

# Blob Detection

Blob

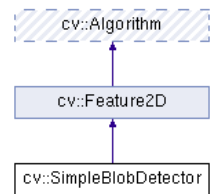
→ A Blob is a group of connected pixels in an image that share some common property ( E.g grayscale value ).

⇒ **Class Name:** cv::SimpleBlobDetector

→ struct Params

→ Public attributes of this structure.

uchar	blobColor	•	float	maxThreshold	•
bool	filterByArea	•	float	minArea	•
bool	filterByCircularity	•	float	minCircularity	•
bool	filterByColor	•	float	minConvexity	•
bool	filterByConvexity	•	float	minDistBetweenBlobs	•
bool	filterByInertia	•	float	minInertiaRatio	•
float	maxArea	•	size_t	minRepeatability	•
float	maxCircularity	•	float	minThreshold	•
float	maxConvexity	•	float	thresholdStep	•
float	maxInertiaRatio	•			



⇒ The class implements a simple algorithm for extracting blobs from an image:

- ① Convert the source image to binary images by applying thresholding with several thresholds from minThreshold (inclusive) to maxThreshold (exclusive) with distance thresholdStep between neighboring thresholds.
- ② Extract connected components from every binary image by **findContours** and calculate their centers.
- ③ Group centers from several binary images by their coordinates. Close centers form one group that corresponds to one blob, which is controlled by the minDistBetweenBlobs parameter.
- ④ From the groups, estimate final centers of blobs and their radiuses and return as locations and sizes of keypoints.

⇒ This class performs several filtrations of returned blobs. You should set filterBy\* to true/false to turn on/off corresponding filtration.

⇒ Available filtrations:

## 1. By color

⇒ This filter compares the intensity of a binary image at the center of a blob to blobColor.

⇒ If they differ, the blob is filtered out.

⇒ Use blobColor = 0 to extract dark blobs and blobColor = 255 to extract light blobs.

## 2. By area

⇒ Extracted blobs have an area between minArea (inclusive) and maxArea (exclusive).

## 3. By circularity

⇒ Extracted blobs have circularity ( $4 * \pi * \text{Area} / \text{perimeter} * \text{perimeter}$ ) between minCircularity (inclusive) and maxCircularity (exclusive).

#### 4. By ratio of the minimum inertia to maximum inertia

⇒ Extracted blobs have this ratio between minInertiaRatio (inclusive) and maxInertiaRatio (exclusive).

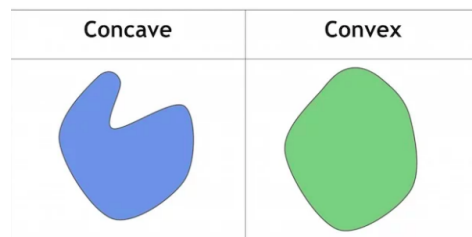
⇒ This measures how elongated a shape

⇒ For a circle, this value is 1, for an ellipse it is between 0 and 1, and for a line it is 0.

#### 5. By convexity

⇒ Extracted blobs have convexity (area / area of blob convex hull) between minConvexity (inclusive) and maxConvexity (exclusive).

↳ Convex Hull of a shape is the tightest convex shape that completely encloses the shape.



#### ★ Static member function

```
static Ptr< SimpleBlobDetector > create (const SimpleBlobDetector::Params &parameters=SimpleBlobDetector::Params())
```

#### ★ cv::KeyPoint

Data structure for salient point detectors.

The class instance stores a keypoint, i.e. a point feature found by one of many available keypoint detectors, such as Harris corner detector, FAST, StarDetector, SURF, SIFT etc.

The keypoint is characterized by the 2D position, scale, orientation and some other parameters.

The keypoint neighborhood is then analyzed by another algorithm that builds a descriptor (usually represented as a feature vector).

##### Public Attributes

- float angle { keypoint orientation }
- int class\_id { object class (if the keypoints need to be clustered by an object they belong to) }
- int octave { octave (pyramid layer) from which the keypoint has been extracted }
- Point2f pt { x & y coordinates of the keypoint }
- float response { keypoint detector response on the keypoint (that is, strength of the keypoint) }
- float size { diameter of the meaningful keypoint neighborhood }

## ★ cv::Feature2D


⇒ Abstract base class for 2D image feature detectors and descriptor extractors.

⇒ Important public member function:

```
virtual void detect (InputArray image, std::vector< KeyPoint > &keypoints, InputArray mask=noArray())  
    Detects keypoints in an image (first variant) or image set (second variant). More...
```

## ★ cv::drawKeypoints()

```
void cv::drawKeypoints (InputArray image, const std::vector< KeyPoint > &keypoints, InputOutputArray outImage, const Scalar &color=Scalar::all(-1),  
    int flags=DrawMatchesFlags::DEFAULT)
```



DEFAULT	Output image matrix will be created ( <a href="#">Mat::create</a> ), i.e. existing memory of output image may be reused. Two source image, matches and single keypoints will be drawn. For each keypoint only the center point will be drawn (without the circle around keypoint with keypoint size and orientation).
DRAW_OVER_OUTIMG	Output image matrix will not be created ( <a href="#">Mat::create</a> ). Matches will be drawn on existing content of output image.
NOT_DRAW_SINGLE_POINTS	Single keypoints will not be drawn.
DRAW_RICH_KEYPOINTS	For each keypoint the circle around keypoint with keypoint size and orientation will be drawn.



All these flages have namespace of DrawMatchesFlags.