Sub-Millimeter Accuracy Autonomous Placement on a Deformable Part

A perception → calibration → manipulation pipeline, built in 2 weeks

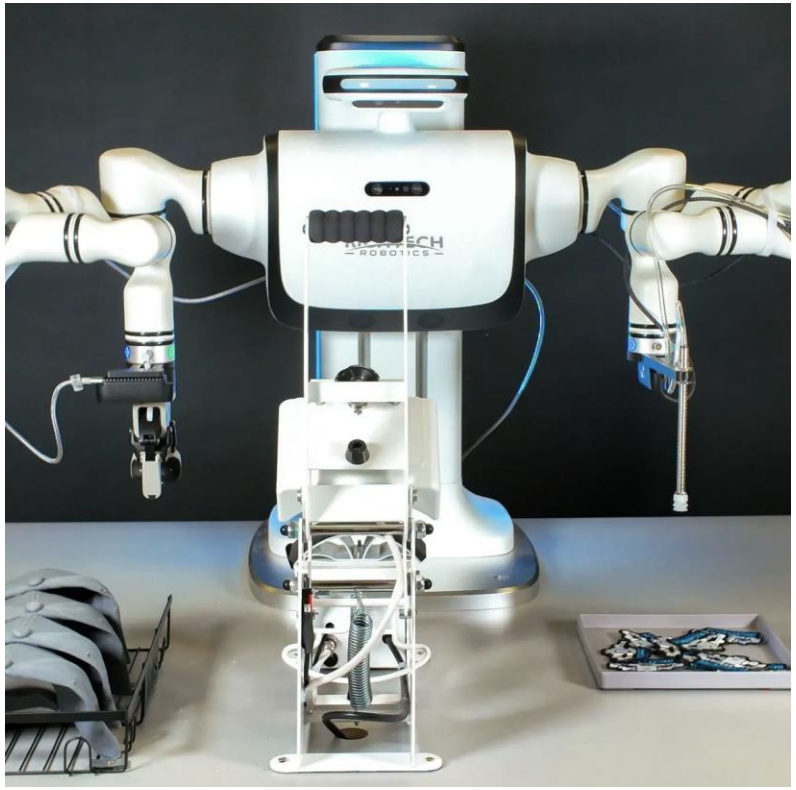
(NVIDIA GTC hat-pressing robot demo)

Jonathan Fu · R&D Engineer, Richtech Robotics

SpaceX On-Site · July 28, 2026

Executive Summary

- The ask
  - Unveil Dex, the new humanoid model
  - Flagship live demo for NVIDIA GTC — Demonstrate humanoid-robot capability for automation in a manufacturing environment
  - Outreach & credibility
- Why a hat-press?
  - Cheap, deformable, real-world part
  - Logo press = real manufacturing task + crowd-pleasing giveaway
- Constraints
  - 2 weeks, concept to demo
  - One live shot, no re-dos, uncontrolled floor lighting
- My solution
  - Eye-in-hand calibrated vision pipeline, target < 0.5 mm placement
  - Chosen after testing a simpler eye-to-hand setup (slide 6)
- Outcome
  - 200+ hats pressed live, 0.04 mm accuracy, 1.6 mm variance
  - Featured by NVIDIA Robotics, praised by NVIDIA's CTO



Agenda

- Background & why this project
- Goal, team & why this solution (vs. simpler options)
- Problem 1 — Calibration (+ a critical problem that I solved)
- Problem 2 — Perception (vision) on a deformable part
- Results, validation, lessons, what I'd do differently (+ the regret)

Background

Bachelor's Mechanical Engineering

Tooling & Automation Engineer

CWRU

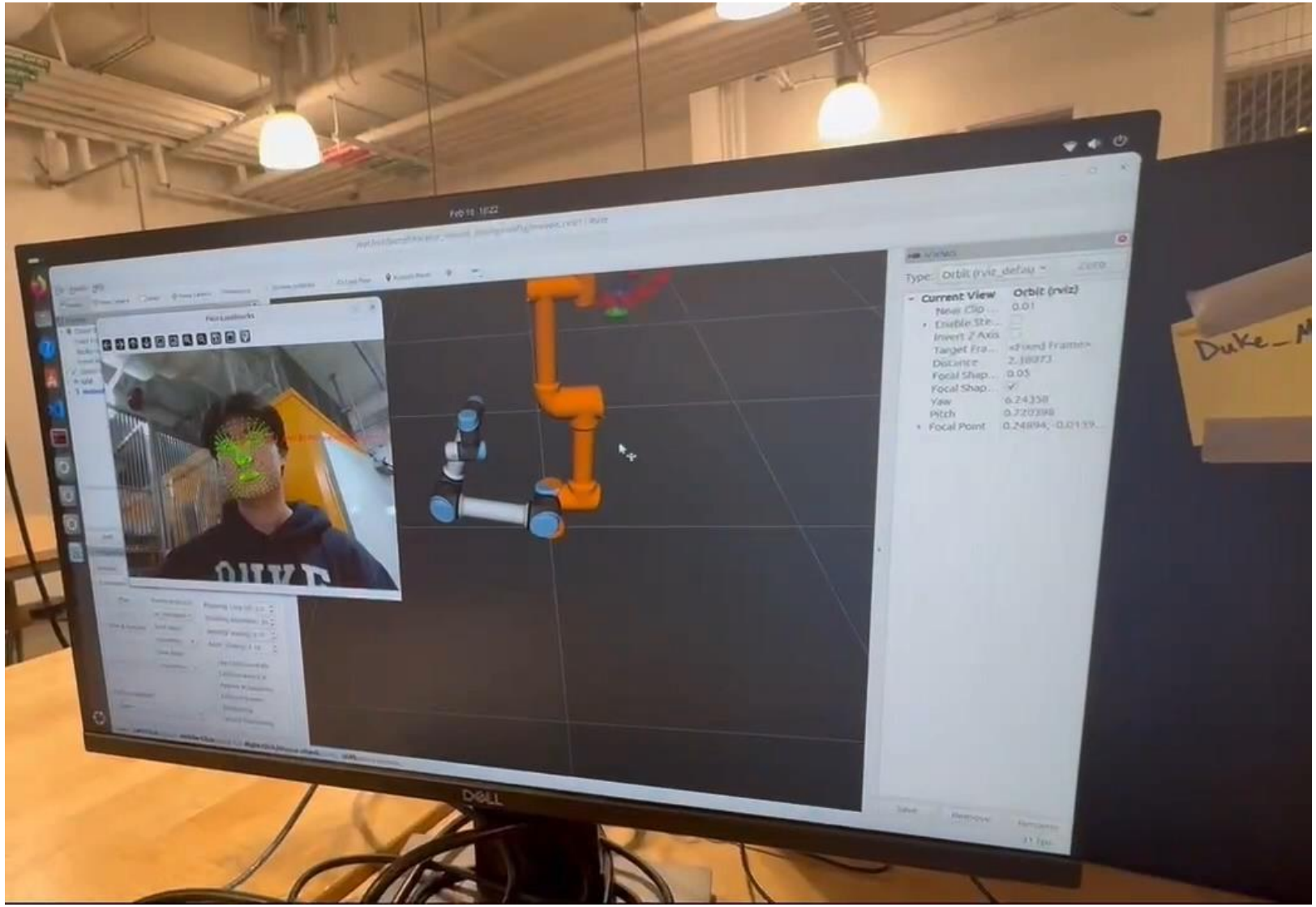
LINCOLN ELECTRIC

Duke

RICHTECH ROBOTICS

Master's Mechanical Engineering

R&D Engineer

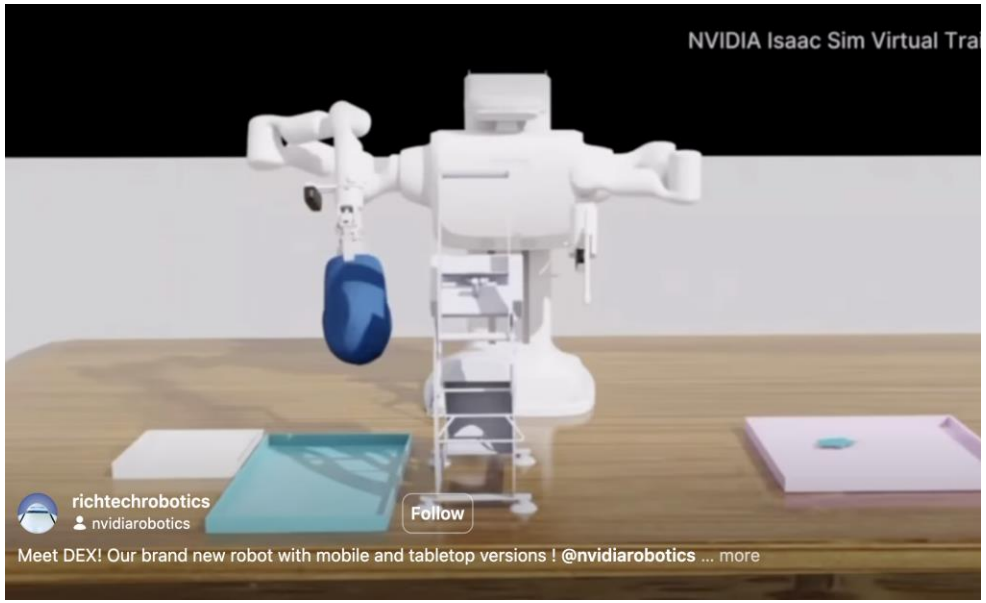


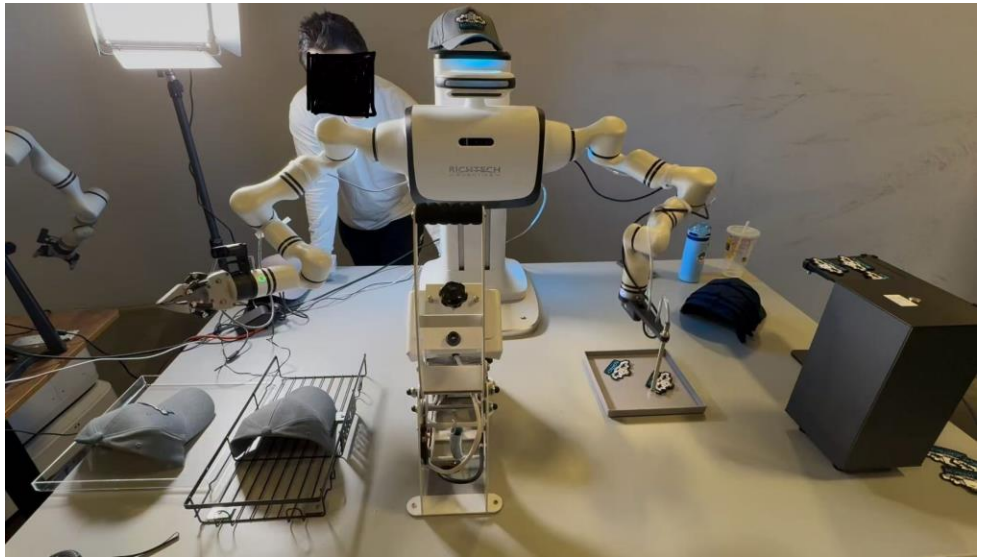
A couple of my recent passion projects.

Goal, Scope & Team

- Goal: press a vinyl logo onto a hat, fully autonomously, at < 0.5 mm placement error
- Pipeline: detect & localize hat + logo → estimate pose → calibrate camera-to-arm → press
- Phases (2 weeks): Design (pipeline architecture) → Build (calibration + vision) → Test (placement-accuracy and precision validation, slide 10)
- No days off, 9am-1am
- Team — 3 R&D engineers

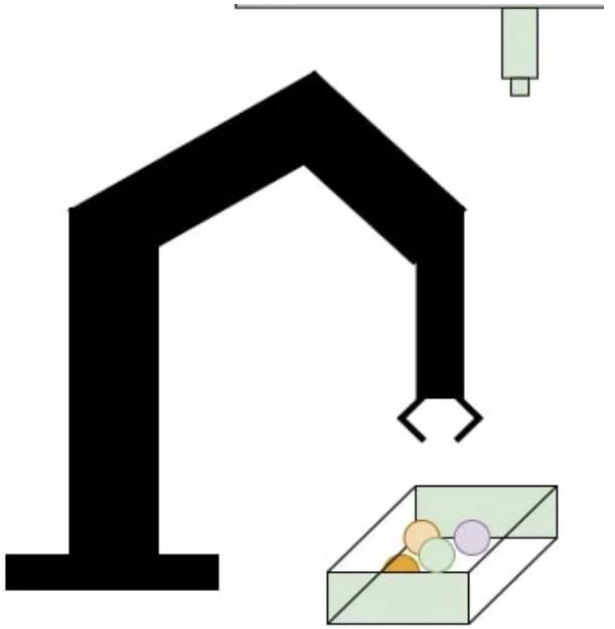
Owners	Responsibilities
Engineer A	Physical robot, task-space logistics, kinematics
Engineer B	Simulation (NVIDIA Isaac Sim)
Me	Calibration + full perception pipeline — critical path to the accuracy and precision number



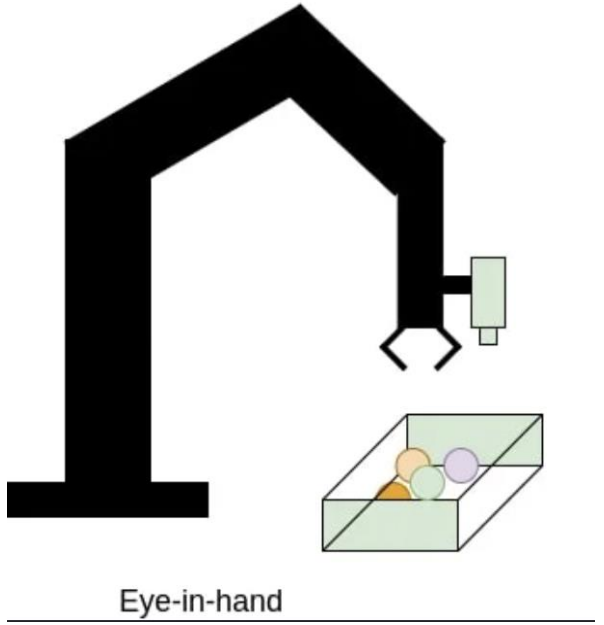


Why This Solution?

Approach	Why not / why yes
No perception — hardcoded positions	✗ Hat is deformable & hand-placed ✗ Target shifts every cycle
Eye-to-hand (camera fixed in workspace) — tested pre-concept	✗ Arm occludes the target on approach ✗ Fixed, far viewpoint caps resolution on a small logo/seam ✗ Camera fixture clutters demo environment
Eye-in-hand (camera on the wrist) — Chosen	✓ Move in for high-res close-ups of the deformable seam ✓ Adaptable viewpoint, sub-mm reachable



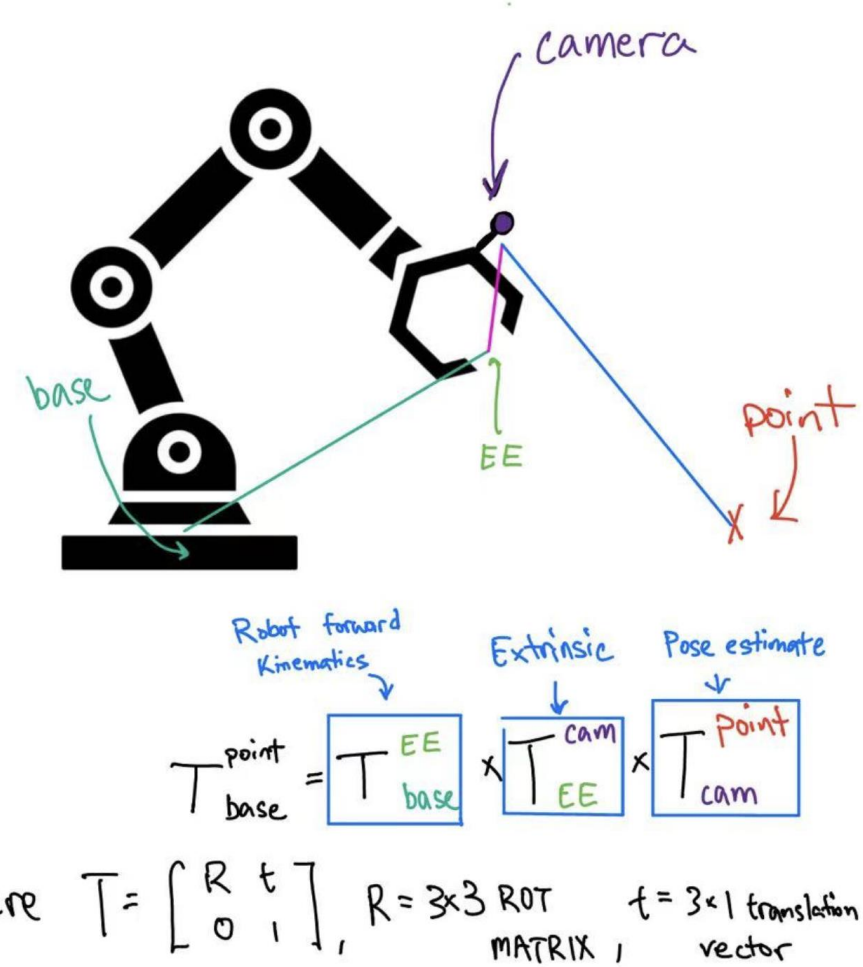
Eye-to-hand



Eye-in-hand

Problem 1: Calibration

- **Objective:** Relate what the camera sees (pixels) to where the arm moves (mm in its base frame), so the robot can act on what it detects.
- Two unknowns to solve:
  - Intrinsics — map image pixels to camera rays (focal length, optical center, lens distortion)
  - Extrinsic — the camera-to-end-effector transform (eye-in-hand: camera fixed to the wrist)
- Approach: Use a ChaRuCo calibration board.
- Machining analogy: this is establishing the datum between two coordinate systems — like homing a machine to a known work-coordinate origin.



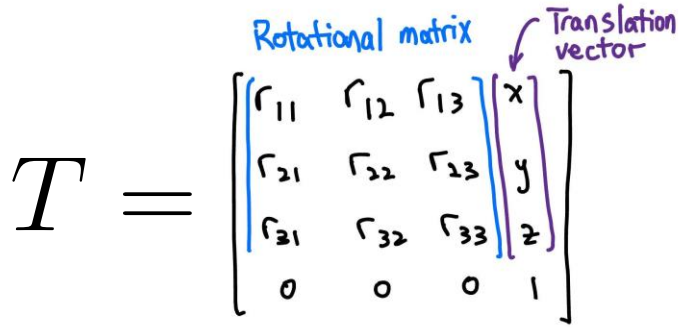
The Core Transform

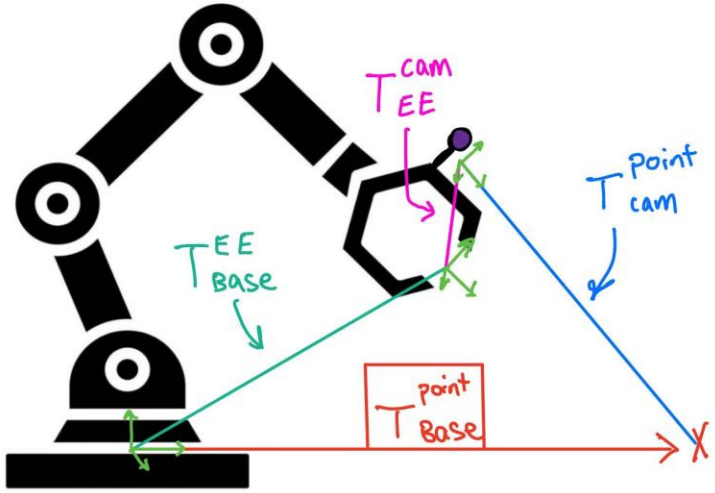
$$T_{base}^{point} = T_{base}^{EE} \cdot T_{EE}^{cam} \cdot T_{cam}^{point}$$

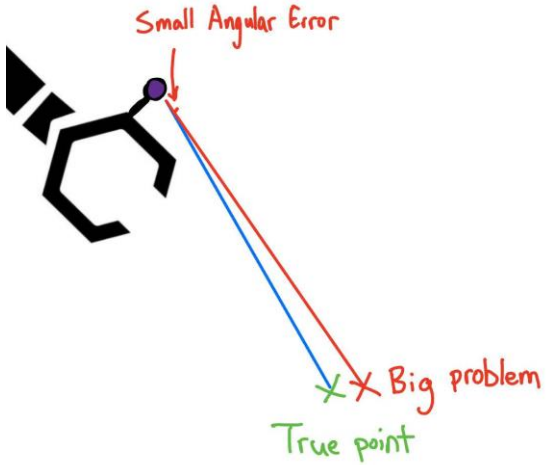
$$T_{base}^{point} = \cancel{T_{base}^{EE}} \cdot \cancel{T_{EE}^{cam}} \cdot \cancel{T_{cam}^{point}}$$

Term	What it is	Where it comes from
T_{base}^{point}	Target point in base frame	Matrix multiplication chain of all intermediates
T_{cam}^{point}	Target point in camera frame	Vision pose estimate + camera intrinsics
T_{EE}^{cam}	Camera in end-effector frame	The extrinsic matrix I calibrate
T_{base}^{EE}	End-effector in base frame	Forward kinematics chain (joint encoders)

Tolerance Stack-Up estimated in Appendix C

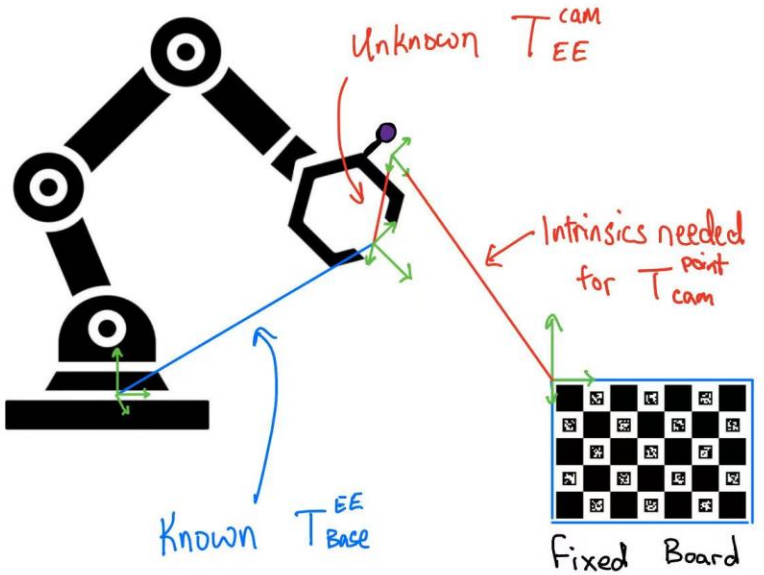






Calibration Pipeline

- How the intrinsics + extrinsic actually get solved:
  - Captured ~50 images of a fixed ChArUco board across varied angles & distances
  - Each image gives two knowns: exact board geometry + detected corner pixels(u, v)
  - OpenCV (open-source computer-vision library) runs a nonlinear least-squares fit. (Appendix D)
  - Solve for f_x, f_y, c_x, c_y, distortion, and T_{EE}^{cam}
- Why 50 images?
  - Too few → under-constrained
  - Stopped where the residual flattened

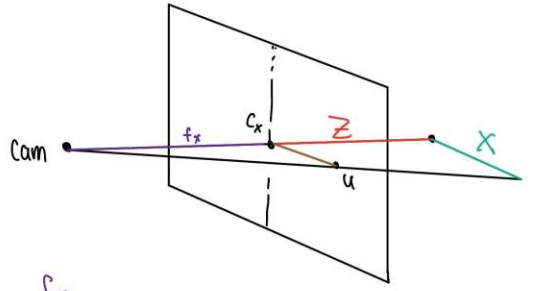


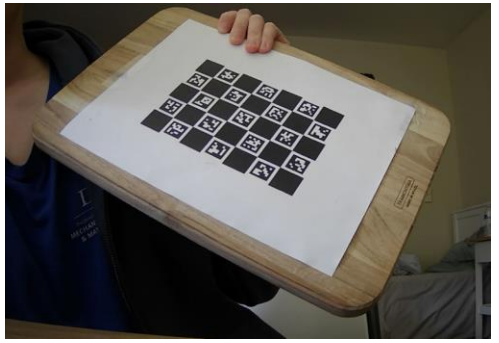
$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsics Matrix

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

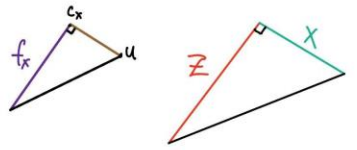
Extrinsic Matrix





$$\frac{u-c_x}{X} = \frac{f_x}{Z}$$

$$X = \frac{(u-c_x)Z}{f_x}$$



$$X = \frac{(u - c_x)\, Z}{f_x}, \quad Y = \frac{(v - c_y)\, Z}{f_y}, \quad Z = Z$$

Problem 1.5: The accuracy gap

Metric	**Measure**	**Units**	**Role**
Reprojection error (5→2mm)	Calibration self-consistency (reproject known 3D pts → compare to detected pixels)	Pixels → mm	Quality check
Placement error	Where the arm physically lands vs. truth	mm	The real goal (0.5mm)



$$\text{error}_{mm} = \text{error}_{px} \times \frac{\text{mm}}{\text{px}}$$

(Where mm/px is the known square size of ChArUco board)

ICP: The Accuracy Fix

- Iterative Closest Point (**ICP**) — refine T_{base}^{point} by aligning two point-cloud sets: where the model predicts target points are vs. where they actually are.
- The loop:
  1. Find ground-truth (base to point) of each ChaRuCo board corner
  2. Take an image of the board for the predicted corner locations in base frame
  3. Calculate the centroid of the measured point cloud and the predicted and subtract it from every point to get rid of translation differences.
  4. Solve the rigid transform (**R, t**) minimizing summed squared distance.
  5. Apply the transform directly to T_{base}^{point}, repeat until error stops dropping
- Result: refined extrinsic → real-world placement to 0.04 mm bias 1.6 mm repeatability from random depth noise. (validation, slide 18).
- Machining analogy: best-fit alignment of measured points to nominal geometry. Same idea as fitting a CMM to CAD.
- (Math in Appendix B)



Pre-ICP

↓



Post-ICP

Problem 2: Perception

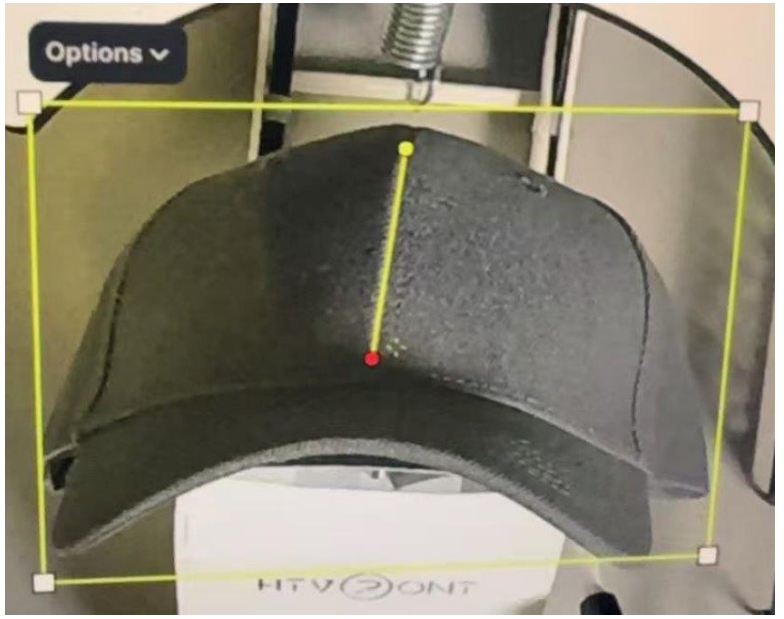
- **Objective**: Localize the hat seam and the vinyl logo center precisely enough to feed the transform chain (problem 1).
- Core barrier — the hat is deformable:
  - Fabric shifts, folds, and changes shape → the features I need aren't in a fixed, predictable place
  - Harder than a rigid pick-and-place, where geometry is known and repeatable



(**NOT** the hats used, just to illustrate the point)

Initial Approach: YOLOv8

- YOLOv8 — "You Only Look Once". A real-time object-detection model.
- The bet: train it to detect the hat, seam region, and vinyl logo, even under bad lighting or partial occlusion
- I led the data effort (my ownership):
  - Coordinated 4 technicians + R&D teammates
  - Labeled ~2,000 images of hats & vinyls in ~10 days (Roboflow)
  - Tested different labeling conventions — box tightness, how to handle a hidden seam
- Key findings:
  - The labeling method moved performance as much as the model did. Data quality > model complexity.
  - Just a couple of poorly labeled images ruins the model.



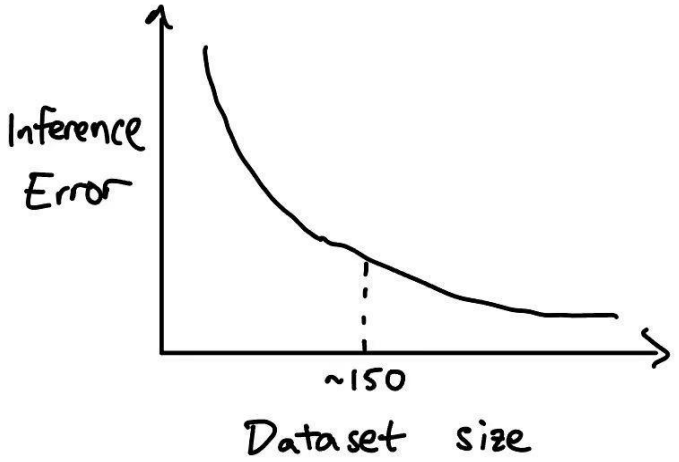


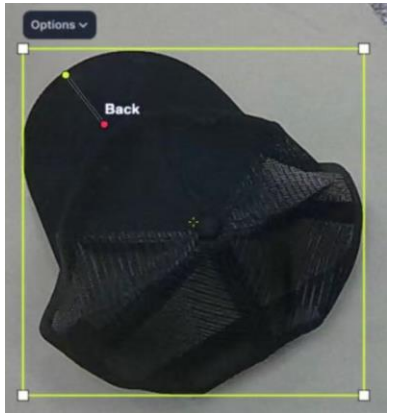
Factors: Data Economics

- I treated labeling as an economics problem — not "label until it's perfect."
- The variables:
  - Labeling efficiency (images/hour)
  - Labeling accuracy
  - Inference error vs. dataset size
  - Different people label slightly differently
- The insight: Error drops steeply, then begins to plateau (~150 images) for this task
- When we changed hat color mid-project, these curves told me how much re-labeling was actually needed

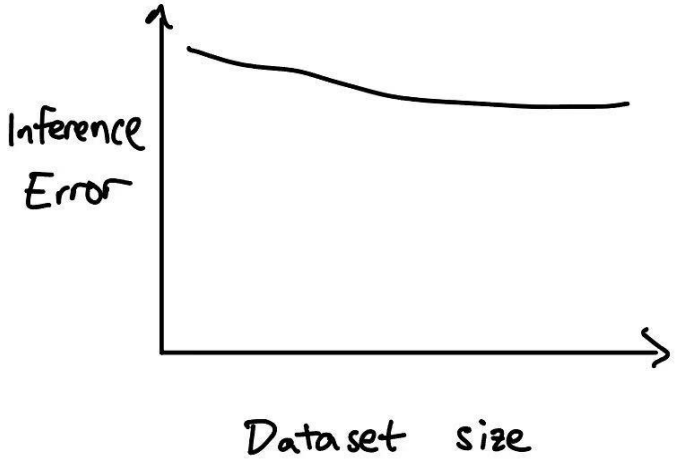


Method A





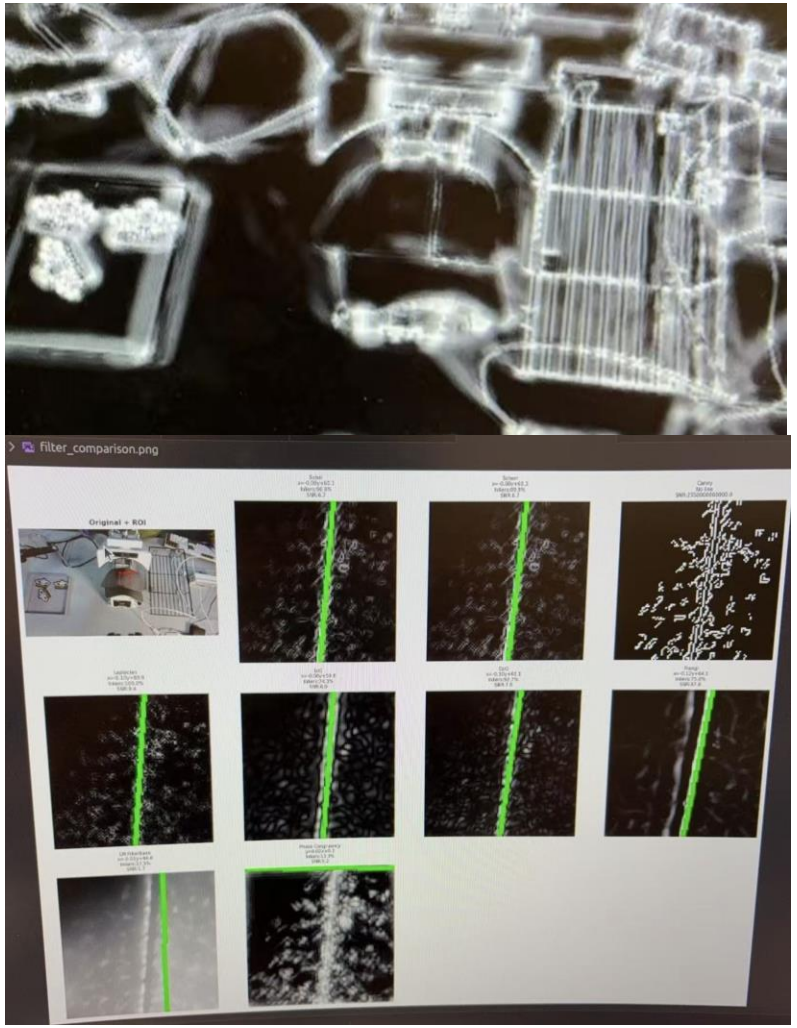
Method B



Labeling methodology completely changes outcome

Alternative Method Explored: Edge Detection

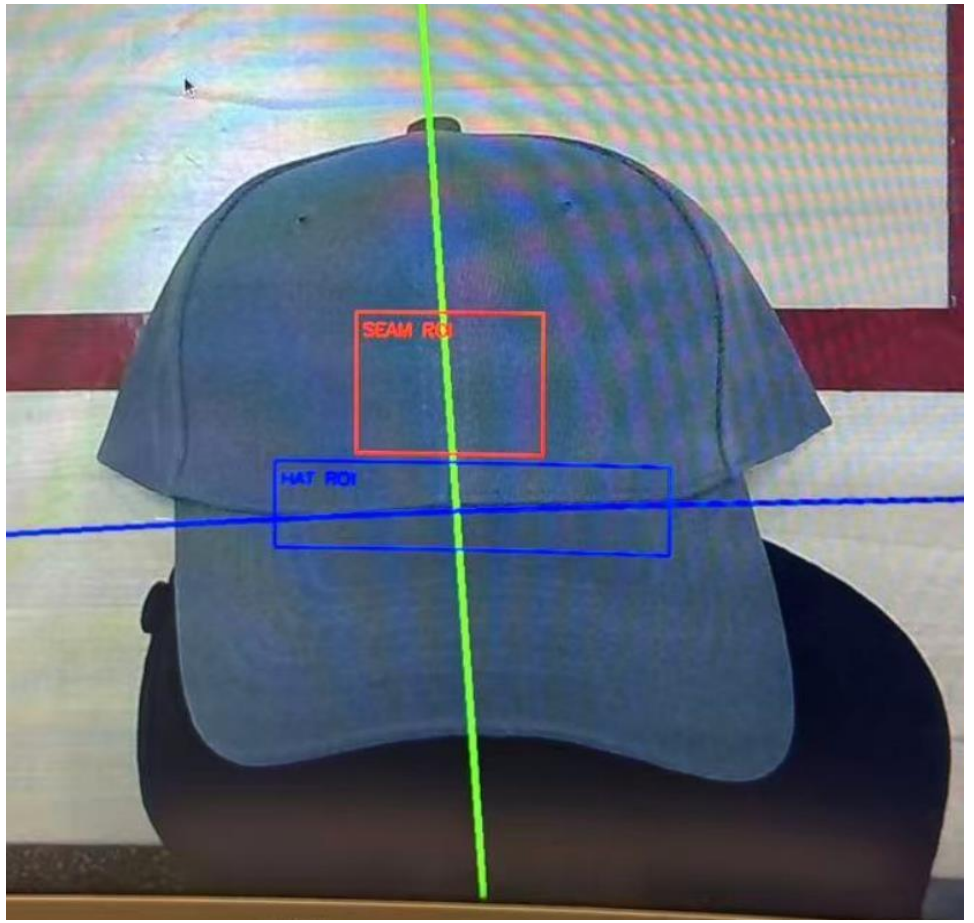
- YOLO accuracy was plateauing, unsure it'd hit target
- Hedged on a classical route in parallel: Edge detection (**no machine learning**).
- Many filters explored. Landed on **Frangi**.
- A "**vesselness**" filter built to enhance thin ridges (originally blood vessels in medical imaging). A hat seam is geometrically similar.



Line of best fit seam predictions from various edge-detection filters

Edge Detection Pipeline

- How Frangi localizes the seam, end to end:
  - Grayscale the image
  - Place a fixed bounding box where the hat seams are always in the frame
  - Convolve directional filters across the box → seam-ridge response
  - Fit a line to the ridge response (best fit across the response matrix)
  - Using the equation of the two lines, find intersect to resolve the centerpixel location (u, v) in the image
  - Use known depth (z) → reproject to the 3D point for the transform chain
- (Convolution example in Appendix)



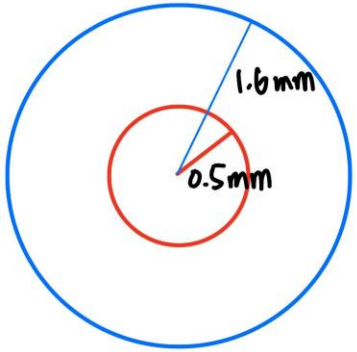
$$X = \frac{(u - c_x)\,Z}{f_x}, \quad Y = \frac{(v - c_y)\,Z}{f_y}, \quad Z = Z$$

Pros and Cons

	YOLO (Shipped)	Frangi (parallel, didn't ship)
Pros	- Infers seam even when dark - Accurately localizes seam	- Extremely accurate and stable - no model or training data needed
Cons	- Time-consuming to train - Not built for precise features on deformable parts - Slight jitter when camera moves	- Unreliable in low/uneven light

- Decision: shipped YOLO
  - Working, robust, already integrated with the manipulation
  - Switching meant redo manipulation in under 2 days + real lighting risk
- **Timeline:** I believed Frangi's accuracy was worth it but deferred because of time constraint.

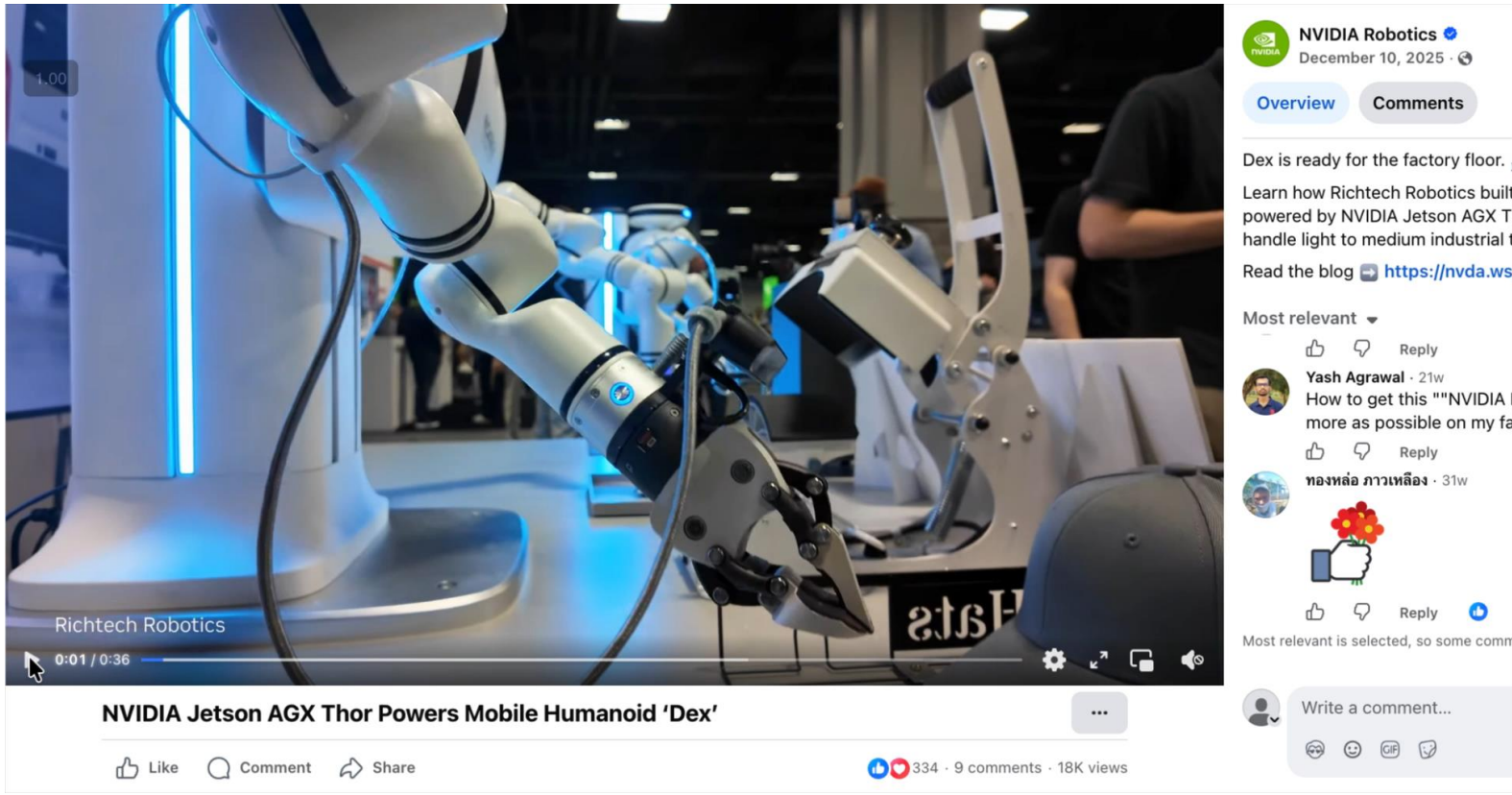
Live at NVIDIA GTC



Target

Achieved

- 200+ hats pressed & given away live at GTC
- Accuracy: 0.04 mm (bias) — beat target
- Precision: ~1.6 mm (mean distance from target) —depth-limited, missed 0.5 mm; cause: Orbbec RGBD noise
- Validated like a machine tool
  - Drove the arm to known ChArUco corners
  - Accuracy (bias): 0.04 mm — magnitude of the mean error vector. No detectable systematic offset
  - Precision (repeatability): Averaged error $\sqrt{(x^2+y^2+z^2)}$ over 25 trials. Per-placement spread, floored by Orbbec RGBD depth noise



Posted by NVIDIA Robotics · praised by NVIDIA's CTO

What I Learned

- The whole pipeline determines performance, not any single model. Calibration, vision, and manipulation had to be right together.
- Validation is part of the design. Reprojection error ≠ placement error. Measure what actually matters, physically.
- Calibration removes systematic error, not random sensor noise.
- Data quality > model complexity. Labeling method moved accuracy delta drastically.

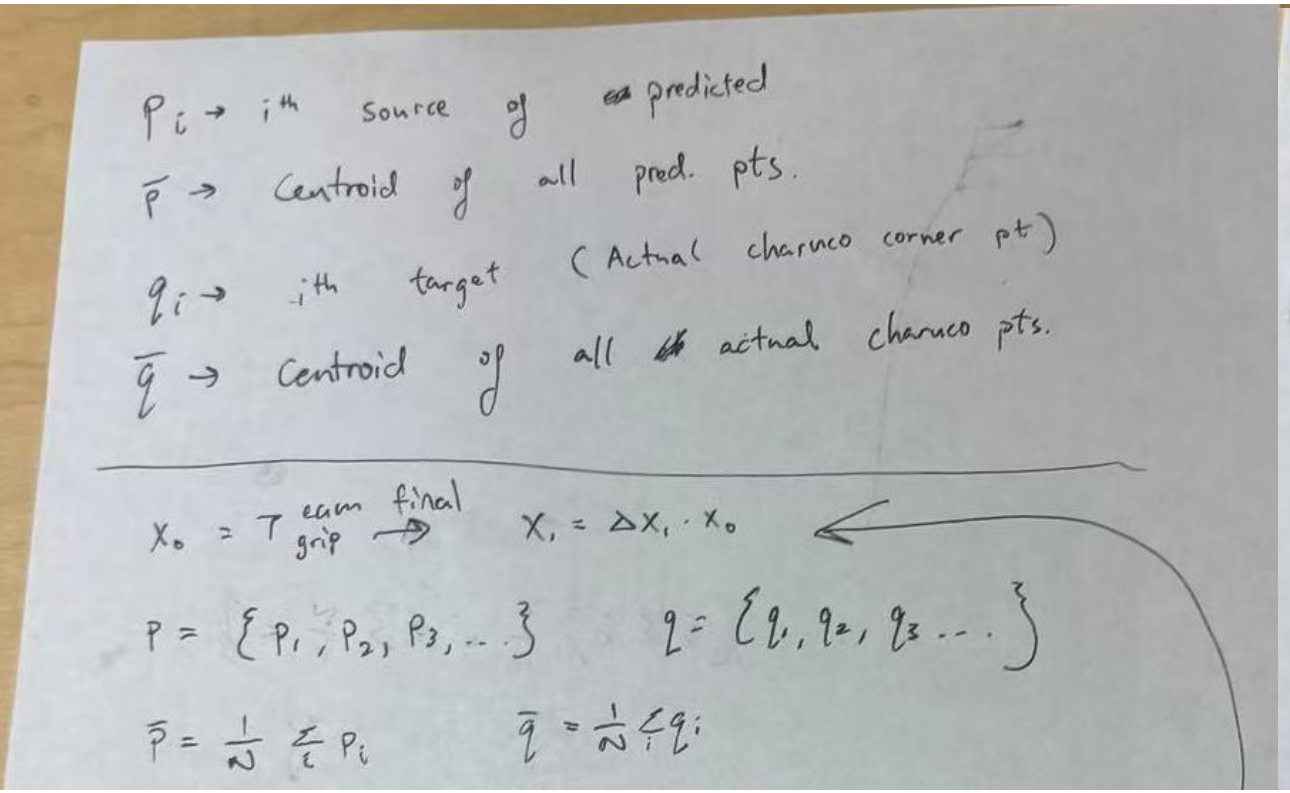
If I Did It Again: ~Half the Time

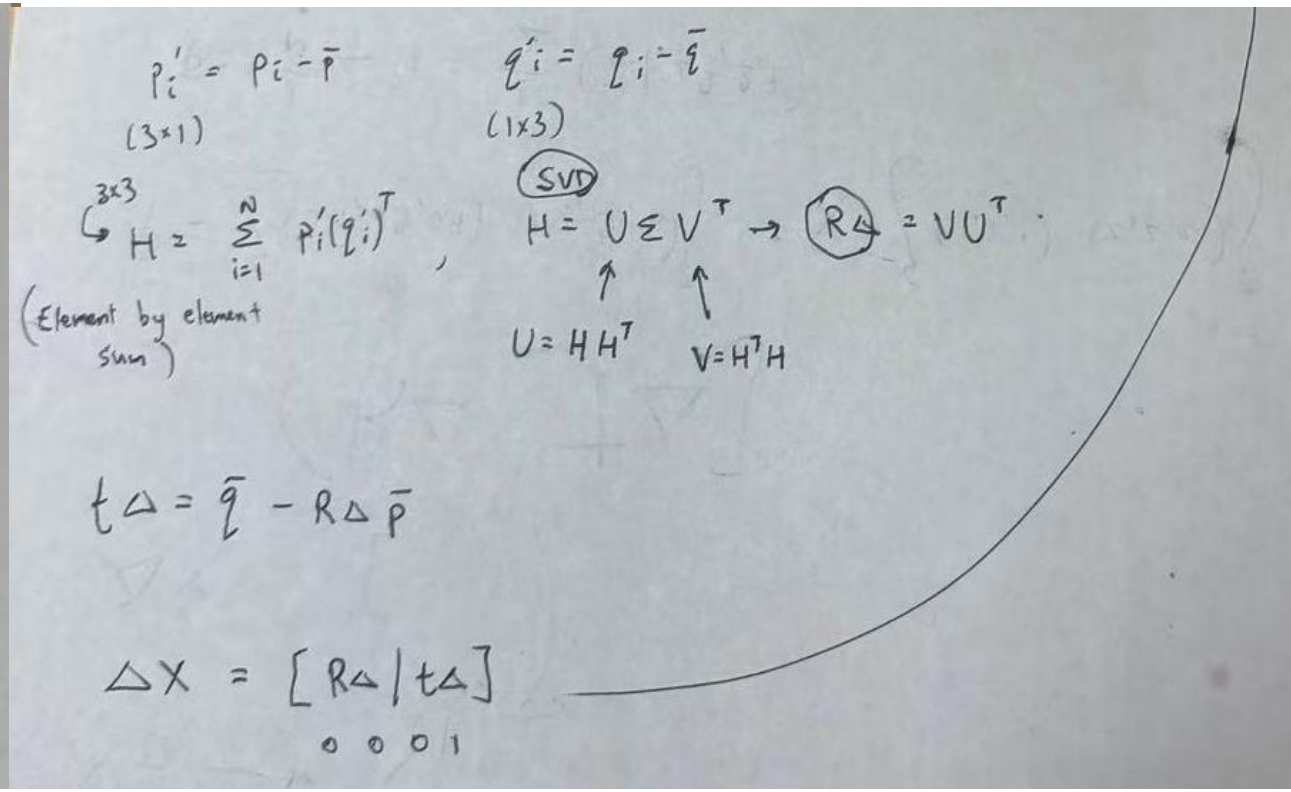
- Temporally average depth frames to cut the 1.6 mm precision. Tradeoff is a bit of latency.
- Use the edge-detection pipeline for hat seam detection and YOLOv8 for the non-deforming vinyl logos.
- Make sure the data labeling method is effective at decreasing error as more images are added to the dataset.
- Keep both vision approaches integrated. Decide on the floor, with data, once lighting is known.
- Reuse the calibration + ICP routine I've now built.

Why This Transfers

NVIDIA GTC Experience	Precision Machining
Camera-to-arm calibration	Machine-tool calibration
Compounding transform error	Tolerance stack-up
Separating accuracy from precision — a 0.04 mm bias from a 1.6 mm repeatability floor	Metrology and process capability
Choosing eye-in-hand over eye-to-hand	Fixturing and datum decision

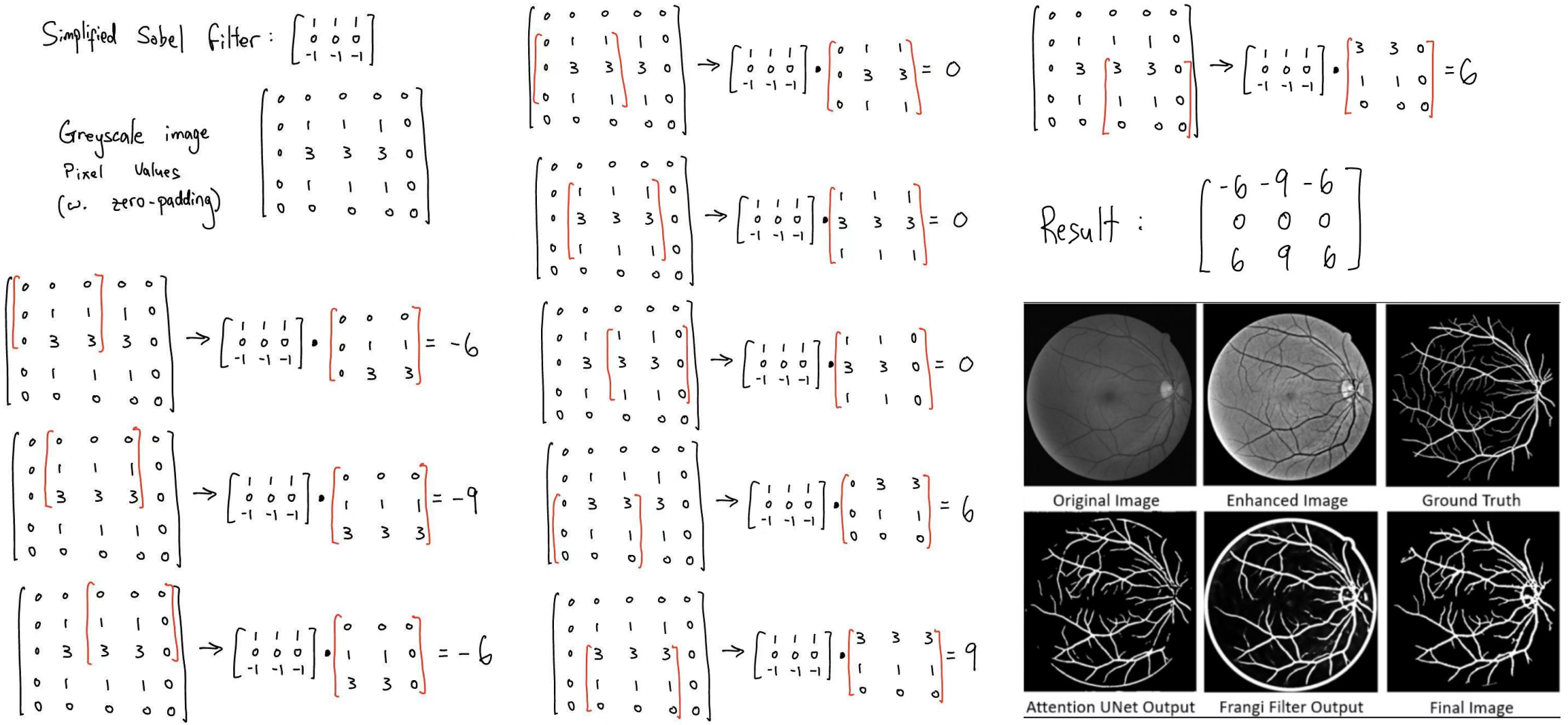
Appendix A: ICP





Actual notes I took after realizing this method was the solution we were looking for

Appendix B: Convolution During Edge-Detection



Appendix C: Estimated Error Budget

Source	Bias (mm)	Spread (mm)
Extrinsic (hand-eye), post-ICP	~0.03	--
Orbbec RGBD depth noise	~0	1.5 (dominant)
Camera Intrinsics	~0.02	~0.3
Vision localization	--	~0.4
Robot repeatability	--	~0.1
Combined (Root-Sum-Square)	= 0.04 (accuracy)	= 1.6 (precision)

Appendix D: Intrinsics Calibration

Known	Unknown
3D board points X_i(ChArUco geometry)	Intrinsics K: f_x, f_y, c_x, c_y
Detected Corner Pixels (u_i, v_i)	Distortion k_1, k_2, k_3, p_1, p_2
	Per-image pose (R_j, t_j)

- Forward projection model — predict a pixel from a board point:
  - 1. To camera frame: $X_c = R_j X_i + t_j$
  - 2. Perspective divide: $x = X_c/Z_c, ; y = Y_c/Z_c$
  - 3. Distort (radial + tangential): $x_d = x(1 + k_1 r^2 + k_2 r^4 + \dots) + [\text{tangential}]$
  - 4. To pixels: $\hat{u} = f_x x_d + c_x, \quad \hat{v} = f_y y_d + c_y$
  - 5. The prediction model: $\hat{p}_{ij} = \pi(K, \text{dist}, R_j, t_j; , X_i)$
  - What we want to minimize (per point): $p_{ij} - \hat{p}_{ij} = (u - \hat{u}, v - \hat{v})$

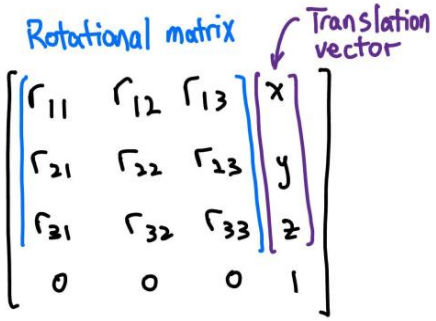
Appendix D Continued: Intrinsics Calibration

- Cost function — minimize total reprojection error:

$$\min_{K,\text{dist},R_j,t_j} \sum_j \sum_i \left\| p_{ij}^{\text{measured}} - \pi(K, \text{dist}, R_j, t_j; X_i) \right\|^2$$

- Measured value — corner (u, v) from the image, no parameters involved (OpenCV corner detection)
- Model value — from the 3D geometry and estimated parameters
- Levenberg–Marquardt to optimize parameters

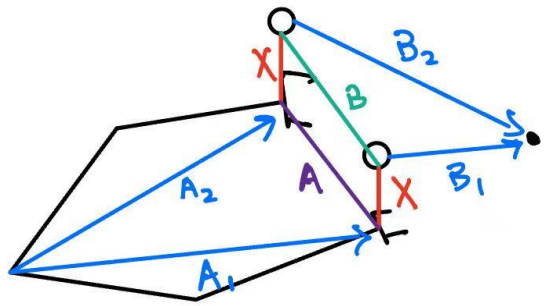
Appendix E: Extrinsic Matrix T_{EE}^{cam}



- Tsai's closed-form solver:
  - AX = XB, each variable is a 4x4 matrix
  - A: Relative motion of robot end-effector
  - B: Relative motion of camera (using board as anchor)
  - X: The hand-eye extrinsic matrix we want

$$A = \left[T_{base}^{EE}(j)\right]^{-1} \cdot T_{base}^{EE}(i) = T_{EE}^{base}(j) \cdot T_{base}^{EE}(i)$$

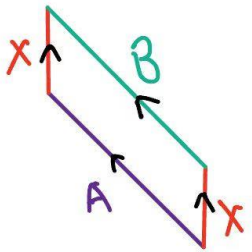
$$B = T_{cam}^{board}(j) \cdot \left[T_{cam}^{board}(i)\right]^{-1} = T_{cam}^{board}(j) \cdot T_{board}^{cam}(i)$$



$A = A_1^{-1} A_2$

$B = B_1 B_2^{-1}$

- A single motion isn't enough. Rotation is under-constrained
- Each AB pair further constrains X
- One single least-squares solve for X
- Think of each AB pair as a point, X is the line of best fit.



$AX = XB$

Appendix F: The Hat-Press

