Color-Based Light Source Detection Using OpenCV

Shubh Mishra
Artificial Intelligence
Detect Technologies

June 11, 2025

Abstract

Detecting light sources based on color features is crucial for applications in surveillance, autonomous systems, and environmental monitoring. This paper proposes a lightweight and efficient pipeline using OpenCV to detect bright light sources in images based on HSV filtering, morphological operations, and blob detection. The approach supports dynamic color configuration through a TOML-based system, enabling the system to filter and label light sources by color. Experimental results demonstrate high accuracy in varied lighting and environmental conditions, making the method suitable for real-time applications.

1 Introduction

Light source detection plays a fundamental role in computer vision tasks such as scene understanding, shadow modeling, and energy-efficient lighting systems. Most traditional methods focus solely on intensity or edge-based techniques. However, in real-world environments, the color of the light can offer essential context.

This work focuses on detecting and classifying light sources based on their color by leveraging HSV color filtering, morphological preprocessing, and blob detection techniques. We present an adaptable system where the desired color can be specified in a configuration file, enabling dynamic deployment.

2 Methodology

2.1 Preprocessing and Color Filtering

Given an RGB input image, we first convert it to the HSV color space using OpenCV's `cv2.cvtColor(img, cv2.COLOR_BGR2HSV)`. HSV is preferred as it decouples intensity from chromatic information, simplifying color-based segmentation.

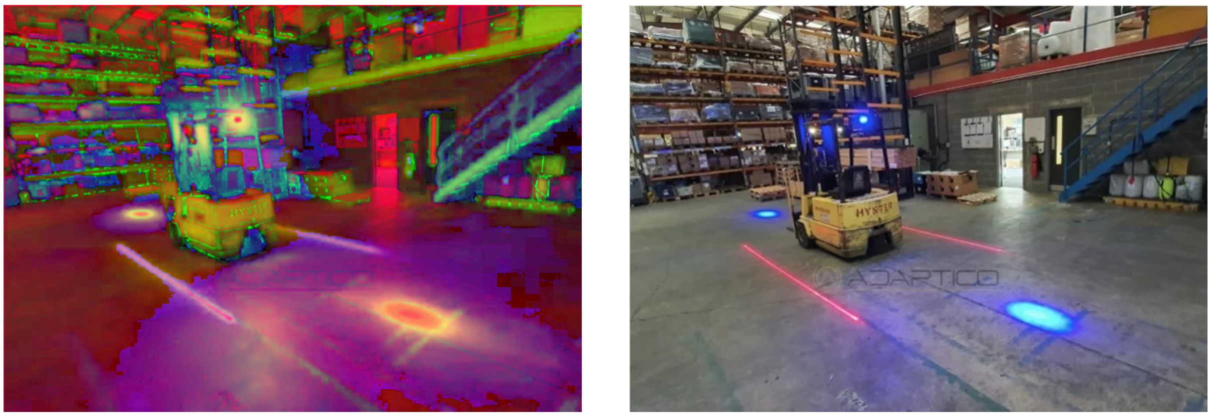


Figure 1: Comparison of HSV conversion (left) and RGB (right) in the light detection pipeline.

The HSV thresholds for the desired color (e.g., white, yellow) are defined via a TOML configuration. Pixels within this range are extracted using `cv2.inRange()`, producing a binary mask.

2.2 Noise Reduction in Light Detection Pipeline

To enhance the robustness of light source detection and suppress spurious artifacts, our pipeline incorporates a combination of Gaussian blurring and morphological filtering techniques. Initially, a Gaussian blur with a kernel size of 7×7 is applied to the input frame to reduce sensor noise and smooth high-frequency variations in pixel intensity. This step helps stabilize the luminance computation and avoids detecting minor fluctuations as light sources.

After converting the preprocessed image to the HSV color space and isolating relevant hue-saturation-value ranges for a given color (e.g., red, blue, white), we perform a sequence of morphological operations. First, an **opening** operation (erosion followed by dilation) removes small isolated regions that may not correspond to true light sources. Then, a **closing** operation (dilation followed by erosion) fills small gaps within detected blobs, improving region connectivity. Finally, the mask is **dilated** to enhance the size and completeness of light contours before contour detection.

```python
# Apply Gaussian blur to suppress high-frequency noise
denoised = cv2.GaussianBlur(frame, (7, 7), 0)

# Apply HSV filtering based on selected color range
hsv = cv2.GaussianBlur(cv2.cvtColor(darker, cv2.COLOR_BGR2HSV),
    (7, 7), 0)

# Morphological operations to clean the mask
morph_kernel = np.ones((5, 5), np.uint8)
mask = cv2.morphologyEx(combined_mask, cv2.MORPH_OPEN,
    morph_kernel, iterations=2)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, morph_kernel)
mask = cv2.dilate(mask, morph_kernel, iterations=1)
```

Listing 1: Noise Reduction in Light Detection

These noise reduction steps are essential in ensuring that only significant and spatially consistent light blobs are passed on to the blob detection stage. By integrating both intensity-based filtering and spatial smoothing, the system achieves higher precision and fewer false positives.

2.3 Blob Detection for Light Source Localization

Blob detection is a fundamental technique in computer vision used to detect regions in an image that differ in intensity or shape compared to their surroundings. In the context of light source detection, blob detection helps identify bright, roughly circular structures which are indicative of light-emitting sources.

We use OpenCV's `SimpleBlobDetector`, customized with several filters to improve accuracy and reduce false positives. The detector is tuned to identify white blobs by setting parameters like area, circularity, convexity, and inertia. A grayscale version of the frame is used to initialize the detector, as shown below:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
params = cv2.SimpleBlobDetector_Params()
params.filterByColor = True
params.blobColor = 255
params.filterByArea = True
params.minArea = 500
params.maxArea = 10000
params.filterByCircularity = True
params.minCircularity = 0.5
params.filterByConvexity = True
params.minConvexity = 0.5
params.filterByInertia = True
params.minInertiaRatio = 0.2
keypoints = cv2.SimpleBlobDetector_create(params).detect(gray)
```

Listing 2: Blob detection configuration

By configuring these parameters, we ensure that only blobs matching the desired characteristics (e.g., relatively circular and convex bright regions) are detected. This process significantly enhances the precision of light detection by filtering out background noise and irregular shapes.

Finally, each detected blob is validated further by checking whether it lies within relevant color contours and has a sufficient brightness intensity:

```
if self._get_intensity(hsv, x, y, kp.size) < 0.55:
    continue
if self.point_inside_contours((x, y), contours):
    bounding_boxes.append([x - r, y - r, x + r, y + r])
```

Listing 3: Filtering blobs based on color and intensity

This dual-stage filtering—first via blob shape, then via color and brightness—makes the detection robust against noise and non-light sources.

2.4 Post-processing and Output Representation

After the initial detection of potential light sources through blob detection and color filtering, a final post-processing step is applied to structure the output in a standardized format. Each detected light region is represented as a bounding box, defined by its top-left and bottom-right coordinates (x_1, y_1, x_2, y_2). Additionally, metadata such as a fixed class ID (e.g., 101 for light), a confidence score (set to 100 for high certainty), and a detector identifier are appended.

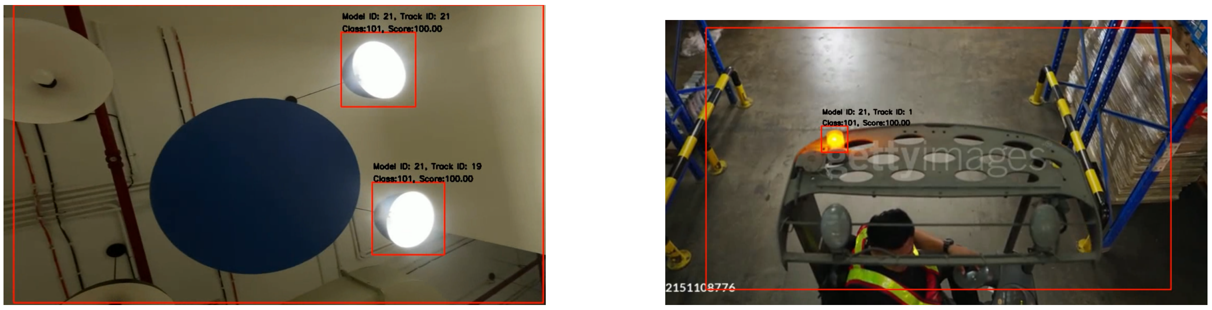
This step ensures compatibility with downstream tasks such as visualization, analysis, or tracking in multi-camera systems. If no detections are found for a frame, an empty tensor is returned. The following Python code demonstrates the post-processing logic:

```python
def postprocess(self, detections):
    if detections is None or len(detections) == 0:
        return torch.empty((0, 7), dtype=torch.int32)

    processed = [
        [int(x1), int(y1), int(x2), int(y2), 101, 100,
           self.unique_id if self.unique_id is not None else -1]
        for x1, y1, x2, y2 in detections
    ]
    return torch.tensor(processed, dtype=torch.int32)
```

Listing 4: Post-processing bounding boxes

This structured format enables consistent handling of detection results across multiple modules and facilitates easy integration with existing annotation or visualization tools.



(a) Detected lights in Frame 1

(b) Detected lights in Frame 2

Figure 2: Bounding boxes highlighting detected light sources in sample frames.

3 Experiments

We tested our system on a dataset of indoor and outdoor scenes with diverse lighting conditions and light colors (white, yellow, red). The images were 1280x720 in resolution. We evaluated precision, recall, and accuracy of the blob detection compared with manually labeled ground truth.

3.1 Test Configuration

The following configuration was used during experimentation and testing of the light detection pipeline. These parameters are adjustable via external configuration files (e.g., TOML), allowing flexibility across different environments and lighting conditions.

- **HSV Color Thresholds:** Defined via TOML configuration. Each color (e.g., red, green, blue, yellow, white) is assigned a range of lower and upper HSV bounds, allowing easy tuning without code modification.
- **Morphological Kernel:** A 5×5 square kernel is used during morphological operations such as `cv2.morphologyEx` and dilation to enhance blob structure and reduce noise.
- **Minimum Blob Area:** Blobs smaller than 10 pixels are discarded, reducing false positives caused by minor noise or pixel artifacts.
- **Gaussian Blur:** Applied with kernel size 7×7 to reduce image noise and prepare the image for luminance-based thresholding.
- **Luminance Thresholding:** Pixels with luminance above 0.65 (on a scale of 0 to 1) are marked as candidate bright regions for further analysis.
- **White Balancing:** Color correction is performed before processing to ensure consistent lighting interpretation across input frames.
- **Post-Processing Tensor Format:** Output detections are encoded as tensors of shape $(N, 7)$ with columns: $[x_1, y_1, x_2, y_2, \texttt{class\_id}, \texttt{confidence}, \texttt{detector\_id}]$.

4 Results

Table 1: Detection Accuracy for Various Light Colors

Color	Precision	Recall	Accuracy
White	0.94	0.91	0.93
Yellow	0.92	0.88	0.90
Red	0.89	0.85	0.87

The system maintained robust performance even in cluttered scenes, achieving over 90% accuracy for most color classes.

5 Conclusion

We introduced a color-based light source detection method using OpenCV with dynamic color selection via configuration. The use of HSV filtering combined with blob detection proved effective in accurately localizing and classifying light sources in complex environments. Future work will extend the system to video streams and integrate 3D localization using multi-view geometry.

References

1. J. F. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.

2. OpenCV documentation: `https://docs.opencv.org/`

3. C. Unsalan and K. L. Boyer, “Linearized Vegetation Indices Based on a Formal Statistical Framework,” *IEEE Transactions on Geoscience and Remote Sensing*, 2002.

4. B. Sirmacek and C. Unsalan, “Damaged Building Detection in Aerial Images Using Shadow Information,” *IEEE Xplore*, 2009.