

基于 ZYNQ 的机械臂运动学逆解加速系统（初稿 1）

摘要：针对具身智能系统中机械臂运动学逆解(IK)面临的计算密集度高、传统计算平台延迟大以及边缘端功耗受限等问题，提出了一种基于 ZYNQ 异构 SoC 的 5 自由度机器人运动学逆解加速系统。为解决 FPGA 传统开发周期长、编程复杂的痛点，采用高层次综合(HLS)语言设计专用 IP 核，并构建了基于 LM 算法的软硬协同求解架构。针对数值迭代中高频且耗时的矩阵运算瓶颈，在 PL 端实现了基于 Cholesky 分解的线性方程组求解加速，并通过数组分区、循环展开及流水线并行等多维度优化策略，最大化硬件并行计算能力；同时利用 PS 端处理非线性函数与误差构建，实现了资源与精度的平衡。实验结果表明：该系统在保证计算精度与软件高度一致的前提下，单步迭代加速比达 9.10 倍，端到端求解速度提升 2.22 倍，且资源占用率较低，验证了该方案在边缘计算场景下的实时性与高效性。

关键词：运动学逆解；ZYNQ；高层次综合；LM 算法；实时

英文摘要及关键词将在初稿完成后提供。

0 引言

具身智能（Embodied AI）是人工智能与机器人技术深度融合的前沿方向，强调智能体通过物理实体与真实环境进行交互，从而具备感知、理解和行动的能力^[1]。在具身智能系统中，运动学控制作为连接高层决策与底层执行的核心环节，直接决定了机器人对环境响应的精度与速度。尤其是运动学逆解

（Inverse Kinematics, IK），作为将末端执行器位姿映射到关节空间的关键计算任务，其求解效率和实时性对机器人的灵活操控至关重要。因此，研究高效、低延迟的运动学逆解方法，对于推动具身智能技术的实际落地具有重要的现实意义。

然而，传统计算平台在运动学逆解任务中面临显著瓶颈。纯 CPU 处理复杂矩阵运算时计算延迟较高，难以满足实时控制需求；GPU 虽具备强大的并行计算能力，但其功耗通常较高，不利于移动电池驱动的独立机器人长时工作^[2]。相比之下，FPGA 凭借其硬件可重构特性、低功耗和极低延迟的优势，被视为边缘计算的理想选择^[3]。研究表明，在并行量适中的情况下，FPGA 在低延迟和高吞吐方面的表现显著优于 GPU 和 CPU^[4]。近年来，学术界围绕 FPGA 加速运动学逆解开展了诸多研究：张得礼等^[5]提出了一种基于 FPGA 的高并行机器人运动学逆解方法，采用 CORDIC 算法与并行流水线设计，显著提高了计算效率；Konrad Gac 等^[6]开发了专用于平方根计算的 FPGA 硬件加速器，有效降低了高速铣削机器人的计算延迟；刘文正等^[7]针对工业机器人末端执行器偏差及逆解延迟问题，提出了一种融合物理机制与数据驱动模型的 FPGA 加速方法。尽管上述研究在特定场景下取得了良好效果，但仍存在以下问题：一是传统 FPGA 开发依赖 Verilog/VHDL 等硬件描述语言，编程复杂、开发周期长、调试

效率低；二是边缘端 FPGA 资源有限，如何在资源约束下实现高效的资源复用与并行优化仍是难点；三是多数研究仍停留在算法验证阶段，缺乏面向实际部署的端到端系统集成方案。

针对上述问题，本文提出了一种基于 ZYNQ 异构 SoC 的 5 自由度机械臂运动学逆解加速系统。该系统采用高层次综合（HLS）语言设计专用运动学逆解 IP 核，有效解决了传统 Verilog 编程开发效率低的问题；通过多维度优化策略，综合运用流水线、循环展开指令及 LUT 查找表加速非线性函数计算，充分利用片内资源，进而发挥了 FPGA 的并行计算能力。

1 运动学控制

机器人运动学控制旨在建立关节空间与末端笛卡尔空间之间的映射关系，其包含正解与逆解两个部分，是连接顶层运动规划与底层电机驱动的桥梁。其中，运动学逆解作为根据目标位姿反求关节角度的核心环节，由于其数学本质为非线性方程组的求解，且易受奇异点影响，一直是影响机器人实时控制性能的关键瓶颈。因此，构建适合硬件并行计算的逆解算法模型，是本系统设计的理论前提。

1.1 正运动学

在整个运动学控制环路的开始，机械臂首先需要知道当前执行器末端的位姿，这可以通过运动学正解（Forward Kinematics, FK）实现。正运动学的目的就是由关节变量的函数，计算末端执行器的位姿。

正运动学的求解的主流方法是 DH 法（Denavit-Hartenberg Method）。DH 法用一系列变换矩阵描述相邻连杆之间的刚体变换，逐级建立坐标系使用矩阵连乘得到末端位姿。在运动学建模过程中，将所有关节变量都看作是转角 θ_i ，关节空间 \mathbf{Q} 表示各独立关节空间的笛卡尔积。当给定一组关节转角 $\boldsymbol{\theta} \in \mathbf{Q}$ 时，希望确定工具坐标系 T 相对于基础坐标系 S 的映射。运动学正解可用一个反映此相对关系的映射 $g_{st} : \mathbf{Q} \rightarrow SE(3)$ 来表示。如果定义 $g_{l_{i-1}l_i}(\theta_i)$ 为相邻连杆坐标系间的变换，那么总的运动方程为：

$$g_{st}(\boldsymbol{\theta}) = g_{sl_1}(\theta_1) g_{l_1l_2}(\theta_2) \cdots g_{l_{n-1}l_n}(\theta_n) g_{l_nt} \quad (1.1)$$

式(1.1)是用相邻连杆坐标系的相对变换表示的开链机器人运动学正解的一般公式，这便是 DH 法求解正运动学的一般方法，其中每个变换 $g_{l_{i-1}l_i}(\theta_i)$ 都由 4 个参数（ a, α, d, θ ）的变换矩阵给出^[9]。

DH 方法要求坐标系间的变换必须按顺序连乘，而基本变换序列（Elementary Transform Sequence, ETS）是相较于 DH 法更简单、更直观的方法，它无需严格定义连杆坐标系，可以任意选择基本变换顺序和坐标轴方向，并且其在开源机器人工具箱 Robotics Toolbox for Python 中被广泛采用。考虑到本系统基于 LeRobot SO-101 开源机器人项目，机器人模型 URDF 文件已被提供，使用 ETS 法实现运动学正解更为简单。

ETS 法也是一种计算机械臂从基坐标系到夹爪坐标系齐次变换矩阵的方法，它将开链机器人的每一个基本运动分解为最基本的平移和旋转，包括平移矩阵 $T_x(a), T_y(b), T_z(c)$ 和旋转矩阵 $R_x(\alpha), R_y(\beta), R_z(\theta)$ ，然后按顺序连乘这些基本齐次变换矩阵。使用 ETS 法，本系统的 5 自由度开链机械臂末端表示为：

$$\begin{cases} \text{ETS} = J_1 J_2 J_3 J_4 J_5 J_6 \\ J_1 = T_x(0.002798) T_z(0.05031) R_z(\theta_1) \\ J_2 = T_x(0.02957) T_z(0.11590) R_y(\theta_2) \\ J_3 = T_x(0.11323) T_z(0.00500) R_y(\theta_3) \\ J_4 = T_x(0.0650) T_z(0.00519) R_y(\theta_4) \\ J_5 = T_x(0.02413) T_z(0) R_x(\theta_5) \\ J_6 = T_x(0.1) \end{cases} \quad (1.2)$$

式(1.2)中 J_i 表示该机器人的第 i 个关节舵机，末端 6 号舵机控制夹爪，对自由度无贡献。例如 $R_z(\theta_1)$ 中的 θ_1 表示关节 1 绕 z 轴的转角，其弧度值可以通过舵机编码器读数转换得到。

1.2 运动学逆解

1.2.1 逆运动学问题概述

正运动学方程建立了关节变量与末端执行器位姿之间的函数关系。而逆运动学则是给定机械臂末端执行器的位置和姿态，计算所有可达给定位置和姿态的关节角^[8]。求解该问题的目的是将提供给末端执行器目标位姿变换为各关节的运动，使得期望的运动能够得到执行。正运动学方程只要关节变量已知，末端执行器的位姿都是唯一的。逆运动学问题则复杂得多，原因是要求解的方程通常是非线性的，因而不一定能找到封闭解，另外还可能存在多解或无解的情况。

求运动学逆解的方法包括数值法和解析法，其目的是求得闭合形式的解。解析法包含几何法和代数法，几何法依赖几何结构寻找系统中的几何关系，通过余弦定理和三角关系列出方程组，解如上方程组即可得到所求的关节转角，

代数法则是基于式(1.1)的正运动学方程和目标位姿列出方程组。所列出的非线性方程组经过变量代换，都将变成多项式方程组。

1.2.2 基础数值迭代算法

对于一般无相交轴线的六自由度机械臂，可以通过分解消元法将多变量多项式消元，转化为单变量的高次多项式求解。虽然解析法能求得所有可行解，但由于一般结构的消元过程异常繁杂且存在数值计算上的病态矩阵问题，在复杂环境下或实时性要求极高时，采用数值法求运动学逆解问题往往更具普适性。在这之前，首先需要引入操作臂雅可比矩阵。

雅可比矩阵构成表征机械手最重要的工具之一，它可以用来计算逆运动学。假设有函数 $\mathbf{Y} = F(\mathbf{X})$ ，其中 \mathbf{X} 、 \mathbf{Y} 和 \mathbf{F} 均为矢量，利用多元函数微分法将两侧对 \mathbf{Y} 求微分，有 $\delta\mathbf{Y} = \frac{\partial\mathbf{F}}{\partial\mathbf{X}}\delta\mathbf{X}$ ，其中 $\frac{\partial\mathbf{F}}{\partial\mathbf{X}}$ 便可用雅可比矩阵 $\mathbf{J}(\mathbf{X})$ 表示，即：

$$\delta\mathbf{Y} = \mathbf{J}(\mathbf{X})\delta\mathbf{X} \quad (1.3)$$

定义位姿 \mathbf{g} 是一个变换矩阵，满足 $\mathbf{g} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ 0 & 1 \end{bmatrix}$ ，其中 \mathbf{R} 为旋转矩阵， \mathbf{p} 为位置矢量。接着设目标位姿为 \mathbf{g}_t ，当前位姿为 $\mathbf{g}(\boldsymbol{\theta})$ ，我们的目的是使得 $\mathbf{g}_t - \mathbf{g}(\boldsymbol{\theta}) = 0$ 。常见的数值求解方法是牛顿迭代法，假设当前的关节角估计值是 $\boldsymbol{\theta}_k$ ，需要找到一个微小的增量 $\Delta\boldsymbol{\theta}$ ，使得更新后的角度 $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$ 能让位姿更接近目标位姿 \mathbf{g}_t 。对位姿 $\mathbf{g}(\boldsymbol{\theta})$ 在当前角度 $\boldsymbol{\theta}_k$ 处进行一阶泰勒展开：

$$\mathbf{g}(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) \approx \mathbf{g}(\boldsymbol{\theta}_k) + \mathbf{J}(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} \quad (1.4)$$

在式(1.3)中， $\mathbf{g}(\boldsymbol{\theta}_k)$ 是当前的末端位姿， $\mathbf{J}(\boldsymbol{\theta}_k)$ 是当前的雅可比矩阵， $\Delta\boldsymbol{\theta}$ 是到达目标位姿需要的关节角增量。我们希望加上这个增量后，机器人的位姿正好到达目标位姿 \mathbf{g}_t 。因此，令等式左边等于 \mathbf{g}_t ：

$$\mathbf{g}_t \approx \mathbf{g}(\boldsymbol{\theta}_k) + \mathbf{J}(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} \quad (1.5)$$

定义当前的误差向量 \mathbf{e}_k 为目标位姿 \mathbf{g}_t 与当前位姿 $\mathbf{g}(\boldsymbol{\theta}_k)$ 之间的差值，那么有：

$$\mathbf{e}_k = \mathbf{J}(\boldsymbol{\theta}_k) \Delta \boldsymbol{\theta} \quad (1.6)$$

上式可以看作是从位姿增量到关节角度增量的线性变换。通过它们之间的增量关系，就可以描述机器人从当前位姿到目标位姿的变换。对于线性方程组(1.6)，如果 \mathbf{J} 是可逆的，那么可以求得关节角增量如下：

$$\Delta \boldsymbol{\theta} = [\mathbf{J}(\boldsymbol{\theta}_k)]^{-1} \mathbf{e}_k \quad (1.7)$$

上式是牛顿迭代法关于关节角增量的迭代步长。它首先需要计算当前误差 \mathbf{e} ，接着计算当前雅可比矩阵 \mathbf{J} ，然后按照上式计算 $\Delta \boldsymbol{\theta}$ ，最后更新关节角 $\boldsymbol{\theta}$ 。重复上述步骤，直到误差 \mathbf{e} 足够小^[10]。虽然牛顿迭代法收敛速度快，但其计算难点在于矩阵求逆。奇异矩阵是没有逆矩阵的，并且奇异点附近的迭代结果容易发散。

另一种方法是梯度下降法，其推导方法的关键在于最小化目标函数 $L(\boldsymbol{\theta})$ ，

这里不再赘述。经过推导可以得到其关节角增量的迭代步长为 $\Delta \boldsymbol{\theta} = \alpha \mathbf{J}^T \mathbf{e}$ ，其中 α 为学习率。虽然梯度下降法迭代速度慢，但计算过程没那么复杂，在奇异点附近仍然能保证稳定的迭代。

1.2.3 LM 算法

可以看到上述两种算法各有优缺点，针对这个问题，Levenberg^[11]和Marquardt^[12]提出了一种最小二乘问题的求解方法，称作 Levenberg-Marquardt 方法，也称作阻尼最小二乘法（Damped Least Squares, DLS），它正是为了结合牛顿法和梯度下降法的优点而诞生的，它是目前机器人 IK 中最常用、最稳定的数值算法之一。LM 法构造了以下迭代公式：

$$\Delta \boldsymbol{\theta} = (\mathbf{J}^T \mathbf{J} + \lambda^2 \mathbf{I})^{-1} \mathbf{J}^T \mathbf{e} \quad (1.8)$$

上式中的 λ 为阻尼因子。当 $\lambda \approx 0$ 时，上式与牛顿迭代法相近；当 λ 很大时，上式与梯度下降法相近，其中 $\alpha \approx \lambda^{-1}$ 。其迭代算法的实现步骤如下：

Step1: 初始化关节角 $\boldsymbol{\theta}_0$ 和阻尼因子 λ ，给定当前位姿 $\mathbf{g}(\boldsymbol{\theta}_0)$ 和目标位姿 \mathbf{g}_t ；

Step2: 计算当前误差 $\mathbf{e} = \mathbf{g}_t - \mathbf{g}(\boldsymbol{\theta})$ 和当前雅可比矩阵 \mathbf{J} ；

Step3: 把已知参量带入式(1.8)，将求出的 $\Delta \boldsymbol{\theta}$ 迭加到原来的关节角上；

Step4: 一般而言，阻尼因子 λ 不变。若采用动态更新，则根据当前加权误差 E

更新 λ （其中加权误差公式为 $E = \frac{1}{2} \mathbf{e}^T \mathbf{w} \mathbf{e}$ ， \mathbf{w} 为权重矩阵）。一种是将 λE 赋值给 λ ，另一种是将 $\lambda + E$ 赋值给 λ ；

Step5: 回到 Step3，直到误差小于设定值或达到最大迭代次数。

1.2.4 雅可比矩阵计算

在 LM 算法的迭代步骤 Step 2 中，每一轮循环都需要实时更新雅可比矩阵 \mathbf{J} 。雅可比矩阵描述了关节空间速度到笛卡尔空间速度的映射关系，其准确性和计算效率直接决定了逆解算法的收敛性能。一般来说，求解雅可比矩阵的方法主要分为数值法和解析法两种。

数值法主要基于有限差分法来近似偏导数。其核心思想是通过给每个关节角施加一个微小扰动 $\Delta \theta$ ，利用正运动学方程计算扰动后的末端位姿 $\mathbf{g}(\theta)$ ，从而近似求得偏导数。只对第 i 个关节添加扰动，按照如下公式计算雅可比矩阵的第 i 列 \mathbf{J}_i ：

$$\mathbf{J}_i \approx \frac{\mathbf{g}(\theta + \Delta \theta) - \mathbf{g}(\theta)}{\Delta \theta} \quad (1.9)$$

依次扰动机器人的所有关节，就能够计算出当前的雅可比矩阵 \mathbf{J} 。数值法的优点在于通用性强，无需针对特定机械臂构型推导复杂的微分公式。然而，其缺点也十分明显：一是存在截断误差和舍入误差，计算精度受步长 $\Delta \theta$ 影响较大；二是计算量大，这在实时控制中会带来较大的时间开销。

解析法则是基于机械臂的运动学方程，利用代数微分法或矢量积方法，推导出雅可比矩阵各元素的显式数学表达式。对于串联机械臂，可以按照雅可比矩阵的定义计算出代数雅可比矩阵，也可以利用连杆坐标系之间的变换矩阵，直接计算出几何雅可比矩阵，几何雅可比是一种更常用的方法。

对于旋转关节，设机械臂第 i 个关节的旋转轴单位矢量为 \mathbf{z}_{i-1} ，末端执行器位置矢量为 \mathbf{p}_e ，第 i 个坐标系原点位置矢量为 \mathbf{p}_{i-1} ，则根据矢量积法，雅可比矩阵的第 i 列 \mathbf{J}_i 可由下式直接给出：

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} \quad (1.10)$$

解析法的最大优势在于计算结果精确，不存在数值近似误差，且在数学形式上是闭合的。其次是数值法需要大量多次调用正运动学方程，串行计算延时

高，并行资源开销大，而几何解析法只需要调用 1 次正运动学方程计算出 \mathbf{z} 和 \mathbf{p} ，接着只需要进行叉乘和减法计算，在 FPGA 上完全可以循环展开并行处理，以充分发挥硬件加速的优势。因此在本系统中，计算几何雅可比矩阵是更优的选择。

1.2.5 线性方程组求解

在 LM 算法的迭代循环中，核心计算步骤在于求解关节角增量 $\Delta\theta$ 。虽然式 (1.8) 在数学形式上表现为矩阵求逆，但在数值计算中，直接对矩阵求逆不仅计算量巨大，且容易引入数值误差。因此，实际工程中通常将其转化为求解如下形式的线性方程组：

$$(\mathbf{J}^T \mathbf{J} + \lambda^2 \mathbf{I}) \Delta\theta = \mathbf{J}^T \mathbf{e} \quad (1.11)$$

针对形如 $\mathbf{Ax} = \mathbf{b}$ 的线性方程组，数值分析中存在多种求解方法，包括高斯消元法、LU 分解、QR 分解以及 SVD 分解等。不同的方法在计算复杂度及数值稳定性上各有优劣。

在本系统中，我们选择使用 Cholesky 矩阵分解法。其一在于系数矩阵的特殊性质，由于 $\mathbf{J}^T \mathbf{J}$ 本身为半正定对称矩阵，引入阻尼项 $\lambda^2 \mathbf{I}$ 后，系数矩阵

$\mathbf{A} = \mathbf{J}^T \mathbf{J} + \lambda^2 \mathbf{I}$ 必然成为一个对称正定矩阵。针对对称正定矩阵，Cholesky 分解相比于通用的 LU 分解，其无需选主元，计算效率更高，并且数值稳定性也更好。其二是该方法在 FPGA 硬件实现上具有显著的天然优势，其迭代空间和数据依赖关系非常适合开发细粒度流水线并行^[13]。利用 Cholesky 分解求解该线性方程组的具体流程如下：

Step1: 设系数矩阵 $\mathbf{A} = \mathbf{LL}^T$ ，其中 \mathbf{L} 是下三角矩阵，这样有 $\mathbf{LL}^T \mathbf{x} = \mathbf{b}$ ，其中矩阵 \mathbf{L} 按如下公式求解：

$$\begin{cases} L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right), & i > j \\ L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \end{cases} \quad (1.12)$$

Step2: 设 $\mathbf{L}^T \mathbf{x} = \mathbf{y}$ ，先解下三角方程组 $\mathbf{Ly} = \mathbf{b}$ ，按 $y_i = \frac{b_i - \sum_{k=0}^{j-1} L_{ij} y_j}{L_{ii}}$ 求解，这一

步称为前向代入；

Step3: 求解上三角方程组 $\mathbf{L}^T \mathbf{x} = \mathbf{y}$ ，按 $x_i = \frac{y_i - \sum_{k=i+1}^{n-1} L_{ji} x_j}{L_{ii}}$ 求解，这一步称为后向代入。

2 FPGA 端优化策略

第一节建立了基于 LM 法的机械臂运动学逆解模型。虽然该算法稳定性好，但其迭代过程中涉及频繁的雅可比矩阵更新与线性方程组求解，计算复杂度极高。传统的串行实现方式受限于指令执行周期，难以满足实时性要求，并且直接将代码移植至 FPGA 无法有效利用其硬件特性。针对 ZYNQ 异构 SoC 平台的架构特点，本文采用高层次综合（HLS）语言设计方法，提出了一套多维度的硬件加速优化策略。

2.1 Cholesky 分解优化

在 Cholesky 算子内部包含矩阵分解、前向代入和后向代入，这些过程存在严格的数据依赖，是流水线优化的关键。为此，本系统从存储架构、除法优化及循环展开三个维度进行了针对性优化。

2.1.1 数组分区

FPGA 中的 BRAM 通常只有两个读写端口，无法满足矩阵运算中多元素同时访问的需求。在本 SO-101 机械臂系统中，自由度 $N=5$ ，矩阵维度较小。为了实现单时钟周期内的数据吞吐，本系统利用 `ap_fixed` 数据类型的位宽优势，结合 HLS 的数组分区指令，对涉及的雅可比矩阵 \mathbf{J} 、系数矩阵 \mathbf{A} 及下三角矩阵 \mathbf{L} 实施完全数组分区：

```
#pragma HLS ARRAY_PARTITION variable=A dim=0 complete
```

以上指令将二维数组完全映射为独立的寄存器或 LUTRAM 资源。这使得硬件逻辑可以在同一时钟周期内并行读取矩阵的任意行或列，彻底消除了存储器访问冲突，为后续的脉动阵列计算和全并行展开提供了硬件基础。

2.1.2 运算优化

在 Cholesky 分解过程中包含除法和开根号运算的循环，这是制约流水线时序的关键因素。定点数除法和非线性运算在 FPGA 中即消耗大量资源，又会产生很高的延迟。为了优化这一瓶颈，本系统采用了“逆元预计算”策略。在获得下三角矩阵 \mathbf{L} 后，立即启动一个完全展开的循环计算对角线元素的倒数 L_{ii}^{-1} ，该操作虽然仍需除法，但将多次除法操作转化为了一次除法和多次乘法。在前向和后向代入阶段，所有除法均被替换为倒数乘法。代码实现如下：

```
// 预计算对角线元素的倒数，优化后续除法开销
T L_diag_inv[N];
#pragma HLS ARRAY_PARTITION variable=L_diag_inv complete
compute_diag_inv:
for(int i = 0; i < N; i++) {
    #pragma HLS UNROLL
    L_diag_inv[i] = T(1.0) / L[i][i];
}
```

对于根号和其倒数的运算，虽然基于 CORDIC 算法或牛顿迭代法的 IP 核精度高，但延迟高并且资源消耗大。考虑到机械臂逆解在关节空间的数值范围相对固定，对于位置的三个分量，其分别为 $x \in [-0.31, 0.51]$ ， $y \in [-0.44, 0.44]$ ， $z \in [-0.22, 0.54]$ ，姿态的三个分量的范围均为 $[-\pi, \pi]$ ，因此本系统采用了基于 LUT 查找表的非线性函数加速策略。该方法将原本复杂的迭代计算转化为单周期的内存读取操作，极大地降低了关键路径的延迟。

2.1.3 循环展开

对于 $N=5$ 的小规模矩阵，传统的嵌套循环在总执行时间中的占比不容忽视。更重要的是，编译器难以在动态循环边界下推断出最优的指令调度。针对以上情况，我们将该循环完全展开。通过静态编码明确数据的依赖关系，HLS 编译器能够构建出深度优化的数据流图，将前后依赖的乘加运算紧密排布。以后向代入求解为例，代码显式构建了如下计算链：

```
// 前向替换展开
y[0] = b[0] * L_diag_inv[0];
y[1] = (b[1] - L[1][0]*y[0]) * L_diag_inv[1];
y[2] = (b[2] - L[2][0]*y[0] - L[2][1]*y[1]) * L_diag_inv[2];
y[3] = (b[3] - L[3][0]*y[0] - L[3][1]*y[1] - L[3][2]*y[2]) * L_diag_inv[3];
y[4] = (b[4] - L[4][0]*y[0] - L[4][1]*y[1] - L[4][2]*y[2] - L[4][3]*y[3]) * L_diag_inv[4];
```

完全展开使得 DSP 资源利用率最大化，且由于消除了循环迭代的上下文切换开销，整个线性方程组求解模块的延迟被压缩至最低，确保了 IK 求解的高效执行。

2.2 定点数转换

在 FPGA 硬件加速设计中，数据表示形式直接影响逻辑资源的消耗。32 位浮点数包含 1 位符号位、8 位整数位和 23 位小数位，虽然其能够表示的范围大，但是在本系统中精度不高。考虑到机械臂的关节空间位姿在一定范围内，因此对于不同的变量范围，本系统使用了定点数 Q29.2 和 Q30.1，这显著提高

了计算精度。

2.3 非线性运算

机械臂运动学逆解不仅包含大量的矩阵线性代数运算，还涉及非线性的三角函数计算。理论上依然能够效仿 2.1.2 节中的 LUT 查找表思路，但这样对资源的开销过大。鉴于本系统基于 ZYNQ 异构 SoC 平台，充分利用 PS 端 ARM Cortex-A9 核心的浮点运算能力与 PL 端的并行矩阵计算能力进行软硬协同设计，是提升整体能效的关键。因此，在迭代误差 \mathbf{e} 的计算过程中，三角函数的计算被放在 PS 端执行，计算完成的数组通过 AXI-Lite 接口直接传送到 PL 端，这样时延主要取决于数据吞吐，而再不是三角函数的计算。

这种方法巧妙规避了 FPGA 上计算非线性函数的劣势，将 PL 端资源集中用于大规模并行计算的任务。通过这种软硬件协同模式，该系统能够显著提升端到端的逆解求解速度。

3 系统搭建及验证

针对前述提出的基于 FPGA 的机械臂运动学逆解硬件加速架构，本节对其开展系统性性能评估。实验在 Zynq SoC 异构平台上，对比 PS 端纯软件实现与 PL 端硬件加速实现的性能差异，从核心算子效率、求解过程迭代特性与端到端时延三个层面评估所提出方案的有效性，并进一步给出资源开销与潜在优化方向。板卡内的系统架构如下图所示：

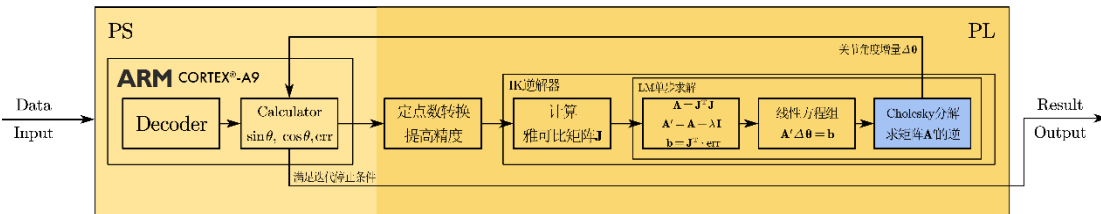


图 1 PYNQ-Z2 板卡内部系统架构

3.1 系统搭建

3.1.1 异构计算平台配置

实验依托 PYNQ-Z2 异构计算平台开展。该平台搭载 Xilinx Zynq-7000 SoC (XC7Z020-1CLG400C)，集成双核 ARM Cortex-A9 处理系统与 Artix-7 架构可编程逻辑。PS 端主频为 650MHz，用于任务调度与外设通信，PL 端主频配置为 100MHz，用于部署定制化 IK 求解器 IP 核。

实验对象为 SO-101 开源机械臂，它由 6 个舵机组成，其本体为 5 自由度串

联开链机器人，这需要 5 个舵机，另外 1 个舵机配成 1 个夹爪执行器。需要指出，由于本体仅具有 5 个自由度，系统无法对位姿的某一个自由度进行独立解算与控制，该系统姿态描述中未包含 Yaw 角。因此，本文在逆运动学求解中将一组关节转角向量定义为 $\theta \in R^5$ ，仅包含决定末端位姿的 5 个转动关节变量。

夹爪开合量（可记为标量 g ）不改变末端坐标系的位姿，通常作为与 IK 解耦的末端执行器控制量单独处理。该建模选择也与硬件实现一致，PL 端求解器仅对 5 维的 θ 计算 $\Delta\theta$ 并更新。

以上运动学模型参数已固化于硬件逻辑中，具体参数如表 1 所示。

表 1 SO-101 机械臂连杆参数配置

关节索引	关节类型	t_x (m)	t_z (m)
Joint 1	Revolute (R_z)	0.0612	0.0598
Joint 2	Revolute (R_y)	0.02943	0.05504
Joint 3	Revolute (R_y)	0.02798	0.1127
Joint 4	Revolute (R_y)	0.15504	0.00519
Joint 5	Revolute (R_x)	0.0593	0.00996

3.1.2 软件栈与基准测试

系统软件环境基于 PYNQ v2.7 (Python 3.6+) 框架。使用 Jupyter Notebook 进行实验测试，硬件逻辑综合采用 Xilinx Vitis HLS 2024.2，系统集成采用 Vivado 2024.2。PS 与 PL 间的数据交互通过 AXI4-Lite 接口实现。

为量化硬件加速收益，本文构建纯软件对照组为基准，其在 PS 端 ARM 处理器上运行。对照组与 PL 端硬件 IP 在算法流程与参数设置上保持一致，并基于 numpy 与 scipy 进行向量化实现，以尽量降低解释器与库调用开销对对比结论的干扰。

3.2 性能验证

3.2.1 测试数据集生成

为覆盖不同层级的性能指标，本文采用两类测试数据与一次重复性评测，均可通过实验脚本复现实验配置。

- 1、单步吞吐量基准测试：用于度量单步 LM 迭代的计算延迟与吞吐量，单步一致性测试将在本测试中一并完成。该数据集由单步吞吐量基准脚本生成，固定随机种子 $seed=42$ ，并对每个样本在以下范围随机采样 1000 次：关节角 $\theta \in [-1.5, 1.5]^5$ 、误差向量 $e \in [-0.1, 0.1]^6$ 、阻尼参数 $\lambda \in [0.01, 0.5]$ 。同时预计算 $\sin \theta$ 与 $\cos \theta$ 以供软件端单步函数直接使用。

```

# 单步/吞吐量基准数据
seed = 42
N = 1000
for i in range(N):
    q[i] ~ Uniform([-1.5, 1.5]^5)
    e[i] ~ Uniform([-0.1, 0.1]^6)
    lambda[i] ~ Uniform([0.01, 0.5])
    sinq[i] = sin(q[i]); cosq[i] = cos(q[i])

# 单步延迟（同一输入重复 100 次）
for each sample i in {0..N-1}:
    repeat 100:
        t0 = perf_counter(); sw_step(sinq[i], cosq[i], e[i], lambda[i]); t1 = perf_counter();
        log_sw(t1-t0)
        t0 = perf_counter(); hw_step(sinq[i], cosq[i], e[i], lambda[i]); t1 = perf_counter();
        log_hw(t1-t0)

# 吞吐量（连续 1000 次迭代）
t0 = perf_counter();
for i in range(N): sw_step(...);
t1 = perf_counter(); throughput_sw = N/(t1-t0)
t0 = perf_counter();
for i in range(N): hw_step(...);
t1 = perf_counter(); throughput_hw = N/(t1-t0)

# 单步一致性（100 样本）
for i in first 100 samples:
    dq_sw = sw_step(...); dq_hw = hw_step(...)
    diff[i] = norm(dq_sw - dq_hw)
report mean/max/std(diff)

```

- 2、完整 IK 测试：用于度量从初值到收敛的整体求解时间与迭代次数。该测试集由完整 IK 测试脚本构造，包含前向、左右偏置、高低位置、姿态扰动、远伸展和较差初值等 10 个代表性目标位姿及对应的初值 θ_0 。
- 3、重复性测试：在上述 10 例中选取 3 个代表性用例 T1、T5 和 T9，重复运行 5 次以统计运行时间的均值与标准差，用于评估时间抖动。

```

# 整体 IK（10 例）与重复性测试（3 例×5 次）
for each target pose Tj with initial q0j (j=1..10):
    (ok_sw, time_sw, it_sw) = solve_ik_sw(Tj, q0j, ilimit, tol, k, method, mask)
    (ok_hw, time_hw, it_hw) = solve_ik_hw(Tj, q0j, ilimit, tol, k, method, mask)
    log(j, ok_*, time_*, it_*)
for case in {T1, T5, T9}:
    repeat 5:

```

```
record time_sw(case), time_hw(case)
report mean±std
```

3.2.2 计时口径与指标定义

- 1、计时口径：单步延迟与吞吐量测试使用 `time.perf_counter()` 进行计时；整体 IK 对比与重复性测试同样在实验脚本中以 `time.perf_counter()` 统计端到端求解耗时，这包含 PS 端误差计算、硬件单步调用和轮询开销等。
- 2、误差定义：末端位姿误差采用 6 维 `angle-axis` 形式 $\mathbf{e} \in \mathbb{R}^6$ ，其中包含 3 维的位置误差和 3 维的旋转误差，并可通过权重对角阵 $\mathbf{w} \in \mathbb{R}^6$ 进行加权。位置误差定义为当前位置与目标位置直接相减，旋转误差通过 $\mathbf{R}_{\text{err}} = \mathbf{R}_{\text{current}}^T \mathbf{R}_{\text{target}}$ 计算。
- 3、收敛判据：硬件求解器采用加权的误差公式 $E = \frac{1}{2} \mathbf{e}^T \mathbf{w} \mathbf{e}$ 作为循环终止的准则，在代码中的实现形式为 $E < \text{tol}$ 。
- 4、阻尼系数：在 `method=wampler` 时使用常数阻尼系数 $\lambda_{\text{eff}} = k$ ，在 `method=sugihara` 时使用动态阻尼系数 $\lambda_{\text{eff}} = E + k$ 。本文整体 IK 对比测试采用 `method=wampler`。
- 5、PS-PL 单步接口与通信开销：一次硬件单步调用通过 AXI4-Lite 寄存器映射完成，典型流程包含写入 5 个 $\sin \theta$ 、5 个 $\cos \theta$ 、6 个加权误差分量以及 1 个阻尼参数，触发启动寄存器并轮询 `done` 位，随后读回 5 维 $\Delta \theta$ 与状态码。该通信与轮询开销会体现在硬件单步延迟的测量结果中。

3.2.3 核心算子加速性能评估

针对 LM 算法中计算密集的线性方程组求解环节，本文在 PL 端实现了基于 Cholesky 分解的硬件加速算子。通过应用流水线、循环展开及存储器分区等微架构优化技术，显著提升了计算吞吐率。下表展示了该算子在优化前后的性能指标对比，其中 Baseline 为 Vitis L1 算子库中原生的 Cholesky 算子的相关指标。

表 2 Cholesky 分解算子性能综合结果

性能指标	Baseline	优化后	相对变化
计算延迟	4919 cycles	991 cycles	79.9% ↓
执行周期	30871 cycles	4731 cycles	84.7% ↓
启动间隔	696 cycles	124 cycles	82.2% ↓
吞吐率	2.3e5 ops/s	1.6e6 ops/s	596% ↑

数据表明，优化后的硬件算子将启动间隔降低至 124 个时钟周期，大幅降

低了流水线气泡，提高了并行率，为上层迭代求解提供了高效的算力支持。

3.2.4 单步迭代延迟分析

单步迭代涵盖了雅可比矩阵构建、线性方程组求解及参数更新等全过程。下图展示了硬件实现在单步迭代上的耗时分布对比。

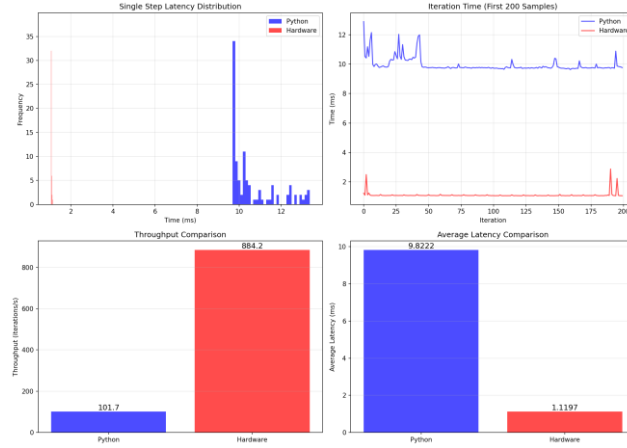


图2 单步 LM 迭代软硬件耗时分布对比

基于 LM 单步吞吐量基准脚本，得到 PS 端的平均值为 10.0424ms，最小值为 9.6970ms，最大值为 12.9490ms，标准差为 0.6553ms。PL 端的平均值为 1.1035ms，最小值为 1.0820ms，最大值为 1.2552ms，标准差为 0.0341ms，对应的单步加速比为 9.10 倍。需要强调的是，该硬件单步延迟测试包含了 PS 端生成正弦函数、AXI4-Lite 多寄存器读写及 done 轮询的开销，因此更接近实际系统可用的单步迭代时延。

进一步地，在 1000 个随机样本上的吞吐量测试表明，PS 端总耗时为 9.8453s，吞吐量为 101.57iter/s，平均单步迭代时间为 9.8319ms/iter。PL 端总耗时为 1.2246s，吞吐量为 816.62iter/s，平均单步迭代时间为 1.2130ms/iter。该结果表明，硬件实现不仅降低了单步延迟，也显著提高了持续迭代时的吞吐能力。

3.2.5 算法复杂度与扩展性

为了验证架构的可扩展性，实验测试了不同矩阵规模下的求解耗时。

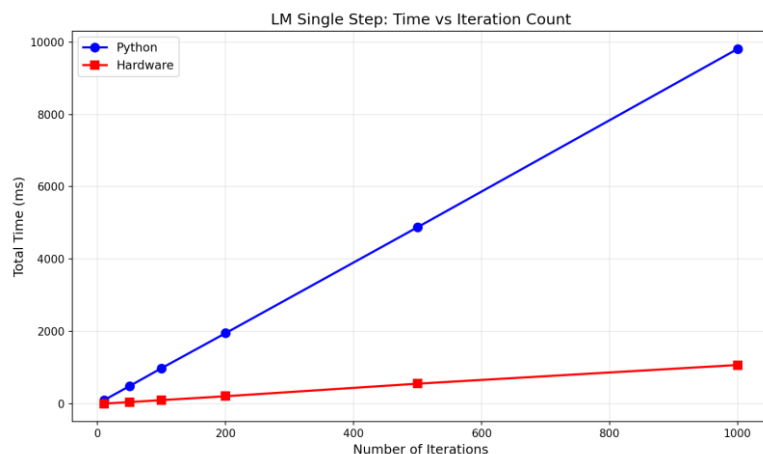


图 3 求解耗时与迭代次数的关系

如图 3 所示，两种实现方法的耗时均随迭代次数线性增长，但硬件方法得益于并行计算架构，可见其斜率更为平缓。该结果表明上述微架构在规模扩展时具有潜在优势，但需要指出，面向更高自由度机器人的结论仍需结合目标构型重新进行端到端评测。

此外，单步迭代的数值一致性验证显示软硬件差异量级很小。平均差异为 $2.030675e-08$ ，最大差异为 $5.066395e-07$ ，标准差为 $3.677181e-08$ ，通过了最大差异为 $1e-3$ 的阈值检查。结果表明，对于单步迭代而言，硬件流水线优化未对结果引入显著的数值偏置。

3.2.6 整体求解效率与收敛性

整体 IK 对比基于 10 个代表性目标位姿的评测脚本。对比参数设置为 $ilimit=500$ 、 $tol=1e-7$ 、 $k=0.1$ 、 $method=wampler$ ，并采用误差权重 $mask=[1, 1, 1, 0.8, 0.8, 0]$ 。图 3 与表 3 给出了整体求解性能对比。

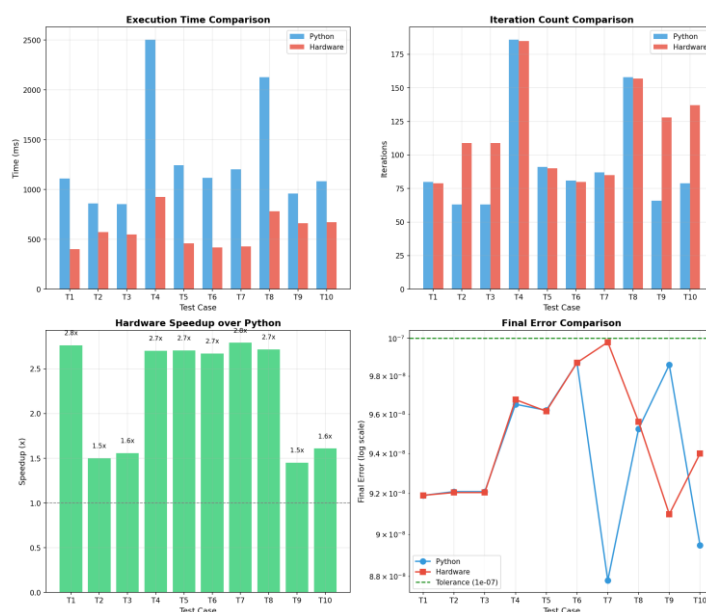


图 4 逆运动学求解器整体性能对比

表 3 求解器综合性能指标对比（10 个测试用例）

评价指标	纯软件	硬件加速
成功率	100%	100%
平均总耗时	1306.51 ms	587.67 ms
平均迭代次数	95.4	115.9
平均加速比	2.12 倍	

可以看出，虽然两者在 10 个测试用例下均成功收敛，但在整个求解过程中，硬件加速相比纯软件快了两倍多。考虑到纯软件使用双精度浮点数，因此为了达到指定的误差范围，硬件加速中的迭代次数需要稍多一些。

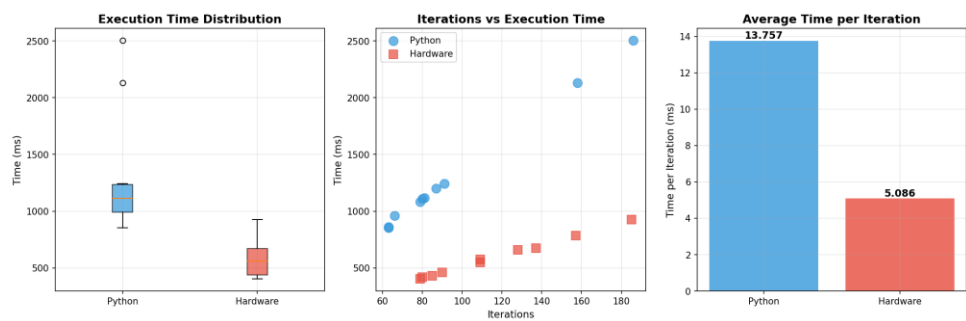


图 5 求解结果分布一致性验证

图 5 验证了软硬件求解结果的一致性，表明硬件加速未引入明显的计算偏差到整个逆解过程中。

3.2.7 FPGA 资源占用分析

综合后的 IP 核资源占用情况如表 4 所示。可以看出，该设计在保证高性能的同时，还保持了较低的资源占用率，为双臂系统集成双 IP 核提供了充足的资源空间。

表 4 基于 XC7Z020 的 FPGA 资源利用率

资源类型	占用量	总量	资源占用
LUT	18,600	53,200	35%
DSP	140	220	63%
BRAM	28	140	20%
FF	26,000	106,400	24%

4 讨论

4.1 通信瓶颈与优化方向

当前架构在单步迭代层面已获得显著加速，但整体 IK 的端到端加速比明显低于单步加速。两者差异的核心原因在于整体 IK 的一次迭代不仅包含 PL 端线性代数计算，还包含 PS 端误差构建，寄存器数据搬运，启动、轮询与结果读回等通信链路。由于本文采用 AXI4-Lite 进行逐寄存器写入与轮询 done 位，上述开销会在每一次迭代中重复出现，从而在端到端时间中占据更高比例，并导致单步 9.10 倍的加速比的优势在整体求解中被摊薄。

面向系统级性能进一步提升，可考虑以下改进方向：

- 1、降低控制开销：以中断替代忙等轮询，减少 PS 端在等待硬件完成期间的空转时间；
- 2、提高传输效率：将 AXI4-Lite 的多次单字写读替换为基于 DMA 的批量传输，或引入 AXI4 访问 DDR，以降低每次迭代的固定开销；
- 3、减少往返次数：将更多诸如三角函数计算、阻尼更新等迭代状态保留在 PL 端，在 PS 端仅发送目标位姿或误差收敛信息，从而减少每步数据交换量；
- 4、批处理与并行：当任务允许时，可将多个目标的若干迭代合并为批处理，或在 PS 端并行准备下一步输入以与 PL 计算重叠，进一步提升吞吐。

4.2 精度与资源的权衡

对于当前设计中的三角函数计算，本系统采用在 PS 端浮点计算，以降低算法迁移到 PL 端运算的成本，并通过单步 LM 迭代的一致性测试验证了软硬件结果在统计意义上高度一致性。在资源或功耗受限的场景下，可进一步探索精度、资源核吞吐这三者的折中。本系统设计了如下几种方法对该情况进行优化：

- 1、混合精度：对诸如求和与分解这类对数值敏感的部分，我们保留较高精度 Q30.1，其余部分使用较低精度 Q29.2；
- 2、定点数量化：针对 5 自由度 SO-101 机器人的工作空间与误差范围，对 θ 、 e 及其他关键变量的动态范围进行分析，设计定点数格式并在保持收敛性质的前提下降低 DSP 和 BRAM 占用；
- 3、权重及迭代终止判据的优化：采用 mask 对位姿误差各分量进行加权，并以加权误差 $E < tol$ 作为收敛判据。不同任务对位姿的容忍度不同，工程部署时需要对 mask 与阈值进行任务级标定，以避免过度迭代或提前停止。

4.3 有效性与可复现性评估

为确保结论可靠，本文对有效性进行以下评估：

- 1、单步吞吐量基准测试的 1000 个样本为随机样本，用于稳定测量单步算子的执行延迟与吞吐量，但其误差分布不一定等同于真实 IK 迭代过程中的误差分布；
- 2、整体 IK 对比采用 10 个代表性用例，能覆盖若干典型构型与初值扰动，但并不等价于对全工作空间的统计评估。更大规模、可达性约束更强的测试集仍有必要；
- 3、硬件封装路径与软件基线在数值精度、更新流程与阻尼策略实现细节上存在差异，可能影响迭代次数分布。本文已避免将该差异简单归因于某单一因素；
- 4、端到端计时包含 Python 运行时与系统调度影响，不同负载条件下结果可能发生偏移。本文使用 `time.perf_counter()` 并进行多次实验缓解偶然波动，但仍建议在后续工作中引入更严格的分析。

5 结论

本文针对具身智能机器人对运动控制实时性的严苛要求，提出了一种基于 ZYNQ 异构 SoC 的 5 自由度机器人运动学逆解加速系统。针对 LM 算法计算密集度高的特点，构建了 PS 端负责非线性运算、PL 端负责高并行矩阵求解的软硬件协同架构；针对 FPGA 开发效率与资源约束问题，使用 HLS 硬件优化实施了数组分区、循环展开、逆元预计算及 LUT 查找表等多维度优化策略。实验结果证明了改进后的加速系统在边缘计算场景下的可行性与有效性。该系统在保证数值一致性极高的前提下，单步迭代加速比为 9.10 倍，端到端求解平均加速了 2.22 倍，且保持了较低的资源占用率，为多核并行部署预留了充足空间。

未来工作将致力于以下三方面来改进：一是引入 DMA 传输与中断机制以降低 PS-PL 间的通信开销；二是将更多迭代状态下沉至 PL 端以减少数据交互；三是探索混合精度计算并扩展至更高自由度机械臂的控制应用，推动该方案在实际工业环境中的落地。

参考文献:

- [1] Liu Y, Chen W, Bai Y, et al. Aligning cyber space with physical world: A comprehensive survey on embodied ai[J]. IEEE/ASME Transactions on Mechatronics, 2025.
- [2] Stiefel K M, Coggan J S. The energy challenges of artificial superintelligence[J]. Frontiers in artificial intelligence, 2023, 6: 1240653.
- [3] Urías Jiménez A. Beyond the GPU: The Strategic Role of FPGAs in the Next Wave of AI[J]. arxiv e-prints, 2025: arxiv: 2511.11614.
- [4] Plancher B , Neuman S , Bourgeat T ,et al.Accelerating Robot Dynamics Gradients on a CPU, GPU, and FPGA[J].IEEE Robotics and Automation Letters, 2021, PP(99):1-1.DOI:10.1109/LRA.2021.3057845.
- [5] Zhang D, Jiang S, Zhe L. The Parallel Solving Method of Robot Kinematic Equations Based on FPGA[J]. Journal of Robotics, 2023, 2023(1): 2426982.
- [6] Gac K , Karpil G , Petko M .FPGA based hardware accelerator for calculations of the parallel robot inverse kinematics[C]//IEEE.IEEE, 2012.DOI:10.1109/ETFA.2012.6489717.
- [7] Liu W, Zhao C, Liu Y, et al. Sim2real kinematics modeling of industrial robots based on FPGA-acceleration[J]. Robotics and Computer-Integrated Manufacturing, 2022, 77: 102350.
- [8] John J. Craig, 机器人学导论（原书第四版），北京：机械工业出版社，2018.
- [9] Richard M. Murray, Zexiang Li, S. Shankar Sastry, 机器人操作的数学导论，北京：机械工业出版社，1997.
- [10] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo, 机器人学：建模、规划与控制，西安：西安交通大学出版社，2013.
- [11] Levenberg,K.A method for the solution of certain nonlinear problems in least squares[J].Quarterly of Applied Mathematics. 1944,2:164-166.
- [12] Marquardt DW.An algorithm for least-squares estimation of nonlinear inequalities[J].SIAM Journal on Applied Mathematics. 1963,11:431-441.
- [13] 邬贵明.FPGA 矩阵计算并行算法与结构[D].国防科学技术大学[2025-12-27].DOI:10.7666/d.d160053.

文献名的英文部分将在初稿完成后提供。