Chapter 6

Quantum Teleportation using Qiskit

Now we will see quantum teleportation using Qiskit in this chapter .
Qiskit stands for Quantum Information Software Kit for Quantum Computation. Qiskit is an open-source software development kit (SDK) for working with quantum computers at the level of circuits, pulses, and algorithms. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Experience or on simulators on a local computer. It follows the circuit model for universal quantum computation, and can be used for any quantum hardware (currently supports superconducting qubits and trapped ions).

For this we will discuss the whole process in several step for the easily understanding. Many comments are written in the codes to explain the next steps in brief. So, here we go -

Importing necessary libraries:

[1]:
```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import Aer, execute
from qiskit.visualization import plot_bloch_multivector, array_to_latex
from qiskit.extensions import Initialize
from qiskit.quantum_info import random_statevector
import qiskit.quantum_info as qi
```

Initializing a random qubit state:

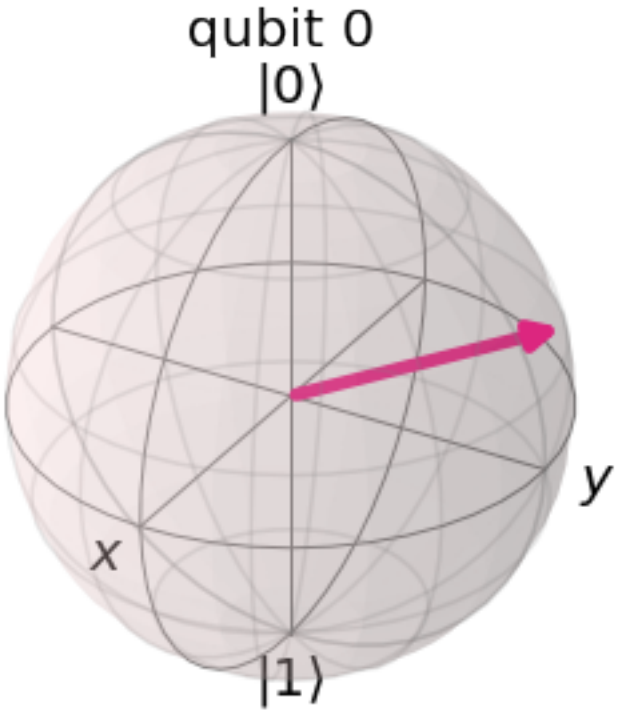
[2]:
```
# Create random 1-qubit state
psi = random_statevector(2)

#print('psi state: {}'.format(psi))

# Display  in LaTeX
display(array_to_latex(psi,prefix = "|\\psi\\rangle ="))
```

$$|\psi\rangle = \begin{bmatrix} 0.40416 - 0.6831i & 0.27831 + 0.54091i \end{bmatrix}$$

[3]:
```
#|ψ⟩ vector in terms of standard basis
display(psi.draw('latex'))
```

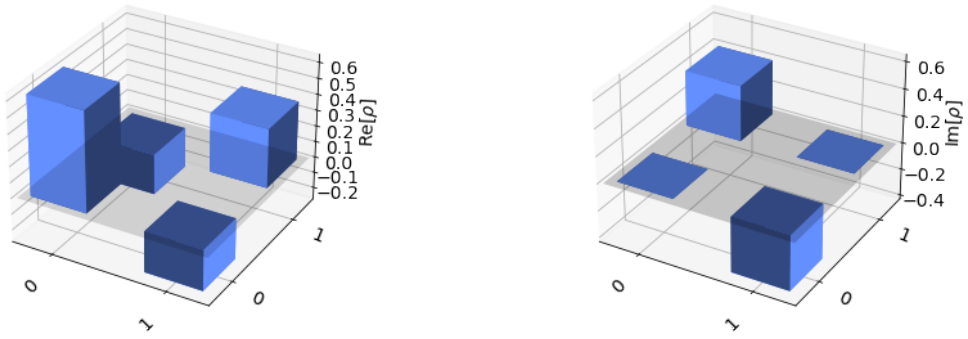
$$(0.404156195507 - 0.683098786632i)|0\rangle + (0.278307579725 + 0.540905452369i)|1\rangle$$

[4]:
```
# Visualize the generated random qubit state in Bloch sphere
display(plot_bloch_multivector(psi))
```



[5]:
```
# Visualize the generated random qubit state in 3D version of Density Matrix
psi.draw('city')
```

[5]:



[6]:
```
# Visualize the generated random qubit state in Density Matrix
qq1 = qi.DensityMatrix(psi)
display(qq1.draw('latex'))
```

$$\begin{bmatrix} 0.62997 & -0.25701 - 0.40872i \\ -0.25701 + 0.40872i & 0.37003 \end{bmatrix}$$

[7]:
```
# Define initialization gate to create |ψ⟩ from the state |0⟩
init_state = Initialize(psi)
init_state.label = "initial_state"
```

Essential functions for quantum teleportation protocol:

[8]:
```
def create_bell_pair(qc, a, b):
    """Creates a bell pair in qc using qubits a & b"""
    qc.h(a) # Put qubit a into state |+>
    qc.cx(a,b) # CNOT with a as control and b as target
```

[9]:
```
def alice_gates(qc, psi, a):
    qc.cx(psi, a)
    qc.h(psi)
```

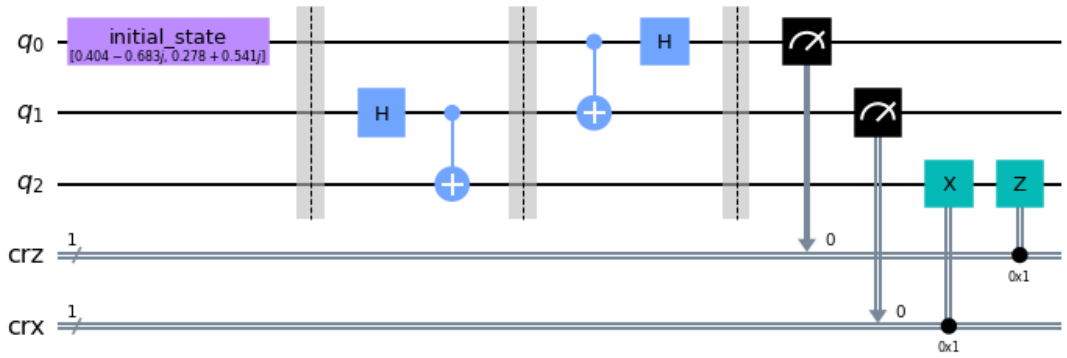
[10]:
```
def measure_and_send(qc, a, b):
    """Measures qubits a & b and 'sends' the results to Bob"""
    qc.barrier()
    qc.measure(a,0)
    qc.measure(b,1)
```

[11]:
```
# This function takes a QuantumCircuit (qc), integer (qubit)
# and ClassicalRegisters (crz & crx) to decide which gates to apply
def bob_gates(qc, qubit, crz, crx):
    # Here we use c_if to control our gates with a classical
    # bit instead of a qubit
    qc.x(qubit).c_if(crx, 1) # Apply gates if the registers
    qc.z(qubit).c_if(crz, 1) # are in the state '1'
```

Executing Quantum Teleportation Protocol:

[12]:
```
## SETUP
qr = QuantumRegister(3, name="q")   # Protocol uses 3 qubits
crz = ClassicalRegister(1, name="crz") # and 2 classical registers
crx = ClassicalRegister(1, name="crx")
qc = QuantumCircuit(qr, crz, crx)

## STEP 0
# First, let's initialize Alice's q0
qc.append(init_state, [0])
qc.barrier()

## STEP 1
# Now begins the teleportation protocol
create_bell_pair(qc, 1, 2)
qc.barrier()

## STEP 2
# Send q1 to Alice and q2 to Bob
alice_gates(qc, 0, 1)

## STEP 3
# Alice then sends her classical bits to Bob
measure_and_send(qc, 0, 1)

## STEP 4
# Bob decodes qubits
bob_gates(qc, 2, crz, crx)

# Display the circuit
qc.draw('mpl')
```

[12]:



Simulating our Quantum Teleportation Circuit:

[13]:
```
# Let's see the result

# To get the eigenvector you should use the statevector simulator in the core
of the circuit (without measurements)
backend = Aer.get_backend('statevector_simulator')

# Execute the circuit
result = execute(qc, backend).result().get_statevector(qc, decimals=3)

# Printing the state after X gate
print("Quantum state is:")
display(array_to_latex(psi,prefix = "|\\psi\\rangle ="))
print()
print()
print("|ψ⟩ vector in terms of standard basis:")
display(result.draw('latex'))
print()
print()
# Visualize the result state in 3D version of Density Matrix
display(result.draw('city'))
print()
print()
print("The Density Matrix is:")
DensityMatrix = qi.DensityMatrix(result)
display(DensityMatrix.draw('latex'))
print()
print()
print("Plotting the Bloch Sphere:")
plot_bloch_multivector(result)
```

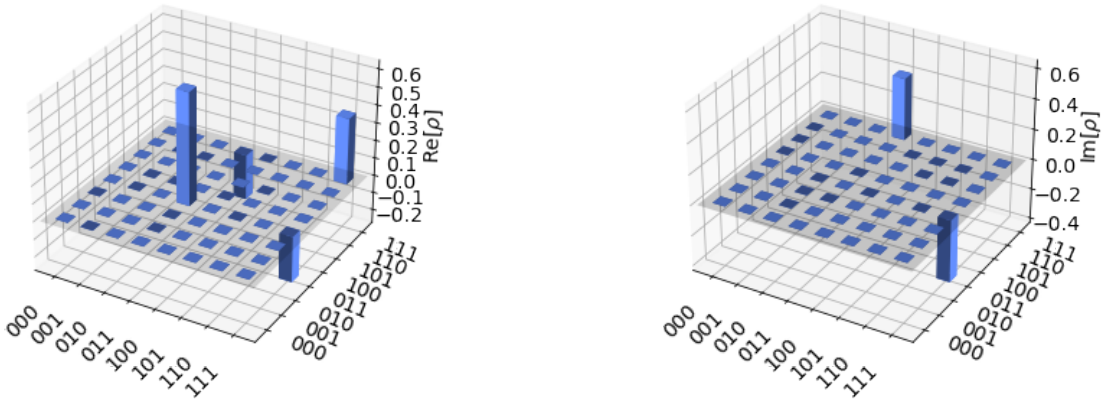
Quantum state is:

$$|\psi\rangle = \begin{bmatrix} 0.40416 - 0.6831i & 0.27831 + 0.54091i \end{bmatrix}$$

|ψ⟩ vector in terms of standard basis:

$$(0.404 - 0.683i)|011\rangle + (0.278 + 0.541i)|111\rangle$$

The Density Matrix visualization is:



The Density Matrix is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.62971 & 0 & 0 & 0 & -0.25719 - 0.40844i \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.25719 + 0.40844i & 0 & 0 & 0 & 0.36997 \end{bmatrix}$$

Plotting the Bloch Sphere:

[13]: