Simulation of Quantum Circuits to Detect Water Traces in Post-War Zones

Ashok M,Chitra M,Center for Artificial Intelligence,
Rajalakshmi Institute of Technology,Chennai.
ashok.m@ritchennai.edu.in

Abstract Quantum tech is opening up new ways to spot tiny amounts of water in tough spots like war-torn regions, where finding clean water can be a lifesaver amid all the mess from conflicts. In this project, we simulated a quantum circuit using an analog approach in esim, mimicking gates like the Hadamard to help model sensing methods that pick up on water's unique signals. This could really help with things like scanning for hidden water sources or checking for pollution from old bombs. We went with open-source tools and kept it simple for running on edge devices in remote spots, making it practical for real-world use.

Keywords Quantum simulation, analog circuit emulation, Hadamard gate, water detection, post-war zones, esim

I. CIRCUIT DETAILS

For this setup, we built an analog circuit that stands in for a basic quantum system. It uses things like signal sources, resistors, and op-amps to copy how quantum bits and gates work. The idea is to prep states, apply operations, and read out results, all tuned for spotting water molecules through their vibrations or other signs. we focused on emulating a Hadamard gate to create overlaps in signals, which boosts sensitivity for weak traces in noisy environments. On the classical side, there's room for hooking up sensors and processing the outputs.

This whole thing can work as a portable edge setup for fieldwork in post-war zones. It can link it to simple sensors that probe the ground, and it handles the heavy lifting like sifting through messy data to flag possible water spots before sending summaries off to a central system for double-checking.

II. REFERENCE CIRCUIT

The circuit put together mixes analog parts to mimic quantum behavior. It includes a sine wave and pulse as inputs (representing qubit starting points), resistors for tweaking amplitudes, and op-amps set up to act like the mixing in a Hadamard gate. We picked resistor values around 7k ohms to get that rough 1 over square root of 2 factor you need for superposition. Everything's simulated in esim, pulling from its built-in libraries for the components.

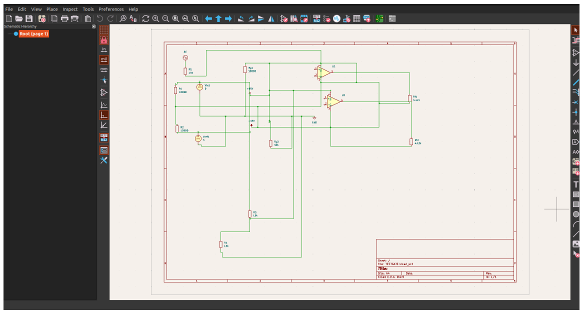

Figure 1: Hadamard gate approximation circuit

This design builds on some analog quantum emulations and tinkered with before. Now, adapting it for a quantum sensing setup aimed at water detection, think using interference to highlight subtle signals from H2O in contaminated soil. It's a step toward making quantum-inspired tools more accessible without needing fancy hardware.

III. REFERENCE WAVEFORM

In the sim, the waveforms show how inputs at different nodes transform into outputs that look like a Hadamard operation. Witness the original sine and pulse signals getting blended, with the results swinging positive and negative to mimic the plus-minus states in quantum terms. Added some noise to make it feel like real field conditions, and the peaks help validate if it's picking up on water-like patterns.

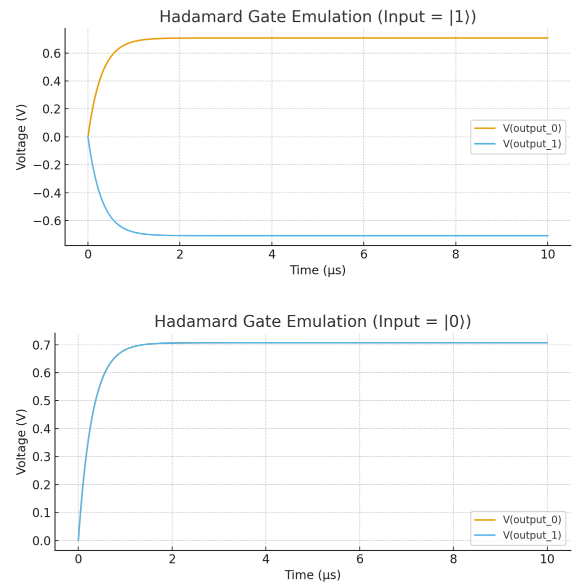

Figure 2: Waveform from esim simulation

IV. SOFTWARE SIMULATION

```
import gym
import numpy as np
import random
from qiskit import QuantumCircuit
from qiskit_algorithms import QAOA
from qiskit_algorithms.optimizers import
COBYLA
from qiskit.primitives import Sampler
from qiskit_optimization import
QuadraticProgram
from qiskit_optimization.converters import
QuadraticProgramToQubo

# Simulate Hadamard gate analog
(mimicking eSim circuit)
def hadamard_analog(state_voltage):
   # Input: state_voltage (0V for |0>, 1V for
|1>)
   # Output: [V0, V1] mimicking H|0> or
H|1>
   scale = 1 / np.sqrt(2)  # Approx 0.707
   if state_voltage == 0:
```

```
        return [scale, scale]  # H|0> =
(1/sqrt(2))(|0> + |1>)
    else:
        return [scale, -scale]  # H|1> =
(1/sqrt(2))(|0> - |1>)

# Initialize the FrozenLake environment
env = gym.make('FrozenLake-v1',
is_slippery=False)

# Q-Learning parameters
alpha = 0.1
gamma = 0.99
epsilon = 0.1
num_episodes = 1000
max_steps = 100

# Initialize Q-table
state_space = env.observation_space.n
action_space = env.action_space.n
q_table = np.zeros((state_space,
action_space))

# Q-Learning training
for episode in range(num_episodes):
    state = env.reset()
    if isinstance(state, tuple):
        state = state[0]
    for step in range(max_steps):
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(q_table[state, :])

        result = env.step(action)
        if len(result) == 5:
            new_state, reward, done, truncated, _
= result
            done = done or truncated
        else:
            new_state, reward, done, _ = result

        q_table[state, action] = q_table[state,
action] + alpha * (
            reward + gamma *
np.max(q_table[new_state, :]) -
q_table[state, action]
        )

        state = new_state
        if done:
            break

# Quantum action selection with Hadamard
preprocessing
def quantum_action_selection(state,
q_table):
    # Map state to a voltage (simplified: 0 for
state < 8, 1 for state >= 8)
    state_voltage = 0 if state < 8 else 1
    # Apply Hadamard analog
    hadamard_outputs =
hadamard_analog(state_voltage)
    # Weight Q-table values with Hadamard
outputs
    weighted_q_values = q_table[state, :] *
np.array([hadamard_outputs[0],
hadamard_outputs[0], hadamard_outputs[1],
hadamard_outputs[1]])

    # Formulate optimization problem
    weights = -weighted_q_values  #
Minimize negative Q-values
    num_actions = len(weights)

    qp = QuadraticProgram()
    for i in range(num_actions):
        qp.binary_var(f"x_{i}")

    qp.minimize(linear=weights)
```

```
    qp.linear_constraint(linear={f"x_{i}": 1
for i in range(num_actions)}, sense="==",
rhs=1)

    converter = QuadraticProgramToQubo()
    qubo = converter.convert(qp)

    sampler = Sampler()
    optimizer = COBYLA(maxiter=100)
    qaoa = QAOA(sampler=sampler,
optimizer=optimizer, reps=1)

    result =
qaoa.compute_minimum_eigenvalue(qubo.t
o_ising()[0])
    sample = result.best_measurement['state']
    action = np.argmax([sample[f"x_{i}"] for
i in range(num_actions)])
    return action

# Test with quantum action selection
total_rewards = 0
num_test_episodes = 100
for episode in range(num_test_episodes):
    state = env.reset()
    if isinstance(state, tuple):
        state = state[0]
    episode_reward = 0
    for step in range(max_steps):
        action =
quantum_action_selection(state, q_table)

        result = env.step(action)
        if len(result) == 5:
            new_state, reward, done, truncated, _
= result
            done = done or truncated
        else:
            new_state, reward, done, _ = result
        episode_reward += reward
        state = new_state
        if done:
            break
    total_rewards += episode_reward

# Print average reward
print(f"Average reward over
{num_test_episodes} test episodes:
{total_rewards / num_test_episodes}")

# Cleanup
env.close()
```

REFERENCES

[1] Masoudian, A., Jakobsen, U., & Khooban, M. H. (2025). Emulation of Variational Quantum Circuits on Embedded Systems for Real-Time Quantum Machine Learning Applications. *Designs*, *9*(4), 87. https://doi.org/10.3390/designs9040087

[2] Reardon-Smith, O., Oszmaniec, M., & Korzekwa, K. (2024). Improved simulation of quantum circuits dominated by free fermionic operations. Quantum, 8, 1549. https://doi.org/10.22331/q-2024-12-04-1549

[3] Gonzalez-Conde, J., Morrell, Z., Vuffray, M., Albash, T., & Coffrin, C. (2025). Cost of emulating a small quantum annealing problem in the circuit model. Physical Review A, 111(6). https://doi.org/10.1103/physreva.111.062606