Improved Confident cell Balloon Detection and Motion Prediction

Tongshu Wu

Code Repository

The full source code and supporting scripts for this project are available on GitHub at:

- **Main Project Directory:** `https://github.com/TongshuWu1/Tony_Swarm/tree/3298cbf0c3b08dae1506b79263ComputerVision`
- **Detection Logic and Runtime Scripts:** `https://github.com/TongshuWu1/Tony_Swarm/tree/main/ComputerVision/data/src`
- **Gaussian Sampling and Gain Tuning:** `https://github.com/TongshuWu1/Tony_Swarm/tree/main/ComputerVision/get_gain`

This repository includes all firmware logic, tracking modules, LAB-based color sampling tools, and calibration scripts necessary to reproduce the results discussed in this report.

1 Problem Statement

This project addresses the challenge of **real-time balloon detection and tracking** on an embedded vision system with **severe hardware constraints**. The objective is to develop a robust system that can reliably detect a specific balloon and maintain consistent tracking under **occlusions**, **lighting variation**, and the presence of multiple distract objects—all while running on the **Arduino Nicla Vision**, a compact camera module with very limited processing power and memory.

Key part of the competition involves detecting and tracking green or purple balloons that serve as scoring targets. These balloons are lightweight and made of shiny aluminum film, which causes their appearance to change drastically under different lighting conditions. In addition, they often move unpredictably and can become briefly hidden behind obstacles, making visual tracking difficult.

Figure 1: Sample target balloon

This project aims to develop a vision system that can **detect the correct balloon**, **stay locked on the target**, and **maintain stability** even when other objects enter the scene or the balloon is briefly occluded. Beyond the competition, this work contributes to more general goals in embedded vision, drone tracking, and multi-robot coordination.

This project will enhance our detection system as part of the **Swarms Lab** project for the robotic competition **"Defend the Republic"**. DTR, a robotic competition hosted by various universities, involves autonomous robot blimps competing in perception and behavior challenges where we simulate robot search and rescue operation. (For more information, visit: `https://sites.google.com/lehigh.edu/dtr2023b/home`). Detecting and tracking a balloon accurately in a dynamic and partially occluded environment is a key challenge in autonomous robotic systems.

2 Previous Work

In earlier versions of our system, we used a **color-based confidence grid** to detect the balloon. The frame was divided into a **fine-resolution grid**, and each cell's average LAB value was compared to a predefined color profile of the balloon. If a cell's color closely matched the target, it was marked as high confidence.

The balloon's location was then estimated by computing the **centroid of all high-confidence cells**. While this approach worked in simple cases, it had several major weaknesses:

1. It did **not consider blob shape or structure**, which made the position estimate unstable when noisy detections occurred elsewhere in the frame.
2. The system was easily confused by background colors that partially matched the balloon.
3. Because it used a fine grid, computation was slower and more sensitive to small changes.

In addition, the tracking algorithm selected the **largest detected blob** as the target. This size-prioritized strategy often failed when unrelated objects—like hands, obstacles, or other balloons—appeared in the scene. The tracker would frequently jump to a new blob, causing the system to lose the correct balloon.

These problems were made worse by the reflective aluminum surface of the balloons, which caused their LAB values to vary significantly depending on angle and lighting. As a result, the old system produced **jittery and inconsistent tracking**, especially in cluttered or dynamic environments.

These limitations motivated the new design in this project, which uses:

1. A **coarser grid** to reduce noise sensitivity as the first layer detection
2. **Blob detection** with mask + morphological operation + contour detection on high confident grid as second layer instead of centroid averaging,
3. **Score-based blob selection** using color confidence × area,
4. And **motion prediction** with a Kalman filter to maintain tracking during occlusions.

Other object tracking methods, such as deep learning-based detectors (e.g. YOLO) or optical flow, are commonly used in modern computer vision systems. However, they are not suitable for this project because the **Arduino Nicla Vision** has extremely limited computational power and memory. Deep learning models require large amounts of memory and often need GPU acceleration. Optical flow techniques, while not as heavy, are still too slow for real-time use on microcontrollers, especially when aiming for 7–10 FPS.

3 Dataset

All data for this project was collected using the **Arduino Nicla Vision** camera in real time. Instead of storing images or using an offline dataset, we used a lightweight and efficient script called `run_ab_sampling()` to gather large volumes of LAB color data directly from the live camera feed.

This function divides the camera's view into a **21×28 grid** and samples average A and B channel values (from the LAB color space) over a **3×3 block** near a user-selected horizontal position (`left`, `center`, or `right`). When a balloon is held in front of the camera, the function outputs 9 LAB pairs every 20 milliseconds which resulting in **thousands of high-quality color samples** across the balloon within seconds. This sampling method gives us a dense, up-to-date dataset that reflects how the balloon appears under current lighting and viewing conditions.

To ensure consistency across sessions, we used `get_gains.py` before each sampling run. This script enables auto white balance and exposure temporarily, reads the optimized gain values from the sensor, and locks them in for stable operation. This avoids needing to re-sample the balloon unless lighting changes significantly.

This setup allows us to:

1. Quickly gather thousands of color data points per balloon color (e.g., green or red),
2. Dynamically update our LAB Gaussian model in response to environmental changes,
3. Avoid the need for traditional, static datasets — which are impractical on the Nicla Vision due to limited storage and RAM.

Strengths of this dataset approach:

1. Adapts to **real-time lighting and sensor conditions**.
2. Produces **thousands of high-quality LAB samples** quickly.
3. Fits the **Nicla Vision's memory and speed limitations**.
4. Removes the need for complex image labeling or dataset management.

Limitations:

1. Only collects **color information**, not spatial annotations like object position or shape.
2. Requires manual balloon placement in the frame during sampling.
3. Needs to be repeated when tracking a new balloon type or working in a new environment.

Because we cannot use any **large machine learning models** on the Nicla Vision due to hardware constraints, we rely on a **Gaussian model in LAB space** to classify balloon color. This method is computationally efficient, easy to update, and works well with the streaming color samples collected using our sampling pipeline.

4 Some trail and error(Method I tried before getting to the Final solution)

Before settling on the final embedded vision pipeline for the Nicla Vision, I explored a range of detection strategies using OpenCV and Python on a desktop environment. One of the major early experiments involved using **LAB color-based Gaussian confidence mapping**, contour analysis, and grid-based scoring to detect and highlight the balloon in still images.

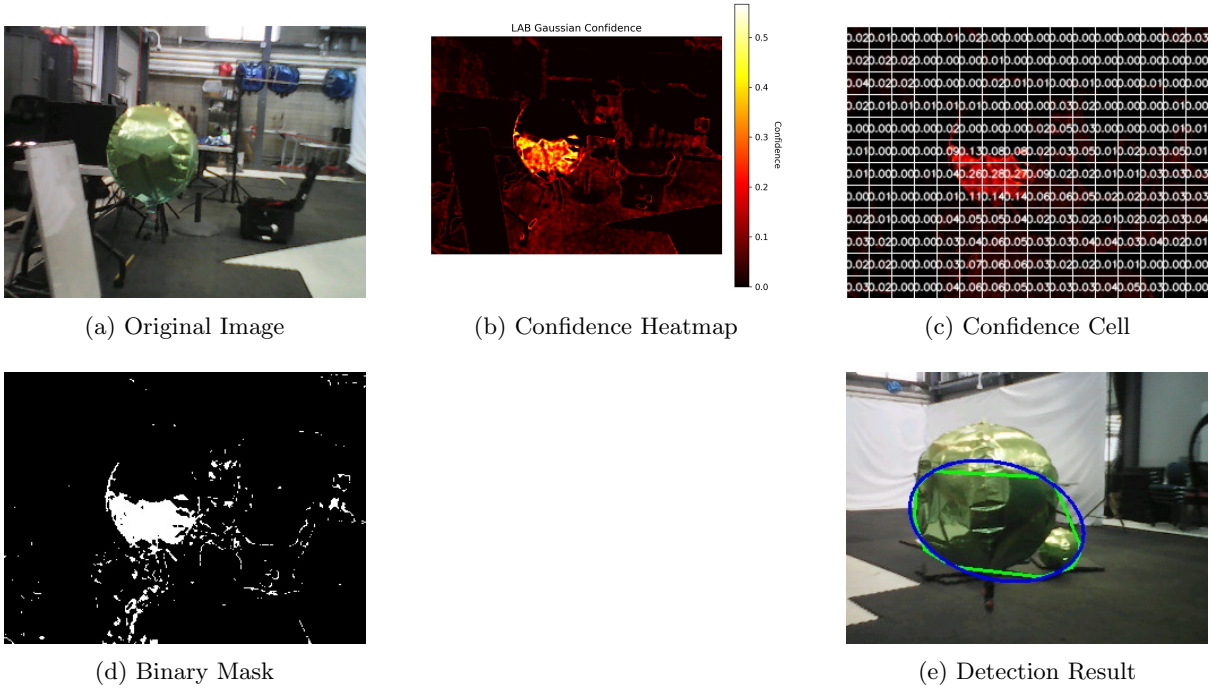
1. Converting RGB images to LAB format using OpenCV.
2. Defining two LAB center ranges corresponding to typical red balloon hues.
3. Applying a **custom Gaussian function** to each pixel to calculate its confidence based on color distance from the target LAB range.
4. Generating a **confidence map** and a **thresholded binary mask** for contour detection.
5. Dividing the image into a coarse grid (e.g., 5×5 cells), and computing **average confidence per cell**.
6. Scoring each detected contour by summing confidence across the cells it covered.
7. Selecting the contour with the highest confidence-weighted score, then drawing its convex hull and fitting an ellipse around it.

This version worked well on static images and gave me insight into how color-based scoring could be structured. It also helped validate the effectiveness of the **Gaussian confidence formulation**, which became a key component of the final solution on Nicla Vision.

However, this method had several key **limitations**:

1. It was designed for offline processing and required full-frame images stored in a directory.
2. It was too computationally intensive to run on the Nicla Vision, which lacks OpenCV, floating-point efficiency, and has limited RAM.
3. It was not real-time — processing a single frame could take several seconds.
4. It didn't support dynamic model updates or onboard data sampling, which are essential for deployment in variable lighting conditions.

Despite these drawbacks, this trial served as a valuable prototyping step. It allowed me to test confidence scoring, validate the Gaussian LAB model, and experiment with spatial weighting and scoring logic — all of which were later adapted into a more efficient, real-time form suitable for embedded use.



(a) Original Image (b) Confidence Heatmap (c) Confidence Cell

(d) Binary Mask (e) Detection Result

Figure 2: Stages of the green balloon detection pipeline

5 Current Approach

Our balloon detection system is designed as a three-layer pipeline optimized for real-time performance on the resource-constrained **Arduino Nicla Vision**. Each layer is responsible for a distinct stage of the tracking process: color confidence estimation, spatial detection via blobs, and temporal tracking using motion estimation.

1. LAB Gaussian Color Modeling

To identify the target balloon based on color, we use a **Gaussian model in the LAB color space**. Specifically, we focus on the A and B channels, which describe opponent color axes and are less affected by brightness and environmental changes than RGB.

The Gaussian model includes:

1. A **mean vector** of A/B values representative of the balloon color.
2. A **2×2 covariance matrix** capturing the expected color variation and correlation.

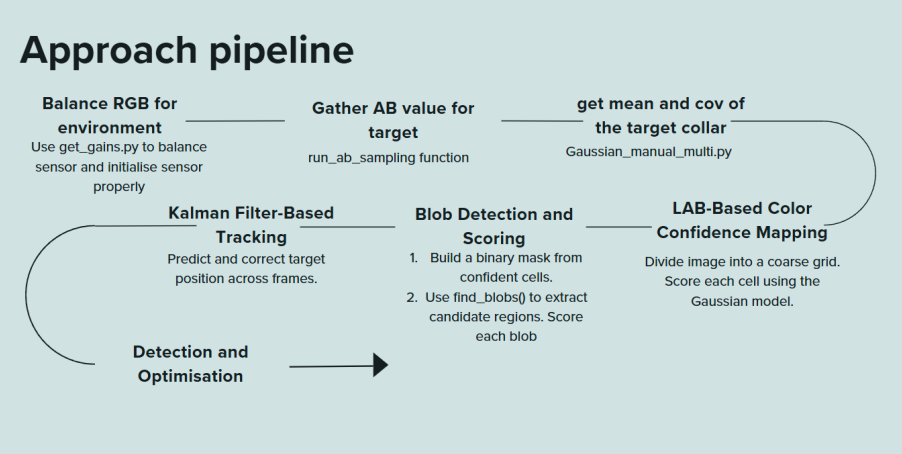


Figure 3: Apprroach Pipeline

The confidence score for a region is calculated using a modified Mahalanobis formula:

$$\text{Confidence}(a, b) = \frac{1}{1 + 0.5m + 0.125m^2}$$

where m is the squared Mahalanobis distance between the measured and modeled A/B values.

Data Collection: To train this model, we developed a script `run_ab_sampling()` that runs directly on the Nicla Vision. It:

1. Divides the frame into a 21×28 grid.
2. Focuses on a 3×3 region from the left, center, or right side.
3. Samples A/B values for each selected region using `get_statistics()`.
4. Outputs thousands of high-quality LAB tuples as the balloon is held in front of the camera.

We then use these samples to compute the Gaussian mean and covariance offline. The resulting model is hardcoded into the embedded script.

Why Gaussian?

- **Lightweight:** Fast to compute on low-power devices.
- **Robust:** Adapts to real-world color variation (reflection, angle).
- **Easily retrainable:** New color profiles can be collected in seconds.

2. Grid-Based Confidence Mapping and Blob Detection

Rather than scanning every pixel, we use a coarse grid to minimize computational load. The frame is divided into a 12×18 grid, with each cell representing a block of approximately 26×20 pixels at 320×240 resolution.

Layer 1: Confidence Grid Computation

1. For each cell, the average A and B LAB values are computed using `get_statistics()`.
2. These values are scored against the Gaussian model to obtain a confidence between 0 and 1.
3. If the confidence exceeds the threshold `CONFIDENCE_THRESHOLD` = 0.35/`SENSITIVITY`, the cell's score increases and is capped at 1.0.
4. If below the threshold, the confidence decays using:

$$c_{t+1} = c_t \times \texttt{DECAY}, \quad \texttt{DECAY} = 0.7$$

5. A grayscale binary mask is drawn in the same pass. High-confidence cells are filled directly into the mask using `mask.draw_rectangle()`.

Optimizations:

1. **Single-pass loop:** Both scoring and mask generation are handled in one traversal.
2. **Temporal smoothing:** Decay prevents rapid flickering due to momentary changes.
3. **Cell-level operation only:** No per-pixel color calculations are used.

Figure 4: Detection with Grid printed

Layer 2: Binary Mask and Blob Selection

1. The mask is cleaned using morphological `close()` and `open()` operations.
2. `find_blobs()` is used to extract blobs from the binary mask.
3. Each blob is scored as:

$$\text{Score} = \text{Blob Area} \times \text{Average Confidence of Overlapping Cells}$$

4. Geometric filters (aspect ratio, solidity, elongation) are applied to reject false positives.
5. The blob with the highest score is selected as the current balloon candidate.

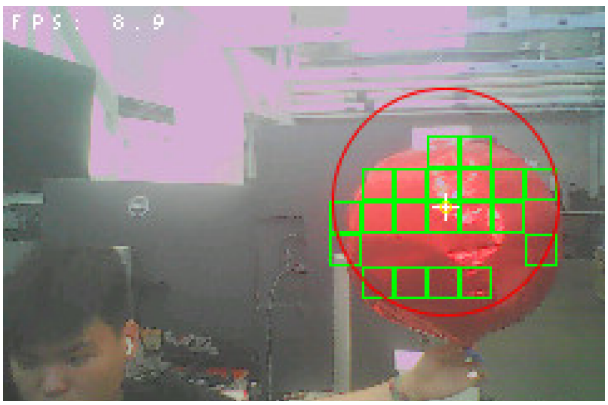


Figure 5: detection with contour and confident grid highlighted

3. Kalman Filter-Based Motion Tracking

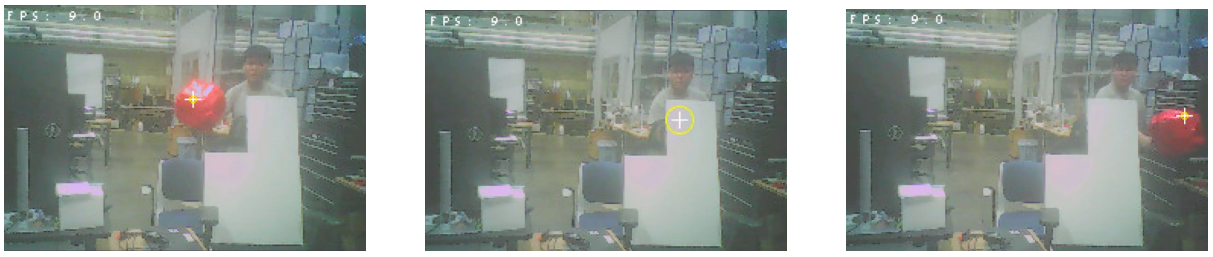
We use a standard Kalman filter with a 4D state vector $[x, y, \dot{x}, \dot{y}]$ to track balloon motion over time.

Prediction Step:

1. Predicts the next state based on velocity and past position.
2. Applies smoothing to velocity using `VEL_DECAY = 0.98` and `VEL_ALPHA = 0.6`.
3. Applies position smoothing using `POS_SMOOTH_ALPHA = 0.8`.

Correction Step:

1. When a new detection is available, the Kalman state is updated using the blob's center coordinates.
2. If no detection occurs, the system continues using prediction only.
3. If the uncertainty (covariance trace) grows beyond `MAX_COV_TRACE = 400`, the tracker resets.



(a) Tracking (b) Lose Track and predict (c) Correction

Figure 6: Sequential detection snapshots at three time points.

Visual Feedback:

1. The estimated position is marked with a white cross.
2. A yellow circle shows the current uncertainty radius.

This layer helps the system stay locked onto the target, even during temporary occlusions or missed detections.

Figure 7: Tracking red balloon

6 Experiment and comparison

To validate my tracking system, I will test the implementation on a **Nicla Vision Pro**, capturing real-time images of moving balloons. The **Nicla Vision Pro** is the primary vision module used in our robot, selected for its compact design and onboard processing capabilities. Initially, an **OpenMV camera** was considered as an alternative due to its ease of use and built-in vision features.

1. Detect a specific balloon and keep tracking the object.
2. Track its movement even when briefly obstructed by objects.
3. Maintain consistent tracking without switching to another balloon.
4. Compare predicted trajectories against actual motion to evaluate accuracy.

Result and Performance

(a) **Confidence Grid:** Efficient, smoothed detection heatmap using LAB Gaussian scoring.
(b) **Mask + Blobs:** Structured binary mask and geometric scoring improve robustness.
(c) **Kalman Tracker:** Maintains stability and continuity under motion and occlusion.
(d) **Frame Rate:** Sustains 8–10 FPS in live deployment scenarios.

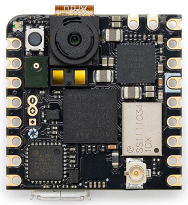


Figure 8: Nicla Vision Pro

Execution and Testing Environment

1. The detection algorithm (`run_detection()`) was deployed directly onto the Nicla Vision board and tested in real-world indoor and outdoor settings.
2. Balloons of different colors (primarily red, purple, and green) were presented at distances ranging from 0.5 m to 8 m.
3. Tests included scenarios involving partial occlusion, fast motion, and background clutter to simulate real competition settings (e.g., "Defend the Republic" robotic tasks).

Approaches Compared

1. **Previous Method: Fine Grid + Centroid Estimation**
   - The original system used a fine-resolution grid to detect color similarity.
   - The balloon center was estimated using the centroid of all high-confidence grid cells.
   - This method was sensitive to background noise and prone to target switching.
2. **Current Method: Coarse Grid + Confidence-Weighted Blob Detection**
   - Uses a coarser 12×18 grid for confidence scoring.
   - Extracts blobs from a binary mask and scores them based on a combination of area and color confidence.
   - A Kalman filter is used to maintain continuity over time, even when detections are temporarily lost.

Hyperparameter Tuning and Variants (All the Hyper param info are in the code comment section)

1. **Sensitivity Thresholds**:

   Sensitivity values between 1.5 and 3.0 were tested. A value of `2.7` was optimal for balancing detection strictness and robustness.

2. **Decay Rate**:

   Confidence decay (`DECAY = 0.7`) ensured gradual fading of stale detections, reducing flickering without abrupt loss of tracking.

3. **Blob Selection Filters**:

   Shape-based filters (aspect ratio, solidity, elongation) helped eliminate non-balloon detections such as hands or nearby objects.

4. **Kalman Filter Tuning**:

   Motion noise (`Q=3.0`) and measurement noise (`R=10.0`) parameters were adjusted to prevent over-reaction to jitter or noise.

Observed Advantages

1. **Reduced False Positives Without Neighbor Pruning**:

   Even without explicitly suppressing nearby cells or performing neighbor reduction, the system significantly reduced false positives in clear color conditions (especially red or purple balloons).

2. **Improved Target Locking**:

   Unlike the previous system, which could easily switch targets based on blob size, the current method maintains focus on the tracked target unless another blob shows significantly higher confidence. This is rare due to the confidence being accumulated over time.

3. **Kalman Filter Support During Occlusion**:

   The Kalman filter plays a critical role in maintaining a stable position estimate when the balloon is temporarily occluded or momentarily not detected. This allows the robot to continue acting based on the estimated target position, rather than freezing or switching targets unnecessarily.

Limitations: Green Balloon Detection in IR-Biased Conditions:
The Nicla Vision's infrared sensitivity causes white backgrounds to appear greenish, making true green balloons more difficult to distinguish. While performance is acceptable in controlled environments, robustness to white-green confusion remains a challenge.

7 What Did You Learn from This Project?

1. **Efficiency Over Complexity on Embedded Hardware:** Traditional deep learning methods are not feasible on platforms like the Nicla Vision. Instead, lightweight statistical model such as the LAB Gaussian confidence mapping can be highly effective when paired with careful optimizations.

2. **Sensor Behavior in Real-World Conditions:** The Nicla Vision's sensitivity to infrared revealed that bright white surfaces often appear greenish in LAB space, leading to false positives. This emphasized the need for validation and environment-specific tuning. I thought my approach is good enough to handle green balloon without neighbor cancellation, but it is not.

3. **Value of Lightweight Models:** The success of the Gaussian-based confidence system demonstrated that simple, explainable models—when optimized for hardware limitations—can deliver robust real-time performance without the need for complex inference engines.

4. **Blending Traditional CV and Data Treatment:** This project opened up new idea for me to combine traditional computer vision techniques such as morphological operation with statistical data treatment methods to enhance detection robustness. The blending of color-space modeling, grid-based abstraction, and Kalman filtering proved to be a powerful alternative to heavyweight learning-based approaches.

8 Future Work

1. **Neighbor Canceling for Background Noise Reduction:** One promising direction is implementing neighbor-based filtering. By adding model of common noise such as green background, it should further reduce false positives caused by isolated high-confidence regions in the background.

2. **Improved Shape Detection Techniques:** Currently, the blob selection relies on basic geometric properties like aspect ratio, solidity, and elongation. Integrating more advanced shape descriptors such as contour curvature analysis or convexity defects could help improve target discrimination.

3. **Hardware Scaling for Higher Accuracy:** As more powerful embedded cameras become available (e.g., those with onboard AI accelerators or higher resolution), the current pipeline can be scaled up. OR! Maybe we should try event camera.

References

[1] MIT, *Kalman Filter*, Available at: `https://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf`

[2] MathWorks, *Kalman Filtering Guide*, Available at: `https://www.mathworks.com/help/control/ug/kalman-filtering.html`

[3] Richard Szeliski, *Computer Vision: Algorithms & Applications*, Springer.

[4] Embedded Vision Team, *Embedded Vision with Nicla Vision Pro*, Source.

[5] P. C. Mahalanobis, *On the generalized distance in statistics*, Proceedings of the National Institute of Sciences of India, Vol. 2, No. 1, pp. 49–55, 1936.

[6] Swarms Lab, *Blob Detection and Tracking Project*, GitHub repository. `https://github.com/LehighBlimpGroup/Blob-detection-and-Tracking`

[7] Bharath Bhatt, *Getting Confidence Estimates from Neural Networks*, April 2021. `https://bharathpbhat.github.io/2021/04/04/getting-confidence-estimates-from-neural-networks.html`