Computer Vision - Exercise 1

In this exercise I was asked to use OpenCV to complete two tasks. The first task was to use photos taken of a chessboard in perform a camera calibration. The second was to detect a known object within a scene to produce a perspective corrected output image of that object.

# Task 1 – Camera Calibration

Many devices today make use of cheap pinhole camera in order to capture photo and video, however due to the nature of these cameras, photos can become distorted. Both radial and tangential distortion need to be taken into account in order to calibrate the camera and produce an undistorted image. Before distortion can be calculated both the intrinsic and extrinsic parameters of the camera need to be found. These parameters can be found by decomposing the camera matrix from the complete camera model below:

Homogenous Image Point

Intrinsic Parameters

Extrinsic Parameters

Homogenous Object Point

The intrinsic parameters define the properties of the camera, namely the focal lengths (measured in pixels) and optical centres of the camera. The intrinsic matrix can be seen as a series of 2D transformations in which take points in the image plane transform them to pixels on the image post projection. The focal length is the distance measured in pixels from the pinhole of the camera to the image pane, the values are given by:

Where and are the scale factors that relate pixels and distance and are typically the same value. The ratio between these two values is known as the aspect ratio.

The Principle Axis of a camera is the line perpendicular to the image plane that passes through the pinhole. The Principle point is located on the intersection of this line and the image plane. The offset of the principle point from the image plane origin in the x and y axis is described by respectively, if the optical centre is aligned with the origin of the image plane (also in the centre) with square pixels and no skew then will both be 0.

Skew between the x and y axis is defined by . Skew will cause a shear distortion within the projected image.

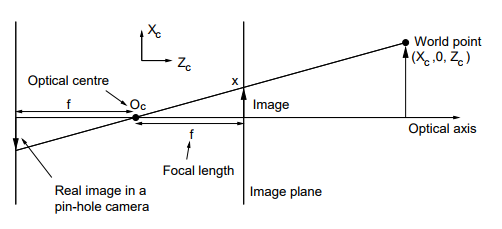


Image taken from Roberto Cipolla’s lecture slides

Extrinsic parameters define the cameras position and orientation within the world with being the rotation matrix and being the transformation matrix from the camera origin to the world origin. When combined these two provide a transformation from the 3D world coordinates to the camera’s 3D coordinate system.

To calculate the distortion of my webcam I built an OpenCV application in Python that loads a series of images containing a chessboard pattern. Once an image is loaded I made use of the “findChessboardCorners” function which when given an image and a pattern size, attempts to find a chessboard of that size in the image and return its approximate corners. Once a pattern has been found its corners are more accurately obtained using the “cornerSubPix” function. The object points and image points of the corners in the pattern are then mapped and stored, due to the nature of the chessboard pattern its object points can be described as a 2D square grid (0,0,0), (1,0,0) … (rows-1, columns-1, 0), as it is assumed that the chessboard was kept in the same XY plane.

Once enough object point – image points mappings have been found a camera calibration can be performed. OpenCV provides a “calibrateCamera” function that when given mapping of object points to image points returns the estimated values of the internal and external parameters.

In the first step of the algorithm the initial intrinsic parameters are calculated, and the distortion coefficients are set to 0. Then the function makes use of another function “solvePnP” to estimate the initial camera pose (extrinsic parameters) using the initial intrinsic parameters. The solvePnP function does this using the object point – image point mapping that was obtained by finding the chessboard patterns, along with the initial estimates for the camera matrix. Once these initial parameters have been calculated the algorithm can use the camera matrix to calculate the reprojection error (the total sum of the squared distances between the observed image points and the projected image points), using the “projectPoints” function.

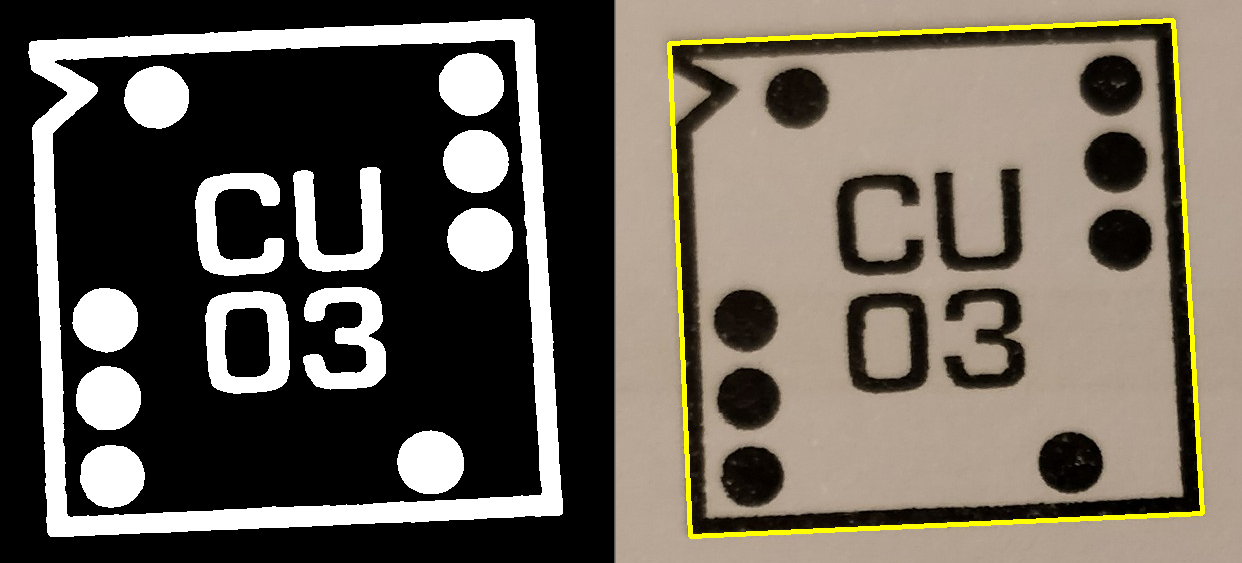
After the algorithm has these initial values; a global Levenberg-Merquardt optimization algorithm is performed to try and find a global minimum reprojection error. Since the algorithm is using a series of image points the global minimum will be the lowest average reprojection error. Once the algorithm has terminated it will return the intrinsic and extrinsic matrices that give the lowest error.

# Task 2 - Perspective correction

For the second task I was asked to take a set of photos of a stamp impression. Then, using OpenCV, write code that loads each image in turn and detects the outer corners of the stamp. Using these corners, the code should then perform a perspective correction that produces a “top down” view of the stamp.

There was a vagueness here in what constituted a photo of the stamp. If it was just a photo of only the stamp then a contour based approach can be used to find the corners of the stamp, if not and the stamp makes up a small portion of the photo then a more robust feature matching approach would be needed. But I assume that a photo of only the stamp in this case.

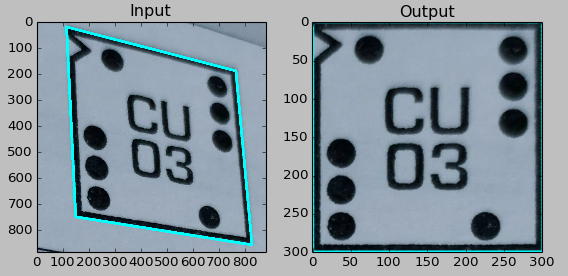
Since the stamp is the only feature in the image a contour approach can be used to find the outer corners. To find the contours I made use of OpenCV’s “findContours” function, but first I had to process the images to make them easier to work with. I converted the input image to grayscale and applied a gaussian blur to smooth the edges of the stamp and make sure small parts of an edge are not being recognised as a contour. After this I converted the Image to a binary image using the “threshold” function. For each pixel in the image the threshold function will assign it either a black or white value depending on where the pixels value is above or below the threshold, the black and white in the image is then inverted as the findContour function treats black as the background and white as the object. Since contours are defined as a curve of continuous points with the same colour intensity making the image binary within the appropriate threshold makes sure that the edges of the stamp all have the same colour intensity, thus allowing the findContour function to find the edges with greater accuracy. The function itself uses an algorithm that segments the image and then applies region merging to create the contours.

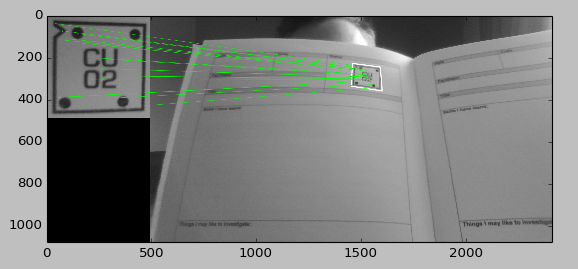
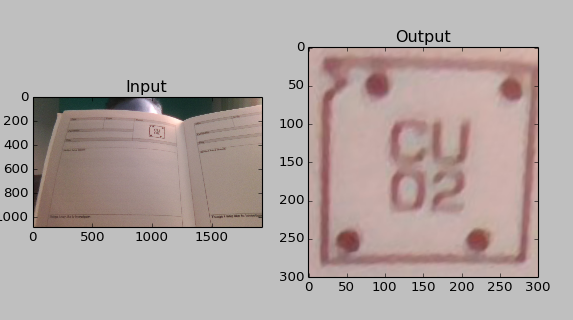


To find the outer edge of the stamp I simply searched through all the found contours for the largest contour, since the outer edge of the stamp is the largest edge. Once the outside edge has been found it must be reduced to the 4 points that make up its corners, to find these points I used contour approximation. OpenCV provides a “approxPolyDP” function that allows for a shape of a contour to be approximated using a value that determines the maximum distance from contour to approximated contour. This function is an implementation of the Ramer–Douglas–Peucker algorithm which takes a curve composed of line segments (the contour, made of many points) and returns a similar curve with fewer points (based on the value, a greater value creates a line with less points). The algorithm marks the start and end points to be kept and then recursively divides the line drawing segments between the Start, End and furthest away point and discarding and points that are closer than to the line segments. Given an appropriate the approximate shape will be a quadrilateral drawn from 4 points.

Using these 4 points I was the able to perform a perspective transformation. By mapping the points of the stamp to their corresponding desired output corners (this was just matching top-left coordinate of the stamp to top left corner of the correct output and so on) I could use the “getPerspectiveTransform” function to calculate the 3 x 3 matrix M such that the coordinates of the quadrangle vertices in the source image multiplied by the matrix give their specified coordinates in the destination image.

I could then use this matrix along with information about the destination image size and the source image to perform a perspective transformation on the image. Using the “warpPerspective” function I obtained the following result:



While this works fine if it is just the stamp in the image to find the stamp when there are other features in the scene would require feature descriptors and matching to locate the stamp in the scene based off a query image:

A perspective correction with 3 points is possible under the conditions that the correction can be described as an affine transformation. Affine transformations preserve parallel lines, but lengths and angles are not preserved and as such it is useful for rotating and transforming a plane, but it cannot be used to produce a ‘depth’ effect on a plane where all lines are converging to infinity.

Knowledge of the intrinsic and extrinsic parameters of the camera can help perform a perspective transform as using these parameters the distortion coefficients can be calculated, and the image can be undistorted. This in turn will mean that the source image and coordinates that are used to calculate the transformation matrix will be more accurate leading to a better correction.