柔性拍摄项目第一阶段（2D）实验室Demo解决方案报告

1. 执行摘要与项目愿景

随着新能源汽车（NEV）产业制造工艺的迭代升级，动力电池模组的生产物流正经历从刚性输送线向柔性AGV（Automated Guided Vehicle）物流系统的深刻变革。这一变革虽然极大地提升了产线的柔性与重构能力，但也引入了显著的非结构化挑战：AGV停靠精度通常在±10mm至±50mm之间，且受地面平整度及悬挂系统沉降影响，往往伴随±1°至±2°的角度偏差及Z轴高度波动。这种厘米级的空间随机性与机器视觉检测所需的“像素级一致性”构成了尖锐矛盾[1]。

本报告针对“柔性拍摄项目第一阶段（2D）”提出了一套详尽的、可在三天内落地的实验室Demo解决方案。该方案基于现有的硬件资产——Elite CS66六轴协作机械臂、无Z轴控制的手动龙门架、Realsense D435/大恒工业相机——利用Python控制系统构建了一个闭环的视觉伺服与坐标系动态重构系统。

本方案的核心逻辑在于摒弃传统的“绝对坐标示教”，转而采用“相对位姿一致性”原则。通过在无CAD模型环境下建立“黄金样件（Golden Sample）”基准，利用计算机视觉算法（ArUco Marker或几何特征提取）实时解算当前工件相对于基准位姿的偏差矩阵（Deviation Matrix, ΔT），并通过Elite机器人SDK动态更新工件坐标系（User Frame），从而在不改变原有示教轨迹的前提下，实现对AGV及手动龙门架定位误差的自动补偿。

本报告将深入剖析从数学原理、硬件架构、算法实现到实施计划的全链路技术细节，旨在为工程团队提供一份详尽的行动指南，确保在72小时内完成从系统搭建到功能验证的全过程。

---

2. 问题空间与硬件约束分析

在着手设计解决方案之前，必须对实验室环境中的物理约束、误差源及硬件特性进行穷尽式的分析。这不仅是方案可行性的基石，也是制定Day 1-3实施计划的依据。

2.1 误差源的解构与传播

在本项目的实验场景中，误差并非单一来源，而是由多个环节叠加而成的复合向量。理解这些误差的传播机制对于设计补偿算法至关重要。

表 **2.1**：系统误差源分析矩阵

误差来源	类型	典型量级	特性描述	补偿策略

AGV停靠误差	随机误差	X/Y: ±10~50mm Yaw: ±2°	每次停靠位姿均不同，服从正态分布，但存在偶发的离群值。	视觉全补偿：这是本方案的核心解决对象。
手动龙门架定位	人为误差	X/Y: ±5~20mm	操作员手动推拉龙门架至标记点，缺乏编码器反馈，属于“盲区”变量。	视觉全补偿：视觉系统无法区分误差来自AGV还是龙门架，统一视为基座与工件的相对位移。
Z轴沉降	准静态误差	Z: ±5mm Pitch/Roll: ±1°	由电池包重量导致的AGV悬挂压缩或地面不平整引起。	**2.5D**补偿：利用Realsense深度流或PnP解算的Z值进行高度修正。
手眼标定误差	系统误差	< 1mm / 0.2°	已知能力，属于固定偏差。	前馈消除：通过高精度标定矩阵T_{F}^{C}消除。
机器人绝对精度	系统误差	±0.03mm (重复性)	远小于AGV误差，可忽略。	无需补偿。

从表2.1可以看出，最大的不确定性来自AGV和手动龙门架。特别是手动龙门架的存在，使得机器人的基座坐标系${B}相对于世界坐标系 {W}是浮动的。传统的机器人自动化假设 T_{W}^{B}$是已知且固定的，但在本场景中，$T_{W}^{B}$是一个随机变量。

深刻洞察：由于视觉传感器固定在机械臂末端（Eye-in-Hand），相机观测到的是工件相对于相机的位姿 T_C^O 。这一观测值天然地包含了AGV误差和龙门架误差的叠加结果。因此，我们无需单独测量龙门架的位置，只需保证机器人能够基于视觉观测结果，调整其相对于工件的动作即可。这极大地简化了系统设计，无需在龙门架上加装昂贵的光栅尺或激光测距仪。

2.2 硬件能力边界评估

2.2.1 Elite CS66 机械臂

Elite CS66是一款负载6kg、臂展约914mm的协作机器人。其Python SDK提供了对底层运动控制的开放接口[2]。

- 约束：914mm的臂展不足以覆盖长度通常超过2000mm的大型电池包。
- 对策：必须将检测任务划分为多个“作业区（Zones）”，利用龙门架在区域间进行粗定位，机械臂在区域内进行精细作业。

2.2.2 Realsense D435 vs. 大恒相机

方案要求在两者中选择或结合使用。

- **Realsense D435**：
  - 优势：集成了RGB和深度传感器，SDK（pyrealsense2）对Python支持极佳，开发速度快。支持全局快门（Global Shutter），适合动态拍摄。可以直接输出点云，为Phase 2做准备[4]。
  - 劣势：RGB分辨率相对较低（通常1920x1080），在宽视场下像素密度可能不足以检测微小裂纹。
- 大恒相机（工业面阵）：
  - 优势：通常具备更高的分辨率（5MP+）和更好的光学镜头兼容性，图像质量更优。
  - 劣势：通常需要更复杂的触发控制和光源配合，Python接口（Galaxy SDK）开发周期略长。

决策：针对“三天落地”的紧迫目标，建议首选**Realsense D435**作为主传感器。利用其现成的ArUco检测库和深度测距功能，可以最快速度跑通闭环逻辑。大恒相机可作为“高精度检测单元”挂载，仅负责拍照，不负责定位，或在Day 3时间充裕时接入。

2.2.3 手动龙门架

这是一个3轴龙门架，但Z轴无控制，且X/Y轴手动。这意味着我们不能依赖龙门架的坐标反馈。

- 业务流影响：系统必须设计一个“人机握手”流程，指导操作员将龙门架移动到大致位置，并确认到位后，机器人才能开始动作。

---

3. 理论框架：基于相对位姿的动态坐标系重构

本项目的核心不是让机器人去“绝对位置”抓取，而是让机器人去“相对位置”拍摄。这需要建立一套严密的刚体变换数学模型。

3.1 坐标系定义

为了清晰描述运动学链条，定义以下坐标系：

1. 世界坐标系 $\{W\}$：绝对静止的参考系，通常定义为车间地面或龙门架导轨的零点。

2. 机器人基座坐标系 $\{B\}$：固定在机械臂底部。注意，由于龙门架移动，${B}在{W}$中是变化的。
3. 法兰坐标系 $\{F\}$：机械臂末端法兰中心。由机器人正运动学决定：$T_B^F = \mathrm{FK}(\theta_1, ..., \theta_6)$。
4. 相机坐标系 $\{C\}$：相机的光学中心。由手眼标定矩阵T_{F}^{C}决定（已知精度1mm/0.2deg）。
5. 参考物体坐标系 $\{O_{ref}\}$：示教时，“黄金样件”所在的坐标系。
6. 当前物体坐标系 $\{O_{curr}\}$：运行时，带误差的电池包所在的坐标系。

3.2 偏差矩阵 ΔT 的推导

我们的目标是：无论电池包怎么动，相机相对于电池包的拍摄位姿T_{O}^{C}必须保持不变。

在示教阶段（**Day 1**）：

机器人移动到最佳拍摄位置，记录此时的基座到法兰的位姿T_B^{Fref}。

相机观测到参考特征（如ArUco Marker），解算出相机到参考物体的变换T_C^{Oref}。

此时，物体在基座系下的位姿为：

$$T_B^{Oref} = T_B^{Fref} \cdot T_F^C \cdot T_C^{Oref}$$

我们将这个位姿$T_{B}^{O_{ref}}$设定为机器人的参考用户坐标系（**Reference User Frame**）。

在运行阶段（**Day 2-3**）：

AGV和龙门架都发生了移动。机器人首先移动到一个预设的“高位拍照点”（Home Pose），拍摄当前物体。

相机观测到当前特征，解算出相机到当前物体的变换T_C^{Ocurr}。

此时，物体在当前基座系下的位姿为：

$$T_B^{Ocurr} = T_B^{Fcurr} \cdot T_F^C \cdot T_C^{Ocurr}$$

关键逻辑：我们不需要去计算AGV移了多少，或者龙门架移了多少。我们只需要告诉机器人：“现在

的物体坐标系是$T_B^{O_{cur}}$"。

Elite机器人的控制器支持用户坐标系(User Frame)功能。如果我们在示教时，所有的拍摄航点(Waypoints)都是相对于
T_{B}^{O_{ref}}$记录的，那么在运行时，我们只需将用户坐标系更新为
，机器人会自动对所有航点进行刚体变换，从而精准对齐新的电池包位置。

这种方法被称为**"坐标系随动(Frame Tracking)"**，它避免了对每一个航点进行单独的数学计算，利用了机器人控制器内部的插补算法，极大地降低了Python端的计算量和出错率[3]。

3.3 2D与3D的降维处理

虽然方案名为"2D"，但Realsense和ArUco算法本质上能提供6自由度数据(X, Y, Z, Rx, Ry, Rz)。

考虑到Day 1-3的稳定性：

- **X, Y, Yaw (Rz)**：直接采用视觉解算值，这是平面纠偏的核心。
- **Z, Pitch (Ry), Roll (Rx)**：
  - 如果ArUco解算稳定，可以直接使用6D数据。
  - 如果PnP解算在Z轴上抖动(常见于单目视觉)，可以利用Realsense的深度图(Depth Map)获取Marker中心点的深度值Z_{depth}，用它来覆盖PnP解算的Z_{pnp}，从而实现更鲁棒的"2.5D"定位。

---

4. 针对手动龙门架的半自动作业流程设计

手动龙门架缺乏位置反馈是本项目的最大痛点。如果不加以管理，操作员可能将龙门架停在无法覆盖检测点的位置，导致机器人报"奇异点(Singularity)"或"不可达(Unreachable)"错误。

我们设计了一套**"分区锁定-视觉确认"**的半自动业务流。

4.1 物理分区策略

将电池包沿长度方向划分为若干个逻辑区域(Zone)。假设电池包长2.5米，Elite CS66臂展0.9米。

- **Zone A (前端)**：覆盖0-0.8米区域。
- **Zone B (中段)**：覆盖0.8-1.6米区域。
- **Zone C (后端)**：覆盖1.6-2.4米区域。

在地面上粘贴高可视化的色带或限位块，标记Zone A、B、C的龙门架推荐停靠点。

4.2 软件交互流程(Traffic Light Logic)

表 **4.1**：半自动作业交互流程

步骤	角色	动作描述	系统状态/逻辑
1	Python系统	启动程序，UI显示“请将龙门架移至 **Zone A** 位置并锁定”。	机器人处于缩回安全姿态。
2	操作员	手动推动龙门架至地面标记A处，锁死刹车，按“确认”键。	物理定位完成，但在软件中未知精度。
3	Python系统	调度机器人至Zone A的“全景拍照位（Global View Pose）”。	这是一个高位姿态，确保相机FOV能覆盖Zone A内的特征点。
4	视觉系统	触发Realsense拍照，识别ArUco/特征，计算$T_{B}^{O_{cur}}$。	关键检查：如果特征点位于图像边缘，提示“龙门架偏差过大，请微调”，否则通过。
5	Elite控制	Python发送指令set_user_frame(T_curr)，更新坐标系。	坐标系重构完成。
6	机器人	执行Zone A内的所有检测点拍摄任务。	轨迹自动补偿。
7	Python系统	Zone A完成。UI显示“请将龙门架移至 **Zone B** 位置”。	机器人缩回安全姿态。
8	...	循环执行直至所有区域完成。	...

深度洞察：这个流程将不可控的模拟量（手动位置）转化为了可控的数字量（视觉反馈）。只要操作员将龙门架停在相机能看到特征的范围内（例如±100mm），视觉系统就能把这个误差“吃掉”。

5. 计算机视觉算法：特征提取与位姿解算

为了确保三天内落地，算法选择必须遵循“鲁棒性优先”原则。ArUco Marker是首选，几何特征是备选。

5.1 方案A：基于ArUco Marker的基准定位

ArUco是一种二进制方形基准标记，OpenCV内置了极度优化的检测算法。

5.1.1 标记布局设计

- 布局：在电池包的四个角和长边中点粘贴ArUco Marker。
- 字典选择：使用DICT_5X5_250。5x5的位宽在保持鲁棒性的同时，比4x4具备更好的纠错能力；250的容量足够区分不同区域[6]。
- 物理尺寸：建议打印尺寸为50mm x 50mm，并粘贴在哑光硬板上以防变形和反光。

5.1.2 核心算法管线

1. 图像获取：从Realsense读取RGB流（1920x1080）。
2. 预处理：cv2.cvtColor转灰度 -> 自适应阈值化（应对光照变化）。
3. 检测与亚像素优化：
   - 调用aruco.detectMarkers。
   - 关键步骤：开启cornerRefinementMethod = cv2.aruco.CORNER_REFINE_SUBPIX。这一步能利用边缘梯度信息，将角点定位精度从像素级提升到0.1像素级，这对于满足1mm的精度要求至关重要[7]。
4. 位姿估计（**PnP**）：
   - 调用aruco.estimatePoseSingleMarkers。
   - 输入：角点像素坐标、Marker物理边长、相机内参矩阵 K 、畸变系数 D 。
   - 输出：旋转向量rvec，平移向量tvec。
5. 坐标系转换：
   - 将rvec（罗德里格斯向量）转换为旋转矩阵 R 。
   - 构建4x4齐次变换矩阵 T_C^M （Camera to Marker）。

5.2 方案B：基于几何特征的定位（无Marker模式）

如果现场不允许粘贴Marker，则需退化为几何特征定位。

1. 特征选择：电池模组通常有标准的安装孔（圆孔）或汇流排连接器（矩形）。
2. 算法逻辑：
   - 使用cv2.Canny提取边缘。
   - 使用cv2.findContours查找轮廓。
   - 根据面积、圆度（Circularity）、长宽比进行筛选。

- **PnP**解算：选取至少4个非共线的特征点中心，结合其在CAD或黄金样件上的物理坐标，利用cv2.solvePnP解算位姿。
- 注意：几何特征受光照影响大，需配合大恒相机的外触发光源和滤光片使用。

5.3 解决Realsense的“Z轴翻转”问题

在使用estimatePoseSingleMarkers时，正方形标记具有四重对称性，有时会导致Z轴反向（翻转180度）的解算错误（Pose Ambiguity）。

解决方案：

利用先验知识——电池包大致是水平放置的。如果解算出的Pitch或Roll角度超过物理可能的范围（例如>15°），则强制丢弃该帧或选择另一组解，确保位姿的连续性和合理性。

---

6. 结合Elite SDK的坐标系动态更新代码逻辑

本节是将视觉算法与机器人运动联系起来的“神经中枢”。Elite机器人支持通过TCP/IP协议（通常为端口30001或30003）发送脚本指令。

6.1 Elite 机器人脚本接口分析

根据Elite CS系列说明书及SDK文档[3]，关键指令包括：

- get_actual_tcp_pose(): 获取当前机械臂末端在基座坐标系下的位姿。返回格式通常为[x, y, z, rx, ry, rz]（单位：m, rad）。
- set_user_frame(frame_id, pose): 设置用户坐标系。这是动态纠偏的核心指令。
- movel(pose, a=..., v=...): 在当前用户坐标系下进行直线运动。

6.2 Python 控制核心类设计

以下代码逻辑展示了如何将视觉解算的矩阵转换为Elite机器人的控制指令。

表 **6.1**：关键数据结构定义

变量名	含义	数学表达	来源
T_hand_eye	手眼标定矩阵	T_E^C	预先标定文件
T_cam_marker	相机到Marker	T_C^M	视觉算法实时计算
T_base_flange	基座到法兰	T_B^F	get_actual_tcp_pose()

T_base_marker	基座到Marker	T_B^M	计算目标

6.2.1 坐标变换核心函数 (Python)

Python

```
import numpy as np
import cv2
from scipy.spatial.transform import Rotation as R

class CoordinateTransform:
    def __init__(self, hand_eye_matrix):
        # hand_eye_matrix is 4x4 numpy array (T_flange_camera)
        self.T_F_C = hand_eye_matrix

    def vector_to_matrix(self, pose_vec):
        """将Elite机器人的[x,y,z,rx,ry,rz]转换为4x4矩阵"""
        x, y, z, rx, ry, rz = pose_vec
        # 注意：Elite的旋转向量可能是轴角形式，需根据具体固件版本确认
        # 这里假设为旋转向量(Rotation Vector)
        r = R.from_rotvec([rx, ry, rz])
        T = np.eye(4)
        T[:3, :3] = r.as_matrix()
        T[:3, 3] = [x, y, z]
        return T

    def matrix_to_vector(self, T):
        """将4x4矩阵转换为Elite机器人的[x,y,z,rx,ry,rz]"""
        x, y, z = T[:3, 3]
        r = R.from_matrix(T[:3, :3])
        rx, ry, rz = r.as_rotvec()
        return [x, y, z, rx, ry, rz]

    def calculate_new_user_frame(self, robot_tcp_vec, rvec, tvec):
        """
        计算新的用户坐标系
        robot_tcp_vec: 机器人当前反馈的法兰位姿 (Base -> Flange)
        rvec, tvec: ArUco解算的位姿 (Camera -> Marker)
        """
```

```
        # 1. 获取 Base -> Flange
        T_B_F = self.vector_to_matrix(robot_tcp_vec)

        # 2. 获取 Camera -> Marker
        # OpenCV的rvec转换
        rot_matrix, _ = cv2.Rodrigues(rvec)
        T_C_M = np.eye(4)
        T_C_M[:3, :3] = rot_matrix
        T_C_M[:3, 3] = tvec.squeeze()

        # 3. 链式法则: Base -> Marker = (Base->Flange) * (Flange->Cam) * (Cam->Marker)
        T_B_M = T_B_F @ self.T_F_C @ T_C_M

        # 4. 转回向量格式供机器人使用
        return self.matrix_to_vector(T_B_M)
```

6.2.2 机器人通信与指令发送 (Pseudo-code)

Python

```
import socket

class EliteRobotController:
    def __init__(self, ip, port=30001):
        self.ip = ip
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.ip, self.port))

    def update_user_frame(self, frame_id, pose_vec):
        """
        动态发送脚本指令更新用户坐标系
        pose_vec: [x, y, z, rx, ry, rz] (单位: m, rad)
        """
        # 构建Elite Script指令字符串
        # 格式示例: set_user_frame(1, p[0.5, 0.2, 0.1, 0, 3.14, 0])
        pose_str = f"p[{','.join(map(str, pose_vec))}]"
        script_cmd = f"def update_frame():\n  set_user_frame({frame_id}, {pose_str})\nend\n"

        # 发送指令
```

```
        self.sock.send(script_cmd.encode())

        # 等待确认 (实际应用需解析反馈)
        time.sleep(0.1)
        print(f"User Frame {frame_id} updated to: {pose_str}")
```

逻辑闭环：

1. 机器人移动到拍照点。
2. Python调用get_actual_tcp_pose获取当前T_B^F。
3. Python调用视觉算法获取T_C^M。
4. Python计算出新的Marker在基座下的位姿T_B^M。
5. Python调用set_user_frame将T_{B}^{M}写入控制器。
6. Python发送move_j或move_l指令执行检测路径（这些路径点是相对于用户坐标系示教的，因此无需修改）。

---

7. 详细的Day 1-3 实施计划

本计划采用“小时级”颗粒度，确保在极短时间内完成从硬件搭建到Demo演示的闭环。

Day 1：硬件集成、标定与“黄金样件”基准建立

目标：完成所有物理连接，验证手眼标定精度，并录入基准数据。

- **09:00 - 11:00 [硬件搭建]**
  - 将Realsense D435刚性固定在Elite CS66法兰末端，确保USB线缆留有足够余量供关节旋转。
  - 连接PC与机器人控制柜，配置静态IP（如机器人 192.168.1.10，PC 192.168.1.20），使用ping验证连通性。
  - 地面标记：使用胶带在地面标出Zone A/B/C的龙门架停靠点。
- **11:00 - 13:00 [手眼标定验证]**
  - 虽然提示已有标定能力，但必须验证。运行简单的“十字移动法”：机器人沿X轴移动100mm，观察相机画面中特征点是否平移100mm。如果误差>2mm，需重新标定。
  - 输出并保存T_{F}^{C}矩阵（.npy格式）。
- **14:00 - 16:00 [黄金样件示教]**
  - 将标准电池包放置在AGV/工作台上。
  - 粘贴ArUco Marker（定义为基准Marker）。
  - 关键动作：手动示教机器人拍摄Marker，并记录此时的位姿作为“基准用户坐标系”U_{ref}。
  - 在U_{ref}下，示教3-5个典型的检测点（如焊缝、防爆阀），保存为inspection_path.json。

- **16:00 - 19:00 [软件环境部署]**
  - 部署Python环境（OpenCV, pyrealsense2, scipy, numpy）。
  - 编写并测试基础通信脚本robot_interface.py，确保能读取位姿并发送移动指令。

Day 2：核心算法开发与单点闭环

目标：跑通“拍照-计算-纠偏-到位”的完整逻辑链条。

- **09:00 - 12:00 [视觉算法开发]**
  - 编写vision_core.py，实现ArUco检测、亚像素优化、PnP解算。
  - 调试曝光参数，防止金属反光导致Marker识别失败（必要时增加偏振片或调整灯光）。
  - 单元测试：手持标定板移动，验证输出的X/Y/Z/Yaw数据是否平滑准确。
- **13:00 - 16:00 [坐标系动态更新联调]**
  - 静态测试：保持电池不动，运行视觉算法，将计算出的T_{B}^{M}发给机器人，验证机器人是否认为当前位置就是示教位置。
  - 动态测试（模拟误差）：
    1. 人为移动电池包（平移20mm，旋转1度）。
    2. 运行自动纠偏程序。
    3. 命令机器人移动到“检测点1”。
    4. 验收标准：机器人TCP末端是否精准指在检测点1上（物理误差<1mm）。
- **16:00 - 20:00 [半自动流程逻辑编写]**
  - 编写主控脚本main_controller.py，集成“UI提示 -> 龙门架确认 -> 拍照 -> 纠偏 -> 动作”的状态机。
  - 加入异常处理逻辑（如未检测到Marker则报警停止，防止撞机）。

Day 3：系统优化、压力测试与Demo演示

目标：提升系统稳定性，处理Z轴等边界情况，完成最终演示。

- **09:00 - 11:00 [Z轴与深度补偿]**
  - 针对AGV悬挂沉降，利用Realsense的深度流数据校验PnP解算的Z轴。如果两者偏差过大，取深度流数据修正Z值，确保拍摄距离（Working Distance）恒定，保证图像清晰度。
- **11:00 - 14:00 [全流程压力测试]**
  - 龙门架盲测：操作员故意将龙门架停在偏离标记点50mm的位置，验证系统能否自动补偿回来。
  - 极限角度测试：将电池包偏转2度（项目要求上限），验证PnP解算的稳定性。
- **14:00 - 16:00**
  - 清理代码，固化曝光时间、检测阈值等参数。
  - 准备演示脚本：包含“未开启纠偏（失败案例）”与“开启纠偏（成功案例）”的对比演示。
- **16:00 - 18:00 [最终验收]**
  - 向项目干系人展示Demo：
    1. 手动推龙门架到位。
    2. 一键启动。
    3. 机器人自动寻找Marker，调整姿态。
    4. 精准飞拍各个检测点。

- 展示数据：界面实时显示“检测到偏差：X=12.5mm, Yaw=1.2deg，已补偿”。

---

8. 结论与展望

本方案通过深度整合Elite机器人的开放SDK与Realsense的视觉感知能力，巧妙地利用“相对坐标系重构”数学原理，解决了AGV物流带来的非结构化难题。

核心价值点：

1. 低成本高柔性：无需昂贵的激光定位或龙门架改造，仅靠单目/RGBD视觉即可容忍厘米级误差。
2. 快速部署：Day 1-3计划详尽，算法基于成熟的开源库（OpenCV ArUco），极大降低了开发风险。
3. 可扩展性：当前的2D PnP逻辑可无缝平滑过渡到Phase 2的3D点云配准（ICP），只需替换vision_core.py中的解算模块，而无需改动机器人的控制架构。

该实验室Demo的成功将为后续的大规模产线落地提供坚实的数据支撑和技术信心。

Works cited

1. 柔性拍摄项目解决方案.pdf
2. CS66 Robotic Arm for Machine Tending & Quality Inspection | Elite Robots, accessed February 8, 2026, https://www.eliterobots.com/cobots/cs66
3. ELITE ROBOTS CS Series User Guide, accessed February 8, 2026, https://twtrobot.com/wp-content/uploads/2025/11/CS_UserManual_CS625_Ver2.13.2.pdf
4. How to convert coordinate systems from a camera, to a robotic arm, to the real 3D world?, accessed February 8, 2026, https://robotics.stackexchange.com/questions/116727/how-to-convert-coordinate-systems-from-a-camera-to-a-robotic-arm-to-the-real-3
5. turlab / ros2-aruco-pose-estimation · GitLab, accessed February 8, 2026, https://turlab.itk.ntnu.no/turlab/ros2-aruco-pose-estimation
6. Detection of ArUco Markers - OpenCV Documentation, accessed February 8, 2026, https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html
7. Aruco Pose Estimation with ROS2, using RGB and Depth camera images from Realsense D435 - GitHub, accessed February 8, 2026, https://github.com/AIRLab-POLIMI/ros2-aruco-pose-estimation
8. How to improve PnP pose estimation? - Python - OpenCV Forum, accessed February 8, 2026, https://forum.opencv.org/t/how-to-improve-pnp-pose-estimation/13545
9. CS ScriptManual V2.10.0 | PDF | Control Flow | Parameter (Computer Programming), accessed February 8, 2026, https://www.scribd.com/document/855875409/CS-ScriptManual-V2-10-0