Package: opencv (via r-universe)

October 17, 2026

Type Package

Title Bindings to 'OpenCV' Computer Vision Library

Version 0.6.1

Description Exposes some of the available 'OpenCV' <https://opencv.org/> algorithms, such as a QR code scanner, and edge, body or face detection. These can either be applied to analyze static images, or to filter live video footage from a camera device.

License MIT + file LICENSE

SystemRequirements OpenCV 4 or newer: libopencv-dev (Debian, Ubuntu) or opencv-devel (Fedora). The QR code detector requires at least libopencv 4.5.2.

URL https://docs.ropensci.org/opencv/ https://ropensci.r-universe.dev/opencv

BugReports https://github.com/ropensci/opencv/issues

LinkingTo Rcpp

Imports Rcpp, magrittr

Suggests grDevices, qrcode, testthat (>= 3.0.0)

Encoding UTF-8

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/pak/sysreqs libopencv-dev

Repository https://ropensci.r-universe.dev

Date/Publication 2026-09-17 08:35:49 UTC

RemoteUrl https://github.com/ropensci/opencv

RemoteRef master

RemoteSha 223ac3b7b237b5c9167fd24b4f0ff1d47eb29f5d

Contents

ocv_face 2
ocv_keypoints 4
ocv_qr_detect 5
opencv-area 6

Index **8**

`ocv_face` *OpenCV Computer Vision*

Description

Tools to experiment with computer vision algorithms. Use ocv_read and ocv_write to load/save images on disk, or use ocv_picture / ocv_video to use your webcam. In RSudio IDE the image objects will automatically be displayed in the viewer pane.

Usage

```
ocv_face(image)

ocv_facemask(image)

ocv_read(path)

ocv_write(image, path)

ocv_destroy(image)

ocv_bitmap(image)

ocv_edges(image)

ocv_picture()

ocv_resize(image, width = 0, height = 0)

ocv_mog2(image)

ocv_knn(image)

ocv_hog(image)

ocv_blur(image, ksize = 5)

ocv_sketch(image, color = TRUE)
```

```
ocv_stylize(image)

ocv_markers(image)

ocv_info(image)

ocv_copyto(image, target, mask)

ocv_display(image)

ocv_video(filter, stop_on_result = FALSE)

ocv_grayscale(image)

ocv_version()
```

Arguments

image	an ocv image object created from e.g. ocv_read()
path	image file such as png or jpeg
width	output width in pixels
height	output height in pixels
ksize	size of blurring matrix
color	true or false
target	the output image
mask	only copy pixels from the mask
filter	an R function that takes and returns an opecv image
stop_on_result	stop if an object is detected

Examples

```
# Silly example
mona <- ocv_read('https://jeroen.github.io/images/monalisa.jpg')

# Edge detection
ocv_edges(mona)
ocv_markers(mona)

# Find face
faces <- ocv_face(mona)

# To show locations of faces
facemask <- ocv_facemask(mona)
attr(facemask, 'faces')

# This is not strictly needed
ocv_destroy(mona)
```

`ocv_keypoints` *OpenCV keypoints*

Description

Find key points in images

Usage

```
ocv_keypoints(
  image,
  method = c("FAST", "Harris"),
  control = ocv_keypoints_options(method, ...),
  ...
)
```

Arguments

`image`	an ocv grayscale image object
`method`	the type of keypoint detection algorithm
`control`	a list of arguments passed on to the algorithm
`...`	further arguments passed on to ocv_keypoints_options

FAST algorithm arguments

- threshold threshold on difference between intensity of the central pixel and pixels of a circle around this pixel.
- nonmaxSuppression if true, non-maximum suppression is applied to detected corners (keypoints).
- type one of the three neighborhoods as defined in the paper: TYPE_9_16, TYPE_7_12, TYPE_5_8

Harris algorithm arguments

- numOctaves the number of octaves in the scale-space pyramid
- corn_thresh the threshold for the Harris cornerness measure
- DOG_thresh the threshold for the Difference-of-Gaussians scale selection
- maxCorners the maximum number of corners to consider
- num_layers the number of intermediate scales per octave

Examples

```
mona <- ocv_read('https://jeroen.github.io/images/monalisa.jpg')
mona <- ocv_resize(mona, width = 320, height = 477)

# FAST-9
pts <- ocv_keypoints(mona, method = "FAST", type = "TYPE_9_16", threshold = 40)
# Harris
pts <- ocv_keypoints(mona, method = "Harris", maxCorners = 50)

# Convex Hull of points
pts <- ocv_chull(pts)
```

---

ocv_qr_detect *Detect and Decode a QR code*

Description

Detect and decode a QR code from an image or camera. By default it returns the text value from the QR code if detected, or NULL if no QR was found. If `draw = TRUE` then it returns an annotated image with the position and value of the QR drawn into the image, and qr text value as an attribute. The `qr_scanner` function opens the camera device (if available on your computer) and repeats ocv_qr_detect until it a QR is detected.

Usage

```
ocv_qr_detect(image, draw = FALSE, decoder = c("wechat", "quirc"))

qr_scanner(draw = FALSE, decoder = c("wechat", "quirc"))
```

Arguments

`image`	an ocv image object created from e.g. `ocv_read()`
`draw`	if TRUE, the function returns an annotated image showing the position and value of the QR code.
`decoder`	which decoder implementation to use, see details.

Details

OpenCV has two separate QR decoders. The 'wechat' decoder was added in libopencv 4.5.2 and generally has better performance and fault-tolerance. The old 'quirc' decoder is available on some older versions of libopencv as a plug-in, but many Linux distros did not include it. If you get an error *Library QUIRC is not linked. No decoding is performed.* this sadly means your Linux distribution is too old and does not support QR decoding.

Value

if a QR code is detected, this returns either the text value of the QR, or if `draw` it returns the annotated image, with the value as an attribute. Returns NULL if no QR was found in the image.

Examples

```
png("test.png")
plot(qrcode::qr_code("This is a test"))
dev.off()
ocv_qr_detect(ocv_read('test.png'))
unlink("test.png")
```

`opencv-area` *OpenCV area manipulation*

Description

Manipulate image regions

Usage

```
ocv_rectangle(image, x = 0L, y = 0L, width, height)

ocv_polygon(image, pts, convex = FALSE, crop = FALSE, color = 255)

ocv_bbox(image, pts)

ocv_chull(pts)
```

Arguments

`image`	an ocv image object
`x`	horizontal location
`y`	vertical location
`width`	width of the area
`height`	height of the area
`pts`	a list of points with elements x and y
`convex`	are the points convex
`crop`	crop the resulting area to its bounding box
`color`	color for the non-polygon area

Examples

```
mona <- ocv_read('https://jeroen.github.io/images/monalisa.jpg')

# Rectangular area
ocv_rectangle(mona, x = 400, y = 300, height = 300, width = 350)
ocv_rectangle(mona, x = 0, y = 100, height = 200)
ocv_rectangle(mona, x = 500, y = 0, width = 75)

# Polygon area
img <- ocv_resize(mona, width = 320, height = 477)
pts <- list(x = c(184, 172, 146, 114,  90,  76,  92, 163, 258),
            y = c(72,   68,  70,  90, 110, 398, 412, 385, 210))
ocv_polygon(img, pts)
ocv_polygon(img, pts, crop = TRUE)
ocv_polygon(img, pts, convex = TRUE, crop = TRUE)

# Bounding box based on points
ocv_bbox(img, pts)

# Bounding box of non-zero pixel area
area <- ocv_polygon(img, pts, color = 0, crop = FALSE)
area
area <- ocv_bbox(area)
area
```

Index

ocv_bbox *(opencv-area)*, 6
ocv_bitmap *(ocv_face)*, 2
ocv_blur *(ocv_face)*, 2
ocv_chull *(opencv-area)*, 6
ocv_copyto *(ocv_face)*, 2
ocv_destroy *(ocv_face)*, 2
ocv_display *(ocv_face)*, 2
ocv_edges *(ocv_face)*, 2
ocv_face, 2
ocv_facemask *(ocv_face)*, 2
ocv_grayscale *(ocv_face)*, 2
ocv_hog *(ocv_face)*, 2
ocv_info *(ocv_face)*, 2
ocv_keypoints, 4
ocv_knn *(ocv_face)*, 2
ocv_markers *(ocv_face)*, 2
ocv_mog2 *(ocv_face)*, 2
ocv_picture, *2*
ocv_picture *(ocv_face)*, 2
ocv_polygon *(opencv-area)*, 6
ocv_qr_detect, 5, *5*
ocv_read, *2*
ocv_read *(ocv_face)*, 2
ocv_read(), *3*, *5*
ocv_rectangle *(opencv-area)*, 6
ocv_resize *(ocv_face)*, 2
ocv_sketch *(ocv_face)*, 2
ocv_stylize *(ocv_face)*, 2
ocv_version *(ocv_face)*, 2
ocv_video, *2*
ocv_video *(ocv_face)*, 2
ocv_write, *2*
ocv_write *(ocv_face)*, 2
opencv-area, 6

qr_scanner *(ocv_qr_detect)*, 5