InfiniteFoundry
3D Digital Plant

TECHNICAL CHALLENGE

Computer Vision & Robotics Engineer

Bruno Scholles Soares Dias

October 28, 2025

Technical Report

1 Usage

The usage instructions can be found in the README file.

2 Technical Methodology

This section details the methodology for recognizing the four specific actions. The solutions are presented in their developmental order, not their operational sequence, to articulate the problem-solving progression.

Although the "Makes a marking (scratch)" solution was applicable to some other tasks, **distinct techniques were deliberately chosen. This decision was made, as it is recommended to use creativity to solve the proposed problems, and therefore I wanted to demonstrate the ability to solve the challenges in different ways**. I also understood that the solutions should be based only on the provided data. Since the dataset exclusively contained hand annotations, it was inferred that no additional annotations or model training were intended beyond hand detection. Consequently, the developed methods rely solely on the available hand detection information.

The specific threshold values used were empirically derived from Fig. 1. This graph plots the distance between hand centroids over time (representing the first ≈ 30s) and serves as the empirical basis for justifying the model parameters.

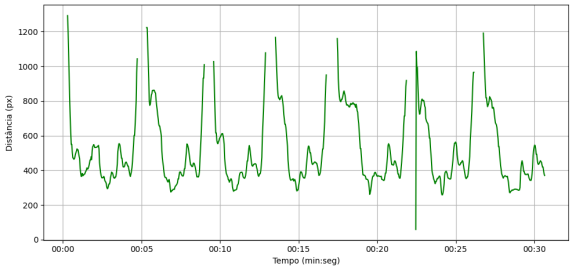

Figure 1: Distance (px) between hand centroids as a function of time (min:sec). This plot, representing approximately the first 30 seconds of the video, served as the empirical basis for defining decision thresholds.

2.1 Hand Detection

The hand detection module utilizes a YOLOv12n (Nano) architecture, fine-tuned using transfer learning on the provided dataset for 50 epochs. The final model weights were selected from the best validation epoch (Epoch 45), which achieved a Mean Average Precision (mAP) of 0.9047 (mAP_{50-95}). Key metrics for this epoch are detailed in Table 1.

Table 1: Key Validation Metrics for the Optimal Model (Epoch 45)

Epoch	Precision	Recall	mAP_{50}	mAP_{50-95}	Box Loss	Class Loss	DFL Loss	VRAM (Train) [GB]	VRAM (Inference) [GB]
45	0.9996	1.0000	0.9950	0.9047	0.4988	0.2508	0.9775	6.360	1.493

During inference, a post-processing heuristic differentiates the hands based on spatial location. If two hands are detected, they are classified as `hand_Left` or `hand_Right` by sorting their horizontal centroid coordinates. Based on the observation that the right hand remains persistently in the whole video, a single detected hand defaults to `hand_Right`. This classification enables downstream logic, as visualized in Fig. 2.

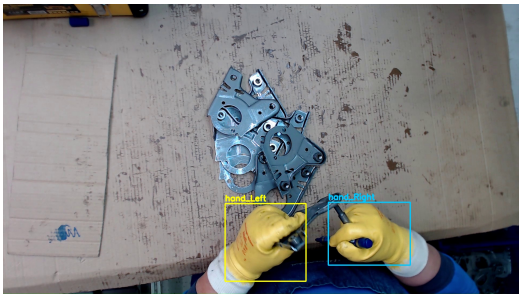

Figure 2: Example of the hand detection and left/right classification heuristic applied during inference.

2.2 Piece-in-the-Box Detection

The detection of the act of putting the pieces into the box was the first solution developed. A visual analysis of the operational video revealed a consistent pattern: the operator's left hand invariably exits the camera's field of view (FoV) to deposit the piece. This observation is corroborated by inter-hand proximity graphs, where the distance metric becomes undefined during this action.

Programmatically, this is interpreted as a state transition from two detected hands to one. The `count_piece_in_box` function increments a counter (`only_one_hand_frames`) for consecutive frames where only the right hand is visible. If this state persists for 14 frames, the event is registered: the main `counter` is incremented, logging that one piece entered the box, and a 2-second refractory period (`cooldown_frames`) is initiated to prevent spurious re-detections.

Figure 3 visually documents this sequence. The frames illustrate the left hand's movement, its disappearance from the FoV, and the subsequent activation of the "PIECE IN THE BOX" visual alert.

Figure 3: Temporal sequence illustrating the "Piece in the Box" detection logic. The operator's left hand moves towards the container, exits the camera's field of view, triggers the event recognition, and activates a visual alert.

An alternative strategy was tested, based on the left hand entering a specific Region of Interest (ROI) while distant from the right, **but it was discarded due to a number of false positives**. The "hand disappearance" heuristic was thus adopted as the simplest and most robust solution. It must be emphasized that this method's efficacy is intrinsically coupled to the static camera pose, so any modification to the FoV would invalidate the detection logic.

2.3 Piece Pick-up Detection

The piece pick-up detection methodology employs a virtual tripwire technique, an approach derived empirically and validated via positional data graphs. The core logic, detailed in the code, defines a static horizontal *pick line* ($PICK\_LINE\_Y = 715$). A pick-up event is registered when the centroid of the operator's left hand (yellow bounding box) traverses this line in a downward direction (moving from above to below), a motion correlating with the physical acquisition of a component.

While this heuristic demonstrated generally robust performance, it was susceptible to false positives. This limitation stems from the difficulty of calibrating a single, optimal y-coordinate threshold for all operational variations; manual manipulation proved unable to ascertain an ideal, error-free value. The detection sequence is visualized in Fig. 4.

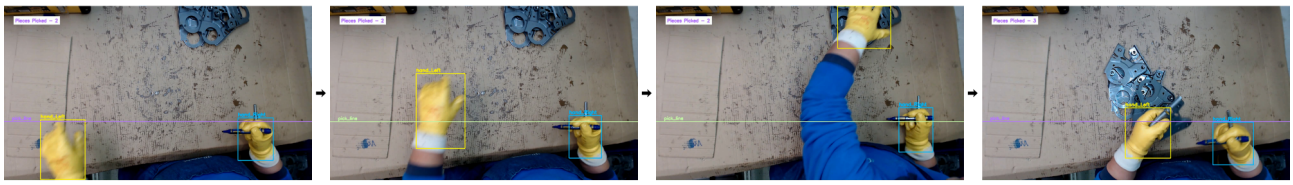

Figure 4: Sequential visualization of the pick-up detection logic. The system tracks the left hand (yellow bounding box) relative to the horizontal *pick line*. The "Pieces Picked" counter increments (from 1 to 2) as the hand completes the action relative to the line.

An alternative strategy, analogous to the pen-marking methodology (Task 3) and reliant on positional graph data, was considered. Although that approach might be more efficient for timing complete operational cycles, the implemented line-crossing solution was deemed superior for the specific objective of event counting, despite its need for refinement. This method was also selected to intentionally introduce a methodological divergence from the pen-marking task's detection logic.

2.4 Pen Marking (Scratch) Detection

The algorithm for this task was a hybrid applied methodology, combining empirical analysis with detailed observation of the signal that represents the Euclidean distance between the centroids of the two hands over time.

By analyzing slow-motion video alongside this distance graph, I saw a distinct pattern: the operator's subtle movements are highly repetitive and consistent during the marking action. This regularity is visualized in Figure 5, which plots the centroid distance (pixels) versus time (min:sec). The red segments highlight the consistent, low-amplitude oscillatory pattern occurring during the mark.

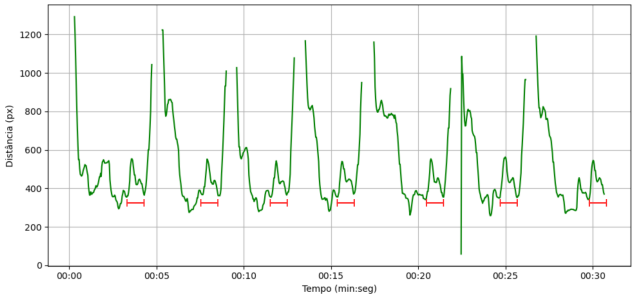

Figure 5: Graph of the distance (in pixels) between the centroids of the two hands as a function of time (min:sec). The red segments highlight the consistent, low-amplitude oscillatory patterns identified during the pen marking (scratch) action.

Based on this observation, a state-based detection logic was implemented. The algorithm requires both hands to be present and the distance between their centroids, $d(\mathbf{c}_L, \mathbf{c}_R)$, to be below a threshold. A critical precondition is the stability of the left (piece-holding) hand, verified by ensuring the mean standard deviation of its centroid position (over a window) is below a low pixel threshold. While the left hand is stable, the "scratch" motion is identified if the distance history exhibits a specific oscillatory behavior (a defined number of sign changes in its derivative) confined within a low amplitude.

The detection logic is summarized by the following simultaneous conditions:

$$\text{Detection}(t) = \begin{cases} \text{True}, & \text{if } (d(t) \leq \tau_d) \wedge (\text{Stab}(H_L) < \tau_s) \wedge (\text{Osc}(H_D)) \\ \text{False}, & \text{otherwise} \end{cases}$$

where $d(t)$ is the distance, H_L and H_D are the recent histories of the left-hand centroid and the distance, τ_d is the max distance threshold (500 px), τ_s is the stability threshold (10 px), $\text{Stab}(\cdot)$ calculates the mean standard deviation, and $\text{Osc}(\cdot)$ verifies the specific oscillatory signature (amplitude ≤ 120 px and $2 \leq$ sign changes ≤ 6).

A significant limitation must be noted: the operator makes two distinct marks, but the algorithm consistently fails to detect the second. This movement is extremely subtle, and its signal is not distinct enough to be captured. No alternative method was devised for this failure. Therefore, as a programmatic compromise to match the ground truth, the counter is manually incremented by two upon a single successful detection. This solution assumes the operator always performs two marks, introducing a potential error if only one is made.

2.5 Probe Pass Detection

I initially considered applying the same methodology described in the previous section, which involved analyzing the distance graph. A noticeable drop in distance could potentially indicate the probe pass action. However, this strategy did not yield results as consistent as those obtained for the penmark detection. Consequently, I decided to approach this problem differently. To implement this alternative solution, I structured the logic into two sequential stages, designated as Phase 0 and Phase 1. **It should be noted that the algorithm was specifically calibrated to the standardized motions observed in the reference video. Therefore, any substantial deviation in operator movement or timing may adversely affect its performance.**

2.5.1 Phase 0: Initial Contact Detection

This first stage identifies the initial probe pass from the right to the left hand, activating when both hand centroids are close ($\leq$ 500 pixels). A detection is registered when the "hand_Right" bounding box's top-left corner intersects (or is within 15 pixels of) the "hand_Left" box, as shown in Figure 6. This event confirms contact and immediately transitions the system to Phase 1.

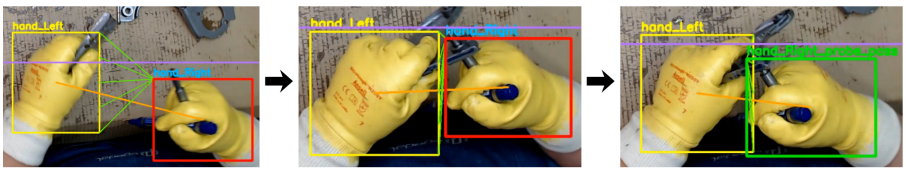

Figure 6: The "Probe Pass" Phase 0 detection sequence. The system monitors the hands (yellow, red) until the "hand_Right" box (now green) overlaps the "hand_Left" box, triggering a count.

2.5.2 Phase 1: ROI-Based Motion Detection

Phase 1 is designed to detect the second action in the sequence, which is the insertion (or removal) of the probe. Following the Phase 0 detection, I implemented a short delay window (14 frames) to allow the "hand_Right" to stabilize in its new position.

After this delay, a small Region of Interest (ROI) is activated. This ROI is dynamically centered on the top-left corner of the "hand_Right" bounding box. The core logic of Phase 1 is to detect significant motion within this specific ROI, as illustrated conceptually in Figure 7.

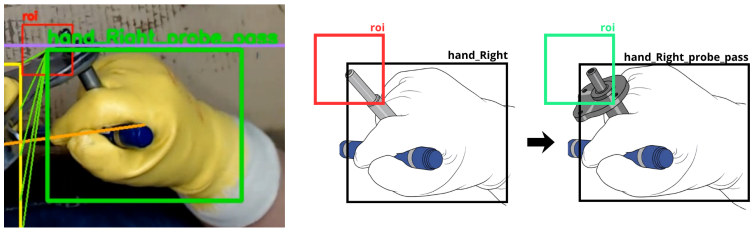

Figure 7: Conceptual illustration of Phase 1. After Phase 0 (contact), the system activates an ROI (red box) to monitor the probe's movement. When motion is detected (insertion/removal), the state is updated (green box).

To detect this motion, I use a simple frame subtraction method. The system compares the current frame's ROI (F_t) with the previous frame's ROI (F_{t-1}). A motion score (M) is calculated based on the pixel-wise absolute difference, which can be represented as:

$$M = \sum |F_t(x, y) - F_{t-1}(x, y)|$$

If this motion score M exceeds an experimentally determined threshold (set to 25), the system registers a Phase 1 detection.

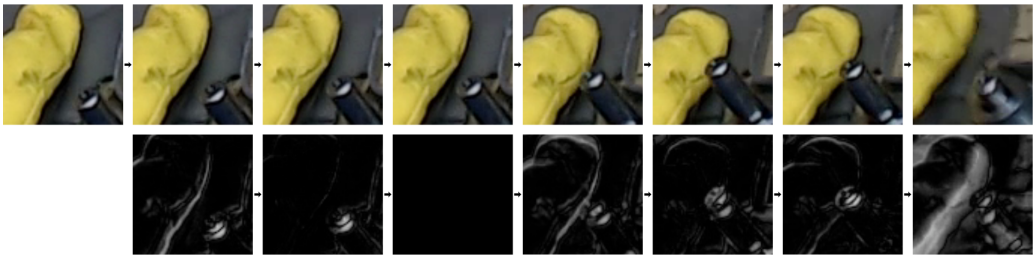

Figure 8: Visualizing the Phase 1 motion detection logic. The top row shows the image sequence from the ROI. The bottom row shows the corresponding pixel-wise difference (motion). A detection (final frame) is triggered by the significant change caused by the probe's movement.

Figure 8 demonstrates this process: the top row shows the ROI's visual feed, and the bottom row displays the frame subtraction result, capturing the significant pixel change from the probe's insertion or removal. I must emphasize a critical limitation: the system detects motion, not the probe itself. Since only hands are tracked, the algorithm cannot differentiate between insertion and removal, registering both as a successful "Phase 1" detection.

This method is sensitive to any motion within the ROI; consequently, operator hand stabilization after the 14-frame delay (but before the probe action) can cause a false positive, though the delay helps mitigate this. Finally, the ROI is not monitored indefinitely. I implemented an abort condition: if the "hand_Right" moves away from "hand_Left" (horizontal distance > 150 pixels) before motion is detected, Phase 1 is canceled, and the system resets.

3 Results

Figure 9 summarizes the end-of-video metrics for two inference runs of the same recording executed on a remote server with two different GPUs, an **NVIDIA RTX 3090** and an **NVIDIA RTX 4090**. I used this server because my personal computer is old and cannot train the models or even run inference reliably. The codebase, thresholds, and input video were identical across runs.

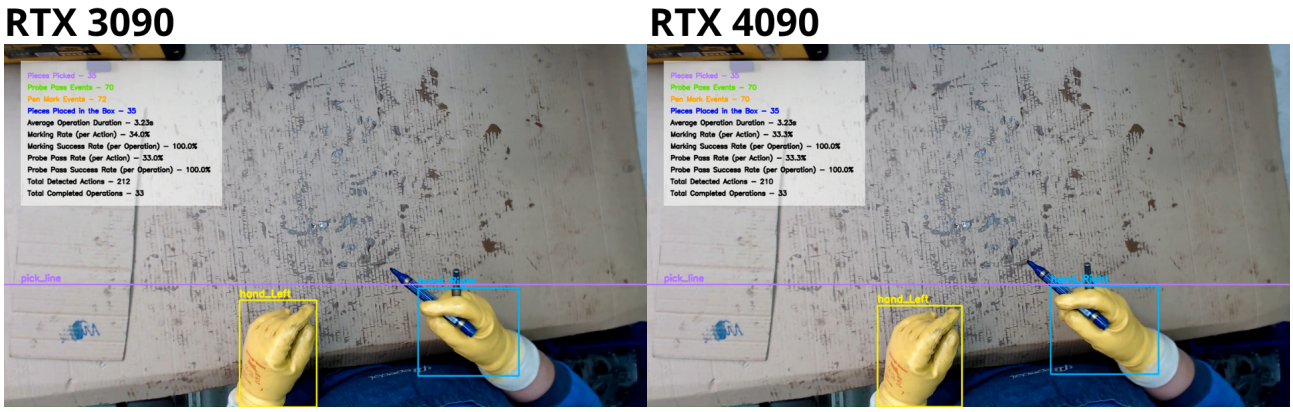

Figure 9: Side-by-side dashboard of computed metrics at the end of the video analysis on two GPUs, RTX 3090 (left) and RTX 4090 (right).

The high-level outcomes are consistent across GPUs. In both runs, the **'Pieces Picked' (35)** and **'Pieces Placed' (35)** counters matched, indicating reliable tracking of operation start and end events at the cycle level. Likewise, both the **Marking Success Rate (per Operation)** and the **Probe Pass Success Rate (per Operation)** reached **100.0%**, which confirms that every completed cycle included the required *Pen Mark* and *Probe Pass* sub-tasks. The system also reported a stable **Average Operation Duration** of **3.23 s** in both executions.

However, small cross-GPU differences appeared in some action-level tallies. The **RTX 3090** run reported **72** *Pen Mark* events and **212** total detected actions, whereas the **RTX 4090** run reported **70** *Pen Mark* events and **210** total detected actions. The *Probe Pass* count remained stable at **70** in both runs, and the system registered **33** *Total Completed Operations* in both cases. Consequently, the action-level rates converged, as expected, to approximately **33.33%** for each action among the four types, with the 3090 run showing a slightly higher *Pen Mark* rate due to the two extra marks.

Two factors help explain the observed behavior:

- **Pairing sensitivity for completed operations.** Manual inspection indicates that the metric *Total Completed Operations*, defined as the sequential pair *'Pieces Picked'* → *'Pieces Placed'*, remains vulnerable to local detection errors in either counter. As discussed previously, approximately two false positives and two false negatives in the `Pieces Picked` detector can cancel in the final pick tally yet still desynchronize the pairwise matching. This desynchronization explains why the system reports **33** completed operations rather than the **35** visible in the ground truth, even though picks and placements each sum to **35**.

- **Cross-GPU discrepancies in action-level counts.** I did not have time to debug the GPU-dependent differences in depth. My working hypothesis is that temporal logic in the code, such as frame sampling, debouncing windows, cooldowns, and integration thresholds, interacts with the video processing throughput. On the **RTX 4090**, the pipeline processes frames substantially faster, nearly twice as fast as on the **RTX 3090**. This difference in throughput likely shifts which frames fall inside or outside the temporal windows that trigger an action, particularly for *Pen Mark*, where the function `detect_pen_mark` increments the counter by `+2` per detected event to account for the back and forth motion. In the **RTX 3090** run, one abrupt tool movement appears to have been interpreted as a valid marking pattern, adding **+2** to the total and yielding the **72** count. On the **RTX 4090**, the higher throughput likely altered the frame-wise evidence entering these windows, preventing that spurious trigger and yielding **70**. The choice to increment by two amplifies the impact of any false positive and should be revisited.

In summary, operation-level metrics are good across hardware, while small action-level discrepancies arise between **RTX 3090** and **RTX 4090** runs. The most plausible cause is the interaction between GPU-dependent frame throughput and the temporal thresholds present in the implementation. Further investigation should decouple event logic from raw frame rate, for example by operating on timestamps rather than frame indices and by harmonizing cooldowns to real time.

4 Insights and Ideas

The detectors were intentionally constrained to the provided hand annotations, consequently, their reliability depends on a fixed camera pose and on the standardized motions present in the reference video. Under the same viewpoint and workflow, performance is strong; generalization to a new operator is less certain if kinematics or layout change. On a factory floor, however, operators are typically trained for repeatable actions under stable viewpoints, therefore comparable results are plausible when these conditions hold. I also chose distinct strategies across the tasks to illustrate that different problems can be solved by different, creative approaches; in a production setting, a more uniform design with shared primitives and centralized configuration would be preferable.

Small discrepancies observed between the RTX 3090 and RTX 4090 suggest sensitivity to processing throughput, CUDA stack versions, and effective parallelism. Temporal rules expressed in frames can drift with frame-rate variation.

The cycle plots reveal strongly periodic behavior, with a stable operation duration, which indicates concrete opportunities for improvement and extension:

- **Robotic assistance and automation**, supported by the empirical plots that reveal highly periodic and standardized cycles, which makes the workflow readily automatable. Similar routines have probably already been applied in comparable industrial operations.
- **Multi-camera fusion**, adding complementary viewpoints to reduce occlusions, increase confidence, and enable basic triangulation of hand centroids.
- **3D pose and a lightweight digital twin**, estimating upper-limb keypoints and linking them to a simplified 3D workstation model for kinematic analysis and replay.
- **Quality, traceability, and analytics**, converting event streams into per-lot dashboards that track cycle counts, dwell times, and outliers, with basic proxies for defective pieces.
- **Standardization and domain adaptation**, stabilizing camera pose and lighting, then applying small viewpoint-specific adjustments or limited fine-tuning when variation is unavoidable.

Given the time constraints and dataset scope, I focused on hand detection without adding new annotations. Some components could likely be improved with multi-view sensing or richer labels, and with a bit more development time I would expect to correct the occasional errors observed. The primary objective was to demonstrate viable, well justified solutions for each task. Overall, the project was technically engaging, and I found the challenge genuinely enjoyable.