**Lưu ý:**

1. **Cài đặt thư viện:** Đảm bảo bạn đã cài đặt các thư viện cần thiết:

pip install PyQt5 pyqtgraph python-can pyserial cantools # cantools là tùy chọn nâng cao

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Quyền truy cập (Linux):** Trên Linux, bạn có thể cần thêm người dùng vào nhóm dialout (cho SLCAN) hoặc cấu hình SocketCAN đúng cách (ví dụ: sudo ip link set can0 type can bitrate 500000 và sudo ip link set up can0).
2. **Quyền truy cập (Windows):** Đảm bảo driver cho adapter CAN-to-USB (như CANable ở chế độ VCP) đã được cài đặt và bạn biết cổng COM tương ứng.
3. **DBC:** Chức năng giải mã DBC cần có file .dbc và thư viện cantools. Mã ví dụ này chưa tích hợp sâu DBC nhưng có chỗ để bạn thêm vào.
4. **Đồ thị:** Phần đồ thị hiện tại rất cơ bản, chỉ vẽ dữ liệu byte đầu tiên của một ID cụ thể. Bạn cần tùy chỉnh để phân tích và vẽ các tín hiệu cụ thể (đặc biệt khi dùng DBC).

**Cấu trúc file:**

* main\_can\_analyzer.py: File mã nguồn chính của ứng dụng.
* (Tùy chọn) can\_settings\_dialog.ui: File thiết kế giao diện cho hộp thoại cài đặt (tạo bằng Qt Designer).
* (Tùy chọn) can\_settings\_dialog.py: File Python được tạo từ file .ui bằng pyuic5.

**main\_can\_analyzer.py (Mã nguồn chính)**

**Giải thích các phần chính:**

1. **CanWorker(QObject):**
   * Sử dụng QObject thay vì threading.Thread để tích hợp tốt hơn với cơ chế tín hiệu/slot của Qt và quản lý thread bằng QThread.
   * message\_received và error\_occurred là các pyqtSignal để giao tiếp an toàn từ thread worker về thread GUI chính.
   * run(): Khởi tạo bus CAN bằng can.interface.Bus(\*\*self.interface\_config). Sử dụng can.Notifier để nhận message không đồng bộ thay vì bus.recv() trong vòng lặp. Điều này hiệu quả hơn và phản ứng nhanh hơn.
   * \_message\_listener(): Callback được Notifier gọi khi có message mới, nó sẽ phát tín hiệu message\_received.
   * stop(): Đặt cờ \_is\_running thành False và dừng Notifier. Việc dừng bus sẽ diễn ra trong khối finally của run.
   * send\_message(): Được gọi thông qua tín hiệu send\_request từ thread chính để đảm bảo việc gửi diễn ra trên cùng bus và thread đã tạo ra nó.
2. **SettingsDialog(QDialog):**
   * Một hộp thoại đơn giản để người dùng chọn interface, channel (COM port hoặc tên device) và bitrate.
   * Trả về một dictionary chứa các cài đặt đã chọn.
3. **MainWindow(QMainWindow):**
   * **\_\_init\_\_**: Khởi tạo giao diện (\_init\_ui), kết nối tín hiệu (\_connect\_signals), khởi tạo các thuộc tính (trạng thái kết nối, logging, plot data), và tạo QTimer để cập nhật plot định kỳ.
   * **\_init\_ui**: Tạo menu, bảng (QTableWidget), khu vực gửi (QGroupBox), khu vực vẽ đồ thị (pg.PlotWidget), và thanh trạng thái (QStatusBar). Cấu hình layout và các widget.
   * **\_connect\_signals**: Kết nối các action trong menu (Connect, Disconnect, Configure, Start/Stop Log, Exit, About) và nút Send tới các phương thức xử lý tương ứng.
   * **connect\_can**:
     + Tạo CanWorker và QThread mới.
     + Di chuyển worker vào thread.
     + Kết nối các tín hiệu từ worker (message, error) tới các slot trong MainWindow.
     + Kết nối tín hiệu started của thread tới worker.run.
     + Kết nối tín hiệu finished của thread tới on\_thread\_finished để dọn dẹp.
     + Kết nối tín hiệu send\_request từ MainWindow tới worker.send\_message.
     + Bắt đầu thread.
     + Cập nhật trạng thái GUI và bật ui\_update\_timer.
   * **disconnect\_can**:
     + Gọi worker.stop() để báo hiệu dừng.
     + Gọi thread.quit() và thread.wait() để dừng thread một cách an toàn.
     + Cập nhật trạng thái GUI. Dừng ui\_update\_timer.
   * **on\_thread\_finished**: Slot được gọi khi QThread thực sự kết thúc. Dọn dẹp tài nguyên worker và thread, cập nhật trạng thái cuối cùng.
   * **update\_connection\_status**: Bật/tắt các nút và action dựa trên self.is\_connected.
   * **configure\_settings**: Hiển thị SettingsDialog và cập nhật self.can\_settings.
   * **handle\_message**: Slot được kết nối với CanWorker.message\_received.
     + Định dạng dữ liệu từ can.Message.
     + Thêm một dòng mới vào QTableWidget.
     + Ghi vào file CSV nếu is\_logging là True.
     + Thu thập dữ liệu cho đồ thị (chỉ lưu trữ, không vẽ ngay).
   * **handle\_can\_error**: Slot được kết nối với CanWorker.error\_occurred. Hiển thị lỗi trên status bar.
   * **prepare\_send\_message**: Lấy dữ liệu từ các ô nhập liệu, tạo can.Message, và phát tín hiệu send\_request để CanWorker gửi đi.
   * **start\_logging, stop\_logging**: Xử lý việc mở/đóng file CSV và bật/tắt cờ is\_logging.
   * **update\_plots**: Slot được gọi bởi QTimer. Lấy dữ liệu đã thu thập trong handle\_message và cập nhật các đường cong trên pyqtgraph.PlotWidget.
   * **closeEvent**: Đảm bảo mọi thứ được dừng và đóng đúng cách khi người dùng đóng cửa sổ.

**Cách chạy:**

1. Lưu mã trên thành file main\_can\_analyzer.py.
2. Chạy từ terminal: python main\_can\_analyzer.py
3. Vào menu **Settings -> Configure CAN...** để đặt đúng loại Interface (slcan/socketcan), Channel (COMx/can0), và Baudrate.
4. Nhấn **Settings -> Connect**.
5. Quan sát dữ liệu CAN đến trong bảng và đồ thị (nếu ID khớp).
6. Nhập ID, Data và nhấn **Send** để gửi tin nhắn.
7. Sử dụng **File -> Start/Stop Logging** để ghi dữ liệu vào file CSV.

**Hướng phát triển tiếp theo:**

1. **Lọc dữ liệu:** Thêm các ô nhập liệu hoặc dropdown để lọc bảng theo ID, Type.
2. **Giải mã DBC:** Tích hợp thư viện cantools để load file DBC. Khi nhận message, tìm message tương ứng trong DBC, giải mã các tín hiệu và hiển thị tên/giá trị tín hiệu thay vì chỉ dữ liệu hex thô. Cập nhật đồ thị để vẽ các tín hiệu đã giải mã.
3. **Gửi định kỳ:** Triển khai logic sử dụng QTimer để gửi message lặp lại dựa trên giá trị send\_rate\_spin.
4. **Highlighting:** Thay đổi màu nền của các dòng trong bảng dựa trên ID hoặc các tiêu chí khác.
5. **Cải thiện đồ thị:** Cho phép chọn nhiều tín hiệu/ID để vẽ, tùy chỉnh màu sắc, trục, thêm chức năng zoom/pan tốt hơn.
6. **Ghi/Phát lại (Replay):** Thêm chức năng đọc file log (CSV, ASC) và gửi lại các message lên bus CAN theo đúng timing gốc.
7. **Hỗ trợ nhiều bus:** Chỉnh sửa cấu trúc để quản lý nhiều kết nối CAN đồng thời (ví dụ: sử dụng dictionary để lưu trữ worker/thread cho từng bus).
8. **Xuất dữ liệu:** Thêm tùy chọn xuất sang các định dạng khác như Vector ASC, MDF.
9. **Đóng gói:** Sử dụng PyInstaller hoặc cx\_Freeze để tạo file thực thi độc lập cho Windows/Linux.