsilon_{I_T}(t)(b + b^\dagger) + \epsilon_{Q_T}(t)(a - a^\dagger)i \tag{2.6}$$

each ϵ is a pulse that we will optimize with GRAPE as described in much details in the next chapter.

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 [7] 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 varies between 0 and 1. So for example, the fidelity between state $|0\rangle$ and state $|1\rangle$ is equal to 0, because they are the most different two quantum states can be, while

[7] discussed in details in the next section

for example the fidelity between state $|0\rangle$ and state $|0\rangle$ 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 [8]. We can write:

$$F(\psi_1, \psi_2) = |\langle\psi_1|\psi_2\rangle|^2 \tag{3.1}$$

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*:

$$i\hbar\frac{d}{dt}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle \tag{3.2}$$

And also, the Hamiltonian of the system is in the form[9]:

$$H(t) = H_0 + \sum_k \epsilon_k(t)H_k \tag{3.3}$$

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[10]

$$U(t) = e^{-\frac{i}{\hbar}\int_{T_0}^{T_1} H(t)dt} \tag{3.4}$$

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

$$U(t) = e^{-\frac{i\cdot\delta t}{\hbar}H(t)} \tag{3.5}$$

[8] Assuming both states are pure states
[9] Further details were given in section 2.4
[10] Add a simple justification

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

$$U(\epsilon(t)) = \prod_{k=1}^{N} U_k \tag{3.6}$$

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:

$$|\Psi_{final}\rangle = U\,|\Psi_{initial}\rangle \tag{3.7}$$

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,

$$F(\epsilon(\vec{t})) = F(\Psi_{target}, \Psi_{final}) = |\langle\Psi_{target}|\,U\,|\Psi_{initial}\rangle|^2 \tag{3.8}$$

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,

$$c = \langle\Psi_{target}|\Psi_{final}\rangle = \langle\Psi_{target}|\,U\,|\Psi_{initial}\rangle \tag{3.9}$$

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,

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

We can write that for a constant Hamiltonian(from the schrodinger equation)

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

We can approximate the derivative $\frac{\partial U_k}{\partial \epsilon_k}$ in the limit of small δt by writing

$$\frac{\partial U_k}{\partial \epsilon_k} \approx -\frac{i \cdot \delta t}{\hbar} \frac{\partial H}{\partial \epsilon_k} \cdot e^{-\frac{i \cdot \delta t}{\hbar} H(t)} = -\frac{i \cdot \delta t}{\hbar} \frac{\partial H}{\partial \epsilon_k} U_k$$

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

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

Now the derivitive of the cost function by an amplitude of the wave has become

$$\frac{dc}{d\epsilon_k} = -\frac{i \cdot \delta t}{\hbar} \langle \Psi_{target} | \, U_N U_{N-1} ... U_{k+1} \frac{dH}{d\epsilon_k} U_k ... U_2 U_1 \, | \Psi_{initial} \rangle \tag{3.10}$$

Let's define 2 wave wave functions ψ_{bwd} and ψ_{fwd}, they will be the multiplication component before and after the derivative of H, so:

$$\frac{\partial c}{\partial \epsilon_k} = -\frac{i \cdot \delta t}{\hbar} \left\langle \psi_{bwd}^{(k+1)} \right| \frac{\partial H}{\partial \epsilon_k} \left| \psi_{fwd}^{(k)} \right\rangle \tag{3.11}$$

We can easily see from 3.10 that

$$\left| \psi_{fwd}^{(k)} \right\rangle = \begin{cases} |\psi_{init}\rangle & k = 0 \\ U_k \left| \psi_{fwd}^{(k-1)} \right\rangle & otherwise \end{cases}$$

$$\left| \psi_{bwd}^{(k)} \right\rangle = \begin{cases} |\psi_{targ}\rangle & k = N+1 \\ U_k^\dagger \left| \psi_{bwd}^{(k+1)} \right\rangle & otherwise \end{cases}$$

Now all we need is to do $2N$ calculations in the beginning(N for bwd and N for fwd)

then calculating the actual gradient is trivial multiplication from equation 3.11. This improves the computation complexity in an order of magnitude($o(N^2)$ to $o(N)$) and the memory complexity by is still in the same order of magnitude($o(N)$ to $o(3N)$).

It's important to note that c is not the fidelity, but the overlap. We can get the fidelity from c like so [11]

$$F = |c|^2$$

since c might be complex this derivative is a bit less trivial than it might look like. We can write $c(\vec{\epsilon})$ as $a(\vec{\epsilon}) + b(\vec{\epsilon})i$, where $a, b \in R$ and we get that

$$\frac{\partial F}{\partial \epsilon_k} = \frac{\partial |c|^2}{\partial \epsilon_k} = \frac{\partial ||a+bi|^2}{\partial \epsilon_k} = \frac{\partial (a^2+b^2)}{\partial \epsilon_k} = 2(a\frac{\partial a}{\partial \epsilon_k} + b\frac{\partial b}{\partial \epsilon_k})$$

We can notice that[12] $c(\frac{\partial c}{\partial \epsilon_k})^* = a\frac{\partial a}{\partial \epsilon_k} + b\frac{\partial b}{\partial \epsilon_k} + (ab - \frac{\partial a}{\partial \epsilon_k}\frac{\partial b}{\partial \epsilon_k})i$, more importantly we can see that the real part of that expression is exactly what we need, putting it all into one formula we get

$$\frac{\partial F}{\partial \epsilon_k} = 2 \cdot \mathrm{Re}\left\{ c(\frac{\partial c}{\partial \epsilon_k})^* \right\} \tag{3.12}$$

Now all you need is to plug 3.11 and 3.9 into 3.12 and you got your gradient :)

3.3 Constraints

Having our simulation doing whatever it wants is nice and all but still unfortunally, we're living in the real world, and in the real world we can't just make ultra-fast frequencies at close to infinite power, it's just impossible because of the limitation of our devices. We need to add constraints to the cost function so we won't get solutions that use too much power or that change too rapidly.

We define a set of constraints on the solution $g_i \geq 0$(ideally $g_i = 0$). We can associate a Lagrange multiplier λ_i to each constraint.
Now our goal is to maximize

$$F(\vec{\epsilon}) - \sum_i \lambda_i g_i(\vec{\epsilon})$$

Let's add a constraint to each of the most problematic physical limitation

[11] See initial definitions of c and the fidelity(3.9 and 3.1 respectively)
[12] $(\frac{\partial c}{\partial \epsilon_k})^*$ is the complex conjugate of the derivative of c

3.3.1 Limiting the Pulse Amplitude

This is the most obvious physical limitation, we can't generate pulses with infinite energy, so we have to restrict it. There are two ways we can do so, the first is to create a hard cut-of amplitude, no matter what, the amplitude will never go above this amplitude, this will usually be the limit of our pulse generator. But normally we don't want our generator to work at it's absolute limit[13], so we can add also a soft amplitude maximum by "rewarding" the cost function to stay at a lower amplitude. Let's see how we would implement such a thing, starting with the hard cut-off.

Instead of controlling and changing the amplitude($\vec{\epsilon}$) directly, we'll introduce a variable $\vec{x}$ and relate them as

$$\vec{\epsilon} = \epsilon_{max} \tanh \vec{x}$$

As you properly guessed, ϵ_{max} is the maximum amplitude of the pulse. Since the optimization algorithm can only change $\vec{x}$, the amplitude of the pulse will always be between $-\epsilon_{max}$ and ϵ_{max}. Unluckily for us, this changes the gradient of the cost function since we now want the derivative with respect to $\vec{x}$ instead of $\vec{\epsilon}$. We can relate the two

$$\frac{\partial F}{\partial \vec{x}} = \frac{\partial F}{\partial \vec{\epsilon}} \frac{\partial \vec{\epsilon}}{\partial \vec{x}} = \frac{\epsilon_{max}}{\cosh^2 \vec{x}} \frac{\partial F}{\partial \vec{\epsilon}}$$

We can use the derivative $\frac{\partial F}{\partial \vec{\epsilon}}$ we got from 3.12 and simply calculate $\frac{\epsilon_{max}}{\cosh^2 \vec{x}}$ and we again have the gradient.

For the soft limitation, there are two limitation we can make, linear and non-linear. The linear goal is for general preference of low amplitude pulses and the non-linear is if you have a specific $\epsilon_{max,soft}$ that you want to be well below of(You can use both or either one depends on your desired pulse properties). We'll start with the linear case since it's simpler.

For the linear amplitude penalty all we want is that *bigger amplitudes* $\Rightarrow$ *bigger cost function*, since our algorithm seeks to minimize the cost function, this will lead to the overall amplitude being smaller. The way we do so is simple, we can define a constraint

[13]Not only that it might damage the device but also with stronger pulses the non-linear optics effects increase and then it's no fun :(

$g_{amp,lin}$ that sums all the amplitudes of the steps of the pulse, so

$$g_{amp,lin} = \sum_k |\epsilon_k|^2$$

and we maximize the cost function as explained at the beginning of the section.

Still, since it is added to our cost function we need to find the gradient of the penalty as well. In the case it's rather simple since it's a basic parabola

$$\frac{\partial g_{amp,lin}}{\partial \epsilon_k} = 2\epsilon_k$$

and now we have all we need in order to add this penalty to our cost function. Let's move on to the non-liner penalty.

For the non-linear amplitude penalty we have a slightly different goal in mind

3.3.2 Limiting the Pulse Bandwidth and Slope

Again we encounter a physical limitation, we can't produce any pulse we want, there's a limitation of the maximum frequency our AWG(Arbitrary Waveform Generator) can create because the device can't change the voltage instantaneously. Again, like we had with the amplitude limit, there are 2 types of limits we can make, hard and soft. Let's start with the hard limit.

We have some frequency ω_{max} which is the maximum frequency that our AWG can generate. To make sure that our simulation doesn't produce such a pulse we can go from time space to the frequency space with a Fourier transform(Discrete Fourier Transform to be exact).

$$\vec{\epsilon} = (DFT)^{-1}\vec{x}$$

The numerical optimization algorithm controls $\vec{x}$ which is in the frequency space. Now if we want to limit the frequency we can simply set to 0 any frequency that is above our

maximum frequency.[14]

$$x(\omega > \omega_{max}) = 0$$

The gradient of the new cost function is simply the Fourier transform of x

$$\frac{\partial \vec{x}}{\partial \epsilon_k} = (DFT)\frac{\partial \vec{\epsilon}}{\partial \epsilon_k}$$

And we know $\frac{\partial \vec{\epsilon}}{\partial \epsilon_k}$ from previous sections. It's important to note that the hard cut-off of the amplitude and the hard cut-off of the frequency do not work together since one is in the time space and one is in the frequency space. This is not much of a problem since we can compensate with the soft limits that do work well together(mainly since they require adding to the cost function instead of changing coordinate). In my simulations I use the amplitude hard limit instead of the frequency one since it only requires "squishing" the function instead of going from time space to frequency space.

For the soft limits, we're limiting the slope(derivative) of the pulses and not the frequency directly. There are, again, two types of limits we can make, a linear limit and a non-linear limit. The linear limit simply incentives for a lower maximum slope overall, and the non-linear limit incentives a specific maximum soft limit on the slope, we can use the two together. Let's start with the simpler, linear limit.

The slope of a step function is simply $\epsilon_{k+1} - \epsilon_k$, we want to limit the size of the slope so we'll look at the expression $|\epsilon_{k+1} - \epsilon_k|^2$ instead. Summing all the slopes(to get an overall slope size of the entire pulse) we get the expression[15]

$$g_{slope,lin} = \sum_{k=0}^{N-1} |\epsilon_{k+1} - \epsilon_k|^2 \tag{3.13}$$

Remember that for each limit we associate a number $\lambda_{slope,lin}$ that is the "strength" of the limit, and that the cost function now becomes: $*old\ cost\ function* + \lambda_{slope,lin} g_{slope,lin}$, so we need to calculate the gradient of this limit and add it to the total gradient of cost

[14]Might be more efficient to set these frequency to 0 and not let the optimization algorithm to try to change them(living the completely out of the pulse then putting them back in as 0 after the optimization is finished)

[15]Note that we have a problem at the edges since the slope of the end points is not well defined, we'll fix this problem later but for now we just ignore the last point $k = N$

function.

Unlike the amplitude, since the slope of the boundaries is not well defined we'll have the edges defined differently then the center of the pulse. The gradient of g in the center is a simple derivative, notice that each ϵ_k appears only twice in the sum

$$\frac{\partial g_{slope,lin}}{\partial \epsilon_k} = 4\epsilon_k - 2(\epsilon k + 1 + \epsilon k - 1)$$

It's nice to see that the expression looks like how'd you numerically estimate the second derivative, since the gradient of the slope(which is the derivative) is the second derivative, this is nice and reassuring :). Now we need to define the gradient at the edges, you can see that the derivative of ϵ_k depends on his +neighbors on both sides, since the first and last element of the pulse don't have 2 neighbors they are treated a little differently. each of the edges appears only once in the sum 3.13 unlike the others that appear twice, we can simply take the derivative of that one term and get

$$\frac{\partial g_{slope,lin}}{\partial \epsilon_0} = 2(\epsilon_1 - \epsilon_0)$$
$$\frac{\partial g_{slope,lin}}{\partial \epsilon_N} = 2(\epsilon_N - \epsilon_{N-1})$$

And now the slope soft linear limit is defined and so is it's gradient.

Now, before we continue to the non linear limit, we'll add another small constraint that will also solve the problem of the slope at the boundaries(you can think of it as a sort of boundary condition). It might seem weird at first, but we want to pulse to zero-out at the edges(the amplitude of the first and last steps of the pulse being equal to 0), this is since our AWG device can't immediately start a pulse with some amplitude, it can't get from 0 to that amplitude instantaneously(for the same reason we limit the slope in the first place). This could be achieved by simply setting the first and last steps of the pulse and their gradient to 0.

$$\epsilon_0 = \epsilon_N = \frac{\partial\, Cost}{\partial \epsilon_0} = \frac{\partial\, Cost}{\partial \epsilon_N} = 0$$

This solves the problem we were trying to solve we were having with the slope at the boundaries, since the gradient is 0 at the edges and does not depend on it's neighbors.

We can move on to the non-linear limitation now.

As we mentioned earlier, the goal of the non-linear limitation is to get a specific wanted maximum slope instead of making the slope as small as possible, this is, like in the

3.3.3 Limiting the photon number

"Hilbert space is a big place."

-Name

Here's the thing about the cavity levels, there are infinite amount of them. This might be a problem since our computers can't really deal with infinite amount that well. We can make an assumption that the cavity only has N levels but it is still possible that something happens in the higher levels that may affect the physical result that we didn't include in the simulation. We want to limit that and make sure that everything interesting is contained in the N levels that we have

3.4 Implementing Qubit Operations with GRAPE

3.4.1 DRAG - Imperfect Qubits

When we did all of our calculation on the qubit we didn't include one detail, it's really hard to create one. In the way our qubits are implemented, there are actually more then 2 levels, it's not a 2 level system but we treat it as one since the higher levels are off-resonance[16], but still there's a chance some of the higher levels will get excited by our pulses or some other physical phenomena and we want to account for it. The way we can do so is with the Derivative Removal via Adiabatic Gate(DRAG) algorithm. The idea being we make the qubit in the simulation to be more then a 2 level system, change the Hamiltonian a little bit(as you'll see later) so it accounts for the off-resonance higher levels and use the same grape algorithm to optimize for the entire system and not only the lower 2 levels.

[16]as explained in the beginning

3.5 Implementing Cavity Operations

...

3.6 From States to Gates(?)

Maybe add here how to create a gate-GRAPE from the state-GRAPE

4 Controlling a Superconducting Quantum Computer

4.1 Overview

4.2 The AWG

Consider for a moment what we want to do with our microwave generator to control a qubit, we need to send a high frequency signal and change it by a really small amount compared to its frequency(sending a GHz signal and changing it by MHz, 3 orders of magnitude difference). We can't just take a wave generator that generates a 10GHz signal and tell it to change from 10GHz to 10.0001GHz because it won't be precise enough, and even if the frequency generator is good enough so that we can make really accurate changes to the frequency, we still want to have the wave interact with the quantum system and analyze the result so we also need a super-frequency-analyzer to understand what the hell is going on in the quantum system, not only that, we also want the frequency analyzer and generator to work together(matching the same input with the corresponding output of the system). Now that we understand what are some of the challenges working with such high frequencies, what can we do about it?

Well the solution is simple, we can take a high frequency wave(10GHz for example) and a lower frequency wave(1MHz for example) and mix them together *somehow* to get a 10GHz + 1MHz = 10.0001GHz signal, and in the output of the quantum system we can simply un-mix the result to get back the 10GHz signal and some other signal(in the MHz) that can tell us something about the system.

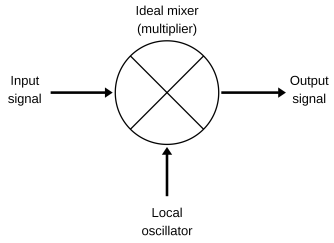
That sounds all nice and good but how do we actually mix/un-mix 2 signals in that way and how can we control that proccess? That's where the IO-Mixer comes in, we'll look into them in just a moment, first we need to understand a regular mixer work.

4.3 The Mixer

The idea is simple, the mixer has 2 inputs and one output, when you enter 2 waves as an input, you get their product as the output(inputting for example $\cos(t)$ and $\cos(2t)$ will result in $\cos(t)cos(2t)$ at the output). We look more into mixers, couplers and their implementation in appendix B.

We draw a mixer in a diagram like so,

Figure 4.1: Ideal mixer in a diagram

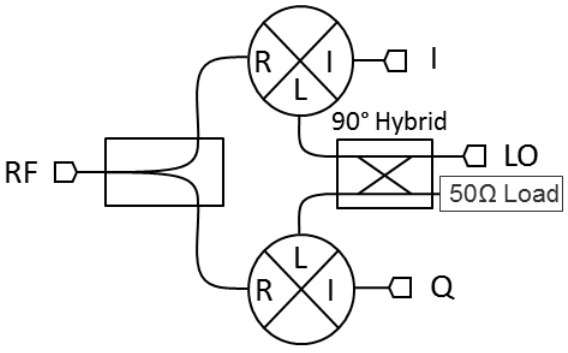


We can change what are the outputs and what are the inputs to get different ways for the mixer to work

4.4 The IQ-Mixer

We've seen what's a *regular*(and *ideal*) mixer is, but how can we use it for the desired effect? remember, we want to input a high frequency and a lower frequency and we want the output to be a wave with a frequency that is the sum of the 2 frequencies. To do so, we can consider the following diagram

Figure 4.2: The IQ mixer



Where the *90°hybrid* in the diagram is simply a 90°hybrid coupler, what it does is that it splits the signal to 2 waves at a 90°phase difference, we look into those in appendix B. The square near the RF sign simply adds the 2 waves, we also look into it in the appendix

As we can see, the IQ mixer has 3 inputs, I, Q(hence the name) and LO. We can also see that there's only one output, RF(although you can play with what are the inputs and what are the outputs).

How can we use this IQ mixer to add frequencies? Let's consider the following inputs[17]

$$I - -- > \cos(\omega_{IQ}t)$$
$$Q - -- > \sin(\omega_{IQ}t)$$
$$LO - -- > \sin(\omega_{LO}t)$$

In this case, the input into the top mixer will be I and a LO, which is $\cos(\omega_{IQ}t)\sin(\omega_{LO}t)$. Similarly, the input into the bottom mixer will be Q and 90°phase of LO, which is $\sin(\omega_{IQ}t)\cos(\omega_{LO}t)$.

The total output(in RF) will be the sum of the two waves

$$RF = \cos(\omega_{IQ}t)\sin(\omega_{LO}t) + \sin(\omega_{IQ}t)\cos(\omega_{LO}t)$$

and we know from simple trigonometry that $sin(a+b) = \cos(a)\sin(b) + \sin(a)\cos(b)$, so we get

$$\boxed{RF = \sin((\omega_{LO} + \omega_{IQ})t)} \tag{4.1}$$

Perfect! this is exactly what we wanted, the output is a wave with frequency that is the some of the input frequencies. Only one problem, it doesn't work :(.

4.5 Theory VS Reality :(

What is the problem? We've proved mathematically that it should work, so why wouldn't it? The problem is that we can't just assume the waves to come and go with the same phase, the waves travel through the wire at some speed so if we input into two different

[17] You can flip I and Q and get subtraction instead of addition

wires, two waves that are at the same phase, at the other side of the wire they might come at different phases because of differences in wire length, resistance, etc... So what can we do about it? You could try to make identical parts and make everything just perfect but even slight deviation will cause the system no to work properly, a better solution is to input more complex waves and have some parameters to play with so we can simply find the right parameters for the system and then it will work.

We can analyze the frequency space of the output of out IQ mixer and we can see 2 types of it not working correctly

- Leakage at the LO frequency
- Leakage at the harmonies($-\omega_{LO}, 2\omega_{LO}, 3\omega_{LO}, -2\omega_{LO}...$)

We can solve the first type of Leakage, at the LO frequency, by adding DC offsets to the input frequencies(We'll prove this mathematically later), and we can solve the second type of leakage by adding phase offsets to the waves(We'll prove this mathematically later). This is how the frequency spectrum looks without making any changes to the input waves

4.6 Solution for the real world

Let's consider now inputting into the IQ mixer the following waves instead of what we had earlier

$$\begin{aligned} I --> A_I \cos(\omega_{IQ} t + \phi_I) + \epsilon_I \\ Q --> A_Q \sin(\omega_{IQ} t + \phi_Q) + \epsilon_Q \\ LO --> \sin(\omega_{LO} t) \end{aligned}$$

Using the same calculation we did before, we get that

$$RF = I \cdot LO + Q \cdot LO(90°)$$

$$RF = (A_I \cos(\omega_{IQ} t + \phi_I) + \epsilon_I) \cdot \sin(\omega_{LO} t) + (A_Q \sin(\omega_{IQ} t + \phi_Q) + \epsilon_Q) \cdot \cos(\omega_{LO} t) \quad (4.2)$$

$$= \quad (4.3)$$

4.7 Finding Optimal Constants

As we've seen in the previous section, there are 4 variables we can "play" with to get the best variables for our system, as long as we don't change the system, these variables stay the same. What we want to do now is to actually find them. Our system is connected like so

We have the Quantum Machine[18] that generates the I and Q inputs that go into the mixer(and also to an oscilloscope for debugging). There's the frequency generator[19] that is connected to the mixer and generates 7GHz wave, and there's the frequency spectrum analyzer[20] that is connected to the computer.

Let's first attack the leakage at the LO frequency. For now we'll have an LO frequency of 7GHz that we want to change by 25MHz(The same variables work for all frequencies this is just as an example)

4.7.1 LO frequency Leakage

As proven in section 4.6, to minimize this kind of leakage all we need is to play with the DC offsets of the IQ inputs. To do so, we first need to define what we want to minimize, which in this case is simply the power of the frequency at 7GHz, we can measure that power with our spectrum analyzer, we'll call that our *cost function*. We have a 2-dimensional variable space, we need to find where in this space the cost function is at a minimum. To do so, we'll start by using a brute force method to find the general location of the minimum in the variable space, since brute force is very inefficient we can't really use it to find the exact location of the minimum so we start by only doing a low precision brute force and then use a different optimazition algorithm to find the exact location of the minimum. We'll use the `scipy.optimize.fmin` as the algorithm for precise minimum location.[21]

[18] this is the heart of the system, for now we'll use it to make the MHz waves with different phases, DC offsets and frequencies

[19] KeySight N5173B

[20] SignalHound USB SA-124B

[21] see more in appendix D

4.7.2 Harmonies Leakage

Now that we've minimized the LO frequency leakage, we want to minimize the harmony leakage, we do that by changing to phases of the IQ waves from the quantum machine, it's important that changing the phases doesn't change the LO leakage and luckily for us, as proven in section 4.6 that's whats happening. to change the phases we don't simply specify the phases, we use the correction matrix of the Quantum Machine. This time our cost function is the frequency at $\omega_{LO} - \omega_{IQ}$, we can do the same as we did in the LO leakage and use first a brute force optimization to find the general location of the minimum and the use the fmin algorithm to find the exact location of the minimum of the cost function(this time in the scale-angle variable space). You can see the result here

5 Future Work and Conclusions

...

A The Jaynes–Cummings Model

Our goal is to mathematically model the Hamiltonian of a system of a two-level atom interacting with a single quantized mode of an optical cavity's electromagnetic field.

First we'll divide the system into 3 parts, The atom(it can be other two-level quantum systems), the cavity(electromagnetic field with quantized modes) and the interaction between the atom and the cavity(an atom can emit a photon to the cavity and change it's electromagnetic field, or catch a photon from the cavity and go up an energy level).

Let's start with the cavity(we'll consider a one dimensional cavity for now).

A.1 The Homogeneous Electromagnetic Wave Equations

Maxwell's equations in free space are:

$$\boldsymbol{\nabla} \cdot \mathbf{E} = 0 \tag{A.1a}$$

$$\boldsymbol{\nabla} \cdot \mathbf{B} = 0 \tag{A.1b}$$

$$\boldsymbol{\nabla} \times \mathbf{E} = \frac{\partial \mathbf{B}}{\partial t} \tag{A.1c}$$

$$\boldsymbol{\nabla} \times \mathbf{B} = \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \tag{A.1d}$$

Taking the curl of A.1c and A.1d we get

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{E}) = \boldsymbol{\nabla} \times (-\frac{\partial \mathbf{B}}{\partial t}) = -\frac{\partial}{\partial t}(\boldsymbol{\nabla} \times \mathbf{B}) = -\mu_0 \epsilon_0 \frac{\partial^2 E}{\partial t^2} \tag{A.2a}$$

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{B}) = \boldsymbol{\nabla} \times (\mu_0 \epsilon_0 \frac{\partial E}{\partial t}) = \mu_0 \epsilon_0 \frac{\partial}{\partial t}(\boldsymbol{\nabla} \times E) = -\mu_0 \epsilon_0 \frac{\partial^2 \mathbf{B}}{\partial t^2} \tag{A.2b}$$

We can use the vector identity

$$\nabla \times (\nabla \times \mathbf{V}) = \nabla (\nabla \cdot \mathbf{V}) - \nabla^2 \mathbf{V} \tag{A.3}$$

And obtain from A.2a and A.2b

$$\nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E} = -\mu_0 \epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{A.4a}$$

$$\nabla(\nabla \cdot \mathbf{B}) - \nabla^2 \mathbf{B} = -\mu_0 \epsilon_0 \frac{\partial^2 \mathbf{B}}{\partial t^2} \tag{A.4b}$$

Now, we can use A.1a and A.1b To cancel the left most term and get

$$\nabla^2 \mathbf{E} = \mu_0 \epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{A.5a}$$

$$\nabla^2 \mathbf{B} = \mu_0 \epsilon_0 \frac{\partial^2 \mathbf{B}}{\partial t^2} \tag{A.5b}$$

Now because we know that $v_{ph} = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$ and that the phase velocity of electromagnetic waves in a vacuum is the speed of light, c_0, we get

$$\begin{aligned} \nabla^2 \mathbf{E} &= \frac{1}{c_0^2} \frac{\partial^2 \mathbf{E}}{\partial t^2} \\ \nabla^2 \mathbf{B} &= \frac{1}{c_0^2} \frac{\partial^2 \mathbf{B}}{\partial t^2} \end{aligned} \tag{A.6}$$

These equation are called *the homogeneous electromagnetic wave equations*. We'll pick a polarization arbitrarily to be in the x direction(that way we get only the component of the electric field and the y component of the magnetic field, E_x and B_y) so now we get,

$$\begin{aligned} \frac{\partial^2 E_x}{\partial x^2} &= \frac{1}{c_0^2} \frac{\partial^2 E_x}{\partial t^2} \\ \frac{\partial^2 B_y}{\partial y^2} &= \frac{1}{c_0^2} \frac{\partial^2 B_y}{\partial t^2} \end{aligned} \tag{A.7}$$

A.2 The Hamiltonians

We want to separate the total Hamiltonian into approachable parts, we can separate it like so,

$$H = H_{atom} + H_{cavity} + H_{interaction}$$

where the atom and cavity Hamiltonians are the Hamiltonian of the atom and cavity if they were the only part of the (closed)system and the interaction Hamiltonian is the Hamiltonian of the interaction between the atom and the cavity in the system.

A.2.1 cavity

We can easily solve A.7 using separation of variables,

$$E_x(z,t) = Z(z)T(t)$$

Yielding the solution,

$$\begin{aligned} E_x(z,t) &= \sqrt{\frac{2\omega_c^2}{V\epsilon_0}} q(t) \sin kz \\ B_y(z,t) &= \sqrt{\frac{2\mu_0}{V}} \dot{q}(t) \cos kz \end{aligned} \tag{A.8}$$

where V is the effective volume of the cavity, q is a time-dependent amplitude with units of length, and $k = m\pi/L$ for an integer $m > 0$

The Hamiltonian is given by

$$H = \frac{1}{2} \int \epsilon_0 \mathbf{E}^2 + \frac{\mathbf{B}^2}{\mu_0} dV \tag{A.9}$$

$$= \frac{1}{2} \int \epsilon_0 E_x^2(z,t) + \frac{B_y^2(z,t)}{\mu_0} dz \tag{A.10}$$

$$= \frac{1}{2}[\dot{q}^2(t) + \omega_c^2 q^2(t)] \tag{A.11}$$

This looks like the Hamiltonian of an harmonic oscillator.

Now, going from dynamical variables to operators(considering $\dot{q} \equiv p$) we get,

$$\hat{E}_x(z,t) = \sqrt{\frac{2\omega_c^2}{V\epsilon_0}} \hat{q}(t) \sin kz \tag{A.12}$$

$$\hat{B}_y(z,t) = \sqrt{\frac{2\mu_0}{V}} \hat{p}(t) \cos kz \tag{A.13}$$

$$\hat{H} = \frac{1}{2}[\hat{p}^2(t) + \omega_c^2 \hat{q}^2(t)] \tag{A.14}$$

Let's introduce creation and annihilation operators,

$$\hat{a}(t) = \frac{1}{\sqrt{2\hbar\omega_c}}[\omega_c \hat{q}(t) + i\hat{p}(t)]$$

$$\hat{a}^\dagger(t) = \frac{1}{\sqrt{2\hbar\omega_c}}[\omega_c\hat{q}(t) - i\hat{p}(t)]$$

We can write the electric and magnetic field as,

$$\hat{E}_x(z,t) = E_0[\hat{a}(t) + \hat{a}^\dagger(t)]\sin kz \tag{A.15}$$

$$\hat{B}_y(z,t) = \frac{E_0}{c}[\hat{a}(t) - \hat{a}^\dagger(t)]\cos kz \tag{A.16}$$

And we can write the Hamiltonian as,

$$\boxed{\hat{H}_{cavity} = \hbar\omega_c[\hat{a}\hat{a}^\dagger + \frac{1}{2}] \approx \hbar\omega_c\hat{a}\hat{a}^\dagger} \tag{A.17}$$

We can ignore the zero-point energy $\frac{\hbar\omega_c}{2}$ if we define it as the zero energy point.

A.2.2 atom

Now that we have the cavity's Hamiltonian, we can go on to calculate the atom(qubit) Hamiltonian.

Remember that the qubit is a 2-level system, meaning we can define it has a superposition of the ground, $|g\rangle$, and excited, $|e\rangle$, states. The energy of the atom is the sum of the energy of each state times it's energy($\sum E_s P(|s\rangle)$). The probability to be in a state $|s\rangle$ is given by $|s\rangle\langle s|$ so we can write,

$$\hat{H}_{atom} = E_g|g\rangle\langle g| + E_e|e\rangle\langle e| \tag{A.18}$$

Using the vector representation of these states we'll write,

$$\begin{aligned}\hat{H}_{atom} &= E_e \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + E_g \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} E_e & 0 \\ 0 & E_g \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} E_g + E_e & 0 \\ 0 & E_g + E_e \end{bmatrix} + \frac{1}{2} \begin{bmatrix} E_e - E_g & 0 \\ 0 & -(E_e - E_g) \end{bmatrix} \\ &= \frac{1}{2}(E_g + E_e) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{1}{2}(E_e - E_g) \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ &= \frac{1}{2}(E_g + E_e)\mathbb{I} + \frac{1}{2}(E_e - E_g)\hat{\sigma}_z\end{aligned}$$

Again, we can define the zero point energy so that the first term becomes 0. We know the difference between the excited state energy and the ground state energy because it's approximately an harmonic oscillator so $E_e - E_g = \hbar\omega_a$ where ω_a is the atom frequency. Now we can write,

$$\boxed{\hat{H}_{atom} = \frac{1}{2}\hbar\omega_a\hat{\sigma}_z} \tag{A.19}$$

A.2.3 interaction

Now for the last part of the Hamiltonian, we want to know the interaction Hamiltonian between the atom and the cavity. The interaction Hamiltonian is now

$$\hat{H}_{interaction} = -\hat{\mathbf{d}} \cdot \hat{\mathbf{E}} = -\hat{d}E_0 \sin kz(\hat{a} + \hat{a}^\dagger)$$

Where we introduced $\hat{d} = \hat{\mathbf{d}} \cdot \hat{x}$ (remember that we defined the axis so that x is the direction of polarization of the electromagnetic field). We'll also introduce the atomic transition operators

$$\hat{\sigma}_+ = |e\rangle \langle g| , \qquad \hat{\sigma}_- = |g\rangle \langle e| = \hat{\sigma}_+^\dagger$$

and the inversion operator

$$\hat{\sigma}_3 = |e\rangle \langle e| - |g\rangle \langle g|$$

only the off-diagonal elements of the dipole operator are nonzero so we may write

$$\hat{d} = d\left|e\right\rangle\left\langle g\right| + d^*\left|g\right\rangle\left\langle e\right| = d\hat{\sigma}_- + d^*\hat{\sigma}_+ = d(\hat{\sigma}_+ + \hat{\sigma}_-)$$

thus the interaction Hamiltonian is

$$\hat{H}_{interaction} = \hbar\lambda(\hat{\sigma}_+ + \hat{\sigma}_-)(\hat{a} + \hat{a}^\dagger) = \hbar\lambda(\hat{\sigma}_+\hat{a} + \hat{\sigma}_+\hat{a}^\dagger + \hat{\sigma}_-\hat{a} + \hat{\sigma}_-\hat{a}^\dagger) \tag{A.20}$$

We shown the the operators $\hat{a}$ and $\hat{a}^\dagger$ evolve as

$$\hat{a}(t) = \hat{a}(0)e^{-i\omega t}, \qquad \hat{a}^\dagger(t) = \hat{a}^\dagger(0)e^{i\omega t} \tag{A.21}$$

And similarly we can show for the free-atomic case that

$$\hat{\sigma}_\pm(t) = \hat{\sigma}_\pm(0)e^{\pm i\omega t} \tag{A.22}$$

We can write while approximating $\omega_0 \approx \omega$

$$\begin{aligned}
\hat{\sigma}_+\hat{a} &\sim e^{i(\omega_0-\omega)t} \\
\hat{\sigma}_-\hat{a}^\dagger &\sim e^{-i(\omega_0-\omega)t} \\
\hat{\sigma}_+\hat{a}^\dagger &\sim e^{i(\omega_0+\omega)t} \\
\hat{\sigma}_-\hat{a} &\sim e^{-i(\omega_0+\omega)t}
\end{aligned} \tag{A.23}$$

We can see that the last two term vary much more rapidly than the first two. Furthermore, the last two terms do not conserve energy(They correlate to [photon addition + atom excitation] and [photon reduction + atom grounded]), we're going to drop the last two terms and finally get

$$\boxed{H_{interaction} = \hbar\lambda(\hat{\sigma}_+\hat{a} + \hat{\sigma}_-\hat{a}^\dagger)} \tag{A.24}$$

Finally, we can write the full JC Hamiltonian

$$\boxed{\hat{H} = \frac{1}{2}\hbar\omega_0\hat{\sigma}_3 + \hbar\omega\hat{a}^\dagger\hat{a} + \hbar\lambda(\hat{\sigma}_+\hat{a} + \hat{\sigma}_-\hat{a}^\dagger)} \tag{A.25}$$

A.3 Effective Hamiltonian in the Dispersive Limit

...

B Couplers and Mixers(?)

Our goal here is to understand some ways you could physically implement some of the components in chapter 4, there are a lot of ways to implement each component but we're going to look at only a few. First we'll look into how would you implement a mixer, and then we'll see how would you implement a coupler(to make a phase difference).

B.1 Single-Ended Diode Mixer

B.2 Single-Ended FET Mixer

B.3 Ring Modulator

B.4 Branch-Line Quadrature(90°) Coupler

B.5 Rat-Race Coupler

C Codes

C.1 IQ Mixer Optimization

C.2 GRAPE

C.3 Transmon-Cavity GRAPE

D Optimization Algorithms(?)

D.1 fmin

D.2 L-BFGS-B

References

[1] J. R. Johansson, P. D. Nation, and F. Nori: "QuTiP 2: A Python framework for the dynamics of open quantum systems.", Comp. Phys. Comm. 184, 1234 (2013) [DOI: 10.1016/j.cpc.2012.11.019].

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[3] Knuth: Computers and Typesetting,
`http://www-cs-faculty.stanford.edu/~uno/abcde.html`