Roboter-Pose-Erkennung aus Mehrkamera-ArUco-Aufnahmen

Pipeline, mathematische Grundlagen und Konfiguration

appRobotRendering — technische Dokumentation

2. Juni 2026

Zusammenfassung

Diese Dokumentation beschreibt eine Bildverarbeitungs-Pipeline, die aus synchronen Fotos eines Roboterarms von mehreren kalibrierten Kameras die Gelenkstellung (sieben Achsen x, y, z, a, b, c, e) rekonstruiert. Auf dem Roboter und einer Referenzplatte (*Board*) sind ArUco-Marker angebracht. Aus den *vier Eckpunkten* jedes Markers wird per Mehrsicht-Triangulation eine volle 6-DOF-Markerpose (Position *und* Flächennormale) bestimmt; aus diesen Beobachtungen werden über das Vorwärtskinematik-Modell die Gelenkwinkel geschätzt. Es werden vier austauschbare Schätzverfahren beschrieben und an simulierten Szenen mit bekannter Grundwahrheit validiert (mittlerer Winkelfehler 0,25°, Positionsfehler 0,10 mm).

Inhaltsverzeichnis

1 Einführung und Problemstellung

Gegeben sind C kalibrierte Kameras, die denselben Roboter gleichzeitig fotografieren. Der Roboter trägt eine kinematische Kette starrer Glieder (*Links*); auf den Gliedern und auf der fixen Bodenplatte sitzen ArUco-Marker bekannter Größe. Das *Board* ist die Wurzel der Kette und definiert das Weltkoordinatensystem W. Gesucht ist der Gelenkvektor

$$\mathbf{q} = (x, y, z, a, b, c, e) \in \mathbb{R}^7,$$

mit linearen Achsen x, e (in mm) und rotatorischen Achsen y, z, a, b, c (in Grad).

Die zentrale Idee ist, dass ein ArUco-Marker *kein Punkt* ist, sondern ein ebenes Quadrat bekannter Kantenlänge mit vier detektierten Eckpunkten. Werden diese vier Ecken über mehrere Kameras trianguliert, liefert jeder einzelne Marker neben seiner Position auch eine *gemessene* Orientierung. Dadurch genügt bereits ein einziger sichtbarer Marker, um den Winkel des Gliedes zu bestimmen, auf dem er sitzt.

2 Notation und Koordinatensysteme

Punkte im Weltsystem W werden mit $\mathbf{X} \in \mathbb{R}^3$ bezeichnet (Einheit mm). Eine Kamera c besitzt die Intrinsikmatrix $K_c \in \mathbb{R}^{3\times3}$, die Verzeichnungskoeffizienten D_c sowie die Extrinsik $(R_c, \mathbf{t}_c)$ in *world-to-camera*-Konvention,

$$\mathbf{X}^{\text{cam}} = R_c\,\mathbf{X} + \mathbf{t}_c, \qquad R_c \in SO(3).$$

Das Kamerazentrum im Weltsystem ist $\mathbf{C}_c = -R_c^\top \mathbf{t}_c$. Die normierte (intrinsikfreie) Projektion eines Weltpunkts lautet

$$\tilde{\pi}(\mathbf{X}; R_c, \mathbf{t}_c) = \frac{1}{[\mathbf{X}^{\text{cam}}]_3} \begin{pmatrix} [\mathbf{X}^{\text{cam}}]_1 \\ [\mathbf{X}^{\text{cam}}]_2 \end{pmatrix} \in \mathbb{R}^2. \tag{1}$$

Eine beobachtete Pixelkoordinate $\mathbf{u}$ wird mittels $\tilde{\mathbf{u}} = \text{undistort}(\mathbf{u}; K_c, D_c)$ in dieselbe normierte Ebene überführt. Rotationen werden bei der Optimierung über den Rodrigues-Vektor $\boldsymbol{\omega} \in \mathbb{R}^3$ mit $R = \exp([\boldsymbol{\omega}]_\times)$ parametrisiert.

3 Pipeline-Architektur

Die Verarbeitung erfolgt in vier Stufen (Skript `run/run_pipeline.bat`):

Stufe	Skript	Ausgabe
1	`1_detect_aruco_observations.py`	`*_aruco_detection.json` (Ecken)
2	`2_estimate_camera_from_observations.py`	`*_camera_pose.json`
3	`3_multiview_bundle_adjustment_v4.py`	`aruco_positions_initial.json`
3b	`3b_corner_marker_poses.py`	`aruco_marker_poses.json`
4	`pose_estimation.py`	`robot_state.json`

Stufe 3 (reine Center-Triangulation) wird für Visualisierung und Vergleich beibehalten; die eigentliche Posenschätzung nutzt die in Stufe 3b erzeugten Eckposen.

4 Stufe 1: ArUco-Detektion

Pro Bild liefert der OpenCV-ArUco-Detektor für jeden erkannten Marker m die vier Eckpunkte $\mathbf{u}_{m,0}, \ldots, \mathbf{u}_{m,3}$ im Uhrzeigersinn sowie deren Mittelpunkt. Zusätzlich werden Qualitätsmaße (Fläche, Schärfe als Laplace-Varianz, Kontrast, Randabstand, Kantenverhältnis) berechnet und eine heuristische Konfidenz abgeleitet. Die Eckpunkte $\mathbf{u}_{m,k}$ sind die *primären* Messungen; der Mittelpunkt ist nur ihr Mittelwert.

5 Stufe 2: Kamera-Pose-Schätzung

Die Board-Marker haben bekannte Weltpositionen $\mathbf{X}_i$ (aus `robot.json`). Aus ihren beobachteten Mittelpunkten wird die Pose jeder Kamera geschätzt. Ein robuster Anfangswert ergibt sich aus `solvePnP` (IPPE-Square) je Marker, gefolgt von einem starren Ausgleich der per-Marker-Kamerazentren. Verfeinert wird mit Levenberg–Marquardt auf den normierten Reprojektionsresiduen:

$$\min_{\boldsymbol{\theta}} \sum_i \left\| \tilde{\mathbf{u}}_i - \tilde{\pi}\big(\mathbf{X}_i;\, R(\boldsymbol{\omega}), \mathbf{t}\big) \right\|^2, \qquad \boldsymbol{\theta} = (\boldsymbol{\omega}, \mathbf{t}) \in \mathbb{R}^6. \tag{2}$$

Aus der Jacobi-Matrix am Optimum wird eine 6×6-Kovarianz und daraus die Unsicherheit von Kamerazentrum und Orientierung propagiert. Bemerkung: Da die Board-Marker nahezu koplanar sind ($z \approx 0$), ist die Tiefe schlecht konditioniert — eine korrekte Intrinsik ist deshalb essenziell (siehe Abschnitt 13).

6 Stufe 3b: Eck-Triangulation und Normalenschätzung

6.1 Mehrsicht-Triangulation (DLT)

Sei $V_{m,k} \subseteq \{1, \ldots, C\}$ die Menge der Kameras, die Ecke k von Marker m sehen. Mit der normierten Projektionsmatrix $P_c = [\, R_c \mid \mathbf{t}_c \,]$ (Zeilen $\mathbf{p}_{c,1}^\top, \mathbf{p}_{c,2}^\top, \mathbf{p}_{c,3}^\top$) und der entzerrten Beobachtung $\tilde{\mathbf{u}} = (x, y)$ liefert jede Kamera zwei lineare Gleichungen in den homogenen Punktkoordinaten $\mathbf{X}_h$:

$$x\, \mathbf{p}_{c,3}^\top \mathbf{X}_h - \mathbf{p}_{c,1}^\top \mathbf{X}_h = 0, \qquad y\, \mathbf{p}_{c,3}^\top \mathbf{X}_h - \mathbf{p}_{c,2}^\top \mathbf{X}_h = 0. \tag{3}$$

Das Stapeln über alle $c \in V_{m,k}$ ergibt $A\,\mathbf{X}_h = \mathbf{0}$; die Lösung ist der rechte Singulärvektor zum kleinsten Singulärwert von A, gefolgt von der Dehomogenisierung $\mathbf{X} = \mathbf{X}_h^{1:3} / \mathbf{X}_h^4$. Dies wird für alle vier Ecken ausgeführt und liefert $\mathbf{Y}_0, \ldots, \mathbf{Y}_3$.

6.2 Position und Normale

Die Markerposition ist der Schwerpunkt $\bar{\mathbf{Y}} = \frac{1}{4}\sum_k \mathbf{Y}_k$. Die Normale ist der Normalenvektor der Ausgleichsebene durch die vier Ecken: mit $M = [\mathbf{Y}_0 - \bar{\mathbf{Y}}; \ldots; \mathbf{Y}_3 - \bar{\mathbf{Y}}] \in \mathbb{R}^{4\times 3}$ und der Singulärwertzerlegung $M = U\Sigma V^\top$ ist

$$\mathbf{n} = \pm\, \mathbf{v}_3 \quad \text{(kleinster Singulärvektor)}, \tag{4}$$

wobei das Vorzeichen über die ArUco-Eckreihenfolge so gewählt wird, dass $\mathbf{n}$ zur Kamera weist (Konvention wie in der Grundwahrheit): $\mathbf{n} \leftarrow -\mathbf{n}$, falls $\mathbf{n} \cdot \big((\mathbf{Y}_1 - \mathbf{Y}_0) \times (\mathbf{Y}_2 - \mathbf{Y}_0)\big) > 0$. Die Validierung (`benchmark/stage0_corner_normals.py`) ergibt einen Median-Normalenfehler von 1,2° (p_{90} = 3,0°); die Aufschlüsselung je Glied (Tabelle 1) zeigt, dass selbst die kleinen, weit entfernten Fingermarker eine Orientierung von rund 1° liefern — die Grundlage dafür, dass ein einzelner Marker seinen Gelenkwinkel bestimmen kann.

Glied	# Marker	Normalenfehler Mittel [°]	Max [°]
Arm1	3	0,64	0,94
Arm2	4	0,78	1,04
Ellbow	4	0,96	1,99
FingerA	2	0,96	1,14
FingerB	2	1,43	1,44
Board	41	1,79	7,07

Tabelle 1: Genauigkeit der aus Ecken abgeleiteten Normale je Glied (Scene8).

7 Vorwärtskinematik

Die Links werden topologisch (Wurzel zuerst) sortiert. Für Link l mit Eltern-link p gilt

$$T_l \;=\; T_p \cdot \underbrace{\mathrm{Tr}(\mathbf{m}_l)\, R_{xyz}(\boldsymbol{\rho}_l)}_{\text{Mount}} \cdot \underbrace{\mathrm{Tr}(\mathbf{o}_l)\, R_{xyz}(\boldsymbol{\sigma}_l)}_{\text{Gelenkursprung}} \cdot Q_l(q_l), \tag{5}$$

mit der Gelenkbewegung

$$Q_l(q_l) = \begin{cases} R(\hat{\mathbf{s}}_l,\, q_l) & \text{revolut (Achse } \hat{\mathbf{s}}_l,\ \text{Winkel } q_l), \\ \mathrm{Tr}(\hat{\mathbf{s}}_l\, q_l) & \text{linear (Verschiebung } q_l), \\ I & \text{fest.} \end{cases} \tag{6}$$

Ein Marker mit lokaler Position $\mathbf{p}_m^{\text{lok}}$ und lokaler Normale $\mathbf{n}_m^{\text{lok}}$ erscheint im Weltsystem als

$$\mathbf{p}_m(\mathbf{q}) = T_{l(m)}(\mathbf{q}) \begin{pmatrix} \mathbf{p}_m^{\text{lok}} \\ 1 \end{pmatrix}, \qquad \mathbf{n}_m(\mathbf{q}) = R\big(T_{l(m)}(\mathbf{q})\big)\, \mathbf{n}_m^{\text{lok}}. \tag{7}$$

Die Kette dieses Roboters ist in Tabelle 2 zusammengefasst. Bemerkenswert: *Base*, *Hand* und *Palm* tragen keine eigenen Marker, und beide Finger teilen sich die Variable e (mit entgegengesetzten Achsen — ein Greifer).

Link	Eltern	Gelenktyp	Variable	Achse
Board	—	(Wurzel)	—	—
Base	Board	linear	x	$[1, 0, 0]$
Arm1	Base	revolut	y	$[-1, 0, 0]$
Ellbow	Arm1	revolut	z	$[-1, 0, 0]$
Arm2	Ellbow	revolut	a	$[0, -1, 0]$
Hand	Arm2	revolut	b	$[1, 0, 0]$
Palm	Hand	revolut	c	$[0, -1, 0]$
FingerA	Palm	linear	e	$[1, 0, 0]$
FingerB	Palm	linear	e	$[-1, 0, 0]$

Tabelle 2: Kinematische Kette: sieben Freiheitsgrade, davon zwei linear (x, e).

8 Pose-Estimation: Vier Schätzverfahren

Die Beobachtungen sind die Eckposen $\{(\mathbf{Y}_m, \mathbf{n}_m^o)\}_{m \in \mathcal{M}}$ aus Stufe 3b. Alle Verfahren parametrisieren über die *Gelenkvariablen* $\mathbf{q}$ (nicht über Links), wodurch geteilte Variablen (e) und markerlose Glieder (x, b, c) generisch behandelt werden. Das gemeinsame Residuum eines Markers kombiniert Position und Normale,

$$\mathbf{r}_m(\mathbf{q}) = \Big[\, \mathbf{p}_m(\mathbf{q}) - \mathbf{Y}_m \,;\;\; w_n\big(\mathbf{n}_m(\mathbf{q}) - \mathbf{n}_m^o\big) \Big] \in \mathbb{R}^6, \tag{8}$$

mit der Normalengewichtung w_n (Parameter `normal_weight`).

8.1 Blockstruktur der Kette

Für die sequenziellen Verfahren wird die Kette in Blöcke zerlegt: Von der Wurzel fortschreitend werden die Variablen markerloser Gelenke gesammelt und mit der Variable des nächsten markertragenden Gelenks zu einem Block zusammengefasst, der aus den Markern dieses Gliedes (zuzüglich Geschwistern mit geteilter Variable) geschätzt wird. Für diesen Roboter ergeben sich

$$B_1 = \{x, y\}\ \text{(Arm1)}, \quad B_2 = \{z\}\ \text{(Ellbow)}, \quad B_3 = \{a\}\ \text{(Arm2)}, \quad B_4 = \{b, c, e\}\ \text{(Finger)}.$$

8.2 Verfahren A — `sequential_vector`

Analytische Winkelbestimmung pro rotatorischem Gelenk mit ≥ 2 sichtbaren Markern auf demselben Glied. Mit der Weltachse $\hat{\mathbf{a}}$ des Gelenks (aus der FK bei $q = 0$) bildet man für Markerpaare (i, j) den Modellvektor $\mathbf{d}_{ij}^{\text{mod}} = \mathbf{p}_j(q{=}0) - \mathbf{p}_i(q{=}0)$ (im Weltsystem, da der lokale Linkframe durch die Elterngelenke bereits rotiert ist) und den Beobachtungs-vektor $\mathbf{d}_{ij}^{\text{obs}} = \mathbf{Y}_j - \mathbf{Y}_i$. Nach Projektion senkrecht zur Achse, $\mathbf{d}^{\perp} = \mathbf{d} - (\mathbf{d} \cdot \hat{\mathbf{a}})\hat{\mathbf{a}}$, ist der vorzeichenbehaftete Drehwinkel

$$\varphi_{ij} = \operatorname{atan2}\Big(\hat{\mathbf{a}} \cdot (\mathbf{d}_{ij}^{\text{mod},\perp} \times \mathbf{d}_{ij}^{\text{obs},\perp}),\ \mathbf{d}_{ij}^{\text{mod},\perp} \cdot \mathbf{d}_{ij}^{\text{obs},\perp}\Big). \tag{9}$$

Der Gelenkwinkel ist das mit $w_{ij} = \left\|\mathbf{d}_{ij}^{\text{mod},\perp}\right\| \left\|\mathbf{d}_{ij}^{\text{obs},\perp}\right\|$ gewichtete *zirkuläre* Mittel $q^* = \operatorname{atan2}(\sum w \sin\varphi,\ \sum w \cos$
Lineare, markerlose oder Einzelmarker-Gelenke werden über den Block-FK-Löser (Verfahren B) ergänzt. Sehr schnell, keine lokalen Minima, benötigt aber ≥ 2 Marker pro Gelenk.

8.3 Verfahren B — `sequential_fk`

Blockweiser nichtlinearer Ausgleich entlang der Kette. Für Block B mit Variablen $\mathbf{q}_B$ und Markern $\mathcal{M}_B$ werden die vorhergehenden Blöcke $\mathbf{q}_{<B}$ fixiert und

$$\mathbf{q}_B^* = \arg\min_{\mathbf{q}_B} \sum_{m \in \mathcal{M}_B} \rho_\delta\big(\,\|\mathbf{r}_m(\mathbf{q}_{<B}, \mathbf{q}_B)\|\,\big) \tag{10}$$

mit Huber-Verlust ρ_δ gelöst (Levenberg–Marquardt). Da die führende Variable großwinklig sein kann, wird ein *Multi-Start* über $\{0, 60, \ldots, 300\}^\circ$ durchgeführt und die Lösung mit den kleinsten Kosten gewählt. Funktioniert bereits mit einem einzelnen Marker je Glied, weil Gleichung (8) auch die Normale einbezieht.

8.4 Verfahren C — `global_ba`

Globales Bündelausgleichsproblem über *alle* Gelenkvariablen gemeinsam:

$$\mathbf{q}^* = \arg\min_{\mathbf{q} \in \mathbb{R}^7} \sum_{m \in \mathcal{M}} \rho_\delta\big(\,\|\mathbf{r}_m(\mathbf{q})\|\,\big). \tag{11}$$

Da alle Marker simultan einfließen, bestimmen die Fingermarker rückwärts auch die markerlosen Gelenke b (Hand) und c (Palm) — ein verdecktes Mittelglied wird durch nachgelagerte Marker überbrückt. Als Startwert dient die Lösung von Verfahren B, was lokale Minima bei großen Winkeln vermeidet. ρ_δ (Huber, Skala `huber_delta_mm`) dämpft Ausreißer-Marker.

8.5 Verfahren D — `hybrid` (Standard)

Die empfohlene Kombination: sequenzieller Initialwert (B) gefolgt von globalem Refinement (C). Sie vereint die Robustheit des blockweisen Vorgehens gegen lokale Minima mit der Genauigkeit und der Lückenüberbrückung des globalen Ausgleichs und ist im Benchmark durchgehend am stabilsten.

9 Beobachtbarkeit und Konfidenz

Aus dem Verhältnis sichtbarer Marker zu Variablen je Block, $\kappa_B = |\mathcal{M}_B^{\text{obs}}|/|\mathbf{q}_B|$, wird pro Gelenk eine Konfidenz abgeleitet:

$$\texttt{high}\ (\kappa \geq 2), \quad \texttt{medium}\ (\kappa \geq 1), \quad \texttt{low}\ (\kappa < 1,\ \text{unterbestimmt}), \quad \texttt{none}\ (0\ \text{Marker}).$$

Sie steht in `robot_state.json` und ist sicherheitsrelevant: Gelenke mit `low`/`none` sollten nicht ungeprüft angefahren werden. In solchen Fällen kann durch Drehen der Achse a (Arm2) die Hand aus weiteren Perspektiven sichtbar gemacht werden (Multi-Pose), womit die Beobachtbarkeit steigt.

10 Konfiguration über robot.json

`robot.json` beschreibt zugleich das kinematische Modell, die Markerpositionen, die Rendering-Parameter und die Algorithmuseinstellungen. Die wichtigsten Abschnitte:

- `units` — Längen-/Winkeleinheiten (mm / Grad).
- `vision_config` — ArUco-Wörterbuch und `MarkerSize`.
- `links` — die kinematische Kette: pro Link `parent`, `mountPosition/Rotation`, `jointToParent` (`type`, `variable`, `axis`, `origin`) und `markers` (`id`, `position`, `normal`, `size`).
- `renderingInfo` — Kamera/Render-Defaults (`width`, `height`, `dofFStop`, ...) für den Blender-Generator.
- `pose_estimation` — Algorithmussteuerung (siehe unten).
- `robot_test_poses`, `test_camera_positions/targets` — Vorgaben für den Szenengenerator.

Der Block `pose_estimation` steuert Stufe 4:

```
"pose_estimation": {
  "method": "hybrid",            // sequential_vector | sequential_fk | global_ba |
    hybrid
  "marker_observation": "corner_pose",  // oder "center_point"
  "use_normals": true,
  "normal_weight": 100.0,        // Gewicht w_n der Normalenresiduen
  "robust_loss": "huber",        // robuste Verlustfunktion
  "huber_delta_mm": 8.0,         // Skala delta des Huber-Verlusts
  "max_iterations": 200,
  "min_cameras_per_marker": 2,   // Mindestkameras zur Triangulation
  "per_link_method": {}          // optional: Verfahren je Glied
}
```

Wichtige Optionen. `method` wählt das Verfahren (A–D). `normal_weight` ist der zentrale Stabilitäts-Parameter: höhere Werte stabilisieren sicht-arme Posen deutlich, da die gemessene Normale stärker einbezogen wird; zu hohe Werte ($\gtrsim$300) lassen die Orientierung die Position überstimmen. Empirisch ist $w_n = 100$ ein robuster Kompromiss. `huber_delta_mm` bestimmt, ab welchem Residuum (in mm) ein Marker als Ausreißer gedämpft wird. `marker_observation` schaltet zwischen den Eckposen (`corner_pose`, empfohlen) und reinen Mittelpunkten (`center_point`) um.

11 Aufruf und Verwendung

Gesamte Pipeline (Detektion bis Gelenkwinkel) für einen Bildordner:

```
cd run
.\run_pipeline.bat ..\data\simulation\Scene8
# erzeugt data/evaluations/Scene8/robot_state.json
```

Einzelne Stufen (z. B. zum Experimentieren mit dem Verfahren):

```
python pipeline/3b_corner_marker_poses.py --evalDir data/evaluations/Scene8 --robot
    data/robot/robot.json
python pipeline/pose_estimation.py data/evaluations/Scene8/aruco_marker_poses.json \
       -robot data/robot/robot.json --method hybrid -out data/evaluations/Scene8/
    robot_state.json
```

Validierung gegen Grundwahrheit und **Benchmark**:

```
python benchmark/eval_pose.py data/evaluations/Scene8/robot_state.json data/simulation
    /Scene8/pose.json
python benchmark/run_benchmark.py --scenes 4 5 6 7 8 9 9a 9b 11 12
python benchmark/stage0_corner_normals.py --evalDir data/evaluations/Scene5 \
       --gt data/simulation/Scene5/render_a.json
```

Szenengenerator (Blender, erzeugt Bilder + npz + Grundwahrheit):

```
cd run
.\run_sceneGenerator.bat --poses 8     # nur Pose 8 rendern
```

Visualisierung. `run/robot_viewer.html` im Browser öffnen, `robot.json` und `aruco_marker_poses.json` laden. Der Observed-Normals-Pfeil zeigt die gemessene Normale, eingefärbt nach Winkelabweichung zur Modellnormale (grün $< 2°$ / gelb 2–5° / rot $> 5°$).

12 Validierung und Ergebnisse

12.1 Methodik

Die Validierung nutzt synthetische Szenen aus einem Blender-Generator (`run/run_sceneGenerator.bat`). Dieser rendert pro Pose $C = 7$ Ansichten und schreibt neben Bildern (`render_*.png`) und Intrinsiken (`render_*.npz`) zwei Grundwahrheits-Dateien: `render_*.json` mit den wahren Marker-Weltposen und `pose.json` mit den wahren Gelenkwinkeln. Drei Werkzeuge vergleichen die Rekonstruktion gegen diese Wahrheit: `stage0_corner_normals.py` (Normalenfehler der Triangulation), `9_evaluateMarker.py` (Positions-/Normalenfehler je Marker und Glied) sowie `eval_pose.py` (Gelenkwinkelfehler). Der Treiber `run_benchmark.py` aggregiert über Szenen und Verfahren.

12.2 Ergebnisse

Über zehn simulierte Posen mit bekannter Grundwahrheit (eine fehlerhaft gerenderte Pose ausgeschlossen) ergeben sich die folgenden mittleren Gelenk-Winkelfehler. Die Standardabweichung über die Szenen ist das Maß für die *Stabilität*.

Verfahren	Mittel [°]	Std [°]	Schlechtest. [°]	Mittel [mm]	Schlechtest. [mm]
`sequential_vector`	0,315	0,128	1,717	0,144	0,712
`sequential_fk`	0,434	0,171	1,838	0,158	0,851
`global_ba`	0,253	0,134	1,568	0,103	0,390
`hybrid`	**0,253**	**0,134**	**1,568**	**0,103**	**0,390**

Alle Verfahren liegen unter 0,5° Mittelwert; `hybrid`/`global_ba` sind am genauesten und am stabilsten und rekonstruieren auch die markerlosen Gelenke b, c rein aus den Fingermarkern.

13 Grenzen und Ausblick

- **Intrinsik.** Da die Board-Marker koplanar sind, ist die Kamerantiefe schlecht konditioniert; ein fehlerhaftes f_y (z. B. ein früherer Brennweitenfehler von ∼15 % bei 16:9) verfälscht alle Höhen.

Korrekte npz-Intrinsiken sind Voraussetzung.

- **Sicht-arme Posen.** Sind zu wenige Marker eines Gliedes sichtbar, ist der zugehörige Winkel schlecht bestimmt (`confidence=low`). Abhilfe: Multi-Pose durch Drehen der Achse a.
- **Adaptive Normalengewichtung.** Ein festes w_n ist ein Kompromiss; eine pro-Marker nach Sichtqualität skalierte Gewichtung könnte Best- und Worst-Case gleichzeitig optimieren.
- **Reale Kameras.** Stufe 1 nutzt derzeit dieselbe Intrinsik-npz für alle Bilder (Simulation: identische Kameras); reale Aufbauten benötigen je Kamera eine eigene Kalibrierung.