[OpenCV] [cheatsheet]

Basic Image Operations:

- **Read Image**: `image = cv2.imread('image.jpg')`
- **Show Image**: `cv2.imshow('Image', image)`
- **Save Image**: `cv2.imwrite('output.jpg', image)`
- **Convert to Grayscale**: `gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`
- **Resize Image**: `resized_image = cv2.resize(image, (width, height), interpolation=cv2.INTER_AREA)`
- **Crop Image**: `cropped_image = image[y1:y2, x1:x2]`
- **Rotate Image**: `rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale); rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))`
- **Flip Image**: `flipped_image = cv2.flip(image, flipCode)`

Image Filtering and Enhancement:

- **Blur Image**: `blurred_image = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigmaX)`
- **Median Filter**: `median_filtered_image = cv2.medianBlur(image, ksize)`
- **Bilateral Filter**: `bilateral_filtered_image = cv2.bilateralFilter(image, d, sigmaColor, sigmaSpace)`
- **Image Thresholding**: `ret, thresholded_image = cv2.threshold(gray_image, threshold_value, max_value, threshold_type)`
- **Adaptive Thresholding**: `adaptive_thresholded_image = cv2.adaptiveThreshold(gray_image, max_value, adaptiveMethod, threshold_type, blockSize, C)`
- **Histogram Equalization**: `equalized_image = cv2.equalizeHist(gray_image)`

Edge Detection and Feature Extraction:

- **Canny Edge Detection**: `edges = cv2.Canny(image, threshold1, threshold2)`

By: Waleed Mousa

- **Sobel Operator**: `sobelx = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)`
- **Scharr Operator**: `scharrx = cv2.Scharr(gray_image, cv2.CV_64F, 1, 0)`
- **Harris Corner Detection**: `corners = cv2.cornerHarris(gray_image, blockSize, ksize, k)`
- **Shi-Tomasi Corner Detection**: `corners = cv2.goodFeaturesToTrack(gray_image, maxCorners, qualityLevel, minDistance)`

Feature Matching:

- **ORB (Oriented FAST and Rotated BRIEF)**: `orb = cv2.ORB_create(); keypoints, descriptors = orb.detectAndCompute(image, mask=None)`
- **SIFT (Scale-Invariant Feature Transform)**: `sift = cv2.SIFT_create(); keypoints, descriptors = sift.detectAndCompute(gray_image, mask=None)`
- **SURF (Speeded-Up Robust Features)**: `surf = cv2.xfeatures2d.SURF_create(); keypoints, descriptors = surf.detectAndCompute(gray_image, None)`

Image Segmentation:

- **Simple Thresholding**: `ret, thresholded_image = cv2.threshold(gray_image, threshold_value, max_value, cv2.THRESH_BINARY)`
- **Contour Detection**: `contours, hierarchy = cv2.findContours(thresholded_image, mode, method)`
- **Connected Component Analysis (CCA)**: `num_labels, labels = cv2.connectedComponents(thresholded_image)`

Image Morphology:

- **Erosion**: `eroded_image = cv2.erode(image, kernel, iterations)`
- **Dilation**: `dilated_image = cv2.dilate(image, kernel, iterations)`
- **Opening**: `opened_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)`
- **Closing**: `closed_image = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)`

By: Waleed Mousa

- **Gradient**: `gradient_image = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)`
- **Top Hat**: `tophat_image = cv2.morphologyEx(image, cv2.MORPH_TOPHAT, kernel)`
- **Black Hat**: `blackhat_image = cv2.morphologyEx(image, cv2.MORPH_BLACKHAT, kernel)`

Image Transformation and Warping:

- **Affine Transformation**: `affine_matrix = cv2.getAffineTransform(src_points, dst_points); transformed_image = cv2.warpAffine(image, affine_matrix, (width, height))`
- **Perspective Transformation**: `perspective_matrix = cv2.getPerspectiveTransform(src_points, dst_points); transformed_image = cv2.warpPerspective(image, perspective_matrix, (width, height))`

Image Contours and Shape Analysis:

- **Bounding Rectangle**: `x, y, w, h = cv2.boundingRect(contour)`
- **Bounding Rotated Rectangle**: `rect = cv2.minAreaRect(contour)`
- **Minimum Enclosing Circle**: `center, radius = cv2.minEnclosingCircle(contour)`
- **Approximate Polygonal Curve**: `epsilon = 0.1 * cv2.arcLength(contour, True); approx_curve = cv2.approxPolyDP(contour, epsilon, True)`
- **Convex Hull**: `hull = cv2.convexHull(points)`

Object Detection and Recognition:

- **Haar Cascades Object Detection**: `faces = face_cascade.detectMultiScale(gray_image, scaleFactor, minNeighbors)`
- **Cascade Classifier Training**: `cascade_classifier = cv2.CascadeClassifier(); cascade_classifier.train(filenames, object_labels)`
- **Cascade Classifier Loading**: `cascade_classifier = cv2.CascadeClassifier('cascade.xml')`
- **Object Recognition with DNN**: `net = cv2.dnn.readNetFromTensorflow(model, config)`

By: Waleed Mousa

Image Features and Descriptors:

- **ORB Descriptors**: `orb = cv2.ORB_create(); keypoints, descriptors = orb.detectAndCompute(image, mask=None)`
- **SIFT Descriptors**: `sift = cv2.SIFT_create(); keypoints, descriptors = sift.detectAndCompute(gray_image, mask=None)`
- **SURF Descriptors**: `surf = cv2.xfeatures2d.SURF_create(); keypoints, descriptors = surf.detectAndCompute(gray_image, None)`

Image Histograms and Color Spaces:

- **Histogram Calculation**: `histogram = cv2.calcHist([image], channels, mask, histSize, ranges)`
- **Histogram Backprojection**: `backprojection = cv2.calcBackProject([image], channels, histogram, ranges, scale)`
- **Color Space Conversion**: `converted_image = cv2.cvtColor(image, conversion_code)`

Camera Calibration and 3D Reconstruction:

- **Camera Calibration**: `ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints, imageSize, None, None)`
- **Undistort Image**: `undistorted_image = cv2.undistort(image, mtx, dist, None, newCameraMatrix)`
- **3D Reconstruction from Stereo Images**: `disparity_map = stereo.compute(gray_image1, gray_image2)`
- **Depth Map from Stereo Images**: `depth_map = cv2.reprojectImageTo3D(disparity_map, Q)`

Video Processing:

- **Read Video Stream**: `cap = cv2.VideoCapture('video.mp4')`
- **Read Frames from Video**: `ret, frame = cap.read()`
- **Write Video Stream**: `out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))`
- **Write Frame to Video**: `out.write(frame)`

By: Waleed Mousa

Optical Flow and Motion Detection:

- **Lucas-Kanade Optical Flow**: `lk_params = dict(winSize=(15, 15), maxLevel=2, criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)); nextPts, status, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)`
- **Background Subtraction**: `fgmask = fgbg.apply(frame)`

Image Utilities:

- **Image Blending**: `blended_image = cv2.addWeighted(image1, alpha, image2, beta, gamma)`
- **Image Pyramids**: `pyramid = cv2.pyrDown(image)`
- **ROI (Region of Interest)**: `roi = image[y:y+h, x:x+w]`

Image Annotation and Drawing:

- **Draw Line**: `cv2.line(image, start_point, end_point, color, thickness)`
- **Draw Rectangle**: `cv2.rectangle(image, start_point, end_point, color, thickness)`
- **Draw Circle**: `cv2.circle(image, center, radius, color, thickness)`
- **Draw Text**: `cv2.putText(image, text, org, fontFace, fontScale, color, thickness)`

Image Augmentation and Transformation:

- **Image Translation**: `translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]]); translated_image = cv2.warpAffine(image, translation_matrix, (width, height))`
- **Image Scaling**: `scaling_matrix = np.float32([[sx, 0, 0], [0, sy, 0]]); scaled_image = cv2.warpAffine(image, scaling_matrix, (width, height))`
- **Image Rotation**: `rotation_matrix = cv2.getRotationMatrix2D(center, angle, scale); rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))`

By: Waleed Mousa

Image Stitching and Panorama:

- **Feature Detection (SIFT, SURF, ORB)**: `keypoints, descriptors = sift.detectAndCompute(gray_image, None)`
- **Feature Matching (FLANN)**: `matcher = cv2.FlannBasedMatcher(index_params, search_params); matches = matcher.knnMatch(descriptors1, descriptors2, k=2)`
- **Homography Estimation**: `M, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, ransacReprojThreshold)`

Image Segmentation and Clustering:

- **K-Means Clustering**: `criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0); ret, label, center = cv2.kmeans(data, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)`
- **Mean Shift Clustering**: `ret, labels, centers = cv2.meanShift(probImage, window, criteria)`

Facial Recognition and Detection:

- **Face Detection with Haar Cascades**: `faces = face_cascade.detectMultiScale(gray_image, scaleFactor, minNeighbors)`
- **Facial Landmark Detection**: `facial_landmarks = predictor(gray_image, face)`
- **Face Recognition with Dlib**: `face_descriptor = facerec.compute_face_descriptor(image, shape)`
- **Facial Expression Recognition**: `emotion = emotions[np.argmax(model.predict(face))]`

Object Tracking:

- **Object Tracking with Meanshift**: `ret, track_window = cv2.meanShift(prob_image, track_window, term_crit)`
- **Object Tracking with CAMShift**: `ret, track_window = cv2.CamShift(prob_image, track_window, term_crit)`

Camera and Video Stream Processing:

By: Waleed Mousa

- **Capture Video from Camera**: `cap = cv2.VideoCapture(0)`
- **Capture Frames from Camera**: `ret, frame = cap.read()`
- **Release Camera**: `cap.release()`
- **Display Video Stream**: `cv2.imshow('Frame', frame)`

Image Alignment and Registration:

- **Image Registration with ECC (Enhanced Correlation Coefficient)**: `warp_matrix = cv2.findTransformECC(template, image, warp_matrix, motionType, criteria)`

Optical Character Recognition (OCR):

- **Text Detection with EAST (Efficient and Accurate Scene Text Detection)**: `scores, geometry = cv2.text.detectText(image, confThreshold, nmsThreshold)`

Medical Image Processing:

- **DICOM Image Reading**: `import pydicom; ds = pydicom.dcmread('image.dcm')`

Augmented Reality:

- **Augmented Reality with ArUco Markers**: `corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(image, dictionary, parameters=parameters)`

Background Subtraction:

- **Background Subtraction with MOG2**: `fgbg = cv2.createBackgroundSubtractorMOG2()`

Stereo Vision:

- **Disparity Map Calculation**: `stereo = cv2.StereoBM_create(numDisparities, blockSize)`

Video Compression and Encoding:

By: Waleed Mousa

- **Codec Selection for Video Writing**: `fourcc = cv2.VideoWriter_fourcc(*'XVID')`

Lane Detection:

- **Lane Detection with Hough Transform**: `lines = cv2.HoughLinesP(image, rho, theta, threshold, np.array([]), minLineLength, maxLineGap)`

Human Pose Estimation:

- **Human Pose Estimation with OpenPose**: `keypoints, _ = pose.process(image)`

Gesture Recognition:

- **Hand Gesture Recognition**: `gesture = classifier.predict(hand_features)`

Optical Character Recognition (OCR):

- **OCR with Tesseract**: `import pytesseract; text = pytesseract.image_to_string(image)`

Point Cloud Processing:

- **Point Cloud Visualization**: `import open3d; pcd = open3d.io.read_point_cloud('point_cloud.ply')`

3D Object Detection and Tracking:

- **3D Object Detection**: `objects = detector.detect(image)`

Real-Time Object Detection:

- **Real-Time Object Detection with YOLO (You Only Look Once)**: `net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)`

Content-Based Image Retrieval (CBIR):

By: Waleed Mousa

- **CBIR using Histogram Comparison**: `score = cv2.compareHist(hist1, hist2, method)`

Pose Estimation:

- **Human Pose Estimation with PoseNet**: `keypoints, _ = posenet.process(image)`

Image Restoration:

- **Image Deblurring**: `deblurred_image = cv2.deblur(image, kernel)`

Image Segmentation:

- **Image Segmentation with GrabCut**: `mask, bgdModel, fgdModel = cv2.grabCut(image, mask, rect, bgdModel, fgdModel, iterCount, mode)`

Image Denoising:

- **Image Denoising with Non-Local Means**: `denoised_image = cv2.fastNlMeansDenoisingColored(image, None, h, hColor, templateWindowSize, searchWindowSize)`

Camera Calibration:

- **Camera Calibration with Zhang's Method**: `ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints, imageSize, None, None)`

Text Recognition:

- **Text Detection with EAST (Efficient and Accurate Scene Text Detection)**: `scores, geometry = cv2.text.detectText(image, confThreshold, nmsThreshold)`

Barcode Detection:

- **Barcode Detection with ZBar**: `decoded_objects = pyzbar.decode(image)`

By: Waleed Mousa

Video Tracking:

- **Object Tracking with KLT (Kanade-Lucas-Tomasi) Tracker**: `trackers = cv2.MultiTracker_create(); success, boxes = trackers.update(frame)`

Image Annotation:

- **Image Annotation with Arrows and Text**: `annotated_image = cv2.arrowedLine(image, start_point, end_point, color, thickness); annotated_image = cv2.putText(image, text, org, fontFace, fontScale, color, thickness)`

Image Registration:

- **Image Registration with ECC (Enhanced Correlation Coefficient)**: `warp_matrix = cv2.findTransformECC(template, image, warp_matrix, motionType, criteria)`

Image Conversion:

- **Convert Image to Grayscale**: `gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`

Video Stream Processing:

- **Capture Frames from Video Stream**: `ret, frame = cap.read()`

Image Morphology:

- **Erosion and Dilation**: `eroded_image = cv2.erode(image, kernel, iterations); dilated_image = cv2.dilate(image, kernel, iterations)`

Face Recognition:

- **Face Recognition with LBPH (Local Binary Patterns Histograms)**: `recognizer = cv2.face.LBPHFaceRecognizer_create(); recognizer.train(faces, labels)`

Feature Extraction:

By: Waleed Mousa

- **ORB Feature Detection and Description**: `orb = cv2.ORB_create(); keypoints, descriptors = orb.detectAndCompute(image, mask=None)`

Optical Flow:

- **Dense Optical Flow**: `flow = cv2.calcOpticalFlowFarneback(prev, next, flow, pyr_scale, levels, winsize, iterations, poly_n, poly_sigma, flags)`

Stereo Vision:

- **Stereo Correspondence Matching**: `disparity = stereo.compute(left_image, right_image)`

Image Utilities:

- **Image Arithmetic Operations**: `result = cv2.add(image1, image2)`

Image Quality Assessment:

- **Image Quality Metrics**: `ssim_value = skimage.metrics.structural_similarity(image1, image2, full=True)[0]`

Feature Matching:

- **Feature Matching with FLANN (Fast Library for Approximate Nearest Neighbors)**: `matcher = cv2.FlannBasedMatcher(index_params, search_params); matches = matcher.knnMatch(descriptors1, descriptors2, k=2)`

Real-Time Object Detection:

- **Real-Time Object Detection with SSD (Single Shot Multibox Detector)**: `net = cv2.dnn.readNetFromCaffe(prototxt, model)`

Background Subtraction:

- **Background Subtraction with MOG (Mixture of Gaussians)**: `fgbg = cv2.bgsegm.createBackgroundSubtractorMOG()`

By: Waleed Mousa

Content-Based Image Retrieval:

- **CBIR with Histogram Matching**: `score = cv2.compareHist(hist1, hist2, method)`