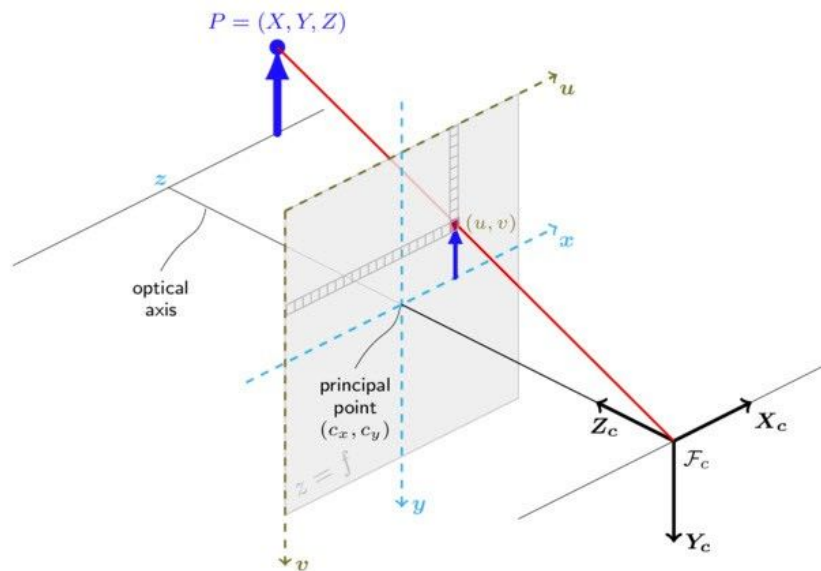


Discovery

Theory

Pinhole Camera Model

In camera calibration, we are exploiting the mathematical properties of the pinhole camera model to take a point from a 3d coordinate system and project it into the 2d coordinate system of the image. At the core of this calibration procedure we are using a set of known 3d points in the target coordinate system and their corresponding location in the image to solve for the model parameters. This ability to perceive depth and distance using only one eye or camera is considered monocular vision.



To obtain the point (u, v) in 2d image space we take a point in 3d space (x, y, z) and apply the following transformations

$$\begin{array}{c} \text{Given this} \end{array} \quad \begin{array}{c} \text{Find This} \end{array}$$
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where:

- (X, Y, Z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- A is a camera matrix, or a matrix of intrinsic parameters
- (c_x, c_y) is a principal point that is usually at the image center
- f_x, f_y are the focal lengths expressed in pixel units.

This equation shows that (u, v) is a function of the camera's intrinsic and extrinsic parameters, multiplied by some scale value, s . The matrix containing f_x and f_y represents part of the intrinsic parameters to describe the effect of the camera from manufacturing. The matrix with r and t values represent the camera's extrinsic parameters, or the pose with respect to the world coordinate system we are imaging.

f_x = focal length in the x-direction

f_y = focal length in the y-direction

c_x = principal point x-coordinate

c_y = principal point y-coordinate

Not represented above is the polynomial coefficients we use to estimate the radial and tangential distortion from the lens.

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Radial Distortion Polynomial

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Tangential Distortion Polynomial

k_1 = Radial distortion coefficient 1
k_2 = Radial distortion coefficient 2
p_1 = Tangential distortion coefficient 1
p_2 = Tangential distortion coefficient 2
k_3 = Radial distortion coefficient 3
r is the distance from the center of the image



Effects of Distortion

So in order to calibrate our camera for real world measurements, we must estimate these coefficients using proper calibration techniques. With OpenCV we can invoke `calibrateCamera` to estimate the values. In a pristine environment such as blender simulation, we can use theoretical values for the intrinsic parameters.

To start, we can set all distortion coefficients equal to zero. The coefficients **c_x** and **c_y** are set to the exact center of the image. To obtain camera focal length in pixels from real world focal length and sensor width, or vice versa we have:

$$F(\text{mm}) = F(\text{pixels}) * \text{Sensor Width}(\text{mm}) / \text{Image Width}(\text{pixel}).$$

Note it's common to specify **f_x = f_y**. This approach can be useful in scenarios where you want to maintain consistent framing and coverage irrespective of whether the image is in landscape or portrait orientation. That is, use the maximum of the width and height if you want to ensure that the field of view (FOV) of your camera to cover the entire image, regardless of its aspect ratio. If you want to maintain consistent coverage across images with different aspect ratios, using the maximum dimension is a suitable choice. If you want to maintain a consistent FOV along the shorter side, then using the minimum dimension might be more appropriate.

If we fix the intrinsic parameters, all that's left to do is find the pose of the camera with respect to the world coordinate system and the appropriate scale values to use. The pose of the camera can be estimated with OpenCV's `solvePnP` function.

Perspective Transformation

Briefly, we can look at how to achieve perspective free images by using reference points in the source image and determining their target locations in the corrected image. Using only four points, the homography (perspective matrix) is calculated and applied to the image to remove the perspective effect. Below we see a photo of the empire state building taken from the ground, and the corrected image:

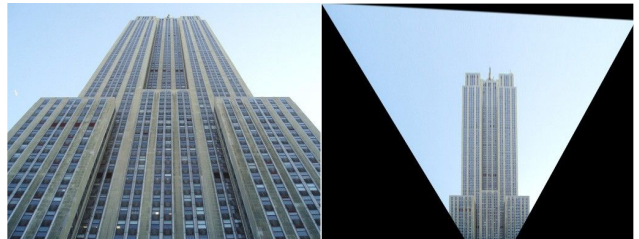
As another example, say we wanted to virtually replace a billboard--
this can be done with a perspective transform.



Original MoTown Billboard



Replacement Billboard



Perspective Correction of Empire State Building



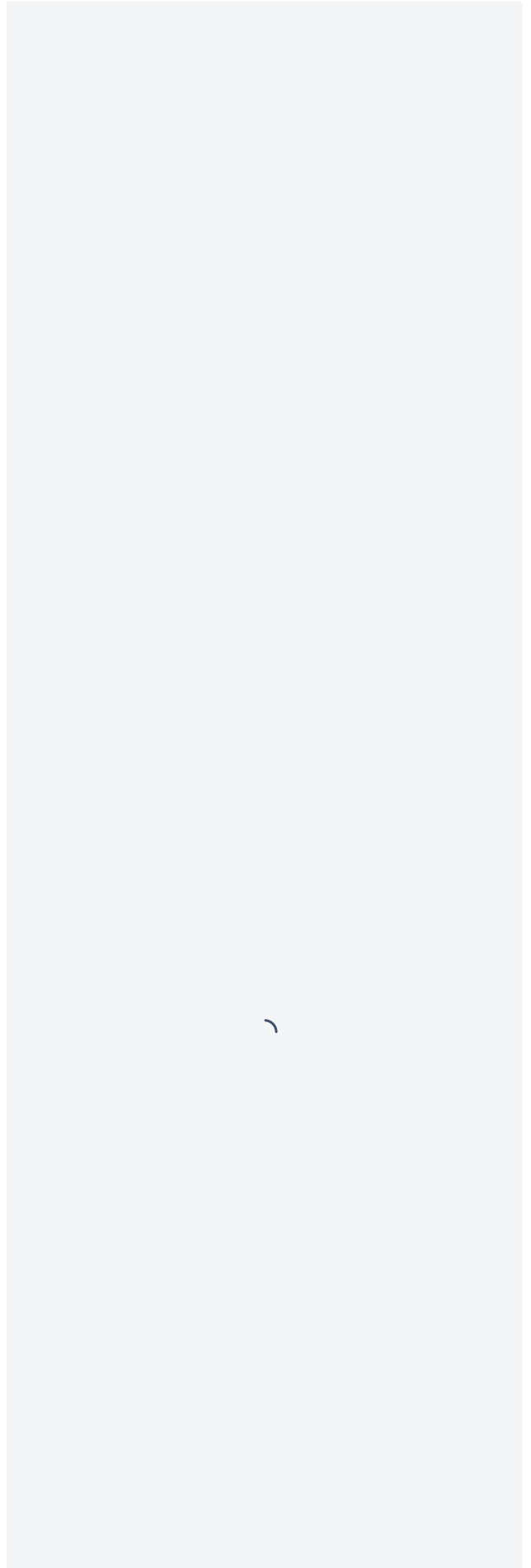
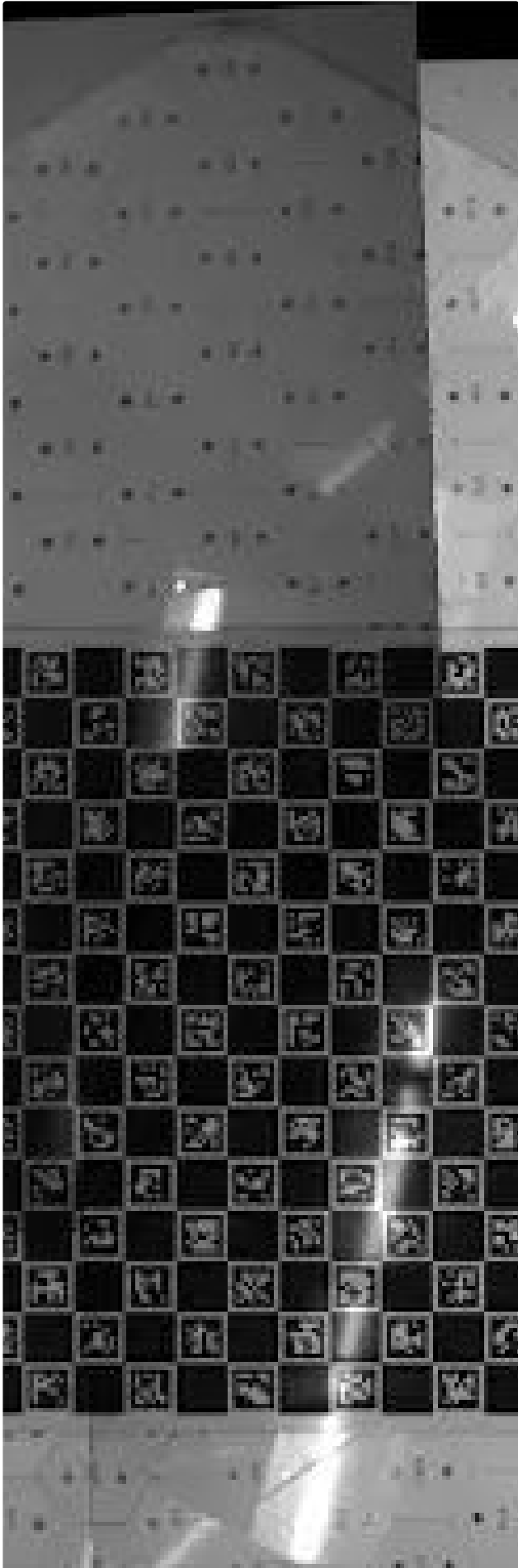
Virtual Billboard Replacement

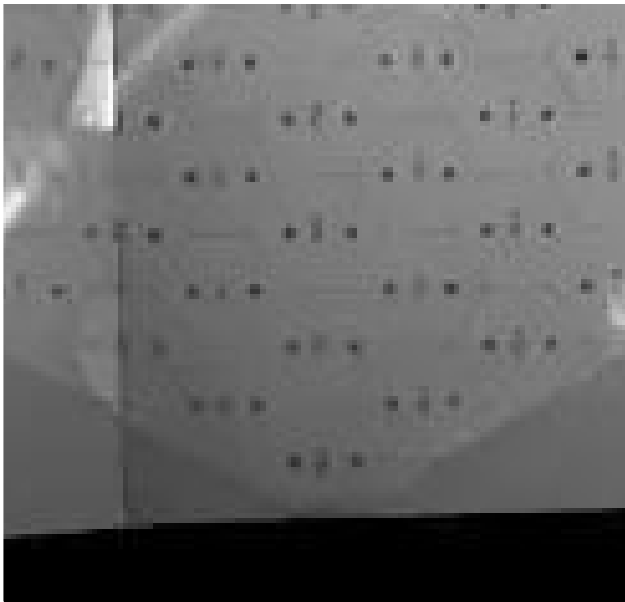
Within OpenCV we can make use of `getPerspectiveTransform` and `warpPerspective` to perform these operations.

Stitching

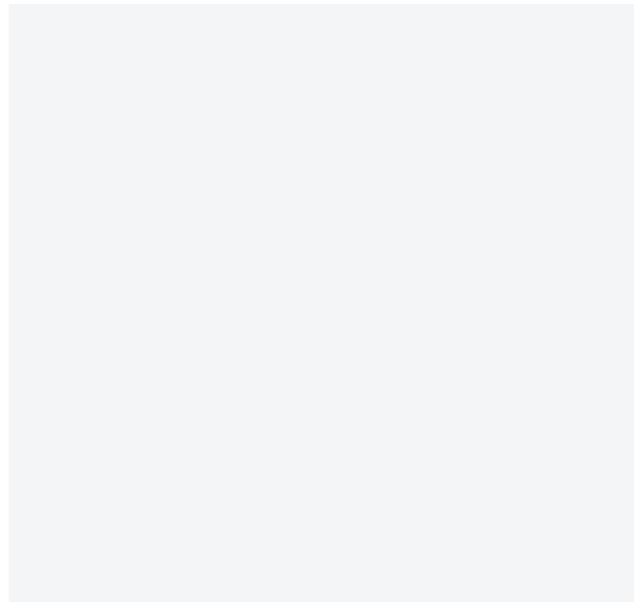
To obtain a larger field of view, we can also stitch multiple images together as long as there is an area of overlap between the two views. We use common features in the two images to find the homography transformation that relates the two images.

Below is an example of the reference used to calibrate such a system, and the type of stitching and real world measurement that can be done:



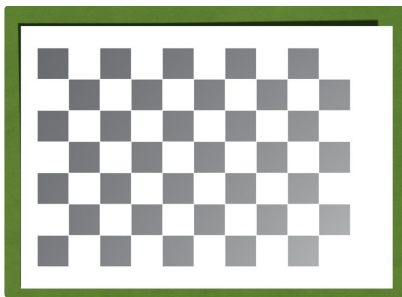


Pristine Image Stitching with CharUco Reference

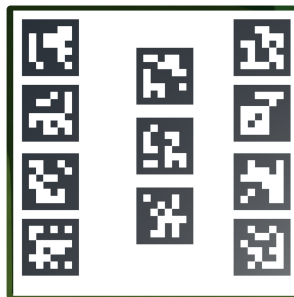


Mouse Tracking after Image Stitching

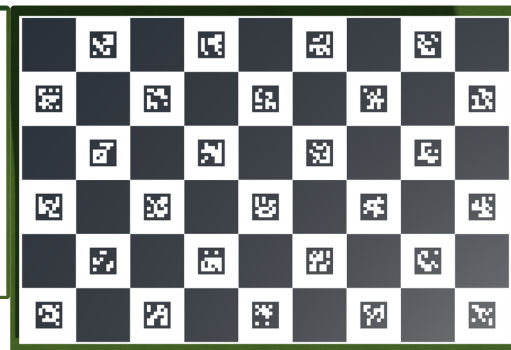
Calibration References



Chessboard



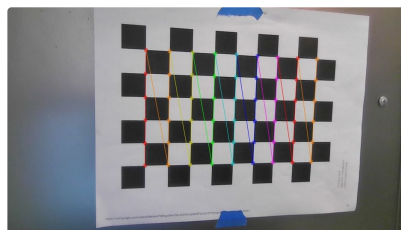
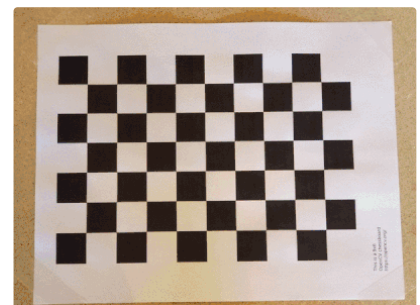
ArUco



CharUco

Among these three, charUco has the most structure to work with. Generally, to estimate the intrinsic parameters, we would capture images of this reference from multiple angles

In each one, we detect the internal corners of the chessboard within the image--this is the set of 2d coordinates. Since we are using a fixed reference, we already know the corresponding 3d real world coordinates of those 2d image points.




Detected Chessboard Corners

References

[Calculate X, Y, Z Real World Coordinates from Image Coordinates using OpenCV](#)

[OpenCV: Camera Calibration](#)

[Homography examples using OpenCV \(Python / C ++ \) |](#)

 [Image rectification and distance measurement using Homography - Intenseye](#)

 [Multi-View Image Stitching based on the pre-calibrated camera homographies.](#)