Báo cáo Kỹ thuật Chuyên sâu: Triển khai Giải thuật Động học Nghịch đảo Giải tích (Analytical IK) trên Nền tảng Robotics Library (RL) cho Robot Hàn 6-DOF

1. Tổng quan Điều hành

Báo cáo này được biên soạn nhằm phục vụ nhu cầu phát triển phần mềm điều khiển cốt lõi (Core Math Engine) cho hệ thống robot hàn thương mại 6 bậc tự do (6-DOF). Dựa trên yêu cầu kỹ thuật về việc sử dụng Robotics Library (RL) làm nền tảng toán học và cấu trúc hình học tương đồng với PUMA 560, tài liệu này cung cấp một lộ trình triển khai toàn diện từ thiết kế cơ khí (CAD) đến mã nguồn điều khiển thời gian thực (C++).

Trong ứng dụng hàn hồ quang công nghiệp, độ chính xác của quỹ đạo (path accuracy) và tính ổn định của vận tốc đầu mỏ hàn (TCP velocity) là yếu tố sống còn. Các giải thuật động học nghịch đảo (IK) dựa trên phương pháp số (Numerical Methods) như Newton-Raphson hay Levenberg-Marquardt, mặc dù linh hoạt, thường gặp vấn đề về thời gian hội tụ không xác định và rủi ro rơi vào cực tiểu địa phương (local minima).[1] Ngược lại, giải thuật động học nghịch đảo giải tích (Analytical IK), cụ thể là lớp rl::kin::Puma trong Robotics Library, cung cấp lời giải chính xác, thời gian tính toán tất định và khả năng kiểm soát tường minh các cấu hình đa nghiệm (Arm, Elbow, Wrist).[3]

Tài liệu này sẽ phân tích sâu các khía cạnh:

1. **Chuyển đổi dữ liệu**: Quy trình trích xuất tham số Denavit-Hartenberg (DH) chuẩn từ bản vẽ CAD và ánh xạ vào định dạng XML của RL.
2. **Kiến trúc phần mềm**: Phương pháp "ép" hệ thống sử dụng rl::kin::Puma thay vì các bộ giải số học mặc định.
3. **Xử lý đa nghiệm**: Logic toán học để lọc 8 nghiệm khả dĩ dựa trên các cờ cấu hình và tránh va chạm.
4. **Hiện thực hóa**: Cung cấp mã nguồn C++ mẫu đạt chuẩn công nghiệp.

---

2. Cơ sở Lý thuyết: Cấu trúc PUMA 560 và Ràng buộc Hình học

Để sử dụng thành công lớp rl::kin::Puma, robot tùy chỉnh bắt buộc phải tuân thủ nghiêm ngặt các ràng buộc hình học mà giải thuật này giả định. Việc hiểu rõ bản chất toán học của PUMA

560 là bước đầu tiên để đảm bảo tính khả thi của dự án.

2.1. Đặc trưng Tô-pô và Tiêu chuẩn Pieper

Robot PUMA (Programmable Universal Machine for Assembly) là đại diện kinh điển của robot chuỗi nối tiếp có cổ tay cầu (spherical wrist). Theo tiêu chuẩn Pieper, một robot 6-DOF sẽ có lời giải IK dạng khép kín (closed-form) nếu ba trục quay liên tiếp cắt nhau tại một điểm.[1]

Trong cấu trúc PUMA 560 và robot hàn của dự án:

- **Khớp 1, 2, 3 (Cơ cấu định vị)**: Chịu trách nhiệm đưa tâm cổ tay (Wrist Center - WC) đến vị trí mong muốn.
- **Khớp 4, 5, 6 (Cơ cấu định hướng)**: Các trục quay của ba khớp này phải giao nhau tại một điểm duy nhất. Điều này cho phép tách rời bài toán vị trí và bài toán hướng.[6]

Ràng buộc hình học bắt buộc đối với rl::kin::Puma: Dựa trên mã nguồn và tài liệu của RL [3], robot tùy chỉnh phải thỏa mãn:

1. **Trục 1 vuông góc với Trục 2**: ($\alpha_1 = \pm 90^\circ$).
2. **Trục 2 song song với Trục 3**: ($\alpha_2 = 0^\circ$). Đây là đặc điểm quan trọng giúp đơn giản hóa phương trình lượng giác của vai và khuỷu.
3. **Trục 3 vuông góc với Trục 4**: ($\alpha_3 = \pm 90^\circ$).
4. **Giao điểm cổ tay**: Trục 4, 5, và 6 phải đồng quy. Nếu thiết kế cơ khí của robot hàn có độ lệch (offset) tại khớp 4 hoặc 5 (thường thấy ở một số robot sơn hoặc hàn rỗng ruột hiện đại), giải thuật rl::kin::Puma sẽ **không chính xác**.
5. **Độ lệch vai (Shoulder Offset)**: PUMA 560 có độ lệch giữa tâm quay khớp 1 và mặt phẳng quay của khớp 2/3 (tham số d_3 trong DH). rl::kin::Puma được thiết kế đặc biệt để xử lý tham số này, khác với các robot phẳng hoàn toàn.

2.2. Hệ tham số Denavit-Hartenberg (DH)

Robotics Library sử dụng quy ước DH Chuẩn (Standard DH), không phải DH Sửa đổi (Modified/Craig's DH).[3] Sự nhầm lẫn giữa hai quy ước này là nguyên nhân hàng đầu dẫn đến sai lệch mô hình.

Trong quy ước DH Chuẩn được RL áp dụng, phép biến đổi từ khung $i-1$ sang khung i được mô tả bởi ma trận:

$$^{i-1}T_i = Rot_{z,\theta_i} \cdot Trans_{z,d_i} \cdot Trans_{x,a_i} \cdot Rot_{x,\alpha_i}$$

Bảng dưới đây tóm tắt ý nghĩa vật lý của các tham số trong ngữ cảnh robot hàn PUMA-like:

Tham số	Ý nghĩa Vật lý	Lưu ý khi trích xuất từ CAD
θ_i	Góc quay khớp	Biến số điều khiển (Joint Angle). Cần xác định vị trí Zero (Home position) chính xác.
d_i	Độ lệch dọc trục z_{i-1}	Khoảng cách giữa hai đường vuông góc chung dọc theo trục khớp. PUMA thường có $d_3 \neq$.
a_i	Độ dài liên kết (Link length)	Khoảng cách ngắn nhất giữa trục z_{i-1} và z_i dọc theo trục x_i.
α_i	Góc xoắn liên kết (Link twist)	Góc giữa trục z_{i-1} và z_i đo quanh trục x_i. Với PUMA, giá trị thường là $0, \pm 90$.

---

3. Quy trình Kỹ thuật: Từ CAD đến Mapping XML

Đây là quy trình chuyển đổi dữ liệu hình học tĩnh từ phần mềm thiết kế (SolidWorks, Inventor) sang định dạng XML động mà Robotics Library có thể hiểu và xử lý.

3.1. Bước 1: Trích xuất Dữ liệu từ CAD

Kỹ sư cần thực hiện gán hệ tọa độ trên mô hình 3D theo quy tắc "Tay phải" (Right-Hand Rule) và tuân thủ quy ước DH chuẩn:

1. **Gốc tọa độ (Base Frame - 0)**: Đặt tại chân đế robot. Trục Z_0 trùng với trục quay khớp 1.
2. **Khung 1**: Gốc tại giao điểm trục 1 và trục 2. Nếu không cắt nhau (do shoulder offset), gốc nằm trên trục 1 tại điểm vuông góc chung. X_1 vuông góc với Z_0 và Z_1.

3. **Khung 2**: Gốc tại tâm khớp 2. Z_2 trùng trục quay khớp 2.
4. **Khung 3**: Gốc tại tâm khớp 3. Z_3 trùng trục quay khớp 3.
5. **Khung 4, 5, 6**: Gốc chung tại tâm giao điểm cổ tay (Wrist Center).

Danh sách tham số cần đo kiểm:

- **Chiều cao bệ (d_1)**: Từ sàn đến tâm trục 2.
- **Độ lệch vai (d_2 hoặc d_3 tùy cách đặt)**: Khoảng cách ngang giữa trục 1 và mặt phẳng cánh tay đòn.
- **Độ dài bắp tay (a_2)**: Khoảng cách tâm trục 2 đến tâm trục 3.
- **Độ lệch khuỷu (d_4)**: Khoảng cách từ mặt phẳng khuỷu đến tâm cổ tay.
- **Độ dài cẳng tay (a_3)**: Thường xấp xỉ 0 hoặc rất nhỏ trên PUMA.

3.2. Bước 2: Định nghĩa File XML cho RL

Robotics Library sử dụng định dạng XML tùy biến để mô tả mô hình động lực học và động học (.rlmdl.xml).[7] Cấu trúc file này phản ánh cấu trúc cây (tree structure) của các liên kết.

Để đảm bảo rl::kin::Puma hoạt động, file XML phải chứa đầy đủ các thẻ mô tả tham số DH. Dưới đây là mẫu file XML chuẩn cho robot hàn tùy chỉnh cấu trúc PUMA:

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rlmdl>
  <model>
    <manufacturer>CustomWeldingCorp</manufacturer>
    <name>WeldBot_6DOF</name>
    <world>
      <rotation>
        <x>0</x><y>0</y><z>0</z>
      </rotation>
      <translation>
        <x>0</x><y>0</y><z>0</z>
      </translation>
      <g>
        <x>0</x><y>0</y><z>9.81</z>
```

```
        </g>
    </world>

    <frame>
        <name>link0</name>
        <revolute id="joint0">
            <min>-160</min><max>160</max> <speed>120</speed>
            <d>0.6718</d> <theta>0</theta> <a>0</a> <alpha>-90</alpha>
        </revolute>
    </frame>

    <frame>
        <name>link1</name>
        <revolute id="joint1">
            <min>-225</min><max>45</max>
            <d>0</d> <theta>0</theta> <a>0.4318</a> <alpha>0</alpha>
        </revolute>
    </frame>

    <frame>
        <name>link2</name>
        <revolute id="joint2">
            <min>-45</min><max>225</max>
            <d>0.15005</d> <theta>0</theta> <a>0.0203</a> <alpha>-90</alpha>
        </revolute>
    </frame>

    <frame>
        <name>link3</name>
        <revolute id="joint3">
            <min>-110</min><max>170</max>
            <d>0.4318</d> <theta>0</theta> <a>0</a> <alpha>90</alpha>
        </revolute>
    </frame>

    <frame>
        <name>link4</name>
        <revolute id="joint4">
            <min>-100</min><max>100</max>
            <d>0</d> <theta>0</theta> <a>0</a> <alpha>-90</alpha>
        </revolute>
    </frame>
```

```
      <frame>
        <name>link5</name>
        <revolute id="joint5">
          <min>-266</min><max>266</max>
          <d>0.05625</d> <theta>0</theta> <a>0</a> <alpha>0</alpha>
        </revolute>
      </frame>
    </model>
</rlmdl>
```

Lưu ý quan trọng về đơn vị: Mặc dù XML có thể ghi độ, nhưng trong code C++, RL sử dụng **Radian** và **Mét**.[8] Bộ nạp (Loader) của RL thường tự động chuyển đổi nếu đơn vị được khai báo, nhưng tốt nhất nên chuẩn hóa đầu vào.

---

4. Kích hoạt Analytical IK (rl::kin::Puma)

Một thách thức lớn khi sử dụng RL là rl::mdl::XmlFactory mặc định sẽ tạo ra một đối tượng rl::mdl::Kinematic chung, sử dụng giải thuật số (Jacobian Inverse Kinematics) cho hàm calculateInverse(). Để sử dụng giải thuật giải tích của PUMA, chúng ta cần "ép" hệ thống khởi tạo lớp dẫn xuất rl::kin::Puma.

4.1. Vấn đề của Factory Mặc định

Khi gọi rl::mdl::XmlFactory::create(), thư viện sẽ phân tích file XML. Nếu không có chỉ thị đặc biệt, nó xây dựng chuỗi động học (kinematic chain) từ các khớp quay/trượt cơ bản. Lớp này không biết rằng robot có cấu trúc PUMA đặc biệt để áp dụng công thức giải tích.[9]

4.2. Giải pháp: Ép kiểu và Khởi tạo Trực tiếp

Có hai phương pháp để tích hợp rl::kin::Puma vào Core Engine của bạn:

Phương pháp A: Kế thừa và Nạp chồng (Khuyên dùng cho tính linh hoạt)

Tạo một lớp quản lý robot hàn riêng, lớp này sở hữu một đối tượng rl::kin::Puma. Thay vì dùng Factory để tạo toàn bộ Model, bạn dùng Factory để đọc dữ liệu, sau đó copy tham số vào đối tượng Puma thủ công hoặc dùng cơ chế nạp chồng.

Phương pháp B: Sử dụng rl::kin::Puma trực tiếp trong C++

Vì kích thước robot là tùy chỉnh nhưng cấu trúc là cố định, cách tối ưu nhất về hiệu năng là khởi tạo trực tiếp lớp rl::kin::Puma và nạp tham số từ file cấu hình (hoặc hardcode nếu robot là duy nhất).

Điều kiện tiên quyết để rl::kin::Puma::calculateInverse hoạt động chính xác:

- Gọi hàm setParameters() (nếu có) hoặc thiết lập trực tiếp các biến thành viên đại diện cho độ dài liên kết (a_2, a_3, d_3, d_4).
- Đảm bảo file header #include <rl/kin/Puma.h> được bao gồm.[4]

---

5. Xử lý Đa nghiệm và Cờ Cấu hình (Configuration Flags)

Với robot 6-DOF PUMA, một vị trí và hướng của mỏ hàn (End-Effector Pose) có thể đạt được bởi tối đa **8 bộ nghiệm** khác nhau. Việc chọn nghiệm nào là bài toán tối quan trọng trong hàn để tránh va chạm và duy trì sự liên tục của đường hàn.

5.1. Định nghĩa 8 Bộ nghiệm (The Geometric Multiverse)

Các nghiệm được phân loại dựa trên 3 chỉ số nhị phân (flags):

Cờ Cấu hình (Flag)	Giá trị (Bit)	Mô tả Hình học	Biểu diễn Toán học trong RL
ARM	RIGHT / LEFT	Xác định xem vai robot đang ở bên phải hay trái của đường thẳng nối gốc và cổ tay.	Dấu của θ_1 và hình chiếu của WC lên mặt phẳng xy.
ELBOW	UP / DOWN	Xác định khuỷu tay chổng lên hay chúc xuống so với đường nối vai-cổ tay.	Dấu của $\sin(\theta_3)$.
WRIST	FLIP / NO-FLIP	Xác định cổ tay có bị "lật" ngược để đạt cùng một hướng hay không.	Dấu của $\cos(\theta_5)$ (thường liên quan đến việc θ_5 dương hay âm).

Tổng hợp: $2 \times 2 \times 2 = 8$ nghiệm.

5.2. Cấu trúc Trả về của calculateInverse

Hàm rl::kin::Puma::calculateInverse(const Transform& x, VectorList& q) không trả về một vector đơn lẻ.

- **Input**: x (Ma trận biến đổi đồng nhất 4x4 của mỏ hàn).
- **Output**: q (Danh sách std::vector chứa các rl::math::Vector).
- **Return**: bool (True nếu tìm thấy ít nhất 1 nghiệm).

5.3. Chiến lược Lọc nghiệm (Filtering Strategy)

Kỹ sư trưởng cần xây dựng một bộ lọc (Wrapper) bọc lấy hàm calculateInverse. Bộ lọc này thực hiện:

1. **Lọc giới hạn khớp (Joint Limit Check)**: Loại bỏ các nghiệm mà θ_i nằm ngoài [min, max] quy định trong XML.
2. **Lọc theo Cờ (Flag Matching)**: Chỉ giữ lại các nghiệm khớp với cấu hình mong muốn (ví dụ: ARM_LEFT, ELBOW_UP để hàn trần).
3. **Tối ưu hóa hành trình (Continuity Check)**: Trong số các nghiệm hợp lệ còn lại, chọn nghiệm gần nhất với vị trí khớp hiện tại (Δq nhỏ nhất) để đảm bảo robot không thực hiện các chuyển động giật cục nguy hiểm.

---

6. Triển khai Mã nguồn C++ (Reference Implementation)

Dưới đây là mã nguồn C++ mẫu mô phỏng lớp WeldingRobotKinematics. Mã này minh họa cách khởi tạo rl::kin::Puma, nạp tham số (giả định từ XML), và thực hiện giải IK có lọc cờ.

6.1. File Header: WeldingRobotKinematics.h

C++

```
/**
 * @file WeldingRobotKinematics.h
 * @brief Lớp xử lý động học chuyên dụng cho robot hàn PUMA-type
 */

#ifndef WELDING_ROBOT_KINEMATICS_H
#define WELDING_ROBOT_KINEMATICS_H
```

```cpp
// Bao gồm các thư viện RL cốt lõi
#include <rl/kin/Puma.h>
#include <rl/math/Transform.h>
#include <rl/math/Vector.h>
#include <vector>
#include <memory>
#include <cmath>

// Định nghĩa các cờ cấu hình (Configuration Flags)
enum class ArmConfig { LEFT = 0, RIGHT = 1, ANY = 2 };
enum class ElbowConfig { UP = 0, DOWN = 1, ANY = 2 };
enum class WristConfig { FLIP = 0, NOFLIP = 1, ANY = 2 };

class WeldingRobotKinematics {
public:
    WeldingRobotKinematics();
    virtual ~WeldingRobotKinematics();

    /**
     * @brief Khởi tạo tham số DH cho robot tùy chỉnh
     * Thay vì dùng XmlFactory generic, ta nạp trực tiếp vào rl::kin::Puma
     */
    void initializeCustomPuma(double a2, double a3, double d3, double d4);

    /**
     * @brief Giải động học nghịch đảo với bộ lọc cờ
     * @param targetPose Vị trí/Hướng mong muốn của mỏ hàn (TCP)
     * @param currentJoints Góc khớp hiện tại (để chọn nghiệm gần nhất)
     * @param flags Các cờ cấu hình mong muốn
     * @param solution [Output] Nghiệm tốt nhất tìm được
     * @return true nếu tìm thấy nghiệm hợp lệ
     */
    bool solveAnalyticalIK(const rl::math::Transform& targetPose,
                           const rl::math::Vector& currentJoints,
                           ArmConfig arm,
                           ElbowConfig elbow,
                           WristConfig wrist,
                           rl::math::Vector& solution);

private:
    // Đối tượng cốt lõi: Sử dụng rl::kin::Puma thay vì rl::kin::Kinematics
    std::shared_ptr<rl::kin::Puma> pumaSolver;

    // Giới hạn khớp (đọc từ XML hoặc cấu hình cứng)
    rl::math::Vector minLimits;
```

```
    rl::math::Vector maxLimits;

    // Hàm tiện ích xác định cấu hình của một nghiệm bất kỳ
    void identifyConfiguration(const rl::math::Vector& q,
                               ArmConfig& arm,
                               ElbowConfig& elbow,
                               WristConfig& wrist);
};

#endif // WELDING_ROBOT_KINEMATICS_H
```

6.2. File Implementation: WeldingRobotKinematics.cpp

C++

```
/**
 * @file WeldingRobotKinematics.cpp
 */

#include "WeldingRobotKinematics.h"
#include <iostream>
#include <limits>

WeldingRobotKinematics::WeldingRobotKinematics() {
    // Khởi tạo đối tượng Puma chuyên dụng
    this->pumaSolver = std::make_shared<rl::kin::Puma>();

    // Khởi tạo giới hạn khớp (Ví dụ: 6 trục)
    this->minLimits.resize(6);
    this->maxLimits.resize(6);
    // Cài đặt giới hạn mặc định (cần cập nhật từ XML thực tế)
    this->minLimits << -2.8, -2.0, -1.0, -3.0, -2.0, -6.0; // Radian
    this->maxLimits <<  2.8,  2.0,  3.0,  3.0,  2.0,  6.0;
}

WeldingRobotKinematics::~WeldingRobotKinematics() {}

void WeldingRobotKinematics::initializeCustomPuma(double a2, double a3, double d3, double d4) {
    // Cập nhật tham số DH cho solver
    // Lưu ý: rl::kin::Puma kế thừa rl::kin::Kinematics, cấu trúc Links được lưu trong vector
    // Ta cần cấu hình lại các Link để khớp với kích thước tùy chỉnh.
```

```
    // Xóa cấu trúc mặc định (nếu có)
    this->pumaSolver->frames.clear();
    this->pumaSolver->transforms.clear();

    // Tái định nghĩa chuỗi động học (Ví dụ đơn giản hóa việc đẩy tham số vào vector)
    // Trong thực tế, bạn sẽ dùng hàm setParameter hoặc thao tác trực tiếp trên public members
    // nếu thư viện cho phép, hoặc xây dựng lại vector Links.

    // QUAN TRỌNG: rl::kin::Puma tính toán dựa trên các biến a2, d3...
    // Hãy đảm bảo cập nhật đúng biến thành viên của lớp Puma (nếu public)
    // hoặc thông qua cơ chế load DH chuẩn.

    // Ví dụ giả định cập nhật tham số (dựa trên API chuẩn của RL):
    // RL sử dụng cấu trúc cây, ta cần duyệt và gán giá trị a, d cho từng khớp.
    //... (Code nạp DH chi tiết sẽ dài, ở đây tập trung vào logic IK)
}

void WeldingRobotKinematics::identifyConfiguration(const rl::math::Vector& q,
                                                   ArmConfig& arm,
                                                   ElbowConfig& elbow,
                                                   WristConfig& wrist) {
    // Logic xác định cờ dựa trên giá trị góc khớp
    // Lưu ý: Logic này phụ thuộc vào quy ước dấu của từng hãng robot

    // 1. Elbow Check: Dựa vào góc khớp 3
    if (q(2) >= 0) elbow = ElbowConfig::UP; // Giả định dương là UP
    else elbow = ElbowConfig::DOWN;

    // 2. Wrist Check: Dựa vào góc khớp 5
    if (q(4) >= 0) wrist = WristConfig::NOFLIP;
    else wrist = WristConfig::FLIP;

    // 3. Arm Check: Phức tạp hơn, thường dựa vào vị trí Wrist Center trong hệ tọa độ vai
    // Một cách đơn giản (nhưng không chính xác tuyệt đối cho mọi vùng):
    // Kiểm tra tương quan góc khớp 1 và vị trí đầu cuối.
    arm = ArmConfig::ANY; // Placeholder
}

bool WeldingRobotKinematics::solveAnalyticalIK(const rl::math::Transform& targetPose,
                                               const rl::math::Vector& currentJoints,
                                               ArmConfig reqArm,
                                               ElbowConfig reqElbow,
```

```
                                WristConfig reqWrist,
                                rl::math::Vector& solution) {
    rl::math::VectorList allSolutions;

    // GỌI HÀM ANALYTICAL IK CỦA RL
    bool found = this->pumaSolver->calculateInverse(targetPose, allSolutions);

    if (!found |

| allSolutions.empty()) {
        return false;
    }

    double minDistance = std::numeric_limits<double>::max();
    bool validSolutionFound = false;

    // Duyệt qua tất cả các nghiệm trả về
    for (const auto& q : allSolutions) {
        // 1. Kiểm tra giới hạn khớp (Hard Limits)
        bool withinLimits = true;
        for (int i = 0; i < 6; ++i) {
            if (q(i) < this->minLimits(i) |

| q(i) > this->maxLimits(i)) {
                withinLimits = false;
                break;
            }
        }
        if (!withinLimits) continue;

        // 2. Kiểm tra Cờ cấu hình (Geometric Config)
        ArmConfig qArm; ElbowConfig qElbow; WristConfig qWrist;
        identifyConfiguration(q, qArm, qElbow, qWrist);

        if (reqElbow!= ElbowConfig::ANY && qElbow!= reqElbow) continue;
        if (reqWrist!= WristConfig::ANY && qWrist!= reqWrist) continue;
        // if (reqArm...) // Tương tự

        // 3. Chọn nghiệm gần nhất (Continuity Optimization)
        // Tính khoảng cách Euclid trong không gian khớp (Weighted distance nếu cần)
        double distance = (q - currentJoints).norm();

        if (distance < minDistance) {
```

```
            minDistance = distance;
            solution = q;
            validSolutionFound = true;
        }
    }

    return validSolutionFound;
}
```

---

7. Tích hợp Tránh va chạm (Collision Avoidance)

Trong môi trường hàn, việc chỉ giải IK là chưa đủ. Mỏ hàn có thể va vào đồ gá hoặc phôi hàn. Robotics Library cung cấp module rl::sg (Scene Graph) để giải quyết vấn đề này.[11]

Quy trình tích hợp vào vòng lặp điều khiển:

1. **Bước 1**: Nhận allSolutions từ solveAnalyticalIK.
2. **Bước 2**: Thay vì chọn ngay nghiệm tốt nhất, hãy tạo một danh sách "Ứng viên Tiềm năng" (Candidate List).
3. **Bước 3**: Với mỗi ứng viên, cập nhật mô hình va chạm:
   C++
   ```
   // Cập nhật mô hình hình học
   robotSceneModel->setPosition(q_candidate);
   robotSceneModel->updateFrames();
   ```
4. **Bước 4**: Kiểm tra va chạm với môi trường:
   C++
   ```
   rl::sg::SimpleScene* scene;
   //... khởi tạo scene...
   if (scene->areColliding()) {
       // Loại bỏ ứng viên này
       continue;
   }
   ```
5. **Bước 5**: Chọn nghiệm tốt nhất từ danh sách ứng viên không va chạm còn lại.

---

8. Kết luận và Khuyến nghị

Việc chuyển đổi từ một robot hàn tùy chỉnh sang mô hình toán học rl::kin::Puma là một quy trình đòi hỏi sự chính xác tuyệt đối trong khâu ánh xạ DH. Bằng cách sử dụng trực tiếp lớp

rl::kin::Puma và bỏ qua cơ chế Factory mặc định, hệ thống điều khiển sẽ đạt được hiệu suất tính toán thời gian thực và độ ổn định cần thiết cho các đường hàn phức tạp.

Các bước hành động ngay:

1. Kiểm tra lại bản vẽ CAD để xác nhận các điều kiện trực giao/song song của trục 1, 2, 3 và sự đồng quy của trục 4, 5, 6.
2. Xây dựng file XML theo mẫu đã cung cấp, chú ý đơn vị đo.
3. Triển khai lớp C++ Wrapper (WeldingRobotKinematics) để quản lý logic lọc nghiệm.
4. Thực hiện Unit Test: So sánh kết quả của rl::kin::Puma::calculateForward (FK) và calculateInverse (IK) để đảm bảo sai số vị trí $< 10^{-6} m$.

Giải pháp này đảm bảo robot hàn của bạn không chỉ hoạt động chính xác mà còn có khả năng xử lý thông minh các tình huống đa nghiệm, đặc trưng của các hệ thống robot công nghiệp cao cấp.

Works cited

1. How to set up KDL (Kinematics and Dynamics Library) for Solving Kinematics of Serial Chain Manipulators in C++ with ROS2 - FusyBots, accessed February 1, 2026, https://www.fusybots.com/post/how-to-set-up-kdl-kinematics-and-dynamics-library-for-solving-kinematics-of-serial-chain-manipulat
2. Inverse Kinematics in Robotics: What You Need to Know - RoboDK blog, accessed February 1, 2026, https://robodk.com/blog/inverse-kinematics-in-robotics-what-you-need-to-know/
3. rl/kin/Puma.cpp File Reference - Robotics Library, accessed February 1, 2026, https://doc.roboticslibrary.org/0.7.0/d9/d15/_puma_8cpp.html
4. rl/kin/Puma.h File Reference - Robotics Library, accessed February 1, 2026, https://doc.roboticslibrary.org/0.6.2/d6/dc9/_puma_8h.html
5. Geometric Approach in Solving Inverse Kinematics of PUMA Robots - SciSpace, accessed February 1, 2026, https://scispace.com/pdf/geometric-approach-in-solving-inverse-kinematics-of-puma-ph4ngr22yh.pdf
6. Kinematic diagram of the PUMA robot. - ResearchGate, accessed February 1, 2026, https://www.researchgate.net/figure/Kinematic-diagram-of-the-PUMA-robot_fig1_263284372
7. Create a Robot Model - Robotics Library, accessed February 1, 2026, https://www.roboticslibrary.org/tutorials/create-a-robot-model/
8. First Steps with RL on Windows - Robotics Library, accessed February 1, 2026, https://www.roboticslibrary.org/tutorials/first-steps-windows/
9. API - Robotics Library, accessed February 1, 2026, https://www.roboticslibrary.org/api

10. First Steps with RL on Linux - Robotics Library, accessed February 1, 2026, https://www.roboticslibrary.org/tutorials/first-steps-linux/
11. Robotics Library: Robotics Library, accessed February 1, 2026, https://doc.roboticslibrary.org/
12. Robotics Library - mediaTUM, accessed February 1, 2026, https://mediatum.ub.tum.de/doc/1287151/581170.pdf