Báo cáo Nghiên cứu Kỹ thuật: Kiến trúc Tích hợp Ruckig OTG và Robotics Library (RL) cho Bộ Điều khiển Robot Hàn 6-DOF

Tóm tắt Điều hành

Báo cáo này trình bày một phân tích toàn diện và chiến lược thực thi kỹ thuật nhằm tích hợp thư viện tạo quỹ đạo trực tuyến (Online Trajectory Generation - OTG) Ruckig (Phiên bản Cộng đồng) vào kiến trúc điều khiển robot hàn 6 bậc tự do (6-DOF). Mục tiêu cốt lõi là giải quyết các yêu cầu về chuyển động nội suy tuyến tính (MOVL), đảm bảo độ mượt mà của chuyển động thông qua giới hạn đạo hàm bậc ba (Jerk/S-Curve), và khả năng phản hồi thời gian thực (Real-time) đối với các tín hiệu cảm biến cho ứng dụng dò đường hàn (Seam Tracking).

Dựa trên nền tảng toán học cốt lõi (Core Math) từ Robotics Library (RL) để giải bài toán Động học Nghịch (Inverse Kinematics - IK), báo cáo đề xuất một kiến trúc luồng dữ liệu (Data Flow) tối ưu, trong đó Ruckig đóng vai trò là bộ nội suy Cartesian thời gian thực. Phân tích chỉ ra rằng để đạt được chuyển động thẳng (Straight Line) chính xác trong không gian 3D, việc sử dụng chế độ "Đồng bộ hóa Pha" (Phase Synchronization) của Ruckig là bắt buộc. Đồng thời, để vượt qua hạn chế về tính toán điểm trung gian (intermediate waypoints) trong phiên bản Cộng đồng, báo cáo xây dựng một cơ chế cập nhật trạng thái mục tiêu liên tục ("Arbitrary Target States") để mô phỏng hành vi Seam Tracking với độ trễ tối thiểu.

---

1. Cơ sở Lý thuyết và Đặt vấn đề trong Điều khiển Robot Hàn

1.1. Tầm quan trọng của Đạo hàm bậc ba (Jerk) trong Công nghệ Hàn Hồ quang

Trong lĩnh vực hàn hồ quang bằng robot (GMAW/TIG), chất lượng mối hàn không chỉ phụ thuộc vào các thông số nguồn hàn (dòng điện, điện áp) mà còn chịu ảnh hưởng trực tiếp bởi độ ổn định của chuyển động mỏ hàn. Các bộ điều khiển truyền thống thường sử dụng biên dạng vận tốc hình thang (Trapezoidal Velocity Profile), chỉ giới hạn vận tốc và gia tốc (đạo hàm bậc hai). Mặc dù đơn giản về mặt tính toán, phương pháp này cho phép gia tốc thay đổi tức thời (step change), dẫn đến giá trị Jerk (đạo hàm bậc ba của vị trí) tiến tới vô cùng tại các điểm chuyển tiếp.

Đối với cơ cấu cơ khí của robot 6 trục, Jerk cao gây ra các rung động tần số cao tại khâu tác động cuối (End-Effector/TCP). Trong quá trình hàn, sự rung động này dẫn đến:

1. **Biến động vũng hàn (Weld Pool Instability):** Kim loại lỏng bị dao động, gây ra các khuyết tật như cháy chân (undercut) hoặc vân hàn không đều.
2. **Sai lệch quỹ đạo tức thời:** Ngay cả khi robot đi đúng đường danh định, rung động có thể làm điểm tiếp xúc dây hàn (Wire Stick-out) dao động, thay đổi dòng hàn thực tế.
3. **Hao mòn cơ khí:** Các hộp số giảm tốc (như Harmonic Drive hay Cycloidal) chịu ứng suất lớn, làm giảm tuổi thọ robot.

Ruckig giải quyết vấn đề này bằng cách tạo ra các quỹ đạo giới hạn Jerk (Jerk-limited), hay còn gọi là biên dạng S-Curve.[1] Bằng cách kiểm soát J_{max}, gia tốc thay đổi tuyến tính thay vì nhảy bậc, triệt tiêu rung động và đảm bảo mỏ hàn di chuyển "mượt" như chất lỏng, yếu tố tiên quyết cho mối hàn chất lượng cao.

1.2. Sự khác biệt giữa Offline Planning và Online Trajectory Generation (OTG)

Hiểu rõ sự khác biệt này là chìa khóa để thiết kế kiến trúc điều khiển đúng đắn.

- **Offline Planning (Lập kế hoạch ngoại tuyến):** Các thư viện như OMPL (trong MoveIt) tính toán toàn bộ quỹ đạo từ điểm A đến B trước khi robot bắt đầu di chuyển. Phương pháp này tối ưu cho việc tránh vật cản trong môi trường tĩnh nhưng hoàn toàn bất lực trước các thay đổi động.
- **Online Trajectory Generation (OTG - Ruckig):** Ruckig tính toán trạng thái tiếp theo của robot ngay trong chu kỳ điều khiển (ví dụ: mỗi 1ms). Nó nhận đầu vào là trạng thái hiện tại (P_0, V_0, A_0) và trạng thái mục tiêu (P_t, V_t, A_t) để đưa ra trạng thái kế tiếp ($P_{next}, V_{next}, A_{next}$) tuân thủ các giới hạn động học.[2]

Thách thức với Ruckig Community Version: Phiên bản Cộng đồng của Ruckig có một giới hạn quan trọng: nó không hỗ trợ tính toán thời gian thực cho danh sách các điểm trung gian (Waypoints).[4] Nếu người dùng nạp một danh sách điểm, nó sẽ chuyển sang tính toán offline hoặc API đám mây, không phù hợp cho vòng điều khiển 1ms. Do đó, kiến trúc đề xuất phải sử dụng cơ chế "State-to-State" (từ Trạng thái đến Trạng thái) và cập nhật mục tiêu liên tục để xử lý các đường hàn phức tạp hoặc tracking.

---

2. Kiến trúc Tích hợp và Luồng Dữ liệu (Data Flow) Tối ưu

Để tích hợp Ruckig (xử lý quỹ đạo) với Robotics Library (xử lý động học) trong một hệ thống thời gian thực, chúng ta cần xác định không gian làm việc của OTG.

2.1. Quyết định Chiến lược: Cartesian OTG hay Joint Space OTG?

Đây là quyết định quan trọng nhất ảnh hưởng đến khả năng thực hiện lệnh MOVL (đi thẳng).

Đặc điểm	Joint Space OTG (Nội suy Khớp)	Cartesian Space OTG (Nội suy Đề các)
Đầu vào Ruckig	Góc khớp ($q_1, \cdots, q_6$)	Tọa độ TCP ($x, y, z, \alpha, \beta, \gamma$)
Đường đi TCP	Đường cong (vòng cung) trong không gian	Đường thẳng (nếu dùng đúng sync)
Singularity	An toàn, không bao giờ gặp điểm kỳ dị	Rủi ro cao khi đi qua điểm kỳ dị
Ứng dụng Hàn	Dùng cho lệnh MOVJ (di chuyển nhanh không hàn)	**Bắt buộc cho lệnh MOVL (đường hàn thẳng)**

Kết luận: Đối với ứng dụng hàn yêu cầu MOVL, **Ruckig phải được cấu hình chạy trong không gian Cartesian**. Điều này có nghĩa là đầu ra của Ruckig là vị trí Cartesian tiếp theo, và giá trị này sẽ được đưa vào bộ giải IK của Robotics Library để tính toán góc khớp.

2.2. Thiết kế Luồng Dữ liệu Thời gian thực (Hard Real-time Loop)

Kiến trúc được đề xuất hoạt động trên một chu kỳ điều khiển cứng (ví dụ: 1kHz / 1ms). Dưới đây là luồng dữ liệu chi tiết từng bước:

Tầng 1: Thu thập và Tiền xử lý (Start of Cycle)

1. **Đọc Feedback:** Nhận vị trí góc khớp hiện tại (q_{actual}) từ Servo Driver qua EtherCAT.
2. **Forward Kinematics (FK - Robotics Library):** Sử dụng RL để tính toán vị trí Cartesian hiện tại (X_{actual}) từ q_{actual} .
    - *Lưu ý:* Mặc dù Ruckig có thể tự lưu trữ trạng thái nội bộ, việc sử dụng X_{actual} từ feedback giúp hệ thống bù trừ sai số cơ khí, nhưng có thể gây nhiễu nếu tín hiệu encoder không sạch. Một phương pháp lai (Hybrid) thường được dùng: Sử dụng V_{actual} và A_{actual} từ bộ quan sát (Observer), nhưng dùng P_{target_prev} làm đầu vào vị trí để tránh rung động do lượng tử hóa encoder.

Tầng 2: Tạo Quỹ đạo (Ruckig OTG)

3. **Cập nhật Mục tiêu (Application Layer):** Nếu đang thực hiện Seam Tracking, cập nhật Target Position mới dựa trên dữ liệu cảm biến (chi tiết ở Chương 4).
4. **Thực thi Ruckig Update:**
   - **Input:** Trạng thái hiện tại ($X_{curr}, V_{curr}, A_{curr}$), Trạng thái mục tiêu (X_{target}, V_{target}), Giới hạn ($V_{max}, A_{max}, J_{max}$).
   - **Function:** ruckig.update(input, output).
   - **Output:** Trạng thái Cartesian tiếp theo cho chu kỳ tới ($X_{next}, V_{next}, A_{next}$).

Tầng 3: Giải Động học Nghịch (RL Core Math)

5. **Inverse Kinematics (IK - Robotics Library):**
   - Sử dụng bộ giải IK (thường là rl::mdl::NloptInverseKinematics hoặc thuật toán Newton-Raphson lặp cho tốc độ cao).
   - **Input:** X_{next} (Tọa độ mong muốn từ Ruckig).
   - **Seed:** q_{actual} (Góc khớp hiện tại làm điểm khởi đầu để đảm bảo IK hội tụ về cấu hình gần nhất).
   - **Output:** q_{cmd} (Góc khớp lệnh).

Tầng 4: Hậu xử lý và An toàn

6. **Kiểm tra Giới hạn:** So sánh q_{cmd} với giới hạn mềm của khớp. Tính toán vận tốc khớp yêu cầu $\dot{q} = (q_{cmd} - q_{actual})/\Delta t$. Nếu $\dot{q}$ vượt quá giới hạn vật lý (do đi qua điểm kỳ dị), kích hoạt dừng khẩn cấp hoặc chế độ giảm tốc (Scaling).
7. **Gửi Lệnh:** Gửi q_{cmd} xuống Servo Driver.
8. **Lưu Trạng thái:** Sao chép output của Ruckig vào input cho chu kỳ tiếp theo (output.pass_to_input(input)).

---

3. Chuyển động MOVL và Đồng bộ hóa (Synchronization)

Yêu cầu "MOVL" (Move Linear) trong robot hàn đòi hỏi đầu công tác di chuyển theo một đường thẳng tuyệt đối trong không gian 3D, đồng thời duy trì hướng mỏ hàn thay đổi mượt mà. Trong Ruckig, việc này phụ thuộc hoàn toàn vào cơ chế **Đồng bộ hóa (Synchronization)**.

3.1. Phân tích các Chế độ Đồng bộ hóa

Ruckig Community cung cấp các chế độ đồng bộ hóa sau [5]:

Chế độ (Enum)	Cơ chế hoạt động	Kết quả Quỹ đạo (Cartesian)
Synchronization::None	Mỗi trục (X, Y, Z...) tính toán thời gian tối ưu riêng biệt. Trục nào xong trước sẽ dừng lại đợi.	**Đường gấp khúc/Cong.** TCP sẽ di chuyển ziczac hoặc theo đường cong ngẫu nhiên. **Không dùng cho hàn.**
Synchronization::Time	Tính toán thời gian của trục chậm nhất (T_{max}), sau đó kéo dài thời gian của các trục còn lại để cùng kết thúc tại T_{max}.	**Đường cong.** Mặc dù bắt đầu và kết thúc cùng lúc, nhưng biên dạng vận tốc khác nhau dẫn đến quỹ đạo không thẳng.
Synchronization::Phase	Không chỉ đồng bộ thời gian đích, mà còn đồng bộ **pha chuyển động**. Các trục được tỷ lệ hóa (scaled) sao cho tại mọi thời điểm t, tỷ lệ quãng đường đi được là hằng số.	**Đường thẳng (Straight Line).** Đây là chìa khóa của MOVL. Đảm bảo vector vận tốc luôn hướng về đích.

Khuyến nghị Kỹ thuật: Để thực hiện MOVL, bạn **bắt buộc** phải cấu hình Ruckig sử dụng Synchronization::Phase.

C++

```
// Cấu hình bắt buộc cho MOVL
input.synchronization = ruckig::Synchronization::Phase;
```

3.2. Tại sao "Phase Synchronization" tạo ra đường thẳng?

Về mặt toán học, nếu ta có chuyển động từ P_0 đến P_t. Một điểm $P(t)$ nằm trên đoạn

thẳng nối hai điểm này nếu và chỉ nếu: $P(t) = P_0 + \lambda(t) \cdot (P_t - P_0)$ Trong đó $\lambda(t)$ là một hàm vô hướng đi từ 0 đến 1. Chế độ Phase của Ruckig đảm bảo rằng tất cả các bậc tự do (X, Y, Z) đều tuân theo cùng một biên dạng thời gian $\lambda(t)$ (profile 1-DOF được tính cho trục hạn chế nhất). Do đó, phương trình đường thẳng được thỏa mãn.[4]

3.3. Xử lý Chuyển động Xoay (Orientation)

Với robot 6-DOF, ngoài X, Y, Z, ta còn phải nội suy hướng (Roll, Pitch, Yaw).

- Ruckig xử lý hướng như các biến số thực (double).
- **Vấn đề:** Euler Angles có thể gặp vấn đề về tính không liên tục (nhảy từ 179 độ sang -179 độ).
- **Giải pháp:** Trong ứng dụng hàn (thường mỏ hàn chúi xuống), việc sử dụng Euler Angles (ZYX hoặc XYZ) thường đủ an toàn và trực quan. Tuy nhiên, cần một bước "Unwind" (tháo gỡ) góc trước khi đưa vào Ruckig. Ví dụ: Nếu góc hiện tại là 10 độ và đích là 350 độ, Ruckig sẽ hiểu là đi quãng đường +340 độ. Ta cần chuyển đổi đích thành -10 độ để quãng đường là -20 độ.
- **Đồng bộ hóa Xoay:** Khi dùng Phase cho cả 6 trục, việc xoay mỏ hàn cũng sẽ được phân bố đều dọc theo chiều dài đường hàn. Điều này rất lý tưởng cho kỹ thuật hàn leo hoặc hàn góc, nơi góc mỏ hàn cần thay đổi dần dần từ đầu đến cuối đường hàn.

---

4. Giải pháp Seam Tracking với "Arbitrary Target States"

Yêu cầu tích hợp Seam Tracking là thách thức lớn nhất khi dùng phiên bản Community vì thiếu Interface Tracking chuyên dụng.[4] Tuy nhiên, bản chất của Ruckig là thuật toán "State-to-State" cực nhanh, cho phép ta cập nhật đích đến (Target) trong mỗi chu kỳ điều khiển.

4.1. Nguyên lý "Moving Target" (Mục tiêu Di động)

Thay vì coi đường hàn là một tập hợp các điểm cố định, hãy coi nó là một mục tiêu đang di chuyển.

Trong mỗi chu kỳ 1ms:

1. **Đọc Cảm biến:** Cảm biến Laser Vision trả về độ lệch ngang (Δy) và độ lệch cao (Δz) so với khung tọa độ mỏ hàn.
2. **Biến đổi Tọa độ:** Chuyển đổi vector lệch này từ khung cảm biến (Sensor Frame) sang khung thế giới (World Frame).
   $v_{offset\_world} = R_{robot} \cdot v_{sensor}$

3. **Cộng dồn (Integration):** Không chỉ cộng độ lệch vào vị trí hiện tại, ta phải cộng nó vào **Vị trí Mục tiêu (Target Position)**.
   $P_{target\_new} = P_{target\_nominal} + v_{offset\_world}$
4. **Cập nhật Ruckig:** Gán giá trị mới này vào input.target_position và gọi ruckig.update().

Ruckig sẽ tính toán lại quỹ đạo từ trạng thái hiện tại (đang di chuyển với vận tốc V) để đến mục tiêu mới. Do Ruckig hỗ trợ vận tốc đầu khác 0, chuyển động chuyển tiếp sẽ rất mượt mà, không bị giật cục.[2]

4.2. Xử lý "Arbitrary Target States" (Trạng thái Mục tiêu Bất kỳ)

Thuật ngữ "Arbitrary Target States" [2] trong tài liệu Ruckig ám chỉ khả năng thay đổi đích đến bất kỳ lúc nào. Điều này cực kỳ quan trọng cho Seam Tracking vì:

- Nếu cảm biến phát hiện đường hàn cong đột ngột, đích đến thay đổi lớn.
- Ruckig sẽ tự động điều chỉnh gia tốc (trong giới hạn Jerk) để lái robot về đích mới nhanh nhất có thể.

Lưu ý quan trọng về vận tốc Tracking:

Khi tracking, robot không bao giờ thực sự "đến" đích vì đích luôn bị đẩy ra xa (dọc theo đường hàn). Để duy trì tốc độ hàn (Travel Speed) ổn định, input.target_velocity cần được thiết lập.

- Nếu tracking điểm cố định: $V_{target} = 0$.
- Nếu tracking đường hàn đang chạy: V_{target} nên được đặt là vector vận tốc hàn danh định. Điều này giúp Ruckig dự đoán chuyển động (Feedforward), giảm thiểu sai số bám (Lag error).

4.3. Bộ lọc và Ổn định (Signal Smoothing)

Dữ liệu từ cảm biến laser thường có nhiễu (noise). Nếu đưa trực tiếp nhiễu này vào target_position của Ruckig, robot sẽ rung lắc vì Ruckig sẽ cố gắng phản ứng với từng gai nhiễu đó (do tính chất "Real-time reaction" [2]).

Giải pháp: Cần một bộ lọc trung gian.

1. **Bộ đệm vòng (Ring Buffer):** Lấy trung bình cộng 10-20 mẫu dữ liệu cảm biến gần nhất.
2. **Giới hạn biên độ (Saturation):** Chỉ cho phép sửa đổi mục tiêu tối đa X mm mỗi chu kỳ để tránh các cú nhảy bất thường do lỗi đọc cảm biến.

---

5. Hiện thực hóa bằng C++ và Robotics Library (RL)

Dưới đây là mẫu code C++ minh họa kiến trúc tích hợp. Code giả định bạn đã có môi trường RL và Ruckig.

5.1. Cấu trúc dữ liệu và Khởi tạo

C++

```cpp
#include <iostream>
#include <vector>
#include <array>
#include <chrono>

// Ruckig Include
#include <ruckig/ruckig.hpp>

// Robotics Library Includes (Giả định các header cơ bản)
#include <rl/mdl/Kinematic.h>
#include <rl/mdl/Model.h>
#include <rl/math/Transform.h>
#include <rl/math/Vector.h>

using namespace ruckig;

// Định nghĩa số bậc tự do (6 trục)
const size_t DOFs = 6;

int main() {
    // 1. KHỞI TẠO RUCKIG
    // Chu kỳ điều khiển: 1ms (0.001s)
    Ruckig<DOFs> otg(0.001);

    InputParameter<DOFs> input;
    OutputParameter<DOFs> output;

    // 2. CẤU HÌNH GIỚI HẠN (Motion Constraints)
    // Đơn vị: m, m/s, m/s^2, m/s^3 (cho tịnh tiến) và rad, rad/s... (cho xoay)
    // Cần tuning các giá trị này dựa trên tải trọng robot thực tế
    input.max_velocity = {1.0, 1.0, 1.0, 1.5, 1.5, 1.5};
    input.max_acceleration = {2.0, 2.0, 2.0, 3.0, 3.0, 3.0};
    input.max_jerk = {10.0, 10.0, 10.0, 20.0, 20.0, 20.0}; // Giá trị quyết định độ mượt (S-Curve)
```

```
    // QUAN TRỌNG: Cấu hình MOVL (Đi thẳng)
    input.synchronization = Synchronization::Phase;

    // 3. KHỞI TẠO ROBOTICS LIBRARY (IK Solver)
    rl::mdl::Kinematic* kinematics = new rl::mdl::Kinematic();
    // kinematics->load("path/to/robot_model.xml"); // Load file URDF/XML của robot

    rl::math::Vector q_current(DOFs);      // Góc khớp hiện tại
    rl::math::Vector q_command(DOFs);      // Góc khớp lệnh
    rl::math::Transform t_current;         // Pose hiện tại (Ma trận 4x4)

    // 4. THIẾT LẬP TRẠNG THÁI BAN ĐẦU
    // Giả sử đọc từ Encoders
    q_current << 0, -1.57, 1.57, 0, 1.57, 0; // Ví dụ
    kinematics->setPosition(q_current);
    kinematics->forwardPosition(); // Tính FK để lấy vị trí Cartesian ban đầu
    t_current = kinematics->getOperationalPosition(0);

    // Map từ RL Transform sang Ruckig Vector (X, Y, Z, Rx, Ry, Rz)
    // Lưu ý: Cần hàm chuyển đổi từ Rotation Matrix sang Euler ZYX
    // std::array<double, 6> start_pose = MatrixToEuler(t_current);

    input.current_position = {0.5, 0.0, 0.5, 0.0, 3.14, 0.0}; // Giả định
    input.current_velocity = {0,0,0,0,0,0};
    input.current_acceleration = {0,0,0,0,0,0};

    // Thiết lập Mục tiêu (Ví dụ: Đi thẳng 100mm theo trục X)
    input.target_position = {0.6, 0.0, 0.5, 0.0, 3.14, 0.0};
    input.target_velocity = {0,0,0,0,0,0};
    input.target_acceleration = {0,0,0,0,0,0};

    // VÒNG LẶP REAL-TIME (Control Loop)
    std::cout << "Bắt đầu hàn..." << std::endl;
    bool welding_active = true;

    while (welding_active) {
        auto start_time = std::chrono::high_resolution_clock::now();

        // --- BƯỚC A: CẬP NHẬT SEAM TRACKING (Arbitrary Target State) ---
        // Giả sử hàm ReadSensor() trả về vector bù [dx, dy, dz, 0, 0, 0]
        // std::array<double, 6> sensor_offset = ReadSensor();

        // Nếu có lệch, cộng dồn vào Target Position HIỆN TẠI
```

```
        // for(int i=0; i<3; ++i) input.target_position[i] += sensor_offset[i];

        // --- BƯỚC B: TÍNH TOÁN QUỸ ĐẠO (Ruckig) ---
        Result result = otg.update(input, output);

        if (result == Result::Working |

| result == Result::Finished) {

            // Lấy vị trí Cartesian tiếp theo từ Ruckig
            std::array<double, DOFs> next_cartesian = output.new_position;

            // --- BƯỚC C: GIẢI IK (Robotics Library) ---

            // 1. Chuyển đổi Ruckig Vector -> RL Transform
            rl::math::Transform t_target;
            // t_target = EulerToMatrix(next_cartesian); // Tự viết hàm helper này

            // 2. Cài đặt trạng thái khớp hiện tại làm seed cho IK
            kinematics->setPosition(q_current);

            // 3. Gọi bộ giải IK
            bool ik_success = kinematics->calculateInversePosition(t_target, q_command);

            if (ik_success) {
                // Gửi q_command xuống Servo Drivers
                // WriteEtherCAT(q_command);

                // Cập nhật q_current cho vòng lặp sau (Simulation hoặc đọc lại từ Encoder)
                q_current = q_command;
            } else {
                // Xử lý lỗi: Không tìm thấy lời giải IK (Ngoài vùng làm việc hoặc Singularity)
                std::cerr << "Lỗi IK! Dừng khẩn cấp." << std::endl;
                break;
            }

            // --- BƯỚC D: CHUẨN BỊ CHO VÒNG SAU ---
            // Quan trọng: Chuyển output thành input cho vòng kế tiếp
            output.pass_to_input(input);

            if (result == Result::Finished) {
                // Đã đến đích (hết đường hàn)
                 welding_active = false;
```

```
        }

    } else {
        std::cerr << "Lỗi Ruckig: " << result << std::endl;
        break;
    }

    // --- BƯỚC E: ĐỒNG BỘ THỜI GIAN ---
    // Đảm bảo vòng lặp chạy đúng 1ms
    // SleepUntil(start_time + 1ms);
  }

  return 0;
}
```

5.2. Giải thích Chi tiết Code

1. **Ruckig<DOFs> otg(0.001);**: Khởi tạo instance Ruckig với chu kỳ 1ms. Tham số này cực kỳ quan trọng để tích phân vị trí chính xác.
2. **input.synchronization = Synchronization::Phase;**: Dòng lệnh quyết định tính chất "thẳng" của đường hàn. Nếu bỏ qua, mỏ hàn sẽ đi đường cong.
3. **Vòng lặp while**: Mô phỏng Thread thời gian thực. Trong hệ thống thực (như Xenomai hay RT-Linux), toàn bộ khối lệnh trong while phải thực thi xong trong <1ms.
   - Ruckig update() thường tốn khoảng 20-50µs.[2]
   - RL calculateInversePosition() (Newton-Raphson) có thể tốn từ 100-300µs tùy độ hội tụ. Cần chú ý monitor thời gian này.
4. **output.pass_to_input(input);**: Đây là cơ chế "State-to-State". Nó copy trạng thái động học (P, V, A) vừa tính được làm điểm khởi đầu cho chu kỳ sau, đảm bảo tính liên tục C2 (gia tốc liên tục) cho đường S-Curve.

---

6. Các vấn đề Kỹ thuật Nâng cao

6.1. Xử lý Điểm Kỳ dị (Singularities)

Khi chạy Ruckig trong không gian Cartesian, rủi ro lớn nhất là quỹ đạo đường thẳng đi ngang qua điểm kỳ dị của robot (ví dụ: cổ tay thẳng hàng với cánh tay).

- **Triệu chứng:** Tại gần điểm kỳ dị, để duy trì vận tốc Cartesian không đổi, vận tốc khớp (V_{joint}) sẽ tăng vọt tới vô cùng.
- **Giải pháp trong Code:**
  Sau bước giải IK, cần tính vận tốc khớp dự kiến:

$\dot{q}_{est} = \frac{q_{command} - q_{current}}{\Delta t}$

Nếu $\max(|\dot{q}_{est}|) > V_{joint\_limit}$, bạn phải can thiệp. Ruckig Community không hỗ trợ ràng buộc vận tốc khớp tự động khi chạy ở chế độ Cartesian. Bạn phải thực hiện "Velocity Scaling": giảm input.target_velocity hoặc tạm thời giảm max_velocity Cartesian và tính lại Ruckig.

6.2. Giới hạn Thời gian quỹ đạo (The 7e3 Limit)

Tài liệu Ruckig đề cập giới hạn thời gian quỹ đạo là 7×10^3 (khoảng 2 giờ).[1]

- **Tác động:** Với các đường hàn thông thường (vài phút), đây không phải vấn đề.
- **Xử lý:** Nếu robot vận hành liên tục (ví dụ: tracking băng chuyền vô tận), biến thời gian nội bộ có thể tràn. Tuy nhiên, Ruckig reset thời gian mỗi khi đạt đích (Result::Finished). Với kỹ thuật "Arbitrary Target States", nếu đích thay đổi liên tục và robot không bao giờ "dừng hẳn", cần theo dõi cẩn thận. Trong thực tế hàn, robot luôn có các điểm dừng (tắt hồ quang), nên bộ đếm sẽ được reset tự nhiên.

6.3. Giới hạn của Robotics Library (RL)

RL là một thư viện mạnh nhưng thiên về tính toán "offline" hoặc tĩnh. Hàm calculateInversePosition sử dụng phương pháp lặp.

- **Tối ưu hóa:** Cần cài đặt số lần lặp tối đa (Max Iterations) và dung sai (Tolerance) phù hợp cho bài toán thời gian thực (ví dụ: 1e-4 mét, 50 iterations). Nếu quá chính xác, IK sẽ tốn quá nhiều thời gian (>1ms), làm vỡ chu kỳ điều khiển (Cycle miss).

---

7. Kết luận

Việc tích hợp Ruckig Community Version vào bộ điều khiển robot hàn 6-DOF sử dụng Robotics Library là hoàn toàn khả thi và mang lại hiệu suất vượt trội so với các bộ lập kế hoạch truyền thống. Chìa khóa thành công nằm ở 3 điểm:

1. **Kiến trúc Cartesian OTG:** Cho phép kiểm soát trực tiếp hình dáng đường hàn (MOVL) và tốc độ hàn.
2. **Phase Synchronization:** Đảm bảo độ thẳng tuyệt đối của đường hàn trong không gian 3D.
3. **Cơ chế Arbitrary Target Updates:** Vượt qua giới hạn không có API Tracking của bản Community, cho phép phản hồi thời gian thực với cảm biến Seam Tracking.

Mô hình này không chỉ đáp ứng yêu cầu về chuyển động mượt mà (S-Curve) giúp nâng cao chất lượng mối hàn mà còn mở ra khả năng thích ứng linh hoạt với sai số phôi thông qua cảm biến, đưa bộ điều khiển robot lên tầm cao mới về công nghệ.

Works cited

1. Background Information - Ruckig, accessed February 1, 2026, https://docs.ruckig.com/background.html
2. Ruckig - Motion Generation for Robots and Machines, accessed February 1, 2026, https://ruckig.com/
3. ruckig - ROS Repository Overview, accessed February 1, 2026, https://index.ros.org/r/ruckig/
4. Tutorial - Ruckig, accessed February 1, 2026, https://docs.ruckig.com/tutorial.html
5. ruckig Namespace Reference, accessed February 1, 2026, https://docs.ruckig.com/namespaceruckig.html
6. pantor/ruckig: Motion Generation for Robots and Machines. Real-time. Jerk-constrained. Time-optimal. - GitHub, accessed February 1, 2026, https://github.com/pantor/ruckig
7. [2105.04830] Jerk-limited Real-time Trajectory Generation with Arbitrary Target States, accessed February 1, 2026, https://arxiv.org/abs/2105.04830