



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Eduardo Willwock Lussi

**Análise Comparativa de Técnicas de Mitigação de Erros em Processos de
Computação Quântica**

Florianópolis
2022

Eduardo Willwock Lussi

**Análise Comparativa de Técnicas de Mitigação de Erros em Processos de
Computação Quântica**

Trabalho de Conclusão de Curso do Curso de Graduação em Ciência da Computação do Campus Florianópolis da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Ciência da Computação.

Orientadora: Profa. Jerusa Marchi, Dra.

Coorientador: Prof. Eduardo Inacio Duzzioni, Dr.

Florianópolis

2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Lussi, Eduardo Willwock

Análise Comparativa de Técnicas de Mitigação de Erros em Processos de Computação Quântica / Eduardo Willwock Lussi ; orientadora, Jerusa Marchi, coorientador, Eduardo Inácio Duzzioni, 2022.

163 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2022.

Inclui referências.

1. Ciências da Computação. 2. Computação Quântica. 3. Controle Quântico. 4. IBM Quantum. 5. Decoerência. I. Marchi, Jerusa. II. Duzzioni, Eduardo Inácio. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Eduardo Willwock Lussi

**Análise Comparativa de Técnicas de Mitigação de Erros em Processos de
Computação Quântica**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “bacharel em Ciência da Computação” e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação.

Florianópolis, 07 de dezembro de 2022.

Prof. Jean Everson Martina, Dr.
Coordenador do Curso

Banca Examinadora:

Profa. Jerusa Marchi, Dra.
Orientadora

Prof. Eduardo Inácio Duzzioni, Dr.
Coorientador
Universidade Federal de Santa Catarina

Evandro de Chagas Ribeiro da Rosa, Msc.
Avaliador
Quantuloop

Este trabalho é dedicado aos meus queridos pais, Edomar
Luiz Lussi e Sandra Beatriz Willwock Lussi.

AGRADECIMENTOS

Tamanho é o esforço necessário para o progresso da ciência, as motivações pessoais para tal são profundamente sustentadas por pessoas que não permitirão que você se perca em sua própria jornada e ao mesmo tempo apoiarão as suas mais intrínsecas ambições. Reconhecer a boa vontade daqueles que de alguma forma se fizeram presentes em sua vida para atribuí-la sentido se faz necessário a medida em que eles têm parte do mérito de seu trabalho já que sem eles você ou não estaria onde está ou não haveria motivos para estar lá.

Meu agradecimento especial orienta-se, obviamente, aos verdadeiros responsáveis por quem sou. Meus pais nunca mediram esforços para me colocar onde eu precisava estar, construíram meu caráter sob seus melhores valores, meus agradecimentos estão para muito além deste trabalho. Levo comigo um profundo reconhecimento de seus esforços, eles são o principal motivo para eu estar onde estou e para eu seguir até onde quero chegar.

Também gostaria de agradecer a todos os que me orgulho de chamar de professor ou professora. Tenho dívida eterna com todos, seus nomes deveriam estar aqui citados, mas neste contexto agradeço principalmente ao apoio e orientação da professora Dra. Jerusa Marchi, do professor Dr. Eduardo Duzzioni e demais membros do Grupo de Computação Quântica da UFSC.

*“We must not wait for things to come,
believing that they are decided by irrevocable destiny.
If we want it, we must do something about it.”
(Erwin Schrödinger, 1943)*

RESUMO

A computação clássica vem chegando cada vez mais próximo dos limites da Lei de Moore, existem problemas complexos que são completamente inviáveis de serem solucionados por computadores clássicos. O paradigma de computação quântica tem a capacidade computacional necessária para resolver alguns problemas de forma muito mais eficiente, entretanto, sistemas quânticos ainda são extremamente instáveis e são muito difíceis de controlar adequadamente. Por esse motivo, existe uma grande demanda pelo desenvolvimento de técnicas para caracterizar e mitigar erros que atrapalham o uso da computação quântica. Este trabalho tem como objetivo implementar técnicas para mitigar os erros de decoerência de sistemas quânticos em aplicações práticas. Para isso, a área de controle quântico em computadores quânticos de supercondutores será investigada com o intuito de utilizar técnicas que calibram o hardware quântico e controlam a computação por meio de portas lógicas quânticas descritas na forma de pulsos de micro-ondas. As aplicações práticas consideram a implementação de um autômato finito quântico MO1QFA para resolver o problema do módulo, onde são realizados diversos experimentos com diferentes implementações de portas lógicas em que a robustez a diferentes erros é avaliada. A IBM Quantum Experience e o kit de desenvolvimento de software Qiskit são utilizados como ambientes de modelagem de sistemas quânticos e aplicação em máquinas reais. Em complemento ao Qiskit, as ferramentas oferecidas pelo Q-CTRL Boulder Opal são utilizadas para automatizar os processos de otimização do hardware quântico. Além da investigação das otimizações, espera-se que este trabalho sirva de referência para estudantes de Ciências Exatas que estejam interessados em estudar controle quântico em computadores quânticos de supercondutores, fornecendo o conteúdo teórico e prático necessário para que seja possível realizar otimizações em seus experimentos.

Palavras-chave: Computação Quântica. Controle Quântico. Decoerência. Hardware Quântico. Supercondutores. Q-CTRL Boulder Opal. IBM Quantum Experience.

ABSTRACT

Classical methods of computing are getting increasingly closer to the limits of Moore's Law, there are complex problems completely impracticable to be solved by classical computers. Quantum computing has the necessary computational resources to solve many problems in a much more efficient way, however, quantum systems are still very noisy and hard to control. Because of that, quantum computing field has a big demand for the development of quantum error characterization and mitigation. This paper aims to implement quantum error mitigation techniques for quantum systems in practical approaches. Thereunto, quantum control field for superconducting quantum computers is investigated in order to use quantum hardware calibration techniques that control the computation using the microwave pulses translation of quantum gates. The practical experiments consider a quantum finite automata MO1QFA implementation for solving the modulus problem, where many experiments are performed with different quantum gates implementations aiming to evaluate their robustness against different noise sources. IBM Quantum Experience interface and the software development kit Qiskit are used for modeling and implementing the quantum systems in real environments. In addition to IBMQ tools, Q-CTRL Boulder Opal techniques are used to automate some optimization processes of quantum hardware. Besides the investigation of the optimization techniques, it is expected that this paper will serve as a reference for exact sciences students that are interested in studying quantum control for superconducting quantum computers, offering a theoretical-practical foundation for quantum error mitigation and hardware optimization.

Keywords: Quantum Computing. Quantum Control. Decoherence. Quantum Hardware. Superconducting. Q-CTRL Boulder Opal. IBM Quantum Experience.

LISTA DE FIGURAS

Figura 1.1 – Crescimento da performance dos processadores de 1978 a 2018.	16
Figura 2.1 – Modelo Genérico de Computação Quântica de Circuitos.	26
Figura 2.2 – Esfera de Bloch.	26
Figura 2.3 – Notação de circuitos.	27
Figura 2.4 – Porta de Pauli X ou NOT quântica.	28
Figura 2.5 – Porta de Pauli Y	29
Figura 2.6 – Porta de Pauli Z	29
Figura 2.7 – Porta de Hadamard.	30
Figura 2.8 – Portas R_x , R_y e R_z	30
Figura 2.9 – Porta U.	31
Figura 2.10 – Porta $CNOT$	32
Figura 2.11 – Circuito para obtenção do estado de Bell $ \beta_{00}\rangle$	32
Figura 2.12 – Porta Z controlada.	33
Figura 2.13 – Porta $SWAP$	33
Figura 2.14 – Porta $CCNOT$	34
Figura 2.15 – Circuito com aplicação da porta de Hadamard e medida.	34
Figura 3.1 – Circuitos supercondutores e o regime transmon.	38
Figura 3.2 – Tradução da porta lógica X em pulsos para o Nairobi.	40
Figura 3.3 – Resultado do experimento de Rabi para um pulso qualquer.	41
Figura 3.4 – Frequências de Rabi para um experimento.	42
Figura 3.5 – Exemplo de pulso gerado pelo Boulder Opal.	43
Figura 3.6 – Exemplo de obtenção do parâmetro S_{rel}	44
Figura 3.7 – Exemplo de obtenção do parâmetro S_{amp}	44
Figura 3.8 – Exemplo de resultado de otimização da frequência.	45
Figura 3.9 – Distinção entre os estados $ 0\rangle$ e $ 1\rangle$	46
Figura 3.10 – Experimento para obtenção de T_1	47
Figura 3.11 – Experimento para obtenção de T_2	47
Figura 5.1 – Representação bidimensional do qubit para o problema MOD^p	62
Figura 5.2 – Diagrama do autômato MO1QFA para o problema MOD^p	62
Figura 5.3 – Probabilidades de aceitação obtidas e esperadas para palavras congru- entes a 0 módulo 11 com nível de otimização 1.	64
Figura 5.4 – Probabilidades de aceitação obtidas e esperadas para palavras congru- entes a 3 módulo 11 com nível de otimização 1.	65
Figura 5.5 – Circuito para palavras de tamanho congruente a 3 módulo 11 transpi- lado para o hardware.	66
Figura 5.6 – Probabilidades de aceitação obtidas e esperadas para palavras congru- entes a 0 módulo 11 com nível de otimização 0.	68

Figura 5.7 – Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 3 módulo 11 com nível de otimização 0.	69
Figura 5.8 – Circuito que implementa a porta lógica $R_x(4\pi/11)$ com nível de otimização 0.	69
Figura 5.9 – Frequências de Rabi para os computadores Nairobi, Oslo e Belém.	70
Figura 5.10–Pulso square R_x	71
Figura 5.11–Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 0 módulo 11 para o computador Nairobi.	72
Figura 5.12–Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 3 módulo 11 para o computador Nairobi.	72
Figura 5.13–Parâmetro S_{rel} para diferentes repetições do pulso.	74
Figura 5.14–Parâmetro S_{amp} para diferentes repetições do pulso.	74
Figura 5.15–Resultado da calibração fine-tuned com parâmetros mal escolhidos.	75
Figura 5.16–Probabilidades de aceitação do pulso square com calibrações básica, fine-tuned e autônoma para palavras de tamanho congruente a 0 módulo 11.	76
Figura 5.17–Probabilidades de aceitação do pulso square com calibrações básica, fine-tuned e autônoma para palavras de tamanho congruente a 3 módulo 11.	77
Figura 5.18–Infidelidade do pulso square com calibrações básica, fine-tuned e autônoma.	77
Figura 5.19–Resultado da otimização da frequência.	78
Figura 5.20–Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 0 módulo 11.	79
Figura 5.21–Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 3 módulo 11.	80
Figura 5.22–Infidelidade do pulso square de 14, 2 ns aplicado na frequência padrão, com $-10 MHz$ e $+10 MHz$	81
Figura 5.23–Infidelidade do pulso square de 227, 33 ns aplicado na frequência padrão, com $-10 MHz$ e $+10 MHz$	82
Figura 5.24–Pulso de 227.33 ns gerado pelo Boulder Opal resistente à <i>dephasing</i>	83
Figura 5.25–Simulações da evolução temporal dos pulsos square e otimizado de 227, 33 ns para implementação da porta lógica R_x	83
Figura 5.26–Infidelidade do pulso resistente à <i>dephasing</i> de 227, 33 ns aplicado na frequência padrão, com $-10 MHz$ e $+10 MHz$	84
Figura A.1–Oscilação de Rabi para um Pulso de Amplitude 0.2	97
Figura A.2–Resultado do Experimento de Rabi	99

Figura B.1 – Decomposição IQ dos Pulsos Q-CTRL e Square	109
Figura B.2 – Evolução Temporal Simulada dos Componentes da Esfera de Bloch para os Pulsos Q-CTRL e Square	110
Figura B.3 – Evolução Temporal dos Componentes da Esfera de Bloch para os Pulsos Q-CTRL e Square	116
Figura C.1 – Porta X Resistente a Dephasing	119
Figura C.2 – Experimento para Obtenção de S_{rel}	125
Figura C.3 – Experimento para Obtenção de S_{amp}	130
Figura C.4 – Menores Custos Durante o Processo de Calibração Autônoma	139
Figura C.5 – Resultado das Diferentes Calibrações do Pulso X Resistente à Dephasing	143

LISTA DE TABELAS

Tabela 1 – Número de Resultados por Combinação de Palavra-chave.	51
Tabela 2 – Comparação de Trabalhos Relacionados.	52
Tabela 3 – Valores de α e β da equação 6 para os computadores quânticos Nairobi, Oslo e Belém.	67
Tabela 4 – Resumo dos erros absolutos entre os resultados dos experimentos e a função aproximada dada pela equação 6.	67

SUMÁRIO

1	INTRODUÇÃO	15
1.1	OBJETIVOS	18
1.2	ORGANIZAÇÃO DO TRABALHO	18
2	INTRODUÇÃO À COMPUTAÇÃO QUÂNTICA	19
2.1	ÁLGEBRA LINEAR PARA COMPUTAÇÃO QUÂNTICA	19
2.2	INTRODUÇÃO À MECÂNICA QUÂNTICA	23
2.3	COMPUTAÇÃO QUÂNTICA DE CIRCUITOS	25
2.4	MEDIÇÃO	34
2.5	TEOREMA DA NÃO-CLONAGEM	35
3	COMPUTADORES QUÂNTICOS SUPERCONDUTORES	36
3.1	SUPERCONDUTIVIDADE	36
3.2	TRANSMON	37
3.3	PROGRAMAÇÃO DE HARDWARE QUÂNTICO COM QISKIT PULSE	39
3.4	EXPERIMENTO DE RABI	40
3.5	DESIGN E CALIBRAÇÃO DE PORTAS LÓGICAS COM Q-CTRL BOULDER OPAL	42
3.6	OTIMIZAÇÃO DA FREQUÊNCIA	45
3.7	DISTINÇÃO ENTRE OS ESTADOS 0 E 1	45
3.8	O TEMPO T_1	46
3.9	O TEMPO T_2	47
3.10	PRINCIPAIS FONTES DE ERROS	48
4	ESTADO DA ARTE	50
4.1	OTIMIZAÇÃO DE PORTAS LÓGICAS QUÂNTICAS ROBUSTAS À RUÍDOS UTILIZANDO UMA INTERFACE CLOUD	54
4.2	UMA ABORDAGEM DE APRENDIZAGEM PROFUNDA POR REFORÇO PARA A CONSTRUÇÃO DE PORTAS LÓGICAS ROBUSTAS À RUÍDOS EM UM COMPUTADOR QUÂNTICO SUPERCONDUTOR	55
4.3	UMA DEMONSTRAÇÃO DE UM CONJUNTO UNIVERSAL DE PORTAS LÓGICAS QUÂNTICAS TOLERANTE À FALHAS	56
4.4	REMOVENDO ERROS DE VAZAMENTO DE ELÉTRONS DE SUPERCONDUTORES EM QUANTUM ERROR CORRECTION	56
4.5	OTIMIZANDO CÓDIGOS DE CORREÇÃO DE ERROS COM APRENDIZADO POR REFORÇO	57
5	EXPERIMENTAÇÃO	59
5.1	MEASURE-ONCE 1-WAY QUANTUM FINITE AUTOMATA	60
5.2	EXPERIMENTOS COM NÍVEL DE OTIMIZAÇÃO 1	64
5.3	EXPERIMENTOS COM NÍVEL DE OTIMIZAÇÃO 0	67

5.4	EXPERIMENTOS COM PORTAS LÓGICAS SQUARE ULTRARRÁPIDAS	70
5.5	CALIBRAÇÕES BÁSICAS, FINE-TUNED E AUTOMATED	73
5.6	OTIMIZAÇÃO DA FREQUÊNCIA	78
5.7	PULSOS RESISTENTES A ERROS DE <i>DEPHASING</i>	80
6	CONSIDERAÇÕES FINAIS	85
	Referências	87
	APÊNDICE A – EXPERIMENTO DE RABI	92
A.1	AUTENTICAÇÃO	92
A.2	EXECUTAR EXPERIMENTO	92
A.3	EXTRAIR RESULTADOS	95
	APÊNDICE B – CONSTRUÇÃO DE PORTAS LÓGICAS RESIS-	
	TENTES À RUÍDOS COM Q-CTRL BOUDER	
	OPAL	101
B.1	AUTENTICAÇÃO	101
B.2	MODELAGEM DE PULSOS COM BOULDER OPAL	102
B.3	VALIDAÇÃO DOS PULSOS	111
	APÊNDICE C – CALIBRAÇÃO DE PORTAS LÓGICAS	117
C.1	AUTENTICAÇÃO	117
C.2	IMPORTAÇÃO DO EXPERIMENTO DE RABI	118
C.3	IMPORTAÇÃO DO PULSO	119
C.4	CALIBRAÇÃO DOS VALORES DE AMPLITUDE DO PULSO	121
C.5	VALIDAÇÃO	140
	APÊNDICE D – ARTIGO SOBRE O TRABALHO	144

1 INTRODUÇÃO

Quando a Máquina de Turing (MT) foi proposta por Turing (1936), em complemento ao trabalho de Church (1936), o que posteriormente se denominou tese de Church-Turing, nada se falou a respeito de complexidade computacional dos algoritmos que resolvem os problemas que ela é capaz de solucionar. A tese de Church-Turing diz que tudo o que é computável, é computável por uma MT, apesar disso, acredita-se que alguns problemas computáveis são complexos demais para se implementar algoritmos eficientes em termos de tempo e espaço, que é o caso dos problemas classificados como NP-Completo.

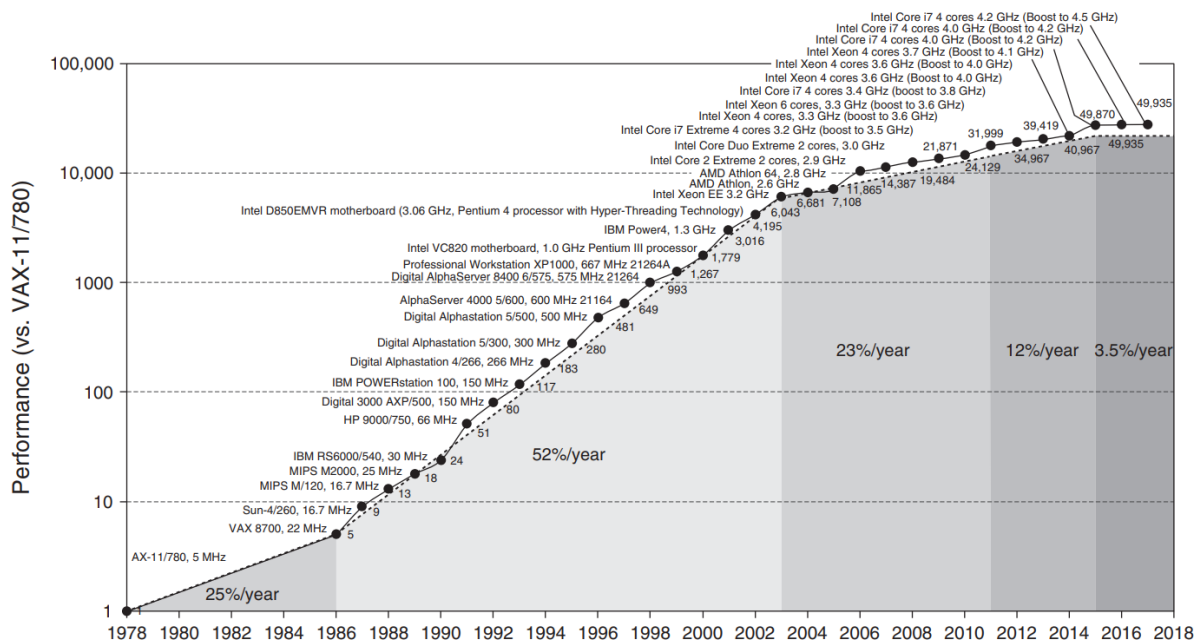
Uma tentativa de melhorar a eficiência da computação veio com o conceito de Máquina de Turing Probabilística (MTP), que é um tipo de MT não-determinística onde cada passo da computação é chamado passo de arremesso-de-moeda e existe uma probabilidade associada a cada ramo da computação (SIPSER, 2012). Por possuírem essa característica, vários problemas possuem solução conhecida mais eficiente para MTP do que para MT convencionais, como a linguagem relacionada à palíndromos investigada por Jill (1977). O problema é que, ainda que a implementação de MT convencionais seja perfeitamente factível por meio de mecanismos clássicos, as MTP não são muito tangíveis, para implementá-las, podem ser necessários recursos exponenciais com relação à MT clássica (KAYE; LAFLAMME; MOSCA, 2007).

Mesmo que alguns problemas demandem recursos exponenciais, os computadores atuais são suficientemente rápidos para solucionar grande parte deles, e desde que seja possível construir computadores cada vez mais rápidos, a complexidade dos algoritmos não deveria ser algo a ser temido. A lei de Moore (MOORE, 1965) inicialmente previa que o número de transistores em um chip dobraria anualmente, alguns anos depois, em Moore (1975), Moore ajustou sua previsão afirmando que esse número dobraria a cada dois anos. O que aconteceu foi que a evolução do poder computacional dos computadores clássicos diminuiu com o passar dos anos, como é possível visualizar no famoso gráfico que aparece logo no início do livro de arquitetura de computadores de Hennessy e Patterson (2019), apresentado na figura 1.1.

Entre 1986 e 2003 houve um crescimento considerável de 52% ao ano no desempenho dos processadores, dobrando aproximadamente a cada dois anos conforme previa a lei de Moore. Isso foi possível graças a escala de Dennard (DENNARD *et al.*, 1974), que diz que a densidade de potência dos chips deve se manter constante à medida que os transistores diminuem de tamanho, permitindo que a frequência dos processadores pudesse ser aumentada cada vez mais sem grandes problemas. Entretanto, o tamanho cada vez mais reduzido dos transistores fez com que essa estratégia não fosse mais tão simples assim, já que os processadores começaram a trabalhar em temperaturas muito altas. Por esse motivo, a partir de 2003 até 2011, a evolução da performance dos processadores

diminuiu para uma taxa de 23% ao ano e os fabricantes começam a construir chips multi-core, e a paralelização dos algoritmos se fez bastante eficiente. Todavia, de 2011 a 2015, o crescimento diminuiu para 12% ao ano, em parte devido à lei de Amdahl (AMDAHL, 1967), que prevê a existência de um limite máximo de speedup em um algoritmo paralelizado. De 2015 a 2018, o crescimento diminuiu ainda mais, deixando claro que não está sendo mais possível construir computadores clássicos mais eficientes.

Figura 1.1 – Crescimento da performance dos processadores de 1978 a 2018.



Fonte: (HENNESSY; PATTERSON, 2019)

Na virada do século XX houve uma grande reviravolta nos estudos da mecânica clássica, quando uma série de experimentos em nível de átomos não conseguiam ser explicados com as leis da física clássica (FACULTY, 2022). Essa crise só começou a ser superada por volta da década de 1920, após Albert Einstein, em 1905, apresentar o trabalho que lhe conferiu o prêmio Nobel de física em 1921, com a teoria de que cada fóton contém um número quantizado de energia, que depende da frequência da luz.

De fato, quando a MT foi conceitualizada, a física quântica ainda não tinha se desenvolvido perfeitamente. Dada sua natureza clássica, vários físicos, incluindo Richard Feynman, observaram que computadores clássicos não eram capazes de simular sistemas quânticos com eficiência (KAYE; LAFLAMME; MOSCA, 2007). A tese de Church-Turing prevê que tudo o que é computável é computável por uma MT, entretanto, sabe-se que isso não é um teorema, já que não há uma definição de “computável” (SHOR, 1998). Peter Shor também observa que essa tese pode ser vista como uma afirmação sobre as leis da física, nesse sentido, se as leis da física são computáveis por uma MT, então a tese

é verdadeira. Sendo assim, considerando que algumas simulações quânticas possam ser impossíveis de serem feitas na prática por MT clássicas, a tese de Church-Turing parece ser agredida de certa forma, já que, apesar de serem computáveis, esses problemas podem não ser realmente computáveis por uma MT clássica devido a sua complexidade.

Feynman foi o primeiro a sugerir a existência de um computador cuja evolução fosse baseada nas leis da mecânica quântica (FEYNMAN, 1982). Contudo, o potencial da computação quântica apenas foi reconhecido quando Peter Shor (SHOR, 1998) apresentou um algoritmo quântico capaz de realizar a fatoração em números primos em tempo polinomial, colocando em xeque todo o sistema de criptografia de chave pública RSA. A partir disso, vários outros algoritmos foram desenvolvidos e a computação quântica surge como uma nova área de pesquisa multidisciplinar com grande potencial.

Apesar das grandes expectativas a respeito da computação quântica, ainda é necessário muito desenvolvimento com relação ao hardware quântico. Sistemas quânticos, de maneira geral, são bastante instáveis e sujeitos a interferências, os principais computadores quânticos supercondutores atuais funcionam em locais completamente isolados com temperaturas próximas do zero absoluto. Essas condições são necessárias devido a dificuldade em se manter esses sistemas em estados quânticos, qualquer interação pode causar problemas de decoerência que fazem com que o estado quântico se perca e se torne um estado clássico (ZUREK, 2002).

Entretanto, já existem soluções que auxiliam os desenvolvedores de tecnologias quânticas a aperfeiçoarem o controle sobre o hardware para melhorar os resultados de seus experimentos. A Q-CTRL (Q-CTRL, 2022a) oferece uma série de ferramentas nesse sentido, elas automatizam grande parte do processo de otimização de portas lógicas e calibração do hardware quântico.

Este trabalho apresentará técnicas para a implementação de portas lógicas quânticas otimizadas que mitigam erros de decoerência dos computadores quânticos. Para isso, será feita uma descrição da área de controle quântico em computadores quânticos supercondutores, mostrando como portas lógicas quânticas são implementadas em hardware, como programar sistemas quânticos a nível de pulsos de micro-ondas e como são feitos os processos de calibração e controle de hardware quântico para a correção de erros. O Q-CTRL Boulder Opal será utilizado de forma a automatizar os processos e facilitar a manipulação de pulsos de micro-ondas que controlam os estados quânticos. Serão realizados experimentos em computadores quânticos reais fornecidos pela IBM Quantum Experience (IBM, 2022b) utilizando o kit de desenvolvimento de software (SDK) Qiskit, também da IBM.

Diante do regime NISQ (Noisy Intermediate-Scale Quantum) de computação quântica que é afetado por erros, serão investigadas diferentes formas de se implementar portas lógicas quânticas e suas suscetibilidades e robustez a diferentes processos de erros. A aplicação prática de validação é um autômato finito quântico MO1QFA que resolve o problema

do módulo, nesse sentido, além da investigação mais aprofundada dos pulsos que implementam as portas lógicas, são fornecidas diferentes alternativas para a implementação desse autômato que resultaram em consideráveis melhorias com relação às implementações existentes.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem como objetivo a investigação de técnicas de mitigação de erros em processos de computação quântica. Nesse sentido, o objetivo também inclui a apresentação de diferentes técnicas de geração de pulsos de micro-ondas e calibração do hardware quântico utilizando tanto as técnicas padrão oferecidas pelo Qiskit quanto as ferramentas do Q-CTRL Boulder Opal. Não obstante, o principal objetivo é construir um trabalho de referência para estudantes de graduação em Ciências Exatas interessados em otimizar o hardware quântico e implementações, oferecendo um referencial teórico e prático para uma melhor escolha dentre as técnicas com relação às suas necessidades e condições.

1.1.2 Objetivos Específicos

Os objetivos específicos do trabalho são os que seguem:

- Avaliar as condições atuais dos computadores quânticos com relação a erros de decoerência.
- Construir um referencial teórico e prático na área de controle quântico.
- Implementar técnicas de mitigação de erros em aplicações práticas utilizando ambientes reais da IBM Quantum Experience.
- Realizar uma análise comparativa das técnicas de mitigação de erros em processos de computação quântica.

1.2 ORGANIZAÇÃO DO TRABALHO

Sendo assim, a seção 2 introduz os principais conceitos de computação quântica, incluindo conceitos de álgebra linear e mecânica quântica. A seção 3 mostra como computadores quânticos supercondutores são implementados e controlados. A seção 4 apresenta o estado da arte do tema deste trabalho. A seção 5 apresenta implementações e resultados das diversas alternativas de mitigação de erros utilizando controle quântico e otimizações circuitais. Por fim, a seção 6 conclui o trabalho.

2 INTRODUÇÃO À COMPUTAÇÃO QUÂNTICA

Sabe-se que a informação da computação clássica é codificada em bits que podem assumir os estados 0 e 1. A computação quântica utiliza um recurso equivalente chamado de qubit, que pode estar nos estados 0, 1 ou em uma infinidade de combinações lineares, inclusive complexas, de 0 e 1. Por esse motivo, a computação quântica tem ganho exponencial na representação da informação, n qubits podem assumir a superposição de 2^n resultados possíveis ao mesmo tempo (GILL *et al.*, 2021). Além da superposição, a computação quântica também utiliza o conceito de entrelaçamento, que permite interligar estados de diferentes qubits. Esta seção apresenta os principais conceitos necessários para o entendimento geral da computação quântica, incluindo noções de álgebra linear, postulados da mecânica quântica e a computação quântica de circuitos. Serão utilizados como referência para a descrição dos conceitos desta seção os livros específicos de computação quântica de Kaye, Laflamme e Mosca (2007) e Nielsen e Chuang (2010) e os trabalhos com uma abordagem mais direta de Pollachini (2018) e Silva (2018).

2.1 ÁLGEBRA LINEAR PARA COMPUTAÇÃO QUÂNTICA

A matemática envolvida na computação quântica deriva da física quântica, que utiliza uma notação especial e modela qubits utilizando espaços vetoriais, transformações lineares e produto tensorial. Os principais conceitos de álgebra linear para computação quântica serão explicados nesta seção, para isso, será utilizada como referência adicional de álgebra linear, o clássico livro de Steinbruch e Winterle (1995).

2.1.1 Notação de Dirac

A notação de Dirac, proposta por Dirac (1939), é uma notação alternativa para a representação de vetores utilizada com mais frequência na mecânica quântica. Com essa notação, um vetor ket $|\psi\rangle$ é definido da seguinte forma:

$$|00\dots 00\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad |00\dots 01\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \dots, \quad |11\dots 10\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad |11\dots 11\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

Em contrapartida, é possível definir um vetor bra $\langle\psi|$ a partir de $|\psi\rangle$ que corresponde a um vetor linha cujos elementos representam o transposto conjugado dos elementos de $|\psi\rangle$:

$$|\psi\rangle^\dagger = \langle\psi| = [\psi_0^* \quad \psi_1^* \quad \dots \quad \psi_{n-1}^*].$$

2.1.2 Espaço Vetorial, Base e Produto Interno

A definição de espaço vetorial é importante para a modelagem de qubits, o estado de um qubit é um vetor que pertence a um tipo especial de espaço vetorial chamado de espaço de Hilbert, que, no contexto da computação quântica, consiste de um espaço vetorial complexo com um produto interno. Sendo assim, um espaço vetorial complexo \mathbb{C}^n pode ser definido como um conjunto de n -uplas $(u_0, u_1, \dots, u_{n-1})$ de números complexos com operações de soma e produto por escalar definidos entrada a entrada:

$$\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} + \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} u_0 + v_0 \\ u_1 + v_1 \\ \vdots \\ u_{n-1} + v_{n-1} \end{bmatrix} \quad e \quad k \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} k \cdot u_0 \\ k \cdot u_1 \\ \vdots \\ k \cdot u_{n-1} \end{bmatrix}.$$

Exemplo 2.1. O espaço de estados de 1 qubit:

$$\mathbb{C}^2 = \left\{ |\psi\rangle = \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix} \mid \psi_0, \psi_1 \in \mathbb{C} e |\psi_0|^2 + |\psi_1|^2 = 1 \right\}.$$

Um espaço vetorial pode ser gerado por um conjunto de vetores linearmente independentes denominado base. A geração de um espaço vetorial consiste na combinação linear dos vetores da base. Além disso, a quantidade de vetores da base é definida como a dimensão do espaço vetorial.

Exemplo 2.2. A base canônica de \mathbb{C}^n é o conjunto de vetores:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots, \quad |n-1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Outra operação importante dos espaços vetoriais é o produto interno. Essa operação define algumas propriedades importantes sobre os vetores, isto é, quando o produto interno entre dois vetores é nulo, eles são ditos ortogonais e também, uma base é dita ortogonal se todos os seus vetores são ortogonais dois a dois. O espaço vetorial \mathbb{C}^n admite o seguinte produto interno:

$$(|\phi\rangle, |\psi\rangle) = \langle \phi | \psi \rangle = \begin{bmatrix} \phi_0^* & \phi_1^* & \dots & \phi_{n-1}^* \end{bmatrix} \cdot \begin{bmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{bmatrix} = \phi_0^* \cdot \psi_0 + \phi_1^* \cdot \psi_1 + \dots + \phi_{n-1}^* \cdot \psi_{n-1}.$$

2.1.3 Transformações Lineares

O conceito de transformação linear ou operador linear é fundamental para modelar a questão da evolução do estado quântico. Dado que o estado de um qubit é um vetor pertencente a um espaço vetorial, uma transformação linear funciona como uma função que mapeia esse vetor a outro vetor dentro do mesmo espaço vetorial. Dessa forma, uma transformação linear em um espaço vetorial complexo é uma aplicação $T : \mathbb{C}^n \rightarrow \mathbb{C}^m$ tal que:

- $T(|\phi\rangle + |\psi\rangle) = T(|\phi\rangle) + T(|\psi\rangle)$.
- $T(k \cdot |\psi\rangle) = k \cdot T(|\psi\rangle)$.

Exemplo 2.3. A função $H : \mathbb{C}^2 \rightarrow \mathbb{C}^2$ dada por:

$$H(\alpha_0 |0\rangle + \alpha_1 |1\rangle) = \frac{\alpha_0 + \alpha_1}{\sqrt{2}} |0\rangle + \frac{\alpha_0 - \alpha_1}{\sqrt{2}} |1\rangle$$

é uma transformação linear que representa uma porta lógica quântica chamada de Hadamard.

Uma forma mais conveniente de representar transformações lineares é por meio da representação matricial, com isso, aplicar a transformação a um vetor é o mesmo que multiplicar a sua matriz pelo vetor. Seja $T : U = \mathbb{C}^n \rightarrow V = \mathbb{C}^m$ uma transformação linear e sejam $\beta_U = \{|u_0\rangle, \dots, |u_{n-1}\rangle\}$ e $\beta_V = \{|v_0\rangle, \dots, |v_{m-1}\rangle\}$ bases de U e V , respectivamente, a matriz da transformação linear T nas bases β_U e β_V é dada por:

$$[T]_{\beta_V}^{\beta_U} = \begin{bmatrix} | & & | \\ [T(u_0)]_{\beta_V} & \cdots & [T(u_{n-1})]_{\beta_V} \\ | & & | \end{bmatrix}$$

onde cada coluna é formada pelas componentes das imagens dos vetores de U na base β_V .

Exemplo 2.4. A matriz da transformação linear do exemplo 2.3 é dada por:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

As transformações lineares correspondentes à descrição da evolução do estado quântico possuem uma característica em comum: elas são operações unitárias. Um operador linear U é dito unitário se ocorrer pelo menos uma das seguintes condições:

- $U^\dagger = U^{-1}$.
- As linhas ou colunas de $[U]_\beta$ são vetores ortonormais de \mathbb{C}^n para alguma base β .
- U preserva o produto interno entre os vetores.

2.1.4 Produto Tensorial

A noção de produto tensorial é o que permite a composição de sistemas quânticos de vários qubits. Dados dois espaços vetoriais U e V , com bases $\beta_U = \{|u_k\rangle\}$ e $\beta_V = \{|v_l\rangle\}$, o produto tensorial de U e V , denotado por $U \otimes V$, é definido como o espaço vetorial gerado pela base:

$$u_k \otimes v_l, \text{ com } k = 0, \dots, \dim U - 1 \text{ e } l = 0, \dots, \dim V - 1$$

Exemplo 2.5. *O sistema quântico composto de 2 qubits é formado pelo produto tensorial de dois espaços vetoriais de 1 qubit. A base desse espaço pode ser a base computacional formada pelos vetores:*

- $|00\rangle = |0\rangle |0\rangle = |0\rangle \otimes |0\rangle$
- $|01\rangle = |0\rangle |1\rangle = |0\rangle \otimes |1\rangle$
- $|10\rangle = |1\rangle |0\rangle = |1\rangle \otimes |0\rangle$
- $|11\rangle = |1\rangle |1\rangle = |1\rangle \otimes |1\rangle$

Uma maneira mais prática de definir o produto tensorial entre dois vetores ou operadores é utilizando o produto de Kronecker. Assim sendo, o produto de Kronecker da matriz $U_{m \times n}$ pela matriz V é feito conforme a equação:

$$U \otimes V = \begin{bmatrix} u_{11}V & \dots & u_{1n}V \\ \vdots & \ddots & \vdots \\ u_{m1}V & \dots & u_{mn}V \end{bmatrix}.$$

Exemplo 2.6. *Suponha a existência de dois qubits $|\psi_0\rangle$ e $|\psi_1\rangle$ preparados nos seguintes estados:*

$$|\psi_0\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}.$$

$$|\psi_1\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

A composição de $|\psi_0\rangle$ e $|\psi_1\rangle$, denotada por $|\psi_0\rangle \otimes |\psi_1\rangle$, é obtida da seguinte forma:

$$|\psi_0\rangle \otimes |\psi_1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |10\rangle.$$

2.2 INTRODUÇÃO À MECÂNICA QUÂNTICA

Até o começo do século XX, acreditava-se que a maioria das leis de Newton, Maxwell e termodinâmica eram suficientes para explicar o funcionamento do universo conhecido até então. Entretanto, por volta da década de 1930, alguns fenômenos como a radiação de corpo negro, o efeito fotoelétrico e a dualidade onda-partícula da luz colocaram a física clássica em xeque. Para explicar esses fenômenos, uma nova base matemática foi desenvolvida com o intuito de oferecer suporte ao estudo da mecânica quântica. Esta seção tem como objetivo apresentar os postulados da mecânica quântica que servem de base para a estruturação da computação quântica.

2.2.1 O Estado de um Sistema Quântico

O primeiro postulado estabelece o conceito de sistema físico isolado, que está relacionado aos qubits do computador quântico.

Postulado 1. *Um sistema físico isolado está associado a um espaço vetorial complexo munido de produto interno, ou seja, um espaço de Hilbert \mathcal{H} , que corresponde ao espaço de estados do sistema. O estado do sistema pode ser descrito por meio de um vetor unitário no espaço de estados do sistema, chamado de vetor estado.*

Dessa forma, um qubit é um sistema físico isolado, matematicamente representado por um espaço de Hilbert \mathcal{H} de dimensão 2. Uma base para esse espaço vetorial pode ser a base canônica $\{|0\rangle, |1\rangle\}$ e um estado do qubit é qualquer combinação linear de norma 1 $\alpha|0\rangle + \beta|1\rangle$ para $\alpha, \beta \in \mathbb{C}$ dos vetores da base.

2.2.2 Evolução Temporal de um Sistema Físico

O estado de um sistema físico isolado pode alterar com o passar do tempo. O postulado 2 determina como essa evolução acontece.

Postulado 2. *A evolução de um sistema físico isolado é descrita por uma operação unitária. Assim sendo, o estado do sistema $|\psi_0\rangle$ no instante t_0 está relacionado ao estado do sistema $|\psi_1\rangle$ no tempo t_1 por uma operação unitária U que depende apenas de t_0 e t_1 :*

$$|\psi_1\rangle = U|\psi_0\rangle.$$

O postulado 2 é de suma importância para a computação quântica. Assim como na computação clássica, onde os bits passam por portas lógicas que definem o processo da computação, os qubits da computação quântica também evoluem utilizando portas lógicas. Entretanto, os bits só podem ser modificados de 0 para 1 e de 1 para 0, e os qubits são modificados de uma combinação linear complexa de 0 e 1 para outra e esse processo é descrito por uma operação unitária U , que leva um vetor estado $|\psi_0\rangle$ do espaço de estados \mathcal{H} para outro, $|\psi_1\rangle$.

2.2.3 Medição

O postulado anterior afirma que sistemas físicos isolados evoluem de acordo com operadores unitários, mas essa evolução é realmente isolada e não interage com o resto do mundo. Quando ocorre uma interação com esse sistema, ele deixa de ser isolado e o postulado 2 não é mais efetivo. Assim, o postulado 3 descreve o comportamento do sistema quando ele for observado.

Postulado 3. *Dada uma base ortonormal $B = \{|\phi_i\rangle\}$ de um espaço vetorial \mathcal{H}_A de um sistema físico A , a medição de Von Neumann do sistema \mathcal{H}_A sobre a base B dado um estado*

$$|\psi\rangle = \sum_i \alpha_i |\phi_i\rangle$$

retorna i com probabilidade $|\alpha_i|^2$ e deixa o sistema no estado $|\phi_i\rangle$.

Dessa forma, este postulado descreve que um estado quântico colapsa em um estado clássico. Supondo um qubit cujo estado é descrito pela combinação linear dos vetores da base ortogonal de \mathcal{H} : $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Ao realizar o processo de medição, segundo o postulado 3, o estado quântico colapsará no estado equivalente ao estado clássico $|0\rangle$ com probabilidade $|\alpha|^2$ e no estado equivalente ao estado clássico $|1\rangle$ com probabilidade $|\beta|^2$.

2.2.4 Composição de Sistemas Quânticos

Os postulados apresentados até então consideram apenas o comportamento de um sistema físico sozinho. O postulado 4 descreve o comportamento composto de diferentes sistemas físicos, que podem ou não interagir entre si sem perderem suas características quânticas.

Postulado 4. *O espaço de estados de um sistema físico composto é descrito como o produto tensorial dos espaços de estados de cada um dos sistemas componentes. Ou seja, a composição dos sistemas dados por $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_N$ é dada por:*

$$\mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \dots \otimes \mathcal{H}_N.$$

Além disso, se o estado do sistema \mathcal{H}_i é dado por $|\psi_i\rangle$, para i de 1 a N , o estado do sistema composto também é o produto tensorial dos estados individuais:

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \dots \otimes |\psi_N\rangle.$$

Este postulado permite que possam ser utilizados diferentes qubits para compor um sistema de n qubits. O exemplo 2.6 mostra uma composição de dois diferentes qubits em um sistema composto de dois qubits. Entretanto, a composição de sistemas quânticos também permite descrever a interação entre os sistemas físicos. Em alguns casos, um sistema físico

composto não pode ser separado em sistemas individuais, quando isso acontece, o estado do sistema está emaranhado.

Um sistema composto emaranhado não pode ser descrito como um produto tensorial de seus subsistemas. O exemplo a seguir mostra um desses casos:

Exemplo 2.7. *Seja um sistema composto de dois qubits cujo estado é dado por:*

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Se o sistema for separável, então ele deveria poder ser escrito como:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

onde a e b estão relacionados com $|0\rangle$ e $|1\rangle$, respectivamente, do primeiro qubit e c e d estão relacionados com $|0\rangle$ e $|1\rangle$, respectivamente, do segundo qubit. O sistema requer que $ac = \frac{1}{\sqrt{2}}$, $ad = 0$, $bc = 0$ e $bd = \frac{1}{\sqrt{2}}$, mas isso não é possível e, portanto, o estado do sistema não pode ser descrito como um produto tensorial dos estados de seus subsistemas.

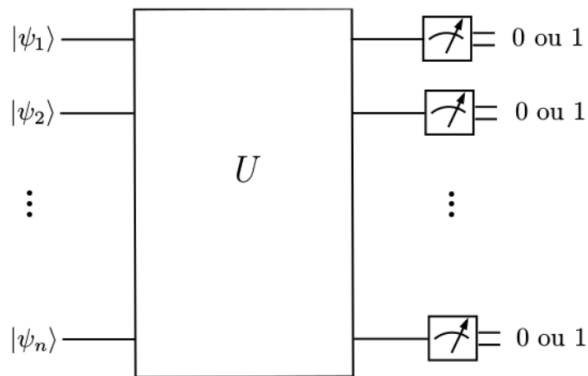
O estado do sistema do exemplo 2.7 é chamado de estado de Bell $|\beta_{00}\rangle$ e é possível perceber que o resultado da medida dos qubits são dependentes, ou seja, se o primeiro qubit for medido 0, o segundo qubit, obrigatoriamente, será medido 0, e se o primeiro qubit for medido 1, o segundo qubit, obrigatoriamente, será medido 1. No total, existem quatro estados de Bell, que são os seguintes:

- $|\beta_{00}\rangle = |\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$
- $|\beta_{01}\rangle = |\Phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$
- $|\beta_{10}\rangle = |\Psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$
- $|\beta_{11}\rangle = |\Psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$

2.3 COMPUTAÇÃO QUÂNTICA DE CIRCUITOS

Para fazer computação quântica, existem uma série de diferentes modelos. A computação quântica de circuitos é a forma mais convencional, que consiste de circuitos análogos aos circuitos dos computadores clássicos com qubits substituindo os bits e as portas lógicas quânticas substituindo as clássicas. Por meio dos circuitos quânticos, é possível descrever os qubits, a evolução de cada sistema por meio das portas lógicas, entrelaçar qubits com portas lógicas de mais de um qubit, realizar medições e todas as demais operações sobre os sistemas físicos isolados. Esta seção descreve as características e funcionamento dos circuitos quânticos, apresentando as notações, qubits, portas lógicas quânticas, medições e a matemática envolvida no processo.

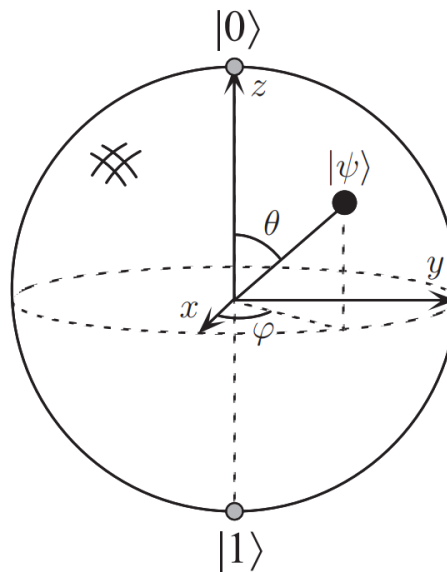
Figura 2.1 – Modelo Genérico de Computação Quântica de Circuitos.



Fonte: (PORTUGAL *et al.*, 2012)

A figura 2.1 mostra um esquema geral de um circuito quântico. $|\psi_i\rangle$ na esquerda são os qubits, U_i representam transformações do sistema por meio de operações unitárias (portas lógicas) e ao final, a medição, onde cada qubit de entrada transformado pelas portas lógicas são medidos com determinada probabilidade de ocorrer 0 ou 1. Vale ressaltar que ao contrário dos circuitos clássicos, os circuitos quânticos sempre possuem o mesmo número de entradas e saídas e, basicamente, essa é uma das condições pela qual a computação é reversível, menos quanto à medição.

Figura 2.2 – Esfera de Bloch.



Fonte: (NIELSEN; CHUANG, 2010)

2.3.1 O Qubit

De maneira geral, um qubit é versão quântica de um bit clássico. Como já dito anteriormente, enquanto um bit pode assumir valores 0 ou 1, um qubit pode assumir o valor 0, 1, ou uma infinidade de combinações lineares, inclusive complexas, de 0 e 1 vinculadas à determinada probabilidade. A esfera de Bloch apresentada na figura 2.2 é uma representação visual de um qubit, que consiste em um vetor com origem no centro da esfera e que aponta para qualquer extremidade dela. Sendo assim, um qubit em determinado estado cuja representação na esfera de Bloch é um vetor que aponta para cima terá 100% de probabilidade de ser medido $|0\rangle$, se apontar para baixo, 100% de probabilidade de ser medido $|1\rangle$, se formar um ângulo de 90° com o eixo z , tem 50% de probabilidade de ser medido $|0\rangle$ e 50% de probabilidade de ser medido $|1\rangle$.

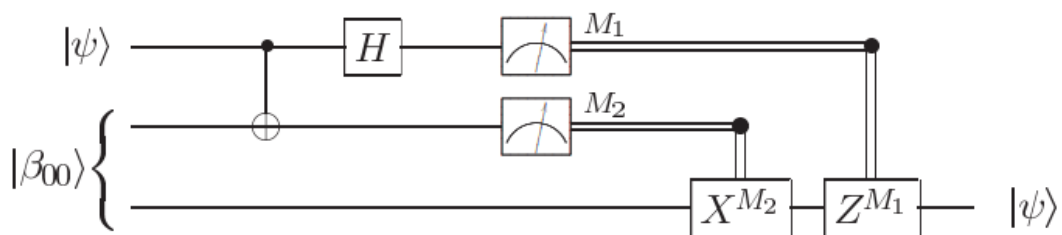
Um qubit é matematicamente representado pela equação abaixo utilizando a notação de Dirac, onde $|\alpha|^2$ representa a probabilidade de medir $|0\rangle$ e $|\beta|^2$ representa a probabilidade de medir $|1\rangle$.

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \text{ onde } |\alpha|^2 + |\beta|^2 = 1 \text{ e } \alpha, \beta \in \mathbb{C}.$$

2.3.2 Notação de Circuitos

A figura 2.3 mostra um exemplo mais completo de circuito. As notações são bastante intuitivas. Quanto à informação do circuito, linhas simples representam estados quânticos e linhas duplas representam estados clássicos. Portas lógicas quânticas podem ser representadas por retângulos ou símbolos especiais, como a primeira porta lógica do circuito em questão, chamada de *CNOT*, que age no primeiro e segundo qubit. A medição parece com uma porta lógica, mas sua entrada é um estado quântico e a saída, um estado clássico. Quanto as entradas, geralmente são representadas por estados definidos como $|0\rangle$ ou o estado de Bell $|\beta_{00}\rangle$ dos dois últimos qubits do circuito da figura 2.3.

Figura 2.3 – Notação de circuitos.



Fonte: (NIELSEN; CHUANG, 2010)

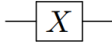

2.3.3 Portas Lógicas Quânticas

Portas lógicas quânticas são operações unitárias que são aplicadas a um ou mais qubits. Cada uma delas possui um símbolo especial e tem um efeito diferente nos n qubits em que ela atua. Matematicamente, as portas lógicas são descritas por matrizes $2^n \times 2^n$ e a aplicação da porta lógica ao qubit se dá pela multiplicação da matriz pelo vetor estado do sistema. Nesta seção, serão apresentadas as portas lógicas mais importantes.

Porta de Pauli X ou NOT Quântica

A porta lógica X é equivalente a porta NOT clássica, ela leva o sistema do estado $|0\rangle$ para o estado $|1\rangle$ e vice-versa. Entretanto, se o qubit está em algum estado de superposição, ela basicamente fará com que o vetor estado na esfera de Bloch faça uma rotação de 180° em torno do eixo x . A figura 2.4 mostra as notações possíveis para essa porta lógica, sua matriz e seu comportamento com relação aos estados fundamentais.

Figura 2.4 – Porta de Pauli X ou NOT quântica.

Símbolo	Matriz	Comportamento
	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{aligned} X 0\rangle &= 1\rangle \\ X 1\rangle &= 0\rangle \end{aligned}$
	Notação alternativa	$\begin{aligned} X b\rangle &= \bar{b}\rangle \\ b=0,1 &; \bar{b}=\text{NOT}(b) \end{aligned}$
	$X = \sigma_x$	$\begin{aligned} X +\rangle &= +\rangle \\ X -\rangle &= - -\rangle \end{aligned}$

Fonte: (POLLACHINI, 2018)

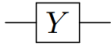
Para ilustrar matematicamente o comportamento dessa porta lógica, suponha sua aplicação sobre um qubit no estado $|0\rangle$, esse processo pode ser calculado da seguinte forma:

$$X |0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle.$$

Porta de Pauli Y

A porta Y de Pauli funciona de maneira semelhante à porta X , ela faz com que o vetor estado na esfera de Bloch realize uma rotação de 180° em torno do eixo y . A figura 2.5 mostra sua notação no circuito, matriz e comportamento. É importante notar que essa porta lógica trabalha basicamente sobre números complexos.

Figura 2.5 – Porta de Pauli Y .

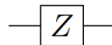
Símbolo	Matriz	Comportamento
	$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$Y 0\rangle = i 1\rangle$ $Y 1\rangle = -i 0\rangle$
	Notação alternativa $Y = \sigma_y$	$Y b\rangle = (-1)^b i \bar{b}\rangle$ $b=0,1 ; \bar{b}=\text{NOT}(b)$
		$Y +\rangle = -i -\rangle$ $Y -\rangle = i +\rangle$

Fonte: (POLLACHINI, 2018)

Porta de Pauli Z

A porta Z de Pauli também realiza rotações de 180° no vetor estado representado na esfera de Bloch, mas em torno do eixo z . A figura 2.6 mostra sua notação no circuito, matriz e comportamento. Essa porta lógica é responsável por fazer o estado mudar de $|+\rangle$ para $|-\rangle$ e vice-versa, em outras palavras, inverter o sentido do vetor apontando para o sentido positivo do eixo x para o sentido negativo e vice-versa.

Figura 2.6 – Porta de Pauli Z .

Símbolo	Matriz	Comportamento
	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$Z 0\rangle = 0\rangle$ $Z 1\rangle = - 1\rangle$
	Notação alternativa $Z = \sigma_z$	$Z b\rangle = (-1)^b b\rangle$ $b=0,1$
		$Z +\rangle = -\rangle$ $Z -\rangle = +\rangle$

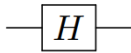
Fonte: (POLLACHINI, 2018)

Porta de Hadamard

Outra porta lógica importante é a porta de Hadamard, representada pela letra H . Essa porta é responsável por colocar o qubit em sobreposição, ou seja, fazer com que haja 50% de probabilidade de se medir 0 e 50% de probabilidade de se medir 1. Na esfera de Bloch, a porta de Hadamard faz uma rotação de 90° em torno do eixo y seguida de uma

rotação de 180° em torno do eixo x , fazendo com que o estado mude de $|0\rangle$ e $|1\rangle$ para $|+\rangle$ e $|-\rangle$ e vice-versa. A figura 2.7 mostra seu símbolo, matriz e comportamento.

Figura 2.7 – Porta de Hadamard.

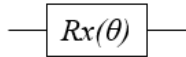
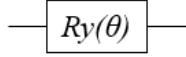
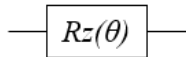
Símbolo	Matriz	Comportamento
	$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$H 0\rangle = +\rangle$ $H 1\rangle = -\rangle$ $H +\rangle = 0\rangle$ $H -\rangle = 1\rangle$

Fonte: (POLLACHINI, 2018)

Exemplificando, suponha a aplicação dessa porta lógica ao estado inicial $|0\rangle$ mostrada abaixo. Pode-se perceber que para esse caso, $\alpha = \beta = \frac{1}{\sqrt{2}}$, sendo assim, a probabilidade de medir 0 e 1 é dada por $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2} = 50\%$.

$$H |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle).$$

Figura 2.8 – Portas R_x , R_y e R_z .

Símbolo	Matriz
	$R_x(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$
	$R_y(\theta) = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$
	$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$

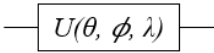
Portas R_x , R_y e R_z

As portas lógicas quânticas R_x , R_y e R_z são bastante úteis para algumas aplicações. Elas permitem a realização de rotações nos vetores estado representados na esfera de Bloch em torno dos eixos x , y e z . Para isso, elas precisam de um argumento θ que corresponde ao ângulo dessa rotação. A figura 2.8 mostra os símbolos de cada uma dessas portas lógicas e suas matrizes.

Porta Genérica U

A última porta lógica de 1 qubit que será apresentada é uma porta genérica U . Essa porta é capaz de descrever qualquer transformação do qubit. Para isso, ela precisa de três argumentos, θ , ϕ e λ , que correspondem aos três ângulos de Euler da esfera de Bloch. A figura 2.9 mostra sua notação de circuito e representação matricial.

Figura 2.9 – Porta U .

Símbolo	Matriz
	$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos \frac{\theta}{2} & -e^{i\lambda} \sin \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} & e^{i(\phi+\lambda)} \cos \frac{\theta}{2} \end{bmatrix}$

Por ser uma porta lógica genérica, valem as seguintes equivalências entre ela e as portas R_x e R_y :

$$U\left(\theta, -\frac{\pi}{2}, \frac{\pi}{2}\right) = R_x(\theta),$$

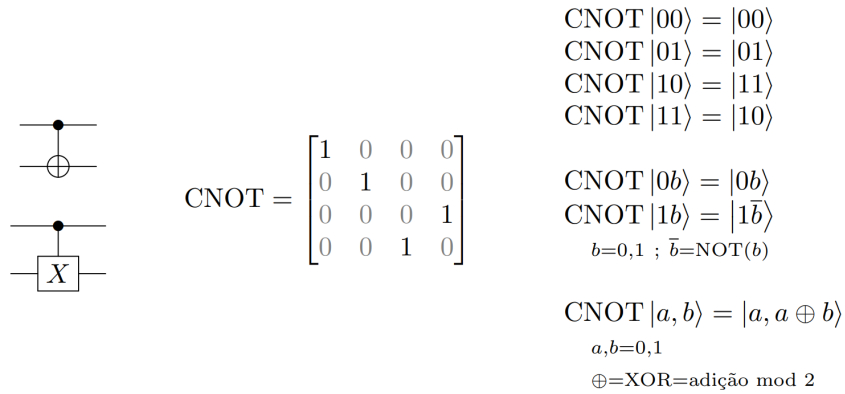
$$U(\theta, 0, 0) = R_y(\theta).$$

Porta $CNOT$

As portas lógicas de 2 qubits são representadas por matrizes 4×4 e são fundamentais para conseguir a propriedade de entrelaçamento quântico. A primeira porta lógica que será vista é a $CNOT$, que é basicamente uma porta NOT controlada, ou seja, se o qubit de controle estiver no estado $|1\rangle$, será aplicada uma porta NOT ao qubit controlado. A figura 2.10 mostra os seus símbolos na notação de circuitos, sua representação matricial e o comportamento nos estados básicos.

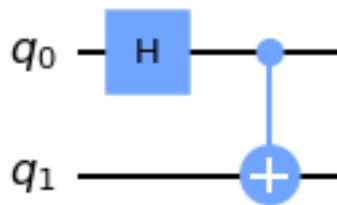
Para exemplificar seu uso, supondo o circuito apresentado na figura 2.11, que possui dois qubits onde uma porta de Hadamard é aplicada ao primeiro qubit e em seguida, uma porta $CNOT$ é aplicada com o primeiro qubit sendo o qubit de controle e o segundo, o alvo.

Figura 2.10 – Porta *CNOT*.



Fonte: (POLLACHINI, 2018)

Figura 2.11 – Circuito para obtenção do estado de Bell $|\beta_{00}\rangle$.



Matematicamente, após a aplicação da porta de Hadamard e composição do sistema já demonstrado no exemplo 2.6, o processo de computação da porta *CNOT* ao sistema de dois qubits é dado da seguinte forma:

$$\text{CNOT} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle).$$


Portas *Z* Controlada

A porta *Z* controlada tem funcionamento similar à porta *CNOT*, a única mudança é que no lugar de aplicar uma porta *X* no qubit alvo, a porta *Z* controlada aplica uma porta *Z* no qubit alvo. Assim sendo, a figura 2.12 mostra o símbolo, matriz e comportamento dessa porta lógica.

Porta *SWAP*

Uma porta lógica de dois qubits bastante importante é a porta *SWAP*. Essa porta é capaz de trocar dois qubits de lugar no circuito, levando o estado de $|01\rangle$ para $|10\rangle$ e


Figura 2.12 – Porta Z controlada.

Símbolo	Matriz	Comportamento
	$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	$CZ 00\rangle = 00\rangle$ $CZ 01\rangle = 01\rangle$ $CZ 10\rangle = 10\rangle$ $CZ 11\rangle = - 11\rangle$ $CZ 0b\rangle = 0b\rangle$ $CZ 1b\rangle = Z_2 1b\rangle = 1\rangle(Z b\rangle)$ $b=0,1$

Fonte: (POLLACHINI, 2018)

vice-versa. A figura 2.13 mostra o seu símbolo na notação de circuitos, sua representação matricial e comportamento nos estados básicos.

Figura 2.13 – Porta $SWAP$.

Símbolo	Matriz	Comportamento
	$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$SWAP 00\rangle = 00\rangle$ $SWAP 01\rangle = 10\rangle$ $SWAP 10\rangle = 01\rangle$ $SWAP 11\rangle = 11\rangle$ $SWAP ab\rangle = ba\rangle$ $a,b=0,1$

Fonte: (POLLACHINI, 2018)

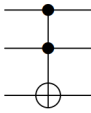
Supondo a aplicação dessa porta lógica ao estado composto obtido no exemplo 2.6. Para calcular a aplicação da porta $SWAP$, basta fazer conforme segue:

$$SWAP \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} (|00\rangle + |01\rangle).$$

Porta Toffoli ou $CCNOT$

As portas lógicas de três qubits são representadas por matrizes 8×8 e funcionam de maneira semelhante às portas de dois qubits. A porta $CCNOT$, também conhecida como Toffoli, é uma $CNOT$ com dois qubits de controle, se ambos forem medidos 1, uma porta NOT é aplicada ao terceiro qubit. A figura 2.14 mostra seu símbolo, matriz e comportamento.

Figura 2.14 – Porta *CCNOT*.

Símbolo	Matriz	Comportamento
	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{aligned} \text{CCNOT } 00c\rangle &= 00c\rangle \\ \text{CCNOT } 01c\rangle &= 01c\rangle \\ \text{CCNOT } 10c\rangle &= 10c\rangle \\ \text{CCNOT } 11c\rangle &= 11\bar{c}\rangle \\ c=0,1 ; \bar{c}=\text{NOT}(c) \end{aligned}$ $\begin{aligned} \text{CCNOT } a, b, c\rangle \\ &= a, b, (a \cdot b) \oplus c\rangle \\ a, b, c=0,1 \\ \cdot &=\text{AND} \\ \oplus &=\text{XOR}=\text{adição mod } 2 \end{aligned}$

Fonte: (POLLACHINI, 2018)

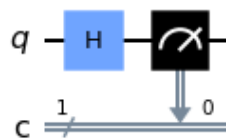
É importante notar que as portas lógicas de três qubits ou mais não são verdadeiramente necessárias, pois elas podem ser obtidas combinando portas lógicas de um e dois qubits utilizando algumas identidades de circuitos. Entretanto, elas ainda são úteis para descrever alguns processos de maneira mais simplificada.

Dentro das portas lógicas controladas de três qubits, ainda existem portas *CCZ*, *CCY* e *CSWAP*, mas elas funcionam de maneira análoga à *CCNOT* e não serão apresentadas neste trabalho.

2.4 MEDIÇÃO

Depois que a computação é feita, é necessário realizar o processo de medição. Quando ela ocorre, o estado quântico é perdido e colapsa em uma saída clássica. A figura 2.15 mostra um exemplo de circuito onde uma porta de Hadamard é aplicada e em seguida, a medida é feita. *c* representa um bit clássico onde a medida feita no qubit quântico *q* será armazenada.

Figura 2.15 – Circuito com aplicação da porta de Hadamard e medida.



A medição é feita com base nas probabilidades do estado do sistema quântico. Por exemplo, ao medir o estado do circuito em questão, tem-se $\left|\frac{1}{\sqrt{2}}\right|^2 = 50\%$ de probabilidade

de ser medido tanto 0 quanto 1.

2.5 TEOREMA DA NÃO-CLONAGEM

Um grande problema da computação quântica é que um estado quântico não pode ser copiado. Não existe uma porta lógica que faça uma cópia idêntica de qualquer qubit e transmita para outro qubit, em outras palavras:

Teorema 1. *Dado um espaço de Hilbert \mathcal{H} , não existe uma operação unitária $U : \mathcal{H} \otimes \mathcal{H} \rightarrow \mathcal{H} \otimes \mathcal{H}$ que satisfaça, existindo um estado $|s\rangle$, para todo estado $|\psi\rangle$ de 1 qubit, o seguinte:*

$$U(|\psi\rangle|s\rangle) = |\psi\rangle|\psi\rangle.$$

.

Esse teorema é fundamental para a lógica da mecânica quântica. O estado de cada sistema físico isolado é sempre único, o que faz com que a medição do sistema siga determinada distribuição de probabilidade, mas não seja previsível. A possibilidade de clonar um sistema físico isolado faria com que essa singularidade seja perdida, já que haveria outro sistema idêntico.

3 COMPUTADORES QUÂNTICOS SUPERCONDUTORES

A computação quântica é claramente uma tecnologia bastante revolucionária para a ciência da computação. Mas no fundo, a modelagem de circuitos quânticos é uma abstração do que ocorre dentro dos computadores quânticos, ela supõe que a evolução dos estados quânticos ocorre magicamente da forma como descrita pelas operações unitárias. Entretanto, a computação quântica surge da mecânica quântica e sistemas quânticos são altamente instáveis e suscetíveis a ruídos que ocorrem tanto quando os qubits interagem de alguma forma com o ambiente quanto por meio de imperfeições no uso dos elementos do circuito quântico (ENDO; BENJAMIN; LI, 2018).

Existem várias formas de se implementar computação quântica em dispositivos físicos, segundo Gill *et al.* (2021), os principais meios para a implementação física de qubits são armadilha de íons, supercondutores, silício e fotônica, dentre os quais, Maslov, Nam e Kim (2018) também confirmam que as tecnologias de armadilha de íons e supercondutores têm sido a base para a geração atual de máquinas quânticas disponíveis para acesso via Cloud.

Os computadores quânticos de supercondutores são os utilizados pela IBM e Google. Mas essa tecnologia ainda não é desenvolvida o suficiente, a implementação de um qubit lógico da forma com que a teoria apresenta ainda é algo distante. Isso implica o surgimento de diferentes áreas de pesquisas focadas em implementar qubits mais precisos, dentre elas, está o controle quântico, que tem como objetivo otimizar a manipulação dos sistemas quânticos.

Esta seção consiste na descrição do funcionamento de um computador quântico de supercondutores, apresentando os conceitos de supercondutividade e qubits baseados em transmon. Com base nisso, será detalhado o funcionamento físico dos qubits com relação às portas lógicas quânticas de 1 qubit que são traduzidas em pulsos de micro-ondas específicos a serem aplicadas aos qubits. Sendo assim, a área de controle quântico será apresentada na perspectiva de um usuário de computadores quânticos que busca melhorar o resultado de seus experimentos.

3.1 SUPERCONDUTIVIDADE

Para entender o funcionamento de um computador quântico supercondutor, é importante entender o que faz um determinado material ser um supercondutor. De acordo com team (2022), existem dois tipos de elétrons em uma corrente elétrica: os da banda de valência e os da banda de condução. Os elétrons da banda de condução tem o poder de trafegar livremente pelo material, enquanto os elétrons da banda de valência estão presos aos átomos do material. Dessa forma, a condutividade de um material corresponde à facilidade em fazer com que os elétrons da banda de valência se tornem elétrons da banda de condução.

Apesar de materiais condutores não precisarem de energia para levar elétrons da banda de valência para a de condução, eles não possuem infinita condutividade por conta de um fenômeno chamado princípio da exclusão de Pauli. O princípio da exclusão diz que existe um limite para o número de elétrons com o mesmo número atômico em um átomo (AAKASH, 2022), ou seja, com o mesmo nível discreto de energia, mas isso não se aplica apenas aos elétrons da camada de valência, os elétrons da camada de condução também possuem diferentes níveis discretos de energia. Por conta disso, quanto mais ocupada estiver a banda de condução, mais energia é necessário para promover os elétrons da banda de valência.

A supercondutividade é um estado da matéria que permite que uma corrente elétrica persista indefinidamente, ela não possui resistência elétrica e também não permite a interferência de campos magnéticos (SUTTER, 2021). Segundo Mooij (2004), para criar um qubit é necessário possuir um sistema quântico isolado que mantenha a interação entre os elétrons e entre os elétrons e fótons, portanto, materiais supercondutores são fundamentais para a computação quântica, já que eles permitem manter o estado quântico do sistema por não necessitarem de interação.

O fenômeno da supercondutividade acontece com materiais específicos em baixas temperaturas. Cern (2022) explica que conforme um elétron da banda de condução passa pelo material, ele atrai o núcleo dos átomos gerando regiões de alta carga positiva que, por sua vez, atrai outro elétron formando um par de Cooper, que são facilmente separados pela energia térmica, é por esse motivo que esse fenômeno normalmente ocorre em baixas temperaturas. Os pares de Cooper têm o poder de se mover pelo material sem resistência alguma porque é necessário uma certa quantidade de energia para separá-los (MOOIJ, 2004) e, além disso, eles não precisam obedecer o princípio da exclusão, portanto, a condução elétrica é infinita.

3.2 TRANSMON

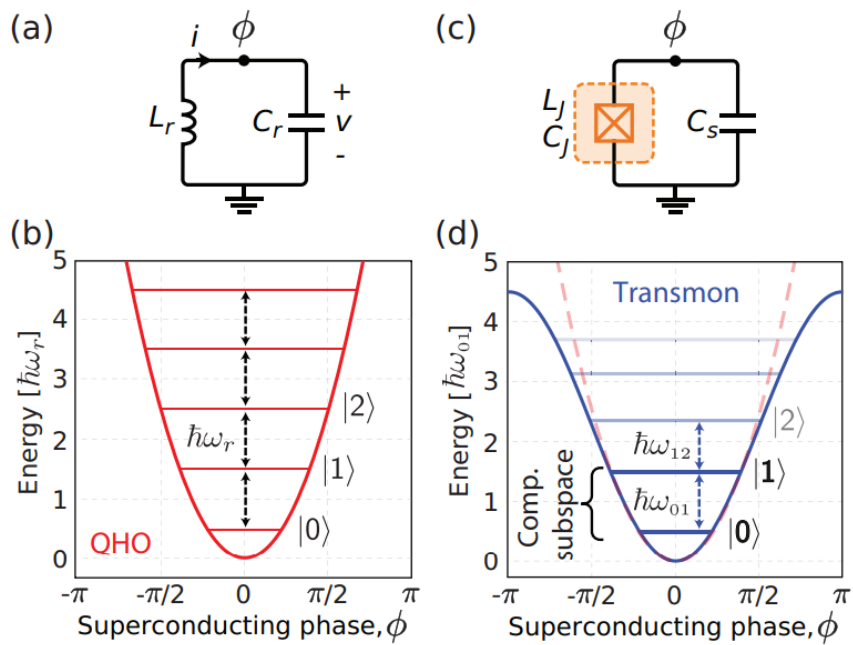
A construção de qubits pode ser feita de várias formas, mas elas precisam de propriedades quânticas. team (2022) e Mooij (2004) apresentam uma descrição bastante didática para a construção de qubits utilizando supercondutores, que serão utilizados como referência para esta seção.

A utilização de um elétron na banda de valência de um átomo corresponde às propriedades necessárias de um qubit, pois há diferentes níveis discretos de energia que poderiam corresponder aos estados $|0\rangle$ e $|1\rangle$ quando medidos. Todavia, átomos são dados pela natureza e é tecnologicamente inviável manipulá-los para construir sistemas quânticos para implementar qubits. O transmon é uma solução baseada em um circuito elétrico supercondutor. A supercondutividade oferece as propriedades quânticas necessárias para a construção de um qubit, basta que seja possível manipulá-lo de alguma forma.

Um qubit transmon funciona como um átomo artificial. Ele é baseado em um

circuito LC, que é um circuito composto de um indutor (L) e um capacitor (C), conforme é possível visualizar na figura 3.1 (a), porém, esse tipo de circuito faz com que o espaço entre os diferentes níveis discretos de energia seja constante (figura 3.1 (b)) e isso faria com que fótons idênticos ocasionem transições entre vários pares de estados de energia, o que não é desejável no caso de um sistema quântico de dois níveis. Mas graças ao tunelamento quântico, é possível adicionar um material isolante entre dois materiais supercondutores (figura 3.1 (c)) para fazer com que o espaçamento entre os níveis de energia seja não uniforme (figura 3.1 (d)), dessa forma, apesar do material ser isolante, alguns pares de Cooper conseguem passar pelo material. Esse efeito é chamado de efeito Josephson.

Figura 3.1 – Circuitos supercondutores e o regime transmon.



Fonte: (KRANTZ *et al.*, 2019)

Ademais, é necessário que um qubit possa ser controlado de alguma forma. Para isso, é preciso adicionar ao circuito um capacitor que atua como uma porta lógica recebendo sinais externos que controlarão o qubit, nesse caso, os sinais são fótons. Contudo, a adição de um capacitor no circuito faz com que haja alterações na posição dos níveis de energia dos elétrons, diante disso, surge um problema em determinar o tamanho do primeiro capacitor do circuito, pois se ele for muito grande, reduz a sensibilidade da porta lógica. Mas existe um regime ideal para ajustar a capacitância e preservar a não linearidade dos espaços entre os níveis de energia dos elétrons, esse regime é chamado de regime de transmon, o que caracteriza um qubit transmon.

Assim sendo, a diferença de energia entre os estados é dada por $E = \hbar\omega$, onde \hbar é a constante de Planck e ω é a frequência dos fótons necessária para estimular o qubit e realizar as transições do estado quântico. Posteriormente será visto que essa variável é

bastante importante para o controle quântico, já que as portas lógicas são traduzidas em pulsos de micro-ondas que devem ser aplicados na frequência ω .

Para se obter informação a respeito do estado do sistema, é necessário utilizar fótons em uma frequência diferente de ω , isso porque os fótons na frequência ω são absorvidos pelo circuito, enquanto os fótons em frequências distantes de ω são dispersados. Quando isso ocorre, além do estado quântico ser perdido porque houve uma interação, a frequência do fóton dispersado sofre uma pequena variação, que pode ser medida afim de obter a informação sobre o estado do qubit, caracterizando a medição.

3.3 PROGRAMAÇÃO DE HARDWARE QUÂNTICO COM QISKIT PULSE

Tendo em mente que qubits são circuitos supercondutores que simulam o comportamento de elétrons individuais que podem estar em diferentes níveis discretos de energia, denominados $|0\rangle$ e $|1\rangle$ para a computação quântica, e esses elétrons podem ser manipulados por meio de fótons em determinada frequência, que atuam como as portas lógicas dos computadores quânticos. Esta seção sobe um nível de abstração com relação à seção anterior e apresenta alternativas para a descrição de portas lógicas quânticas em pulsos de micro-ondas, caracterizando o hardware. Para isso, serão utilizadas as ferramentas oferecidas pelo Qiskit (IBM, 2022c) da IBM Quantum Experience (IBM, 2022b), mais especificamente o Qiskit Pulse (IBM, 2022d), que oferece ferramentas para programação quântica de baixo nível.

Já foi comentado que portas lógicas quânticas são descritas como pulsos de micro-ondas. Mas é importante entender quais são os seus parâmetros. Um pulso é descrito como uma forma de onda, essa onda possui os parâmetros de frequência e amplitude, e um pulso pode ser aplicado em um qubit por um determinado tempo. A frequência corresponde a diferença de energia entre os estados $|0\rangle$ e $|1\rangle$ (IBM, 2022a), é necessário que ela seja devidamente calibrada para que os fótons sejam absorvidos pelo qubit. Os valores de frequência de qubits de supercondutores ficam por volta do espectro de frequência das micro-ondas, de 10^9 a 10^{12} Hz, ou de 1 GHz a 1 000 GHz. A amplitude do pulso corresponde a sua intensidade, será visto que quanto maior a amplitude, maior é a sua influência no estado do qubit. Dessa forma, um pulso com alta amplitude aplicado em um pequeno intervalo de tempo pode ter o mesmo resultado de um pulso com baixa amplitude em um intervalo de tempo maior.

O Qiskit Pulse oferece variadas formas de descrever um pulso, consistindo de modelos genéricos ou parametrizados. O Waveform é a forma mais genérica de se fazer isso, ele recebe uma lista de valores complexos que correspondem a valores de amplitude do pulso que são aplicados em termos do período de amostragem do computador quântico, que é a unidade de tempo dos pulsos. O Waveform permite um controle bastante preciso sobre o hardware, sendo útil para a implementação de técnicas de mitigação de erros e será utilizado com bastante frequência neste trabalho.

O Qiskit Pulse também oferece várias formas de representar pulsos em formas de onda, utilizando padrões parametrizados. Uma breve descrição sobre as representações mais importantes está apresentada a seguir:

- `constant`: gera um pulso com uma amplitude constante por um determinado tempo.
- `square`: gera um pulso padrão square.
- `gaussian`: gera um pulso com o formato gaussiano de desvio padrão σ e determinada amplitude no ponto médio.
- `gaussian_square`: é um pulso do tipo square com formato gaussiano.
- `drag`: vem de Derivative Removal by Adiabatic Gate (DRAG) e gera uma variação do pulso gaussiano com o objetivo de reduzir possíveis erros de vazamentos do nível de energia do estado $|1\rangle$ para o nível superior $|2\rangle$.

A obtenção dos parâmetros da caracterização do computador quântico pode ser feita tanto por métodos disponíveis no Qiskit, que oferecem uma estimativa suficientemente boa, quanto realizando experimentos com o intuito de otimizar esses valores. A frequência de um qubit pode ser obtida utilizando o `backend_defaults.qubit_freq_est`.

Outra funcionalidade importante do Qiskit é o `instruction_schedule_map`, que mostra como as portas lógicas padrão são implementadas em cada computador quântico em função das descrições de pulsos apresentadas anteriormente. Por exemplo, a figura 3.2 mostra como uma porta lógica X é traduzida para pulsos no computador quântico Nairobi. Nesse caso, ela utiliza um pulso do tipo `drag` com seus respectivos parâmetros.

Figura 3.2 – Tradução da porta lógica X em pulsos para o Nairobi.

```
backend_defaults = backend.defaults()
print(backend_defaults.instruction_schedule_map.get('x', 0))
✓ 0.1s
Schedule((0, Play(Drag(duration=160, amp=(0.13966859589497513+0j), sigma=40,
beta=-0.31863344078568917, name='Xp_d0'), DriveChannel(0), name='Xp_d0')), name="x")
```

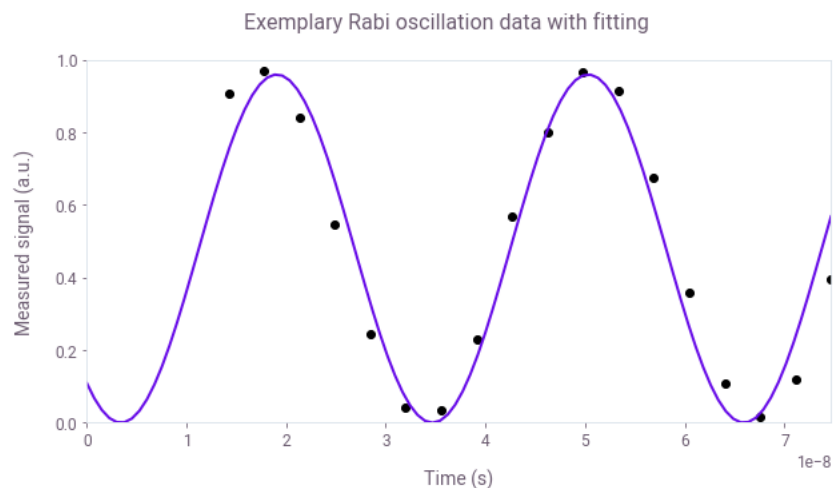
3.4 EXPERIMENTO DE RABI

Dentre os parâmetros de um pulso que descreve uma porta lógica, a frequência pode ser obtida através das estimativas oferecidas pelo Qiskit, mas a amplitude e o tempo são valores relacionados na qual deve se conhecer para que o estado do qubit possa ser controlado. O experimento de Rabi busca testar o hardware quântico com o objetivo de

avaliar o seu comportamento com relação à determinados pulsos, buscando encontrar uma relação entre os atributos de amplitude e tempo e o estado quântico. Esse é o passo inicial para o projeto de portas lógicas, os resultados do experimento de Rabi permitem a criação de qualquer porta lógica. IBM (2022a) e QCTRL (2022) apresentam os materiais utilizados como base para esta seção. Além disso, o apêndice A apresenta um código aprimorado e bem explicado que pode ser executado por meio de um jupyter notebook.

Em termos práticos, o experimento de Rabi consiste em gerar pulsos com diferentes amplitudes e aplicá-los ao qubit por um determinado tempo. Em cada pulso, acompanha-se o estado quântico realizando medições e ao final de cada experimento, espera-se um resultado semelhante ao que aparece no gráfico da figura 3.3. O gráfico mostra o sinal medido ao longo do tempo para um pulso de determinada amplitude, que pode ser entendido como a probabilidade de se medir 1, com pontos indicando as medições feitas e a linha contínua representando um ajuste a uma função cosseno: $y = A \times \cos^2(2\pi \times rabi_freq \times x + \phi)$, que auxilia na caracterização desse comportamento. Com base nele, é possível saber o que acontece com o estado quântico quando submetido a esse pulso, nesse caso, por exemplo, percebe-se que o pulso precisa de aproximadamente $1,56 \times 10^{-8}$ segundos para levar o estado quântico de $|0\rangle$ para $|1\rangle$ e vice-versa. O resultado do gráfico também pode ser definido como oscilações de Rabi, associado a uma certa frequência que determina, por exemplo, a velocidade com que acontecem rotações no vetor estado da esfera de Bloch.

Figura 3.3 – Resultado do experimento de Rabi para um pulso qualquer.

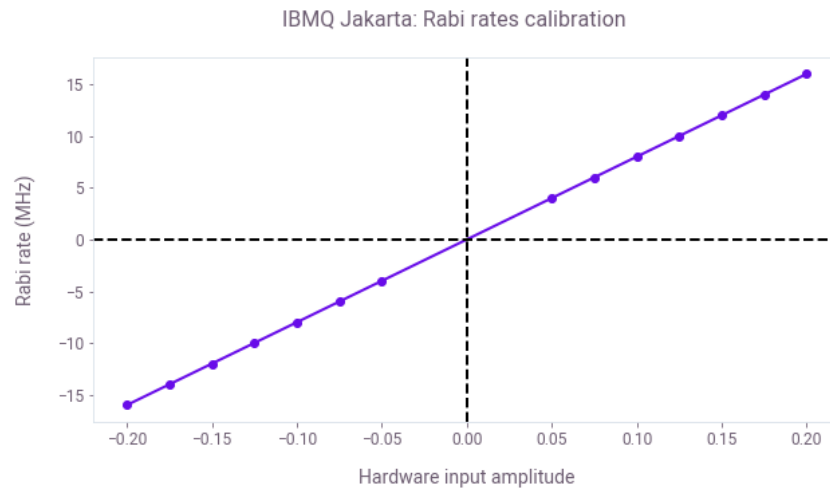


Fonte: (QCTRL, 2022)

A partir da repetição dos experimentos para pulsos com diversas amplitudes, obtém-se as frequências de Rabi para cada amplitude. Com base nisso, é possível definir uma função que dada uma frequência de Rabi, determina qual a amplitude necessária para o pulso. A figura 3.4 mostra um exemplo dessa função, no eixo x estão as diferentes

amplitudes com que os experimentos foram feitos e no eixo y , as frequências de Rabi para essas amplitudes.

Figura 3.4 – Frequências de Rabi para um experimento.



Fonte: (QCTRL, 2022)

A função representada no gráfico da figura 3.4 é de suma importância para a geração de portas lógicas. Com base nela, é possível saber quanto tempo é necessário para realizar uma rotação completa na esfera de Bloch, por exemplo, o que caracteriza uma porta X , em um pulso com determinada amplitude, ou ainda, escolhendo um tempo de aplicação do pulso e o resultado desejado, como uma porta X , saber a amplitude necessária para o pulso.

Um ponto que não foi comentado é que os estados de um qubit admitem valores complexos, ou seja, a esfera de Bloch é tridimensional e podem ser feitas rotações no vetor de estado tanto com relação ao eixo x quanto com o eixo y , por exemplo. O experimento de Rabi com amplitudes reais considera rotações em apenas um dos eixos, mas os resultados são os mesmos se forem consideradas amplitudes complexas, que fará rotações no outro eixo. Dessa forma, é possível utilizar o mesmo resultado para descrever portas lógicas que usam ambos os eixos x e y .

3.5 DESIGN E CALIBRAÇÃO DE PORTAS LÓGICAS COM Q-CTRL BOULDER OPAL

Com base no experimento de Rabi, é possível construir diferentes pulsos que reproduzem o comportamento de uma mesma porta lógica. O Boulder Opal da Q-CTRL (Q-CTRL, 2022b) oferece uma série de ferramentas que facilitam o design e calibração de portas lógicas personalizadas resistentes à ruídos. Os experimentos realizados neste trabalho consideram a utilização dos manuais Q-CTRL (s.d.) e QCTRL (2022) para construir

as diferentes portas lógicas, as explicações de seus funcionamentos estão apresentadas nesta seção.

O Boulder Opal utiliza um hamiltoniano para descrever a evolução do sistema quântico:

$$H(t) = (1 + \beta_\gamma(t)) H_c(t) + \eta(t) \sigma_z \quad (1)$$

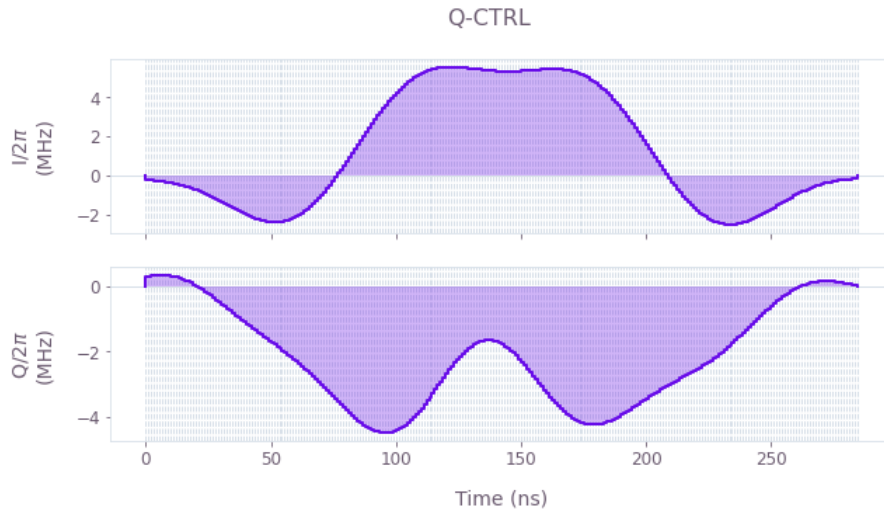
onde $\beta_\gamma(t)$ representa um processo de flutuação da amplitude do pulso, $\eta(t) \sigma_z$ representa um processo estocástico de *dephasing* considerado constante e $H_c(t)$ é o Hamiltoniano controlador definido por:

$$H_c(t) = \frac{1}{2} (\gamma^*(t) \sigma_- + \gamma(t) \sigma_+) = \frac{1}{2} (I(t) \sigma_x + Q(t) \sigma_y)$$

onde $\gamma(t) = I(t) + iQ(t)$ é um valor complexo dependente do tempo que representa a forma de onda do pulso e $\sigma_k, k = x, y, z$ são as matrizes de Pauli.

Nesses termos, uma porta lógica descrita em pulsos tem um formato semelhante ao representado pela figura 3.5. Esse gráfico mostra as frequências de Rabi desejadas para cada instante de tempo no intervalo de ação do pulso. Os valores I e Q se referem às frequências reais e complexas da decomposição IQ dos pulsos conhecida na engenharia elétrica, para a computação quântica, essas frequências correspondem às rotações realizadas nos eixos x e y da esfera de Bloch.

Figura 3.5 – Exemplo de pulso gerado pelo Boulder Opal.



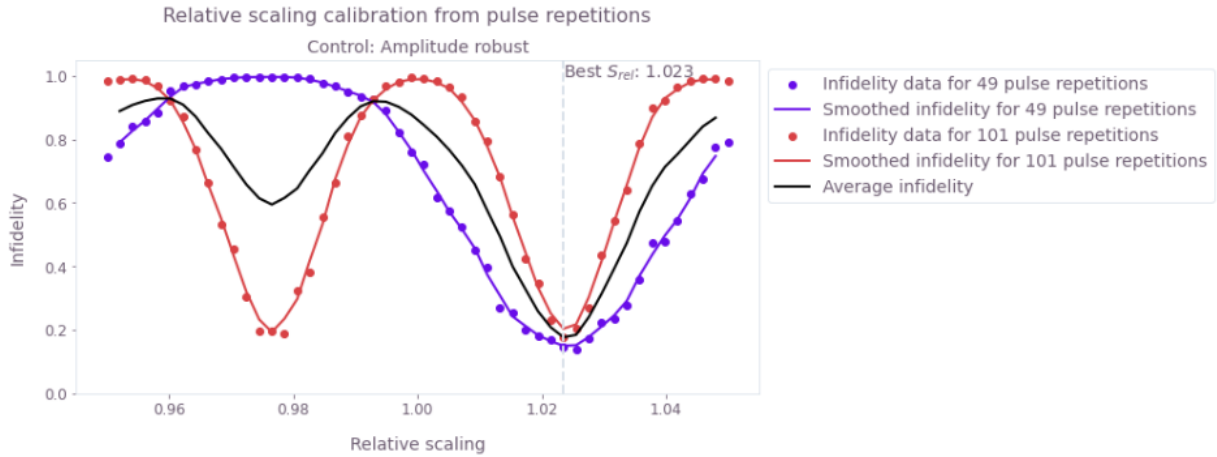
Fonte: (Q-CTRL, s.d.)

Após o pulso ser construído, pode ser necessário calibrá-lo para corresponder melhor à calibração de Rabi. Essa técnica consiste em parametrizar a relação entre as frequências de Rabi e as amplitudes necessárias do pulso, ou seja, a relação $(A_I, A_Q) \longleftrightarrow (I, Q)$. Assim, a função que descreve a amplitude do pulso ao longo do tempo se torna:

$$\gamma(t) = S_{amp} (S_{rel} A_I(t) + i A_Q(t)) \quad (2)$$

onde os parâmetros S_{amp} e S_{rel} são valores próximos de 1.

Figura 3.6 – Exemplo de obtenção do parâmetro S_{rel} .



Fonte: (QCTRL, 2022)

Figura 3.7 – Exemplo de obtenção do parâmetro S_{amp} .



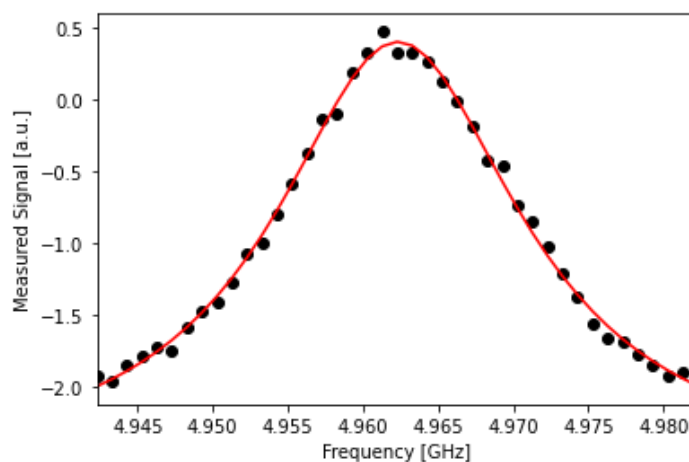
Fonte: (QCTRL, 2022)

Para encontrar S_{amp} e S_{rel} , é necessário realizar um experimento para cada um dos dois parâmetros onde para uma dada quantidade de repetições do pulso, o parâmetro é alterado para valores no intervalo desejado e os valores ótimos são obtidos de um ponto mínimo da interpolação entre os pontos. As figuras 3.6 e 3.7 mostram exemplos dos resultados dos experimentos de obtenção dos valores de S_{rel} e S_{amp} , respectivamente.

3.6 OTIMIZAÇÃO DA FREQUÊNCIA

O Qiskit oferece uma boa estimativa de frequência do qubit para a aplicação das portas lógicas quânticas. Entretanto, a frequência é um ponto crítico para a criação das portas lógicas e quanto melhor for sua estimativa, mais preciso será o processo de computação quântica. A seção 3.2 explica que pulsos na frequência ω são absorvidos pelo qubit, enquanto pulsos consideravelmente fora dessa frequência podem fazer com que o qubit perca sua ressonância. Dessa forma, é importante que sejam feitos experimentos para encontrar o melhor valor de frequência possível, conforme descrito em IBM (2022a).

Figura 3.8 – Exemplo de resultado de otimização da frequência.

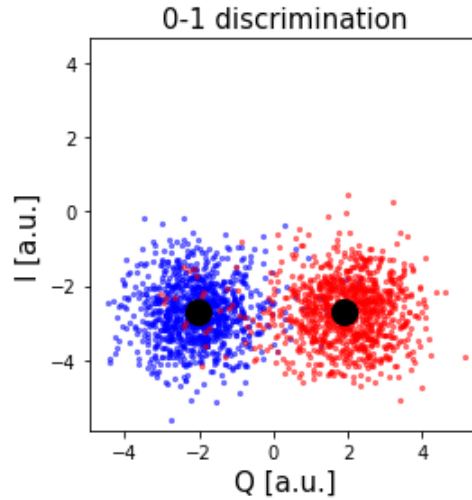


Fonte: (IBM, 2022a)

Para calibrar a frequência do qubit, é feito um experimento onde a partir da estimativa de frequência dada pelo Qiskit, é definido um intervalo de frequências para qual um pulso será aplicado e o resultado, medido. Fazendo isso, espera-se um resultado semelhante ao da figura 3.8. O eixo x representa a frequência de cada pulso e o eixo y mostra o sinal medido. É feito um ajuste dos resultados dos experimentos a uma curva específica que caracteriza o resultado. Assim, a frequência do qubit é obtida por meio do pico no centro do gráfico, que, por ser um valor alto, é um sinal de que foi possível controlar o qubit da melhor forma possível.

3.7 DISTINÇÃO ENTRE OS ESTADOS 0 E 1

Além dos resultados do experimento de Rabi e calibração da frequência dos pulsos, pode ser necessário definir com mais precisão quais são os estados $|0\rangle$ e $|1\rangle$ quando feita a medição. Os valores retornados pela medição mais crua do Qiskit gera um aspecto a ser calibrado afim de melhorar os resultados da computação. IBM (2022a) mostra o código necessário para fazer esse experimento, servindo de base para essa seção.

Figura 3.9 – Distinção entre os estados $|0\rangle$ e $|1\rangle$.

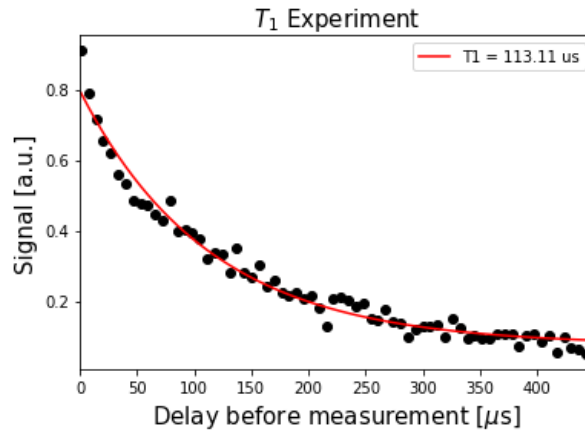
Fonte: (IBM, 2022a)

O experimento para fazer isso consiste em preparar o qubit nos estados $|0\rangle$ e $|1\rangle$ diversas vezes, medi-los e realizar uma calibração aplicando um discriminador que separa ambos os resultados em um plano IQ , que considera os valores medidos reais (I) e complexos (Q) dos estados, utilizando uma linha equidistante à média de cada resultado. Sendo assim, o resultado desse experimento é algo parecido com o gráfico da figura 3.9, onde os pontos da esquerda correspondem ao estado $|0\rangle$ e os da direita, ao estado $|1\rangle$. Os pontos escuros são a média dos resultados, dessa forma, se o resultado medido estiver mais próximo da média de $|0\rangle$, é considerado o estado $|0\rangle$, se estiver mais próximo da média de $|1\rangle$, é considerado o estado $|1\rangle$.

3.8 O TEMPO T_1

As tecnologias de computadores quânticos atuais têm dificuldade em manter o estado quântico por muito tempo, ele tende a se tornar um estado clássico muito rapidamente. Por esse motivo, quanto mais longo for o processo de computação, mais problemático será o resultado. O tempo T_1 de um qubit é o tempo que leva para o estado quântico $|1\rangle$ decair para o estado $|0\rangle$ e ele pode ser obtido a partir de experimentos.

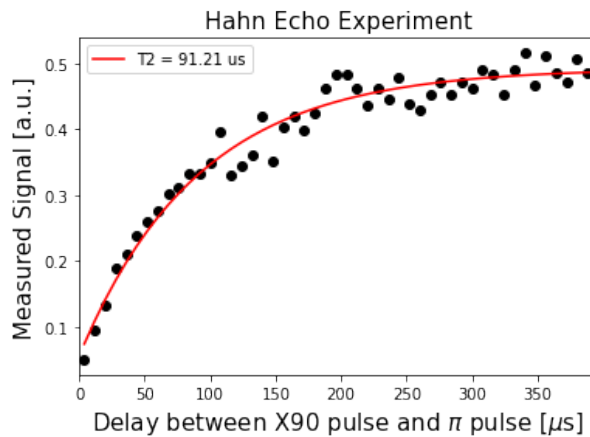
O experimento para a obtenção de T_1 em IBM (2022a) consiste em diversos experimentos aplicando uma porta X ao estado inicial $|0\rangle$ e realizando a medição, mas antes de realizar a medição, é esperado um determinado tempo em cada experimento. Dessa forma, quanto maior for o tempo de espera antes da medição, mais próximo o qubit estará do estado $|0\rangle$. A figura 3.10 mostra um exemplo de resultado desse experimento, onde o eixo x representa o tempo de espera antes da medição e o eixo y , o sinal medido. Nesse caso, o tempo que levou para o estado decair de $|1\rangle$ para $|0\rangle$ foi de $113,11\mu s$.

Figura 3.10 – Experimento para obtenção de T_1 

Fonte: (IBM, 2022a)

3.9 O TEMPO T_2

De forma semelhante e relacionada ao tempo T_1 , o tempo T_2 , ou tempo de decoerência, corresponde ao tempo em que a informação de fase do sistema quântico se mantém coesa. Geralmente, esse número é menor do que o T_1 , e também pode ser obtido através de um experimento. Segundo IBM (2022a), o tempo de decoerência pode ser obtido realizando vários experimentos onde é aplicada uma sequência de um pulso $\pi/2$, um pulso π e mais um pulso $\pi/2$ com um determinado intervalo de tempo entre a aplicação de cada pulso. A figura 3.11 mostra um resultado esperado para esse experimento, onde o eixo x representa, para cada experimento, os intervalos de tempo entre a aplicação de cada pulso e o eixo y representa o sinal medido. Nesse caso, o tempo T_2 foi de 91,21 μs .

Figura 3.11 – Experimento para obtenção de T_2 

Fonte: (IBM, 2022a)

3.10 PRINCIPAIS FONTES DE ERROS

Além dos tempos de decoerência T_1 e T_2 , existem várias outras fontes de ruídos que influenciam os resultados da computação. As principais fontes estão relacionadas com os controles individuais das portas lógicas, que nem sempre estão adaptados às condições do ambiente e do momento em que são aplicadas nos qubits, mas também há erros causados pela própria construção dos qubits e erros nas medidas dos estados quânticos. Esta seção apresentará alguns conceitos sobre as principais fontes de erros em processos de computação quântica.

O erro de *dephasing* está intimamente relacionado com o tempo T_2 . Segundo O'Malley (2016), ele basicamente descreve a perda da coerência da fase do estado quântico, na esfera de Bloch, é como se o vetor de estado encolhesse com relação ao equador da esfera.

A aplicação das portas lógicas é uma das principais fontes de erro. Existem diferentes maneiras de descrever uma porta lógica que muda o estado quântico de uma forma específica, os valores de amplitude dos pulsos podem não ser aplicados exatamente no valor desejado ou os resultados do experimento de Rabi podem não ser suficientemente precisos. Os erros de controle de amplitude (CARVALHO *et al.*, 2021) fazem com que o vetor estado não se comporte como deveria e haja uma perda de precisão na aplicação da porta lógica.

Durante a aplicação das operações, além dos erros de controle de amplitude, também podem ocorrer erros de vazamento de elétrons, conhecidos como leakage. Esse erro acontece quando o sistema quântico de dois níveis ($|0\rangle$ e $|1\rangle$) se torna um sistema quântico de vários níveis, com possíveis estados $|2\rangle$, $|3\rangle$, ... (CHEN *et al.*, 2016). Foi visto anteriormente que o regime Transmon faz com que a diferença de energia entre os níveis seja irregular, permitindo o isolamento de dois níveis de energia para modelar qubits, entretanto, ainda assim existe a chance de alguns elétrons subirem para níveis de energia superiores durante a aplicação dos pulsos, principalmente os de amplitudes mais elevadas (JO; SONG; AHN, 2019).

Computadores quânticos compostos de apenas um qubit não possuem grandes vantagens sobre os computadores clássicos, o ganho na representatividade da informação é exponencial sobre a quantidade de qubits do sistema. Além disso, os diferentes sistemas quânticos que implementam os qubits precisam de determinadas conexões para que o entrelaçamento possa ser utilizado para implementar portas lógicas de vários qubits. Essas conexões podem induzir erros de crosstalk. Segundo Sarovar *et al.* (2020), um sistema livre de crosstalk é um sistema cujo comportamento na implementação de circuitos arbitrários satisfaz as propriedades de localidade e independência, ou seja, cujas operações aplicadas individualmente a determinados qubits não possuem interferência umas nas outras. Dessa forma, erros de crosstalk acontecem quando os pulsos que implementam as portas lógicas são aplicados em seus qubits alvos e acabam interferindo em qubits indesejados.

Além dos erros das operações do circuito, é bastante comum a ocorrência de erros nas medidas dos estados quânticos. A seção 3.7 apresentou uma forma de calibrar a discriminação entre os estados 0 e 1 nas medidas dos estados quânticos dos qubits, erros de medidas estão relacionados com a classificação errônea de um estado 0 como estado 1 ou vice-versa.

4 ESTADO DA ARTE

Para contextualizar os estudos relacionados com a proposta deste trabalho, uma revisão bibliográfica sistemática foi realizada com base nas ferramentas de pesquisa online IEEE Explorer, Science Direct e Google Scholar utilizando as palavras-chave Quantum Computing, Quantum Control, Quantum Error Correction e Superconducting Quantum Computing e pesquisando por artigos de 2015 em diante. Os resultados das combinações possíveis entre as palavras-chave podem ser visualizados na tabela 1.

Com base em uma análise mais detalhada dos títulos e resumos dos artigos encontrados na revisão, foram selecionados 10 trabalhos correlatos que melhor contemplaram o estado da arte de mitigação de erros para serem apresentados nesta seção. Após um estudo dos artigos encontrados, foi construída uma tabela comparativa que está apresentada na tabela 2. Essa tabela auxilia na comparação das características de cada proposta para uma melhor análise do tema do estado da arte relacionados à proposta deste trabalho.

Mitigação de erros em computadores quântico é uma tarefa que pode ser feita de várias formas. Existem técnicas que utilizam a compilação do circuito para otimizar a aplicação das portas lógicas quânticas, como apresentado por Murali *et al.* (2019) (artigo 8 na tabela 2), que busca realizar otimizações agressivas para a linguagem de programação quântica Scaffold, formulando um problema de otimização baseado em restrições que considera tanto as características do programa quando do hardware.

Ainda, existem técnicas baseadas em Quantum Error Correction (QEC) codes, que, segundo Calderbank e Shor (1996), podem ser definidas como um mapeamento de k qubits para n qubits de tal forma que se qualquer t qubits sejam afetados pela decoerência, os n qubits têm a capacidade de reconstruir o estado original dos k qubits. Postler *et al.* (2022), McEwen *et al.* (2021), Nautrup *et al.* (2019), Gertler *et al.* (2021), Fu *et al.* (2019) e Guo *et al.* (2021) (artigos 3 a 7 e 10 na tabela 2) apresentam soluções relacionadas com esse tema. O artigo 3, melhor detalhado na seção 4.3, mostra um conjunto universal de portas lógicas quânticas que utilizam essa redundância de qubits para serem tolerantes à falhas, sendo um trabalho importante para a área. O artigo 10 propõe um código de QEC de 2 qubits a ser implementado em diversas plataformas com o objetivo de comparar seu desempenho em cada uma delas.

Sob outra perspectiva, os artigos 5 e 6 apresentam técnicas para melhorar o desempenho de código de QEC. O artigo 5, detalhado na seção 4.5, apresenta um método de aprendizado por reforço para otimizar a tolerância à falhas de códigos de QEC. O artigo 6 também segue essa ideia, propondo um método autônomo capaz de dobrar o tempo de coerência do sistema.

A programação de computadores quânticos requer arquiteturas que façam o intermédio entre software e hardware. Nesse sentido, o artigo 7 apresenta uma microarquitetura que suporta a computação quântica tolerante à falhas de forma eficiente, destacando um

mecanismo que permite a compilação de um grande número de qubits e um suporte arquitetural para QEC em tempo de execução, reduzindo o tamanho do código do programa e apresentando uma boa escalabilidade.

Tabela 1 – Número de Resultados por Combinação de Palavra-chave.

Palavras-chave	IEEE	Science Direct	Scholar
Quantum Computing	7 667	61 081	76 400
Quantum Control	5 810	163 404	1 230 000
Quantum Error Correction	640	18 403	34 800
Superconducting Quantum Computing	348	3 261	17 400
Quantum Computing, Quantum Control	1 372	34 668	25 800
Quantum Computing, Quantum Error Correction	328	8 711	17 200
Quantum Computing, Superconducting Quantum Computing	348	3 261	17 200
Quantum Control, Quantum Error Correction	77	10 991	18 800
Quantum Control, Superconducting Quantum Computing	84	2 115	17 000
Quantum Error Correction, Superconducting Quantum Computing	29	656	17 600
Quantum Computing, Quantum Control, Quantum Error Correction	48	5 098	17 000
Quantum Computing, Quantum Control, Superconducting Quantum Computing	84	2 115	16 800
Quantum Control, Quantum Error Correction, Superconducting Quantum Computing	7	527	16 800
Quantum Computing, Quantum Control, Quantum Error Correction, Superconducting Quantum Computing	7	527	17 700

Tabela 2 – Comparação de Trabalhos Relacionados.

Nº	Artigo	Tipo de Mitigação	Método de Mitigação	Erros Considerados	Faz experimentos em hardware quântico	Plataforma de Experimentação
1	Error-Robust Quantum Logic Optimization Using a Cloud Quantum Computer Interface. (CARVALHO et al., 2021)	Quantum Control	Numérica e Protocolo	Dephasing, controle de amplitude dos pulsos e crosstalk	X	Supercondutores IBMQ
2	Experimental Deep Reinforcement Learning for Error-Robust Gateset Design on a Superconducting Quantum Computer. (BAUM et al., 2021)	Quantum Control	DRL	Quaisquer	X	Supercondutores IBMQ
3	Demonstration of Fault-Tolerant Universal Quantum Gate Operations. (POSTLER et al., 2022)	QEC	Numérica	Quaisquer	X	Computador quântico de armadilha de íons
4	Removing Leakage-Induced Correlated Errors in Superconducting Quantum Error Correction. (MCEWEN et al., 2021)	QEC e Quantum Control	Protocolo	Vazamento de Elétrons	X	Supercondutor Google Sycamore
5	Optimizing Quantum Error Correction Codes with Reinforcement Learning. (NAUTRUP et al., 2019)	QEC	RL	Quaisquer		Simulações
6	Protecting a Bosonic Qubit with Autonomous Quantum Error Correction. (GERTLER et al., 2021)	QEC e Quantum Control	Numérica e Protocolo	Quaisquer	X	Circuito QED híbrido
7	A Control Microarchitecture for Fault-Tolerant Quantum Computing. (FU et al., 2019)	QEC e Quantum Control	-	-		-
8	Formal Constraint-Based Compilation for Noisy Intermediate-Scale Quantum Systems. (MURALI et al., 2019)	Compilação do Circuito	Numérica	Decoerência	X	Supercondutores IBMQ
9	Learning in Quantum Control: High-Dimensional Global Optimization for Noisy Quantum Dynamics. (PALITTAPONGARNPIM et al., 2017)	Quantum Control	ML	Quaisquer		-
10	Testing a Quantum Error-Correcting Code on Various Platforms. (GUO et al., 2021)	QEC	Protocolo	Quaisquer	X	Plataforma ótica, supercondutores IBMQ e ressonância magnética nuclear

Além de otimizações no circuito e código de QEC, o controle quântico é fundamental para a mitigação de erros. Carvalho *et al.* (2021), Baum *et al.* (2021), McEwen *et al.* (2021), Gertler *et al.* (2021), Fu *et al.* (2019) e Palittapongarnpim *et al.* (2017) (artigos 1, 2, 4, 6, 7 e 9 da tabela 2) apresentam soluções que utilizam o controle quântico do hardware. Os artigos 1 e 2 se referem a produtos comerciais da Q-CTRL (Q-CTRL, 2022a) que serão utilizados nas experimentações deste trabalho, as seções 4.1 e 4.2 mostram mais detalhes desses artigos. A forma com que esses trabalhos tratam a questão do controle quântico é

bastante útil para aplicações práticas, o artigo 1 mostra uma interface para a construção de portas lógicas quânticas que contornam problemas com ruídos, isso é feito com base em otimizações a partir de uma caracterização do ambiente. O artigo 2 já mostra um método autônomo de calibração do hardware quântico que não necessita do conhecimento de modelos específicos do sistema, seus controles ou processos de erros.

De fato, o uso de técnicas de Machine Learning (ML) para mitigação de erros em processos de computação quântica tem se mostrado bastante eficiente, dado que muitas vezes pode ser difícil caracterizar os diversos erros que podem ocorrer para realizar otimizações numéricas específicas para cada um deles, além do fato de que eles podem estar correlacionados, dificultando ainda mais essa tarefa. Dessa forma, o uso de ML não necessita de profundas caracterizações dos processos de erros e consegue manter a eficiência de forma independente disso, como mostra os artigos 2, 5 e 9.

Com relação aos tipos de erros considerados em cada artigo, a maioria deles propõe métodos que mitiguem qualquer erro, que é principalmente os casos de métodos baseados em ML. Códigos de QEC também são úteis para mitigação de erros de maneira geral. Entretanto, o controle quântico geralmente é mais focado em caracterizações do hardware e dos processos de erros, como é possível ver nos artigos 1 e 4 da tabela 2. O artigo 1 considera erros de *dephasing*, controle de amplitude dos pulsos e crosstalk, já o artigo 4 utiliza um protocolo para evitar erros de vazamentos de elétrons em outros níveis de energia considerando o regime de transmon de qubits de supercondutores. O artigo 8, que explora a compilação do circuito, está interessado nos erros de decoerência, já que sua abordagem tem o poder de otimizar o circuito e reduzir o tempo de execução de forma a contornar esse problema.

O tipo de hardware quântico também possui uma considerável influência sobre os erros. O artigo 10 mostra um mesmo código de QEC implementado em plataformas óticas, de supercondutores e ressonância magnética nuclear, comparando as diferentes abordagens necessárias para cada uma delas. Computadores quânticos de supercondutores dominam grande parte das máquinas disponíveis para acesso via Cloud, os artigos 1, 2, 4 e 8 focam no uso desse tipo de computador e 1, 2 e 8 utilizam os computadores quânticos da IBMQ (IBM, 2022b). O artigo 3 utiliza um computador quântico baseado em armadilha de íons para seus experimentos, que também é bastante eficiente.

Os cinco primeiros artigos da tabela 2 são considerados os mais importantes para o estabelecimento da proposta deste trabalho e serão melhor detalhados nas seções a seguir. Os experimentos a serem realizados considerarão as abordagens dos artigos das seções 4.1 e 4.2. Os demais artigos são considerados bastante relevantes à proposta e ao tema do estado da arte e também serão descritos nas próximas seções.

4.1 OTIMIZAÇÃO DE PORTAS LÓGICAS QUÂNTICAS ROBUSTAS À RUÍDOS UTILIZANDO UMA INTERFACE CLOUD

O trabalho proposto em Carvalho *et al.* (2021) possui uma relação direta com este trabalho, ele apresenta uma ferramenta Cloud para geração de portas lógicas de 1 qubit resistente a erros que será utilizada posteriormente. Com base em uma modelagem de qubits em grafo e otimizações feitas utilizando uma versão customizada da biblioteca Tensorflow, os pulsos gerados pelo sistema são resistentes a diferentes erros, incluindo *dephasing*, instabilidades nos controles dos pulsos e crosstalk. Também é investigada a relação temporal e espacial dos erros considerando que os tempos T_1 e T_2 variam diariamente.

Para a realização da otimização que corrige os ruídos do sistema, é definido um hamiltoniano composto por três termos:

$$H_{tot}(t) = H_{ctrl}(t) + H_{leakage} + H_{noise}(t)$$

onde $H_{ctrl}(t)$ representa o hamiltoniano que controla o sistema quântico, $H_{leakage}$ representa vazamento de elétrons para outros estados de energia e $H_{noise}(t)$ se refere aos ruídos do sistema. $H_{noise}(t)$ ainda é composto de outros dois termos, um para descrever erros de *dephasing*, $H_{depth}(t)$ e outro para descrever erros de controle de amplitude dos pulsos, $H_{amp}(t)$.

Com base nos hamiltonianos e parâmetros oferecidos pelos computadores quânticos da IBM, o framework realiza otimizações utilizando uma variação da biblioteca Tensorflow que permite a representação dos termos do hamiltoniano como funções quase arbitrárias dos parâmetros controlados. Essa representação utiliza grafos que podem ser utilizados de maneira bastante eficiente para o cálculo da função custo da otimização.

Para a implementação do hamiltoniano utilizando o Qiskit Pulse, é feito um mapeamento entre os termos do hamiltoniano que definem o controle do sistema para as amplitudes dos pulsos baseados no experimento de Rabi. A partir disso, são feitas calibrações do pulso parametrizando esse mapeamento com o objetivo de corrigir pequenos defeitos do experimento de Rabi.

A validação da robustez dos pulsos se dá por meio de experimentos que comparam as portas lógicas padrão de computadores quânticos da IBM, pulsos que mitigam erros de amplitude, *dephasing* e ambos. Para cada uma das portas lógicas geradas, são feitos experimentos causando erros de *dephasing*, erros de amplitude e repetindo o pulso por uma grande quantidade de vezes. Como esperado, as portas lógicas padrão da IBM possuem uma infidelidade consideravelmente alta com relação aos pulsos gerados pelo framework quando submetidos a erros ou uma grande quantidade de repetições.

Os experimentos mostraram que a performance desses pulsos tem melhorias de performance de: redução de erros de decoerência de aproximadamente 10 vezes em sistemas de 1 qubit e 5 vezes em sistemas de 5 qubits; redução de 12 vezes na variabilidade dos

erros dos pulsos com o tempo e até 9 vezes de redução nos erros de decoerência, incluindo crosstalk, para operações completamente parametrizadas.

4.2 UMA ABORDAGEM DE APRENDIZAGEM PROFUNDA POR REFORÇO PARA A CONSTRUÇÃO DE PORTAS LÓGICAS ROBUSTAS À RUÍDOS EM UM COMPUTADOR QUÂNTICO SUPERCONDUTOR

Grande parte das investigações acerca do design de portas lógicas que mitigam erros de decoerência utilizam técnicas se baseiam em modelos físicos precisos e detalhados que geralmente só são eficientes para operações simples e genéricas. Baum *et al.* (2021) apresenta um modelo capaz de gerar um conjunto de portas lógicas universais para computadores quânticos de supercondutores que são resistentes a ruídos. Esse modelo utiliza uma técnica de aprendizado por reforço profundo (DRL) que não necessita do conhecimento de um hamiltoniano, controles ou caracterização dos erros.

A técnica de DRL utilizada constrói um modelo para descrever os efeitos considerados relevantes de forma iterativa e autônoma baseando-se em um conjunto de controles disponíveis pelo hardware quântico. O sistema é descrito por um controlador hamiltoniano semelhante ao do trabalho descrito na seção 4.1 dado por $H_{ctrl}(t) \left[\left\{ \Omega_{I,j}^\omega(t), \Omega_{Q,j}^\omega(t) \right\} \right]$, onde os pulsos de micro-ondas são descritos pelos sinais $\Omega_{I/Q,j}^\omega(t)$ aplicados na frequência ω sobre os qubits j . O objetivo da otimização é fazer com que esse controlador faça a evolução do sistema se aproximar ao máximo do esperado para um determinado objetivo.

A cada passo do ciclo do DRL, os pulsos são recalibrados e testados no computador quântico, onde são feitas medidas utilizando técnicas básicas de mitigação de erros. Para estimar a qualidade da porta lógica criada, é utilizado um mecanismo de recompensa baseado em uma média ponderada das fidelidades para sequências de diferentes números de repetições dos pulsos.

O artigo realiza experimentos para a criação de portas lógicas de 1 e 2 qubits. A porta de 1 qubit escolhida para os testes foi a $Rx(\pi/2)$, que pode ter sua duração reduzida e robustez contra erros em até 3 vezes com relação aos pulsos padrão da IBM do tipo DRAG. A porta de 2 qubits $ZX(-\pi/2)$ melhorou a fidelidade em aproximadamente 2,68 vezes. Outro ponto importante é que a performance dos pulsos se manteve adequada mesmo após semanas, não sendo necessário recalibrá-los com tanta frequência. Também são realizados testes de benchmark que mostram que o uso de DRL para criação de portas lógicas pode ter desempenho equiparável ou levemente superior do que outras técnicas de caixa preta, mesmo quando o acesso ao hardware é limitado.

4.3 UMA DEMONSTRAÇÃO DE UM CONJUNTO UNIVERSAL DE PORTAS LÓGICAS QUÂNTICAS TOLERANTE À FALHAS

Uma alternativa para mitigar erros em processos de computação quântica com relação ao controle quântico é a utilização de ferramentas derivadas da computação quântica tolerante a falhas. A ideia geral é desenvolver códigos de correção de erros quânticos que desfrutem da redundância de qubits físicos para formar qubits lógicos mais precisos. Todavia, essas técnicas implicam na utilização de um design específico de circuito tolerante a falhas, o que dificulta a implementação. Nesse sentido, Postler *et al.* (2022) apresenta um conjunto de portas lógicas universais tolerantes a falhas para computadores quânticos baseados em armadilha de íons.

A abordagem consiste na utilização de conceitos de *flag fault tolerance*, que se baseia no uso de qubits auxiliares para indicar a presença ou ausência de erros considerados graves. O código utiliza uma codificação de sete qubits para representar um qubit lógico. O conjunto de portas lógicas universais é composto de portas Clifford e uma T, enquanto as portas Clifford são transversais, ou seja, existe uma ação individual sobre os qubits que a implementam, a porta T é feita com a injeção de um estado mágico via teleporte quântico.

O conjunto de portas lógicas universais pode ser usado para implementar qualquer operação unitária. Sendo assim, esse trabalho se trata da primeira implementação de um conjunto universal de portas lógicas quânticas tolerantes a falhas, melhorando consideravelmente a performance de qubits codificados. Parte da eficiência da abordagem se dá graças a grande conectividade presente nas arquiteturas baseadas em armadilha de íons, permitindo um melhor entrelaçamento entre pares de qubits.

4.4 REMOVENDO ERROS DE VAZAMENTO DE ELÉTRONS DE SUPERCONDUTORES EM QUANTUM ERROR CORRECTION

Outro problema que ocorre em computadores quânticos, principalmente computadores quânticos de supercondutores, é a excitação de estados de energia superiores dos qubits. Particularmente em qubits *transmon*, esse vazamento de elétrons faz com que ocorram erros correlacionados no tempo e espaço. McEwen *et al.* (2021) apresenta um protocolo que redefine os qubits para o estado fundamental em estados de energia superiores que sejam relevantes e o testa utilizando um código básico de bit-flip, também são investigadas as dinâmicas e acumulações desses erros durante a correção.

A abordagem utiliza uma porta lógica *swap* adiabática entre o qubit e o ressonador de leitura para redefinir os estados para o estado fundamental e ademais, são utilizados apenas recursos do hardware para operações normais e leituras, não necessitando de controles por meio de pulsos de micro-ondas que poderiam ocasionar erros de crosstalk, sendo mais atrativo para sistemas em larga escala.

O processo de mitigação é feito por meio de três passos denominados *swap*, *hold* e *return*. O *swap* consiste em varrer a frequência do qubit de forma adiabática até aproximadamente $1GHz$ abaixo do ressonador. Então, na etapa *hold*, o qubit é mantido abaixo do ressonador enquanto as excitações decaem. Por fim, o passo *return* faz o qubit voltar para a sua frequência inicial de forma adiabática.

Os experimentos de validação utilizam um código de correção de bit-flip e medição do crescimento e remoção do vazamento de elétrons. Os resultados mostraram que o protocolo é capaz de mitigar erros de vazamento correlacionados induzidos por um tempo considerável e melhora de forma substancial a atenuação de erros lógicos. O artigo também afirma que otimizar as portas lógicas e as leituras é necessário para prevenir esse tipo de erro, entretanto, sua natureza correlacionada faz com que protocolos de *reset* como o apresentado sejam críticos para a mitigação de erros na prática.

4.5 OTIMIZANDO CÓDIGOS DE CORREÇÃO DE ERROS COM APRENDIZADO POR REFORÇO

Os códigos de correção de erros têm sido uma boa solução para a computação quântica tolerante a falhas, entretanto, pode ser difícil codificar erros desconhecidos que aparecem nos experimentos. Nautrup *et al.* (2019) apresenta um método de aprendizado por reforço (RL) que é capaz de melhorar o desempenho de códigos de correção de erros. Considerando um cenário onde um determinado código de correção de erros é implementado em um ambiente suscetível a ruídos arbitrários e dada a capacidade de estimar a taxa de erros lógicos, o objetivo do framework é oferecer um esquema automatizado que determina o melhor código que alcança uma taxa de erros abaixo de um certo limite.

O algoritmo de aprendizado interage com o ambiente modificando o código de correção de erros baseado em um feedback dado pelo ambiente. A cada iteração, o agente recebe uma informação sobre a estrutura atual do código e recebe como tarefa modificar o código com o objetivo de atingir uma determinada taxa de erros lógicos, para isso, ele tem a permissão de realizar certas ações no código. O agente é recompensado se os qubits lógicos foram suficientemente protegidos de erros.

Um ponto importante acerca do uso de RL no trabalho é que o agente é capaz de obter bons resultados em diversas situações, como com diferentes limiares e diferentes modelos de ruídos, e em diferentes implementações físicas de computadores quânticos, como supercondutores ou armadilha de íons. A principal razão para isso se deve ao fato de que o agente proposto não obtém nenhuma informação sobre qualquer modelo de ruídos do sistema, funcionando como uma caixa preta.

Os resultados mostraram que o agente foi capaz de proteger o sistema tanto contra vários ruídos mais simples quanto contra ruídos mais complexos que são correlacionados e difíceis de detectar. Uma grande vantagem do framework é a habilidade de transferir a experiência de um modelo de ruídos para outro completamente diferente, sugerindo que

esse tipo de técnica poderia ser utilizada de maneira adaptativa.

5 EXPERIMENTAÇÃO

Os experimentos para avaliar o comportamento de diferentes pulsos consideram a implementação de um autômato finito quântico (QFA). A teoria de Autômatos Finitos corresponde ao modelo mais simples da computação clássica, eles são capazes de reconhecer a classe das Linguagens Regulares em tempo linear, enquanto isso, os QFAs são mais poderosos e possuem a capacidade de reconhecer algumas linguagens mais abrangentes pertencentes à classe das Linguagens Livres-de-Contexto, destacando o poder dos modelos de máquinas quânticas.

A simplicidade dos QFAs é particularmente interessante para analisar e avaliar a ocorrência de erros na computação, que normalmente envolve operações específicas que se repetem muitas vezes, podendo deixar os circuitos muito longos, o que se faz necessário diminuir a duração das portas lógicas. Os experimentos desta seção buscam analisar a suscetibilidade a erros dos QFAs quando implementados nas plataformas da IBMQ, principalmente, na computação de palavras longas, também, considerando a ocorrência de falsos positivos na classificação das palavras.

A abordagem considera uma implementação do autômato MO1QFA (MOORE; CRUTCHFIELD, 2000), apresentado na seção 5.1, que resolve o problema MOD^p introduzido na seção 5.1.1 para $p = 11$. A implementação utilizará a plataforma IBMQ como um ambiente real de computação quântica para avaliar sua performance como plataforma para a implementação de QFAs considerando o controle quântico dos erros.

Foram utilizados diferentes computadores quânticos para os experimentos, com foco maior no *ibm_nairobi*, que é composto de 7 qubits com $T_1 = 127, 23\mu s$ e $T_2 = 94, 44\mu s$, os computadores *ibm_oslo* (7 qubits, $T_1 = 154, 18\mu s$ e $T_2 = 85, 84\mu s$) e *ibmq_belem* (5 qubits, $T_1 = 102, 19\mu s$ e $T_2 = 108, 27\mu s$) também foram utilizados de maneira a complementar alguns resultados.

Para a implementação do MO1QFA, serão consideradas palavras de tamanho congruente a 0 e a 3 módulo 11 de tamanho em um intervalo de 0 a 1000, totalizando 182 valores diferentes. Com esses valores, é esperado uma probabilidade de aceitação de 100% para palavras congruente a 0 módulo 11 e 2,0254% para palavras congruentes a 3 módulo 11.

Com o objetivo de otimizar os circuitos que executam nos computadores da IBMQ, o Qiskit implementa quatro níveis de otimização, de 0 a 3, quanto maior o número, mais otimizado será o circuito, ao custo de um tempo maior de transpilação (QISKIT, s.d.). O nível de otimização 0 apenas mapeia o circuito para o backend considerando as portas lógicas disponíveis nele, não realizando otimizações. O nível de otimização 1 faz pequenas otimizações colapsando portas lógicas adjacentes. Os níveis de otimização 2 e 3 fazem otimizações mais profundas considerando as relações entre as diferentes portas lógicas. É importante ressaltar que essas otimizações são apenas a nível de circuito, não alterando

as implementações de portas lógicas existentes.

Nesse sentido, as ferramentas do Q-CTRL Boulder Opal serão utilizadas como alternativas às portas lógicas padrão do Qiskit. Dessa forma, o hardware será calibrado de forma a melhorar a performance das portas lógicas. Os experimentos desta seção buscam mostrar a influência dos erros em implementações de autômatos finitos quânticos, mostrando, também, diferentes alternativas para otimizar os circuitos. Portanto, serão feitos experimentos utilizando ambos os níveis de otimização 0 e 1 do Qiskit, utilizando diferentes portas lógicas resistentes a diferentes tipos de erros e utilizando diferentes calibrações oferecidas pelo Q-CTRL Boulder Opal.

A seção 5.1 apresentará o autômato utilizado para os experimentos, tal como um problema solucionável pelo autômato, mapeamento circuital e caracterização dos erros. A seção 5.2 introduz os resultados de uma implementação básica com otimizações circuitais do Qiskit. A seção 5.3 mostra os resultados puros dos pulsos implementados pelo Qiskit no computador quântico. A seção 5.4 inicia a investigação do controle quântico, mostrando a implementação de um pulso simples ultrarrápido. A seção 5.5 mostra como calibrar uma porta lógica mais precisamente, e a influência disso na qualidade da computação. A seção 5.6 busca avaliar a interferência da efetiva caracterização da frequência do qubit em hardware. Por fim, a seção 5.7 está interessada em avaliar a robustez de diferentes pulsos com relação aos erros de *dephasing*.

5.1 MEASURE-ONCE 1-WAY QUANTUM FINITE AUTOMATA

O Measure-Once 1-way Quantum Finite Automata (MO1QFA) é um autômato finito quântico proposto por Moore e Crutchfield (2000). As principais características desse autômato são:

- As transições são feitas por meio de operações unitárias;
- O estado quântico é medido apenas uma vez, no final da execução;
- A cabeça de leitura do autômato tem direção única, ou seja, se movimenta da esquerda para a direita, lendo um símbolo em cada transição.
- No começo da computação, a cabeça de leitura está no primeiro símbolo da fita de leitura.

Dada essas restrições, o MO1QFA é considerado o modelo mais elementar de autômato finito quântico, mas também é o modelo menos expressivo em comparação com outros modelos de autômatos quânticos. A definição formal do MO1QFA é dada por:

MO1QFA é uma 5-tupla $M = (Q, \Sigma, \delta, |q_0\rangle, F)$ onde:

$Q = \{|q_0\rangle, \dots, |q_n\rangle\}$ é um conjunto finito de estados - $Q \subseteq \mathcal{H}$ onde \mathcal{H} é um espaço de Hilbert;

$\Sigma = \{\sigma_0, \dots, \sigma_k\}$ é um conjunto finito dos símbolos de entrada (alfabeto de entrada);

δ é o conjunto de transformações unitárias U_σ , onde cada U_σ descreve uma transição de estado de acordo com σ ;

$|q_0\rangle \in Q$ é o estado inicial do autômato;

$F \subseteq Q$ é o conjunto de estados finais.

O processo de computação de uma palavra w no MO1QFA é dado por:

$$f(w) = \|P_{acc}U_w |q_0\rangle\|^2,$$

onde U_w significa a aplicação da respectiva matriz unitária para cada símbolo w_i de w :

$$U = U_{w_{|w|-1}} \dots U_{w_1} U_{w_0},$$

P_{acc} é o operador projeção dos estados finais:

$$P_{acc} = \sum_{q_i \in F} |q_i\rangle\langle q_i|.$$

e $f(w)$ representa a probabilidade de M aceitar w .

5.1.1 Problema MOD^p

Para exemplificar o uso do MO1QFA para o tratamento de uma linguagem, será considerado o problema do módulo de um número primo. Essa linguagem pode ser modelada por meio de dois estados, $|0\rangle$ e $|1\rangle$, onde ambas utilizam a superposição quântica para rotacionar o vetor de estado em torno do eixo y , utilizando a seguinte transformação unitária:

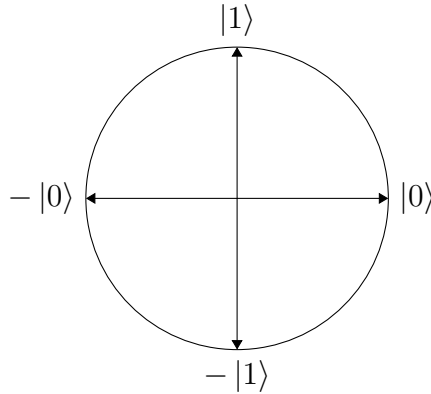
$$U_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}. \quad (3)$$

A superposição de estados do autômato pode ser visualizada como uma derivação da esfera de Bloch com duas dimensões, apresentada na figura 5.1. Com relação à matriz de transição U_θ , valores positivos de θ resultam em rotações no sentido anti-horário do vetor de estado, enquanto valores negativos, no sentido horário.

Say e Yakaryilmaz (2014) propuseram que, com o MO1QFA, é possível resolver problemas com erros limitados utilizando menos estados em comparação com soluções de autômatos clássicos. Para demonstrar isso, eles utilizaram o problema do módulo de números primos:

$$MOD^p = \{a^{j^p} | j \text{ é um inteiro não-negativo}\}$$

Figura 5.1 – Representação bidimensional do qubit para o problema MOD^p .



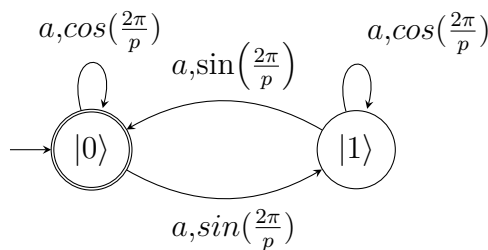
O respectivo autômato quântico é composto de dois estados e reconhece palavras congruentes a 0 módulo p em 100% dos casos, entretanto, permite falsos positivos, com o maior valor de erro encontrado em palavras que se aproximam de $p/2$ módulo p , fazendo com que o erro se aproxime de:

$$\cos^2\left(\frac{\pi}{p}\right).$$

Para realizar a computação desse autômato, é necessário dividir o qubit bidimensional da figura 5.1 em p partes, onde cada fatia representa a congruência possível dos valores $0, 1, 2, \dots, p - 1$. Para esse algoritmo, a matriz que representa a rotação é dada pela equação 4, que divide o círculo em p partes e rotaciona o vetor de estado de acordo com:

$$A = \begin{bmatrix} \cos\left(\frac{2\pi}{p}\right) & \sin\left(\frac{2\pi}{p}\right) \\ \sin\left(\frac{2\pi}{p}\right) & \cos\left(\frac{2\pi}{p}\right) \end{bmatrix}. \quad (4)$$

Figura 5.2 – Diagrama do autômato MO1QFA para o problema MOD^p .



A figura 5.2 descreve visualmente o autômato. É importante notar que o autômato apresentado pode ser modificado facilmente para reconhecer o módulo de qualquer primo. Para fazer isso de maneira clássica, p estados seriam necessários, enquanto apenas 2 são necessários utilizando o MO1QFA. Além disso, conforme p aumenta e o tamanho da palavra se aproxima de valores congruentes a $p/2$ módulo p , o erro do autômato aumenta

rapidamente, permitindo que esses valores sejam aceitos de forma incorreta por mais de 80% nos primos maiores do que 7.

5.1.2 Mapeamento Circuital do MO1QFA

A IBMQ utiliza o modelo circuital de computação, onde a computação é descrita como uma série de portas lógicas quânticas que representam determinadas rotações no vetor estado do qubit. Assim, primeiramente é necessário mapear o MO1QFA para seguir esse modelo.

O modelo apresentado na seção 5.1.1 possui apenas 2 estados, conseqüentemente, a implementação necessita de apenas um qubit. As transições podem ser feitas tanto por meio de portas lógicas $R_x(\theta)$, que realizam rotações θ em torno do eixo x , quanto por portas $R_y(\theta)$, que fazem o mesmo sobre o eixo y , já que nesse caso elas são simétricas. Entretanto, considerando a descrição da porta R_x com relação à matriz de rotação da equação 3, é necessário utilizar portas lógicas R_x com rotação de 2θ , com θ sendo igual a $4\pi/11$ de acordo com a descrição da matriz de transição do autômato (equação 4). Ou seja, o mapeamento do MO1QFA para o modelo circuital do Qiskit é dado por uma sequência de portas lógicas $R_x(4\pi/11)$ que descrevem rotações aplicadas pelas transições obtidas durante a execução de uma entrada no autômato.

Portanto, considerando os níveis de otimização utilizados nos experimentos desta seção, o nível de otimização 0 deve aplicar cada uma das portas R_x ao qubit, assim, quanto mais longa for a palavra, maior será a acumulação do erro no estado do qubit. Enquanto isso, o nível de otimização 1 deve aplicar uma porta R_x otimizada com a respectiva rotação acumulada, dado que esse nível colapsa portas lógicas adjacentes, dessa forma, é esperado que o tamanho da palavra não influencie na acumulação do erro.

5.1.3 Caracterização do Erro

As próximas seções mostrarão que as implementações dos autômatos podem ser fortemente suscetíveis a erros do próprio hardware quântico. Para um melhor entendimento dos fatores que compõe os erros durante a computação do autômato, variações da função apresentada podem ser definidas para caracterizar as diferenças entre os valores obtidos e esperados. Essa caracterização do erro auxilia na compreensão de sua influência na computação do autômato e na busca de alternativas para mitigá-los.

O erro na relação entre a probabilidade de aceitação de w pode ser definido por três fatores: α (erro de amplitude), δ (erro de fase) e β (ruído). Vale ressaltar que essa é uma definição do erro sobre a função que descreve as probabilidades de aceitação do autômato, portanto, não está relacionada com qualquer caracterização do erro no hardware em si:

$$\alpha \times \cos^2\left(\frac{2\pi}{11}|w| + \delta\right) + \beta, \quad (5)$$

quando $\alpha = 1$, $\delta = 0$ e $\beta = 0$ não há erros.

5.2 EXPERIMENTOS COM NÍVEL DE OTIMIZAÇÃO 1

Os primeiros experimentos consideram o nível de otimização 1 do Qiskit na execução do autômato. As figuras 5.3 e 5.4 mostram uma comparação entre os valores obtidos e esperados de probabilidades de aceitação para palavras de tamanho congruente a 0 e a 3, respectivamente, módulo 11 de 0 a 1000, e as tabelas resumem os erros absolutos entre os valores obtidos e o esperados. Os mesmos experimentos foram executados nos computadores quânticos Nairobi, Oslo e Belém.

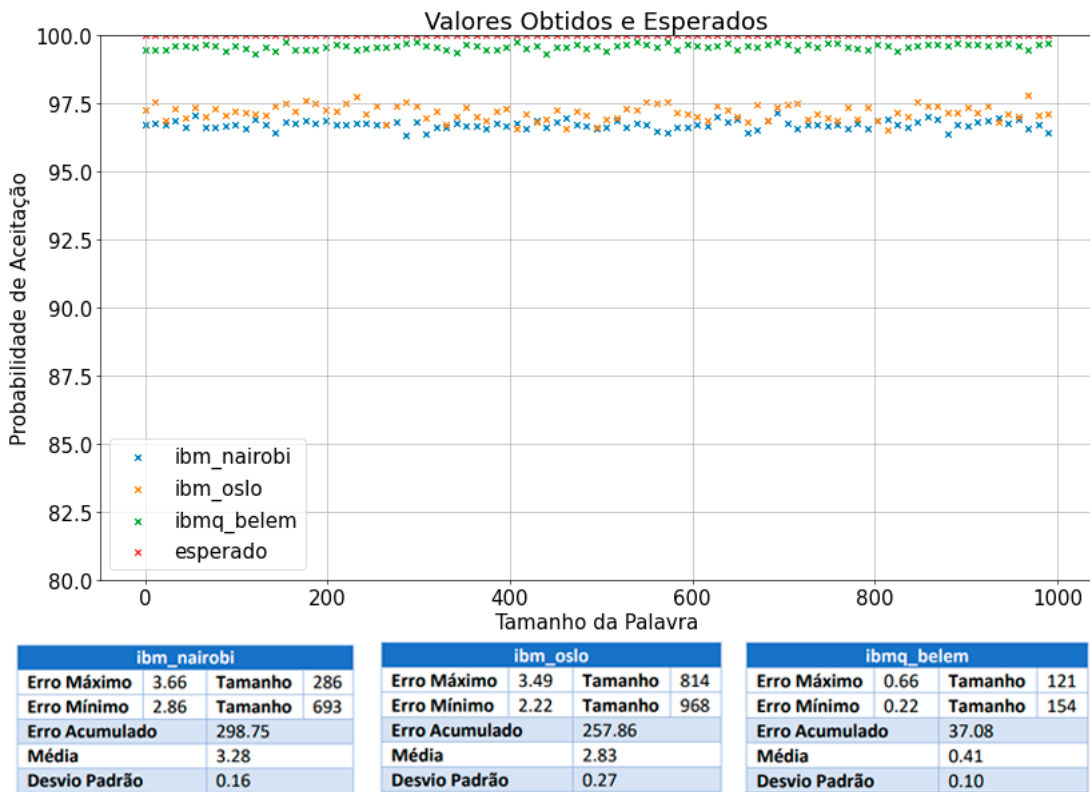


Figura 5.3 – Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 0 módulo 11 com nível de otimização 1.

O primeiro ponto para se observar é que não há nenhuma acumulação do erro, ele se mantém em torno do mesmo valor independentemente do tamanho da palavra. Isso ocorre porque, ao contrário do nível de otimização 0, o nível de otimização 1 colapsa portas lógicas adjacentes, nesse caso, a sequência de portas R_x , e aplica apenas uma sequência de portas lógicas equivalentes a R_x , que são dependentes do hardware, com ângulo proporcional à sequência de todas as portas para aquela palavra. No caso de palavras de tamanho n congruentes a 0 módulo 11, o circuito transpilado contém apenas a medição, já que $R_x(n \times 4\pi/11) = R_x(0)$. Para palavras de tamanho congruente a 3 módulo 11, é necessário aplicar uma porta lógica R_x para $n = 3$, conforme é possível

visualizar na figura 5.5, que mostra como é esse circuito transpilado para as portas lógicas disponíveis no hardware, nesse caso, os três computadores quânticos possuem o mesmo circuito.

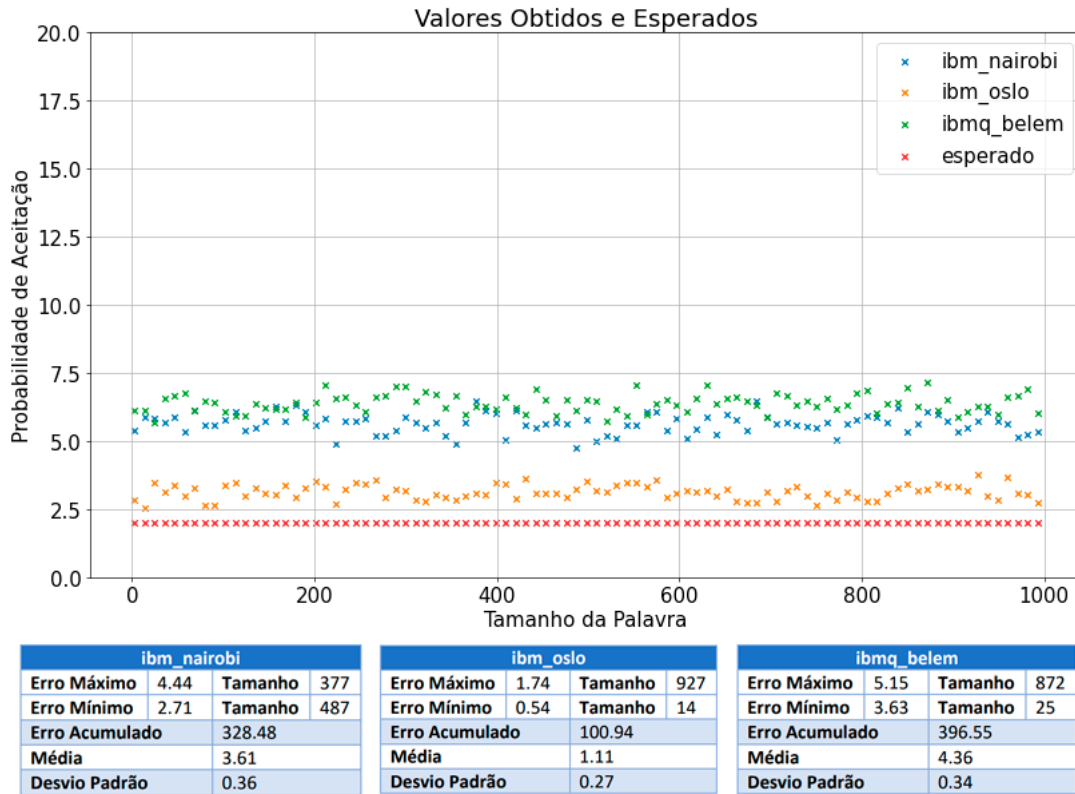


Figura 5.4 – Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 3 módulo 11 com nível de otimização 1.

Utilizando o nível de otimização 1, o problema do acúmulo do erro é superado, mas ainda existe um pequeno erro entre o obtido e o esperado. Em termos de análise de dados, ambos os experimentos com nível de otimização 1 possuem desvio padrão semelhantes, ambos pequenos, indicando que não há uma grande variabilidade dos resultados. Ademais, é possível perceber que nos computadores Nairobi e Belém, as palavras de tamanho congruente a 3 módulo 11 possuem um erro maior que as de tamanho congruente a 0 módulo 11, com uma discrepância mais atípica para o computador Belém. Isso ocorre devido ao fato de que palavras de tamanho congruente a 0 módulo 11 fazem com que seja aplicado apenas uma porta lógica $R_x(0)$ ao qubit, ou seja, nenhuma rotação, enquanto os circuitos de palavras de tamanho congruente a 3 módulo 11 são compostos de uma rotação $\theta = 3 \times \frac{4\pi}{11}$, portanto, existe um erro associado à essa porta lógica. Em conformidade com isso, as tabelas mostram que o valor acumulado do erro é maior para palavras de tamanho congruente a 3 módulo 11 nesses casos. Por fim, os valores de máximo e mínimo dos erros não possuem nenhuma preferência para o tamanho da palavra, dado que todas as palavras de tamanho congruente a n módulo 11 utiliza a mesma transformação, os erros são arbitrários.

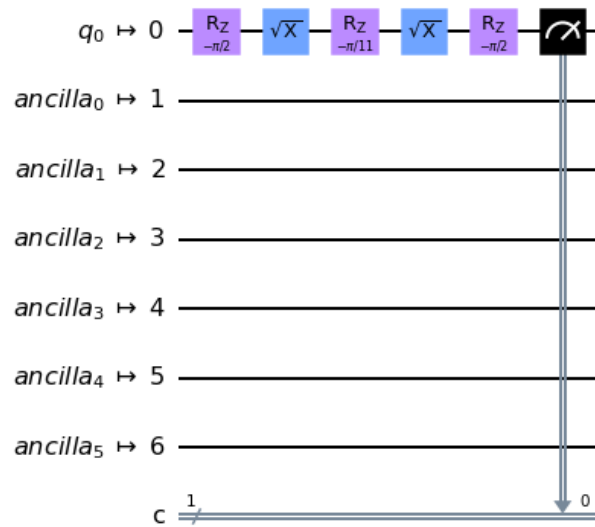


Figura 5.5 – Circuito para palavras de tamanho congruente a 3 módulo 11 transpilado para o hardware.

Com relação às diferentes plataformas, o computador Oslo teve um erro menor em palavras de tamanho congruente a 3 módulo 11, da mesma forma que o computador Belém teve um erro consideravelmente menor para a outra categoria de palavras. Em ambos os casos, acredita-se que o erro tenha colaborado positivamente, o que é possível e interessante de ser explorado em otimizações. Entretanto, o computador Belém obteve bons resultados para o circuito apenas com a medida, e a partir do momento em que portas lógicas foram aplicadas, obteve um erro semelhante ao Nairobi, enquanto o computador Oslo acumulou positivamente um erro ou acumulou um erro menor na aplicação das portas lógicas.

5.2.1 Caracterização do Erro com Nível de Otimização 1

Uma vez que o nível de otimização 1 não acumula os erros das transformações do circuito, os valores de α e β da equação 5 podem ser considerados constantes. δ será desconsiderado nessa caracterização devido ao fato de ele ser irrelevante nesse caso, resultando na equação 6 para uma palavra w .

$$\alpha \times \cos^2\left(\frac{2\pi}{11}|w|\right) + \beta, \quad (6)$$

Dessa forma, com base nos resultados desse nível de otimização, os valores de α e β podem ser obtidos por meio de um ajuste de função, que resulta, aproximadamente, nos valores apresentados na tabela 3. A análise desses valores pode auxiliar a entender o funcionamento dos erros anômalos dos computadores Oslo e Belém vistos anteriormente. É possível perceber que esses dois computadores possuem uma menor redução na amplitude total da função cosseno com valor de α na casa dos 0,95, 3 centésimos a mais do que o valor do Nairobi, entretanto, os valores de β mostram que a função do computador Belém

é mais deslocada para cima, enquanto o Oslo é mais deslocado para baixo. O computador Nairobi permanece com uma função mais equilibrada e menos vícios na localização dos erros com relação ao tamanho da palavra.

Tabela 3 – Valores de α e β da equação 6 para os computadores quânticos Nairobi, Oslo e Belém.

[HTML]D9E2F3[HTML]1F497D	[HTML]1F497D	[HTML]1F497D
[HTML]D9E2F3Nairobi	0.9296	0.0375
[HTML]D9E2F3Oslo	0.9597	0.0119
[HTML]D9E2F3Belém	0.9514	0.0446

A tabela 4 mostra um resumo dos valores de erros absolutos entre o previsto pela caracterização do erro e os valores reais obtidos nos experimentos, ela apresenta a média, desvio padrão e erro acumulado com relação aos resultados de todos os experimentos. É possível perceber que os valores de α e β foram obtidos com uma boa precisão, dado que a função se aproxima bem das probabilidades de aceitação dos resultados dos experimentos com as palavras de tamanho congruente a 0 e 3 módulo 11.

Tabela 4 – Resumo dos erros absolutos entre os resultados dos experimentos e a função aproximada dada pela equação 6.

[HTML]1F497D			
[HTML]D9E2F3Nairobi	0.197	0.189	35.769
[HTML]D9E2F3Oslo	0.218	0.151	39.682
[HTML]D9E2F3Belém	0.177	0.171	32.280

5.3 EXPERIMENTOS COM NÍVEL DE OTIMIZAÇÃO 0

Os próximos experimentos consideram o nível de otimização 0 do Qiskit. A figura 5.6 mostra os resultados obtidos para palavras de tamanho congruente a 0 módulo 11 e a figura 5.7 mostra os resultados obtidos para palavras de tamanho congruente a 3 módulo 11. Os experimentos foram executados nos computadores quânticos Nairobi, Oslo e Belém.

Como já comentado sobre o nível de otimização 0, existe um acúmulo no erro conforme o tamanho da palavra aumenta, as tabelas mostram que os erros mínimos acontecem por volta do menor tamanho da palavra, e conforme ele aumenta, as figuras mostram que o erro também aumenta, até se tornar arbitrário. Entretanto, existe um certo padrão entre os computadores quânticos, as probabilidades de aceitação contêm pontos de máximo e mínimo. Para palavras de tamanho congruente a 0 módulo 11, os computadores Nairobi e Oslo tiveram erros crescentes até por volta de 200 repetições da porta lógica, onde atingiram erros absolutos em torno de 70%, enquanto o computador Belém teve um crescimento menor até atingir um máximo de 47% em 286 repetições. Depois disso, os resultados mostram outros pontos de erros mínimos e máximos, onde até

mesmo para palavras de tamanho congruente a 3 módulo 11 o computador Belém teve oscilações menores. Apesar disso, para o experimento com palavras de tamanho congruente a 3 módulo 11, o computador Belém teve um crescimento no erro semelhante aos demais computadores quânticos.

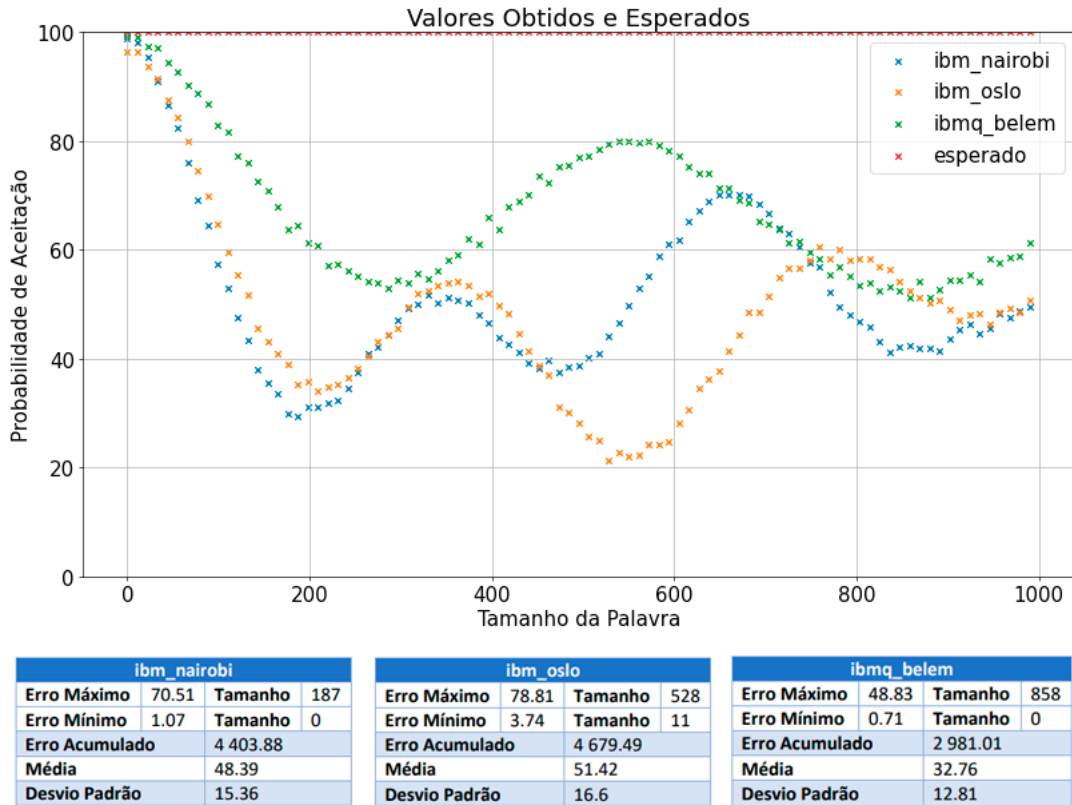


Figura 5.6 – Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 0 módulo 11 com nível de otimização 0.

As tabelas também mostram que o erro máximo absoluto é semelhante entre palavras de tamanho congruente a 0 e 3 módulo 11 em ambos os três computadores, mas o erro mínimo é menor para palavras de tamanho congruente a 0 módulo 11, com exceção do computador Oslo que já teve melhores resultados em palavras de tamanho congruente a 3 módulo 11 com o nível de otimização 1, conforme abordado na seção 5.2. Como dito anteriormente, isso acontece porque a computação dessas últimas não necessita de nenhuma transformação porque representam rotações completas do vetor estado em torno da esfera de Bloch, enquanto o circuito para palavras de tamanho 3 há a aplicação de três portas lógicas $R_x\left(\frac{4\pi}{11}\right)$, acumulando um erro.

Diferentemente do nível de otimização 1, o nível de otimização 0 não colapsa as portas lógicas adjacentes e implementa cada porta lógica R_x como uma sequência de outras cinco portas lógicas, conforme mostra a figura 5.8. Por conta disso, o circuito se torna excessivamente longo e inviável de ser utilizado.

Ademais, é possível perceber que os desvios padrão são muito maiores do que os

valores dos experimentos com nível de otimização 1, portanto, os valores obtidos estão muito longe de serem tão previsíveis quanto os experimentos com nível de otimização 1. Por esse motivo, os elementos α , β e δ da caracterização do erro da equação 5 não podem ser considerados constantes e pode ser muito difícil obter funções que os descrevam.

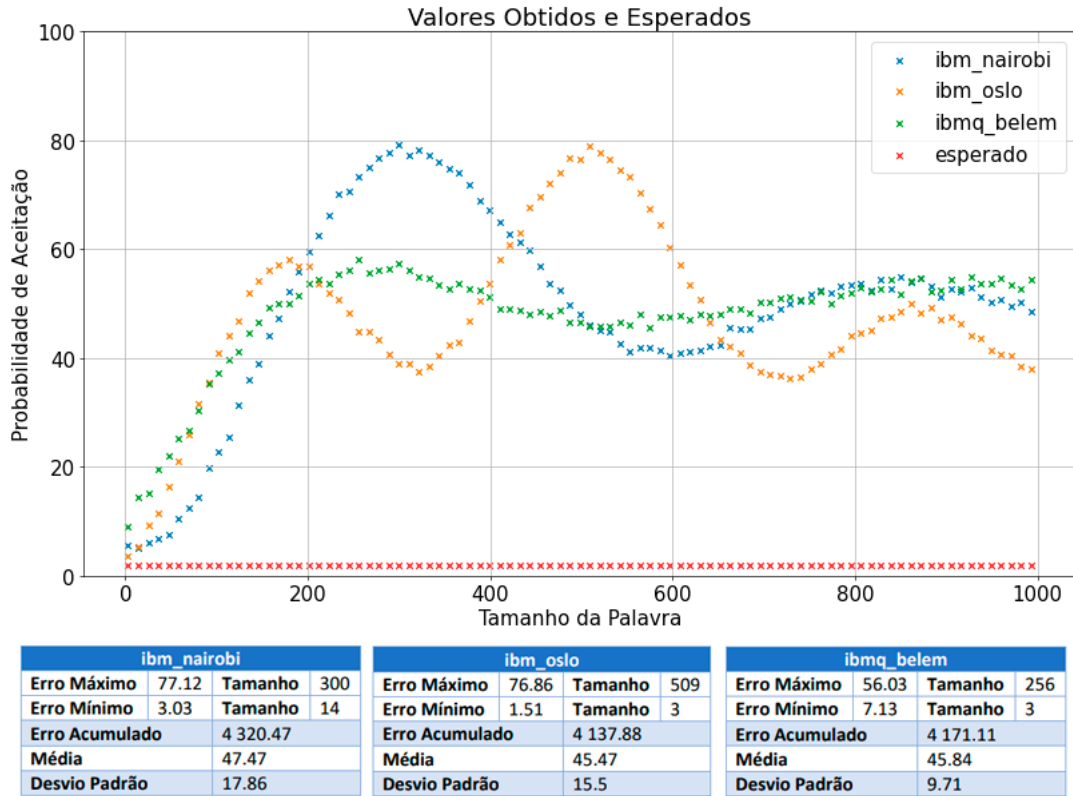


Figura 5.7 – Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 3 módulo 11 com nível de otimização 0.

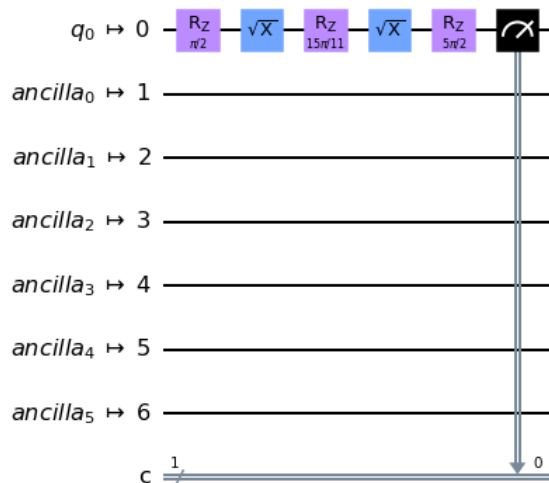


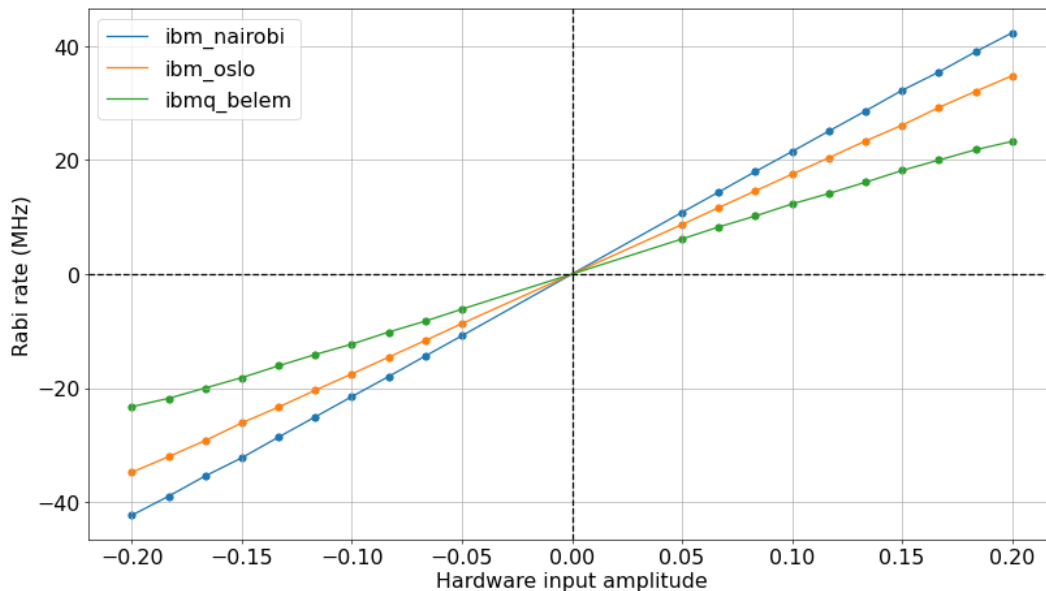
Figura 5.8 – Circuito que implementa a porta lógica $R_x(4\pi/11)$ com nível de otimização 0.

Esses experimentos revelam que é insustentável trabalhar com circuitos não-otimizados, conforme o tamanho da palavra aumenta, é necessário aplicar cada vez mais portas lógicas ao qubit, e a acumulação do erro faz com que o autômato perca o controle de seu estado. Otimizações circuitais são definitivamente necessárias para a implementação de autômatos quânticos. Dado que sua computação é composta de múltiplas portas lógicas que dependem do tamanho da palavra, é necessário otimizar as transformações dos qubits para deixar a computação mais robusta com relação aos tamanhos das palavras.

5.4 EXPERIMENTOS COM PORTAS LÓGICAS SQUARE ULTRARRÁPIDAS

O Q-CTRL Boulder Opal oferece maneiras eficientes de calibrar o hardware quântico e construir portas lógicas quânticas otimizadas resistentes à ruídos. Os experimentos desta seção consideram a criação de uma porta lógica R_x ultrarrápida que busca ser eficiente na computação de palavras longas. Por se tratar de um pulso de baixa duração, a computação do circuito completo é consideravelmente mais rápida e se distancia mais dos tempos de decoerência do computador quântico. Para isso, serão utilizados os manuais Q-CTRL (s.d.) e QCTRL (2022), o primeiro para a geração de portas lógicas e o segundo para a calibração do hardware.

Figura 5.9 – Frequências de Rabi para os computadores Nairobi, Oslo e Belém.



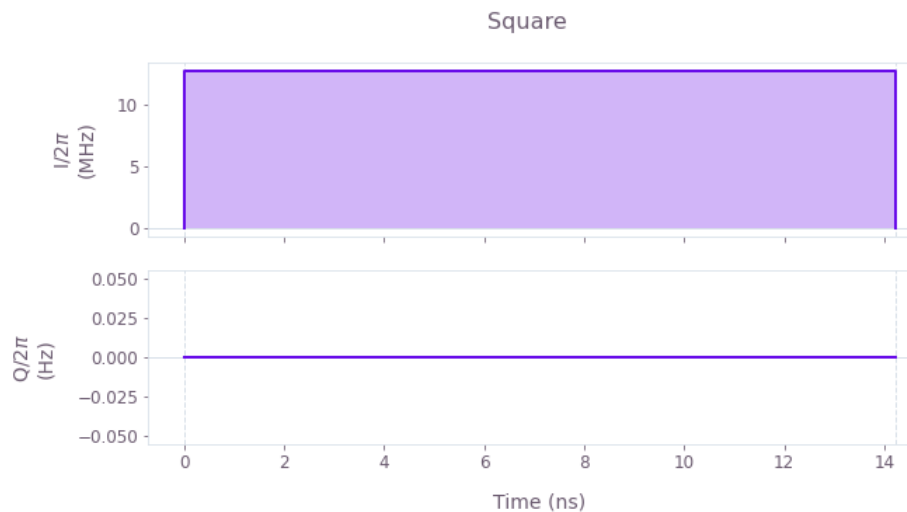
Antes de criar a porta lógica, é necessário realizar a calibração do hardware por meio do experimento de Rabi, conforme apresentado na seção 3.4, onde se obtém as frequências de Rabi, que correspondem aos tempos necessários para realizar rotações na esfera de Bloch para pulsos com diferentes valores de amplitude. O gráfico da figura 5.9 mostra essas frequências obtidas para pulsos de amplitude $-0,2$ a $+0,2$ nos computadores Nairobi, Oslo e Belém. É possível perceber que computadores quânticos diferentes possuem valores

de amplitude diferentes para uma mesma frequência, o computador quântico Nairobi possui frequências de Rabi bastante altas para amplitudes mais baixas, isso significa que é possível construir pulsos consideravelmente rápidos com amplitudes mais baixas, evitando possíveis erros. O computador Belém é o que possui as menores frequências de Rabi, portanto, necessita de pulsos com amplitudes maiores para obter o mesmo efeito que os demais.

Após a realização do experimento de Rabi, é mais fácil estimar os parâmetros para a criação do pulso. Nesse caso, foi definido um pulso do tipo square, ou seja, um pulso ideal que não considera a ocorrência de erros. Apesar do fato de ele não mapear erros de *dephasing* ou controle de amplitude, ele é bastante útil para a implementação do autômato quântico por conta de sua simplicidade e rapidez, permitindo computar longas palavras e se distanciar do tempo de decoerência do sistema quântico. Dessa forma, enquanto a implementação da porta R_x padrão dos computadores considerados é composta de outras cinco portas lógicas conforme mostra a figura 5.8, a porta lógica square gerada é apenas uma de duração $14,2 \text{ ns}$, que reduz absurdamente o tempo de execução dos circuitos.

Nesses termos, o pulso square gerado tem o formato representado pela figura 5.10. Esse gráfico basicamente diz que é desejado um pulso de duração $14,2 \text{ ns}$ com a parte imaginária da amplitude igual a 0 e a parte real necessária para uma frequência de Rabi de aproximadamente $12,78 \text{ MHz}$ (por 2π), conforme equação 1, que é o valor necessário para uma rotação de $4\pi/11$ no tempo de $14,2 \text{ ns}$. A partir do gráfico do experimento de Rabi, se obtém os valores de amplitude dos pulsos para os computadores Nairobi, Oslo e Belém, que necessitam de amplitudes $0,05914853$, $0,07302414$ e $0,10439492$. Nenhum deles necessita de amplitudes maiores do que as disponíveis na função do gráfico da figura 5.9, entretanto, o computador Belém precisa de um valor um pouco maior do que os demais.

Figura 5.10 – Pulso square R_x .



A partir da simples geração de um pulso específico para a operação desejada, já é

possível conseguir resultados consideravelmente melhores. As figuras 5.11 e 5.12 mostram os resultados do pulso diretamente convertido para amplitudes por meio do experimento de Rabi (calibração básica) para palavras de tamanho congruente a 0 e 3, respectivamente, executados no computador quântico Nairobi. Por limitações de tempo com relação às filas de espera para a execução dos circuitos, esse experimento foi executado apenas no computador Nairobi, entretanto, resultados semelhantes devem ser verificados nos demais computadores quânticos.

Figura 5.11 – Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 0 módulo 11 para o computador Nairobi.

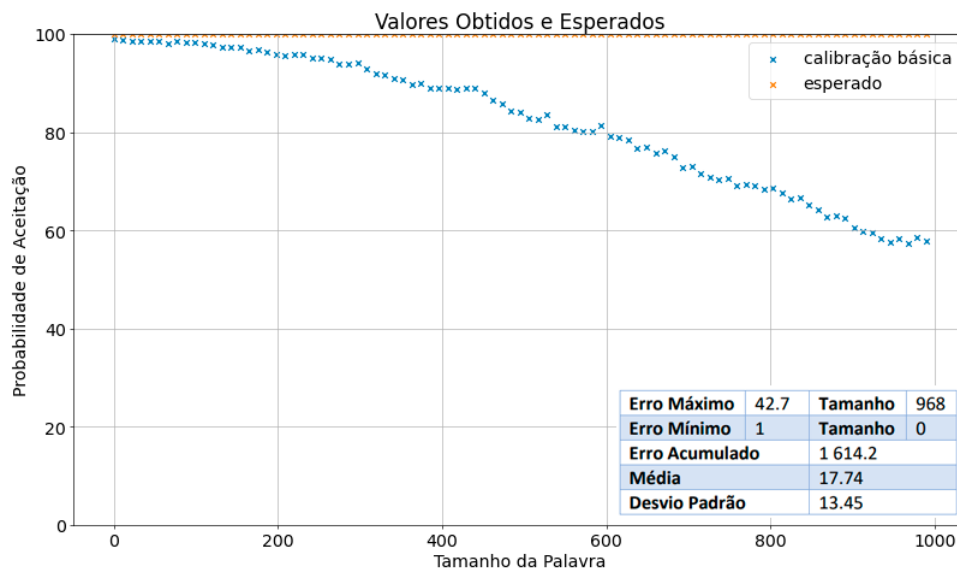
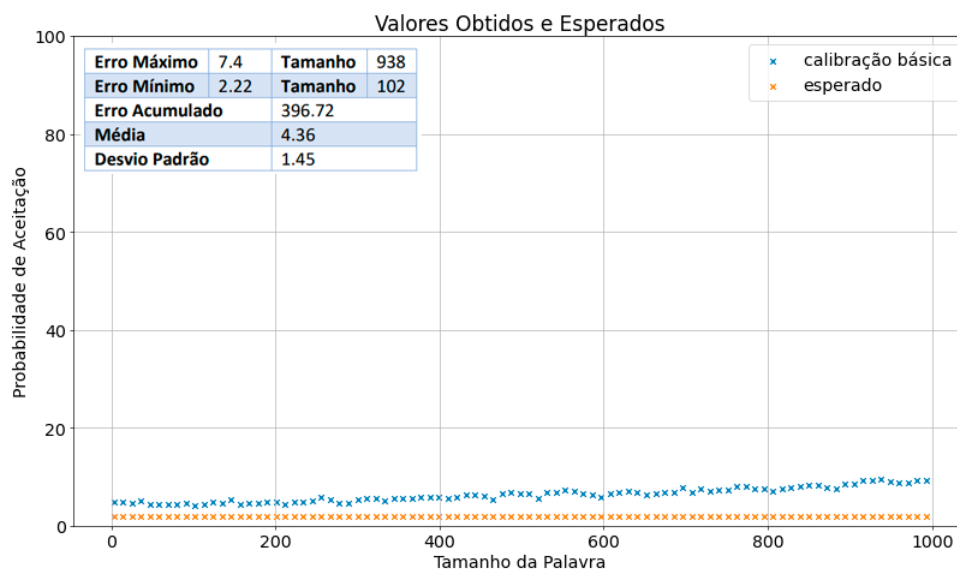


Figura 5.12 – Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 3 módulo 11 para o computador Nairobi.



Os resultados para esse pulso são bastante intrigantes com relação à porta lógica padrão, os desvios nas probabilidades de aceitação com relação ao esperado são muito

menores, especialmente para as palavras congruentes a 3 módulo 11, onde o erro foi consideravelmente pequeno. Dado que a natureza dos erros é por vezes estocástica, os mesmos experimentos podem ter resultados diversos em momentos diferentes devido a imensuráveis fatores.

Em comparação com os resultados da porta lógica padrão com nível de otimização 0, o baixo tempo do pulso se mostrou bastante eficiente mesmo com a calibração básica. Para os experimentos com palavras de tamanho congruente a 0 módulo 11, enquanto a probabilidade de aceitação da porta lógica padrão já acumulava um erro absoluto de 70,51% em 198 repetições, a calibração básica ainda estava com um erro absoluto de 3,64%. O mesmo ocorre para palavras congruentes a 3 módulo 11, enquanto a porta lógica padrão acumulava um erro absoluto de 55,96% em 201 repetições, o pulso rápido estava com um erro absoluto de 2,83%. Além disso, os gráficos mostram que as probabilidades de aceitação são menos afetados pelo tamanho da palavra.

Os experimentos com o pulso rápido de 14,2 *ns* para a implementação do autômato mostram que eles são absolutamente mais eficientes do que a transpilação para os pulsos genéricos do Qiski. Gerar pulsos square individuais para cada circuito pode ser uma alternativa mais interessante do que manter um conjunto de portas lógicas universais calibradas. Todavia, a otimização circuital dos experimentos com nível de otimização 1 é obviamente essencial nesse caso, mas eventualmente pode não ser possível conseguir um resultado tão bom com essa otimização e evitar a aplicação de muitos pulsos, é nesses casos que o controle quântico se mostra necessário para amenizar os erros. Dessa forma, é perceptível que a utilização de controle quântico para construir portas lógicas personalizadas para determinados algoritmos é uma alternativa capaz de reduzir consideravelmente a ocorrência de erros.

Especialmente na implementação de autômatos quânticos, a utilização de pulsos rápidos é uma solução capaz de reduzir o tamanho do circuito e distanciá-lo dos tempos de decoerência dos computadores quânticos. Isso justifica o fato de que os resultados do pulso rápido possuem uma durabilidade maior do estado quântico com relação ao tamanho da palavra do que o pulso padrão, isto é, o erro do pulso padrão se torna arbitrário depois de 200 repetições, enquanto a calibração básica teve apenas um erro crescente até 1000 repetições.

5.5 CALIBRAÇÕES BÁSICAS, FINE-TUNED E AUTOMATED

Os experimentos da seção 5.4 mostraram que portas lógicas ultrarrápidas são uma boa alternativa para melhorar os resultados de circuitos longos na implementação do autômato. Especialmente para esse tipo de aplicação, existem técnicas que aperfeiçoam a calibração do hardware quântico para tornar os pulsos mais duráveis com relação ao tempo de decoerência do sistema. Esta seção apresenta duas alternativas apresentadas no manual QCTRL (2022) para aperfeiçoar essa calibração, que são a calibração fine-tuned

já apresentada na seção 3.5 e um método autônomo apresentado pelo artigo Baum *et al.* (2021), melhor descrito na seção 4.2. Os experimentos desta seção foram realizados apenas no computador quântico Nairobi.

5.5.1 Calibração fine-tuned

Figura 5.13 – Parâmetro S_{rel} para diferentes repetições do pulso.

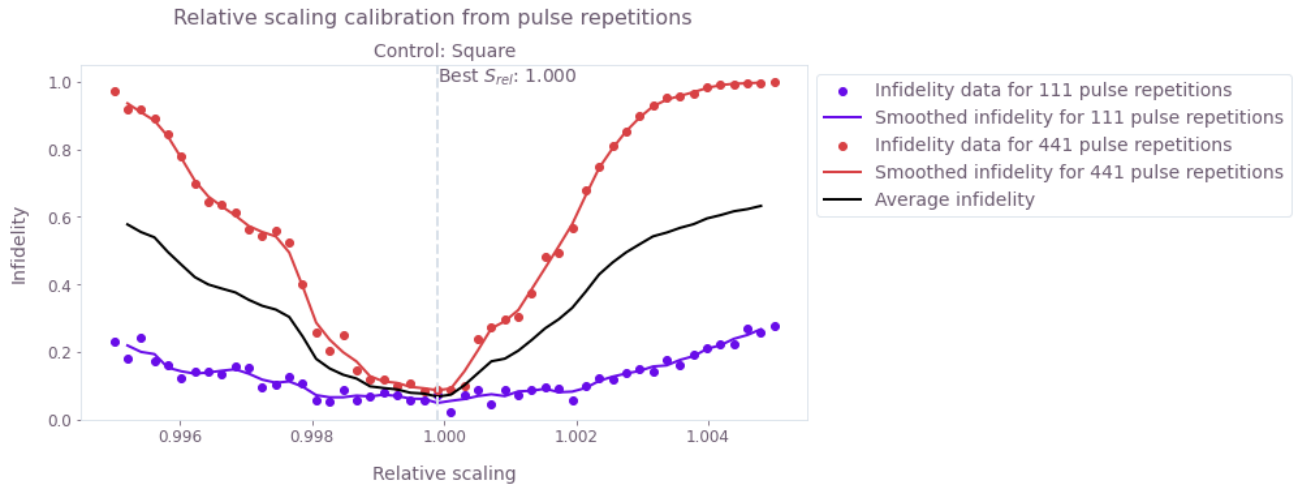
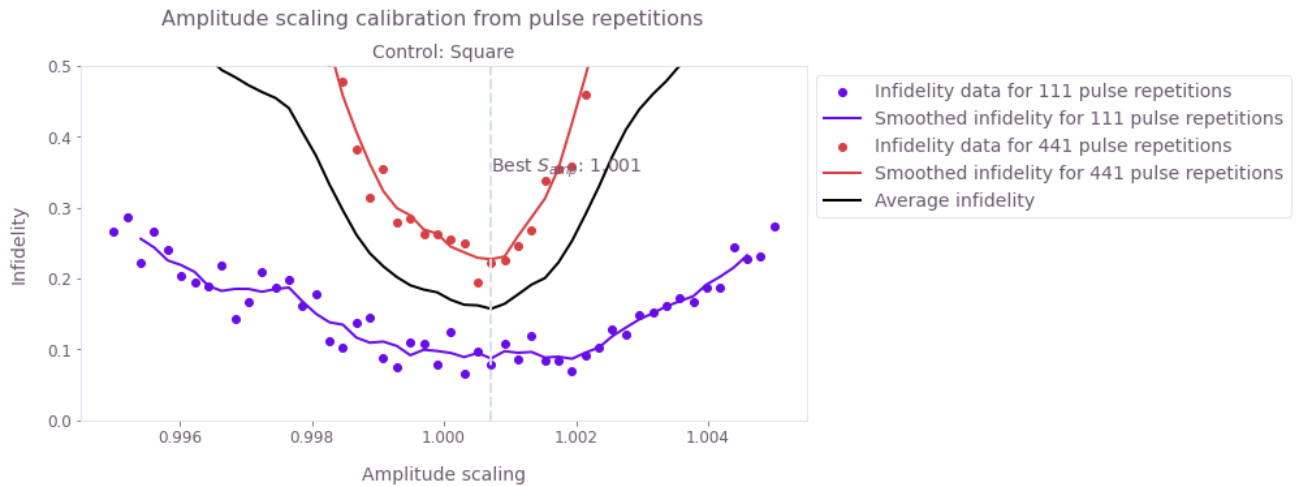


Figura 5.14 – Parâmetro S_{amp} para diferentes repetições do pulso.

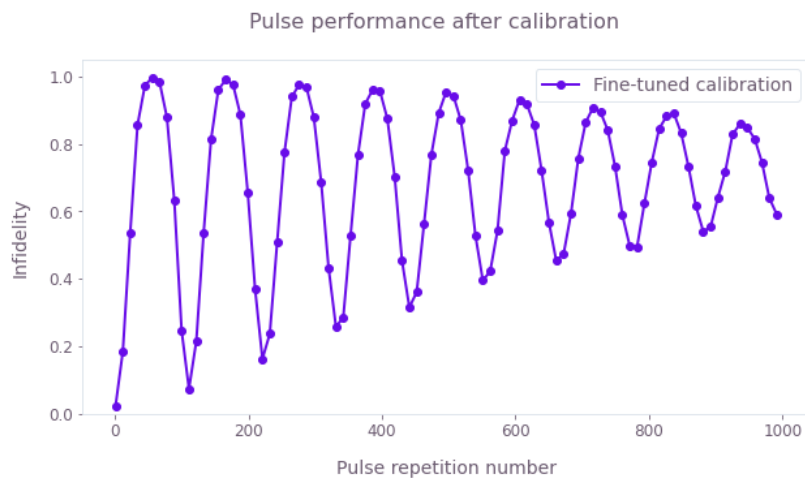


Conforme as definições já apresentadas na seção 3.5, os experimentos realizados consideram os valores de S_{rel} e S_{amp} da equação 2 definidos para o intervalo $[0, 995; 1, 005]$, com 111 e 441 repetições com o objetivo de reduzir a infidelidade dos resultados até 1000 repetições. Para S_{rel} , relativo ao valor I do pulso, o valor encontrado foi de aproximadamente 1,000, conforme mostra o gráfico da figura 5.13, onde a menor infidelidade média dentre as repetições 111 e 441 foi encontrada com esse valor. Já para S_{amp} , relacionado a

todo o pulso, o melhor parâmetro encontrado foi algo muito próximo de 1,001, conforme mostra o gráfico da figura 5.14.

De maneira geral, um intervalo pequeno como $[0,99; 1,01]$ e mais próximo de 1 se mostrou mais interessante para experimentos onde ainda não se conhece nenhum resultado dessa calibração para o pulso, todavia, intervalos menores permitem encontrar valores mais precisos desde que haja o conhecimento de que as menores infidelidades são obtidas nesse intervalo. Também não é desejável encontrar valores muito distantes de 1, pois existe o risco das calibrações se aperfeiçoarem demais sobre as quantidades de repetições definidas e comprometerem as demais. A figura 5.15 mostra um exemplo de gráfico de infidelidade com relação ao número de repetições do pulso. É possível perceber o efeito de se obter os valores de S_{rel} e S_{amp} em pontos de mínimo muito distantes de 1. A infidelidade é realmente baixa para as quantidades de repetições escolhidas para a calibração e muito alta para outras repetições do pulso, já que a calibração acabou se especializando demais nas repetições para qual o pulso foi calibrado. Nesse experimento, foram escolhidas as mesmas quantidades de repetições 111 e 441, é possível perceber que a infidelidade é realmente pequena para elas.

Figura 5.15 – Resultado da calibração fine-tuned com parâmetros mal escolhidos.



5.5.2 Calibração Autônoma

A calibração autônoma do Q-CTRL Boulder Opal é uma alternativa automatizada para encontrar os parâmetros S_{rel} e S_{amp} da equação 2. Essa calibração funciona em um laço fechado onde são definidos intervalos para os parâmetros e repetições do pulsos a serem calibradas, a partir disso, são feitos experimentos em sequência onde o agente busca por alternativas dos parâmetros no intervalo que diminua a infidelidade média dos resultados das repetições.

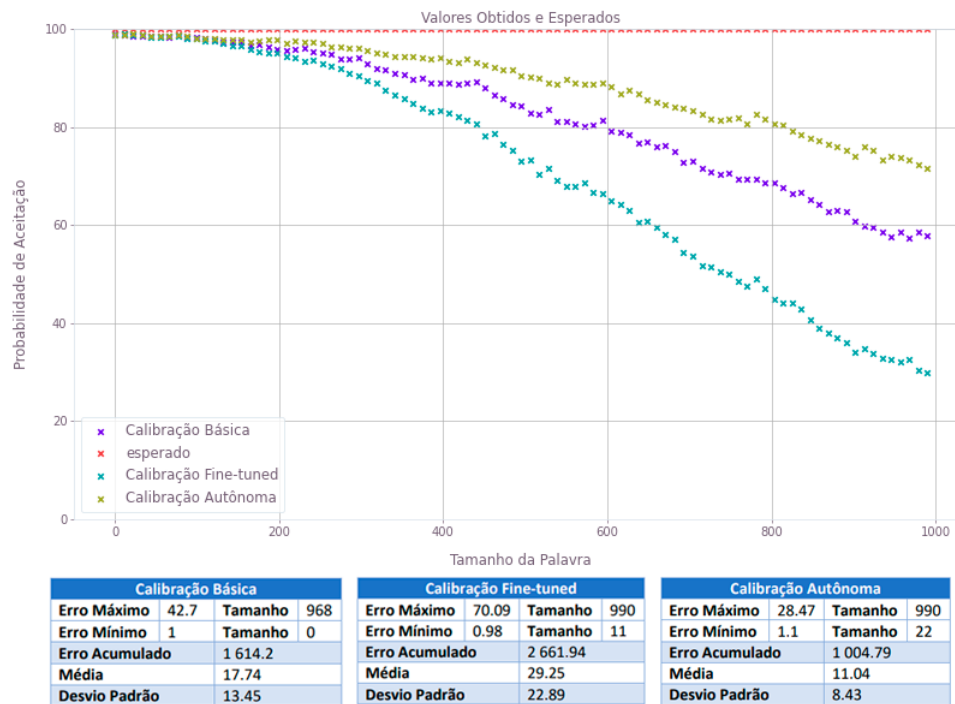
De maneira geral, essa calibração é mais interessante de ser utilizada do que a fine-tuned por conta da consideração de uma quantidade maior de diferentes repetições

do pulso, que seria inviável de ser feito com a calibração fine-tuned. Os experimentos realizados com a calibração autônoma consideraram um intervalo maior de $[0, 97; 1, 03]$ para os parâmetros e repetições 12, 177, 342, 507, 672 e 837. Em 10 iterações, ou 10 experimentos, o agente encontrou os valores $S_{rel} = 0,97414997$ e $S_{amp} = 1,02569915$.

5.5.3 Resultados

A utilização das calibrações se mostrou útil até certo ponto. Foram feitos vários experimentos e percebeu-se que a necessidade de calibrar precisamente um determinado pulso é maior conforme as calibrações do experimento de Rabi vão perdendo sua precisão. As calibrações fine-tuned e autônoma foram feitas no mesmo dia do experimento de Rabi e os valores limites de S_{amp} e S_{rel} foram ajustados para serem mais próximos de 1.

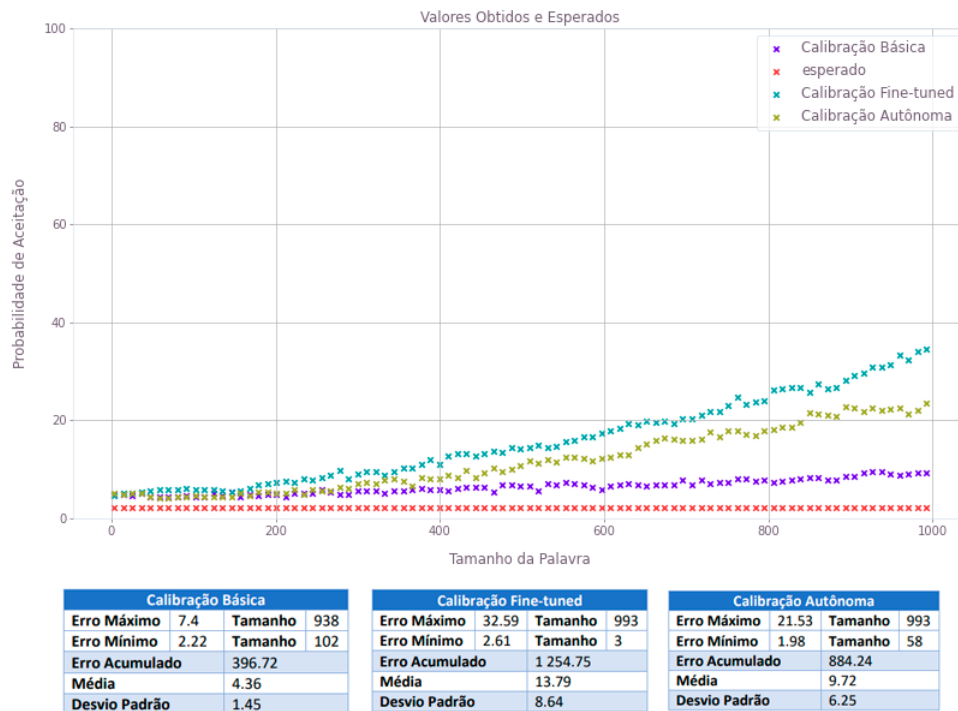
Figura 5.16 – Probabilidades de aceitação do pulso square com calibrações básica, fine-tuned e autônoma para palavras de tamanho congruente a 0 módulo 11.



Os gráficos das figuras 5.16 e 5.17 mostram os resultados para palavras congruentes a 0 e a 3 módulo 11, nessa ordem, para as calibrações básicas, fine-tuned e autônoma. Para palavras de tamanho congruente a 0 módulo 11, a calibração autônoma conseguiu superar os resultados da calibração básica, entretanto, a calibração fine-tuned piorou os resultados nesse teste. As palavras de tamanho congruente a 3 módulo 11 obtiveram melhores resultados para a calibração básica, que teve um resultado excepcionalmente bom, mas a calibração autônoma apresentou resultados melhores do que a fine-tuned. Em geral, a calibração autônoma é mais produtiva e tende a apresentar melhores resultados.

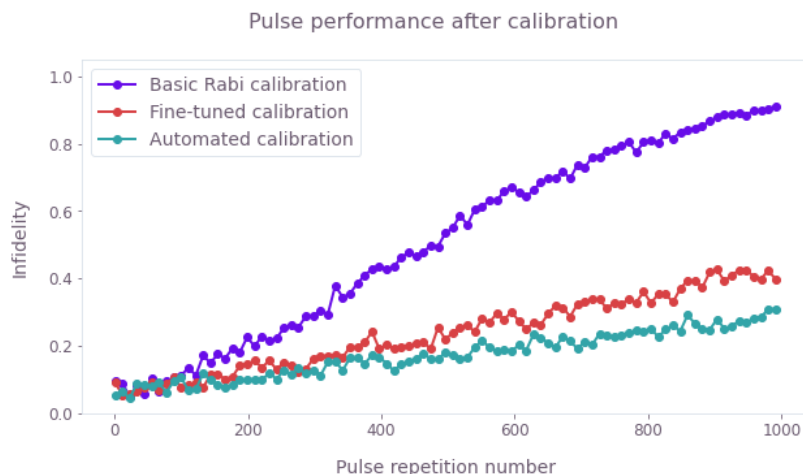
Mas teoricamente, as calibrações fine-tuned e autônoma não deveriam apresentar menores erros absolutos na probabilidade de aceitação do que a calibração básica? A

Figura 5.17 – Probabilidades de aceitação do pulso square com calibrações básica, fine-tuned e autônoma para palavras de tamanho congruente a 3 módulo 11.



resposta para essa pergunta consiste de alguns fatores. O primeiro deles se diz à respeito da precisão da calibração do experimento de Rabi, que é perdida com o tempo, as calibrações fine-tuned e autônoma podem ajudar a tornar o experimento de Rabi mais durável, dessa forma, como as calibrações foram realizadas no mesmo dia do experimento de Rabi, essas otimizações podem ter menos efeito. O segundo fator diz a respeito da medida do erro, a infidelidade calcula a diferença do sinal medido e esperado com relação aos eixos x , y e z da esfera de Bloch, contendo essas informações extras relevantes além da probabilidade de aceitação, que depende apenas do eixo z , dessa forma, as medidas de infidelidade e probabilidade de aceitação podem ter divergências na interpretação dos resultados.

Figura 5.18 – Infidelidade do pulso square com calibrações básica, fine-tuned e autônoma.

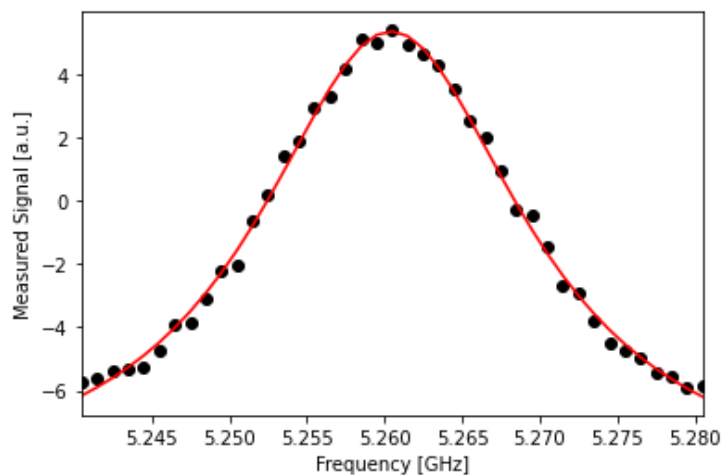


Do ponto de vista da computação quântica teórica, os computadores quânticos estão sujeitos tanto a erros de bit-flip quanto a erros de phase-flip (ROFFE, 2019), a medida de infidelidade é capaz de capturar ambos os erros. As calibrações aqui apresentadas consideram a infidelidade como medida de desempenho, e assim foram calibradas. Mesmo que erros de phase-flip não sejam importantes para a implementação do autômato, o gráfico da figura 5.18 mostra a diferença de infidelidade do pulso square utilizando as três calibrações. Nesse caso, é possível perceber que a calibração básica tem desempenho consideravelmente inferior às demais calibrações, enquanto a fine-tuned e autônoma obtiveram desempenhos semelhantes, mas a autônoma conseguiu uma infidelidade menor. No caso da implementação do autômato, pode ser mais interessante definir a probabilidade de aceitação como medida de erro para as otimizações, mas na maioria dos casos essa informação não tem a mesma eficácia da medida de infidelidade.

5.6 OTIMIZAÇÃO DA FREQUÊNCIA

Como já descrito na seção 3, a frequência é um componente muito importante na descrição dos pulsos, ela corresponde à frequência na qual os fótons do pulso são absorvidos pelo sistema quântico que implementa o qubit, permitindo alterar seu estado. Essa seção busca avaliar o experimento apresentado na seção 3.6, que mostra uma alternativa apresentada por IBM (2022a) para encontrar a frequência do qubit. Apesar do Qiskit já oferecer uma frequência padrão estimada suficientemente precisa, calibrar esse valor pode ser relevante devido a sua importância na ocorrência de erros.

Figura 5.19 – Resultado da otimização da frequência.

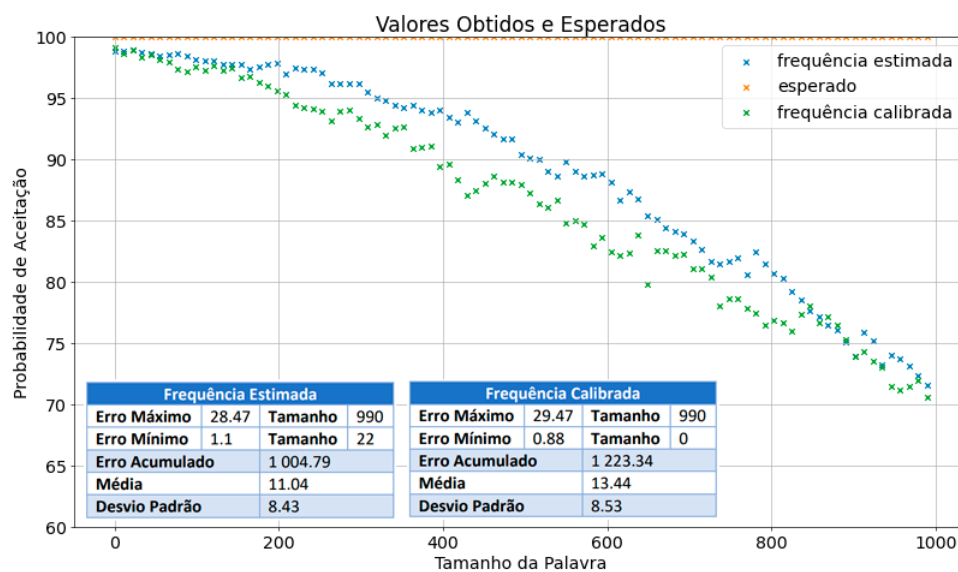


A frequência estimada do qubit 0 do computador quântico Nairobi é de $5,260492122850421 \text{ GHz}$. O experimento realizado consiste em testar valores de frequência 20 MHz maiores e 20 MHz menores do que o valor estimado na aplicação de um pulso, dessa forma, o intervalo de valores no qual a frequência foi testada é de

5,240492122850421 GHz a 5,280492122850421 GHz em intervalos de 1 MHz . A figura 5.19 mostra o resultado do experimento, onde o pico no centro corresponde à localização da frequência do qubit. A linha contínua corresponde a um ajuste dos pontos a uma função Lorentziana, utilizada para obter o valor da frequência, que foi atualizado para 5,260424304745819 GHz , uma diferença na ordem de dezenas de KHz com relação ao valor estimado.

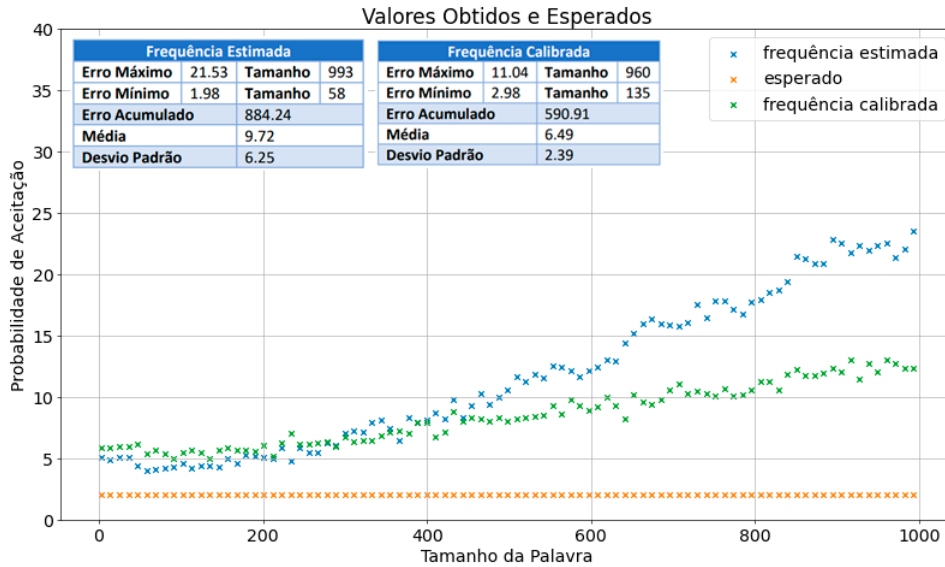
Para validar a necessidade da otimização da frequência, foram feitos experimentos na implementação do autômato para palavras de tamanho congruente a 0 e a 3 módulo 11, conforme os experimentos das demais seções, utilizando a calibração autônoma apresentada na seção 5.5 para a frequência padrão estimada e a calibrada. As figuras 5.20 e 5.21 mostram os resultados para palavras de tamanho congruente a 0 e 3 módulo 11 nessa ordem. É possível perceber que as palavras de tamanho congruente a 0 não tiveram melhorias nos erros absolutos com a frequência calibrada, o desempenho foi até um pouco melhor para a frequência estimada. Entretanto, as palavras de tamanho congruente a 3 módulo 11 obtiveram uma pequena melhoria de desempenho após 400 repetições do pulso.

Figura 5.20 – Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 0 módulo 11.



A frequência é um atributo importante, a sua calibração não é uma tarefa custosa e os experimentos mostraram que ela pode apresentar melhores resultados na execução de circuitos. Em palavras de tamanho congruente a 3 módulo 11, o erro absoluto acumulado foi 293,33 menor para a frequência calibrada, se destacando mais depois de 400 repetições. Mesmo que o comportamento do experimento para palavras de tamanho congruente a 0 tenha sido bastante semelhante para as duas frequências, com um erro acumulado maior para a frequência estimada, pode ser mais interessante utilizar uma frequência mais calibrada. Ainda assim, a frequência padrão estimada é suficientemente precisa para a maioria dos experimentos e também não é uma má alternativa.

Figura 5.21 – Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 3 módulo 11.



5.7 PULSOS RESISTENTES A ERROS DE *DEPHASING*

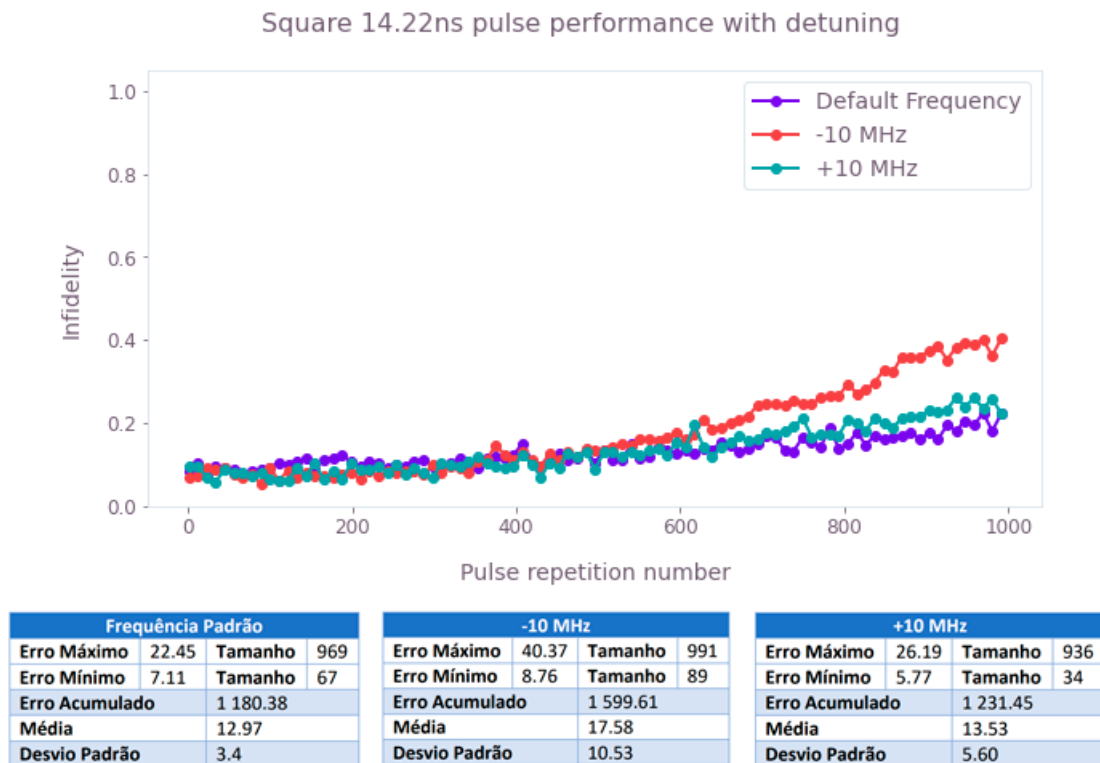
De acordo com o que foi apresentado na seção 3, existem diferentes formas de descrever pulsos. Pulsos square são mais simples porque possuem amplitude constante, mas existem formas mais sofisticadas de descrever pulsos de forma a contornar processos de erros. O manual Q-CTRL (s.d.) mostra como utilizar o Q-CTRL Boulder Opal para modelar pulsos resistentes a *dephasing* e flutuações no controle de amplitude. Basicamente, a descrição do pulso segue a equação 1, onde $\beta_\gamma(t)$ e $\eta(t)$ representam os processos de erros de controle de amplitude e *dephasing*, respectivamente. O hamiltoniano é modelado utilizando um grafo, onde são feitas otimizações considerando esses processos e o pulso é gerado discretizando os resultados da otimização. Esta seção utiliza pulsos square de diferentes durações e pulsos resistentes a erros de *dephasing* gerados pelo Boulder Opal de forma a verificar a robustez das diferentes formas com relação aos erros.

O experimento de verificação de *dephasing* consiste em um experimento de *detuning*, que induz o erro realizando o mesmo experimento de infidelidade da seção anterior aplicando o pulso em frequências diferentes da padrão, nesse caso, em intervalos de 10 *MHz* em torno da frequência padrão, com o intuito de causar decoerência. Os resultados são avaliados por meio da medida de infidelidade, já que o objetivo é avaliar a robustez dos pulsos.

O primeiro pulso square gerado de 14, 2 *ns* teve ótimos resultados na implementação do autômato por conta de sua curta duração, entretanto, ele exige amplitudes mais altas e isso pode fazer com que ele seja mais suscetível a erros. A figura 5.22 mostra a infidelidade do pulso square de 14 *ns* com as frequências padrão, -10 *MHz* e $+10$ *MHz*. É possível perceber que o pulso é consideravelmente resistente a erros na frequência, diferenças

relevantes só podem ser observadas após 600 repetições do pulso, onde a frequência com -10 MHz acaba acumulando um erro maior do que a $+10\text{ MHz}$ e a padrão, nessa ordem. Os números das tabelas confirmam que a acumulação do erro absoluto também segue essa ordem. Apesar disso, a frequência padrão e com $+10\text{ MHz}$ seguem muito próximas e pode-se dizer que é preferível errar a frequência para mais do que para menos.

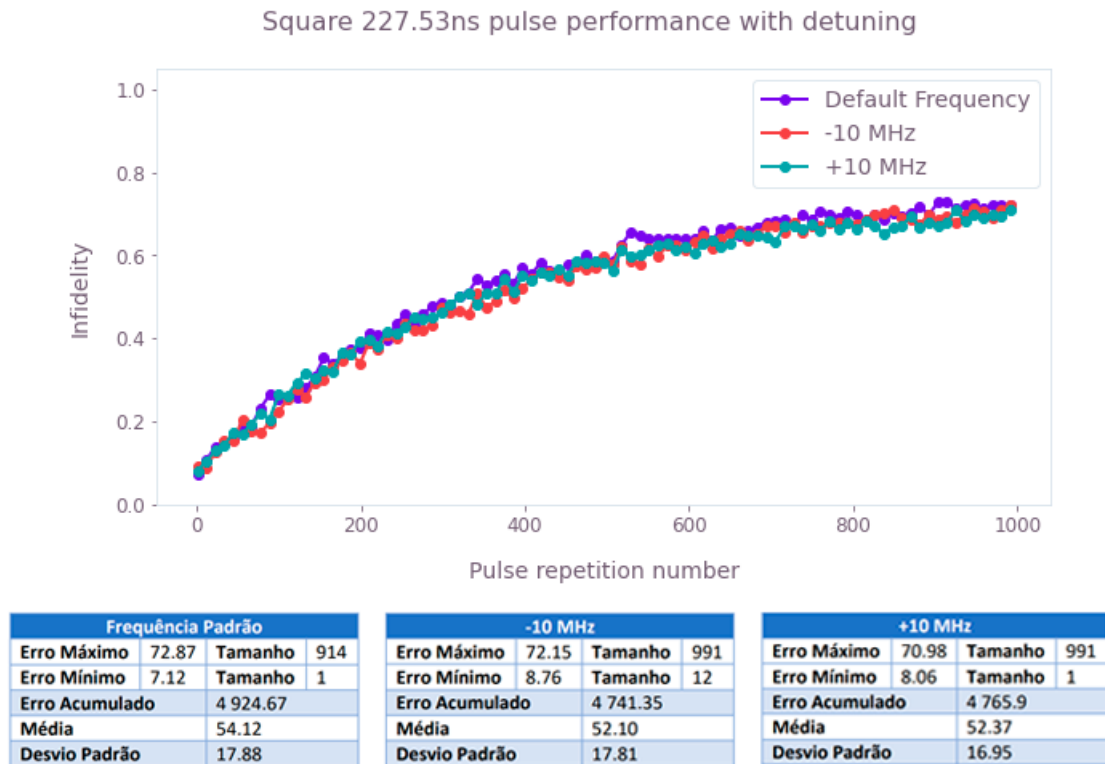
Figura 5.22 – Infidelidade do pulso square de $14,2\text{ ns}$ aplicado na frequência padrão, com -10 MHz e $+10\text{ MHz}$.



De forma a verificar a influência dos valores de amplitude na ocorrência de erros de *dephasing*, foi gerado outro pulso square com duração maior, de $227,33\text{ ns}$, com o mesmo formato do pulso de $14,2\text{ ns}$. A figura 5.23 mostra os resultados do mesmo experimento para esse pulso. Nesse caso, os erros para as diferentes frequências são imperceptíveis, as tabelas mostram que o erro acumulado para a frequência padrão é até ligeiramente maior do que para a frequência com erros de -10 MHz e $+10\text{ MHz}$. A hipótese de que pulsos com amplitudes mais baixas são mais resistentes a erros de frequência parece estar correta de acordo com esse experimento.

Apesar do pulso mais longo ter se mostrado mais resistente a processos de *dephasing*, a infidelidade é substancialmente maior, assim como o seu aumento à medida que o tamanho do circuito aumenta. Pode-se dizer que pulsos rápidos sofrem mais com *dephasing*, mas possuem resultados melhores pela menor exposição ao tempo de decoerência do sistema, que é o maior causador de erros. Dessa forma, a necessidade de robustez com relação a *dephasing* vem da necessidade de diminuir a duração do circuito com a construção de portas lógicas mais rápidas com amplitudes maiores.

Figura 5.23 – Infidelidade do pulso square de 227,33 ns aplicado na frequência padrão, com -10 MHz e +10 MHz.



O Boulder Opal se propõe a gerar pulsos otimizados que contornam problemas de *dephasing*, a figura 5.24 apresenta um pulso gerado que considera apenas o processo de erro de *dephasing*. Diferentemente dos pulsos square apresentados anteriormente, que possuem apenas um segmento de frequência de Rabi constante, o pulso do Boulder Opal é composto de 256 segmentos de tamanho 4 com relação ao tempo de amostragem do computador quântico, nesse caso, esse valor é de 0,2 ns, que resulta em um pulso de 227,33 ns. A frequência de Rabi foi limitada em $2\pi \times 8.5 \times 10^6 / \sqrt{2}$ para *I* e *Q*.

Diferente do pulso square, que é constante, o pulso otimizado pelo Boulder Opal tem uma atuação diferente com relação ao movimento do vetor de estado na esfera de Bloch. A figura 5.25 mostra uma simulação da evolução temporal dos componentes *x*, *y* e *z* da esfera de Bloch para ambos os pulsos de 227,33 ns. É possível perceber que o vetor estado do pulso square tem um movimento direto, enquanto o pulso otimizado realiza diversas movimentações na esfera de Bloch com o intuito de desviar o processo de erro de *dephasing*.

A figura 5.26 mostra os resultados do experimento para o pulso gerado pelo Boulder Opal. O erro acumulado com a frequência padrão é bastante semelhante ao pulso square de mesma duração. No entanto, esse pulso aparentemente foi mais afetado pelos desvios na frequência do que o pulso square, com erros acumulados maiores para as duas frequências em torno da padrão. Acredita-se que para essa duração dos pulsos não há muito a ser melhorado com relação à *dephasing*. Os resultados com o pulso square mais longo já

Figura 5.24 – Pulso de 227.33 ns gerado pelo Boulder Opal resistente à *dephasing*.

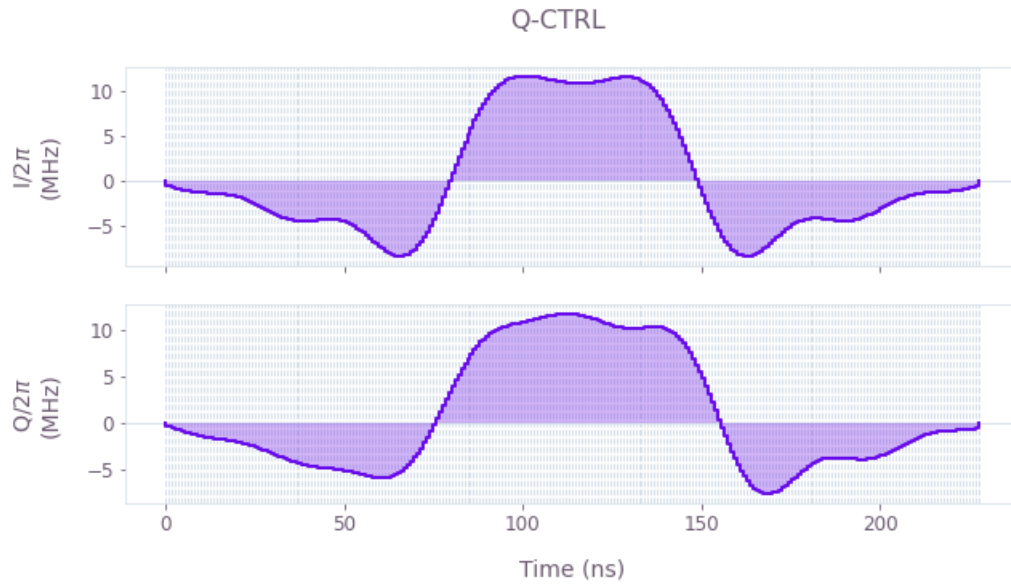
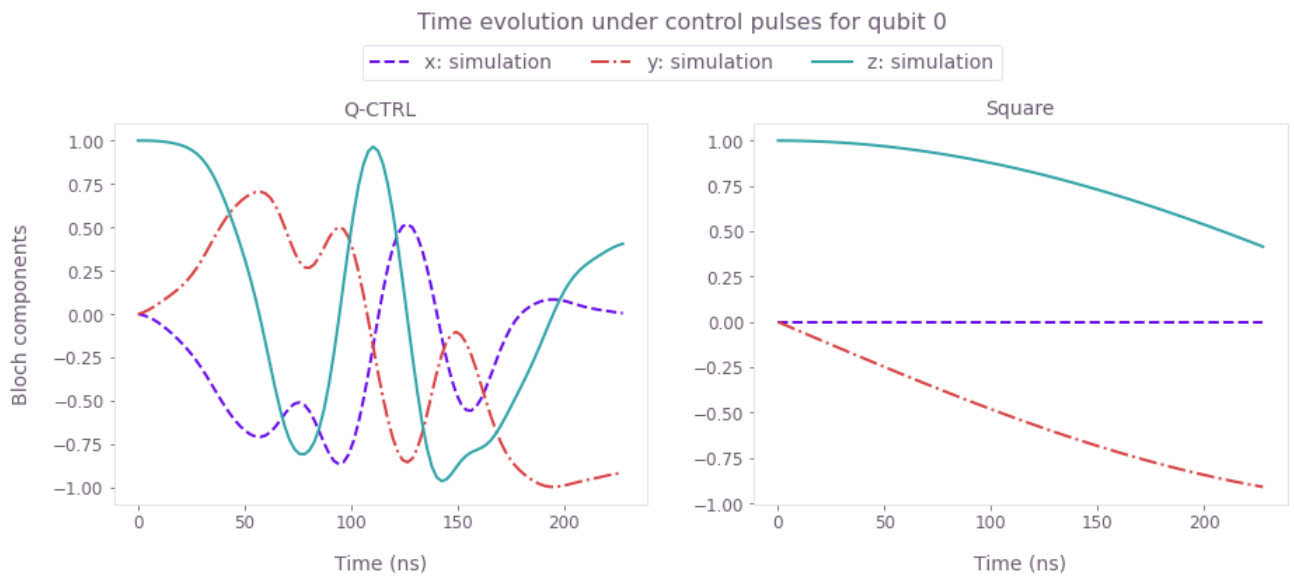
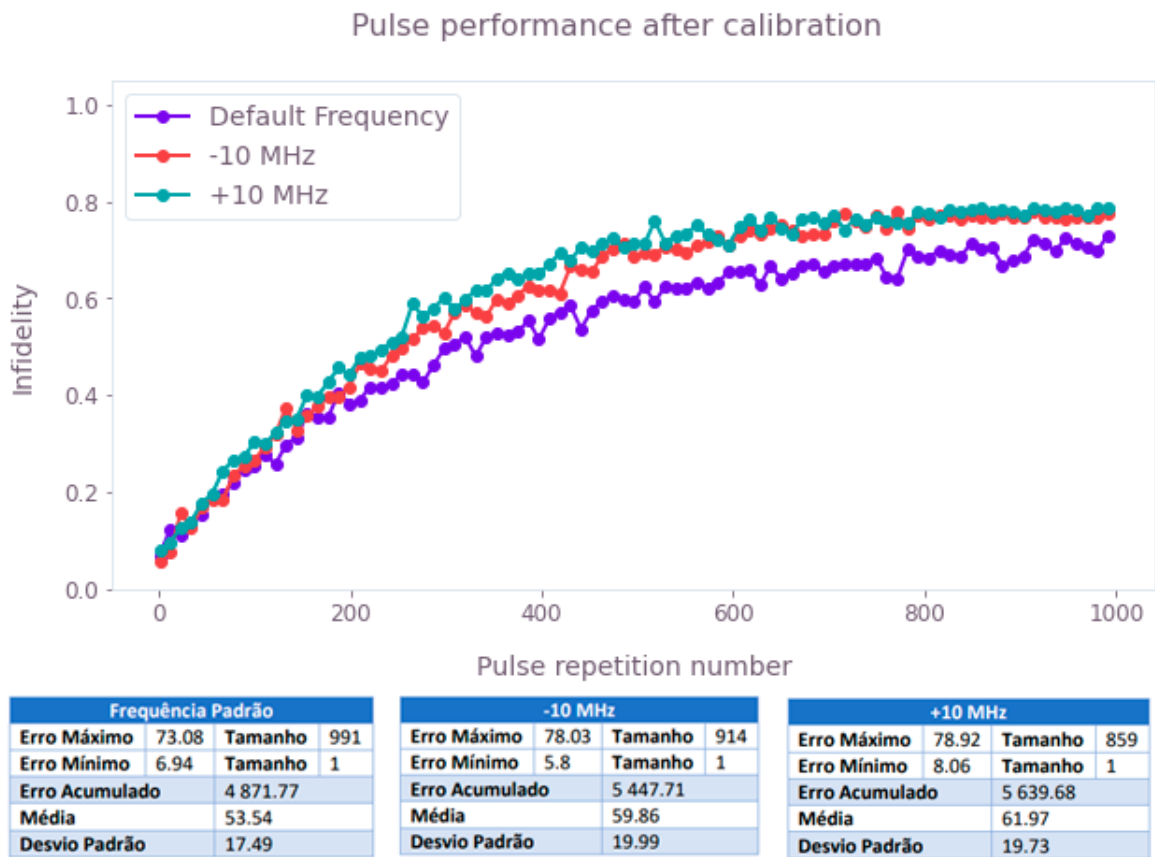


Figura 5.25 – Simulações da evolução temporal dos pulsos square e otimizado de 227,33 ns para implementação da porta lógica R_x .



havia apresentado bons resultados. Sendo assim, a geração de um pulso resistente à *dephasing* se faz necessário à medida em que são geradas portas lógicas mais curtas, que são mais afetadas por esse tipo de erro. Todavia, o cálculo de otimização para geração do pulso resistente à *dephasing* do Boulder Opal para portas lógicas mais curtas não obteve resultados muito bons.

Figura 5.26 – Infidelidade do pulso resistente à *dephasing* de 227,33 ns aplicado na frequência padrão, com -10 MHz e $+10\text{ MHz}$.



6 CONSIDERAÇÕES FINAIS

Este trabalho introduziu os conceitos de controle quântico em computadores quânticos supercondutores para a mitigação de erros em processos de computação quântica. Também foram apresentados trabalhos relacionados com soluções de otimizações circuitais e Quantum Error Correction Codes (QECCs) na computação quântica tolerante a falhas. A apresentação desses conceitos fundamentais é importante dado que, por ser um tema significativamente novo, ainda há poucos materiais mais introdutórios conciliando teoria e prática. Foram realizados diversos testes na implementação do autômato finito quântico MO1QFA para a solução do problema do módulo, onde foram consideradas as implementações padrão com e sem otimizações circuitais e utilizando portas lógicas quânticas personalizadas construídas a partir de experimentos em hardware em nível de pulsos de micro-ondas. Nesse sentido, foram exploradas diferentes formas de calibrações, técnicas de otimização da frequência do qubit e resistência a erros de dephasing para diferentes portas lógicas.

As máquinas quânticas atuais se mostraram ainda bastante ruidosas. É perceptível que existe um ótimo trabalho em otimizações circuitais e transpilação do circuito, onde são mantidas portas lógicas devidamente calibradas e que são bastante robustas, mas nem sempre implementam as operações desejadas. É por esse motivo que a implementação de algoritmos quânticos mais complexos ainda demanda a utilização de técnicas personalizadas de controle quântico e correção de erros. Assim, a linha de raciocínio de manter um conjunto de portas lógicas quânticas universais específicas onde os circuitos são traduzidos para elas pode não ser tão interessante quanto realizar a transpilação direcionada ao próprio hardware, porém, para isso, é necessário manter uma boa caracterização do hardware.

As ferramentas oferecidas pelo Q-CTRL Boulder Opal se mostraram bastante úteis nesse processo de manutenção de portas lógicas e calibração do hardware quântico. Com elas, foi possível gerar portas lógicas quânticas personalizadas e devidamente otimizadas de uma forma bastante prática. Entretanto, os manuais são voltados para um público com um conhecimento maior sobre o hardware quântico e não são muito úteis como materiais introdutórios.

Com relação aos testes realizados, as melhores portas lógicas foram as mais curtas mesmo que tenham sido mais afetadas por erros, a simplicidade das portas lógicas square também se mostrou interessante na implementação do autômato. As calibrações *fine-tuned* e autônoma das portas lógicas também são importantes, com elas, é possível reduzir consideravelmente as infidelidades melhorando a tradução para os valores de amplitude. A frequência dos pulsos, no entanto, não obteve grandes influências nos resultados, tanto em sua calibração quanto durante os testes de indução de dephasing, a frequência padrão oferecida pelo Qiskit já conseguiu bons resultados.

Ainda há muito a se fazer na área de mitigação de erros de computadores quânticos. Técnicas de controle quântico até então se fazem necessárias na execução de algoritmos. A transpilação direta do circuito para o hardware considerando as especificidades das portas lógicas em diferentes localizações do circuito pode ser uma alternativa viável nessa área, assim como considerações a respeito do comportamento do circuito para prever possíveis erros e ajustar os pulsos das portas lógicas.

No domínio da implementação de autômatos finitos quânticos, a investigação de soluções para outros problemas pode ser interessante, assim como a investigação de outros modelos de autômatos. A simplicidade dos autômatos é uma característica importante para a identificação da influência de erros nos processos, caracterizações podem ser úteis para tentar realizar um mapeamento das causas dos erros na execução dos algoritmos como base para uma possível adequação dos pulsos que implementam as portas lógicas.

REFERÊNCIAS

AAKASH. **Pauli Exclusion Principle**. [*S.l.: s.n.*], 2022. Acesso em 12 de julho de 2022. Disponível em: <https://byjus.com/jee/pauli-exclusion-principle/>.

AMDAHL, Gene M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *In: SPRING JOINT COMPUTER CONF.*, p. 483–485.

BAUM, Yuval *et al.* Experimental Deep Reinforcement Learning for Error-Robust Gate-Set Design on a Superconducting Quantum Computer. **PRX Quantum**, v. 2, n. 4, 2021.

CALDERBANCK, A. R.; SHOR, Peter W. Good Quantum Error-Correcting Codes Exist. **Physical Review A**, v. 54, n. 2, p. 1098–1105, 1996.

CARVALHO, Andre R. R. *et al.* Error-Robust Quantum Logic Optimization Using a Cloud Quantum Computing Interface. **Physical Review Applied**, v. 15, n. 6, 2021.

CERN. **Superconductivity**. [*S.l.: s.n.*], 2022. Acesso em 12 de julho de 2022. Disponível em: <https://home.cern/science/engineering/superconductivity>.

CHEN, Zijun *et al.* Measuring and Suppressing Quantum State Leakage in a Superconducting Qubit. **Physical Review Letters**, American Physical Society, v. 116, 2016. ISSN 0031-9007. DOI: 10.1103/PhysRevLett.116.020501. Disponível em: <https://doi.org/10.1103/PhysRevLett.116.020501>.

CHURCH, Alonzo. A Note on the Entscheidungsproblem. v. 1, n. 1, p. 40–41, mar. 1936.

DENNARD, Robert H. *et al.* IEEE Journal of Solid-State Circuits. SC-9, n. 5, p. 256–268, out. 1974.

DIRAC, P. A. M. A New Notation for Quantum Mechanics. **Mathematical Proceedings of the Cambridge Philosophical Society**, v. 35, n. 3, p. 416–418, 1939.

ENDO, Suguru; BENJAMIN, Simon C.; LI, Ying. Practical Quantum Error Mitigation for Near-Future Applications. **Physical Review X**, v. 8, n. 3, 2018.

FACULTY, Caltech's. **What is Quantum Physics?** [*S.l.: s.n.*], 2022. Acesso em 27 de maio de 2022. Disponível em: <https://scienceexchange.caltech.edu/topics/quantum-science-explained/quantum-physics>.

FEYNMAN, Richard P. Simulating Physics with Computers. **International Journal of Theoretical Physics**, v. 21, n. 6/7, p. 467–488, 1982.

FU, X. *et al.* A control microarchitecture for fault-tolerant quantum computing. **Microprocessors and Microsystems**, v. 70, p. 21–30, 2019. ISSN 0141-9331. DOI: <https://doi.org/10.1016/j.micpro.2019.06.011>. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933119303333>.

GERTLER, Jeffrey M. *et al.* Protecting a bosonic qubit with autonomous quantum error correction. **Nature**, v. 590, p. 243–248, 2021. DOI: <https://doi.org/10.1038/s41586-021-03257-0>. Disponível em: <https://doi.org/10.1038/s41586-021-03257-0>.

GILL, Sukhpal Singh *et al.* Quantum computing: A taxonomy, systematic review and future directions. **Software: Practice and Experience**, v. 52, n. 1, p. 66–114, 2021.

GUO, Qihao *et al.* Testing a quantum error-correcting code on various platforms. **Science Bulletin**, v. 66, n. 1, p. 29–35, 2021. ISSN 2095-9273. DOI: <https://doi.org/10.1016/j.scib.2020.07.033>. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2095927320305168>.

HENNESSY, John L.; PATTERSON, David A. **Computer Architecture: A Quantitative Approach**. 6. ed. Boston, USA: Morgan Kaufmann, 2019.

IBM. **Calibrating Qubits with Qiskit Pulse**. [*S.l.: s.n.*], 2022. Acesso em 16 de julho de 2022. Disponível em: <https://qiskit.org/textbook/ch-quantum-hardware/calibrating-qubits-pulse.html>.

_____. **Página Inicial**. [*S.l.: s.n.*], 2022. Acesso em 8 de julho de 2022. Disponível em: <https://quantum-computing.ibm.com/>.

_____. **Qiskit**. [*S.l.: s.n.*], 2022. Acesso em 16 de julho de 2022. Disponível em: <https://qiskit.org/>.

_____. **Qiskit Pulse**. [*S.l.: s.n.*], 2022. Acesso em 16 de julho de 2022. Disponível em: <https://qiskit.org/documentation/apidoc/pulse.html>.

JILL, John. Computation Complexity of Probabilistic Turing Machines. v. 6, n. 4, p. 675–695, dez. 1977.

JO, Hanlae; SONG, Yunheung; AHN, Jaewook. Leakage Suppression by Ultrafast Composite Pulses. **Optics Express**, Optica Publishing Group, v. 27, 2019. DOI: <https://doi.org/10.1364/OE.27.003944>.

KAYE, Phillip; LAFLAMME, Raymond; MOSCA, Michele. **An Introduction to Quantum Computing**. Great Britain: Oxford University Press, 2007.

KRANTZ, P. *et al.* A quantum engineer's guide to superconducting qubits. **Applied Physics Reviews**, AIP Publishing, v. 6, 2 2019. DOI: <https://doi.org/10.1063/1.5089550>.

MASLOV, Dmitri; NAM, Yunseong; KIM, Jungsang. An Outlook of Quantum Computing. **Proceedings of the IEEE**, v. 107, n. 1, p. 5–10, jan. 2018.

MCEWEN, M. *et al.* Removing Leakage-Induced Correlated Errors in Superconducting Quantum Error Correction. **Nature Communications**, v. 12, 2021.

MOOIJ, Hans. **Superconducting Quantum Bits**. [S.l.: s.n.], 2004. Acesso em 11 de julho de 2022. Disponível em: <https://physicsworld.com/a/superconducting-quantum-bits/>.

MOORE, Cristopher; CRUTCHFIELD, James P. Quantum automata and quantum grammars. **Theoretical Computer Science**, Elsevier, v. 237, n. 1-2, p. 275–306, 2000.

MOORE, Gordon E. Cramming More Components onto Integrated Circuits. v. 38, n. 8, abr. 1965.

_____. Progress In Digital Integrated Electronics. *In*: INTERNATIONAL ELECTRON DEVICES MEETING, p. 11–13.

MURALI, Prakash *et al.* Formal constraint-based compilation for noisy intermediate-scale quantum systems. **Microprocessors and Microsystems**, v. 66, p. 102–112, 2019. ISSN 0141-9331. DOI: <https://doi.org/10.1016/j.micpro.2019.02.005>. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933118302710>.

NAUTRUP, Hendrik Poulsen *et al.* Optimizing Quantum Error Correction Codes with Reinforcement Learning. **Quantum**, Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, v. 3, p. 215, dez. 2019. ISSN 2521-327X. DOI: [10.22331/q-2019-12-16-215](https://doi.org/10.22331/q-2019-12-16-215). Disponível em: <https://doi.org/10.22331/q-2019-12-16-215>.

NIELSEN, Michael A.; CHUANG, Isaac L. **Quantum Computation and Quantum Information**. New York, USA: Cambridge University Press, 2010.

O'MALLEY, Peter James Joyce. **Superconducting Qubits: Dephasing and Quantum Chemistry**. 2016. Tese (Doutorado) – University of California.

PALITTAPONGARNPIM, Pantita *et al.* Learning in quantum control: High-dimensional global optimization for noisy quantum dynamics. **Neurocomputing**, v. 268, p. 116–126, 2017. Advances in artificial neural networks, machine learning and computational intelligence. ISSN 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2016.12.087>.

Disponível em:

<https://www.sciencedirect.com/science/article/pii/S0925231217307531>.

POLLACHINI, Giovani Goraiebe. *Computação Quântica: Uma Abordagem para Esudantes de Graduação em Ciências Exatas*, 2018.

PORTUGAL, Renato *et al.* *Uma Introdução à Computação Quântica*. **SBMAC**, 2012.

POSTLER, Lukas *et al.* *Demonstration of Fault-Tolerant Universal Quantum Gate Operations*. **Nature**, v. 605, p. 675–680, 2022.

Q-CTRL. **Designing Noise-Robust Single-Qubit Gates for IBM Qiskit**.

Disponível em: <https://docs.q-ctrl.com/boulder-opal/application-notes/designing-noise-robust-single-qubit-gates-for-ibm-qiskit>. (accessed: 14.06.2022).

_____. **Página Inicial**. [*S.l.: s.n.*], 2022. Acesso em 8 de julho de 2022. Disponível em: <https://q-ctrl.com/>.

_____. **Q-CTRL Boulder Opal**. [*S.l.: s.n.*], 2022. Acesso em 15 de setembro de 2022. Disponível em: <https://boulder.q-ctrl.com/>.

QCTRL. **How to Automate Calibration of Control Hardware**. [*S.l.: s.n.*], 2022. Acesso em 16 de julho de 2022. Disponível em: <https://docs.q-ctrl.com/boulder-opal/user-guides/how-to-automate-calibration-of-control-hardware>.

QISKIT. **Transpiler Passes and Pass Manager**. [*S.l.: s.n.*]. Acesso em 20 de dezembro de 2022. Disponível em: https://qiskit.org/documentation/tutorials/circuits_advanced/04_transpiler_passes_and_passmanager.html.

ROFFE, Joschka. *Quantum error correction: an introductory guide*. **Contemporary Physics**, Taylor Francis, v. 60, n. 3, p. 226–245, 2019. DOI: 10.1080/00107514.2019.1667078. eprint: <https://doi.org/10.1080/00107514.2019.1667078>. Disponível em: <https://doi.org/10.1080/00107514.2019.1667078>.

SAROVAR, Mohan *et al.* *Detecting crosstalk errors in quantum information processors*. **Quantum**, Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, v. 4, p. 321, set. 2020. ISSN 2521-327X. DOI: 10.22331/q-2020-09-11-321. Disponível em: <https://doi.org/10.22331/q-2020-09-11-321>.

SAY, AC Cem; YAKARYILMAZ, Abuzer. *Quantum finite automata: A modern introduction*. *In: COMPUTING with New Resources*. [*S.l.*]: Springer, 2014. P. 208–222.

SHOR, Peter W. *Quantum Computing*. Extra Volume, n. 1, p. 467–486, 1998.

SILVA, Wagner Jorcuvich Nunes da. Uma Introdução à Computação Quântica, 2018.

SIPSER, Michael. **Introduction to the Theory of Computation**. 3. ed. Boston, USA: Cengage Learning, 2012.

STEINBRUCH, Alfredo; WINTERLE, Paulo. **Álgebra Linear**. [S.l.]: Pearson, 1995.

SUTTER, Paul. **What is a Superconductor?** [S.l.: s.n.], 2021. Acesso em 12 de julho de 2022. Disponível em: <https://www.livescience.com/superconductor>.

TEAM, PennyLane dev. **Quantum Computing with Superconducting Qubits**. [S.l.: s.n.], 2022. Acesso em 11 de julho de 2022. Disponível em: https://pennylane.ai/qml/demos/tutorial_sc_qubits.html.

TURING, A. M. On Computable Number, with an Application to the Entscheidungsproblem. s2-42, n. 1, p. 230–265, 1936.

ZUREK, Wojciech H. Decoherence and the Transition from Quantum to Classical. **Los Alamos Science**, n. 27, 2002.

APÊNDICE A – EXPERIMENTO DE RABI

Este notebook faz uma apresentação prática do experimento de Rabi, que é necessário para que se possa construir portas lógicas quânticas em nível de pulsos de micro-ondas. Esse experimento consiste em testar o hardware quântico com o objetivo de avaliar seu comportamento com relação a determinados pulsos. Os pulsos de computadores quânticos supercondutores estão na faixa de frequência de micro-ondas e um valor padrão já é fornecido pelo Qiskit, esse experimento busca encontrar a velocidade com que ocorrem rotações na esfera de Bloch com relação a determinados valores de amplitude dos pulsos. Sabendo disso, é possível construir portas lógicas para realizar qualquer operação de diferentes formas e em diferentes intervalos de tempo.

Espera-se que o leitor já esteja familiarizado com os conceitos básicos de computação quântica, conhecimentos básicos sobre o Qiskit também são bem vindos. Também é necessário possuir uma conta na IBM Quantum (IBM, 2022b). Os pacotes python necessários são o qiskit, numpy, jsonpickle, scipy e matplotlib.

Este notebook é baseado no material fornecido pela ferramenta Boulder Opal, da Q-CTRL, disponível em (QCTRL, 2022).

A.1 AUTENTICAÇÃO

O primeiro passo é autenticar-se com uma conta da IBM Quantum, escolher o provedor e o backend. Neste caso, será utilizado o computador Nairobi, de 7 qubits.

```

1 # Importar qiskit
2 from qiskit import IBMQ
3
4 # Ativar conta com seu token
5 IBMQ.enable_account("SEU TOKEN AQUI")
6 # Seleção de provedor
7 provider = IBMQ.get_provider("ibm-q")
8 # Seleção de computador quântico
9 backend = provider.get_backend("ibm_nairobi")
10
11 # Obtém informações sobre o computador quântico
12 backend_defaults = backend.defaults()
13 backend_config = backend.configuration()

```

A.2 EXECUTAR EXPERIMENTO

Com o usuário devidamente autenticado e computador quântico escolhido, o próximo passo é definir os parâmetros do experimento.

Como já dito anteriormente, precisamos avaliar as velocidades com que acontecem rotações na esfera de Bloch com relação à determinadas amplitudes. Dessa forma, é preciso definir um intervalo de amplitudes para executar os experimentos, nesse caso, serão realizados 10 experimentos, cada um deles com um determinado valor de amplitude no intervalo $[0.005, 0.2]$. Alguns computadores quânticos podem precisar de amplitudes menores e alguns não precisam de amplitudes maiores que 0.2, pode ser necessário executar esta etapa mais de uma vez nesses casos, mas o intervalo definido funciona bem para a grande maioria dos computadores.

Pulsos com amplitudes maiores tendem a realizar mais rotações na esfera de Bloch em um certo intervalo de tempo, assim, os intervalos de tempo para a aplicação dos pulsos são definidos proporcionais às amplitudes dos pulsos, ou seja, intervalos de tempo maiores para pulsos com amplitudes menores, com o objetivo de obter resultados mais semelhantes graficamente, o que facilita na obtenção das frequências de Rabi. Além disso, são definidos alguns pontos nesse intervalo de tempo onde são realizadas medições para avaliar o comportamento do estado quântico com o tempo.

Após a definição desses parâmetros, é preciso executar os experimentos, o que pode levar um tempo.

```
1 # Importação de bibliotecas
2 import qiskit.pulse as pulse
3 import numpy as np
4 from qiskit.tools.monitor import job_monitor
5 import qiskit as qs
6 from qiskit.pulse import Play
7
8 qubit = 0 # Seleção do qubit
9 # Obtém tempo dt: intervalo de tempo mínimo para a aplicação dos pulsos
10 dt = backend_config.dt
11 num_shots_per_point = 1024 # Número de execuções por experimento
12
13 # Definição dos valores de amplitude
14 pulse_amp_array = np.linspace(0.005, 0.2, 10)
15 # Estimativa de frequência de Rabi para o pulso de menor amplitude
16 min_rabi_estimated = 0.5e6
17 # Estimativa de frequência de Rabi para o pulso de maior amplitude
18 max_rabi_estimated = 45e6
19
20 # Retorna múltiplo mais próximo de uma base
21 def get_closest_multiple_of(value, base_number):
```

```

22     return int(value + base_number/2) - (int(value + base_number/2) %
    ↪ base_number)
23
24 # Calcula tempos com base nas estimativas de frequências de Rabi
25 pulse_times = 2 / np.linspace(min_rabi_estimated, max_rabi_estimated, 10) /
    ↪ dt
26 pulse_times = np.array([get_closest_multiple_of(t, 16) for t in
    ↪ pulse_times]) / 16
27 pulse_times = np.array([np.unique(np.linspace(4, t, 20, dtype=int)) for t
    ↪ in pulse_times])
28 pulse_times *= 16
29
30 # Atualiza propriedades do computador quântico com servidor
31 backend.properties(refresh=True)
32
33 # Cria canal onde os pulsos são executados
34 drive_chan = pulse.DriveChannel(qubit)
35
36 # Criação dos experimentos
37 rabi_programs_dic_I = {}
38 for idx, pulse_amplitude in enumerate(pulse_amp_array):
39     rabi_schedules_I = []
40     for duration_pulse in pulse_times[idx]:
41         # Cria circuito
42         circ = qs.QuantumCircuit(1, 1)
43         circ.x(0) # Porta X utilizada para construir o pulso
44         circ.measure(0, 0)
45
46         # Criação do pulso tipo gaussian_square
47         drive_pulse = qs.pulse.library.gaussian_square(
48             duration=duration_pulse, # Duração
49             sigma=1,
50             amp=pulse_amplitude, # Amplitude
51             risefall=1,
52             name=f"square_pulse_{duration_pulse}",
53         )
54
55         # Criação do programa do pulso
56         schedule = pulse.Schedule(name=str(duration_pulse))
57         schedule |= (
58             Play(drive_pulse, pulse.DriveChannel(qubit))

```

```

59         << schedule.duration
60     )
61
62     # Porta lógica X recebe programa do pulso
63     circ.add_calibration('x', [qubit], schedule)
64
65     # Adiciona circuito na lista para ser executado posteriormente
66     rabi_schedules_I.append(circ)
67
68     rabi_programs_dic_I[pulse_amplitude] = rabi_schedules_I
69
70 # Execução dos experimentos
71 rabi_oscillations_results = []
72 for idx, pulse_amplitude in enumerate(pulse_amp_array):
73     # Criação do job
74     job = backend.run(rabi_programs_dic_I[pulse_amplitude],
75                     meas_level=2,
76                     meas_return="single",
77                     shots=num_shots_per_point)
78     job_monitor(job)
79     rabi_results = job.result(timeout=120)
80     rabi_values = []
81
82     # Extração dos resultados
83     time_array = pulse_times[idx] * dt
84     for i, time_idx in enumerate(pulse_times[idx]):
85         counts = rabi_results.get_counts(i)
86         excited_pop = 0
87         for bits, count in counts.items():
88             excited_pop += count if bits[::-1][qubit] == "1" else 0
89         rabi_values.append(excited_pop / num_shots_per_point)
90     rabi_oscillations_results.append(rabi_values)

```

A.3 EXTRAIR RESULTADOS

Os resultados do experimento de Rabi são as frequências de Rabi, que correspondem às frequências com que acontecem rotações na esfera de Bloch para pulsos com diferentes amplitudes. Assim, ao definir um pulso, escolhemos um intervalo de tempo e as rotações que devem ser feitas com relação aos eixos x e y, isso então corresponde às frequências de Rabi que, por meio dos resultados deste experimento, são traduzidas para os valores de amplitude.

Para obter as frequências de Rabi, é necessário realizar diversas medições no intervalo de tempo definido para acompanhar o estado quântico. Essa evolução do estado é visualizada em um gráfico que tem formato típico da seguinte função:

$$y = A \times \cos^2(2\pi \times rabi_freq \times x + \phi)$$

A partir dela, é feito um ajuste aos parâmetros com o objetivo de obter seu período (*rabi_freq*), que corresponde à frequência de Rabi. Por esse motivo, pode ser necessário alterar os intervalos dos parâmetros para o ajuste ser feito adequadamente aos pontos do experimento.

```

1 # Importação de bibliotecas
2 from scipy.optimize import curve_fit
3 import jsonpickle
4
5 # Definição da função que realiza os ajustes
6 def fit_function_bounds(x_values, y_values, function, bound_values):
7     fitparams, conv = curve_fit(function,
8                                 x_values, y_values,
9                                 bounds=bound_values)
10    y_fit = function(x_values, *fitparams)
11    return fitparams, y_fit
12
13 # Faz o ajuste à função cosseno para todos as amplitudes
14 rabi_calibration_exp_I = []
15 fit_parameters_list = []
16 for idx, pulse_amplitude in enumerate(pulse_amp_array):
17     rabi_values = rabi_oscillations_results[idx]
18     time_array = pulse_times[idx] * dt
19
20     fit_parameters, y_fit = fit_function_bounds(
21         time_array,
22         rabi_values,
23         lambda x, A, rabi_freq, phi: A
24         * np.cos(2 * np.pi * rabi_freq * x + phi) ** 2,
25         (
26             # Valores mínimos e máximos do ajuste
27             # para os parâmetros X, rabi_freq e phi
28             # Podem necessitar de ajustes
29             [0.8, np.abs(pulse_amplitude * 7 * 1e7), -2],
30             [1.05, np.abs(pulse_amplitude * 12 * 1e7), 2],
31         ),

```

```

32 )
33 print(fit_parameters)
34 rabi_calibration_exp_I.append(fit_parameters[1]*2)
35 fit_parameters_list.append(fit_parameters)

```

Para verificar o resultado dos ajustes, é necessário analisar se a função está adequada para os pontos do gráfico da figura A.1. Para isso, basta executar o código abaixo modificando a variável i para analisar cada um dos experimentos dos 10 valores de amplitudes. Se a função não estiver adequada, é necessário alterar os parâmetros e executar o código anterior novamente.

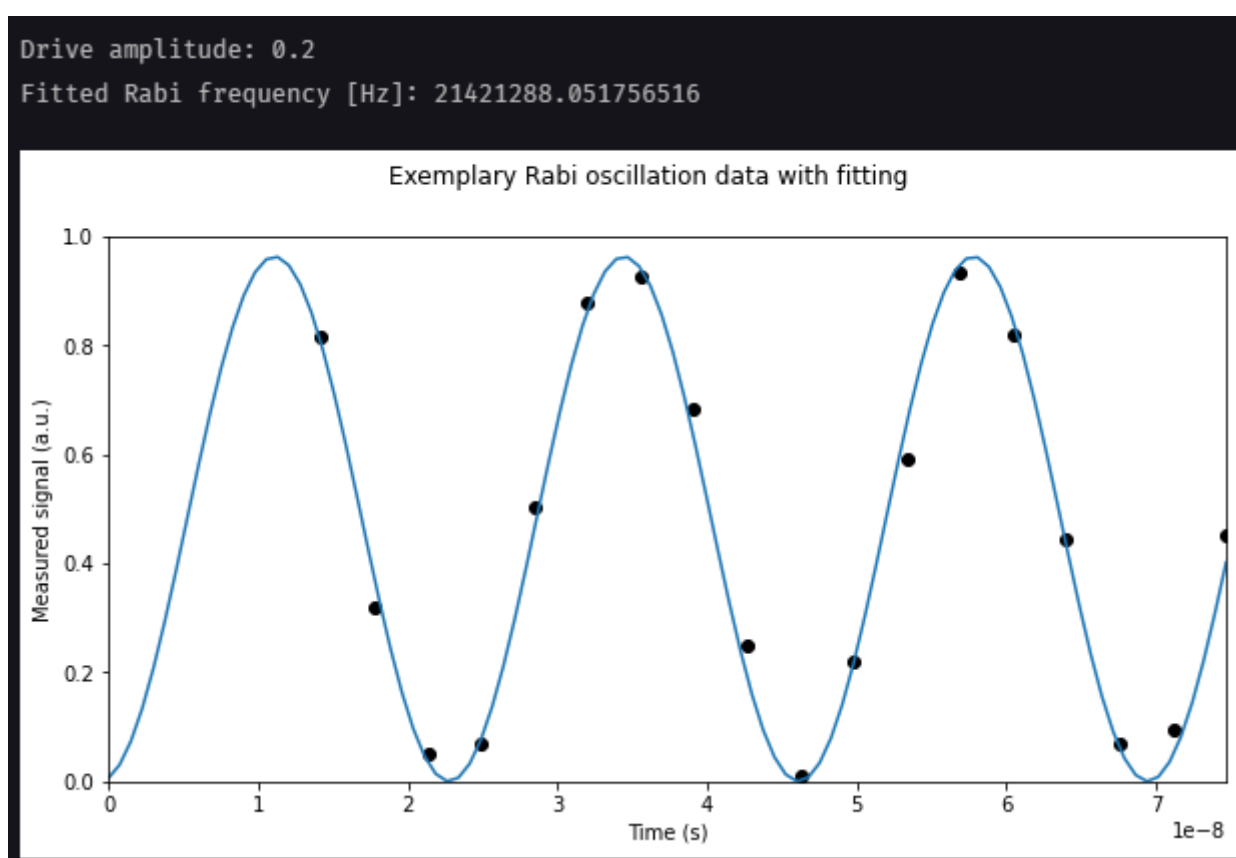


Figura A.1 – Oscilação de Rabi para um Pulso de Amplitude 0.2

```

1 # Importação de bibliotecas
2 import matplotlib.pyplot as plt
3
4 # Índice do experimento
5 # Modificar para visualizar o ajuste para todos os valores de amplitude
6 i = 9
7

```

```

8 time_array = pulse_times[i] * dt
9 print("Drive amplitude:", pulse_amp_array[i])
10 print("Fitted Rabi frequency [Hz]:", fit_parameters_list[i][1])
11
12 # Monta o gráfico
13 fig = plt.figure(figsize=(10, 5))
14 fig.suptitle("Exemplary Rabi oscillation data with fitting")
15 plt.xlabel("Time (s)")
16 plt.ylabel("Measured signal (a.u.)")
17 plt.scatter(time_array,
18             np.real(rabi_oscillations_results[i]),
19             color="black")
20 plt.xlim(0, time_array[-1])
21 plt.ylim(0, 1)
22 plot_times = np.linspace(0, time_array[-1], 100)
23 plt.plot(
24     plot_times,
25     fit_parameters_list[i][0]
26     * np.cos(2 * np.pi * fit_parameters_list[i][1] * plot_times
27             + fit_parameters_list[i][2]) ** 2,
28 )
29 plt.show()

```

Após realizar o ajuste devidamente, é preciso salvar o experimento.

```

1 #Importação de Bibliotecas
2 import time
3
4 # Função para salvar variáveis
5 def save_var(file_name, var):
6     time_file = time.strftime("%Y%m%d-%H%M%S")
7     # Save a single variable to a file using jsonpickle
8     f = open(f"{file_name}_{time_file}", "w+")
9     to_write = jsonpickle.encode(var)
10    f.write(to_write)
11    f.close()
12
13 # Salva calibração
14 save_var("resources/rabi_calibration_nairobi_qubit_0",
15         ↪ rabi_calibration_exp_I)
16 # Salva parâmetros da função

```

```

16 save_var("resources/fit_parameters_nairobi_qubit_0", fit_parameters)
17 # Salva valores dos experimentos
18 save_var("resources/rabi_values_nairobi_qubit_0", rabi_values)

```

O resultado final do experimento de Rabi são as frequências de Rabi para diferentes amplitudes, conforme mostra o gráfico da figura A.2. Amplitudes maiores realizam rotações na esfera de Bloch mais rapidamente, assim, o valor do eixo y corresponde especificamente a frequência com que acontecem duas rotações na esfera de Bloch para um pulso com amplitude do eixo x . É realizado uma interpolação desses valores para poder obter uma função contínua que dada uma frequência de Rabi, retorna o valor de amplitude necessário para o pulso.

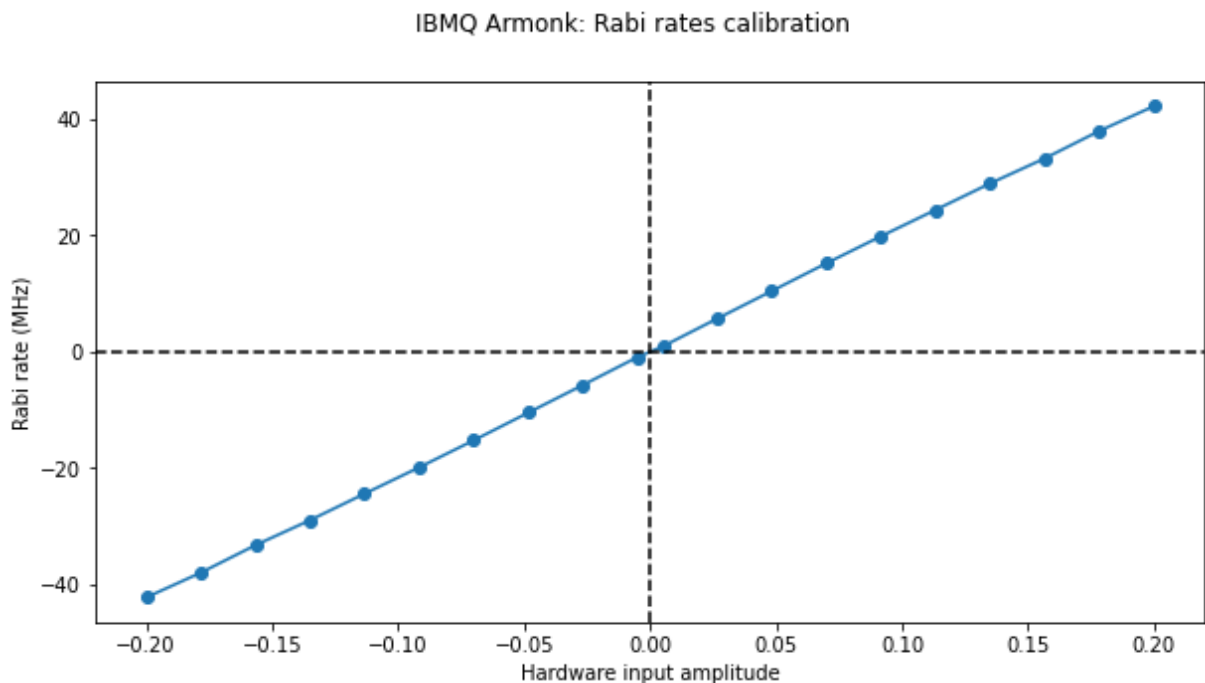


Figura A.2 – Resultado do Experimento de Rabi

```

1 from scipy import interpolate
2
3 # Definição de valores de amplitude para interpolação
4 amplitude_interpolated_list = np.linspace(-0.2, 0.2, 100)
5 # Adiciona amplitudes negativas
6 full_pulse_amp_array = np.concatenate(
7     (
8         -pulse_amp_array[::-1],
9         pulse_amp_array

```

```
10     )
11 )
12 # Adiciona frequências de Rabi negativas
13 full_rabi_calibration_exp_I = np.concatenate(
14     (
15         -np.asarray(rabi_calibration_exp_I[::-1]),
16         np.asarray(rabi_calibration_exp_I)
17     )
18 )
19
20 # Definição da função que converte amplitudes para frequências de Rabi
21 f_amp_to_rabi = interpolate.interp1d(full_pulse_amp_array,
22                                     full_rabi_calibration_exp_I)
23 rabi_interpolated_exp_I = f_amp_to_rabi(amplitude_interpolated_list)
24
25 # Definição da função que converte frequências de Rabi para amplitudes
26 f_rabi_to_amp = interpolate.interp1d(
27     rabi_interpolated_exp_I, amplitude_interpolated_list
28 )
29
30 # Monta o gráfico
31 fig = plt.figure(figsize=(10, 5))
32 fig.suptitle("IBMQ Armonk: Rabi rates calibration")
33 plt.xlabel("Hardware input amplitude")
34 plt.ylabel("Rabi rate (MHz)")
35 plt.scatter(full_pulse_amp_array, full_rabi_calibration_exp_I * 1e-6)
36 plt.plot(amplitude_interpolated_list, rabi_interpolated_exp_I * 1e-6)
37 plt.axvline(0, color="black", linestyle="dashed")
38 plt.axhline(0, color="black", linestyle="dashed")
39 plt.show()
```

APÊNDICE B – CONSTRUÇÃO DE PORTAS LÓGICAS RESISTENTES À RUÍDOS COM Q-CTRL BOUDER OPAL

A ferramenta Boulder Opal da Q-CTRL permite a construção de portas lógicas quânticas otimizadas na forma de pulsos de micro-ondas a serem aplicadas no hardware quântico que implementa um qubit. A descrição de portas lógicas personalizadas é capaz de melhorar significativamente a performance de circuitos quânticos e é uma excelente alternativa para aqueles interessados em melhorar o resultado de seus experimentos. A Q-CTRL oferece uma série de ferramentas que facilitam e automatizam alguns processos importantes, além de oferecer modelagens, materiais e códigos prontos. Este notebook mostra como descrever pulsos personalizados do tipo square ou gerados pelo Boulder Opal resistentes à erros de dephasing e controle de amplitude.

Este material é uma adaptação mais detalhada e simples do notebook oficial disponível gratuitamente na documentação da Q-CTRL (Q-CTRL, s.d.). Espera-se que o leitor esteja familiarizado com os conceitos de computação quântica, conhecimentos sobre o Qiskit também são bem vindos. Também é necessário possuir uma calibração de Rabi por meio do notebook do experimento de Rabi. Para executá-lo, é necessário possuir uma conta na IBM Quantum (IBM, 2022b) e uma conta no Boulder Opal (Q-CTRL, 2022b). As bibliotecas python necessárias são o jsonpickle, matplotlib, numpy, scipy, qctrl e qiskit.

B.1 AUTENTICAÇÃO

O primeiro passo é autenticar-se com uma conta da IBM Quantum, escolher o provedor e o backend. Neste caso, será utilizado o computador Nairobi, de 7 qubits.

```
1 # Importar qiskit
2 from qiskit import IBMQ
3
4 # Importar Q-CTRL
5 from qctrl import Qctrl
6
7 # Login na Q-CTRL
8 qctrl = Qctrl()
9
10 # Ativar conta IBMQ com seu token
11 IBMQ.enable_account("SEU TOKEN AQUI")
12 # Seleção de provedor
13 provider = IBMQ.get_provider("ibm-q")
14 # Seleção de computador quântico
15 backend = provider.get_backend("ibm_nairobi")
```

```

16
17 # Obtém informações sobre o computador quântico
18 backend_defaults = backend.defaults()
19 backend_config = backend.configuration()

```

B.2 MODELAGEM DE PULSOS COM BOULDER OPAL

Uma porta lógica é descrita por um Hamiltoniano que descreve a evolução de estado do sistema quântico:

$$H(t) = (1 + \beta_\gamma(t)) H_c(t) + \eta(t)\sigma_z,$$

onde $\beta_\gamma(t)$ representa um processo de flutuação no controle de amplitude do pulso, $\eta(t)$ representa um processo estocástico de dephasing e $H_c(t)$ é o Hamiltoniano controlador definido por:

$$\begin{aligned} H_c(t) &= \frac{1}{2} (\gamma^*(t)\sigma_- + \gamma(t)\sigma_+) \\ &= \frac{1}{2} (I(t)\sigma_x + Q(t)\sigma_y). \end{aligned}$$

onde $\gamma(t) = I(t) + iQ(t)$ é um valor complexo dependente do tempo que representa a forma de onda do pulso e σ_k , $k = x, y, z$, são as matrizes de Pauli.

Os valores I e Q se referem às frequências reais e complexas da decomposição IQ dos pulsos conhecida na engenharia elétrica, para a computação quântica, essas frequências correspondem às rotações realizadas nos eixos x e y da esfera de Bloch.

Para exemplificar, este notebook mostra a criação de uma porta lógica X square e gerada pela Q-CTRL.

Antes de iniciar a criação dos pulsos, é preciso instanciar algumas variáveis.

```

1 # Importação de bibliotecas
2 import numpy as np
3 import jsonpickle
4 import time
5
6 # Definição de algumas portas lógicas e matriz de Pauli
7 sigma_z = np.array([[1.0, 0.0], [0.0, -1.0]], dtype=complex)
8 sigma_x = np.array([[0.0, 1.0], [1.0, 0.0]], dtype=complex)
9 sigma_y = np.array([[0.0, -1.0j], [1.0j, 0.0]], dtype=complex)
10 sigma_p = np.array([[0.0, 0.0], [1.0, 0.0]], dtype=complex)
11 X_gate = np.array([[0.0, 1.0], [1.0, 0.0]], dtype=complex)
12 X90_gate = np.array([[1.0, -1j], [-1j, 1.0]], dtype=complex) / np.sqrt(2)

```

```

13 rabi_rotation = np.pi
14
15 # Obtém tempo dt: intervalo de tempo mínimo para a aplicação dos pulsos
16 dt = backend_config.dt
17
18 scheme_names = [] # Nome dos pulsos
19
20 # ----- Instanciação de variáveis para registrar parâmetros dos pulsos
21 # Quantidade de segmentos dos pulsos
22 number_of_segments = {}
23 # Quantidade de variáveis de otimização do pulso da Q-CTRL
24 number_of_optimization_variables = {}
25 # Limite de frequências
26 cutoff_frequency = {}
27 # Tamanho dos segmentos com relação à dt
28 segment_scale = {}
29 # Duração do pulso com relação à dt
30 duration_int = {}
31 # Duração do pulso
32 duration = {}
33
34 # Função para salvar variáveis
35 def save_var(file_name, var):
36     time_file = time.strftime("%Y%m%d-%H%M%S")
37     # Save a single variable to a file using jsonpickle
38     f = open(f"{file_name}_{time_file}", "w+")
39     to_write = jsonpickle.encode(var)
40     f.write(to_write)
41     f.close()

```

B.2.1 Criação do Pulso Square

Um pulso square é um pulso simples e de frequência de Rabi constante.

O primeiro passo para a definição do pulso square é escolher uma duração, de maneira geral, é mais interessante gerar pulsos mais rápidos para diminuir a duração do circuito, entretanto, pulsos com amplitudes maiores tendem a ser mais sensíveis a ruídos. Nesse caso, foi definido um pulso com um segmento de tamanho 1280 com relação ao tempo dt para uma melhor didática, que resulta em um pulso de $258.44ns$ para um $dt = 0.2$.

Depois disso, é preciso decidir qual a rotação desejada na esfera de Bloch. Para uma porta lógica X, é desejável uma rotação π com relação ao eixo x . Assim, a frequência

de Rabi para o valor I deve ser de $\pi/duration$, dado que desejamos obter π em um tempo $duration$.

```

1 square_controls = {}
2
3 number_of_segments["Square"] = 1      # Quantidade de segmentos do pulso
4 segment_scale["Square"] = 1280       # Tamanho dos segmentos com relação à
   ↪ dt
5 duration["Square"] = segment_scale["Square"] * dt # Duração do pulso
6
7 # Operação do pulso
8 square_pulse_value = np.array([rabi_rotation / duration["Square"]])
9
10 # Define valores I e Q
11 square_sequence = {
12     # Rotação com relação ao eixo x
13     "I": qctrl.utils.pwc_arrays_to_pairs(duration["Square"],
   ↪ square_pulse_value),
14     # Rotação com relação ao eixo y
15     "Q": qctrl.utils.pwc_arrays_to_pairs(duration["Square"],
   ↪ np.zeros([1])),
16 }
17 scheme_names.append("Square")
18
19 square_controls['Square'] = square_sequence

```

Após gerado o pulso, é interessante salvá-lo para poder ser utilizado em outros experimentos

```

1 save_var("resources/square_controls", square_controls)

```

B.2.2 Criação do Pulso da Q-CTRL

A criação do pulso otimizado pelo Boulder Opal é um pouco mais complexa, porém, mais automatizada.

O primeiro passo é a definição dos parâmetros para a otimização.

A frequência de Rabi máxima é definida por ω_{max} , valores maiores permitem uma melhor otimização, entretanto, variações muito grandes nas frequências de Rabi dos segmentos do pulso podem ser fontes de erros. Além disso, esse valor não pode ser estar acima dos limites da função do experimento de Rabi que será utilizada para traduzir o pulso para o computador quântico desejado.

A duração do pulso é definida pelo tamanho dos segmentos $segment_scale$ e o número de segmentos $number_of_segments$. Para este experimento, está sendo criado um pulso com 256 segmentos de tamanho 5 com relação à dt , assim, esse pulso tem a mesma duração do pulso square com um segmento de tamanho 1280, onde cada segmento possui uma frequência de Rabi constante.

```

1 # Frequência de Rabi máxima
2 omega_max = 2 * np.pi * 8.5e6
3 I_max = omega_max / np.sqrt(2)
4 Q_max = omega_max / np.sqrt(2)
5
6 # ----- Definição dos Parâmetros
7 # Número de variáveis de otimização
8 number_of_optimization_variables["Q-CTRL"] = 64
9 # Quantidade de segmentos do pulso
10 number_of_segments["Q-CTRL"] = 256
11 # Tamanho do segmento com relação à dt
12 segment_scale["Q-CTRL"] = 5
13 # Limite de frequências
14 cutoff_frequency["Q-CTRL"] = omega_max * 2
15 # Duração do pulso
16 duration["Q-CTRL"] = number_of_segments["Q-CTRL"] *
   → segment_scale["Q-CTRL"] * dt

```

Após a definição dos parâmetros, é preciso criar o grafo por meio da qual é realizada a otimização em Cloud do pulso considerando os parâmetros definidos anteriormente.

Nesta etapa, é preciso decidir se o pulso considerará erros de controle de amplitude (*amplitude*), dephasing (*dephasing*), ambos ou nenhum.

Ao executar a otimização, é preciso estar atento ao resultado da otimização. Se ele não estiver próximo de 0, é necessário redefinir os parâmetros para considerar frequências de Rabi maiores, segmentos menores, ou mais segmentos.

```

1 robust_dephasing_controls = {}
2
3 graph = qctrl.create_graph() # Criação do grafo
4
5 # Definição das variáveis de otimização I e Q
6 I_values = graph.optimization_variable(
7     count=number_of_optimization_variables["Q-CTRL"],
8     lower_bound=-I_max,
9     upper_bound=I_max,

```

```

10 )
11 Q_values = graph.optimization_variable(
12     count=number_of_optimization_variables["Q-CTRL"],
13     lower_bound=-Q_max,
14     upper_bound=Q_max,
15 )
16
17 # Âncora termina em zero com aumento/diminuição da amplitude
18 time_points = np.linspace(
19     -1.0, 1.0, number_of_optimization_variables["Q-CTRL"] + 2
20 )[1:-1]
21 envelope_function = 1.0 - np.abs(time_points)
22 I_values = I_values * envelope_function
23 Q_values = Q_values * envelope_function
24
25 # Criação dos sinais I e Q
26 I_signal = graph.pwc_signal(values=I_values, duration=duration["Q-CTRL"])
27 Q_signal = graph.pwc_signal(values=Q_values, duration=duration["Q-CTRL"])
28
29 # Aplica filtro e rediscrretiza os sinais I e Q
30 I_signal = graph.utils.filter_and_resample_pwc(
31     pwc=I_signal,
32     cutoff_frequency=cutoff_frequency["Q-CTRL"],
33     segment_count=number_of_segments["Q-CTRL"],
34     name="I",
35 )
36 Q_signal = graph.utils.filter_and_resample_pwc(
37     pwc=Q_signal,
38     cutoff_frequency=cutoff_frequency["Q-CTRL"],
39     segment_count=number_of_segments["Q-CTRL"],
40     name="Q",
41 )
42
43 # Criação do Hamiltoniano
44 I_term = I_signal * graph.pauli_matrix("X") / 2.0
45 Q_term = Q_signal * graph.pauli_matrix("Y") / 2.0
46
47 # Hamiltoniano controlador
48 control_hamiltonian = I_term + Q_term
49
50 # Criação de  $\eta_\gamma(t)$ , descriptor do processo de erro de dephasing

```

```

51 dephasing = graph.constant_pwc_operator(
52     duration=duration["Q-CTRL"],
53     operator=graph.pauli_matrix("Z") / 2.0 / duration["Q-CTRL"],
54 )
55
56 # Criação de \beta_\gamma(t), descritor do processo de erro de controle de
57   \leftrightarrow amplitude
58 amplitude = control_hamiltonian
59
60 # Cria medida de infidelidade
61 infidelity = graph.infidelity_pwc(
62     # Hamiltoniano que descreve os pulsos
63     hamiltonian=control_hamiltonian,
64     # Operador requisitado, neste caso, a porta X
65     target=graph.target(X_gate),
66     # Opcional, utilizar 'amplitude', 'dephasing' ou ambos
67     noise_operators=[dephasing],
68     name="infidelity",
69 )
70
71 # Calcula otimização
72 result = qctrl.functions.calculate_optimization(
73     graph=graph, cost_node_name="infidelity", output_node_names=["I", "Q"]
74 )
75 print(f"Cost: {result.cost}") # Imprime custo da otimização
76
77 robust_dephasing_controls["Q-CTRL"] = result.output
78 scheme_names.append("Q-CTRL")

```

Após gerado o pulso, é interessante salvá-lo para poder ser utilizado em outros experimentos

```

1 save_var("resources/robust_dephasing_controls", robust_dephasing_controls)

```

B.2.3 Visualização dos Pulsos

Após a criação dos pulsos Square e da Q-CTRL, podemos visualizá-los em um gráfico com os valores de I e Q com relação ao tempo (figura B.1).

É possível perceber que o pulso da Q-CTRL, diferentemente do pulso Square, não possui valores constantes.

```

1 # Importação de bibliotecas
2 import matplotlib.pyplot as plt
3 from qctrlvisualizer import get_qctrl_style, plot_controls
4
5 plt.style.use(get_qctrl_style()) # Utilizar estilo da Q-CTRL
6
7 # Monta os gráficos
8 gate_schemes = {**robust_dephasing_controls, **{"Square":
  ↳ square_sequence}}
9 for scheme_name, gate in gate_schemes.items():
10     plot_controls(gate)
11     plt.suptitle(scheme_name)
12     plt.show()

```

Para visualizar a ação dos pulsos na esfera de Bloch, o Boulder Opal oferece um simulador que simula a evolução de estado dos componentes x , y e z da esfera de Bloch para um determinado pulso.

O gráfico da figura B.2 mostra que o pulso gerado pelo Boulder Opal não é linear como o pulso Square, o vetor estado realiza movimentações diferentes para otimizar o pulso.

```

1 # Definição da função que realiza a simulação de um pulso atuando em um
  ↳ qubit
2 def simulation_coherent(control, time_samples):
3     graph = qctrl.create_graph()
4
5     drive_I = graph.pwc(*qctrl.utils.pwc_pairs_to_arrays(control["I"]))
6     drive_Q = graph.pwc(*qctrl.utils.pwc_pairs_to_arrays(control["Q"]))
7     drive = drive_I + 1j * drive_Q
8     duration = np.sum(drive.durations)
9
10    hamiltonian = graph.hermitian_part(drive * graph.pauli_matrix("P"))
11
12    initial_state_vector = np.array([1.0, 0.0])
13
14    sample_times = np.linspace(0, duration, time_samples)
15    unitaries = graph.time_evolution_operators_pwc(
16        hamiltonian=hamiltonian, sample_times=sample_times
17    )
18    states = unitaries @ initial_state_vector[:, None]
19

```

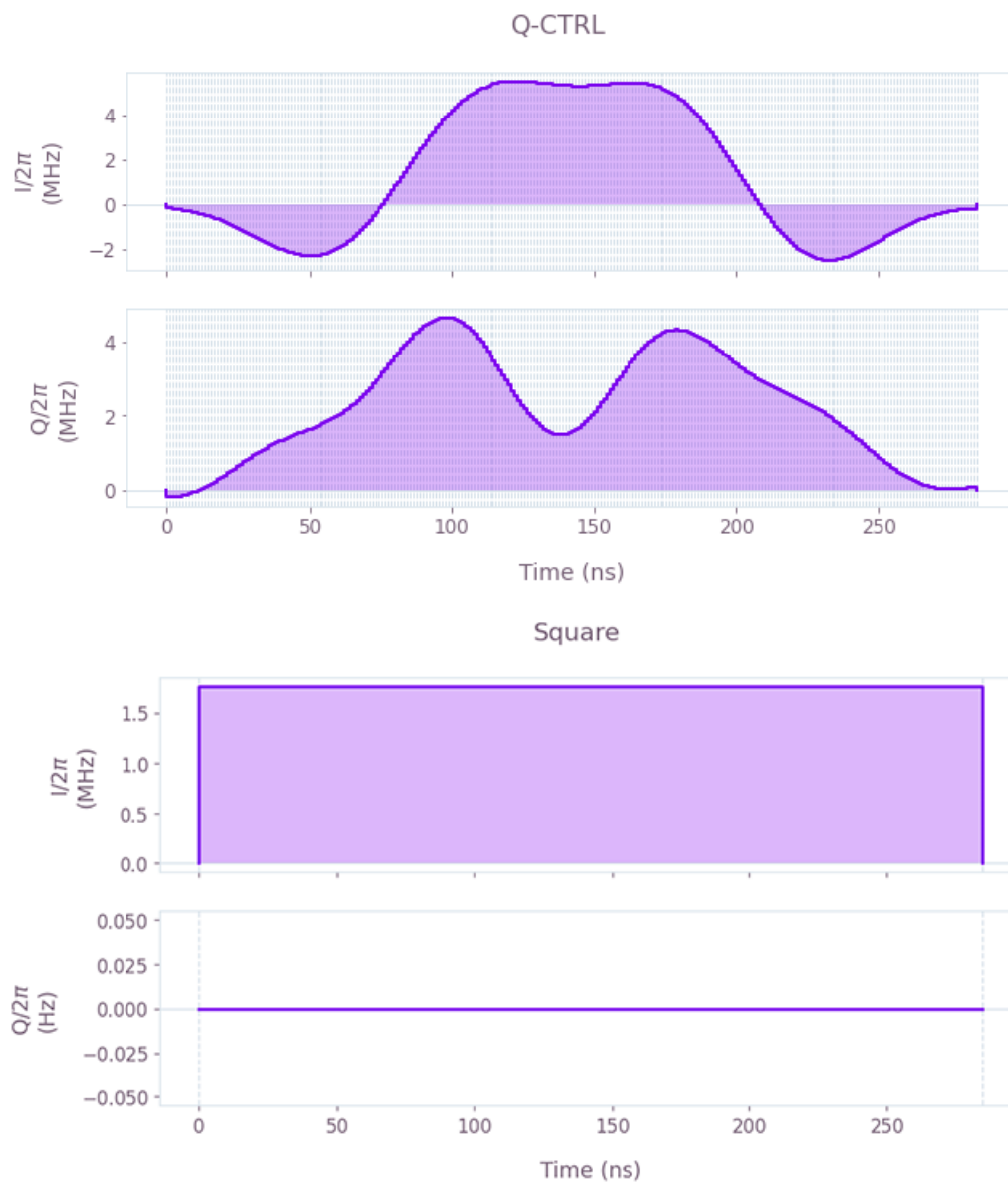


Figura B.1 – Decomposição IQ dos Pulsos Q-CTRL e Square

```

20     for name in ["X", "Y", "Z"]:
21         graph.real(
22             graph.expectation_value(states[...], 0),
23             ↪ graph.pauli_matrix(name)),
24             name=name
25         )
26     result = qctrl.functions.calculate_graph(
27         graph=graph, output_node_names=["X", "Y", "Z"]
28     )
29 
```

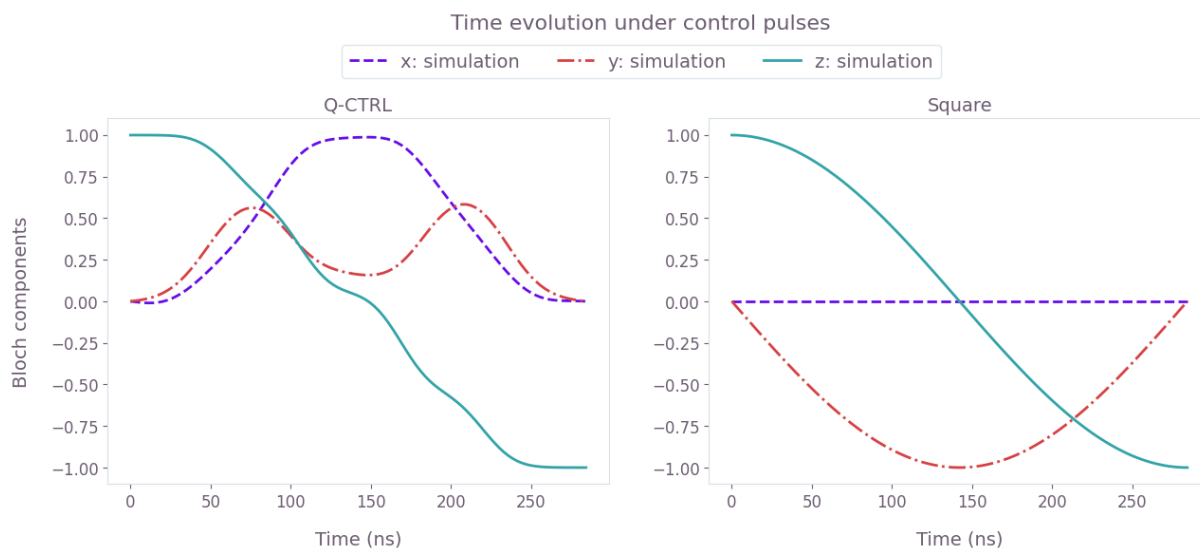


Figura B.2 – Evolução Temporal Simulada dos Componentes da Esfera de Bloch para os Pulsos Q-CTRL e Square

```

30     return {k.lower(): v["value"] for k, v in result.output.items()},
        ↪ sample_times
31
32     # Executa simulação dos pulsos
33     simulated_bloch = {}
34     gate_times = {}
35     for scheme_name, control in gate_schemes.items():
36         simulated_bloch[scheme_name], gate_times[scheme_name] =
        ↪ simulation_coherent(
37             control, 100
38         )
39
40     # Imprime resultados
41     fig, axs = plt.subplots(1, len(gate_schemes.keys()), figsize=(15, 5))
42     fig.suptitle(f"Time evolution under control pulses", y=1.1)
43
44     bloch_lines = {"x": "--", "y": "-.", "z": "-"}
45     bloch_basis = ["x", "y", "z"]
46     nano = 1e-9
47
48     for idx, scheme_name in enumerate(gate_schemes.keys()):
49         ax = axs[idx]
50         for meas_basis in bloch_basis:
51             ax.plot(

```

```

52         gate_times[scheme_name] / nano,
53         simulated_bloch[scheme_name][meas_basis],
54         ls=bloch_lines[meas_basis],
55         label=f"{meas_basis}: simulation",
56     )
57     ax.set_title(scheme_name)
58     ax.set_xlabel("Time (ns)")
59
60     axs[0].set_ylabel("Bloch components")
61     hs, ls = axs[0].get_legend_handles_labels()
62     fig.legend(handles=hs, labels=ls, loc="center", bbox_to_anchor=(0.5, 1.0),
63               ↪ ncol=3)
64     plt.show()

```

B.3 VALIDAÇÃO DOS PULSOS

Com os pulsos gerados, é interessante testá-los em um computador quântico para verificar se ele executa conforme o esperado pela simulação.

Primeiramente é necessário possuir um arquivo com a calibração do experimento de Rabi que pode ser feita por meio do notebook do experimento de Rabi, o caminho do arquivo deve ser inserido no local apresentado no código. É preciso tomar cuidado com os intervalos de amplitude escolhidos para o experimento de Rabi, este código está preparado para receber uma calibração com amplitudes no intervalo $[0.005, 0.2]$, caso o experimento tenha sido feito com outro intervalo, é necessário alterar esse valor nos locais indicados no código.

Após execução, é possível visualizar algo semelhante ao representado na figura B.3

```

1  # ----- Importação de bibliotecas
2  from scipy import interpolate
3  from qiskit import QuantumCircuit
4  from qiskit.pulse import Play, Schedule, DriveChannel, MeasureChannel
5  from qiskit.pulse.library import Waveform
6  from qiskit.tools.monitor import job_monitor
7  from qctrlvisualizer import QCTRL_STYLE_COLORS
8
9  # Definição de função que lê variáveis de um arquivo
10 def load_var(file_name):
11     f = open(file_name, "r+")
12     encoded = f.read()

```



```
13     decoded = jsonpickle.decode(encoded)
14     f.close()
15     return decoded
16
17
18     # ----- Obtém dados do experimento de Rabi
19     rabi_calibration_exp_I = np.array(
20         load_var("!!! CAMINHO DO EXPERIMENTO DE RABI AQUI !!!")
21     )
22     rabi_calibration_exp_I = np.concatenate(
23         (-rabi_calibration_exp_I[::-1], rabi_calibration_exp_I)
24     )
25
26     # Intervalo de amplitudes do experimento de Rabi
27     pulse_amp_array = np.linspace(0.005, 0.2, 10)
28     pulse_amp_array = np.concatenate((-pulse_amp_array[::-1],
29     ↪ pulse_amp_array))
30
31     # Definição da função que converte amplitudes para frequências de Rabi
32     f_amp_to_rabi = interpolate.interp1d(pulse_amp_array,
33     ↪ rabi_calibration_exp_I)
34
35     # Intervalo de amplitudes para interpolação
36     amplitude_interpolated_list = np.linspace(-0.2, 0.2, 100)
37     rabi_interpolated_exp_I = f_amp_to_rabi(amplitude_interpolated_list)
38
39     # Definição da função que converte frequências de Rabi para amplitudes
40     f_rabi_to_amp = interpolate.interp1d(
41         rabi_interpolated_exp_I, amplitude_interpolated_list
42     )
43
44     # Definição das formas de onda em valores de amplitude
45     waveform = {}
46     for scheme_name, control in gate_schemes.items():
47         I_values = qctrl.utils.pwc_pairs_to_arrays(control["I"])[1] / (2 *
48         ↪ np.pi)
49         Q_values = qctrl.utils.pwc_pairs_to_arrays(control["Q"])[1] / (2 *
50         ↪ np.pi)
51         A_I_values = np.repeat(f_rabi_to_amp(I_values),
52         ↪ segment_scale[scheme_name])
```

```

48     A_Q_values = np.repeat(f_rabi_to_amp(Q_values),
    ↪     segment_scale[scheme_name])
49     waveform[scheme_name] = A_I_values + 1j * A_Q_values
50
51     # ----- Parâmetros do experimento
52     ibm_evolution_times = {}
53     times_int = {}
54     pulse_evolution_program = {}
55     time_min = 64     # Tempo mínimo para a primeira medição
56     time_step = 64   # Intervalo de tempo para demais medições
57
58     # Definição de tempos para medições
59     for scheme_name in gate_schemes.keys():
60         time_max = int(segment_scale[scheme_name] *
    ↪         number_of_segments[scheme_name])
61         times_int[scheme_name] = np.arange(time_min, time_max + time_step,
    ↪         time_step)
62         ibm_evolution_times[scheme_name] = times_int[scheme_name] * dt
63
64     num_shots_per_point = 2048     # Número de execuções por experimento
65
66     # Atualiza propriedades do computador quântico com servidor
67     backend.properties(refresh=True)
68
69     # Qubit para aplicar os pulsos
70     qubit = 0
71
72     drive_chan = DriveChannel(qubit)     # Canal para aplicação do pulso
73     meas_chan = MeasureChannel(qubit)    # Canal de medição do qubit
74
75     # Mapa de portas lógicas do computador quântico
76     inst_sched_map = backend_defaults.instruction_schedule_map
77
78     # Executa experimentos
79     for scheme_name in gate_schemes.keys():
80         evolution_schedules = []
81         # Um circuito para obter cada componente do vetor estado
82         for meas_basis in bloch_basis:
83             # Um circuito para cada intervalo de tempo
84             for time_idx in times_int[scheme_name]:
85                 # Cria circuito

```

```

86     circ = QuantumCircuit(1, 1,
87         ↪ name=f"Basis_{meas_basis}s_duration_{time_idx}")
88     circ.x(0) # Porta X utilizada para construir o pulso
89     circ.measure(0, 0)
90
91     schedule =
92         ↪ Schedule(name=f"Basis_{meas_basis}s_duration_{time_idx}")
93
94     # Criação do pulso do tipo Waveform
95     schedule += Play(Waveform(waveform[scheme_name][:time_idx]),
96         ↪ drive_chan)
97
98     if meas_basis == "x":
99         schedule += inst_sched_map.get("u2", [0], P0=0.0,
100             ↪ P1=np.pi)
101     if meas_basis == "y":
102         schedule += inst_sched_map.get("u2", [0], P0=0.0, P1=np.pi
103             ↪ / 2)
104
105     # Porta lógica X recebe programa do pulso
106     circ.add_calibration('x', [qubit], schedule)
107
108     # Adiciona circuito na lista para ser executado posteriormente
109     evolution_schedules.append(circ)
110
111     pulse_evolution_program[scheme_name] = evolution_schedules
112
113 # Executa os circuitos
114 evolution_exp_results = {}
115 for scheme_name in gate_schemes.keys():
116     job = backend.run(pulse_evolution_program[scheme_name],
117         meas_level=2,
118         meas_return="single",
119         shots=num_shots_per_point)
120     job_monitor(job)
121     evolution_exp_results[scheme_name] = job.result(timeout=120)
122
123 # Extrai resultados dos circuitos
124 evolution_results_ibm = {}
125 for scheme_name in gate_schemes.keys():
126     evolution_basis = {}

```

```

122     for meas_basis in bloch_basis:
123         evolution_exp_data = np.zeros(len(times_int[scheme_name]))
124         for idx, time_idx in enumerate(times_int[scheme_name]):
125             counts = evolution_exp_results[scheme_name].get_counts(
126                 f"Basis_{meas_basis}s_duration_{time_idx}"
127             )
128             excited_pop = 0
129             for bits, count in counts.items():
130                 excited_pop += count if bits[::-1][qubit] == "1" else 0
131             evolution_exp_data[idx] = excited_pop / num_shots_per_point
132             evolution_basis[meas_basis] = evolution_exp_data
133         evolution_results_ibm[scheme_name] = evolution_basis
134
135     # Salva resultados
136     save_var("results/bloch_vectors_dephasing", evolution_results_ibm)
137
138     # Impressão dos resultados
139     fig, axs = plt.subplots(1, len(gate_schemes.keys()), figsize=(15, 5))
140     fig.set_figheight(5)
141     fig.set_figwidth(16)
142     fig.suptitle(f"Time evolution under control pulses for qubit {qubit}",
143                 ↪ y=1.15)
144
145     bloch_markers = {"x": "x", "y": "s", "z": "o"}
146     bloch_colors = {
147         "x": QCTRL_STYLE_COLORS[0],
148         "y": QCTRL_STYLE_COLORS[1],
149         "z": QCTRL_STYLE_COLORS[2],
150     }
151
152     for idx, scheme_name in enumerate(gate_schemes.keys()):
153         ax = axs[idx]
154         for meas_basis in bloch_basis:
155             ax.plot(
156                 gate_times[scheme_name] / nano,
157                 simulated_bloch[scheme_name][meas_basis],
158                 ls=bloch_lines[meas_basis],
159                 color=bloch_colors[meas_basis],
160                 label=f"{meas_basis}: Q-CTRL simulation",
161             )
162         ax.plot(

```

```

162         ibm_evolution_times[scheme_name] / nano,
163         1 - 2 * evolution_results_ibm[scheme_name][meas_basis],
164         bloch_markers[meas_basis],
165         color=bloch_colors[meas_basis],
166         label=f"{meas_basis}: IBM experiments",
167     )
168     ax.set_title(scheme_name)
169     ax.set_xlabel("Time (ns)")
170
171     axs[0].set_ylabel("Bloch components")
172     hs, ls = axs[0].get_legend_handles_labels()
173     fig.legend(handles=hs, labels=ls, loc="center", bbox_to_anchor=(0.5, 1.02),
174               ↪ ncol=3)
175     plt.show()
    
```

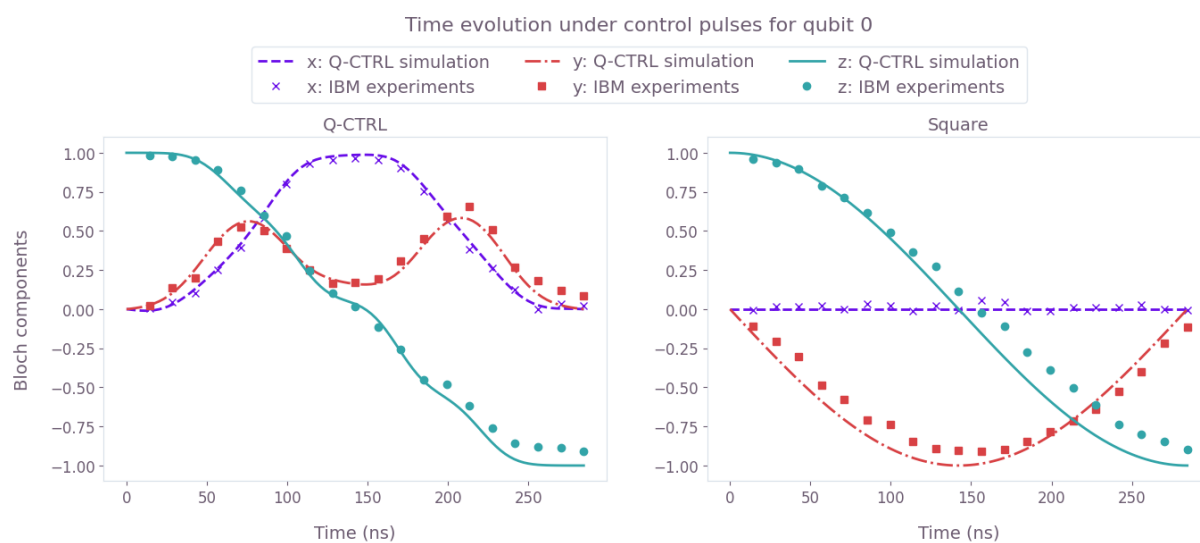


Figura B.3 – Evolução Temporal dos Componentes da Esfera de Bloch para os Pulsos Q-CTRL e Square

APÊNDICE C – CALIBRAÇÃO DE PORTAS LÓGICAS

O experimento de Rabi é uma boa estimativa do comportamento do hardware com relação à determinados pulsos, com ele já é possível obter bons resultados. No entanto, após a geração de uma porta lógica é interessante que ela seja calibrada de forma mais precisa para executar a operação desejada e não estar tão dependente de uma boa caracterização do hardware. Este notebook apresenta duas formas de aperfeiçoar a tradução de pulsos descritos em frequências de Rabi para valores de amplitude.

Este material é uma adaptação do material disponibilizado pela Q-CTRL (QCTRL, 2022). É necessário possuir uma calibração de Rabi e uma porta lógica gerada pelos demais notebooks do experimento de Rabi e construção de portas lógicas resistentes à ruídos com Q-CTRL Boulder Opal. Também é necessário possuir uma conta na IBM Quantum (IBM, 2022b) e uma conta no Boulder Opal (Q-CTRL, 2022b). Espera-se que o leitor esteja familiarizado com os conceitos de computação quântica, experimento de Rabi e construção de portas lógicas a nível de pulsos de micro-ondas, conhecimentos sobre o Qiskit também são bem vindos. As bibliotecas necessárias são o qiskit, qctrl, numpy, jsonpickle, scipy e matplotlib.

C.1 AUTENTICAÇÃO

O primeiro passo é autenticar-se com uma conta da IBM Quantum, escolher o provedor e o backend. Neste caso, será utilizado o computador Nairobi, de 7 qubits.

```
1 # Importar qiskit
2 from qiskit import IBMQ
3
4 # Importar Q-CTRL
5 from qctrl import Qctrl
6
7 # Login na Q-CTRL
8 qctrl = Qctrl()
9
10 # Ativar conta IBMQ com seu token
11 IBMQ.enable_account("SEU TOKEN AQUI")
12 # Seleção de provedor
13 provider = IBMQ.get_provider("ibm-q")
14 # Seleção de computador quântico
15 backend = provider.get_backend("ibm_nairobi")
16
17 # Obtém informações sobre o computador quântico
18 backend_defaults = backend.defaults()
```

```
19 backend_config = backend.configuration()
```

C.2 IMPORTAÇÃO DO EXPERIMENTO DE RABI

Para executar os experimentos no computador quântico escolhido, já é preciso possuir uma calibração de Rabi, cujo caminho do arquivo deve ser inserido no local indicado no código.

Também é necessário alterar os valores de amplitudes correspondentes ao experimento de Rabi utilizado. Neste caso, são 10 valores de 0.005 a 0.2.

```
1 # Importação de Bibliotecas
2 import numpy as np
3 import jsonpickle
4 from scipy import interpolate
5
6 # Definição de função que carrega variáveis de um arquivo
7 def load_var(file_name):
8     f = open(file_name, "r+")
9     encoded = f.read()
10    decoded = jsonpickle.decode(encoded)
11    f.close()
12    return decoded
13
14 # Importação dos dados do experimento de Rabi
15 rabi_calibration_exp_I = np.array(
16     load_var("ARQUIVO DA CALIBRAÇÃO")
17 )
18 rabi_calibration_exp_I = np.concatenate(
19     (-rabi_calibration_exp_I[::-1], rabi_calibration_exp_I)
20 )
21
22 # Intervalo de amplitudes do experimento de Rabi
23 pulse_amp_array = np.linspace(0.005, 0.2, 10)
24 pulse_amp_array = np.concatenate((-pulse_amp_array[::-1],
25     ↪ pulse_amp_array))
26
27 # Definição da função que converte amplitudes para frequências de Rabi
28 f_amp_to_rabi = interpolate.interp1d(pulse_amp_array,
29     ↪ rabi_calibration_exp_I)
30
31 # Intervalo de amplitudes para interpolação
```

```

30 amplitude_interpolated_list = np.linspace(-0.2, 0.2, 100)
31 rabi_interpolated_exp_I = f_amp_to_rabi(amplitude_interpolated_list)
32
33 # Definição da função que converte frequências de Rabi para amplitudes
34 f_rabi_to_amp = interpolate.interp1d(
35     rabi_interpolated_exp_I, amplitude_interpolated_list
36 )

```

C.3 IMPORTAÇÃO DO PULSO

Após a importação do experimento de Rabi, é preciso importar o pulso a ser calibrado. Para isso, basta inserir o caminho do arquivo onde o pulso está salvo no local indicado no código.

Ao executar o código, é possível visualizar o formato do pulso no gráfico da figura C.1. O exemplo consiste em uma porta X resistente a dephasing.

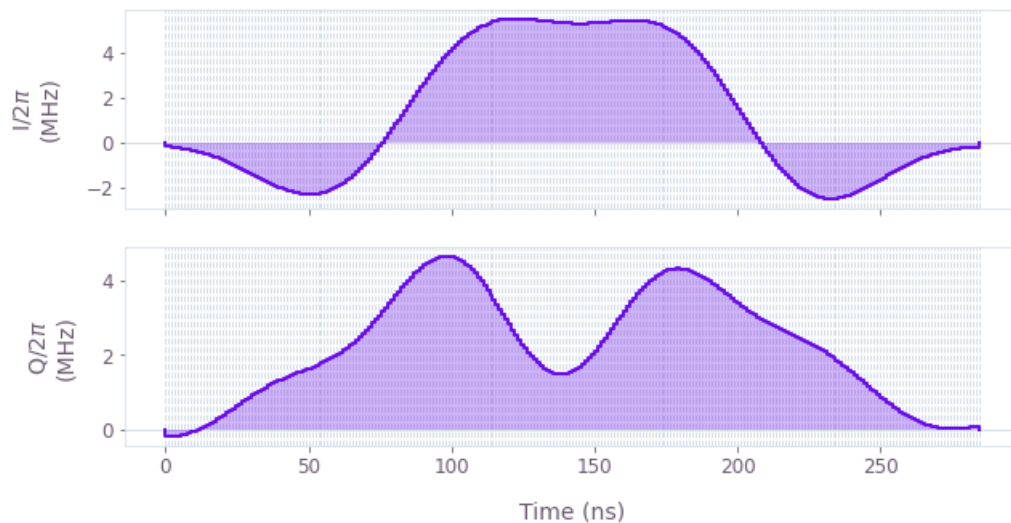


Figura C.1 – Porta X Resistente a Dephasing

```

1 # Importação de Bibliotecas
2 import matplotlib.pyplot as plt
3 from qctrlvisualizer import get_qctrl_style, plot_controls,
  ↪ plot_cost_history
4 plt.style.use(get_qctrl_style())
5
6 # Importação do Pulso
7 gate_schemes = load_var("CAMINHO DO ARQUIVO DO PULSO")
8
9 # Obtém tempo dt: intervalo de tempo mínimo para a aplicação dos pulsos

```



```
10 dt = backend_config.dt
11
12 # Extrai propriedades do pulso
13 number_of_segments = {}
14 duration_int = {}
15 duration = {}
16 segment_scale = {}
17 waveform = {}
18 for scheme_name, control in gate_schemes.items():
19     number_of_segments[scheme_name] = len(control["I"])
20     duration[scheme_name] = sum(values["duration"]
21                                 for values in control["I"])
22     duration_int[scheme_name] = int(round(duration[scheme_name] / dt))
23     segment_scale[scheme_name] = (
24         duration_int[scheme_name] / number_of_segments[scheme_name]
25     )
26     I_values = qctrl.utils.pwc_pairs_to_arrays(control["I"])[1]
27     I_values /= 2 / np.pi
28     Q_values = qctrl.utils.pwc_pairs_to_arrays(control["Q"])[1]
29     Q_values /= 2 / np.pi
30     A_I_values = np.repeat(f_rabi_to_amp(I_values),
31                             segment_scale[scheme_name])
32     A_Q_values = np.repeat(f_rabi_to_amp(Q_values),
33                             segment_scale[scheme_name])
34     waveform[scheme_name] = A_I_values + 1j * A_Q_values
35
36 # Invoca pulso para procedimento de calibração
37 pulse_duration = dt * waveform["Q-CTRL"].shape[0]
38 pulse_to_calibrate = {
39     "I": qctrl.utils.pwc_arrays_to_pairs(
40         pulse_duration, np.real(waveform["Q-CTRL"])
41     ),
42     "Q": qctrl.utils.pwc_arrays_to_pairs(
43         pulse_duration, np.imag(waveform["Q-CTRL"])
44     ),
45 }
46
47 # Monta o gráfico do pulso
48 for scheme_name, gate in gate_schemes.items():
49     plt.suptitle(scheme_name)
50     plot_controls(gate)
```

C.4 CALIBRAÇÃO DOS VALORES DE AMPLITUDE DO PULSO

O objetivo deste notebook é melhorar a tradução $(A_I, A_Q) \longleftrightarrow (I, Q)$ para uma determinada porta lógica. Até então, a tradução das frequências de Rabi para valores de amplitude dos pulsos é feita por meio dos resultados do experimento de Rabi, entretanto, esses resultados podem ser estimativas não tão precisas, ainda mais conforme vão perdendo sua validade. Nesse sentido, precisamos encontrar os fatores (S_{amp}, S_{rel}) na equação abaixo que minimizem a infidelidade do pulso.

$$\gamma(t) = S_{amp}(S_{rel}A_I(t) + iA_Q(t)).$$

Este notebook apresenta duas formas de se fazer isso. A primeira delas é denominada fine-tuned, que consiste em uma forma mais manual de fazer esse procedimento. A segunda, consiste na utilização de um agente autônomo do Boulder Opal que se propõe a encontrar os melhores valores de S_{amp} e S_{rel} em um looping fechado.

Antes de iniciar os experimentos, algumas definições básicas.

É importante alterar o valor esperado para os componentes x , y e z para o cálculo da infidelidade no local indicado no código.

```

1  # Importação de Bibliotecas
2  from qiskit.pulse import Play, Schedule, DriveChannel
3  from qiskit.pulse.library import Waveform
4  from qiskit.tools.monitor import job_monitor
5  from qiskit import QuantumCircuit
6  import time
7
8  # Função para salvar variáveis
9  def save_var(file_name, var):
10     time_file = time.strftime("%Y%m%d-%H%M%S")
11     # Save a single variable to a file using jsonpickle
12     f = open(f"{file_name}_{time_file}", "w+")
13     to_write = jsonpickle.encode(var)
14     f.write(to_write)
15     f.close()
16
17  qubit = 0    # Qubit para aplicação do pulso
18
19  bloch_basis = ["x", "y", "z"]
20
21  drive_chan = DriveChannel(qubit)    # Canal para aplicação do pulso
22

```

```

23 num_shots_per_point = 1024 # Número de execuções por experimento
24
25 # Mapa de portas lógicas do computador
26 inst_sched_map = backend_defaults.instruction_schedule_map
27
28 # Resultado esperado do operador para os componentes x, y e z
29 target_bloch = np.array([0, 0, -1])

```

C.4.1 Calibração Fine-tuned

A calibração fine-tuned é uma calibração mais interessante quando ainda não se tem muitas informações sobre o sistema quântico ou a própria técnica de calibração. Ela funciona de forma mais manual, permite visualizar os resultados com mais detalhes e tende a apresentar menos erros.

Aqui serão executados dois experimentos, um para obtenção de S_{rel} e outro para S_{amp} . Os experimentos consistem na definição de um intervalo de valores para testar os parâmetros e diferentes quantidades de repetições para o cálculo da infidelidade do vetor estado. Os melhores valores dos fatores são encontrados onde a infidelidade média das diferentes repetições é mínima.

C.4.1.1 Obtenção de S_{rel}

Para obter S_{rel} é necessário definir um intervalo de valores e diferentes quantidades de repetições do pulso.

O intervalo de valores deve ser algo próximo de 1. Neste caso, foram definidos 50 pontos entre 0.95 e 1.05.

Esses 50 valores serão testados para as repetições 49 e 101. Aqui é preciso prestar atenção para definir uma quantidade de pulsos que faça sentido, ou seja, para uma porta X , valem repetições $N = (1 + 2n)$, com $n \in \{0, 1 \dots\}$, caso contrário, o vetor estado não finalizará no local esperado para essa porta lógica.

```

1 # Repetições para cálculo de infidelidade
2 repetitions = np.array([49, 101])
3 # Valores para teste de S_rel
4 scalings = np.linspace(0.95, 1.05, 50)
5
6 # Monta circuitos
7 pulse_program = {}
8 for scheme_name in gate_schemes.keys(): # Para cada porta lógica
9     pulse_program[scheme_name] = {}
10     # Para cada base da esfera de Bloch (x, y e z)

```

```

11     for meas_basis in bloch_basis:
12         schedules = []
13         for repetition in repetitions: # Para cada repetição (111, 441)
14             for scaling_factor_I in scalings: # Para cada fator S_rel
15                 # Cria circuito
16                 circ = QuantumCircuit(1, 1,
17                 ↪ name=f"pi_pulse_scal_{scaling_factor_I}_reps_" +
18                 ↪ str(repetition) + "_basis_" + str(meas_basis))
19                 circ.x(0) # Porta X utilizada para construir o pulso
20                 circ.measure(0, 0)
21
22                 # Parametriza os valores de amplitude com S_rel
23                 Ival = np.real(waveform[scheme_name]) * scaling_factor_I
24                 Qval = np.imag(waveform[scheme_name])
25                 scaled_waveform = Ival + 1j * Qval
26
27                 # Criação do pulso do tipo Waveform
28                 control_pulse = Play(Waveform(scaled_waveform),
29                                     drive_chan)
30                 schedule = Schedule(
31                     name=f"pi_pulse_scal_{scaling_factor_I}_reps_"
32                     + str(repetition)
33                     + "_basis_"
34                     + str(meas_basis)
35                 )
36
37                 # Repete pulso pela quantidade de vezes definida
38                 for reps in np.arange(repetition):
39                     schedule += control_pulse
40
41                 # Aplica porta lógica para obter dos componentes x, y e z
42                 if meas_basis == "x":
43                     schedule += inst_sched_map.get("u2", [0],
44                                                     P0=0.0, P1=np.pi)
45                 if meas_basis == "y":
46                     schedule += inst_sched_map.get("u2", [0],
47                                                     P0=0.0, P1=np.pi / 2)
48
49                 # Porta lógica X recebe programa do pulso
50                 circ.add_calibration('x', [qubit], schedule)

```

```

50         # Adiciona circuito na lista para ser executado
51         schedules.append(circ)
52         pulse_program[scheme_name][meas_basis] = schedules
53
54     # Executa experimentos
55     exp_results = {}
56     for scheme_name in gate_schemes.keys():
57         exp_results[scheme_name] = {}
58         print("Control Scheme:", scheme_name)
59         for meas_basis in bloch_basis:
60             print("Measurement basis:", meas_basis)
61             job = backend.run(pulse_program[scheme_name][meas_basis],
62                             meas_level=2,
63                             meas_return="single",
64                             shots=num_shots_per_point)
65             print("Job id:", job.job_id())
66             job_monitor(job)
67             exp_results[scheme_name][meas_basis] = job.result(timeout=120)
68
69     # Extrai resultados
70     I_calibration_data = {}
71     for scheme_name, meas_basis_results in exp_results.items():
72         result_bloch = []
73         for meas_basis in bloch_basis:
74             idx = 0
75             full_res = []
76             for idx_rep, number_of_repetitions in enumerate(repetitions):
77                 rep_res = []
78                 for scaling_factor in scalings:
79                     counts = meas_basis_results[meas_basis].get_counts()[idx]
80                     excited_pop = 0
81                     idx += 1
82                     for bits, count in counts.items():
83                         excited_pop += count if bits[::-1][qubit] == "1" else
84                             ↪ 0
85                     rep_res.append(excited_pop / num_shots_per_point)
86                 full_res.append(rep_res)
87                 result_bloch.append(1 - 2 * np.asarray(full_res))
88             I_calibration_data[scheme_name] = np.asarray(result_bloch)
89     save_var("results/relative_calibration_data_nairobi_qubit_0",

```

90

I_calibration_data)

Ocorrendo tudo conforme esperado para a execução do experimento, é possível montar o gráfico da figura C.2 que mostra a infidelidade do pulso para as diferentes repetições e diferentes valores de S_{rel} . O valor escolhido é o que apresenta menor infidelidade média com relação as repetições.

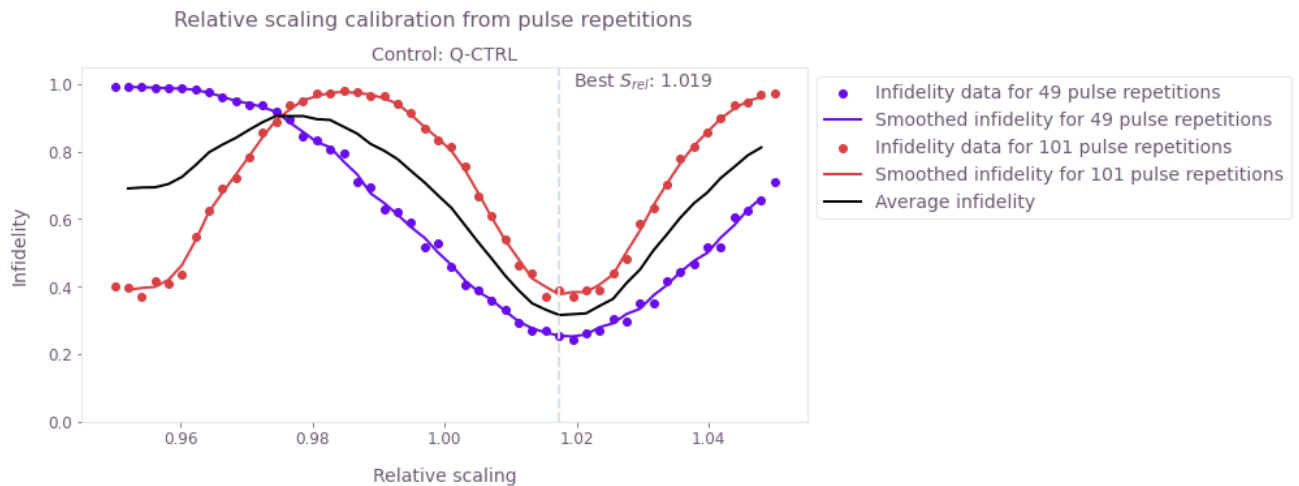


Figura C.2 – Experimento para Obtenção de S_{rel}

```

1 # Calcula infidelidade
2 bloch_infidelity = {}
3 for scheme_name in gate_schemes.keys():
4     bloch_vectors = np.array(
5         [
6             np.array(
7                 [
8                     I_calibration_data[scheme_name][:, rep, idx]
9                     for idx in range(scalings.shape[0])
10                ]
11            )
12            for rep in range(repetitions.shape[0])
13        ]
14    )
15    bloch_infidelity[scheme_name] = (
16        1 - ((1 + np.dot(bloch_vectors, target_bloch)) / 2) ** 2
17    )
18
19 # Monta o gráfico

```

```

20 fig, axs = plt.subplots(1,
21                         len(gate_schemes.keys()),
22                         figsize=(10, 5),
23                         squeeze=False)
24 fig.suptitle("Relative scaling calibration from pulse repetitions", y=1)
25
26 def movingaverage(x, w):
27     return np.convolve(x, np.ones(w), "valid") / w
28
29 # Calcula valor de S_rel
30 best_scaling_I = {}
31 average_infidelity = {}
32 for idx, scheme_name in enumerate(gate_schemes.keys()):
33     ax = axs[0][idx]
34     infidelity_smoothed = []
35     for idx, pulse_repetition in enumerate(repetitions):
36         infidelity_smoothed.append(
37             movingaverage(bloch_infidelity[scheme_name][idx, :], 3)
38         )
39     ax.set_title(f"Control: {scheme_name}")
40     ax.scatter(
41         scalings,
42         bloch_infidelity[scheme_name][idx, :],
43         label=f"Infidelity data for {pulse_repetition} pulse
44             ↪ repetitions",
45     )
46     ax.plot(
47         scalings[1:-1],
48         infidelity_smoothed[-1],
49         label=f"Smoothed infidelity for {pulse_repetition} pulse
50             ↪ repetitions",
51     )
52     ax.set_xlabel("Relative scaling")
53     ax.set_ylabel("Infidelity")
54
55     average_infidelity[scheme_name] = np.average(infidelity_smoothed,
56         ↪ axis=0)
57     best_scaling_I[scheme_name] = scalings[
58         np.argmin(average_infidelity[scheme_name]) + 1
59 ]
60     ax.plot(

```

```

58     scalings[1:-1],
59     average_infidelity[scheme_name],
60     label="Average infidelity",
61     c="k",
62 )
63 ax.axvline(best_scaling_I[scheme_name], ls="--")
64 ax.text(
65     scalings[np.argmin(infidelity_smoothed) + 1],
66     np.max(infidelity_smoothed),
67     "Best  $S_{rel}$ : {scalings[np.argmin(infidelity_smoothed) +
68     ↪ 1]:.3f}",
69     fontsize=14,
70 )
71 ax.set_ylim(0, 1.05)
72 ax.legend(loc="upper left", bbox_to_anchor=(1, 1))
73 plt.show()

```

C.4.1.2 Obtenção de S_{amp}

De maneira similar, a obtenção de S_{amp} também precisa da definição de um intervalo de valores e quantidades de repetições do pulso. Neste caso, esses valores foram definidos para os mesmos valores do experimento para obtenção de S_{rel} .

```

1  # Repetições para cálculo de infidelidade
2  repetitions = np.array([49, 101])
3  # Valores para teste de  $S_{amp}$ 
4  scalings = np.linspace(0.95, 1.05, 50)
5
6  # Monta circuitos
7  pulse_program = {}
8  for scheme_name in gate_schemes.keys(): # Para cada porta lógica
9      pulse_program[scheme_name] = {}
10     # Para cada base da esfera de Bloch (x, y e z)
11     for meas_basis in bloch_basis:
12         schedules = []
13         for repetition in repetitions: # Para cada repetição (111, 441)
14             for scaling_factor in scalings: # Para cada fator  $S_{amp}$ 
15                 # Cria circuito
16                 circ = QuantumCircuit(1, 1,
17                 ↪ name=f"pi_pulse_scal_{scaling_factor}_reps_"
18                 + str(repetition)

```



```

18         + "_basis_"
19         + str(meas_basis))
20     circ.x(0) # Porta X utilizada para construir o pulso
21     circ.measure(0, 0)
22
23     # Parametriza os valores de amplitude
24     # com S_rel já calculado e S_amp
25     scaled_waveform = (
26         np.real(waveform[scheme_name])
27         * best_scaling_I[scheme_name]
28         + 1j * np.imag(waveform[scheme_name])
29     ) * scaling_factor
30
31     # Criação do pulso do tipo Waveform
32     control_pulse = Play(Waveform(scaled_waveform),
33                          drive_chan)
34
35     schedule = Schedule(
36         name=f"pi_pulse_scal_{scaling_factor}_reps_"
37         + str(repetition)
38         + "_basis_"
39         + str(meas_basis)
40     )
41
42     # Repete pulso pela quantidade de vezes definida
43     for reps in np.arange(repetition):
44         schedule += control_pulse
45
46     # Aplica porta lógica para obter componentes x, y e z
47     if meas_basis == "x":
48         schedule += inst_sched_map.get("u2", [0],
49                                       P0=0.0, P1=np.pi)
50     if meas_basis == "y":
51         schedule += inst_sched_map.get("u2", [0],
52                                       P0=0.0, P1=np.pi / 2)
53
54     # Porta lógica X recebe programa do pulso
55     circ.add_calibration('x', [qubit], schedule)
56
57     # Adiciona circuito na lista para ser executado
58     schedules.append(circ)

```

```
59
60     pulse_program[scheme_name][meas_basis] = schedules
61
62 # Executa experimentos
63 exp_results = {}
64 for scheme_name in gate_schemes.keys():
65     exp_results[scheme_name] = {}
66     print("Control Scheme:", scheme_name)
67     for meas_basis in bloch_basis:
68         job = backend.run(pulse_program[scheme_name][meas_basis],
69                           meas_level=2,
70                           meas_return="single",
71                           shots=num_shots_per_point)
72         print("Job id:", job.job_id())
73         job_monitor(job)
74         exp_results[scheme_name][meas_basis] = job.result(timeout=120)
75
76 # Extrai resultados
77 amplitude_calibration_data = {}
78 for scheme_name, meas_basis_results in exp_results.items():
79     result_bloch = []
80     for meas_basis in bloch_basis:
81         idx = 0
82         full_res = []
83         for idx_rep, number_of_repetitions in enumerate(repetitions):
84             rep_res = []
85             for scaling_factor in scalings:
86                 counts = meas_basis_results[meas_basis].get_counts()[idx]
87                 excited_pop = 0
88                 idx += 1
89                 for bits, count in counts.items():
90                     excited_pop += count if bits[::-1][qubit] == "1" else
91                     ↪ 0
92                 rep_res.append(excited_pop / num_shots_per_point)
93             full_res.append(rep_res)
94             result_bloch.append(1 - 2 * np.asarray(full_res))
95         amplitude_calibration_data[scheme_name] = np.asarray(result_bloch)
96
97 # Salva resultados
98 save_var("results/amplitude_calibration_data_nairobi_qubit_0",
99         amplitude_calibration_data)
```

Após executar os experimentos de obtenção de S_{amp} , é necessário montar o gráfico da figura C.3 que mostra a infidelidade do pulso para as diferentes repetições e diferentes valores de S_{amp} tal como foi feito com S_{rel} .

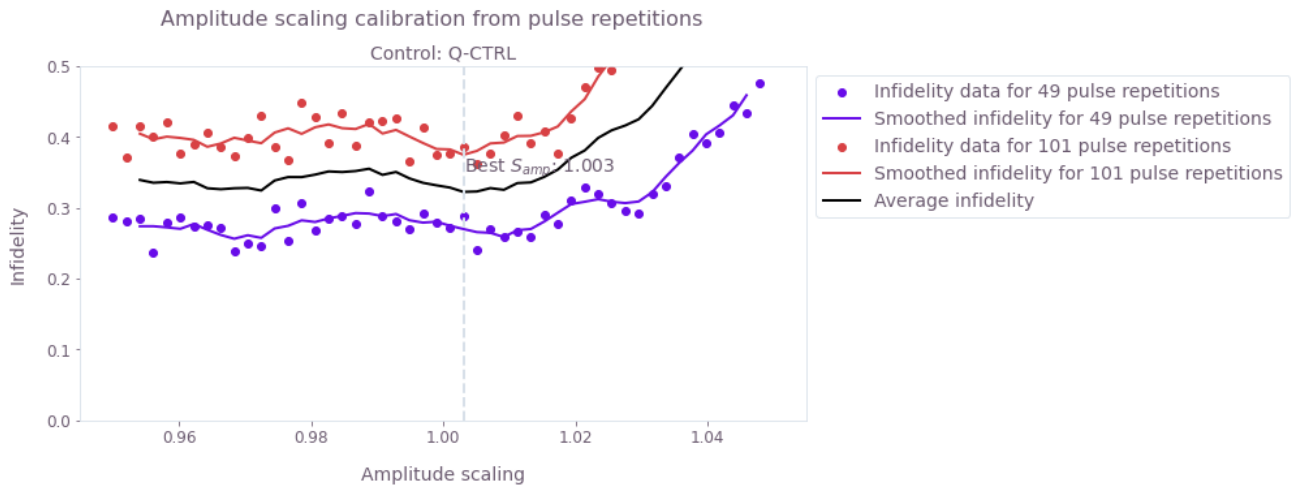


Figura C.3 – Experimento para Obtenção de S_{amp}

```

1 # Calcula infidelidade
2 bloch_infidelity = {}
3 for scheme_name in gate_schemes.keys():
4     bloch_vectors = np.array(
5         [
6             np.array(
7                 [
8                     amplitude_calibration_data[scheme_name][:, rep, idx]
9                     for idx in range(scalings.shape[0])
10                ]
11            )
12            for rep in range(repetitions.shape[0])
13        ]
14    )
15    bloch_infidelity[scheme_name] = (
16        1 - ((1 + np.dot(bloch_vectors, target_bloch)) / 2) ** 2
17    )
18
19 # Monta o gráfico
20 fig, axs = plt.subplots(1, len(gate_schemes.keys()), figsize=(10, 5),
21     ↪ squeeze=False)
22 fig.suptitle(f"Amplitude scaling calibration from pulse repetitions", y=1)

```

```
22
23 # Calcula valor de S_amp
24 best_scaling_amplitude = {}
25 average_infidelity = {}
26 for idx, scheme_name in enumerate(gate_schemes.keys()):
27     ax = axs[0][idx]
28     infidelity_smoothed = []
29     for idx, pulse_repetition in enumerate(repetitions):
30         infidelity_smoothed.append(
31             movingaverage(bloch_infidelity[scheme_name][idx, :], 5)
32         )
33     ax.set_title(f"Control: {scheme_name}")
34     ax.scatter(
35         scalings,
36         bloch_infidelity[scheme_name][idx, :],
37         label=f"Infidelity data for {pulse_repetition} pulse
38         ↪ repetitions",
39     )
40     ax.plot(
41         scalings[2:-2],
42         infidelity_smoothed[-1],
43         label=f"Smoothed infidelity for {pulse_repetition} pulse
44         ↪ repetitions",
45     )
46     ax.set_xlabel("Amplitude scaling")
47     ax.set_ylabel("Infidelity")
48
49     average_infidelity[scheme_name] = np.average(infidelity_smoothed,
50     ↪ axis=0)
51     best_scaling_amplitude[scheme_name] = scalings[
52         np.argmin(average_infidelity[scheme_name]) + 2
53     ]
54     ax.plot(
55         scalings[2:-2],
56         average_infidelity[scheme_name],
57         label="Average infidelity",
58         c="k",
59     )
60     ax.text(
61         best_scaling_amplitude[scheme_name],
62         0.35,
```

```

60     "Best  $S_{amp}$ : %.3f" % best_scaling_amplitude[scheme_name],
61     fontsize=14,
62 )
63 ax.axvline(best_scaling_amplitude[scheme_name], ls="--")
64 ax.set_ylim(0, 0.5)
65 ax.legend(loc="upper left", bbox_to_anchor=(1, 1))
66 plt.show()

```

C.4.2 Calibração Autônoma

O Boulder Opal oferece uma alternativa automatizada para a obtenção dos parâmetros S_{rel} e S_{amp} . Ele implementa um agente autônomo que executa em um looping fechado para encontrar os melhores valores dos fatores. As vantagens de se usar esse método é que ele permite executar otimizações para um número maior de repetições do que a calibração fine-tuned, entretanto, não se tem muito controle sobre o comportamento do agente e ele pode encontrar valores inadequados se não utilizado corretamente.

Antes de iniciar o looping, é necessário definir algumas funções e realizar um experimento de mitigação de erros de medidas.

```

1  # Importação de bibliotecas
2  from qiskit.result import marginal_counts
3  from qiskit.utils.mitigation import complete_meas_cal, CompleteMeasFitter
4  from qiskit import execute
5
6  # Definição de função que executa um experimento para
7  # diferentes valores de  $S_{rel}$  e  $S_{amp}$  e diferentes repetições
8  def run_ibm_experiments(control, calibration_pairs, repetitions, qubit):
9      I_values = qctrl.utils.pwc_pairs_to_arrays(control["I"])[1]
10     Q_values = qctrl.utils.pwc_pairs_to_arrays(control["Q"])[1]
11
12     schedules = []
13     # Para cada par de fatores  $S_{rel}$  e  $S_{amp}$ 
14     for scal in calibration_pairs:
15         # Parametriza os valores de amplitude
16         # com  $S_{rel}$  já calculado e  $S_{amp}$ 
17         scaled_waveform = (I_values * scal[0] + 1j * Q_values) * scal[1]
18
19         # Criação do pulso do tipo Waveform
20         control_pulse = Play(Waveform(scaled_waveform), drive_chan)
21
22         # Para cada base da esfera de Bloch (x, y e z)

```

```
23     for meas_basis in bloch_basis:
24         for repetition in repetitions: # Para cada repetição
25             # Cria circuito
26             circ = QuantumCircuit(1, 1,
27                                     name=f"pulse_scal_{scal}_reps_"
28                                     + str(repetition)
29                                     + "_basis_"
30                                     + str(meas_basis))
31             circ.x(0) # Porta X utilizada para construir o pulso
32             circ.measure(0, 0)
33
34             schedule = Schedule(
35                 name=f"pulse_scal_{scal}_reps_"
36                 + str(repetition)
37                 + "_basis_"
38                 + str(meas_basis)
39             )
40
41             # Repete pulso pela quantidade de vezes definida
42             for reps in np.arange(repetition):
43                 schedule += control_pulse
44
45             # Aplica porta lógica para obter componentes x, y e z
46             if meas_basis == "x":
47                 schedule += inst_sched_map.get("u2", [qubit],
48                                               P0=0.0, P1=np.pi)
49             if meas_basis == "y":
50                 schedule += inst_sched_map.get("u2", [qubit],
51                                               P0=0.0, P1=np.pi / 2)
52
53             # Porta lógica X recebe programa do pulso
54             circ.add_calibration('x', [qubit], schedule)
55
56             # Adiciona circuito na lista para ser executado
57             schedules.append(circ)
58
59     # Executa experimentos
60     job = backend.run(schedules, shots=num_shots_per_point)
61     print("Job id:", job.job_id())
62     job_monitor(job)
63     exp_results = job.result(timeout=120)
```

```
64
65     # Retorna resultados
66     return exp_results
67
68 # Definição de função que extrai resultados
69 def extract_results(results, calibration_pairs, repetitions):
70     idx = 0
71     pairs_results = []
72
73     # Aplica filtro de medida
74     results = measurementFilter.apply(results)
75
76     # Para cada par de fatores S_rel e S_amp
77     for scaling_pair in calibration_pairs:
78         basis_results = []
79         # Para cada base da esfera de Bloch (x, y e z)
80         for meas_basis in bloch_basis:
81             rep_results = []
82             # Para cada repetição
83             for number_of_repetitions in repetitions:
84                 count = marginal_counts(results.get_counts()[idx])
85                 excited_pop = 0
86                 idx += 1
87                 if "1" in count.keys():
88                     excited_pop += count["1"]
89                 rep_results.append(excited_pop / num_shots_per_point)
90             basis_results.append(rep_results)
91         pairs_results.append(1 - 2 * np.asarray(basis_results))
92     calibration_data = np.asarray(pairs_results)
93
94     # Calcula infidelidade
95     bloch_vectors = np.array(
96         [
97             np.array([
98                 calibration_data[idx, :, rep] for rep in
99                 ↪ range(len(repetitions))
100             ])
101         ]
102     )
103     bloch_infidelity = 1
```

```

104     bloch_infidelity -= ((1 + np.dot(bloch_vectors, target_bloch)) / 2) **
    ↪ 2
105
106     # Imprime infidelidade média com os parâmetros utilizados
107     cost_infidelity = np.average(bloch_infidelity, axis=1)
108     print("Best cost in this step:", np.min(cost_infidelity))
109
110     # Retorna infidelidade média
111     return cost_infidelity
112
113 # Definição de função que realiza mitigação de erros de medida
114 def conductErrorMitigation(shots):
115     cal_circuits, state_labels = complete_meas_cal(
116         qubit_list=np.arange(1), circlabel="MeasErrMitCal"
117     )
118
119     cal_job = execute(cal_circuits, backend=backend, shots=shots)
120     job_monitor(cal_job)
121     cal_results = cal_job.result()
122
123     meas_fitter = CompleteMeasFitter(
124         cal_results, state_labels
125     )
126
127     # Define filtro de medida
128     meas_filter = meas_fitter.filter
129
130     ErrorMitigationComplete = True
131
132     # Retorna filtro de medida
133     return meas_filter
134
135 measurementFilter = conductErrorMitigation(shots=4096)

```

Em seguida, é possível executar o looping que utiliza o agente para encontrar os melhores valores de S_{amp} e S_{rel} .

A calibração autônoma permite definir uma quantidade maior de repetições do pulso para o cálculo da infidelidade e otimização dos parâmetros. Mas também é preciso estar atento para escolher valores que façam sentido. Para a porta X , serão escolhidas as repetições 5, 17, 29, 41 e 53. Dependendo de suas necessidades, podem ser escolhidas mais e maiores repetições.

Os intervalos dos parâmetros S_{amp} e S_{rel} também podem ser maiores. Nesse experimento foram escolhidos intervalos de $[0.9, 1.1]$.

Outra variável que pode ser ajustada é a quantidade de combinações dos parâmetros a serem testados por vez. Nesse experimento, esse valor foi definido para 5.

Além disso, é importante ressaltar que a escolha de muitas repetições ou muitos pares de parâmetros por vezes pode exceder o limite de circuitos a serem executados em um job do computador quântico escolhido.

```
1 # Repetições para cálculo de infidelidade
2 repetitions = np.arange(5, 54, 12)
3 # Valores para teste de S_rel
4 relative_bounds = (0.9, 1.1)
5 # Valores para teste de S_amp
6 amplitude_bounds = (0.9, 1.1)
7 # Quantidade de pares de parâmetros por teste
8 test_point_count = 5
9
10 calibration_result = {}
11 best_cost_array = []
12 best_param_array = []
13
14 # Cria pares de parâmetros aleatórios para primeira execução
15 rng = np.random.default_rng()
16 parameter_set = [
17     [
18         rng.uniform(relative_bounds[0], relative_bounds[1]),
19         rng.uniform(amplitude_bounds[0], amplitude_bounds[1]),
20     ]
21     for idx in np.arange(test_point_count)
22 ]
23
24 # Executa experimentos com parâmetros iniciais
25 exp_results = run_ibm_experiments(
26     pulse_to_calibrate, parameter_set, repetitions, qubit
27 )
28 cost_results = extract_results(exp_results, parameter_set, repetitions)
29
30 # --- Setup do otimizador
31 # Define valores iniciais do agente
32 bound = qctrl.types.closed_loop_optimization_step.BoxConstraint(
33     lower_bound=relative_bounds[0], upper_bound=relative_bounds[1]
```

```
34 )
35 bound_2 = qctrl.types.closed_loop_optimization_step.BoxConstraint(
36     lower_bound=amplitude_bounds[0], upper_bound=amplitude_bounds[1]
37 )
38 initializer_GP = (
39     qctrl.types.closed_loop_optimization_step.GaussianProcessInitializer(
40         bounds=[bound, bound_2], rng_seed=0
41     )
42 )
43
44 # Define estado para o agente
45 optimizer = qctrl.types.closed_loop_optimization_step.Optimizer(
46     gaussian_process_initializer=initializer_GP
47 )
48
49 best_cost, best_controls = min(
50     zip(cost_results, parameter_set), key=lambda params: params[0]
51 )
52 optimization_count = 0
53
54 best_cost_array.append(best_cost)
55 best_param_array.append(best_controls)
56
57 # Executa os experimentos em um looping até que o custo seja
↪ suficientemente pequeno ou alcance 10 passos
58 idx = 0
59 params_array = []
60 while best_cost > 0.005 and idx <= 10:
61     # Imprime o menor custo atual
62     optimization_steps = (
63         "optimization step" if optimization_count == 1 else "optimization
64         ↪ steps"
65     )
66     print(
67         f"Best infidelity after {optimization_count} \
68         Boulder Opal {optimization_steps}: {best_cost}"
69     )
70
71     # Organiza os resultados dos experimentos em um formato adequado
72     results = [
73         qctrl.types.closed_loop_optimization_step.CostFunctionResult(
```

```
73         parameters=list(parameters), cost=cost, cost_uncertainty=0.1
74     )
75     for parameters, cost in zip(parameter_set, cost_results)
76 ]
77
78 # Chama o otimizar e obtém os próximos parâmetros
79 optimization_result =
80     qctrl.functions.calculate_closed_loop_optimization_step(
81         optimizer=optimizer,
82         results=results,
83         test_point_count=test_point_count
84     )
85 optimization_count += 1
86
87 # Organiza os dados retornados pelo otimizador
88 parameter_set = np.array(
89     [test_point.parameters
90     for test_point in optimization_result.test_points]
91 )
92 optimizer = qctrl.types.closed_loop_optimization_step.Optimizer(
93     state=optimization_result.state
94 )
95
96 # Obtém resultados dos experimentos
97 exp_results = run_ibm_experiments(
98     pulse_to_calibrate,
99     parameter_set,
100     repetitions,
101     qubit
102 )
103 cost_results = extract_results(exp_results, parameter_set,
104     ↪ repetitions)
105
106 params_array.append(parameter_set)
107
108 # Registra os melhores resultados
109 cost, controls = min(
110     zip(cost_results, parameter_set), key=lambda params: params[0]
111 )
112
113 if cost < best_cost:
```

```
113     best_cost = cost
114     best_controls = controls
115     idx += 1
116     best_cost_array.append(best_cost)
117     best_param_array.append(best_controls)
118     print("Best parameters:", best_controls)
119
120 # Imprime o melhor custo
121 print(f"Infidelity: {best_cost}")
122 print(f"Calibration parameters: {best_controls}")
123 calibration_result[qubit] = best_controls
124
125 # Monta o gráfico de convergência
126 fig = plt.figure()
127 fig.suptitle("Convergence of automated pulse calibration")
128 plot_cost_history(fig,
129                   best_cost_array,
130                   y_axis_log=False,
131                   initial_iteration=0)
```

Após o agente autônomo encontrar os melhores parâmetros, é montado um gráfico que mostra a convergência do pulso com relação ao custo da função que representa a melhor infidelidade encontrada ao longo das repetições. A figura C.4 mostra a convergência para este exemplo.

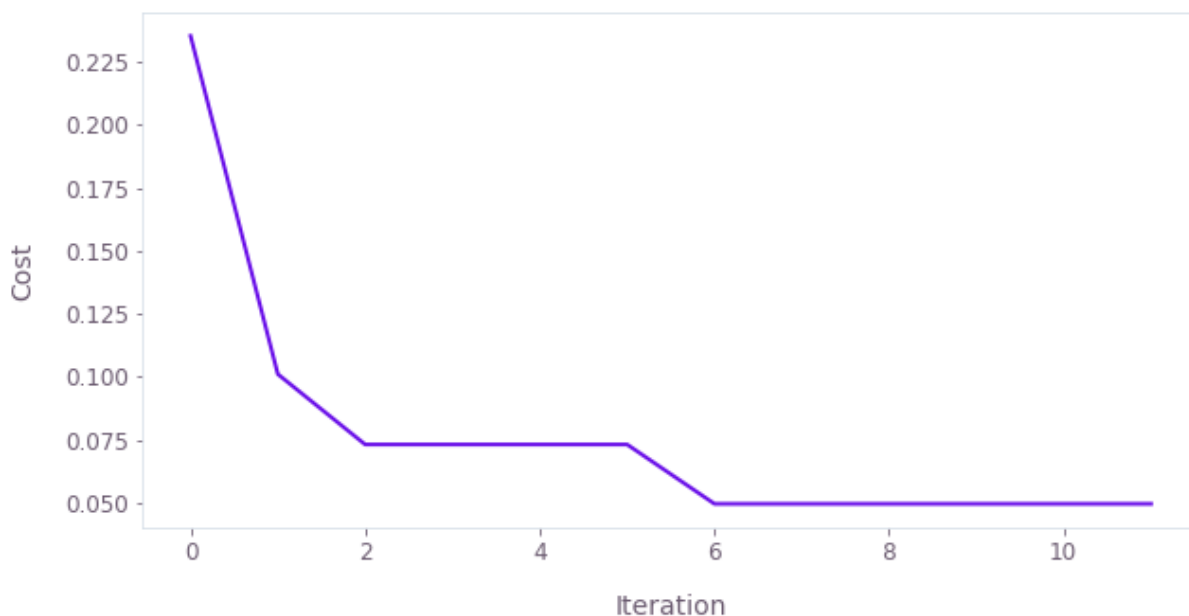


Figura C.4 – Menores Custos Durante o Processo de Calibração Autônoma

C.5 VALIDAÇÃO

Após a calibração, podemos realizar um experimento para verificar se as otimizações funcionaram adequadamente. Esse experimento consiste em avaliar a infidelidade das portas lógicas repetindo-a diversas vezes. Nesses termos, será possível avaliar os resultados das diferentes calibrações.

A quantidade de repetições pode ser definida conforme necessidade, nesse experimento ela foi definida em valores de 1 a 151 em intervalos de 4.

Também é possível escolher quais calibrações serão testadas por meio da variável `calibration_results`.

```
1 # Repetições para cálculo de infidelidade
2 repetitions = np.arange(1, 151, 4)
3
4 # Calibrações para testar
5 calibration_results = {
6     "Basic Rabi calibration": [1, 1],
7     "Fine-tuned calibration": [
8         best_scaling_I["Q-CTRL"],
9         best_scaling_amplitude["Q-CTRL"],
10    ],
11     "Automated calibration": best_param_array[-1],
12 }
13
14 # Definição das formas de onda em valores de amplitude
15 calibrated_waveform = {}
16 for scheme_name, scalings in calibration_results.items():
17     calibrated_waveform[scheme_name] = (
18         scalings[0] * np.real(waveform["Q-CTRL"])
19         + 1j * np.imag(waveform["Q-CTRL"])
20     ) * scalings[1]
21
22 # Monta circuitos
23 pulse_program = {}
24 # Para cada calibração
25 for scheme_name, calibrated_pulse in calibrated_waveform.items():
26     pulse_program[scheme_name] = {}
27     # Para cada base da esfera de Bloch (x, y e z)
28     for meas_basis in bloch_basis:
29         schedules = []
30         # Para cada repetição
```

```
31     for repetition in repetitions:
32         # Cria circuito
33         circ = QuantumCircuit(1, 1,
34                             name=f"pulse_reps_"
35                                 + str(repetition)
36                                 + "_basis_"
37                                 + str(meas_basis))
38         circ.x(0) # Porta X utilizada para construir o pulso
39         circ.measure(0, 0)
40
41         # Criação do pulso do tipo Waveform
42         control_pulse = Play(Waveform(calibrated_pulse), drive_chan)
43
44         schedule = Schedule(
45             name=f"pulse_reps_"
46                 + str(repetition)
47                 + "_basis_"
48                 + str(meas_basis)
49         )
50
51         # Repete pulso pela quantidade de vezes definida
52         for reps in np.arange(repetition):
53             schedule += control_pulse
54
55         # Aplica porta lógica para obtenção dos componentes x, y e z
56         if meas_basis == "x":
57             schedule += inst_sched_map.get("u2", [0],
58                                           P0=0.0, P1=np.pi)
59         if meas_basis == "y":
60             schedule += inst_sched_map.get("u2", [0],
61                                           P0=0.0, P1=np.pi / 2)
62
63         # Porta lógica X recebe programa do pulso
64         circ.add_calibration('x', [qubit], schedule)
65
66         schedules.append(circ)
67
68     pulse_program[scheme_name][meas_basis] = schedules
69
70 # Executa experimentos
71 exp_results = {}
```

```

72 for scheme_name, program in pulse_program.items():
73     exp_results[scheme_name] = {}
74     print("Control Scheme:", scheme_name)
75     for meas_basis in bloch_basis:
76         job = backend.run(program[meas_basis],
77                             meas_level=2,
78                             meas_return="single",
79                             shots=num_shots_per_point)
80         print("Job id:", job.job_id())
81         job_monitor(job)
82         exp_results[scheme_name][meas_basis] = job.result(timeout=120)
83
84 # Extrai resultados
85 calibrated_pulse_performance = {}
86 for scheme_name, results in exp_results.items():
87     print(scheme_name)
88     result_bloch = []
89     for meas_basis in bloch_basis:
90         idx = 0
91         full_res = []
92         for idx_rep, number_of_repetitions in enumerate(repetitions):
93             rep_res = []
94             counts = results[meas_basis].get_counts()[idx]
95             excited_pop = 0
96             idx += 1
97             for bits, count in counts.items():
98                 excited_pop += count if bits[::-1][qubit] == "1" else 0
99             rep_res.append(excited_pop / num_shots_per_point)
100            full_res.append(rep_res)
101            result_bloch.append(1 - 2 * np.asarray(full_res))
102            calibrated_pulse_performance[scheme_name] = np.asarray(result_bloch)
103
104 # Salva resultados
105 save_var("results/calibrated_pulse_performance_nairobi_qubit_0",
106         calibrated_pulse_performance)
107
108 # Calcula infidelidades
109 bloch_infidelity = {}
110 for scheme_name, performance in calibrated_pulse_performance.items():
111     bloch_vectors = performance[:, :, 0].T
112     bloch_infidelity[scheme_name] = (

```

```

113     1 - ((1 + np.dot(bloch_vectors, target_bloch)) / 2) ** 2
114 )
115
116 # Monta o gráfico
117 fig = plt.figure(figsize=(10, 5))
118 fig.suptitle(f"Pulse performance after calibration")
119 for scheme_name, infidelity in bloch_infidelity.items():
120     plt.plot(repetitions, infidelity, marker="o", label=scheme_name)
121     plt.xlabel("Pulse repetition number")
122     plt.ylabel("Infidelity")
123     plt.ylim(0, 1.05)
124 plt.legend()
125 plt.show()

```

Dessa forma, é esperado um resultado conforme mostra a figura C.5, com as calibrações fine-tuned e autônoma tendo resultados melhores do que a calibração básica de Rabi.

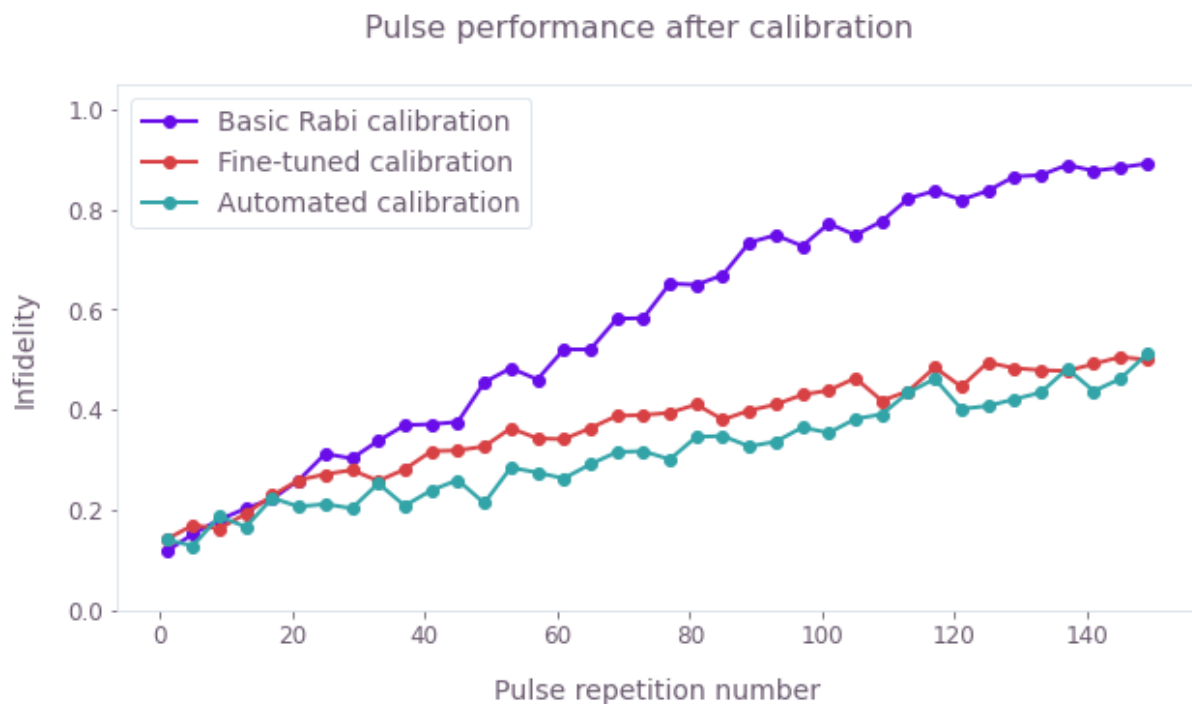


Figura C.5 – Resultado das Diferentes Calibrações do Pulso X Resistente à Dephasing

Análise Comparativa de Técnicas de Mitigação de Erros em Processos de Computação Quântica

Eduardo W. Lussi¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

Abstract. *The Quantum Computing paradigm has been considered a disruptive technology capable to solve some complex or hard problems efficiently. However, quantum systems are still very unstable and hard to control. This paper proposes implementations of quantum error mitigation techniques for quantum systems using practical applications. For doing so, quantum control for superconducting quantum systems is investigated. The practical applications consider the implementation of the Measure Once 1-way Quantum Finite Automata (MO1QFA) for solving the modulus problem. IBM Quantum Experience and Qiskit software development kit are used for modeling and running quantum circuits on real quantum computers. Additionally, Q-CTRL Boulder Opal tools are used in order to automate some hardware optimization process.*

Resumo. *O paradigma de computação quântica tem sido considerado uma tecnologia disruptiva capaz de solucionar problemas complexos de uma maneira bastante eficiente. Entretanto, sistemas quânticos ainda são extremamente instáveis e difíceis de se controlar adequadamente. Este trabalho tem como objetivo implementar técnicas de mitigação de erros de sistemas quânticos em aplicações práticas. Para isso, a área de controle quântico em computadores quânticos de supercondutores será investigada. As aplicações práticas consideram a implementação de um autômato finito quântico MO1QFA para resolver o problema do módulo. A IBM Quantum Experience e o kit de desenvolvimento de software Qiskit são utilizados como ambientes de modelagem de sistemas quânticos e aplicação em máquinas reais. Em complemento ao Qiskit, as ferramentas oferecidas pelo Q-CTRL Boulder Opal são utilizadas para automatizar os processos de otimização do hardware quântico.*

1. Introdução

Recentemente, a computação quântica se tornou um campo bastante atrativo para cientistas da computação. Em 2016, a IBM lança a IBM Quantum Experience (IBMQ) [Research 2016], oferecendo computadores e simuladores quânticos acessíveis via Cloud, facilitando, assim, as avaliações experimentais dos pesquisadores de computação quântica. Em 2019, a Google diz ter alcançado a supremacia quântica [Arute et al. 1974] por meio de um experimento utilizando seu processador quântico de 53 qubits, que foi executando em 3 minutos e 20 segundos, e que, teoricamente, teria levado cerca de 10 000 anos para executar no computador clássico mais poderoso da época, o Summit. Todavia, há controvérsas sobre o uso do termo "supremacia" nesse experimento [Bonifacic 2019].

Apesar dos grandes avanços e expectativas infladas da computação quântica, ela ainda sofre com a fragilidade dos sistemas quânticos. As implementações atuais dos

qubits ainda são muito instáveis e suscetíveis a ruídos do ambiente. As técnicas para o controle dos qubits, os equipamentos de medição e outros processos completamente aleatórios causam decoerência [Krantz et al. 2019], fazendo com que o sistema perca suas propriedades quânticas e se tornar um sistema clássico [Maslov et al. 2018].

De qualquer forma, há três principais formas de melhorar os resultados na implementação de um circuito quântico: otimização circuital, códigos de correção de erros (QEC) e controle quântico. As otimizações circuitais são mais triviais e consistem em reduzir o tamanho do circuito reduzindo o número de portas lógicas substituindo-as por outras equivalentes. Ainda em um nível de abstração mais alto, os códigos de correção de erros reduzem as probabilidades na ocorrência de erros implementando qubits lógicos utilizando uma redundância de qubits físicos [Maslov et al. 2018]. Essa técnica considera apenas a informação computacional do circuito quântico, considera apenas erros de bit-flip e phase-flip e não está interessada em manipulações do hardware.

Em um nível de abstração mais baixo, existem as técnicas de controle quântico, que consistem em melhorar a performance do hardware quântico manipulando os qubits de uma forma mais eficiente para executar o comportamento desejado [Q-CTRL 2022]. Em computadores quânticos supercondutores, as portas lógicas quânticas são traduzidas em pulsos de micro-ondas que alteram o estado quântico do sistema quântico. Qualquer pulso problemático ou oscilação nesse controle podem causar decoerência. Na IBMQ, os computadores quânticos podem ser controlados fisicamente por meio do Qiskit Pulse [Alexander et al. 2020, Qiskit 2022], que consiste em uma biblioteca de programação de computadores quânticos a nível de pulsos.

Nesse sentido, este trabalho apresenta técnicas para a implementação de portas lógicas quânticas otimizadas que mitigam os erros de decoerência dos computadores quânticos. Para isso, será feita uma descrição da área de controle quântico em computadores quânticos supercondutores, mostrando como portas lógicas quânticas são implementadas em hardware, como programar sistemas quânticos a nível de pulsos de micro-ondas e como são feitos os processos de calibração e controle de hardware quântico para a correção de erros. O Q-CTRL Boulder Opal será utilizado de forma a automatizar os processos e facilitar a manipulação de pulsos de micro-ondas que controlam os estados quânticos. Serão realizados experimentos em computadores quânticos reais fornecidos pela IBM Quantum Experience [IBM 2022b] utilizando o kit de desenvolvimento de software (SDK) Qiskit, também da IBM.

Diante do regime NISQ de computação quântica que é afetado por erros, serão investigadas diferentes formas de se implementar portas lógicas quânticas e suas suscetibilidades e robustez a diferentes processos de erros. A aplicação prática de validação é um autômato finito quântico MO1QFA que resolve o problema do módulo, nesse sentido, além da investigação mais aprofundada dos pulsos que implementam as portas lógicas, são fornecidas diferentes alternativas para a implementação desse autômato que resultaram em consideráveis melhorias com relação às implementações existentes

2. Controle Quântico em Computação Quântica

Os computadores quânticos supercondutores da IBMQ podem ser controlados fisicamente por meio do Qiskit Pulse [Alexander et al. 2020, Qiskit 2022], que permite ao usuário extrair o máximo da performance do hardware quântico. Especificamente em computadores

quânticos supercondutores, a tradução de portas lógicas para pulsos é feita por meio de pulsos de micro-ondas que interagem fisicamente com o sistema quântico que implementa os qubits.

Modelar pulsos é uma tarefa complicada. As descrições básicas de pulsos consistem de três variáveis: frequência, amplitude e duração. A frequência corresponde ao valor no espectro de micro-ondas na qual os fótons são absorvidos pelo qubit, dessa forma, se esse valor é ligeiramente diferente da frequência do qubit, o pulso pode tirar o qubit de sua ressonância [dev team 2022]. Embora os computadores quânticos da IBMQ já ofereçam uma boa estimativa do valor da frequência, [IBM 2022a] apresenta um experimento para encontrar esse valor.

Em computação quântica, o experimento de Rabi consiste em testar o hardware quântico com o objetivo de avaliar o seu comportamento com relação a determinados pulsos, mais precisamente, ele encontra a relação entre os valores de amplitude e tempo e estado quântico. Quanto maior a amplitude, mais rápido o estado quântico oscila, portanto, o experimento de Rabi permite a descoberta das frequências de Rabi, que, visualmente, correspondem às frequências que o vetor de estado oscila na esfera de Bloch com relação à determinado eixo. O conhecimento das frequências de Rabi é essencial para construir diferentes pulsos que obtêm o mesmo efeito sobre o estado quântico.

Sabendo que diferentes pulsos podem ter o mesmo efeito no estado quântico, construir pulsos pode ser complicado. Pulsos mais curtos obtêm melhores resultados por diminuir a duração do circuito, distanciando-a dos tempos de decoerência do sistema quântico, porém, eles necessitam de amplitudes maiores que podem causar erros de vazamentos de elétrons [Jo et al. 2019]. A durabilidade dos pulsos também é um problema, as frequências dos qubits oscilam com o tempo e pode ser necessário recalibrar periodicamente as portas lógicas e as frequências dos qubits.

3. Measure Once 1-Way Quantum Finite Automata

Os experimentos para avaliar o comportamento de diferentes pulsos consideram a implementação de um autômato finito quântico (QFA). A teoria de Autômatos Finitos corresponde ao modelo mais simples da computação clássica, eles são capazes de reconhecer a classe das Linguagens Regulares em tempo linear, enquanto isso, os QFAs são mais poderosos e possuem a capacidade de reconhecer algumas linguagens mais abrangentes pertencentes à classe das Linguagens Livres-de-Contexto, destacando o poder dos modelos de máquinas quânticas.

A simplicidade dos QFAs é particularmente interessante para analisar e avaliar a ocorrência de erros na computação, que normalmente envolve operações específicas que se repetem muitas vezes, podendo deixar os circuitos muito longos, o que se faz necessário diminuir a duração das portas lógicas. Os experimentos deste trabalho buscam analisar a suscetibilidade a erros dos QFAs quando implementados nas plataformas da IBMQ, principalmente, na computação de palavras longas, também, considerando a ocorrência de falsos positivos na classificação das palavras.

A abordagem considera uma implementação de um autômato MO1QFA [Moore and Crutchfield 2000] que resolve o problema MOD_p para $p = 11$. A implementação utiliza a plataforma IBMQ como um ambiente real de computação

quântica para avaliar sua performance como plataforma para a implementação de QFAs considerando o controle quântico dos erros.

O Measure-Once 1-way Quantum Finite Automata (MO1QFA) é um autômato finito quântico proposto por [Moore and Crutchfield 2000]. As principais características desse autômato são:

- As transições são feitas por meio de operações unitárias;
- O estado quântico é medido apenas uma vez, no final da execução;
- A cabeça de leitura do autômato tem direção única, ou seja, se movimenta da esquerda para a direita, lendo um símbolo em cada transição.
- No começo da computação, a cabeça de leitura está no primeiro símbolo da fita de leitura.

Dada essas restrições, o MO1QFA é considerado o modelo mais elementar de autômato finito quântico, mas também é o modelo menos expressivo em comparação com outros modelos de autômatos quânticos.

3.1. Problema MOD^p

Para exemplificar o uso do MO1QFA para o tratamento de uma linguagem, será considerado o problema do módulo de um número primo. Essa linguagem pode ser modelada por meio de dois estados, $|0\rangle$ e $|1\rangle$, onde ambas utilizam a superposição quântica para rotacionar o vetor estado ao longo do eixo x .

[Say and Yakaryilmaz 2014] propuseram que, com o MO1QFA, é possível resolver problemas com erros limitados utilizando menos estados em comparação com soluções de autômatos clássicos. Para demonstrar isso, eles utilizaram o problema do módulo de números primos:

$$MOD^p = \{a^{jp} | j \text{ é um inteiro não-negativo}\}$$

O autômato respectivo quântico é composto de dois estados e reconhece palavras congruentes a 0 módulo p em 100% dos casos, entretanto, permite falsos positivos, com o maior valor de erro encontrado em palavras que se aproximam de $p/2$ módulo p , fazendo com que o erro se aproxime de:

$$\cos^2\left(\frac{\pi}{p}\right).$$

Para realizar a computação desse autômato, é necessário dividir o plano xz da esfera de Bloch que representa o qubit em p partes, onde cada fatia representa a congruência possível dos valores $0, 1, 2, \dots, p - 1$. Para esse algoritmo, a matriz que representa a rotação é dada pela equação 1, que divide o círculo em p partes e rotaciona o vetor estado de acordo.

$$R_x = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}, \text{ para } \theta = \frac{4\pi}{p} \quad (1)$$

É importante notar que o autômato apresentado pode ser modificado facilmente para reconhecer o módulo de qualquer primo. Para fazer isso de maneira clássica, p estados seriam necessários, enquanto apenas 2 são necessários utilizando o MO1QFA.

4. Experimentação

Os experimentos dessa seção consideram a implementação do autômato apresentado na seção 3 para resolver o problema do módulo para $p = 11$. Nesse sentido, foram utilizados diferentes computadores quânticos para os experimentos, com foco maior no *ibm_nairobi*, que é composto de 7 qubits com $T_1 = 127.23\mu s$ e $T_2 = 94.44\mu s$, os computadores *ibm_oslo* (7 qubits, $T_1 = 154.18\mu s$ e $T_2 = 85.84\mu s$) e *ibmq_belem* (5 qubits, $T_1 = 102.19\mu s$ e $T_2 = 108.27\mu s$) também foram utilizados de maneira a complementar alguns resultados.

Para a implementação do MO1QFA, serão consideradas palavras de tamanho congruente a 0 e a 3 módulo 11 de tamanho em um intervalo de 0 a 1000, totalizando 182 valores diferentes. Com esses valores, é esperado uma probabilidade de aceitação de 100% para palavras congruente a 0 módulo 11 e 2.0254% para palavras congruentes a 3 módulo 11.

Com o objetivo de otimizar os circuitos que executam nos computadores da IBMQ, o Qiskit implementa quatro níveis de otimização, de 0 a 3, quanto maior o número, mais otimizado será o circuito, ao custo de um tempo maior de transpilação [Qiskit]. O nível de otimização 0 apenas mapeia o circuito para o backend considerando as portas lógicas disponíveis nele, não realizando otimizações. O nível de otimização 1 faz pequenas otimizações colapsando portas lógicas adjacentes. Os níveis de otimização 2 e 3 fazem otimizações mais profundas considerando as relações entre as diferentes portas lógicas. É importante ressaltar que essas otimizações são apenas a nível de circuito, não alterando as implementações de portas lógicas existentes.

Nesse sentido, as ferramentas do Q-CTRL Boulder Opal serão utilizadas como alternativas às portas lógicas padrão do Qiskit. Dessa forma, o hardware será calibrado de forma a melhorar a performance das portas lógicas. Os experimentos desta seção buscam mostrar a influência dos erros em implementações de autômatos finitos quânticos, mostrando, também, diferentes alternativas para otimizar os circuitos. Portanto, serão feitos experimentos utilizando ambos os níveis de otimização 0 e 1 do Qiskit, utilizando diferentes portas lógicas resistentes a diferentes tipos de erros e utilizando diferentes calibrações oferecidas pelo Q-CTRL Boulder Opal.

4.1. Experimentos com Nível de Otimização 1

Os primeiros experimentos consideram o nível de otimização 1 do Qiskit na execução do autômato. As figuras 1 e 2 mostram uma comparação entre os valores obtidos e esperados de probabilidades de aceitação para palavras de tamanho congruente a 0 e a 3, respectivamente, módulo 11 de 0 a 1000, e as tabelas resumem os erros absolutos entre os valores obtidos e o esperados. Os mesmos experimentos foram executados nos computadores quânticos Nairobi, Oslo e Belém.

O primeiro ponto para se observar é que não há nenhuma acumulação do erro, ele se mantém em torno do mesmo valor independentemente do tamanho da palavra. Isso ocorre porque, ao contrário do nível de otimização 0, o nível de otimização 1 colapsa

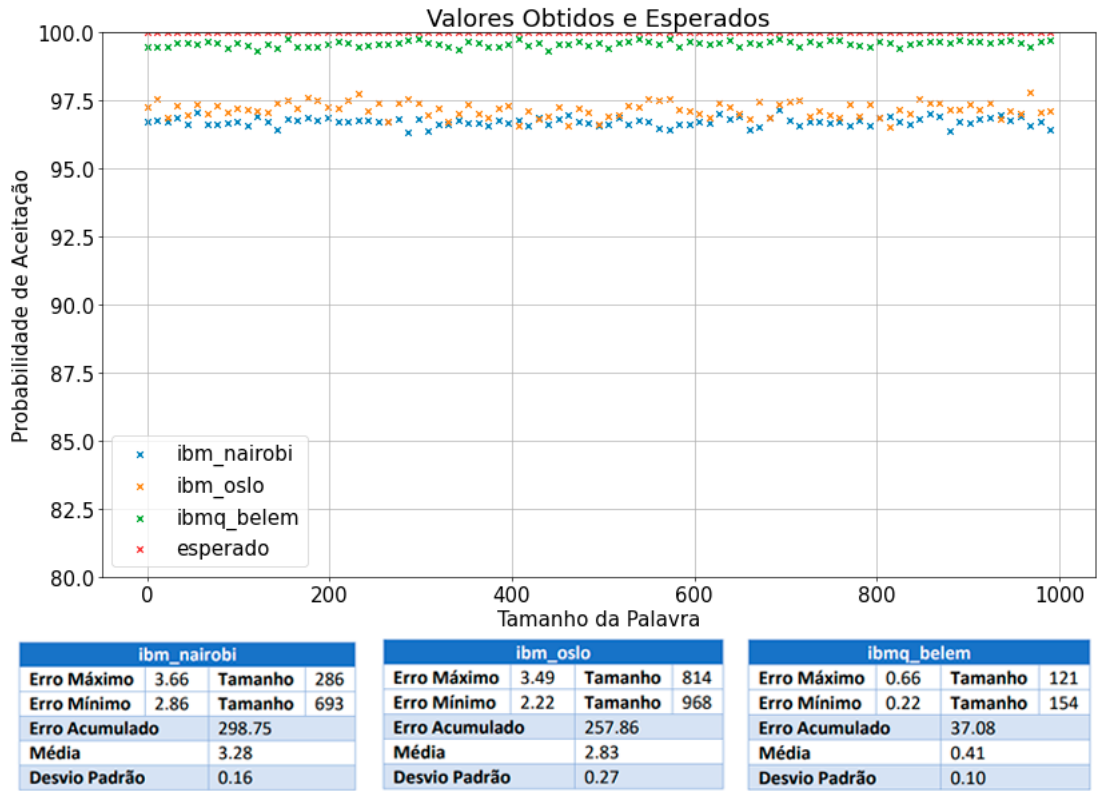


Figure 1. Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 0 módulo 11 com nível de otimização 1.

portas lógicas adjacentes, nesse caso, a sequência de portas R_x , e aplica apenas uma sequência de portas lógicas equivalentes a R_x , que são dependentes do hardware, com ângulo proporcional à sequência de todas as portas para aquela palavra.

Utilizando o nível de otimização 1, o problema do acúmulo do erro é superado, mas ainda existe um pequeno erro entre o obtido e o esperado. Em termos de análise de dados, ambos os experimentos com nível de otimização 1 possuem desvio padrão semelhantes, ambos pequenos, indicando que não há uma grande variabilidade dos resultados. Ademais, é possível perceber que nos computadores Nairobi e Belém, as palavras de tamanho congruente a 3 módulo 11 possuem um erro maior que as de tamanho congruente a 0 módulo 11, com uma discrepância mais atípica para o computador Belém, isso ocorre devido ao fato de que palavras de tamanho congruente a 0 módulo 11 fazem com que seja aplicado apenas uma porta lógica $R_x(0)$ ao qubit, ou seja, nenhuma rotação, enquanto os circuitos de palavras de tamanho congruente a 3 módulo 11 são compostos de uma rotação $\theta = 3 \times \frac{4\pi}{11}$, portanto, existe um erro associado à essa porta lógica. Em conformidade com isso, as tabelas mostram que o valor acumulado do erro é maior para palavras de tamanho congruente a 3 módulo 11 nesses casos. Por fim, os valores de máximo e mínimo dos erros não possuem nenhuma preferência para o tamanho da palavra, dado que todas as palavras de tamanho congruente a n módulo 11 utiliza a mesma transformação, os erros são arbitrários.

Com relação às diferentes plataformas, o computador Oslo teve um erro atípico menor em palavras de tamanho congruente a 3 módulo 11, da mesma forma que o com-

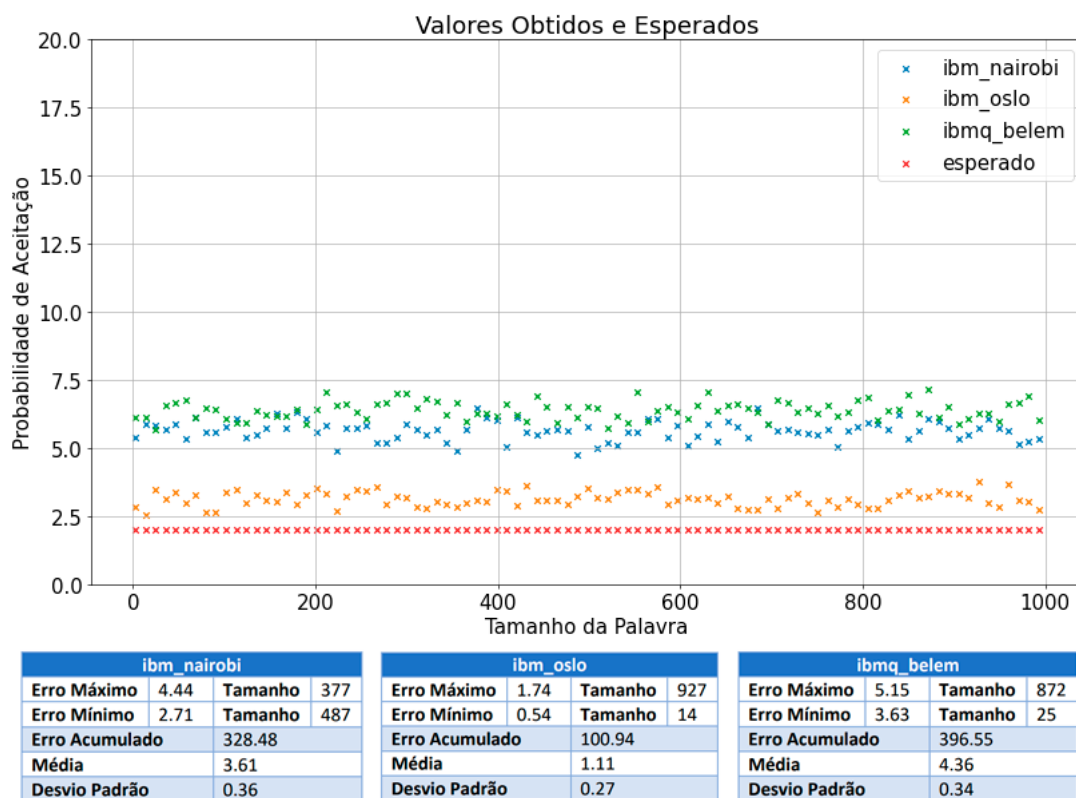


Figure 2. Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 3 módulo 11 com nível de otimização 1.

putador Belém teve um erro consideravelmente menor para a outra categoria de palavras. Em ambos os casos, acredita-se que o erro tenha colaborado positivamente, o que é possível e interessante de ser explorado em otimizações. Entretanto, o computador Belém obteve bons resultados para o circuito apenas com a medida, e a partir do momento em que portas lógicas foram aplicadas, obteve um erro semelhante ao Nairobi, enquanto o computador Oslo acumulou positivamente um erro ou acumulou um erro menor na aplicação das portas lógicas.

4.2. Experimentos com Nível de Otimização 0

Os próximos experimentos consideram o nível de otimização 0 do Qiskit. A figura 3 mostra os resultados obtidos para palavras de tamanho congruente a 0 módulo 11 e a figura 4 mostra os resultados obtidos para palavras de tamanho congruente a 3 módulo 11. Os experimentos foram executados nos computadores quânticos Nairobi, Oslo e Belém.

Como já comentado sobre o nível de otimização 0, existe um acúmulo no erro conforme o tamanho da palavra aumenta, as tabelas mostram que os erros mínimos acontecem por volta do menor tamanho da palavra, e conforme ele aumenta, as figuras mostram que o erro também aumenta, até se tornar arbitrário. Entretanto, existe um certo padrão entre os computadores quânticos, as probabilidades de aceitação contêm pontos de máximo e mínimo. Para palavras de tamanho congruente a 0 módulo 11, os computadores Nairobi e Oslo tiveram erros crescentes até por volta de 200 repetições da porta lógica, onde atingiram erros absolutos em torno de 70%, enquanto o computador Belém teve um crescimento

menor até atingir um máximo de 47% em 286 repetições. Depois disso, os resultados mostram outros pontos de erros mínimos e máximos, onde até mesmo para palavras de tamanho congruente a 3 módulo 11 o computador Belém teve oscilações menores. Apesar disso, para o experimento com palavras de tamanho congruente a 3 módulo 11, o computador Belém teve um crescimento no erro semelhante aos demais computadores quânticos.

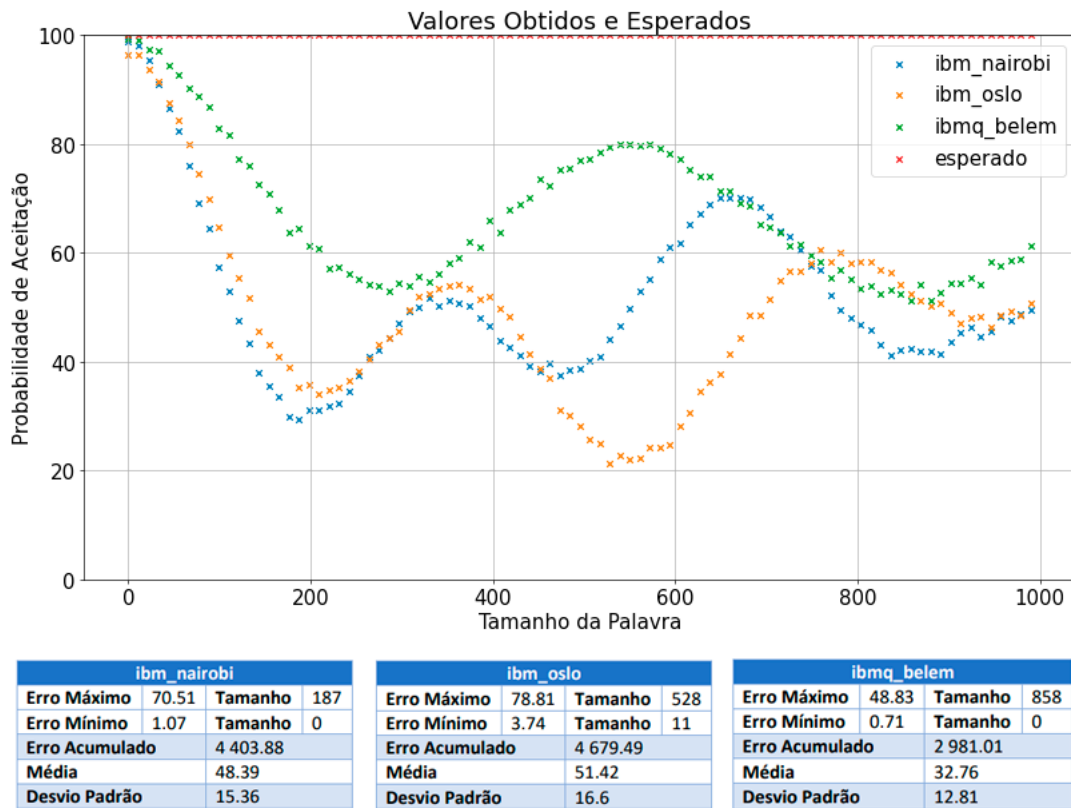


Figure 3. Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 0 módulo 11 com nível de otimização 0.

As tabelas também mostram que o erro máximo absoluto é semelhante entre palavras de tamanho congruente a 0 e 3 módulo 11 em ambos os três computadores, mas o erro mínimo é menor para palavras de tamanho congruente a 0 módulo 11, com exceção do computador Oslo que já teve melhores resultados em palavras de tamanho congruente a 3 módulo 11 com o nível de otimização 1, conforme abordado na seção 4.1. Como dito anteriormente, isso acontece porque a computação dessas últimas não necessita de nenhuma transformação porque representam rotações completas do vetor estado em torno da esfera de Bloch, enquanto o circuito para palavras de tamanho 3 há a aplicação de três portas lógicas $R_x\left(\frac{4\pi}{11}\right)$, acumulando um erro.

Diferentemente do nível de otimização 1, o nível de otimização 0 não colapsa as portas lógicas adjacentes e implementa cada porta lógica R_x como uma sequência de outras cinco portas lógicas. Por conta disso, o circuito se torna excessivamente longo e inviável de ser utilizado.

Ademais, é possível perceber que os desvios padrão são muito maiores do que os

valores dos experimentos com nível de otimização 1, portanto, os valores obtidos estão muito longe de serem tão previsíveis quanto os experimentos com nível de otimização 1.

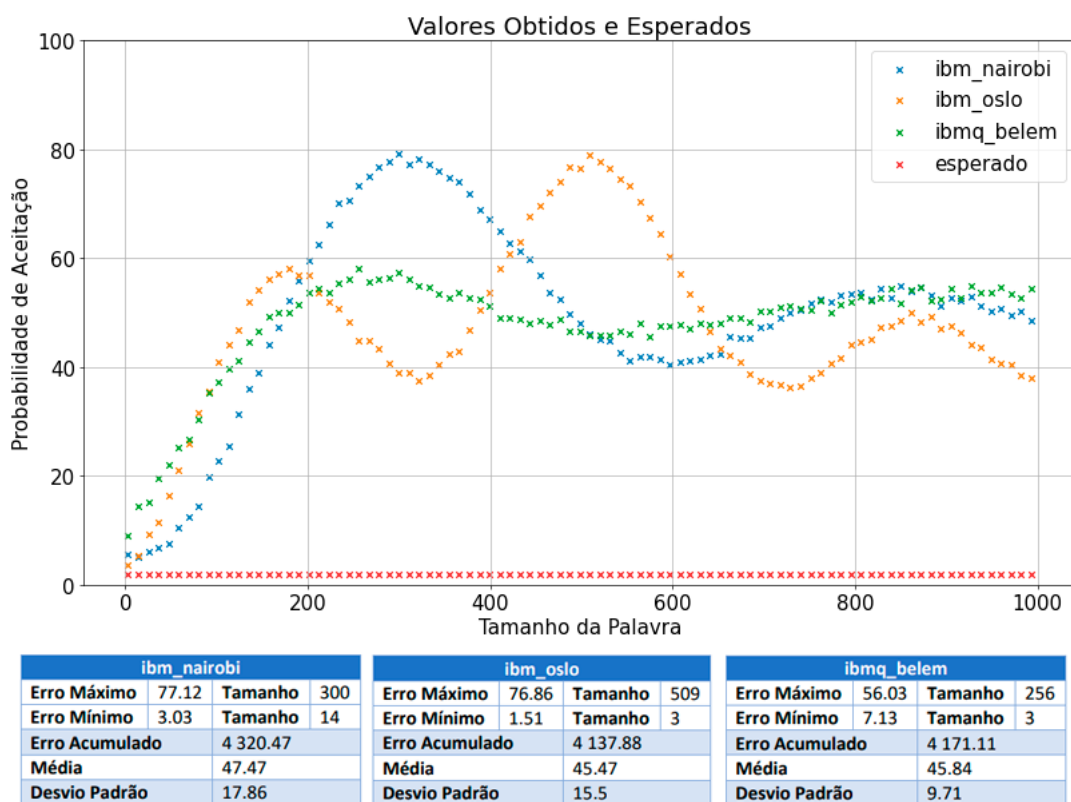


Figure 4. Probabilidades de aceitação obtidas e esperadas para palavras congruentes a 3 módulo 11 com nível de otimização 0.

Esses experimentos revelam que é insustentável trabalhar com circuitos não-otimizados, conforme o tamanho da palavra aumenta, é necessário aplicar cada vez mais portas lógicas ao qubit, e a acumulação do erro faz com que o autômato perca o controle de seu estado. Otimizações circuitais são definitivamente necessárias para a implementação de autômatos quânticos. Dado que sua computação é composta de múltiplas portas lógicas que dependem do tamanho da palavra, é necessário otimizar as transformações dos qubits para deixar a computação mais robusta com relação aos tamanhos das palavras.

4.3. Experimentos com Portas Lógicas Square Ultrarrápidas

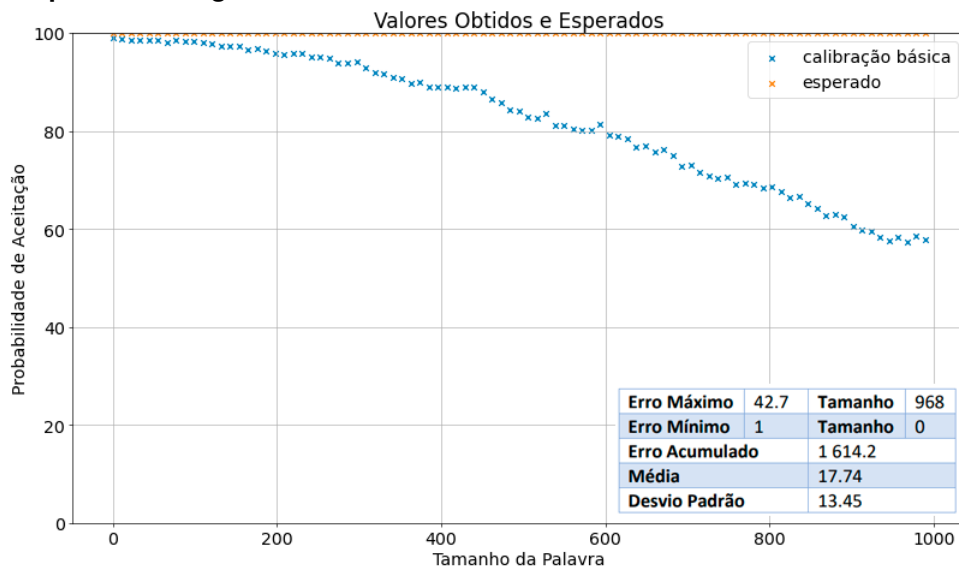
O Q-CTRL Boulder Opal oferece maneiras eficientes de calibrar o hardware quântico e construir portas lógicas quânticas otimizadas resistentes à ruídos. Os experimentos desta seção consideram a criação de uma porta lógica R_x ultrarrápida que busca ser eficiente na computação de palavras longas. Por se tratar de um pulso de baixa duração, a computação do circuito completo é consideravelmente mais rápida e se distancia mais dos tempos de decoerência do computador quântico. Para isso, serão utilizados os manuais [Q-CTRL] e [QCTRL 2022], o primeiro para a geração de portas lógicas e o segundo para a calibração do hardware.

Nesse sentido, foi definido um pulso do tipo square, ou seja, um pulso ideal que não considera a ocorrência de erros. Apesar do fato de ele não mapear erros de de-

phasing ou controle de amplitude, ele é bastante útil para a implementação do autômato quântico por conta de sua simplicidade e rapidez, permitindo computar longas palavras e se distanciar do tempo de decoerência do sistema quântico. Dessa forma, enquanto a implementação da porta R_x padrão dos computadores considerados é composta de outras cinco portas lógicas, a porta lógica square gerada é apenas uma de duração 14.2 ns , que reduz absurdamente o tempo de execução dos circuitos.

A partir da simples geração de um pulso específico para a operação desejada, já é possível conseguir resultados consideravelmente melhores. As figuras 5 e 6 mostram os resultados para palavras de tamanho congruente a 0 e 3, respectivamente, executados no computador quântico Nairobi. Por limitações de tempo com relação às filas de espera para a execução dos circuitos, esse experimento foi executado apenas no computador Nairobi, entretanto, resultados semelhantes devem ser refletidos nos demais computadores quânticos.

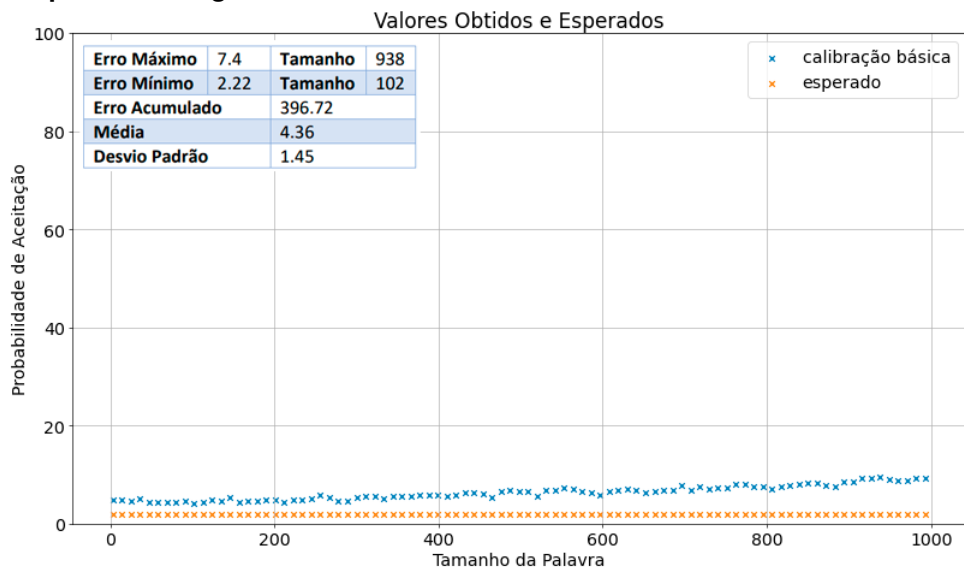
Figure 5. Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 0 módulo 11.



Os resultados para esse pulso são bastante intrigantes com relação à porta lógica padrão, os desvios nas probabilidades de aceitação com relação ao esperado são muito menores, especialmente para as palavras congruentes a 3 módulo 11, onde houve uma anormalidade em que o erro foi absurdamente pequeno. Dado que a natureza dos erros é por vezes estocástica, os mesmos experimentos podem ter resultados diversos em momentos diferentes devido a imensuráveis fatores.

Em comparação com os resultados da porta lógica padrão com nível de otimização 0, o baixo tempo do pulso se mostrou bastante eficiente mesmo com a calibração básica. Para os experimentos com palavras de tamanho congruente a 0 módulo 11, enquanto a probabilidade de aceitação da porta lógica padrão já acumulava um erro absoluto de 70.51% em 198 repetições, a calibração básica ainda estava com um erro absoluto de 3.64%. O mesmo ocorre para congruentes a 3 módulo 11, enquanto a porta lógica padrão acumulava um erro absoluto de 55.96% em 201 repetições, o pulso rápido estava com um erro absoluto de 2.83%. Além disso, os gráficos mostram que as probabilidades de

Figure 6. Probabilidades de aceitação do pulso square com calibração básica para palavras congruentes a 3 módulo 11.



aceitação são mais previsíveis com o pulso gerado, sendo menos afetados pelo tamanho da palavra.

Os experimentos com o pulso rápido de 14.2 ns para a implementação do autômato mostram que eles são absolutamente mais eficientes do que a transpilação para os pulsos genéricos do Qiskit, gerar pulsos square individuais para cada circuito pode ser uma alternativa mais interessante do que manter um conjunto de portas lógicas universais calibradas, entretanto, a otimização circuital dos experimentos com nível de otimização 1 é obviamente essencial nesse caso. Mas eventualmente pode não ser possível conseguir um resultado tão bom com essa otimização e evitar a aplicação de muitos pulsos, é nesses casos que o controle quântico se mostra necessário para amenizar os erros. Dessa forma, é perceptível que a utilização de controle quântico para construir portas lógicas personalizadas para determinados algoritmos é uma alternativa capaz de reduzir consideravelmente a ocorrência de erros.

Especialmente na implementação de autômatos quânticos, a utilização de pulsos rápidos é uma solução capaz de reduzir o tamanho do circuito e se distanciar dos tempos de decoerência dos sistemas. Isso justifica o fato de que os resultados dos pulsos rápidos e padrão, nessa ordem, possuem durabilidade maior do estado quântico com relação ao tamanho da palavra, isto é, o erro do pulso padrão se torna arbitrário depois de 200 repetições, enquanto a calibração básica teve apenas um erro crescente até 1000 repetições.

4.4. Calibrações Básicas, Fine-tuned e Automated

Os experimentos da seção 4.3 mostraram que portas lógicas ultrarrápidas são uma boa alternativa para melhorar os resultados de circuitos longos na implementação do autômato. Especialmente para esse tipo de aplicação, existem técnicas que aperfeiçoam a calibração do hardware quântico para tornar os pulsos mais duráveis com relação ao tempo de decoerência do sistema. Esta seção apresenta duas alternativas apresentadas no manual

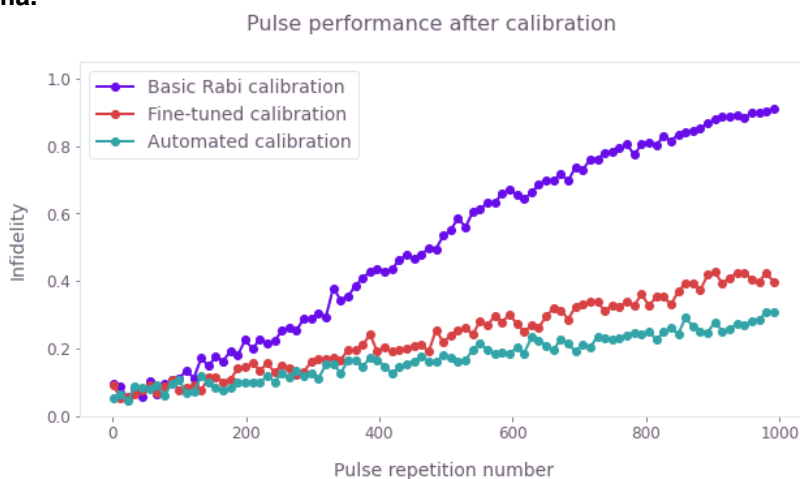
[QCTRL 2022] para aperfeiçoar essa calibração, que são a calibração fine-tuned e um método autônomo apresentado pelo artigo [Baum et al. 2021]. Os experimentos desta seção foram realizados apenas no computador quântico Nairobi.

Tanto a calibração fine-tuned quanto autônoma consistem em aperfeiçoar o experimento de Rabi para um determinado pulso, dessa forma, a caracterização do hardware se torna mais precisa para as necessidades do experimento. A calibração autônoma do Q-CTRL Boulder Opal é uma alternativa automatizada para essa tarefa com relação à fine-tuned. Essa calibração funciona em um laço fechado onde são definidos intervalos para os parâmetros e repetições do pulsos a serem calibradas, a partir disso, são feitos experimentos em sequência onde o agente busca pelos melhores parâmetros para o experimento.

A utilização das calibrações se mostrou útil até certo ponto. Foram feitos vários experimentos e percebeu-se que a necessidade de calibrar precisamente um determinado pulso é maior conforme as calibrações do experimento de Rabi vão perdendo sua precisão.

O gráfico da figura 7 mostra os resultados de infidelidade para as calibrações básicas, fine-tuned e autônoma. A medida de infidelidade apresenta resultados mais precisos na medição dos erros com relação às probabilidades de aceitação, enquanto o cálculo do erro das probabilidades de aceitação conseguem capturar apenas erros de bit-flip, a infidelidade corresponde à distância do ponto onde o vetor de estado do estado quântico foi medido e onde ele deveria estar, assim, ela é capaz de capturar tanto erros de bit-flip quanto erros de phase-flip.

Figure 7. Infidelidade do pulso square com calibrações básica, fine-tuned e autônoma.



As calibrações aqui apresentadas consideram a infidelidade como medida de desempenho, e assim foram calibradas. Mesmo que erros de phase-flip não sejam importantes para a implementação do autômato, o gráfico da figura 7 mostra a diferença de infidelidade do pulso square utilizando as três calibrações. Nesse caso, é possível perceber que a calibração básica tem desempenho consideravelmente inferior às demais calibrações, enquanto a fine-tuned e autônoma obtiveram desempenhos semelhantes, mas a autônoma conseguiu uma infidelidade menor. No caso da implementação do autômato, pode ser mais interessante definir a probabilidade de aceitação como medida de erro para as otimizações, mas na maioria dos casos essa informação não tem a mesma eficácia da

medida de infidelidade.

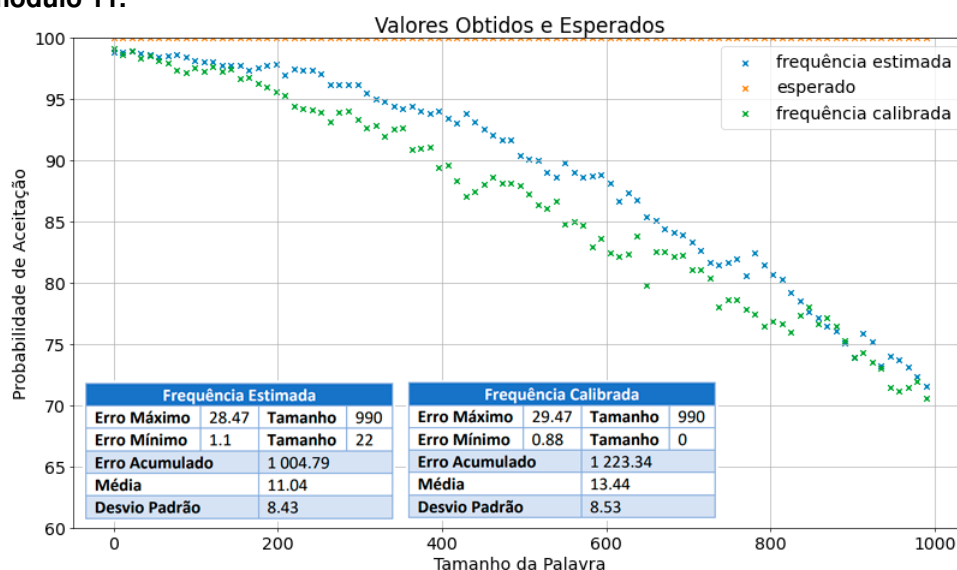
4.5. Otimização da Frequência

O valor de frequência é um componente importante na descrição dos pulsos, ela corresponde à frequência na qual os fótons do pulso são absorvidos pelo sistema quântico que implementa o qubit, permitindo alterar seu estado, e desvios nesse valor fazem com que ocorra uma interação com o sistema, que então decai para um sistema clássico. Essa seção busca avaliar o experimento apresentado por [IBM 2022a] para encontrar a frequência do qubit. Apesar do Qiskit já oferecer uma frequência padrão estimada suficientemente precisa, calibrar esse valor pode ser relevante devido a sua importância na ocorrência de erros.

A frequência estimada do qubit 0 do computador quântico Nairobi é de $5.260492122850421 \text{ GHz}$. Após a realização do experimento de otimização da frequência, ela foi atualizada para $5.260424304745819 \text{ GHz}$, uma diferença na ordem de dezenas de KHz com relação ao valor estimado.

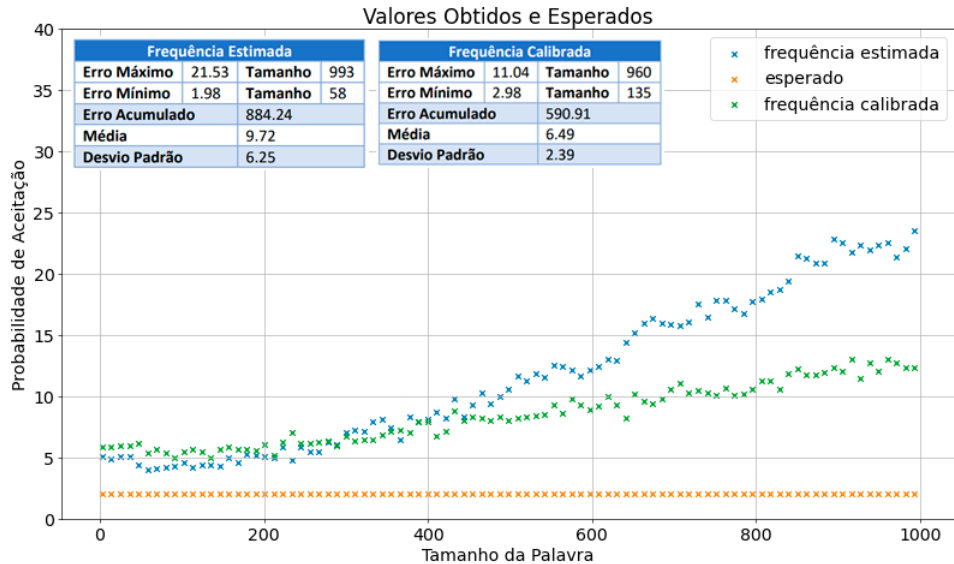
Para validar a necessidade da otimização da frequência, foram feitos experimentos na implementação do autômato para palavras de tamanho congruente a 0 e a 3 módulo 11, conforme os experimentos das demais seções, utilizando a calibração autônoma apresentada na seção 4.4 para a frequência padrão estimada e a calibrada. As figuras 8 e 9 mostram os resultados para palavras de tamanho congruente a 0 e 3 módulo 11 nessa ordem. É possível perceber que as palavras de tamanho congruente a 0 não tiveram melhorias nos erros absolutos com a frequência calibrada, o desempenho foi até um pouco melhor para a frequência estimada. Entretanto, as palavras de tamanho congruente a 3 módulo 11 obtiveram uma melhoria de desempenho significativa após 400 repetições do pulso.

Figure 8. Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 0 módulo 11.



A frequência é um atributo importante, a sua calibração não é uma tarefa custosa e os experimentos mostraram que ela pode apresentar melhores resultados na execução

Figure 9. Probabilidades de aceitação do pulso square com calibração autônoma e frequências estimadas e calibradas para palavras de tamanho congruente a 3 módulo 11.



de circuitos. Em palavras de tamanho congruente a 3 módulo 11, o erro absoluto acumulado foi 293.33 menor para a frequência calibrada, se destacando mais depois de 400 repetições. Mesmo que o comportamento do experimento para palavras de tamanho congruente a 0 tenha sido bastante semelhante para as duas frequências, com um erro acumulado maior para a frequência estimada, é mais interessante utilizar uma frequência mais calibrada. Ainda assim, a frequência padrão estimada é suficientemente precisa para a maioria dos experimentos e também não é uma má alternativa.

4.6. Pulsos Resistentes a Erros de Dephasing

Existem diferentes formas de descrever pulsos, pulsos square são mais simples porque possuem amplitude constante, mas existem formas mais sofisticadas de descrever pulsos de forma a contornar processos de erros. O manual [Q-CTRL] mostra como utilizar o Q-CTRL Boulder Opal para modelar pulsos resistentes a dephasing e flutuações no controle de amplitude. Basicamente, a descrição do pulso segue um hamiltoniano modelado utilizando um grafo, onde são feitas otimizações considerando esses processos e o pulso é gerado discretizando os resultados da otimização. Esta seção utiliza pulsos square de diferentes durações e pulsos resistentes a erros de dephasing gerados pelo Boulder Opal de forma a verificar a robustez das diferentes formas com relação aos erros.

O experimento de verificação de dephasing consiste em um experimento de detuning, que induz o erro realizando o mesmo experimento de infidelidade da seção anterior aplicando o pulso em frequências diferentes da padrão, nesse caso, em intervalos de 10 MHz em torno da frequência padrão, com o intuito de causar decoerência. Os resultados são avaliados por meio da medida de infidelidade, já que o objetivo é avaliar a robustez dos pulsos.

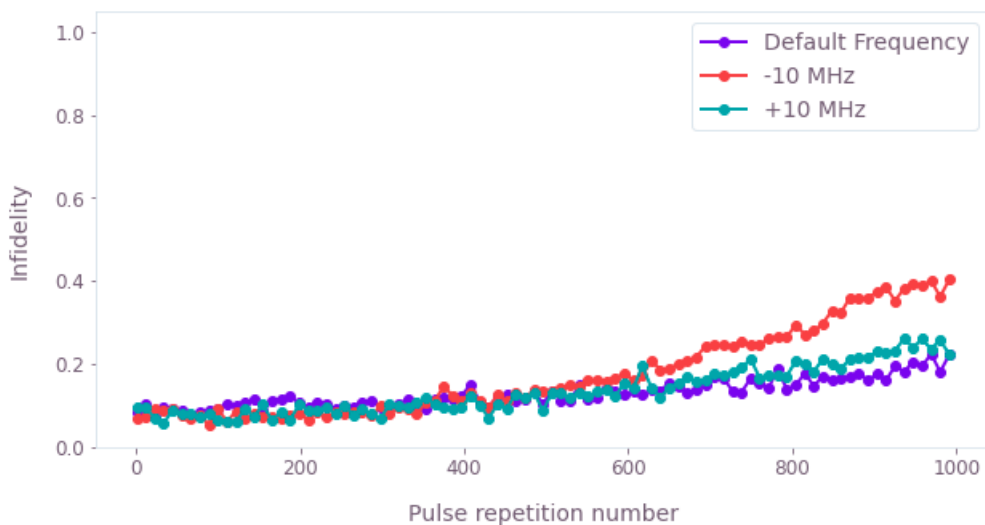
O primeiro pulso square gerado de 14.2 ns teve ótimos resultados na implementação do autômato por conta de sua curta duração, entretanto, ele exige amplitudes mais altas e isso pode fazer com que ele seja mais suscetível a erros. A figura

10 mostra a infidelidade do pulso square de 14 ns com as frequências padrão, -10 MHz e $+10\text{ MHz}$. É possível perceber que o pulso é consideravelmente resistente a erros na frequência, diferenças relevantes só podem ser observadas após 600 repetições do pulso, onde a frequência com -10 MHz acaba acumulando um erro maior do que a $+10\text{ MHz}$ e a padrão, nessa ordem. Os números das tabelas confirmam que a acumulação do erro absoluto também segue essa ordem. Apesar disso, a frequência padrão e com $+10\text{ MHz}$ seguem muito próximas e pode-se dizer que é preferível errar a frequência para mais do que para menos.

De forma a verificar a influência dos valores de amplitude na ocorrência de erros de dephasing, foi gerado outro pulso square com duração maior, de 227.33 ns , com o mesmo formato do pulso de 14 ns . A figura 11 mostra os resultados do mesmo experimento para esse pulso. Nesse caso, os erros para as diferentes frequências são imperceptíveis, as tabelas mostram que o erro acumulado para a frequência padrão é até ligeiramente maior do que para a frequência com erros de -10 MHz e $+10\text{ MHz}$. A hipótese de que pulsos com amplitudes mais baixas são mais resistentes a erros de frequência parece estar correta de acordo com esse experimento.

Figure 10. Infidelidade do pulso square de 14 ns aplicado na frequência padrão, com -10 MHz e $+10\text{ MHz}$.

Square 14.22ns pulse performance with detuning



Frequência Padrão			
Erro Máximo	22.45	Tamanho	969
Erro Mínimo	7.11	Tamanho	67
Erro Acumulado	1 180.38		
Média	12.97		
Desvio Padrão	3.4		

-10 MHz			
Erro Máximo	40.37	Tamanho	991
Erro Mínimo	8.76	Tamanho	89
Erro Acumulado	1 599.61		
Média	17.58		
Desvio Padrão	10.53		

+10 MHz			
Erro Máximo	26.19	Tamanho	936
Erro Mínimo	5.77	Tamanho	34
Erro Acumulado	1 231.45		
Média	13.53		
Desvio Padrão	5.60		

Apesar do pulso mais longo ter se mostrado mais resistente a processos de dephasing, a infidelidade é substancialmente maior, assim como o seu aumento à medida que o tamanho do circuito aumenta. Pode-se dizer que pulsos rápidos sofrem mais com dephasing, mas possuem resultados melhores pela menor exposição ao tempo de decoerência do sistema, que é o maior causador de erros. Dessa forma, a necessidade de robustez com relação a dephasing vem da necessidade de diminuir a duração do circuito com a

construção de portas lógicas mais rápidas com amplitudes maiores.

O Boulder Opal se propõe a gerar pulsos otimizados que contornam problemas de dephasing. Diferentemente dos pulsos square apresentados anteriormente, que possuem apenas um segmento de frequência de Rabi constante, o pulso do Boulder Opal é composto de segmentos de tamanho 4 com relação ao tempo de amostragem do computador quântico, nesse caso, esse valor é de 0.2 ns , que resulta em um pulso de 227.33 ns . A frequência de Rabi foi limitada em $2\pi \times 8.5 \times 10^6 / \sqrt{2}$ para I e Q . Diferente do pulso square, que é constante, o pulso otimizado pelo Boulder Opal tem uma atuação diferente com relação ao movimento do vetor estado na esfera de Bloch. A evolução do vetor de estado do pulso square tem um movimento direto, enquanto o pulso otimizado realiza diversas movimentações na esfera de Bloch com o intuito de desviar o processo de erro de dephasing.

Figure 11. Infidelidade do pulso square de 227.33 ns aplicado na frequência padrão, com -10 MHz e $+10 \text{ MHz}$.



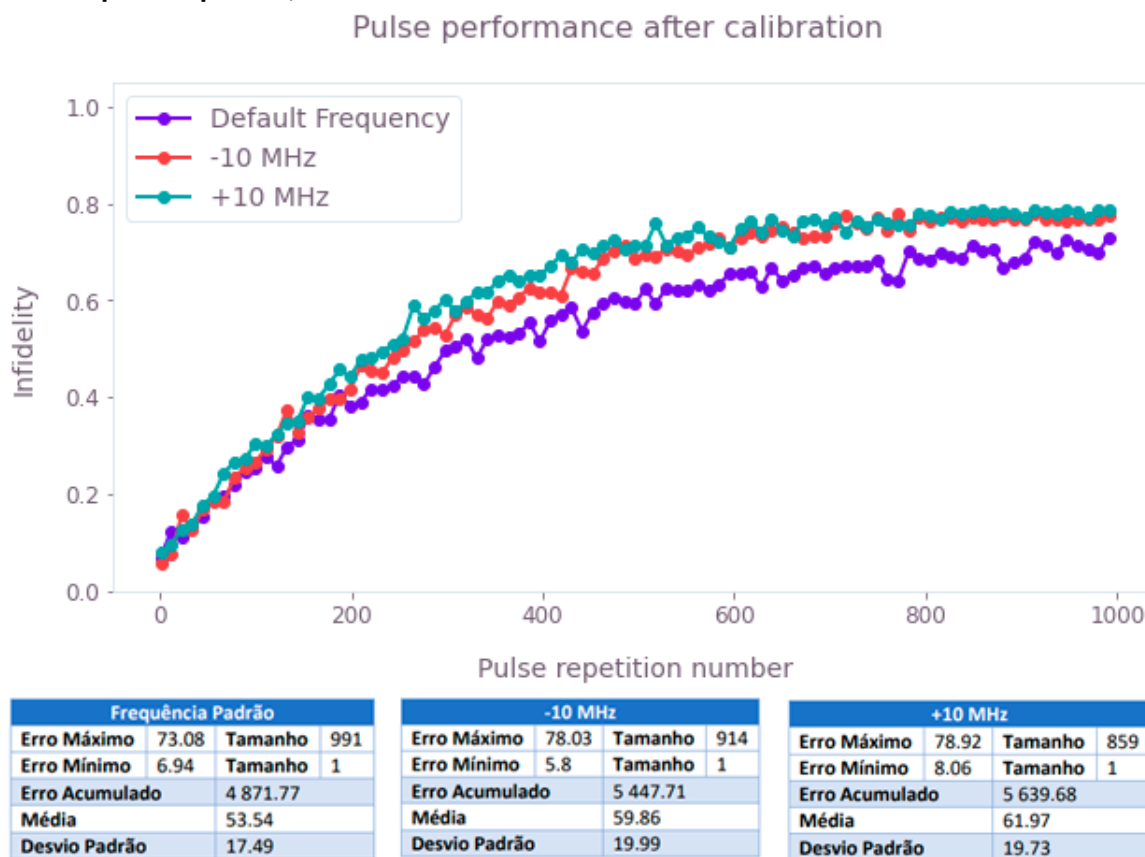
Frequência Padrão			
Erro Máximo	72.87	Tamanho	914
Erro Mínimo	7.12	Tamanho	1
Erro Acumulado	4 924.67		
Média	54.12		
Desvio Padrão	17.88		

-10 MHz			
Erro Máximo	72.15	Tamanho	991
Erro Mínimo	8.76	Tamanho	12
Erro Acumulado	4 741.35		
Média	52.10		
Desvio Padrão	17.81		

+10 MHz			
Erro Máximo	70.98	Tamanho	991
Erro Mínimo	8.06	Tamanho	1
Erro Acumulado	4 765.9		
Média	52.37		
Desvio Padrão	16.95		

A figura 12 mostra os resultados do experimento para o pulso gerado pelo Boulder Opal. O erro acumulado com a frequência padrão é bastante semelhante ao pulso square de mesma duração, entretanto, esse pulso aparentemente foi mais afetado pelos desvios na frequência do que o pulso square, com erros acumulados maiores para as duas frequências em torno da padrão. Acredita-se que para essa duração dos pulsos não há muito a ser melhorado com relação à dephasing, os resultados com o pulso square mais longo já havia apresentado bons resultados, a geração de um pulso resistente à dephasing se faz necessário à medida em que são geradas portas lógicas mais curtas, que são mais afetadas por esse tipo de erro, todavia, o cálculo de otimização para portas lógicas mais curtas não obteve resultados muito bons.

Figure 12. Infidelidade do pulso resistente à dephasing de 227.33 ns aplicado na frequência padrão, com -10 MHz e +10 MHz.



5. Conclusão e Trabalhos Futuros

As máquinas quânticas atuais ainda são bastante ruidosas. É perceptível que existe um ótimo trabalho em otimizações circuitais e transpilação do circuito, onde são mantidas portas lógicas devidamente calibradas e que são bastante robustas, mas nem sempre implementam as operações desejadas. É por esse motivo que a implementação de algoritmos quânticos mais complexos ainda demanda a utilização de técnicas personalizadas de controle quântico e correção de erros. Assim, a linha de raciocínio de manter um conjunto de portas lógicas quânticas universais específicas onde os circuitos são traduzidos para elas pode não ser tão interessante quanto realizar a transpilação direcionada ao próprio hardware, porém, para isso, é necessário manter uma boa caracterização do hardware.

As ferramentas oferecidas pelo Q-CTRL Boulder Opal se mostraram bastante úteis nesse processo de manutenção de portas lógicas e calibração do hardware quântico. Com elas, foi possível gerar portas lógicas quânticas personalizadas e devidamente otimizadas de uma forma bastante prática. Entretanto, os manuais são voltados para um público com um conhecimento maior sobre o hardware quântico e não são muito úteis como materiais introdutórios.

Com relação aos testes realizados, as melhores portas lógicas foram as mais curtas mesmo que tenham sido mais afetadas por erros, a simplicidade das portas lógicas square

também se mostrou interessante na implementação do autômato. As calibrações fine-tuned e autônoma das portas lógicas também são importantes, com elas, é possível reduzir consideravelmente as infidelidades melhorando a tradução para os valores de amplitude. A frequência dos pulsos, no entanto, não obteve grandes influências nos resultados, tanto em sua calibração quanto durante os testes de indução de dephasing, a frequência padrão oferecida pelo Qiskit já conseguiu bons resultados.

Ainda há muito a se fazer na área de mitigação de erros de computadores quânticos. Técnicas de controle quântico até então se fazem necessárias na execução de algoritmos. A transpilação direta do circuito para o hardware considerando as especificidades das portas lógicas em diferentes localizações do circuito pode ser uma alternativa viável nessa área, assim como considerações a respeito do comportamento do circuito para prever possíveis erros e ajustar os pulsos das portas lógicas.

No domínio da implementação de autômatos finitos quânticos, a investigação de soluções para outros problemas pode ser interessante, assim como a investigação de outros modelos de autômatos. A simplicidade dos autômatos é uma característica importante para a identificação da influência de erros nos processos, caracterizações podem ser úteis para tentar realizar um mapeamento das causas dos erros na execução dos algoritmos como base para uma possível adequação dos pulsos que implementam as portas lógicas.

References

- Alexander, T., Kanazawa, N., Egger, D. J., Capelluto, L., Wood, C. J., Javadi-Abhari, A., and McKay, D. C. (2020). Qiskit pulse: Programming quantum computers through the cloud with pulses. *Quantum Science and Technology*, 5.
- Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G. S. L., Buell, D. A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M. P., Hartmann, M. J., Ho, A., Hoffmann, M., Huang, T., Humble, T. S., Isakov, S. V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P. V., Knysh, S., Korotkov, A., Kostrița, F., Landhuis, D., Lindmark, M., Lyakh, E. L. D., Mandrà, S., McClean, J. R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mutus, M. M. J., Naaman, O., Neeley, M., Neill, C., Niu, M. Y., Ostby, E., Petukhov, A., Platt, J. C., Quintana, C., Rieffel, E. G., Roushan, P., Rubin, N. C., Sank, D., Satzinger, K. J., Smelyanskiy, V., Sung, K. J., Trevithick, M. D., Vainsencher, A., Villalonga, B., White, T., Yao, Z. J., Yeh, P., Zalcman, A., Neven, H., and Martinis, J. M. (1974). Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510.
- Baum, Y., Amico, M., Howell, S., Hush, M., Liuzzi, M., Mundada, P., Merkh, T., Carvalho, A. R. R., and Biercuk, M. J. (2021). Experimental deep reinforcement learning for error-robust gate-set design on a superconducting quantum computer. *PRX Quantum*, 2(4).
- Bonifacic, I. (2019). Google may have taken a step towards quantum computing 'supremacy' (updated).
- dev team, P. (2022). Quantum computing with superconducting qubits.
- IBM (2022a). Calibrating qubits with qiski pulse.

- IBM (2022b). Página inicial. Acesso em 8 de julho de 2022.
- Jo, H., Song, Y., and Ahn, J. (2019). Leakage suppression by ultrafast composite pulses. *Optics Express*, 27.
- Krantz, P., Kjaergaard, M., Yan, F., Orlando, T., Gustavsson, S., and Oliver, W. D. (2019). A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6.
- Maslov, D., Nam, Y., and Kim, J. (2018). An outlook for quantum computing. *Point of View*, 107(1):5–10.
- Moore, C. and Crutchfield, J. P. (2000). Quantum automata and quantum grammars. *Theoretical Computer Science*, 237(1-2):275–306.
- Q-CTRL. Designing noise-robust single-qubit gates for ibm qiskit.
- Q-CTRL (2022). What is quantum control?
- QCTRL (2022). How to automate calibration of control hardware. Acesso em 16 de julho de 2022.
- Qiskit. Transpiler passes and pass manager. Available at https://qiskit.org/documentation/tutorials/circuits_advanced/04_transpiler_passes_and_passmanager.html.
- Qiskit, I. (2022). Qiskit pulse.
- Research, I. (2016). Ibm makes quantum computing available on ibm cloud to accelerate innovation.
- Say, A. C. and Yakaryılmaz, A. (2014). Quantum finite automata: A modern introduction. In *Computing with New Resources*, pages 208–222. Springer.