Implementation Details

Single Qubit

Overview of qubit class located in SingleQubit.py:

```
class Qubit:

def __init__(self, vec)
"""
input: vec = [rx, ry, rz]
Blochvector entries:
rho = 0.5*(I + vec*sigma_vector)
"""
def get_matrix(self)
# outputs current density matrix rho

def _set_Bloch_vector(self, vec)
# sets new state via new Bloch vector

def get_Bloch_vector(self)
# outputs current Bloch vector
```

Single Qubit Gates

Single qubit gates located in SingleQubitGates.py:

```
def operator_sum(blochvector, list)
# performs operator sum
# with operators in list
# on density matrix defined by the Bloch vector
```

Gate function	Matrices E_k for $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$
`def I_gate(blochvector)`	$E_0 = I$
`def X_gate(blochvector)`	$E_0 = \sigma_1$
`def Y_gate(blochvector)`	$E_0 = \sigma_2$
`def Z_gate(blochvector)`	$E_0 = \sigma_3$
`def H_gate(blochvector)`	$E_0 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
`def S_gate(blochvector)`	$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$
`def T_gate(blochvector)`	$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$
`def amplitude_damping(blochvector, Gamma, t)`	$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & e^{-0.5 \cdot \mathtt{Gamma} \cdot \mathtt{t}} \end{pmatrix}$ $E_1 = \begin{pmatrix} 0 & \sqrt{1 - e^{-\mathtt{Gamma} \cdot \mathtt{t}}} \\ 0 & 0 \end{pmatrix}$
`def phase_damping(blochvector, Gamma, t)`	$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & e^{-\mathtt{Gamma} \cdot \mathtt{t}} \end{pmatrix}$ $E_1 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{1 - e^{-2 \cdot \mathtt{Gamma} \cdot \mathtt{t}}} \end{pmatrix}$
`def bit_flip(blochvector, p)`	$E_0 = \sqrt{1 - \mathtt{p}} \cdot I$ $E_1 = \sqrt{\mathtt{p}} \cdot \sigma_1$
`def phase_flip(blochvector, p)`	$E_0 = \sqrt{1 - \mathtt{p}} \cdot I$ $E_1 = \sqrt{\mathtt{p}} \cdot \sigma_3$
`def bit_phase_flip(blochvector, p)`	$E_0 = \sqrt{1 - \mathtt{p}} \cdot I$ $E_1 = \sqrt{\mathtt{p}} \cdot \sigma_2$
`def depolarization(blochvector, p)`	$E_0 = \sqrt{1 - \mathtt{p}} \cdot I$ $E_1 = \sqrt{\frac{\mathtt{p}}{3}} \cdot \sigma_1$ $E_2 = \sqrt{\frac{\mathtt{p}}{3}} \cdot \sigma_2$ $E_3 = \sqrt{\frac{\mathtt{p}}{3}} \cdot \sigma_3$

Table 1: Single qubit gate functions defined in file SingleQubitGates.py of Quantum Circuit Simulation project

Two Qubits

Overview of qubit class located in TwoQubits.py:

```
class Qubit:

def __init__(self, vec1, vec2):
"""
input:

r = [rx, ry, rz]
Blochvector entries:
rho = 0.5*(I + r*sigma_vector)

uses:
a00 ... a03
coef_matrix =    .   .     .
.     . .
a30 ... a33

where
rho = a00*tensor(I, I)
+ a01*tensor(I, sigmaX)
+ ...
+ a33*tensor(sigmaZ, sigmaZ)

and
tensor(a,b) = tensorproduct of a and b
"""

def get_matrix(self):
# outputs current density matrix rho

def set_Pauli_basis_matrix(self, matrix):
# sets new state in Pauli basis
```

Two Qubit Gates

Two qubit gates located in file TwoQubitGates.py:

```
def operator_sum(rho, list)
# performs operator sum
# with operators in list
# on density matrix rho
```

Gate function	Matrices E_k for $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$
`def I2_gate(coef_matrix, qubitnumber)`	$E_0 = I \otimes I$
`def X2_gate(coef_matrix, qubitnumber)`	$E_0 = \begin{cases} \sigma_1 \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes \sigma_1 & \texttt{qubitnumber} = 2 \end{cases}$
`def Y2_gate(coef_matrix, qubitnumber)`	$E_0 = \begin{cases} \sigma_2 \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes \sigma_2 & \texttt{qubitnumber} = 2 \end{cases}$
`def Z2_gate(coef_matrix, qubitnumber)`	$E_0 = \begin{cases} \sigma_3 \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes \sigma_3 & \texttt{qubitnumber} = 2 \end{cases}$
`def H2_gate(coef_matrix, qubitnumber)`	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ $E_0 = \begin{cases} H \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes H & \texttt{qubitnumber} = 2 \end{cases}$
`def S2_gate(coef_matrix, qubitnumber)`	$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$ $E_0 = \begin{cases} S \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes S & \texttt{qubitnumber} = 2 \end{cases}$
`def T2_gate(blochvector)`	$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$ $E_0 = \begin{cases} T \otimes I & \texttt{qubitnumber} = 1 \\ I \otimes T & \texttt{qubitnumber} = 2 \end{cases}$
`def CNOTgate(coef_matrix)`	$E_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
`def CZgate(coef_matrix)`	$E_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
`def SWAPgate(coef_matrix)`	$E_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
`def R2_gate(coef_matrix)`	$E_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}$

Table 2: Two qubit gate functions defined in file TwoQubitGates.py of Quantum Circuit Simulation project

Gate function	Matrices E_k for $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$
`def individual_amplitude_damping (coef_matrix, Gamma, t)`	$E_{single} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & \sqrt{1 - e^{-\texttt{Gamma} \cdot \texttt{t}}} \\ 0 & 0 \end{pmatrix}$ $E_1 = E_{single} \otimes I$ $E_2 = I \otimes E_{single}$ $E_0 = \sqrt{I - \sum_i^2 E_i^\dagger E_i}$
`def fully_correlated_amplitude_damping (coef_matrix, Gamma, t)`	$E_{single} = \begin{pmatrix} 0 & \sqrt{1 - e^{-\texttt{Gamma} \cdot \texttt{t}}} \\ 0 & 0 \end{pmatrix}$ $E_1 = E_{single} \otimes E_{single}$ $E_0 = \sqrt{I - E_1^\dagger E_1}$
`def individual_phase_damping (coef_matrix, Gamma, t)`	$E_{single} = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{1 - e^{-2 \cdot \texttt{Gamma} \cdot \texttt{t}}} \end{pmatrix}$ $E_1 = E_{single} \otimes I$ $E_2 = I \otimes E_{single}$ $E_0 = \sqrt{I - \sum_i^2 E_i^\dagger E_i}$
`def fully_correlated_phase_damping (coef_matrix, Gamma, t)`	$E_{single} = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{1 - e^{-2 \cdot \texttt{Gamma} \cdot \texttt{t}}} \end{pmatrix}$ $E_1 = E_{single} \otimes E_{single}$ $E_0 = \sqrt{I - E_1^\dagger E_1}$
`def individual_phase_flip (coef_matrix, p)`	$E_1 = \sqrt{p}\sigma_3 \otimes I$ $E_2 = I \otimes \sqrt{p}\sigma_3$ $E_0 = \sqrt{I - \sum_i^2 E_i^\dagger E_i}$
`def fully_correlated_phase_phase (coef_matrix, p)`	$E_1 = \sqrt{p}\sigma_3 \otimes \sqrt{p}\sigma_3$ $E_0 = \sqrt{I - E_1^\dagger E_1}$
`def individual_bit_flip (coef_matrix, p)`	$E_1 = \sqrt{p}\sigma_1 \otimes I$ $E_2 = I \otimes \sqrt{p}\sigma_1$ $E_0 = \sqrt{I - \sum_i^2 E_i^\dagger E_i}$
`def fully_correlated_bit_phase (coef_matrix, p)`	$E_1 = \sqrt{p}\sigma_1 \otimes \sigma_1$ $E_0 = \sqrt{I - E_1^\dagger E_1}$
`def individual_bit_phase_flip (coef_matrix, p)`	$E_1 = \sqrt{p}\sigma_2 \otimes I$ $E_2 = I \otimes \sqrt{p}\sigma_2$ $E_0 = \sqrt{I - \sum_i^2 E_i^\dagger E_i}$
`def fully_correlated_bit_phase_phase (coef_matrix, Gp)`	$E_1 = \sqrt{p}\sigma_2 \otimes \sqrt{p}\sigma_2$ $E_0 = \sqrt{I - E_1^\dagger E_1}$

Table 3: Two qubit noise gate functions defined in file TwoQubitGates.py of Quantum Circuit Simulation project

All other single qubit gates can be easily combined to two qubit gates by using helper functions in helper_functions_2Qubits.py and the above operator sum function.

```python
def get_rho_from_Pauli_basis(coef_matrix)
# converts coefficient matrix coef_matrix
# in Pauli basis to density matrix rho

def get_Pauli_basis_from_rho(res)
# converts density matrix res
# to Pauli basis coefficient matrix
```

Single Qudit

Single qudit class is implemented in SingleQudit.py:

```
class Qudit:

def __init__(self, d, vec):
"""
input:
d = dimension
vec = [v1, v2, v3, ...]: Qudit
=> matrix = vec*vec.T
"""

def get_matrix(self):
# outputs current density matrix rho

def _set_matrix(self, mat):
# sets new density matrix

def get_dimension(self):
# outputs qudit dimension
```

Single Qudit Gates

Similar to above gates qudit gates, iplemented in SingleQuditGates.py, can be performed via operator sum:

```
def qudit_operator_sum(density, list):
# performs operator sum
# with operators in list
# on density matrix density
# and returns new density matrix
```

In constants.py generalizations of σ_1 and σ_3 and in equations **??** and **??** is given by:

```
def X(dim):
# returns generalization of sigma 1

def Z(dim):
# returns generalization of sigma 3
```

Gate function	**Matrices E_k for $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$**
def `I_gate(mat)`	$E_0 = \left(\begin{array}{c\|c} I & 0_{2\times(d-2)} \\ \hline 0_{(d-2)\times 2} & I_{(d-2)} \end{array}\right)$
def `X_gate(mat)`	$E_0 = \left(\begin{array}{c\|c} \sigma_1 & 0_{2\times(d-2)} \\ \hline 0_{(d-2)\times 2} & I_{(d-2)} \end{array}\right)$
def `Y_gate(mat)`	$E_0 = \left(\begin{array}{c\|c} \sigma_2 & 0_{2\times(d-2)} \\ \hline 0_{(d-2)\times 2} & I_{(d-2)} \end{array}\right)$
def `Z_gate(mat)`	$E_0 = \left(\begin{array}{c\|c} \sigma_3 & 0_{2\times(d-2)} \\ \hline 0_{(d-2)\times 2} & I_{(d-2)} \end{array}\right)$
def `H_gate(mat)`	$E_0 = \left(\begin{array}{c\|c} \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} & 0_{2\times(d-2)} \\ \hline 0_{(d-2)\times 2} & I_{(d-2)} \end{array}\right)$
def `S_gate(mat)`	$E_0 = \left(\begin{array}{cc\|c} 1 & 0 & \\ 0 & e^{i\frac{\pi}{2}} & 0_{2\times(d-2)} \\ \hline \multicolumn{2}{c\|}{0_{(d-2)\times 2}} & I_{(d-2)} \end{array}\right)$
def `T_gate(mat)`	$E_0 = \left(\begin{array}{cc\|c} 1 & 0 & \\ 0 & e^{i\frac{\pi}{4}} & 0_{2\times(d-2)} \\ \hline \multicolumn{2}{c\|}{0_{(d-2)\times 2}} & I_{(d-2)} \end{array}\right)$
def `general_bit_flip(mat, p)`	$E_1 = \sqrt{\mathtt{p}} X_{d\times d}$ $E_0 = \sqrt{1-\mathtt{p}} I_{d\times d}$
def `general_phase_flip(mat, p)`	$E_1 = \sqrt{\mathtt{p}} Z_{d\times d}$ $E_0 = \sqrt{1-\mathtt{p}} I_{d\times d}$
def `depolarization(mat, p)`	$E_{nm} = \frac{\sqrt{\mathtt{p}}}{d^2} W_{nm}, n, m \in \{0, ..., d-1\}$ $E_0 = \sqrt{1-\mathtt{p}} I$
def `single_Weyl_channel(mat, n, m, p)`	$E_1 = \frac{\sqrt{\mathtt{p}}}{d^2} W_{\mathtt{nm}}$ $E_0 = \sqrt{1-\mathtt{p}} I$
def `amplitude_damping(mat, gammalist)` """ $\mathtt{gammalist} = [\mathtt{gamma}_{01}, \mathtt{gamma}_{02}, ..., \mathtt{gamma}_{0d-1},$ $\mathtt{gamma}_{12}, ..., \mathtt{gamma}_{1d-1},$ $...,$ $\mathtt{gamma}_{(d-2)(d-1)}]$ """	$(E_{nm})_{ij} = \sqrt{\mathtt{gamma}_{nm}} \delta_{i,n} \delta_{j,m}$, $n, m \in \{0, ...d-1\}$ $E_0 = \sqrt{I - \sum_{n,m=0}^{d-1} E_{nm}^\dagger E_{nm}}$
def `general_phase_damping(mat, p)`	$E_k =$ $\sqrt{\binom{d-1}{k} \left(\frac{1-\mathtt{p}}{2}\right)^k \left(\frac{1+\mathtt{p}}{2}\right)^{d-1-k}} Z_{d\times d}^k$, $k \in \{0, ..., d-1\}$
def `physical_phase_damping(mat, gamma)`	$(E_k)_{i,j} = \sum_{j=0}^{d-1} \frac{\left[j\sqrt{-2\ln(\lambda)}\right]^m \lambda^{j^2}}{\sqrt{m!}} \delta_{i,j}$

Table 4: Single qudit gate functions with dimension d defined in file SingleQuditGates.py of Quantum Circuit Simulation project using density matrix `mat` as input