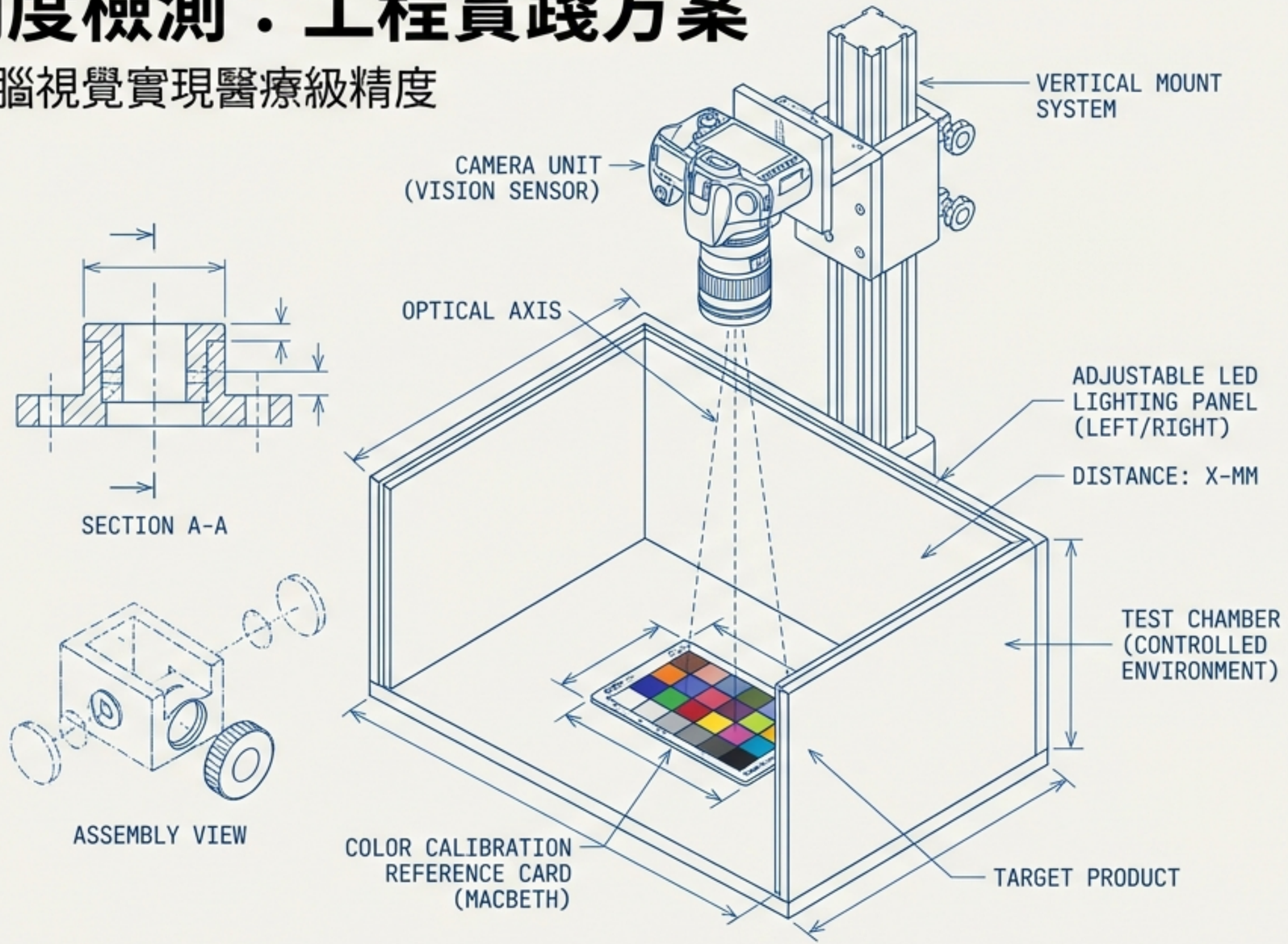
自動化色差與透明度檢測：工程實踐方案

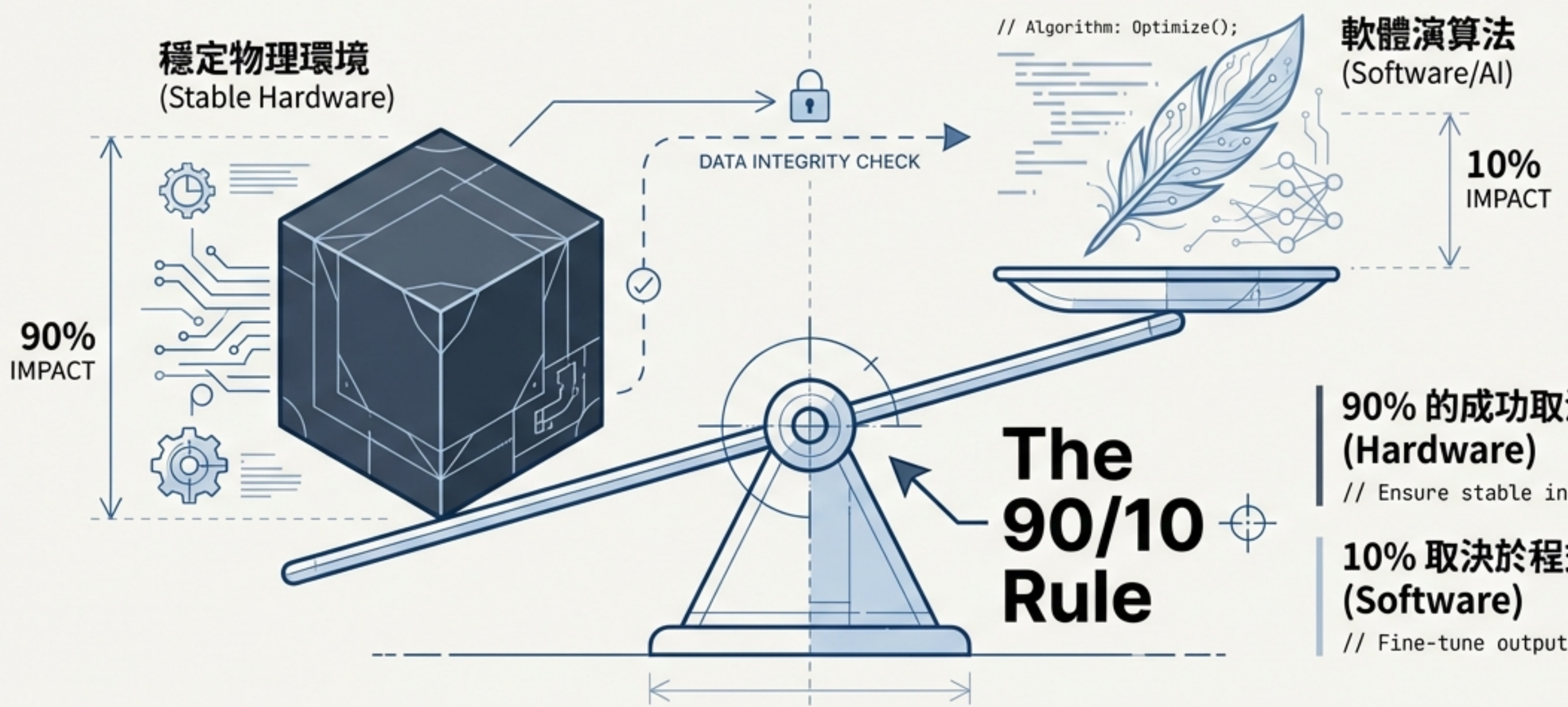
從概念到代碼：利用低成本電腦視覺實現醫療級精度



Python + OpenCV + Engineering Rigorousness

REV 1.0 // ENGINEERING GUIDE

核心哲學：環境一致性 > 程式碼複雜度



90% 的成功取決於物理環境 (Hardware)

```
// Ensure stable inputs
```

10% 取決於程式邏輯 (Software)

```
// Fine-tune outputs
```

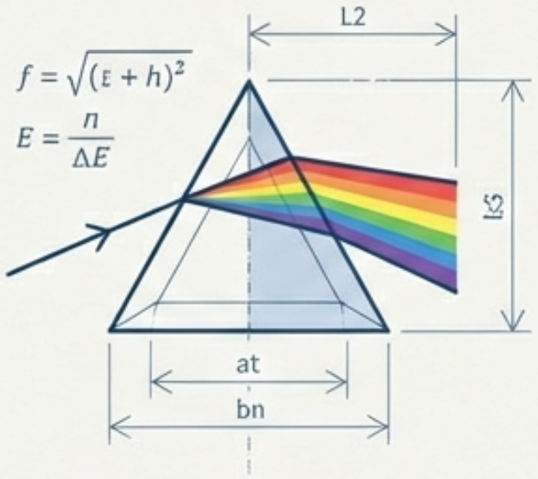
“軟體無法修復物理上的糟糕數據。我們建立『黑盒』以確保光學數據的絕對完整性。”

系統架構：工程實踐的三大支柱

Step 1: 硬體架構 (The Rig)

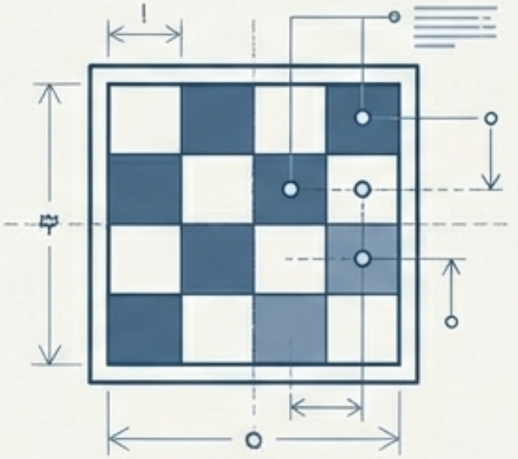
標準光源 + 治具定位

Step 2: 色彩分析 (Color Analysis)



白平衡校正 + Delta E 計算

Step 3: 透明度檢測 (Transparency)



對比度損失分析

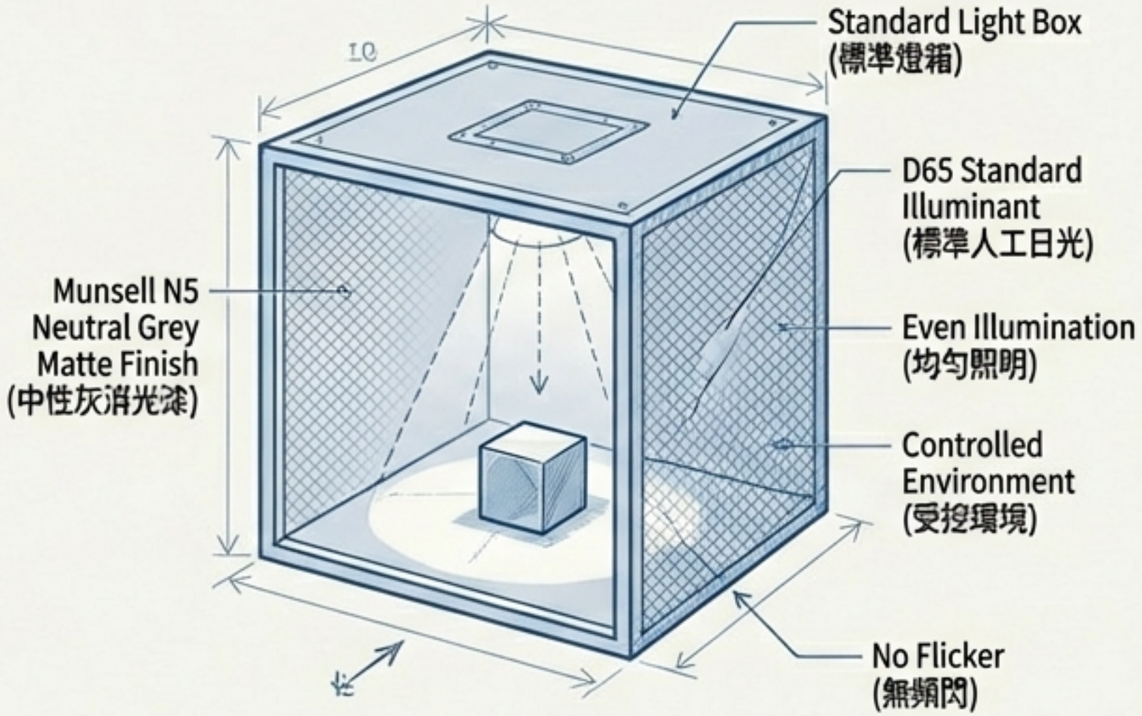
本方案不依賴昂貴的設備，而是依賴嚴格的標準化流程 (Standardization) 與基礎物理學。

步驟一：物理環境——光源是真理的來源

D65 標準光源與環境控制

❌ 錯誤示範：辦公桌環境 (不可控反射)

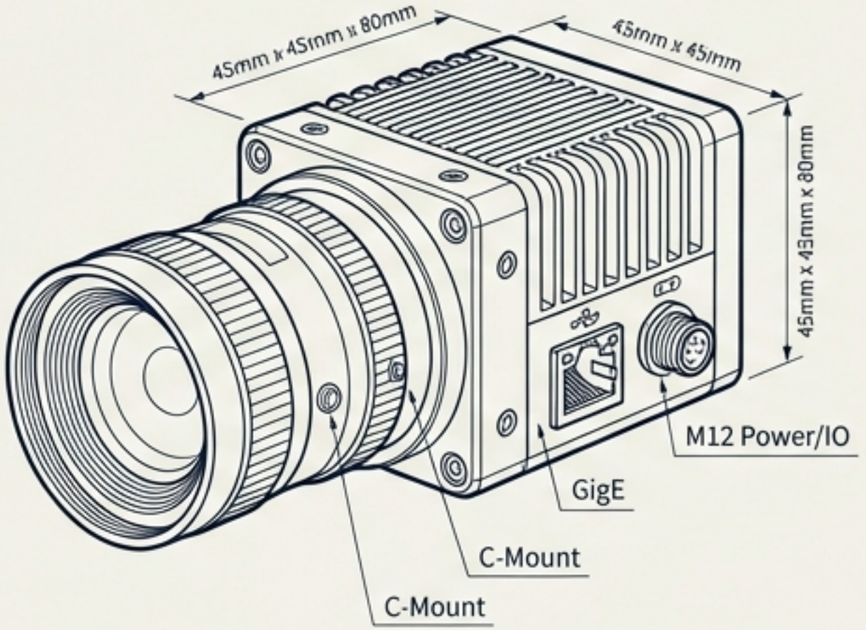
✅ 正確示範：標準燈箱 (D65 / Munsell N5)



- 光源：D65 標準人工日光 (6500K)
  D65 Illuminant
- 背景：Munsell N5/N7 中性灰消光漆
  Neutral Grey Matte
- 頻閃：高頻無頻閃 (No Flicker)
  High Frequency > 20kHz

JetBrains Mono

步驟一：視覺傳感器——拒絕「智慧」優化



Do Not Use Phones.
ISP 自動修圖會破壞原始數據。

Hardware Recommendation: **工業相機 (Hikrobot / Basler) 或 Logitech Brio**

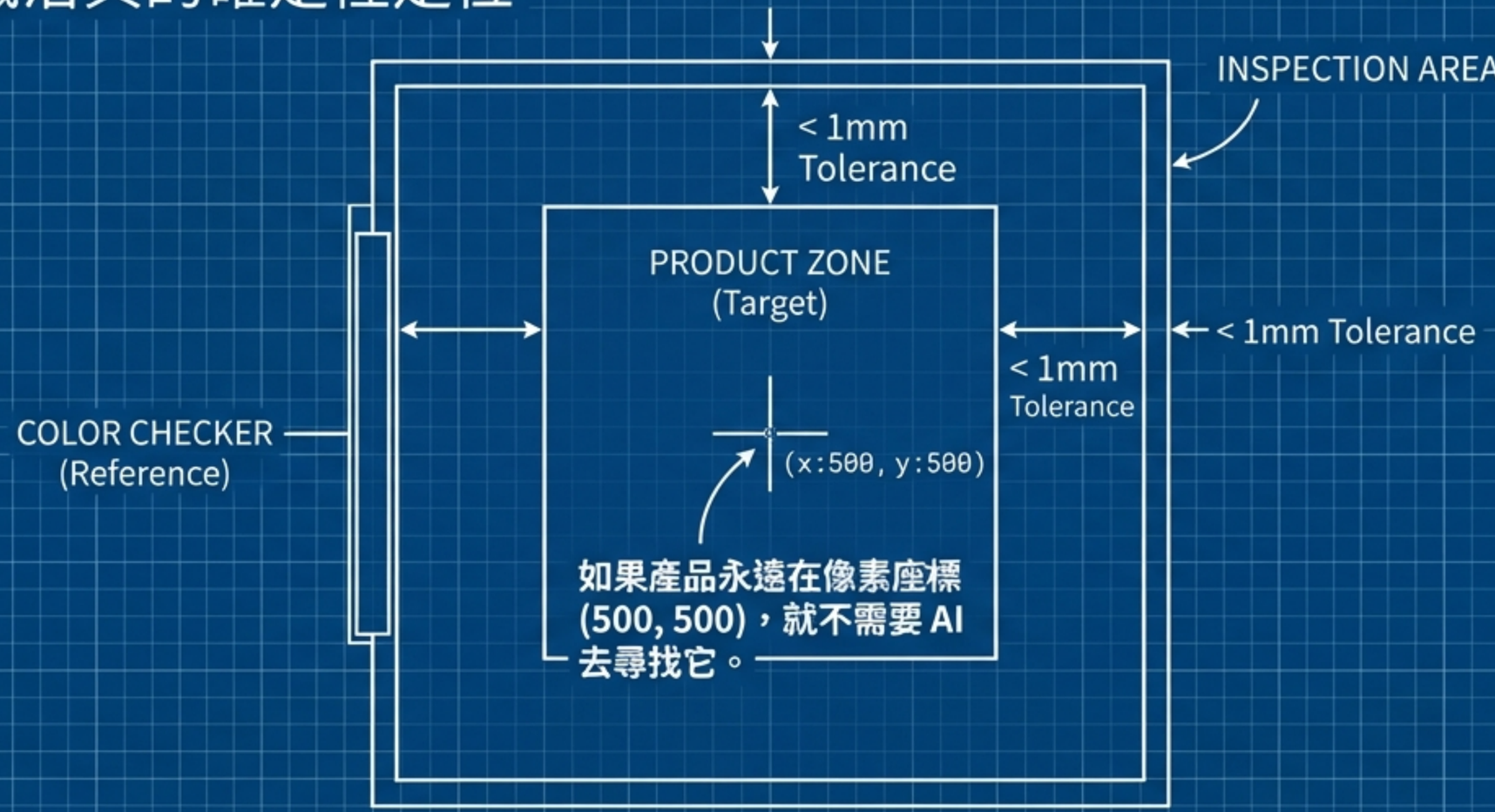
鎖定白平衡 (Fixed WB)：
6500K (嚴禁 Auto)

鎖定曝光 (Fixed Exposure)：
固定快門與 Gain

無損格式 (Lossless)：
RAW / PNG (嚴禁 JPG)

步驟一：治具與幾何——用物理取代演算法

使用機械治具的確定性定位



工程師思維：用物理條件消除軟體複雜性。

步驟二：軟體邏輯——線性工作流

Tech Stack: Python + OpenCV + Numpy

程式碼不需要複雜，只需要穩健 (Robust)。

核心邏輯 I：即時白平衡校正

Real-time White Balance Correction" (optional, for context)

Reference Grey Patch

$$\text{Gain} = \frac{\text{Target\_Value (128)}}{\text{Current\_Avg\_Value}}$$

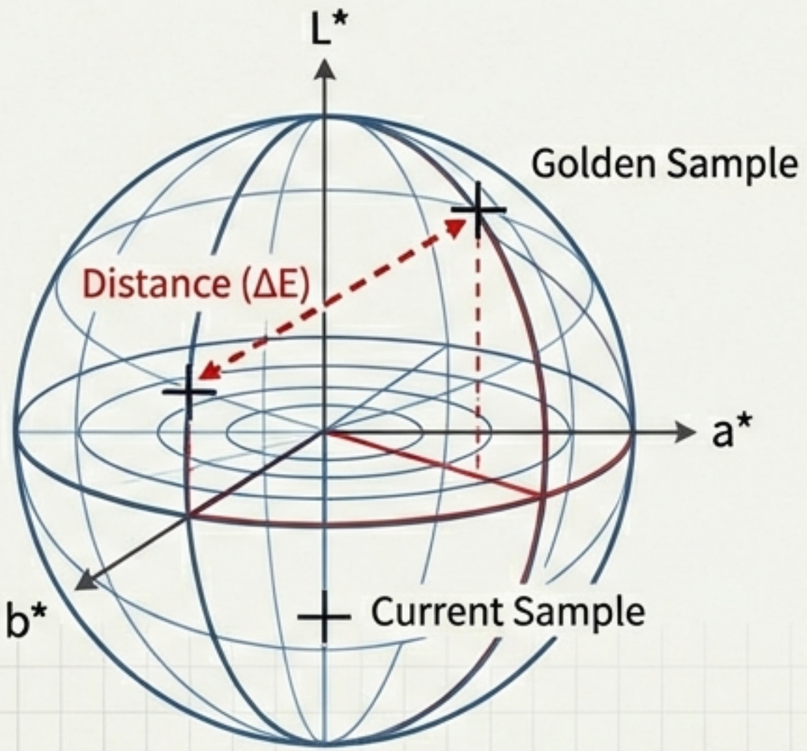
calibrate_image.py

```python
def calibrate_image(img, ref_rect):
    # Calculate Gain from Reference Card
    gain_b = target_gray / avg_b    # Blue Channel
    gain_g = target_gray / avg_g    # Green Channel
    gain_r = target_gray / avg_r    # Red Channel

    # Apply to entire image
    img_corrected = img.copy()
    img_corrected[:,:,0] *= gain_b  # Apply Blue
    img_corrected[:,:,1] *= gain_g  # Apply Green
    img_corrected[:,:,2] *= gain_r  # Apply Red
    return img_corrected
```

每一張照片都根據畫面中的「標準色卡」進行歸一化，消除燈管老化影響。

核心邏輯 II：色差計算 (Delta E)



color_difference_calculation.py

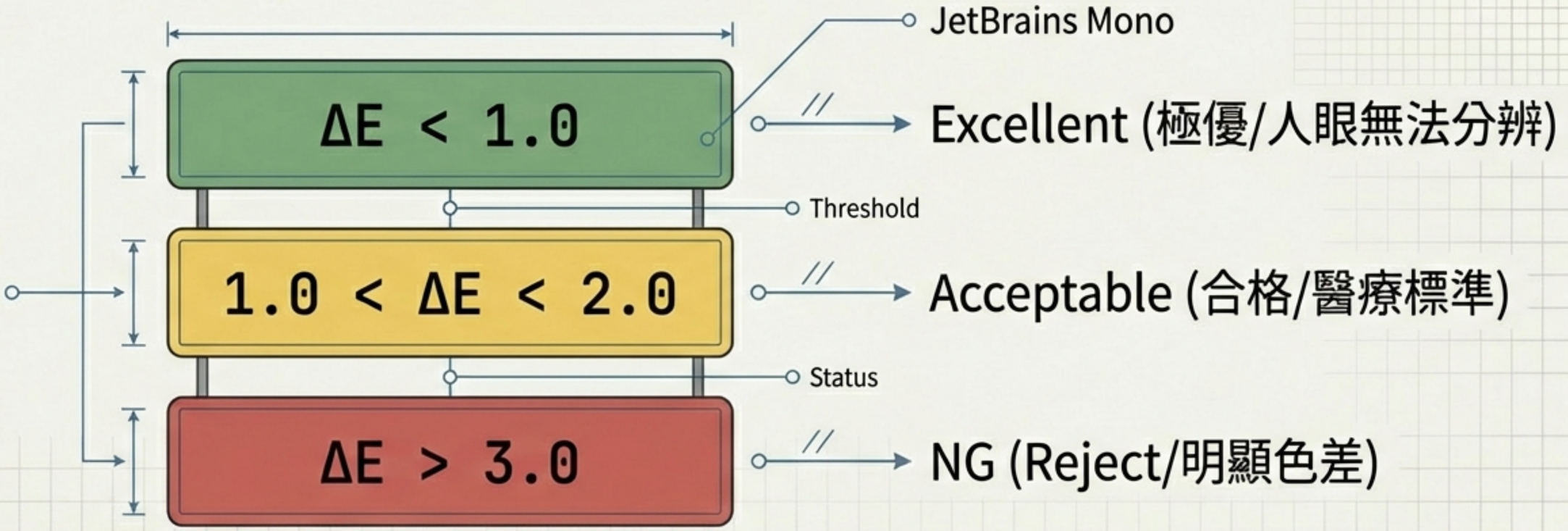
```python
# Convert to LAB space
lab_sample = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

# Euclidean Distance
delta_E = np.sqrt(delta_L**2 + delta_a**2 + delta_b**2)
```

$$\Delta E = \sqrt{((\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2}$$

L*a*b* 接近人類視覺感知 (L=亮度，a=紅綠，b=黃藍)

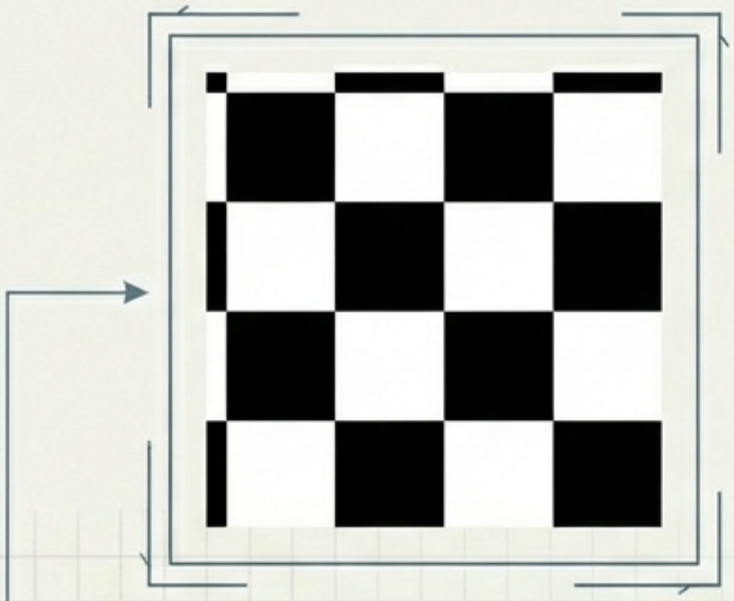
判定標準：數據的意義



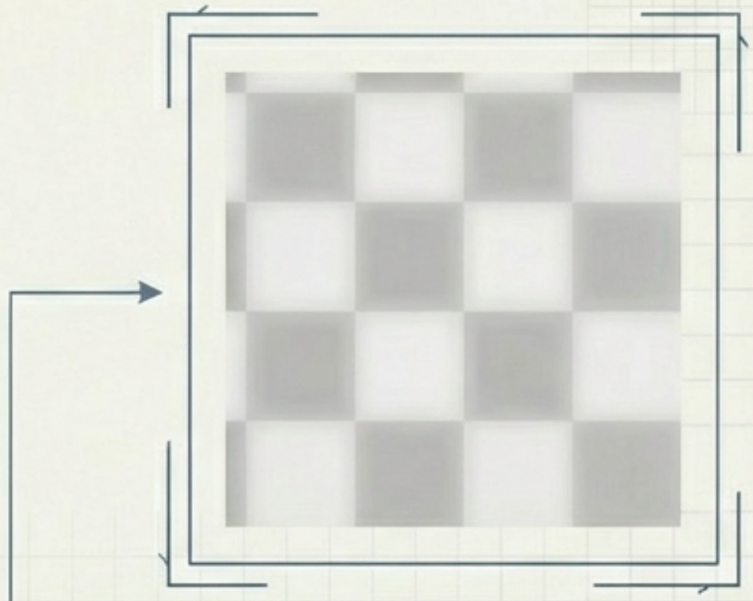
閾值應根據實際「黃金樣品」與客戶需求進行微調。

步驟三：透明度檢測挑戰

黑白棋盤格法 (The Checkerboard Method)



Clear Product (高對比度)

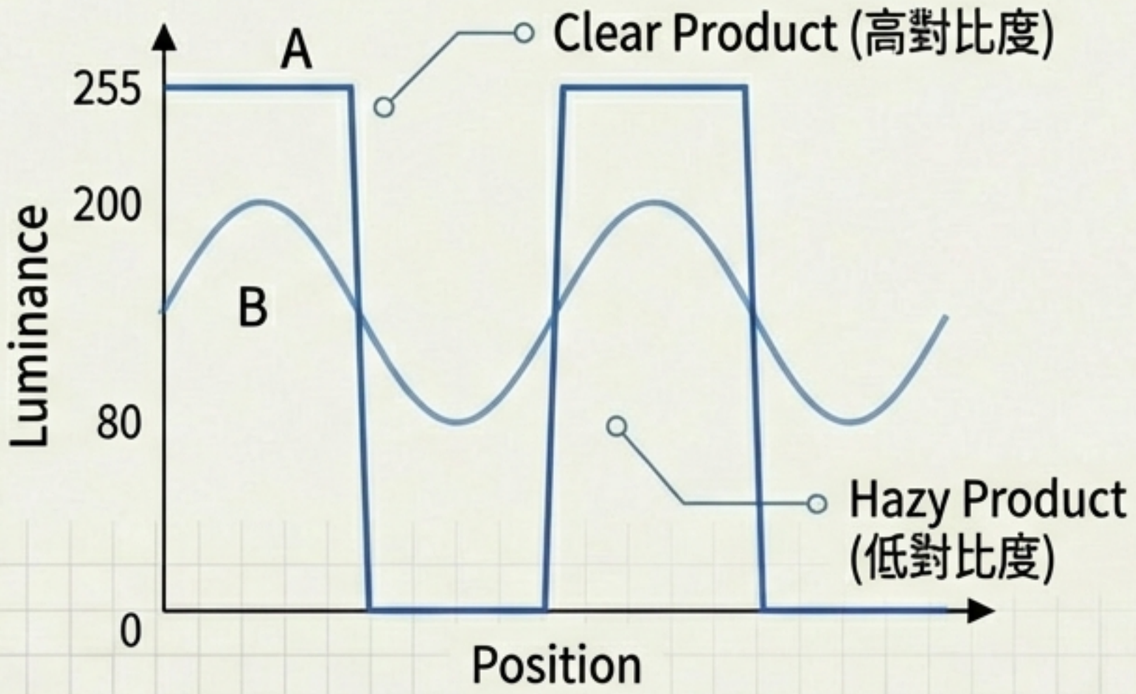


Hazy Product (低對比度/混濁)

問題：單靠反射光難以區分微混濁。
解法： 測量透過產品看到的「對比度損失 (Loss of Contrast)」

核心邏輯 III：對比度計算 (Michelson Contrast)

Luminance Profile



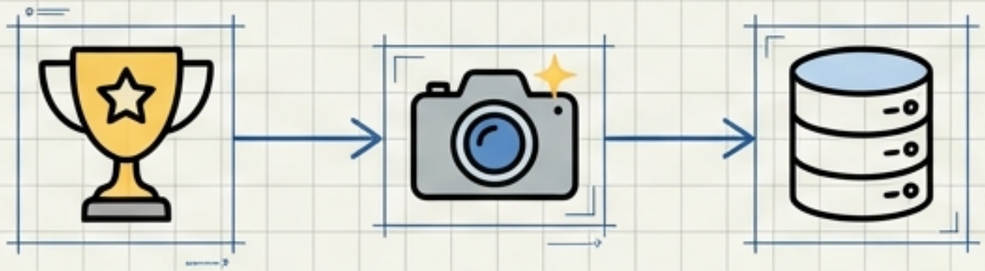
Python Code (JetBrains Mono)

```python
def check_transparency(img, roi_black, roi_white):
    L_min = np.mean(roi_black)
    L_max = np.mean(roi_white)
    contrast = (L_max - L_min) / (L_max + L_min)
    return contrast
```

$$\text{Contrast} = \frac{(L_{max} - L_{min})}{(L_{max} + L_{min})}$$

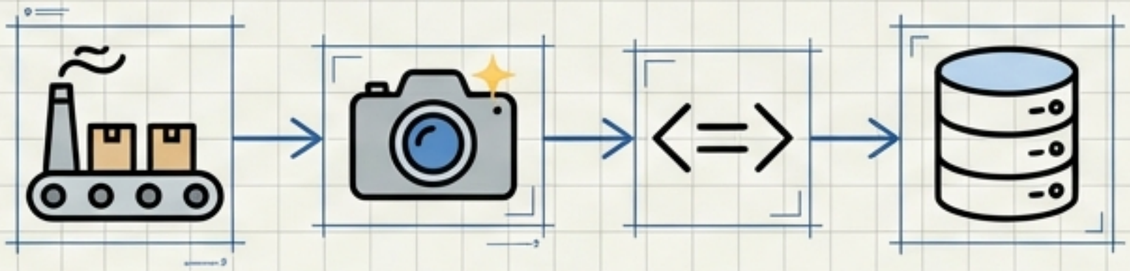
操作流程：黃金樣品策略

Learn Mode (錄入模式)



1. 放入黃金樣品 -> 2. 記錄 RGB/Lab/Contrast -> 3. 存入 DB

Production Mode (生產模式)



1. 測量當前產品 -> 2. 與 DB 黃金數值比對 -> 3. 判定 Pass/Fail

Key Rule: 不要將數值寫死 (Hardcode)。讓系統「學習」標準。

常見陷阱與工程解法

反光干擾
(Specular Highlights)

↓

ROI 避開反光區，或使用 Threshold 過濾高亮像素 (>240)。

震動
(Vibration)

↓

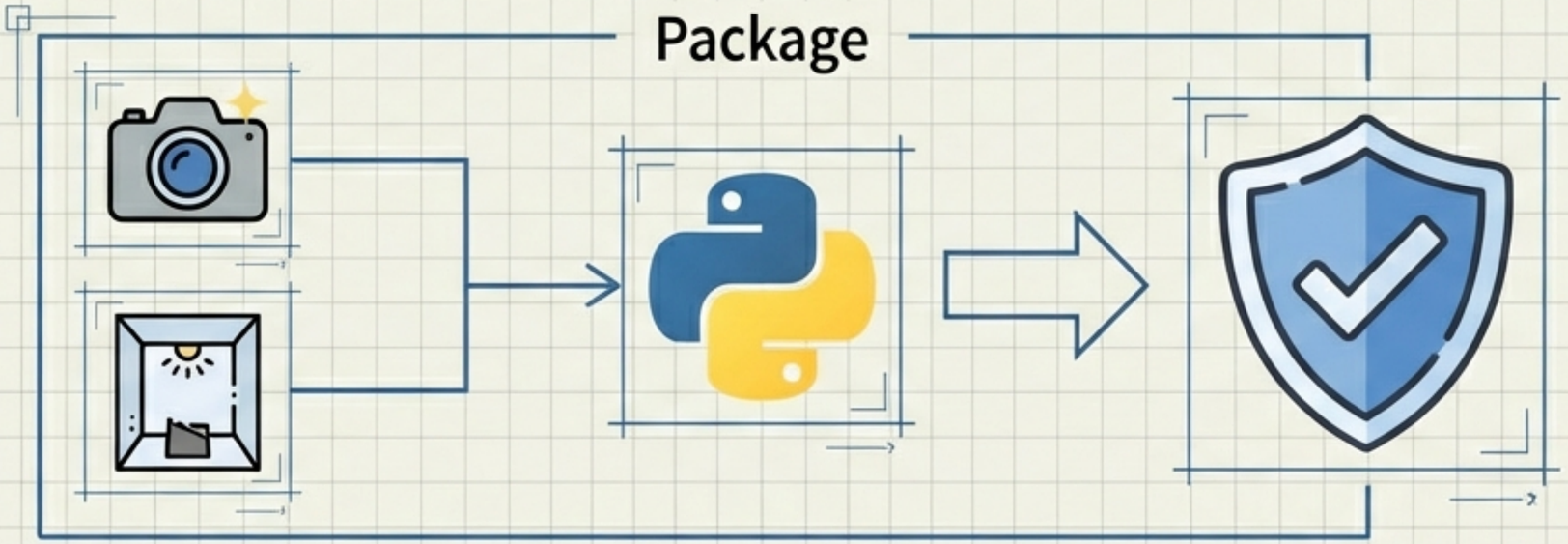
嚴格的物理治具固定，防止 ROI 偏移。

JPG 壓縮
(Compression)

↓

堅持使用 PNG/RAW 格式，避免色彩雜訊。

總結與效益：低成本實現醫療級品控



Cost:

< 20k TWD

Consistency:

100%

Human Error:

0%

- 客觀性：消除人眼疲勞與主觀誤差
- 可追溯性：每一件產品都有數據記錄

自動化的前提是標準化。先搭建好您的「黑盒」，剩下的就是簡單的 Python 數學。