GHZ State Preparation with DDrf Ultimately Implementing CRX Gate

Donghun Jung

24 Oct 2025

Department of Physics, Sungkyunkwan University

Center for Quantum Technology QuiME Lab, Korea

Institute of Science Technology

Outline

- [Quick Recap] DDrf
- [Quick Recap] How DDrf can realize GHZ state
- Machine Learning Approach
    - Why it fails?
- Strategy
    - Analytical analysis for good parameters
    - New Schrödinger equation solver with PyTorch
- Implementation of Runge-Kutta 7
    - Why Euler Method fails
    - Key features of RK7 implementation
- Dynamic Decoupling & DDrf explanation
- [illegible]

NV Center System

Spatially separated magnetic dipoles interact with each other via magnetic dipolar interaction (i.e., magnetic dipole-dipole interaction), which dominates the hyperfine coupling between the NV$^-$ electron spin and surrounding ^{13}C nuclear spins.

NV Center System

Note: The nuclear spin precession axes depend on the NV electron spin state.

Hamiltonian for NV coupled with N ^{13}C nuclear spins

(Electron spin in rotating frame, others are in lab frame)

Note: We use two level space in NV electrons

Available Control Sources

Microwave for Electron Spin

The control of electron spin is performed by transverse AC magnetic field in microwave range.

$$\mathcal{H}_{\mathrm{MW}} = \sqrt{2}\Omega_{\mathrm{MW}} \cos(\omega_{\mathrm{MW}} t + \phi_{\mathrm{MW}}(t)) S_x$$

where Ω_{MW} is MW Rabi Amplitude which can be time-series value. ω_{MW} is set to resonance frequency of NV electron spin. Here, we implement π-pulse via MW operation.

RF for Nuclear Spin

It is possible to make a coherent drive between nuclear spin states in radio-frequency range of AC magnetic field.

$$\mathcal{H}_{\circ} = 2\Omega_{\circ} \cos(\omega_{\circ} t + \phi_{\circ}(t)) I_x$$

where $\Omega_{\circ}$ is RF Rabi Amplitude which can be time series value. The driving field phase can also be

Machine Learning Approach

Solving Time-Dependent Schrödinger Equation

Goal: Obtain a GHZ state after time evolution by selecting optimal parameters

Full Hamiltonian (with CPMG-based MW and RF)

(Electron spin in rotating frame, others are in lab frame)

$$\mathcal{H} = \underbrace{\gamma_c B_z I_z^i + A_{||}^i S_z I_z^i + A_\perp S_z I_x^i}_{} + \underbrace{\frac{1}{\sqrt{2}} \Omega_{\mathrm{MW}} S_x}_{} + \underbrace{\Omega_{\circ}(t) \cos(\omega^i_{\circ} t + \phi^i_{\circ}(t))}_{}$$

Cost Function

$$f = |\langle GHZ | U(t, t_0) | \psi_0 \rangle|^2 = |\langle GHZ || \psi_f \rangle|^2$$

Define cost function as the fidelity of the final state to the GHZ state.

$U(t,t_0) = \mathcal{T}\left[\exp \left(-i \int_{t_0}^t dt^{\prime} \mathcal{H}(t^{\prime}) \right)

Why Machine Learning Approach Fails

Issues:

1. GRAPE algorithm unavailable: The GRAPE (Gradient Ascent Pulse Engineering) algorithm, which finds time-ordered sequences of parameter values, is impractical for DDrf systems. The fundamental mismatch lies in the timescales—DDrf sequences for implementing gates like CRX operate on microsecond to millisecond timescales, while the control interval for optimization is on the order of nanoseconds. This creates an enormous optimization space with an impractical number of parameters to optimize.

2. Unknown optimal time: Even when employing known DDrf pulse sequences, determining the optimal total evolution time remains unclear. Since the number of π-pulses must be discrete (one cannot apply a fractional number of pulses), the total evolution time must be selected explicitly without clear analytical guidance, adding another layer of complexity to the optimization problem.

3. Gradient descent cannot consider realistic conditions: Gradient descent methods are designed to

Issue 3: Realistic Constraints

RF Amplitude ($\Omega_\circ$) Constraints

- **Theoretical:** Large amplitude causes unwanted detuning to other nuclear spins
- **Experimental:** Large amplitude can burn the wire
- **Problem:** Enforcing this condition in learning is not straightforward

Attempted Solutions

Qutip's solver can be integrated with the scipy solver. What about adding bounds?

Q: Why not use a constrained (bounded) solver?

A: I tried the L-BFGS-B optimizer, but it failed, took too long, and ultimately did not converge.

Q: What about using a regularization method?

$$f_r(\Omega_\circ) = \Omega_\circ^2, \quad S = f + \lambda f_r$$

Strategy

New Approach:

1. **Analytical analysis first** (Jiwon contributed significantly!)
    - Find good parameters beforehand
    - Use ML for fine-tuning only
2. **Better optimizer needed**
    - Use PyTorch with the Adam optimizer
    - Better convergence (personally experienced with GRAPE algorithm)
3. **New Schrödinger Equation Solver**
    - Implemented in PyTorch
    - Enables gradient-based optimization with better convergence

[illegible]

Solving Differential Equations

Time-Dependent Schrödinger Equation

Dedicated ~3 months to studying NV system control and implementing a new solver

Naive Approach (Euler Method)

Can we do something like this:

$$|\psi(t+dt)\rangle = |\psi(t)\rangle - iH(t)dt$$

(General idea of the Euler method)

Why Euler Method Fails

Norm Preserving Issue

Error proportional to $O(dt^2)$

Solution: Runge-Kutta 7th Order

Runge-Kutta Method

RK4 Example (4th Order)

The Runge-Kutta method improves upon Euler by evaluating the derivative at multiple intermediate points:

$$\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\
k_4 &= f(t_n + h, y_n + hk_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

For the Schrödinger equation $\frac{d}{dt}|\psi\rangle = -iH(t)|\psi\rangle$, we have $f(t, \psi) = -iH(t)|\psi\rangle$

RK7: Uses a 7th order method for higher accuracy with error $O(h^8)$.

Runge-Kutta Method

RK7 Example (4th Order)

Key Features of Implementation

1. Interpolation

We generally runing 100 micro second simulation upto few milisecond simulation.

Given π pulse time, about few nano second, and its pulse shape(Gaussian), rapidly changing during π-pulse operation, time step must be much shorter than π-pulse duration. If we set `dt` in that way and save every vector or observable expectation value results, the amount of required memory would be large, considering potential larger simulation. So, I avoid this problem, I tried to take middle step during calculation between `dt`, larger compared to π-pulse time.

```
def _interpolate_step(self, t: float) -> torch.Tensor:
    """Interpolate solution at time t"""
    if not self.interpolate:
        return self._y_front.clone()

    dt = self._dt_int
    tau = (t - self._t_prev) / dt
```

Key Features of Implementation

2. Norm Preserving Check

To ensure the norm of state vector to stay at 1, I checked this every (interpolated) time step.

```
def _error(self, dt: float) -> float:
   ...
      # Compute error vector: dt * sum(e[i] * k[i])
      self._y_temp.zero_()
      self._accumulate(self._y_temp, self.e, dt, self.rk_step)

      self._norm_front = torch.norm(self._y_front).item()
      error_norm = torch.norm(self._y_temp).item()

      tol = self.atol + max(self._norm_prev, self._norm_front) * self.rtol

      return error_norm / tol
```

If error is not tolerable, the solver will use smaller `dt`.

Key Features of Implementation

3. Adaptive Time Step (Under development and Testing)

Checking how rapidly the Hamiltonian changes, determine interpolated time step adaptively. RF amplitude changes relatively slower than MW amplitude. That is, during π-pulse time, the solver use smaller interpolation step, otherwise, relatively larger interpolation step is acceptable.

```python
def _adaptive_dt_safe(self, t: float) -> float:
    """Alternative implementation: Look ahead to next event(Hard coded)"""
    if self.event_period is None:
        return self._base_dt_safe

    next_event = self._get_next_event_time(t)
    if next_event is None:
        return self._base_dt_safe

    time_to_event = next_event - t

    # If next event is very close, use very small steps
    if time_to_event < self.event_tolerance:
        # Ensure we don't overshoot the event
        max_step_to_event = time_to_event * 0.1  # 10% of remaining time
        adaptive_step = min(self._base_dt_safe * self.event_step_factor, max_step_to_event)
```

Dynamic Decoupling

Original Purpose: Extending Coherence Time

Can be used for many purposes, including implementing conditional gates

Note: Nuclear spin precession depends on the NV electron spin state

Hamiltonian(nuclear spin case)

$$\mathcal{H} = \underbrace{\gamma_c B_z I_z^i + A_{||}^i S_z I_z^i + A_\perp S_z I_x^i}_{} + \underbrace{\frac{1}{\sqrt{2}} \Omega_{\mathrm{MW}} S_x}_{}$$

$$\rightarrow \mathcal{H} = |0\rangle\langle 0| \otimes \omega_0 I_z + |-1\rangle\langle -1| \otimes \left(\omega_L I_z - A_{||} I_z - A_\perp I_x\right)$$
$$\rightarrow \mathcal{H} = |0\rangle\langle 0| \otimes H_0 + |-1\rangle\langle -1| \otimes H_1$$

That is, nuclear spin evolves by $e^{-iH_{0(1)}t}$ when electron spin lies on spin state $|0\rangle(|1\rangle)$.

CPMG Sequence

Sequence Structure

CPMG is a repetition of $(\tau - \pi - 2\tau - \pi - \tau)$ for $N/2$ times

Unitary Operations

- Initial electron spin state 0(1): $e^{-i\phi \hat{I} \cdot \hat{\sigma^i}}$
    - For each cell, if NV spin is 0-state: $e^{-i\phi \hat{I} \cdot \hat{\sigma^0}} = e^{-iH_0\tau} e^{-iH_1 2\tau} e^{-iH_0\tau}$
    - For each cell, if NV spin is 1-state: $e^{-i\phi \hat{I} \cdot \hat{\sigma^1}} = e^{-iH_1\tau} e^{-iH_0 2\tau} e^{-iH_1\tau}$
- When vectors σ^0 and σ^1 are anti-parallel $\rightarrow$ conditional gate
- Directions are generally parallel, but at a specific τ they become anti-parallel

Note: ϕ is equal whatever initial state is 0 or 1 in CPMG cell.

Finding τ Analytically

To enable conditional Gate via repeatation of CPMG cell, we should choose τ such that $\hat{\sigma^0} \cdot \hat{\sigma^1} = -1.$

τ can be found analytically under approximation

$$<! - -Analyticalexpressionfor\tau - - >$$

Then, the number of repeatation number of CPMG cell is chosen mediculously, to acheive $N\phi = \frac{\pi}{2}.$

Conditional Operation

At a certain τ, we observe a conditional operation.

Note: Since the assumption is a very short π-pulse time, further study is required.

Conditional Operation

Multi-Qubit System

In a multi-qubit system, one nuclear spin responds to the CPMG sequence

Further analysis is required

Multi-Qubit CPMG

DDrf: RF Pulse Control

RF pulses can be employed to add additional active operations on nuclear spins

Lab Frame Behavior

Acts as an oscillating effect on the precession axis

Conditional Operation (CRX Gate)

With meticulous RF phase updates, we can derive a conditional operation

"For DDRF gates, the interpulse delays do not need to follow the dynamics of the target nuclear spin: Direct spin-state selective radio-frequency driving with tailored phase updating enables a conditional rotation of the nuclear spin."

DDrf Simulation: Single Nuclear Spin

DDrf Single Qubit

Multi-Qubit Simulation

Multi-qubit simulation results are shown

Note: Further study is required to choose good parameters

TODO

1. **Analytical study:** Find better parameters
2. **Polish code:**
    - Memory management
    - Find better hyperparameters for faster simulation
    - Reduce simulation time to ~1min (for learning purposes)
3. **Run learning** after obtaining good parameters
4. **(Later) Develop density matrix solver in PyTorch**
    - NV/nuclear spins are not fully initialized to pure states
    - Need to consider mixed states $\rightarrow$ density matrix approach