Your Thesis Title Here

Bachelor Thesis

submitted by

Alexander Roth

Supervised by

Prof. Dr. Immanuel Bloch

Chair of Quantum Optics, Ludwig-Maximilians-Universität München
and Max Planck Institute of Quantum Optics

Dr. Titus Franz, Dr. Philipp Preiss

Max Planck Institute of Quantum Optics

Munich, Month Day, 2025

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Theoretical Background

2.1 Trapping on a Lattice

An optical lattice is being created by a standing laser light which acts as a periodic potential due to the Stark Shift.

2.2 Gates

To run quantum algorithms on a lattice, one needs to perform gates on it. Experimentally, not all gates are equally easy to implement.

2.3 Fermions vs. Qubits

2.4 Fermi-Hubbard Model

2.5 Jordan-Wigner Mapping

Chapter 3

Implementing a N-dimensional Unitary on a Lattice

3.1   Interferometry

In interferometry, Reck et al. have shown in 1994 that any $N \times N$ Unitary matrix can be implemented using only 50:50 beam splitters and phase shifters. This is done by decomposing the U(N) matrix using U(2) matrices acting on a 2-dimensional subspace of the Hilbert space [?]. These U(2) matrices can be implemented in the optical setup by using two 50:50 beam splitters and two phase shifters acting on wires n,m that can be represented by the following matrix:

$$
T_{n,m}(\theta, \phi) =
\begin{pmatrix}
1 & 0 & \cdots & & & & & 0 \\
0 & 1 & & & & & & \vdots \\
\vdots & & \ddots & & & & & \\
& & & e^{i\phi}\cos\theta & -\sin\theta & & & \\
& & & e^{i\phi}\sin\theta & \cos\theta & & & \\
& & & & & \ddots & & \\
& & & & & & 1 & 0 \\
0 & \cdots & & & & & 0 & 1
\end{pmatrix}_{N \times N}
\tag{3.1}
$$

Mathematically speaking, the matrix $T_{n,m}(\theta, \phi)$ resembles a Givens Rotation acting on wires n,m. A Givens rotation is a rotation in a plane and, if applied from the right, mixes two columns in the same row. It is often used to null specific elemtents of a matrix in numerical methods. Following the QR decomposition algorithm, we can find a decomposition of U into an upper triangular matrix (R) and a number of Givens rotations

(Q):

$$U(N) = R * Q \tag{3.2}$$

Since U is unitary, R is not only an upper diagonal matrix but has to be a diagonal matrix with modulus 1 and complex entries which we will call D in the following.

The decomposition of an arbitrary Unitary of dimension N in a diagonal matrix and Givens Rotations is therefore:

$$U(N) = D \left(\prod_{(m,n)\in S} T_{m,n} \right) \; ; \; n, m \, \epsilon \, [0, N-1] \tag{3.3}$$

where S is a specific ordered sequence of two-mode transformations [?].

The order in which the Givens rotations are applied is not arbitrary. One starts from the bottom-left corner and moves to the nearest subdiagonal, moving from top to bottom [?]. This is also illustrated in Figure ??.

3.2   Application on a Quantum Circuit

In the context of quantum simulation this can be represented using Pauli-Z ($\hat{Z}$) and Pauli-X ($\hat{X}$) Rotations. The $\hat{X}$ gate resembles the beam splitter and the $\hat{Z}$ gate the phase shift.

$$\hat{X}(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & i \sin(\frac{\theta}{2}) \\ i \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}, \qquad \hat{Z}(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}. \tag{3.4}$$

For $\hat{X}(\theta)$ to resemble a 50:50 beam splitter, we set $\theta = \pi/2$. The angle for the $\hat{Z}$-gate is freely choosable, providing the necessary degrees of freedom to resemble any matrix U(N). One can easily show, that

$$T_{n,m}(\theta, \phi) = e^{i\alpha} \hat{X}(\pi/2) \hat{Z}(\omega) \hat{X}(\pi/2) \hat{Z}(\psi) \tag{3.5}$$

since

$$e^{i\alpha} \hat{X}(\pi/2) \hat{Z}(\omega) \hat{X}(\pi/2) \hat{Z}(\psi) = i * e^{i(\alpha+\psi/2)} \begin{pmatrix} \sin(\omega/2)\, e^{-i\psi} & \cos(\omega/2) \\ \cos(\omega/2)\, e^{-i\psi} & \sin(\omega/2) \end{pmatrix}. \tag{3.6}$$

We see that by choosing $\alpha = -\pi + \phi/2$, $\omega = 2\theta - \pi$ and $\psi = -(\pi + \phi)$ we get $T_{n,m}(\theta, \phi)$. We can therefore find the decomposition of any Unitary into a combination of X2Z2 acting on two spatial modes in the lattice with two global tunneling gates ($\hat{X}(\pi/2)$) and

5dim_example_decomposition.png

Figure 3.1: Example circuit to decompose a 5 dimensional Fourier transformation (or any five dimensional unitary matrix)

two $\hat{Z}$-rotations. Both gates act on the 2 qubit subspace $\{|01\rangle, |10\rangle\}$. To implement this in PennyLane, the *givens_ decomposition* function from the PennyLane Python library, v.0.40.0, was used to create the decomposition of an arbitrary input unitary in equation **??**. The code for the decomposition of the single Givens rotations into X2Z2 can be seen in Appendix **??**.

It is important to note, that all calculations are in the non-interacting case. Therefore, we can limit ourselves to the N-dimensional non-interacting subspace of the $2^N \times 2^N$ total Hilbert space.

3.3   Use Cases

The use cases of being able to implement any unitary matrix on the lattice are numerous. To measure momentum space, a standard method is to switch the optical lattice and harmonic trapping potential off and perform a time-of-flight imaging. This method, though, has a number of limitations. The measuring quality is affected by the inhomogeneity of the trapping potential and the accuracy of the absolute number of atoms is around $\pm 10\%$ [**?**]. With the above described algorithm, we are able to implement a discrete Fourier Transformation on the lattice. This allows to measure momentum space directly without

workarounds and would only be limited by the gate fidelities.

Chapter 4

Methods

Chapter 5

Results

Chapter 6

Discussion

Chapter 7

Conclusion and Outlook

Appendix A

Decomposition into XZXZ

```python
1  import numpy as np
2  from scipy import optimize
3  import pennylane as qml
4
5  def normalize_angle(angle):
6      """
7      Normalizes an angle into the interval [0, 4*pi].
8      """
9      return np.mod(angle, 4*np.pi)
10
11 # ----------------------------------------------------------------------------
12 # Helper: Convert a 2x2 unitary U into an SU(2) matrix by factoring out a global phase
         .
13 # Returns U_su2 and the phase factor such that U = exp(i*alpha) * U_su2.
14 def convert_to_su2(U, return_global_phase=True):
15     U = np.array(U, dtype=complex)
16     det = np.linalg.det(U)
17     # Let alpha be such that U = exp(i*alpha) * (U * exp(-i*alpha)) has determinant 1.
18     alpha = np.angle(det) / 2
19     alpha = normalize_angle(alpha)
20     U_su2 = U * np.exp(-1j * alpha)
21     if return_global_phase:
22         return U_su2, alpha
23     return U_su2
24
25 # ----------------------------------------------------------------------------
26 # 1. Decomposition of the form:
27 # U = phase * RZ(psi) RX(phi) RZ(theta)
28 # We derive an analytic solution by writing out the matrix products.
29 def ZXZ_decomp(U, wire, return_global_phase=False):
30     U_su2, alpha = convert_to_su2(U, return_global_phase=True)
31     # Let U_su2 = [ a b ]
32     #            [ c d ]
```

```python
33     # and note that for our decomposition the product is:
34     # RZ(psi) RX(phi) RZ(theta) =
35     # [ cos(phi/2) exp(-i(psi+theta)/2) -i sin(phi/2) exp(-i(psi-theta)/2) ]
36     # [ -i sin(phi/2) exp(i(psi-theta)/2) cos(phi/2) exp(i(psi+theta)/2) ]
37     #
38     # Hence we can set:
39     # phi = 2 arctan2(|b|, |a|)
40     # psi+theta = -2 arg(a)
41     # psi-theta = -2 ( arg(b) + pi/2 )
42     a = U_su2[0, 0]
43     b = U_su2[0, 1]
44
45     # Calculate angles
46     phi = 2 * np.arctan2(np.abs(b), np.abs(a))
47     psi_plus_theta = -2 * np.angle(a)
48     psi_minus_theta = -2 * (np.angle(b) + np.pi/2)
49
50     psi = 0.5 * (psi_plus_theta + psi_minus_theta)
51     theta = 0.5 * (psi_plus_theta - psi_minus_theta)
52
53     # Normalize all angles into the interval [0, 4*pi]
54     phi = normalize_angle(phi)
55     psi = normalize_angle(psi)
56     theta = normalize_angle(theta)
57
58     ops = []
59     if return_global_phase:
60         ops.append(qml.GlobalPhase(alpha))
61     ops.extend([
62         qml.RZ(psi, wires=wire),
63         qml.RX(phi, wires=wire),
64         qml.RZ(theta, wires=wire)
65     ])
66     return ops
67
68 # ----------------------------------------------------------------------------
69 # 2. Decomposition of the form:
70 # U = phase * RX(pi/2) RZ(phi) RX(pi/2) RZ(psi)
71 # Here the two RX(pi/2) rotations are fixed; we numerically determine the free angles
       phi and psi.
72 def XZXZ_decomp(U, wire, return_global_phase=False):
73     U_su2, alpha = convert_to_su2(U, return_global_phase=True)
74
75     # Define the basic gates with PennyLane's matrix definitions.
76     def RX(theta):
77         return np.array([[np.cos(theta/2), -1j*np.sin(theta/2)],
78                          [-1j*np.sin(theta/2), np.cos(theta/2)]], dtype=complex)
```

```
79
80    def RZ(theta):
81        return np.array([[np.exp(-1j*theta/2), 0],
82                         [0, np.exp(1j*theta/2)]], dtype=complex)
83
84    # Fixed gate RX(pi/2)
85    F = RX(np.pi/2)
86
87    # Define the candidate matrix in SU(2) (ignoring the overall phase)
88    def M(params):
89        phi, psi = params
90        # Circuit order: first RX(pi/2), then RZ(phi), then RX(pi/2), then RZ(psi)
91        return F @ RZ(phi) @ F @ RZ(psi)
92
93    def cost(params):
94        return np.linalg.norm(M(params) - U_su2)**2
95
96    # Initial guess
97    initial_guess = np.array([0.0, 0.0])
98    res = optimize.minimize(cost, initial_guess, method="BFGS")
99    phi_opt, psi_opt = res.x
100
101    # Normalize angles
102    phi_opt = normalize_angle(phi_opt)
103    psi_opt = normalize_angle(psi_opt)
104
105    ops = []
106    if return_global_phase:
107        ops.append(qml.GlobalPhase(alpha))
108    ops.extend([
109        qml.RX(np.pi/2, wires=wire),
110        qml.RZ(phi_opt, wires=wire),
111        qml.RX(np.pi/2, wires=wire),
112        qml.RZ(psi_opt, wires=wire)
113    ])
114    return ops
115
116 # ----------------------------------------------------------------------------
117 # 3. Decomposition of the form:
118 # U = phase * RZ(psi) RX(pi/2) RZ(phi) RX(pi/2) RZ(theta)
119 # Again we have fixed RX(pi/2) gates. We now determine the free angles psi, phi, theta
       .
120 def ZXZXZ_decomp(U, wire, return_global_phase=False):
121     U_su2, alpha = convert_to_su2(U, return_global_phase=True)
122
123     def RX(theta):
124         return np.array([[np.cos(theta/2), -1j*np.sin(theta/2)],
```

```python
125                             [-1j*np.sin(theta/2), np.cos(theta/2)]], dtype=complex)
126
127     def RZ(theta):
128         return np.array([[np.exp(-1j*theta/2), 0],
129                          [0, np.exp(1j*theta/2)]], dtype=complex)
130
131     F = RX(np.pi/2)
132
133     def M(params):
134         psi, phi, theta_val = params
135         # Circuit order: first RZ(psi), then RX(pi/2), then RZ(phi), then RX(pi/2),
                  then RZ(theta)
136         return RZ(psi) @ F @ RZ(phi) @ F @ RZ(theta_val)
137
138     def cost(params):
139         return np.linalg.norm(M(params) - U_su2)**2
140
141     initial_guess = np.array([0.0, 0.0, 0.0])
142     res = optimize.minimize(cost, initial_guess, method="BFGS")
143     psi_opt, phi_opt, theta_opt = res.x
144
145     # Normalize angles
146     psi_opt = normalize_angle(psi_opt)
147     phi_opt = normalize_angle(phi_opt)
148     theta_opt = normalize_angle(theta_opt)
149
150     ops = []
151     if return_global_phase:
152         ops.append(qml.GlobalPhase(alpha))
153     ops.extend([
154         qml.RZ(psi_opt, wires=wire),
155         qml.RX(np.pi/2, wires=wire),
156         qml.RZ(phi_opt, wires=wire),
157         qml.RX(np.pi/2, wires=wire),
158         qml.RZ(theta_opt, wires=wire)
159     ])
160     return ops
161
162 # ------------------------------------------------------------------------------
163 # Dispatch function: choose the desired decomposition form.
164 def one_qubit_decomposition(U, wire, rotations="ZYZ", return_global_phase=False):
165     supported_rotations = {
166         "ZXZ": ZXZ_decomp,
167         "XZXZ": XZXZ_decomp,
168         "ZXZXZ": ZXZXZ_decomp,
169     }
170
```

```python
171    if rotations in supported_rotations:
172        return supported_rotations[rotations](U, wire, return_global_phase)
173
174    raise ValueError(
175        f"Value {rotations} passed to rotations is either invalid or currently
               unsupported."
176    )
```

Listing A.1: Python code for $one_q ubit_d ecomposition$.