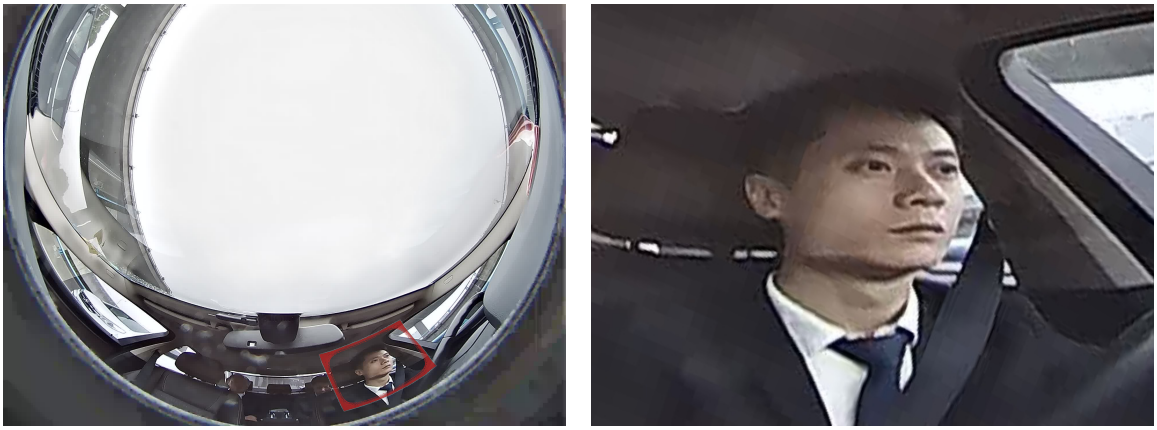
Methods for Dewarping Secure360 Images

Zachary Pitcher*

August 15, 2018



(a) *Region of interest in warped photo* (b) *Desired dewarped view of RoI*

Figure 1: *An image taken by a Secure360™ (a) can be dewarped to yield an image that appears to have been taken by a regular camera (b). These images can be processed by traditional detection algorithms for various applications.*

*Massachusetts Institute of Technology (zackpi@mit.edu)

1 Abstract

This paper will compare different methods for dewarping images from the Waylens Secure360™ 360° dashcam. The primary goal of this work is to create images that appear to have been taken with a regular camera. This will allow traditional computer vision algorithms, which are designed for cameras that approximate a pinhole camera, to work without having to retrain or adapt them to the Secure360™ camera. The methods we shall test range from a simple homographic transformation, to a more thoughtful polar transformation, to a more generalized remapping scheme that considers empirically measured lens properties.

2 Introduction

Images from the Secure360™ are very warped because of its unique extremely wide-angle lens. The Secure360™ captures 360° horizontally and 220° vertically. The camera is positioned facing straight up, so the points of interest—the driver, passengers, and the road—are located near the edges of the image and are therefore severely warped by the lens. We need a proper dewarping scheme to prioritize the edges of the image in order to ensure that any driver- or lane-detection software works in spite of the warped source image. We propose several methods for dewarping Secure360™ images: homography, polar distortion, and a remap based on empirically determined mapping functions. We analyze the properties of these methods, and determine which is the most viable and effective.

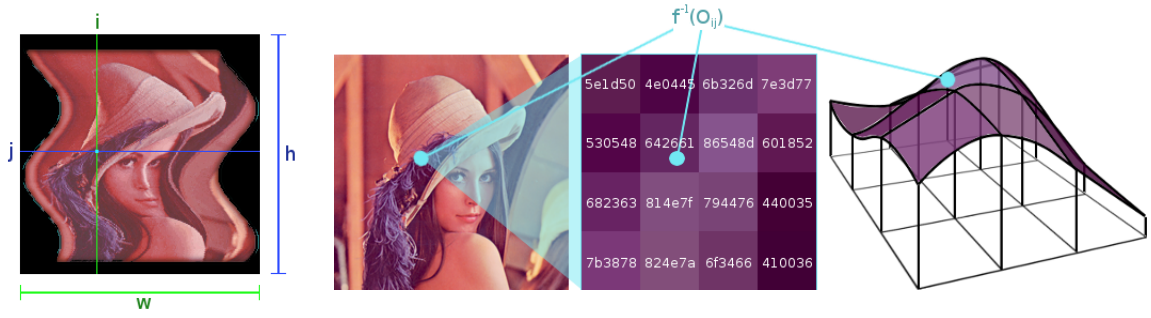
3 General Remap Algorithm

Every image remap is based on the same algorithm, and so will the dewarp methods described in this paper. Here is an outline of this general algorithm:

Let the input image be I and the output O. Then the goal of remapping is to apply some mapping $M(I) = O$. We can imagine iterating over each pixel I_{xy} in I and placing the color from I_{xy} at $M(I_{xy})$ in O. But since M can be any reasonably smooth function, this gives limited control over the resolution of O, and for most M would require truncation to make valid integer coordinates in O.

Instead, we find $M^{-1}(O) = I$. Then for a given resolution $w \times h$ of O, we iterate over i up to w and j up to h. For each pixel O_{ij}, we can estimate the color value of $M^{-1}(O_{ij})$ in I using a chosen interpolation method, then place that color value at O_{ij}. This method gives the user control over the resolution by choosing the dimensions of the output and the quality by choosing the interpolation method.

Figure 2 demonstrates this remapping scheme where M is some sinusoidal function in y and bicubic interpolation is applied to determine the color values.



(a) Iterating over (i, j) in the $w \times h$ output image

(b) Determining the color value of O_{ij} using bicubic interpolation around $M^{-1}(O_{ij})$ in the input image.

Figure 2: General remapping algorithm

4 Methods

4.1 Homography

The first method is a homographic, or perspective, transform. Homography allows an image on a 2D plane to be projected onto another plane. In our case, we will choose to project onto the plane that rectifies a quadrilateral region of interest in the source image.

Let the 4 points of the input quadrilateral have coplanar points (x_i, y_i) for $1 \leq i \leq 4$. We represent this quadrilateral as a 3×4 matrix Q with columns of the form $\langle x_i, y_i, 1 \rangle$. We would like to map these points to (x'_i, y'_i) that form a rectangle in another plane, represented as a matrix with columns $\langle x'_i, y'_i, 1 \rangle \in Q'$. The 3×3 matrix H satisfying $cQ' = HQ$ is called the homographic matrix, where c simply defines the scale of the output rectangle. Then for each point in the input:

$$c \cdot \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{1}$$

Each of the 4 point relations describes 3 equations, and there are 9 unknown h_{ji}. Therefore, H can be found for a given mapping $Q \mapsto Q'$. Similar linear algebra yields H^{-1}, which can be used as M^{-1} in the general remap algorithm outlined in Section 3.

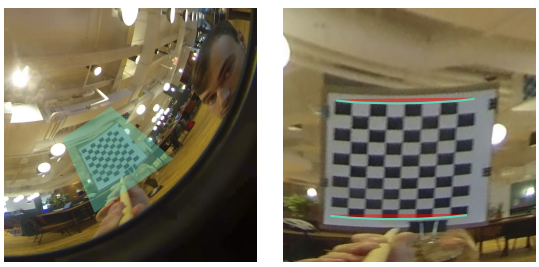
This method assumes that the original image was taken by a camera that approximates a pinhole camera, so it can't properly dewarp entire images taken by the Secure360™ camera. However, there might still be some useful applications of homography in dewarping Secure360™ images.

One idea is to use information from previous frames in conjunction with homography to quickly determine the location of the driver's face in the source image. This would help to set the focus point for a more sophisticated and computationally expensive dewarp method to be applied after. Due to the temporal locality of objects within the car, this might be able to detect small movements and increase effectiveness of other algorithms. Work is yet to be done in this area.

Another idea is to use tiled homography. Homography is very fast, but due to the nature of the Secure360™ lens it's only accurate for very small quadrilateral source regions. Then we can perform small homographic transformations on many subdivisions of the source region. This introduces a tradeoff between the size of the individual homographic regions—and by extension, the accuracy of the remap—and the speed of the operation.

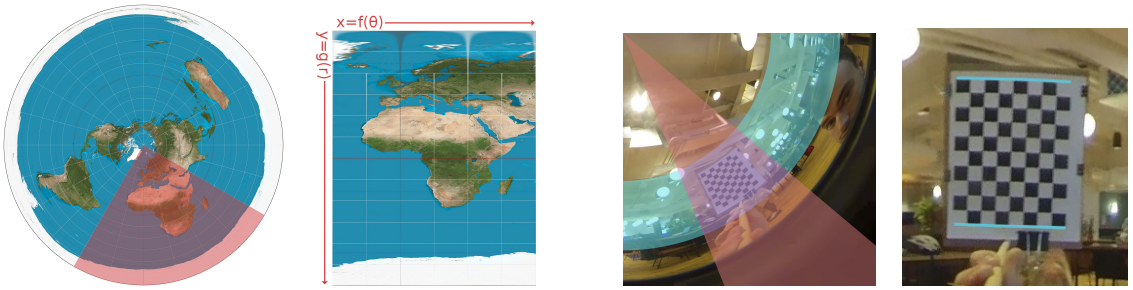
Assume we know the mapping M^{-1} from the output space to the input space for a given remapping scheme. We will subdivide the output rectangle into rectangles of width n. If the desired dimensions of the output are $w \times h$, then each rectangle will have dimensions $n \times \frac{nh}{w}$. For each rectangle, map its four corners to points in the input space and use the quadrilateral that they form as the source region for a homographic transformation. As $n \to 1$, tiled homography approaches the accuracy of general remapping algorithm, but takes longer to execute. As $n \to w$, tiled homography approaches the speed and innacuracy of regular homography. By manipulating n, we might be able choose how accurate or how fast the remap should be. No work has been done in this area yet.

(a) Homography between two planes can be used to map images between points of view



(b) Homography applies to planes, but the Secure360™ lens is not planar, so even a good choice of Q (teal,left) yields an output image with deflection (red) as a function of the curvature of the lens

Figure 3: Homographic Transformation



(a) A sector in the input image becomes a rectangle in the output. The transform maps r to y and θ to x

(b) Linear polar has less vertical deflection and area distortion compared to homography

Figure 4: Linear Polar Transformation

4.2 Linear Polar Transformation

Homography is the wrong approach because it relates planes, whereas the Secure360™ lens has positive curvature. Linear polar transformation attempts to account for the shape of the lens and avoid the distortion that homography causes in the output (compare Figure 3 to Figure 4).

If we assume that the lens is part of a sphere, then radii and concentric circles in the sensor image are vertical and horizontal lines in real space. With this assumption, we can define polar axes (r, θ) in the sensor image and (x, y) in the output image. A polar transform sets $x = f(\theta)$ and $y = g(r)$. Linear polar is the special case $f(\theta) = c_1 \cdot \theta$ and $g(r) = c_2 \cdot r$. In terms of the general remapping algorithm, $M^{-1}(x, y) = \big(f^{-1}(x), g^{-1}(y)\big) = (\theta, r)$.

Since the Secure360™ faces straight up, the horizon is near the outer rim of the circle. We ensure that information in this region is preserved, because that's where the driver's face and the road will be. Let R be the radius of the sensor image. Then we choose c_1 and c_2 such that the width of the output image $w = 2\pi R$ and the height $h = R$. Therefore none of the information in the largest concentric circle is lost in the transformation.

Similar to the problem of projecting the globe onto a 2D map, attempting to map the entire lens onto an image leads to trade-offs. In the case of linear polar, all concentric circles with $r < R$ are stretched to achieve unity scaling for $r = R$. This has the unfortunate side-effect that as $r \to 0$ the corresponding line in the output image appears more horizontally distorted. Indeed, the top horizontal of the output image represents a single pixel in the input image.

Luckily, the most important regions for driver- and lane-detection with the Secure360™ are near the bottom of the output image and are therefore not greatly affected by this property. Nonetheless, this is an area where another dewarp method could improve upon linear polar.

4.3 Empirical Remap

Linear polar is a good approximation if the lens is indeed part of a sphere, and it performs well in practice. But we don't know whether the assumptions that linear polar makes are true, nor to what extent they affect the output. In order to justify linear polar, we remove all assumptions to find the true mapping. Then we can evaluate whether linear polar's assumptions have a significant effect on performance, and whether a better method is necessary.

Per the goals of this research, the correct mapping yields images that appear to have been taken by a regular camera. There are two major steps to accomplish this task: reverse the effects of the Secure360™ lens and apply the effects of a regular lens. First we map the sensor image onto a virtual lens using empirically-determined properties. Then we project that image onto a plane tangent to the virtual lens using the properties of a regular camera. This plane is a virtual camera's sensor and contains the output image (Figure **??**).

For our purposes, a regular camera is one that approximates a pinhole camera. A pinhole camera only lets light through a single point, so the image of an object on the sensor has dimensions that follow a simple geometric relationship. Let the camera's forward vector be the normal of the camera's sensor

that passes through the pinhole. Then consider a point d units away from the pinhole along the forward vector and at a displacement (p, q) off that axis. If the focal length of the pinhole camera is F, then the image of that point on the sensor will be at $(-\frac{Fp}{d}, -\frac{Fq}{d})$. This follows from the similar right triangles formed by the light ray passing from the point to the sensor through the pinhole. The virtual camera's sensor is tangent to the virtual lens, so F is equal to the lens' radius.

4.3.1 Projecting Sensor onto Virtual Lens

There are two coordinate systems involved in this step: lens coordinates and sensor coordinates. There are two ways of imagining lens coordinates, using cartesian points (x, y, z) or spherical points (θ, ϕ, ρ). Note that θ is the azimuthal angle, and ϕ the polar angle. We can convert between them using:

$$\begin{cases} x = \rho \sin\phi \cos\theta \\ y = \rho \sin\phi \sin\theta \\ z = \rho \cos\phi \end{cases} \tag{2}$$

$$\begin{cases} \rho = \sqrt{x^2 + y^2 + z^2} \\ \theta = \arctan\frac{y}{x} \\ \phi = \arccos\frac{z}{\rho} \end{cases} \tag{3}$$

The virtual lens is approximately a sector of a unit sphere centered on the origin. And the sensor image is a rectangle in the xy-plane, also centered on the origin. Define sensor axes a, b where a increases to the right and b increases downward. The top-left corner of the image is $(a, b) = (0, 0)$. Notice that the ab-plane is also the xy-plane. Then if the center of the image is (h, k), we can convert sensor coordinates to lens coordinates using:

$$\begin{cases} x = a - h \\ y = b - k \end{cases} \tag{4}$$

Consider a point (a, b) on the image that we would like to project back onto the lens. Apply Equation 4 to get the position of the point relative to the center of the lens. The distance from the point to the center is:

$$r = \sqrt{x^2 + y^2} = \sqrt{(a-h)^2 + (b-k)^2}$$

As we will see in Section 5, there is a function $\phi = f^{-1}(r)$ that yields the polar angle on the lens for a given radius in pixels from the center of the sensor image. Then we can apply Equations 3 and f^{-1} to find the point on the virtual lens that the light must have passed through in order to get mapped to (a, b):

$$\begin{cases} \phi = f^{-1}\left(\sqrt{(a-h)^2 + (b-k)^2}\right) \\ \theta = \arctan\left(\frac{b-k}{a-h}\right) \\ \rho = 1 \end{cases} \tag{5}$$

Following similar mathematics in the opposite direction, we find the mapping from the point that light enters the lens to the point that it hits on the sensor. Since $r = f(\phi)$ and $x = r\cos(\theta)$ and $y = r\sin(\theta)$:

$$\begin{cases} a = f(\phi)\cos(\theta) + h \\ b = f(\phi)\sin(\theta) + k \end{cases} \tag{6}$$

4.3.2 Projecting Lens onto Tangent Plane

If we apply Equation 5 to each pixel on the sensor, we can imagine covering the virtual lens in the color of light that entered through each point. Then if we place a virtual pinhole camera inside the lens on the origin and look in a given direction, it should see what a regular camera would have seen if it had been in place of the Secure360™ at the time the image was captured (see Figure **??**). This section outlines the mathematics that performs that operation.

Let the camera's forward vector be $\vec{p}$. This is the normal vector of the tangent plane that we will project onto. For a given focus point (a_f, b_f) on the sensor and center (h, k), we can use Equations 5 and 2 to calculate $\vec{p}$ such that the point where the plane is tangent to the lens is also where light that eventually hits the focus point enters the lens. Therefore, in cartesian lens coordinates:

$$\vec{p} = \begin{pmatrix} \sin\phi_f \cos\theta_f \\ \sin\phi_f \sin\theta_f \\ \cos\phi_f \end{pmatrix} \tag{7}$$
$$= \begin{pmatrix} \sin\left(f^{-1}\left(\sqrt{(a-h)^2+(b-k)^2}\right)\right)\cos\left(\arctan\left(\frac{b-k}{a-h}\right)\right) \\ \sin\left(f^{-1}\left(\sqrt{(a-h)^2+(b-k)^2}\right)\right)\sin\left(\arctan\left(\frac{b-k}{a-h}\right)\right) \\ \cos\left(f^{-1}\left(\sqrt{(a-h)^2+(b-k)^2}\right)\right) \end{pmatrix}$$

In this way, we can specify a point of interest about which the output image should be centered.

We now introduce a new puv coordinate system for the tangent plane. The first axis, $\hat{p}$, is the unit vector in the direction of $\vec{p}$. Since $\vec{p}$ points from the origin to the unit sphere, it is already a unit vector and so $\hat{p} = \vec{p}$. Define $\hat{u}$ to be a unit vector in the tangent plane. We add the additional constraint

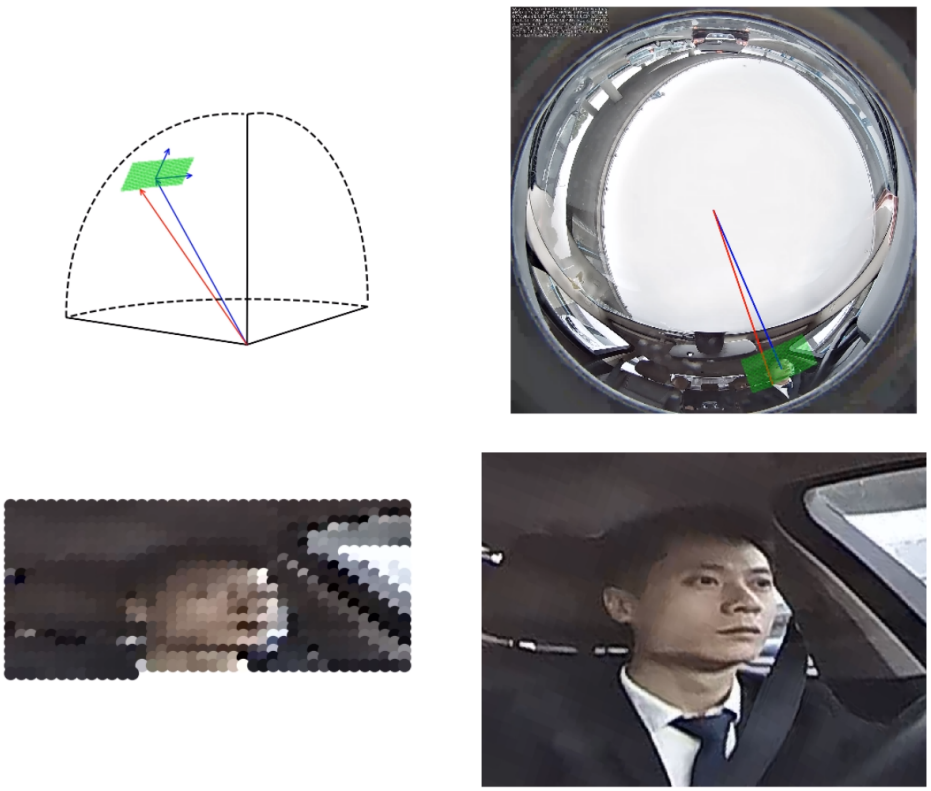


Figure 5: Remap images 1,2,3 and 4

that $\hat{u}$ is parallel to the xy-plane in lens coordinates so there is no roll rotation in the virtual camera. This makes sense physically because it ensures that "up" in the output images will always be aligned towards the top of the lens. Finally $\hat{v} = \hat{u} \times \hat{p}$. For a given focus (a_f, b_f), we know p from Equation **??**. Then the puv system is defined to be the following in cartesian lens coordinates:

$$\begin{cases} \hat{p} = \vec{p} \\ \hat{u} = \frac{1}{\sqrt{p_x^2+p_y^2}} \cdot \langle -p_y, p_x, 0 \rangle = \langle -\sin\theta_f, \cos\theta_f, 0 \rangle \\ \hat{v} = \hat{u} \times \hat{p} = \left\langle \begin{matrix} \cos\theta_f \cos\phi_f \\ \sin\theta_f \cos\phi_f \\ -\cos(2\theta_f)\sin\phi_f \end{matrix} \right\rangle \end{cases} \tag{8}$$

Where $\vec{p} = \langle p_x, p_y, p_z \rangle$.

4.3.3 Populating the Mapping Matrix

In order to apply the general remapping algorithm outlined in Section 3, we need the mapping function M^{-1} from coordinates on the plane to the sensor image. We know how to find $\vec{p}$ given a focus point on the sensor. Then in order to create an output image, we can iterate over a rectangle in the uv-plane centered on the point of tangency with the lens. We will populate a matrix R with the same dimensions as the rectangle, where each entry $R[j][i]$ is equal to the corresponding point on the sensor image. Then $R = M^{-1}$ is precisely the mapping function to be used in our general algorithm.

Let's say we wanted an $m \times n$ image as output, so R is an $m \times n$ matrix. Then we need to test $m \times n$ points in the rectangle around $\vec{p}$ to find out what color should go at each pixel in the output. Let the scaling factor of the rectangle be ϵ, so each step in $\hat{u}$ or $\hat{v}$ is ϵ units. Then while iterating over i up to m and j up to n, the position of the point on the rectangle is given by:

$$\vec{q}_{ij} = \vec{p} + \epsilon \cdot \left(\left(i - \frac{m}{2} \right) \hat{u} + \left(j - \frac{n}{2} \right) \hat{v} \right) \tag{9}$$

Then we can find $M^{-1}[j][i]$ by applying Equations 2 and 6 to $\vec{q}_{mn}$.

This scaling factor ϵ, along with the dimensions $m \times n$, will determine how much of the sensor is captured by the mapping. The scaling factor also determines the sampling frequency of the mapping; if ϵ is too large then a single step in the rectangle

will be multiple pixels in the sensor image, effectively losing information. But if ϵ is sufficiently small no information will be lost. We will fix the value of ϵ so that any step in the input space is at most one pixel. The largest steps occur where $\vec{q} \approx \vec{p}$, so ϵ should be chosen so the points $\vec{p} \pm \epsilon\hat{u}$ and $\vec{p} \pm \epsilon\hat{v}$ correspond to pixels on the sensor directly adjacent to the focus point. With ϕ, θ, r as defined in Equation 5, the points on the rectangle are:

$$\begin{cases} \vec{p} \pm \epsilon\hat{u} = \begin{pmatrix} \sin\phi_f \cos\theta_f \\ \sin\phi_f \sin\theta_f \\ \cos\phi_f \end{pmatrix} + \epsilon \begin{pmatrix} -\sin\theta_f \\ \cos\theta_f \\ 0 \end{pmatrix} \\ \vec{p} \pm \epsilon\hat{v} = \begin{pmatrix} \sin\phi_f \cos\theta_f \\ \sin\phi_f \sin\theta_f \\ \cos\phi_f \end{pmatrix} + \epsilon \begin{pmatrix} \cos\theta_f \cos\phi_f \\ \sin\theta_f \cos\phi_f \\ -\cos(2\theta_f)\sin\phi_f \end{pmatrix} \end{cases}$$

Following Equation 3, we can compute the lens coordinates (ρ', θ', ϕ') that correspond to these points:

$$\begin{aligned} \rho' &= \sqrt{\begin{array}{l} \left(\sin\phi_f \cos\theta_f - \epsilon\sin\theta_f\right)^2 \\ +\left(\sin\phi_f \sin\theta_f + \epsilon\cos\theta_f\right)^2 \\ +\cos^2\phi_f \end{array}} && \text{(apply 3)} \\ &= \sqrt{\begin{array}{l} \sin^2\phi_f \cancelto{1}{\left(\cos^2\theta_f + \sin^2\theta_f\right)} \\ +\epsilon^2 \cancelto{1}{\left(\cos^2\theta_f + \sin^2\theta_f\right)} \\ +2\epsilon\sin\phi_f \cos\theta_f \sin\theta_f \cancelto{0}{(1-1)} \\ +\cos^2\phi_f \end{array}} && \text{(reduce)} \\ &= \sqrt{1+\epsilon^2} && (10) \end{aligned}$$

$$\begin{aligned} \theta' &= \arctan\left(\frac{\sin\phi_f \sin\theta_f + \epsilon\cos\theta_f}{\cos\phi_f}\right) && \text{(trig ident)} \\ &= \arctan\left(\tan\phi_f \sin\theta_f + \epsilon \cdot \sec\phi_f \cos\theta_f\right) && (11) \end{aligned}$$

$$\phi' = \arccos\left(\frac{\cos\phi_f}{\sqrt{1+\epsilon^2}}\right) \qquad (12)$$

And applying Equation 6 to those lens coordinates, we can determine the sensor coordinates (a', b') for these points:

$$\begin{aligned} a' &= f(\phi') \cdot \cos(\theta') + h \\ b' &= f(\phi') \cdot \sin(\theta') + k \end{aligned}$$

Setting the euclidean distance in sensor pixels from the focus point (a_f, b_f) to the shifted point (a', b') to at most 1, we can find the value of ϵ that doesn't waste information during the remap:

$$\begin{aligned} 1^2 &\geq (a' - a_f)^2 + (b' - b_f)^2 \\ &= \left(f(\phi')\cos(\theta') - a_f\right)^2 + \left(f(\phi')\sin(\theta') - b_f\right)^2 \\ &= \left(f(\phi')\right)^2 \cancelto{1}{\left(\cos^2\theta' + \sin^2\theta'\right)} \\ &\quad + 2f(\phi')\left(a_f\cos\theta' + b_f\sin\theta'\right) + \left(a_f^2 + b_f^2\right) \\ &= a_f^2 + b_f^2 + f^2(\phi') + 2f(\phi')(a_f\cos\theta' + b_f\sin\theta') \end{aligned}$$

Even if we expand the ϕ' and θ' to a_f, b_f, h, k, and f, this equation is intractible as there are three unknowns entangled in trigonometric and empirical functions. But we can evaluate ϵ for a *specific* choice of (a_f, b_f) since we would know all of the variables except ϵ. Then we can populate a 2D matrix E with dimensions of the size of the sensor (1920×1920) where $E[a][b] = \epsilon_{ab}$. Since the properties of the lens shouldn't change over time, we can use this look-up table to determine the optimal scaling factor in constant time for a given focus point on the sensor.

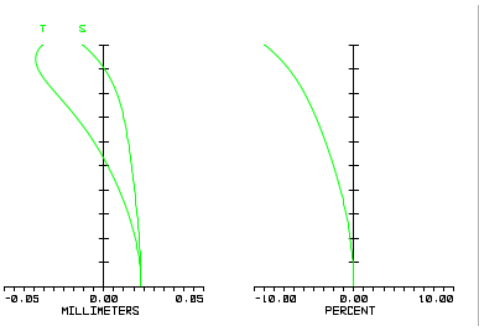


Figure 6: F-Theta Distortion Graph from Lens Manufacturer

5 Measuring Lens Properties

The empirical mapping method by definition relies on experimentally recorded values. The lens manufacturer supplied data from synthetic testing about the lens' properties, shown in Figure 6. These graphs implies that $f_{\text{lens}(r)}$ is not perfectly linear. We will set up an experiment to test this property for use in our empirical remap, and also to see how the physical results compare to the theoretical values.

The goal is to test how polar angle ϕ of a point on the lens relates to radius r on the sensor. We will construct a measuring device with evenly spaced markings on a part of a circle with the Secure360™ lens at its center. If the measuring device is rotated so the 0-degree mark is located directly above the camera, then its image on the sensor will be a straight line from the center outward. Since each marking is a constant change in polar angle, we just need to record the radius at which each marking appears on the sensor to measure f. This process was repeated for several z-rotations of the camera, in order to determine whether f varies with respect to azimuthal angle θ.

The result is a function that appears approximately linear and that doesn't depend significantly on θ. Based on the appearance of f, we start with a linear model, $\hat{f}_1$. On the plot of the residuals $(f - \hat{f}_1)$ for the linear regression of the data, the curved trend implies that f has significant higher-degree properties. This justifies testing higher-degree models $(\hat{f}_2, \hat{f}_3, \hat{f}_4, \cdots)$ to find a better fit for f. As you can see in Figure **??**, the error in the model decreases greatly for the first few additional terms. But after about $\hat{f}_4$, error only decreases marginally and we run the risk of overfitting.

From this testing, it's clear that f is close to but not quite linear, which agrees with the manufacturer's claim. Now we can use this data in the empirical remap method.

(a) Measuring device

(b) Alignment

(c) Even lighting

Figure 7: Measurement Setup

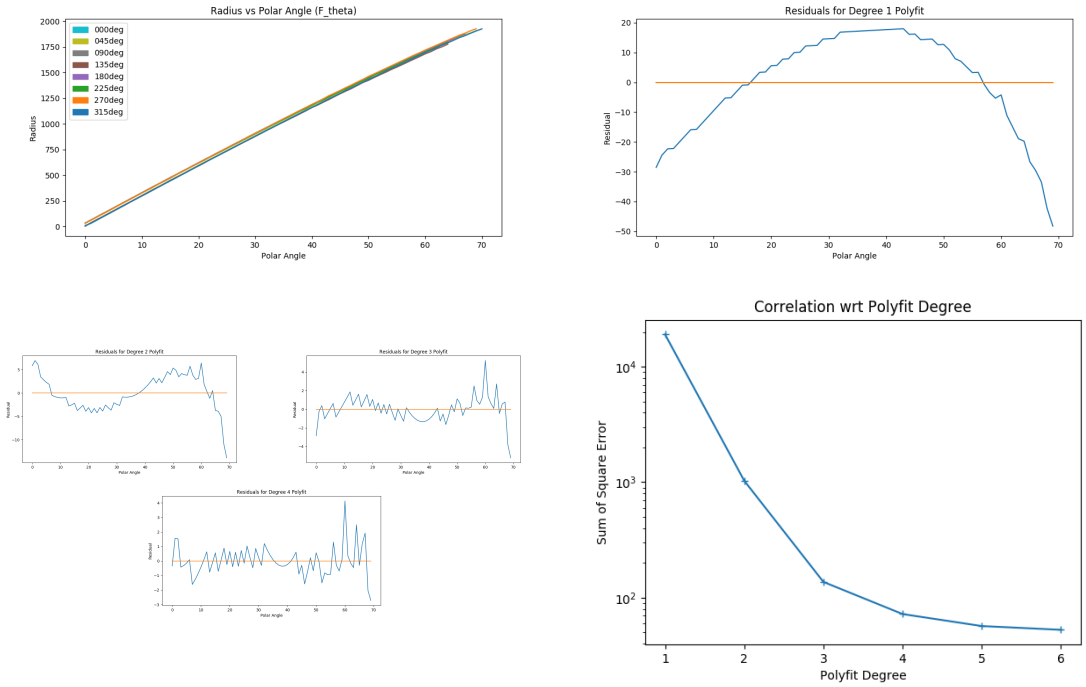


Figure 8: Measurement Results

6 Performance

The metrics for evaluating the performance of Secure360™ dewarping methods are based on those methods' intended use-case as the first step in applying neural networks and object detection algorithms to Secure360™ images. Two metrics define success in that task: how often a frame can be processed and how consistently an object detector can find relevant objects in a frame.

In order to evaluate the speed of each algorithm, we will record the time that it takes to process a 10-second (250-frame) video. Here are the results:

Method	**Overhead** (s)	**Loop** (s/frame)
Homography	n/a	0.0519
Linear Polar	n/a	0.3376
Emp.Remap	33.843	0.0983
conv. E.R.**	34.304	0.0688

*Building the remap matrix takes 0m34.043s, but its only computed once

**OpenCV's `convertMaps()` speeds up inner loop but adds 0.461s initial overhead

Homography, as expected, is the fastest during the inner loop, but not by much. The empirical remap is almost as fast per frame, especially after applying the `convertMaps()` function to compress the mapping matrix. Linear polar is surprisingly slow, but it could be improved by performing a local mapping instead of using the canned global `linearPolar()` function in OpenCV.

The next metric is how well a method's images work in object detection algorithms. We will run two face-detection algorithms, a classical Haar Detector and a Convolutional Neural Network, on each method's output from the 10-second test. Each algorithm will output a percent consistency, measured by the fraction of frames in which a face was detected for the Haar algorithm and by the sum of the maximum confidence scores (on a [0,1] scale) of each frame for the CNN algorithm. This operation will also be applied to the raw Secure360™ output and a video taken with a regular camera as controls.

Method	**Detection Consistency (%)**	
	Haar*	CNN**
Raw Secure360™	0	11.0
Homography	93.6	83.3
Linear Polar	100	93.1
Emp.Remap	100	96.7
Real camera	100	99.6

*Count of frames where traditional Haar Detector found a face

**Sum of max confidence score from convolutional neural network

The controls behaved as expected. If no dewarping is applied, both methods fail to find any faces in the video. This shows that some kind of dewarping is necessary to make these algorithms work at all. If the input is from an actual camera, then the Haar finds the face every time and the CNN does so with high confidence. The remaining tests also had predictable outcomes. Homography worked most of the time, but the low confidence score from the CNN implies that the faces it detected were still warped. Linear Polar worked every time and with relatively high confidence, but Empirical Remap did better, performing almost as well as the real camera.

7 Conclusion

The purpose of this research was to find the best dewarping method for Secure360™ images. We defined that to be the method whose output is most like that of a pinhole camera. Linear Polar does this pretty well, but the assumptions that we made to justify it are what hold it back. By getting rid of these assumptions, Empirical Remap improves upon Linear Polar by yielding images in which objects are more recognizable to traditional object detectors.

Once the mapping matrix has been compressed to a format that makes the inner loop faster, using empirical data to remap the Secure360™ images is both effective and practical. The initial overhead to create the matrices can be performed well in advance and stored on the device, if not in the cloud. Overall, real-time and accurate dewarping of Secure360™ images is achievable, but further work is required to optimize these operations for computation "at the edge."