Technical Memorandum

Navigation and Mission Design Branch (Code 595)
Goddard Space Flight Center
NASA

Date: 21 April 2017
Memo: 595–17–001

To: ARRM Retina Analysis Team
From: Andrew Liounis

Title: A Comparison of the OpenCV and Owen Camera Models

1 Summary

This memo attempts to examine the similarities between the OpenCV and the Owen Camera matrices and distortion models. First, a brief review of gnomic projections and the pinhole camera model is presented, primarily as a way of introducing the notation to be used throughout this memo[1]. Next, a review of the components of each model is presented. Then, an attempt is made to examine how to map from one camera representation to another. Finally, a brief numerical comparison of the two representations is examined (possibly).

2 A Brief Review of Gnomic Projections and the Pinhole Camera Model

The pinhole camera model attempts to describe how objects in three-dimensional space are projected onto a two-dimensional plane to form an image. It assumes that the world is being viewed through a pinhole, such that light travels in a straight path from the object, through the pinhole, and onto the focal plane of the camera. Thus, the pinhole camera model is actually just a simple gnomic projection from $\mathbb{R}^3$ to $\mathbb{R}^2$. In this section we briefly develop the pinhole camera model, loosely following the description in [1].

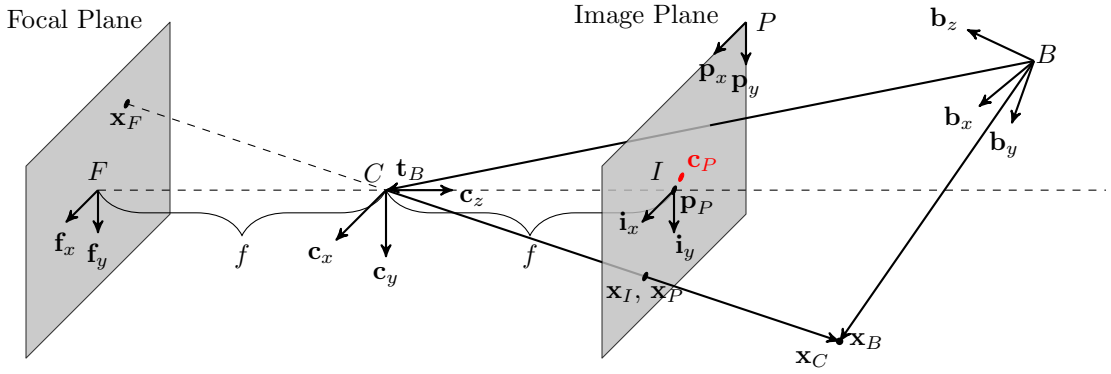


Figure 1: A point, $\mathbf{x}_B$, is projected onto the image plane through the pinhole camera model.

To develop the mathematics behind the pinhole camera model, consider the scene in Fig. 1. In the scene, we have a point, $\mathbf{x}_B$ defined in frame B. We want to project this point onto the focal plane of the camera

[1] A brief note on notation: throughout this memo bold lowercase variables will indicate vectors, bold uppercase variables will indicate matrices, and all other variables will be scalars

with focal length f and camera center located at $\mathbf{t}_B$ in frame B. Our first step is to express $\mathbf{x}_B$ in the camera frame (frame C). We can do this using the simple rotation and translation given by

$$\mathbf{x}_C = \mathbf{T}_C^B (\mathbf{x}_B - \mathbf{t}_B) \tag{1}$$

where $\mathbf{x}_C$ is point $\mathbf{x}_B$ expressed in the camera frame, $\mathbf{t}_B$ is the location of the camera center (and origin of the camera frame) in frame B, and $\mathbf{T}_C^B$ is a rotation matrix from frame B to the camera frame. Further, we can also express this transformation from frame B to frame C using homogeneous coordinates as

$$\mathbf{x}_C = \mathbf{T}_C^B \left[\begin{array}{c|c} \mathbf{I}_{3\times3} & -\mathbf{t}_B \end{array} \right] (\mathbf{x}_h)_B = \mathbf{E}(\mathbf{x}_h)_B \tag{2}$$

where $\mathbf{I}_{3\times3}$ is the 3×3 identity matrix, $(\mathbf{x}_h)_B$ is the homogeneous version of $\mathbf{x}_B$, and $\mathbf{E} = \mathbf{T}_C^B \left[\begin{array}{c|c} \mathbf{I}_{3\times3} & -\mathbf{t}_B \end{array} \right]$ is the extrinsic camera matrix (thus called because it is entirely dependent on the scene or external parameters).

Now that we have expressed our point of interest in the camera frame we can begin considering the projection. Start by examining the slice of the c_y–c_z plane from Fig. 1 shown in Fig. 2. As should be

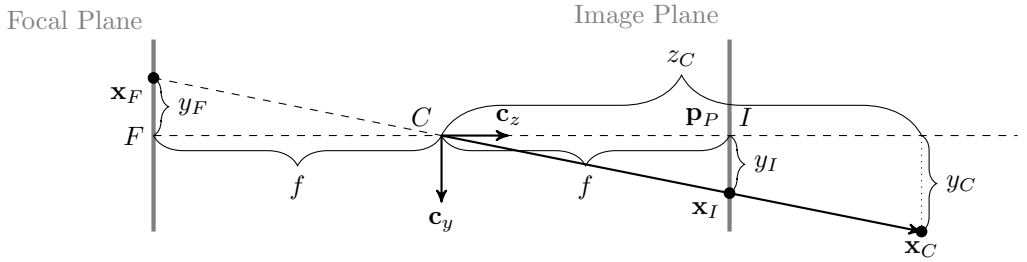


Figure 2: A slice of the c_y–c_z plane from the scene in Fig. 1

.

apparent, we are simply working with similar triangles. To determine the $y-$coordinate of the point in the focal plane, we just need to multiply by the scaling term f/z_c and flip the sign to account for the fact that we have crossed the principal axis. This allows us to express the coordinates where the point projects onto the focal plane as follows:

$$\mathbf{x}_F = -\frac{f}{z_C} \left[\begin{array}{c} x_C \\ y_C \end{array} \right] \tag{3}$$

where $\mathbf{x}_F$ is the point expressed in the focal frame and f is the focal length of the camera.

This is the simplest version of the pinhole camera model; however, in this case our image of the real world has actually been flipped upside down and left/right due to the fact that we crossed the principal axis on the way to the focal plane[2]. In most modern cameras, the image that is output after capturing a scene has been corrected to be in the same orientation as the world and this is what most people expect when they see an image. In order to account for the internal corrections of the camera it is common to skip using the focal plane projection and instead define a new imaginary "image plane" placed a distance of f in front of the camera center. Working in the image frame allows us to work with the image as it actually appears (and how scenes appear to our eyes). The only thing we need to change to work in this frame is the negative sign in front of equation 3 to give us

$$\mathbf{x}_I = \frac{f}{z_C} \left[\begin{array}{c} x_C \\ y_C \end{array} \right]. \tag{4}$$

where $\mathbf{x}_I$ is the location of the point in the image frame[3].

[2]As a fun thought experiment: this is also how our eyes work. The image received by our cones and rods is flipped upside down of the orientation of the objects in the real world. Presumably our brain then corrects this image to match the actual orientation of the real world... or does it? Perhaps what we see is actually the inverse of the real world and we would never know it because we have never experienced differently. Of course this might then imply that the image as it appears on the focal plane would appear in the correct orientation, or would it? Anyway, I digress.

[3]Note that if the $x-$ and $y-$axes of our picture frame are flipped from the $x-$ and $y-$axes of our image frame, we can account for this here by changing the sign of the $x-$term, $y-$term, or both terms to account for the flips.

One assumption that we have made here that is commonly broken is that the principal axis aligns with the origin of the coordinate system we are using in the image frame. Frequently, the coordinate system origin is placed in one of the corners of the image, (see frame P in Fig. 1 for an example). In addition, the principal axis may not always be perfectly aligned with the center of the image plane (note that the principal axis is by definition perpendicular to the image and focal planes). This then leads to two coordinate systems with origins on our image plane, the image frame, whose origin is located at the principal point (frame I in Fig. 1), and the picture frame, whose origin is located at the beginning of the coordinate system used to reference the image itself (frame P in Fig. 1). To correct for these differences, it is common to include an offset term to account for the difference between the principal point (point $\mathbf{p}$ in Fig. 1) and the origin of the picture coordinates (the picture frame, frame P in Fig. 1). This is given by

$$\mathbf{x}_P = \mathbf{x}_I + \mathbf{p}_P \tag{5}$$

where $\mathbf{x}_P$ is the location of the point in the picture frame and $\mathbf{p}_P$ is the location of the principal point in the picture frame. Further, we can describe the entire projection and translation as

$$\mathbf{x}_P = \frac{1}{z_C}\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \end{bmatrix}\mathbf{x}_C = \frac{1}{z_C}\mathbf{K}\mathbf{x}_C \tag{6}$$

where

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \end{bmatrix} \tag{7}$$

is the intrinsic camera matrix (thus called because it is entirely dependent on the camera itself, or the internal parameters) and p_x and p_y are the x and y components of the principal point expressed in the picture frame. Finally, we can define the complete camera matrix to be

$$\mathbf{C} = \mathbf{K}\mathbf{E} \tag{8}$$

such that we have

$$\mathbf{x}_p = \frac{1}{z_C}\mathbf{C}(\mathbf{x}_h)_B \tag{9}$$

as the full mapping from an arbitrary frame B to the picture frame of a camera. This completes the basic pinhole camera model.

In real life the pinhole camera model does not fully account for everything that is happening in a camera. Things like lens distortions, mis-alignment of the focal plane, and other issues cause differences between the results predicted by the pinhole camera model and the actual results from a camera. Luckily much work has gone into ways to account for these errors through what are called distortion models.

3 The OpenCV Model

Now that we have developed the basic pinhole camera model we can turn our attention to the modifications made by the OpenCV and Owen models that are being considered in this memo. First is the OpenCV model, for which we will follow the description given in the documentation for the 3.0.0-dev release [2].

The OpenCV camera model is very similar to the basic pinhole camera model with 2 minor differences. The first difference is that OpenCV does not assume an ideal camera and thus includes a distortion model to account for the effects of the lens and imaging system of the camera. To apply the distortion model, begin with the point of interest expressed in the camera frame and project it onto the plane located at $z = 1$ in the camera frame:

$$\begin{aligned} x' &= \frac{x_c}{z_c} \\ y' &= \frac{y_c}{z_c} \end{aligned} \tag{10}$$

where x' and y' are the coordinates of the point of interest projected onto the plane $z = 1$ in the camera frame.[4] The distortion model used by OpenCV considers radial distortion, tangential distortion, and thin prism distortion[5].

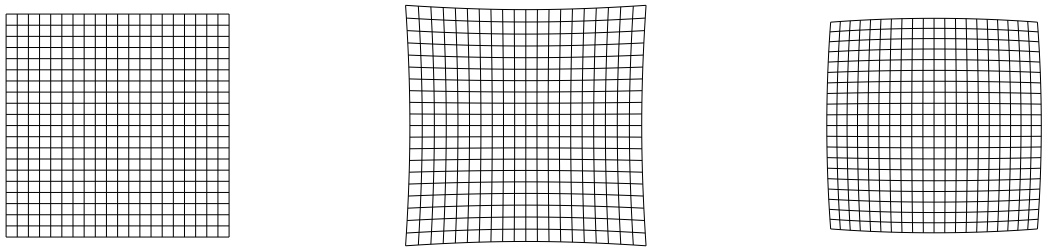
The radial distortion takes the form of a multiplicative scalar based on the radius from the origin of the $z = 1$ frame (the origin is located at the point where the camera frame $z-$axis pierces the $z = 1$ plane) and can be expressed as

$$d = \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \tag{11}$$

where $r^2 = x'^2 + y'^2$ is the radial distance from the origin and k_i are the radial distortion coefficients. The radial distortion is applied by multiplying the coordinates of the point in the $z = 1$ frame by d, that is

$$\begin{aligned} x'_{rad} &= dx' \\ y'_{rad} &= dy' \end{aligned} \tag{12}$$

where x'_{rad} and y'_{rad} are the locations in the $z = 1$ frame after the distortion has been applied. Radial distortions have a pincushion or barrel effect and are created by the way lenses bend the light. An example of radial distortion is shown in Fig. 3.



(a) None (b) Pincushion (c) Barrel

Figure 3: An example of the two different kinds of radial distortion caused by the OpenCV radial distortion model. The pincushion distortion shown here was created by setting $k_1 = 0.0005$ and $k_4 = 0.0001$ with all other terms set to 0. The barrel distortion was created by setting $k_1 = 0.0001$ and $k_4 = 0.0005$ with all other terms set to 0. Setting the other term values to nonzero amplify the effects shown here.

The tangential distortions take the form of additive errors that affect the x and y locations differently. They are given as

$$\begin{aligned} \Delta x'_{tan} &= 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ \Delta y'_{tan} &= p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{aligned} \tag{13}$$

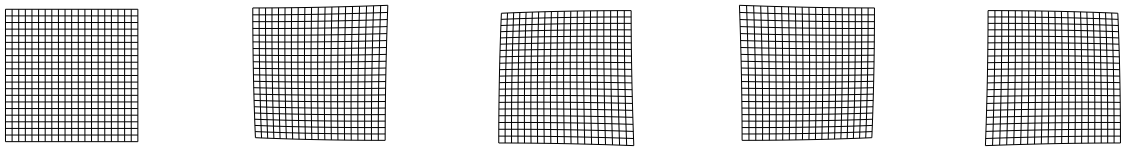
where p_1 and p_2 are the tangential distortion coefficients. Tangential distortions are mostly caused by lens elements that are not perfectly aligned with the principal axis [3][6]. An example of the effect of tangential distortions in shown in Fig. 4.

The final distortion accounted for by OpenCV (and only accounted for in OpenCV 3.0.0 [2]) are thin prism distortions. Thin prism distortions are caused by lens design/implementation issues as well as the sensor array not being placed perfectly perpendicular to the principal axis [3]. It creates a prism type effect on the resulting image. Thin prism distortions are modeled as:

$$\begin{aligned} \Delta x'_{prism} &= s_1 r^2 + s_2 r^4 \\ \Delta y'_{prism} &= s_3 r^2 + s_4 r^4 \end{aligned} \tag{14}$$

where s_{1-4} are the thin prism distortion coefficients (see [3] for a more complete breakdown of what goes into these coefficients)[7]. An example of the thin prism distortions is shown in Fig. 5.

[4]Note here that x' and y' are dimensionless here, which is a distinction between the two models that will be discussed later.
[5]The thin prism distortion is only available in OpenCV 3.0
[6]the tangential distortions are referred to as decentering distortions in [3].
[7]There appears to be an error in the OpenCV 3.0.0 documentation with the placement of the thin prism coefficients. The



(a) None (b) p_1+, p_2+ (c) p_1-, p_2+ (d) p_1+, p_2- (e) p_1-, p_2-

Figure 4: Examples of tangential distortions. Each subfigure is labeled with the signs of the tangential distortion coefficients that have been used to generate them.

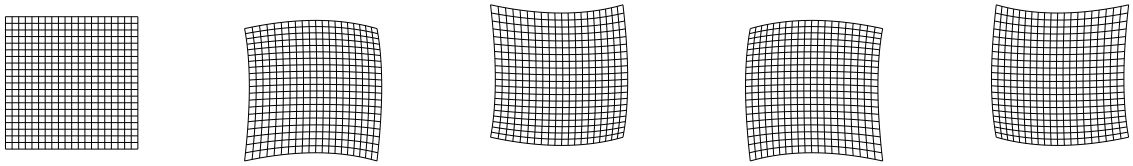


Figure 5: Examples of thin prism distortions. To be honest I am not sure that I am doing this correctly so I will not bother saying how I generated these. Take them with a grain of salt.

Now, all that remains is to combine all of the distortions together, and then to use the intrinsic camera matrix to transform from the non-dimensional space in the $z = 1$ plane to the actual picture space. To combine all the distortions we simply need to add the results of Eqs. 12-14. This results in

$$\begin{aligned} x'' &= x'_{rad} + \Delta x'_{tan} + \Delta x'_{prism} \\ y'' &= y'_{rad} + \Delta y'_{tan} + \Delta y'_{prism} \end{aligned} \tag{15}$$

where x'' and y'' are the distorted location coordinates for the point in the $z = 1$ plane [2]. To move from the $z = 1$ plane frame to the picture frame we simply need to multiply x'' and y'' by the focal length to convert to pixels, and then add the principal point offset. This is where the final distinction from the standard pinhole camera model occurs. In order to allow for rectangular instead of square pixels (or to account for square pixels that span different amounts of field of view) the OpenCV model estimates both an x and y focal length in pixels [1, 2]. Therefore we have

$$\begin{aligned} x_p &= f_x x'' + p_x \\ y_p &= f_y y'' + p_y \end{aligned} \tag{16}$$

or more succinctly

$$\mathbf{x}_p = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \end{bmatrix} \mathbf{x}''_h \tag{17}$$

where f_x and f_y are the x and y focal lengths respectively in pixels and $\mathbf{x}''_h = [\ x'' \quad y'' \quad 1\ ]^T$. This completes the OpenCV camera model.

4 The Owen Model

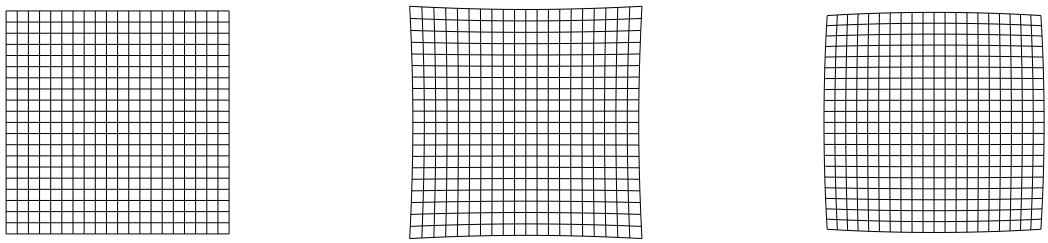
The Owen Model, as described in [4] and in the Stereophotoclinometry source code, is very similar to the basic pinhole camera model with a few slight modifications. The first modification is the inclusion of a distortion model to account for real world lenses. In this case, to apply the distortion, we first must transform our point in the camera frame to a point in the image plane as is done in Eq. 4. In this case f is the actual physical focal length of the camera (usually expressed in millimeters). Now that we are in the image space we can begin applying the distortions.

way that is currently described cannot be correct so attention will need to be paid to see when the documentation is corrected to get the actual placement of the coefficients

The first distortion type that is considered by the Owen model is a fourth order radial distortion given by

$$\Delta(\mathbf{x}_I)_{rad} = \epsilon_1 r^2 \mathbf{x}_I + \epsilon_2 r^4 \mathbf{x}_I \tag{18}$$

where $\epsilon_{1,2}$ are the radial distortion coefficients and $r = \sqrt{x_I^2 + y_I^2}$ is the radial distance from the principal point of the camera. Radial distortions take the form of pincushion or barrel effects and are caused by the way that lenses bend the light that passes through them. An example of the Owen radial distortion is shown in Fig. 6.



(a) None (b) Pincushion (c) Barrel

Figure 6: An example of the two different kinds of radial distortion. Pincushion distortion occurs when ϵ_1 and ϵ_2 are positive and barrel distortion occurs when ϵ_1 and ϵ_2 are negative. A mixing of the signs of ϵ_1 and ϵ_2 will lead to a mixing of the distortions.

The next distortion considered is a tangential distortion according to OpenCV and a decentering distortion in [3] (although it is referred to as a tip/tilt/prism distortion in [4]). This is given by

$$\Delta(\mathbf{x}_I)_{tan} = \epsilon_3 y_I \mathbf{x}_I + \epsilon_4 x_I \mathbf{x}_I \tag{19}$$

where $\epsilon_{3,4}$ are the tangential distortion coefficients and x_I and y_I are the coordinates of the point in the image frame. Tangential distortions are mostly caused by lens elements that are not perfectly aligned with the principal axis [3]. An example of the effect of tangential distortions in shown in Fig. 4 (note that the tangential model here is exactly the same as the tangential model for OpenCV so the distortions look the same. Do note however that the coefficients won't be the same since they operate in different planes with different units).

The final distortion considered by the Owen model is not documented anywhere and only seems to appear in the SPC version of the Owen model. It appears that it may be some form of radial distortion but the implementation is odd due to the cross with the x and y terms and the negative sign for the x terms. This distortion is given as

$$\Delta(\mathbf{x}_I)_{pin} = (\epsilon_5 r + \epsilon_6 r^3) \begin{bmatrix} -y_I \\ x_I \end{bmatrix} \tag{20}$$

where $\epsilon_{5,6}$ are the extra distortion coefficients. This distortion was used to generate the grids in Fig. 7. As can be seen, it appears to create something of a pinwheel[8] effect on the grids. [9]

The total distortion model is found by summing the terms from Eqs. 18-20:

$$(\mathbf{x}_I)_{dist} = \mathbf{x}_I + \Delta(\mathbf{x}_I)_{rad} + \Delta(\mathbf{x}_I)_{tan} + \Delta(\mathbf{x}_I)_{pin} \tag{21}$$

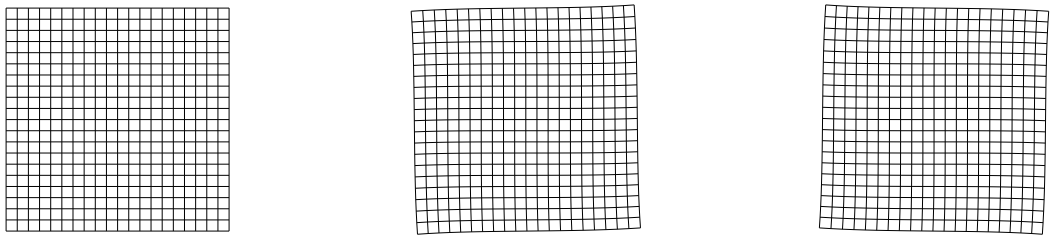
where $(\mathbf{x}_I)_{dist}$ is the distorted location of the point in the image frame.

Now all that remains is to move the points from the image frame to the picture frame. This is done using

$$\mathbf{x}_P = \begin{bmatrix} K_x & K_{xy} & p_x \\ K_{yx} & K_y & p_y \end{bmatrix} \begin{bmatrix} (\mathbf{x}_I)_{dist} \\ 1 \end{bmatrix} \tag{22}$$

[8] This term does not appear anywhere in the literature. I have come up with it myself

[9] Ask Dr. Gaskell about this!



(a) None (b) Clockwise (c) Counter-clockwise

Figure 7: Examples of radial pinwheel distortions. Clockwise pinwheels are generated with ϵ_5 and ϵ_6 being positive. Counter-clockwise pinwheels are generated with ϵ_5 and ϵ_6 being negative.

where the K terms scale/rotate into the picture frame and have units of pixels/distance (with distance being the same unit as the focal length)[4]. The K terms in Eq. 22 require some more discussion. First, in an ideal detector with square pixels, $K_x = K_y$ and $K_{xy} = K_{yx} = 0$ such that K_x and K_y would just serve as a unit conversion from distance to pixels [4]. If the pixels were rectangular then we would have that $K_x \neq K_y$ while K_{xy} and K_{yx} would still be 0 [4]. It is therefore apparent that K_x and K_y account for both the conversion from distance to pixels as well as potential differences in pixel size (whether by design or not). The off-diagonal terms only come into play if the x and y axes are not perfectly perpendicular due to a manufacturing error and therefore estimate the skewness of the imaging array. In the SPC version of the Owen model there are also two additional terms, K_{xxy} and K_{xyy} which are multiplied by product of the x and y components of the point $(\mathbf{x}_I)_{dist}$ and added to the picture frame coordinates. It is not apparent what these terms are supposed to measure based off of the SPC source code and they are not described in the literature. In addition, in practice it appears that these terms are usually 0, so we will not pay any further attention to them for now. [10]

In practice it is impossible to estimate all of the K parameters due to observability issues, therefore the following is common practice for implementation according to [4]. First, the value of K_x is held fixed at the manufacturer's specified value, while the value of K_y is allowed to vary to account for the non-squareness of the pixels. Second, the value for f is estimated in order to account for changes in overall scale. Finally, the value of K_{xy} is held fixed at 0 while the value of K_{yx} is allowed to vary in order to account for a non-perpendicular angle between the x and y axes of the detector. This completes the Owen camera model.

5 Relationships Between the Models

Now that we have developed the models for OpenCV and Owen independently we can begin to compare them. First, start by considering an ideal camera with no distortions. In this case the OpenCV model reduces to

$$\mathbf{x}_p = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \end{bmatrix} \mathbf{x}'_h = \begin{bmatrix} f_x \frac{x_c}{z_c} + p_x \\ f_y \frac{y_c}{z_c} + p_y \end{bmatrix} \tag{23}$$

where $\mathbf{x}'_h = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$. The Owen model reduces to (taking into account the "common practices for the K parameters)

$$\mathbf{x}_p = \begin{bmatrix} K_x & K_{xy} & p_x \\ K_{yx} & K_y & p_y \end{bmatrix} \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} = \begin{bmatrix} K_x f \frac{x_c}{z_c} + p_x \\ K_{yx} f \frac{x_c}{z_c} + K_y f \frac{y_c}{z_c} + p_y \end{bmatrix}. \tag{24}$$

From here it should be easy to see that $f_x = K_x f$ and $f_y = K_y f$ and the p_x and p_y are the same in both models. Further it should be noted that the only main difference between these models is that the Owen model allows for non-rectangular array elements while the OpenCV model does not[11]. It should be

[10] Ask Dr. Gaskell about this!

[11] It is my guess that the OpenCV model's more rigorous distortion model can handle the effects of non-perpendicular axes just as well as estimating the offset directly though I cannot actually confirm or deny this

noted that it is relatively easy to incorporate a skewness term into the OpenCV model but it would require modifying some of the source code and in this memo we are only considering the models as they are[12].

As for the distortion models used in each case, the comparison is more difficult. The OpenCV model applies distortion in a dimensionless frame located at the principal point on the $z = 1$ plane in the camera frame while the Owen model applies distortions in the image frame using units of distance. Neither of these techniques has any advantage over the other computationally as far as I can tell. The Owen model may be slightly more preferable to a human because it is frequently more intuitive to work in a frame with units of distance as opposed to being unit-less; however, the results should be the same. As to the distortion models themselves, the OpenCV model is more in-depth than the Owen model, specifically for the radial and prism components. It should also be noted that subsets of the OpenCV distortion model can be used (that is, you can estimate/use lower order radial distortions and choose to ignore the other distortion corrections if so desired). In terms of a direct comparison, the tangential model used by Owen is nearly exactly the same as the one used in OpenCV (with the exception of being applied in a dimensioned space versus a dimensionless space). The Owen model does not include a distortion to account for thin prism distortions (although the literature on the Owen model claims that the tangential distortions account for thin prism distortions) and for the radial distortion model, the OpenCV implementation allows a higher order of estimation than the Owen implementation.

6 A Brief Comparison of Performance

To be done later, if desired, time permitting.

7 Conclusion

This memo presents a review of the pinhole camera model, the OpenCV camera model, and the Owen Camera model. Along the way, a description of the effects of various image distortion types is included. The Owen and OpenCV camera models are then compared and similarities are developed.

Acknowledgments

References

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[2] "Camera calibration and 3d reconstruction." `http://docs.opencv.org/master/d9/d0c/group__calib3d.html`, July 2015.

[3] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 965–980, 1992.

[4] W. M. Owen Jr, "Methods of optical navigation," *Spaceflight Mechanics*, vol. 140, 2011.

[12] In fact, most estimators do estimate a skewness term despite the fact that OpenCV does not provide a way to use this value.