Config Summary

This document answers some of the broader questions about how the problem is setup to be solved by a reinforcement learning (RL) agent.

State Space

The state space in this problem is made of the expectation values of all single-qubit Pauli operators (X, Y, and Z) on the current quantum state ($|\psi\rangle$). For $N = 3$ qubit system, this would be a vector of size $3N$:

$$s = [\langle X_1\rangle, \langle Y_1\rangle, \langle Z_1\rangle, \langle X_2\rangle, \langle Y_2\rangle, \langle Z_2\rangle, \langle X_3\rangle, \langle Y_3\rangle, \langle Z_3\rangle]$$

Using these expectation values allows the RL agent to avoids the exponential scaling problem of using the full 2^N dimensional quantum state vector as input to the neural network.

Action Space

The action space is the set of quantum gates the agent can apply to the qubits at each time step. In my setup I'm only looking at a limited set of gates: Hadamard, Pauli X, CNOT, and two rotation gates around the X and Y axes by a $\pi/2$ angle. Each action corresponds to applying one of these gates to one or more qubits.

Episodes

An episode starts with a all-zero state $|0\rangle^{\otimes N}$, and the parameters such as fidelity and gate count are reset as well. Then, the agent selects an action (gate), the gate is applied to the circuit, and the environment returns the next state, reward, and done flag. An episode ends if either of the following conditions is met:

- The state fidelity between the current state and the target state exceeds the `FIDELITY_THRESHOLD`.
- The number of applied gates reaches the `MAX_GATES` limit.

Reward Structure

The reward structure is designed to encourage the agent to reach the target state efficiently.

The reward at each step is calculated based on the following components:

- **Baseline Reward**: A small negative reward (penalty) is given at each step to encourage the agent to find shorter circuits.
- **Incremental Reward**: The agent receives a reward or penalty based on the change in fidelity from the last step to the current step. This helps guide the agent towards improving fidelity over time.
- **Terminal Reward**: A large positive reward is given if the agent successfully reaches the target fidelity threshold.

Below is a pseudocode implementation of the reward calculation (non-functional):

```
# Constants
PENALTY_GATE = -0.01
REWARD_SUCCESS = 10.0
WEIGHT_FIDELITY = 5.0
THRESHOLD = 0.999

def reward(fidelity, last_fidelity):
    # 1. Baseline Reward: Penalize every step to encourage short circuits
    reward = PENALTY_GATE

    # 2. Incremental Reward: Reward/Penalize change in fidelity (Reward Shaping)
    fidelity_change = fidelity - last_fidelity
    reward += WEIGHT_FIDELITY * fidelity_change

    # 3. Terminal Reward: Large bonus for achieving the goal
    if fidelity >= THRESHOLD:
        reward += REWARD_SUCCESS
        done = True
    else:
        done = False

    # 4. Termination by Length (No extra reward/penalty, just ends the episode)
    if gate_count >= MAX_GATES:
        done = True

    return reward, done
```