# Image Grayscale World White Balance

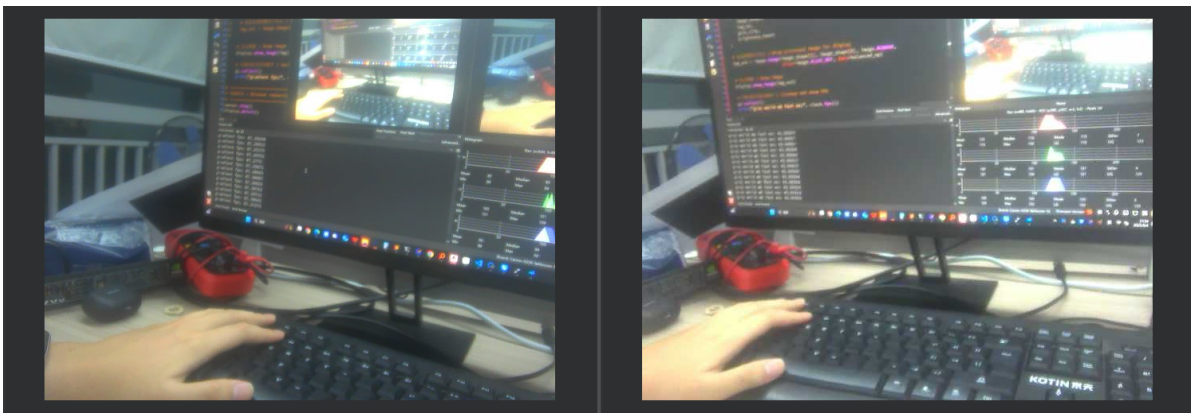


## Example Results

Run this section's example code [Source Code/06.cv_lite/28.rgb888_white_balance_gray_world_fast_ex.py]

In this section, we'll use the `cv_lite` extension module to implement grayscale world white balance processing for RGB888 format images on an embedded device, optimizing image color correction and brightness enhancement through adjustable parameters.

> Original image on the left, processed image on the right



## Principle Explanation

### What is Image Grayscale World White Balance?

Image grayscale world white balance is a simple color correction technique based on the "grayscale world assumption"—the assumption that the average color of an image in a natural scene tends to be neutral gray. It adjusts the image's RGB channels to equalize the average value of each channel, thereby eliminating color shifts caused by the light source. This algorithm is particularly useful on embedded devices because it is computationally lightweight and efficient.

- **How it works**: Grayscale world white balancing first calculates the average value for each color channel (R, G, B) of the image. It then calculates the gain for each channel to bring its average closer to the global average (typically set to neutral gray). Optionally, brightness enhancement can be applied to improve overall brightness. A simplified formula is: For each channel, gain = (global average / channel average), then multiplied by the pixel value. Finally, gain limiting is applied to prevent oversaturation.
- **Effect**: This operation effectively corrects image color temperature issues, making colors more natural and realistic. Gain and brightness can be controlled by adjusting parameters to avoid color distortion or overexposure. The processed image is then more suitable for display or further analysis.
- **Application Scenarios**: Grayscale world white balancing is widely used in camera image processing, real-time video correction, mobile photography, and robotic vision. In embedded systems, it is the preferred method for fast white balancing because it does not require complex calibration hardware and relies solely on algorithms.

In practical applications, adjustable parameters such as gain limiting and brightness enhancement can be optimized based on ambient light and image characteristics to balance color accuracy and visual quality.

# Code Overview

## Importing Modules

```
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import _thread
import cv_lite # cv_lite C extension module
import ulab.numpy as np # NumPy-like ndarray for MicroPython
```

### Setting the Image Size

```
image_shape = [480, 640] # Height x Width
```

Define the image resolution to 480x640 (height x width). The camera and display modules will be initialized based on this size.

### Initialize the camera (RGB888 format)

```
sensor = Sensor(id=2, width=1280, height=960,fps=90)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.RGB888) # Set pixel format to RGB888
```

- Initialize the camera, set the resolution to 1280x960 and the frame rate to 90 FPS.
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for color correction).

### Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True, quality=50)
```

Initialize the display module, using the ST7701 driver chip, with the resolution consistent with the image size. `to_ide=True` indicates that the image will be simultaneously transmitted to the IDE for virtual display, and `quality=50` sets the image transmission quality.

### Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run()
```

Initialize the media resource manager and start the camera to capture the image stream.

### Set the frame rate timer

```
clock = time.clock() # Start the frame rate timer / Start FPS timer
```

Initialize the frame rate timer, which is used to calculate and display the number of frames processed per second (FPS).

### Setting white balance parameters

```
gain_clip = 2.5 # Gain limit to prevent color blowout
brightness_boost = 1.25 # Global brightness boost
```

- `gain_clip`: Gain limit, used to prevent color distortion caused by over-amplification of color channels.
- `brightness_boost`: Brightness boost, used to increase overall image brightness.

### Image processing and white balance operation

```
while True:
    clock.tick() # Record the start time of the current frame / Start timing

    # Capture a frame / Capture a frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref() # Get RGB888 image reference / Get RGB888
ndarray reference (HWC)

    # Use cv_lite for accelerated grayscale world white balance processing
(adjustable parameters)
    # Apply fast gray world white balance with tunable parameters
    balanced_np = cv_lite.rgb888_white_balance_gray_world_fast_ex(
        image_shape,
        img_np,
        gain_clip,
        brightness_boost
    )


    # Wrap image for display / Wrap processed image for display
```

```
    img_out = image.Image(image_shape[1], image_shape[0], image.RGB888,
                          alloc=image.ALLOC_REF, data=balanced_np)

    # Display image
    Display.show_image(img_out)

    # Cleanup and show FPS
    gc.collect()
    print("gray world wb fast ex:", clock.fps())
```

- **Image capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **White balance processing**: Call `cv_lite.rgb888_white_balance_gray_world_fast_ex()` to perform grayscale world white balancing, returning the processed image array.
- **Image packaging and display**: Package the processed result into an RGB888 image object and display it on the screen or in an IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clear memory, and print the current frame rate through `clock.fps()`.

### Resource release

```
sensor.stop()
Display.deinit()
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
time.sleep_ms(100)
MediaManager.deinit()
```

## Parameter adjustment instructions

- `gain_clip`: Gain limit coefficient. Larger values (such as 1.5-3.0) allow for a wider range of color adjustment, but may cause saturation; smaller values (such as 1.0) maintain the original color. It is recommended to start testing with 2.0 to balance color correction and prevent color cast.
- `brightness_boost`: Brightness boost factor. Values greater than 1.0 (e.g., 1.0-1.5) increase overall brightness and improve shadow details. A value of 1.0 results in no enhancement. Excessively large values (e.g., 2.0) may cause overexposure. It is recommended to adjust this value based on ambient light conditions. Start with a value of 1.1 for testing.